Towards Private and Compliable Blockchain Applications

TIAN QIU

Supervisor: Qiang Tang

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Computer Science
Faculty of Engineering
The University of Sydney
Australia

24 July 2025

Abstract

Privacy and compliance are essential pillars of modern society, with privacy safeguarding individual autonomy and compliance ensuring adherence to norms and regulations. Effective compliance must uphold privacy to ensure trust and legitimacy, while privacy frameworks depend on compliance to implement safeguards and ensure accountability. The transparent nature of emerging blockchain technology presents unique challenges to balancing these principles, especially in regulated sectors.

To study the privacy and compliance issues in blockchain systems, this thesis focuses on three key domains, each exemplified by a representative application. For each application, it provides concrete and efficient constructions. Additionally, it introduces a novel cryptographic tool, serving as a building block for developing private and compliant blockchain applications.

The thesis begins by addressing privacy concerns in centralized cryptocurrency exchange platforms, proposing a privacy-preserving exchange system that achieves user anonymity and regulatory compliance simultaneously. Next, it focuses on privacy as a regulatory mandate, presenting a secure dark pool system that employs secure multiparty computation to preserve privacy while ensuring market stability. It then explores scenarios where privacy is exploited, proposing a novel mechanism to prevent gas fee concurrency attacks in relay systems, ensuring fair compensation without compromising anonymity. This work also introduces predicate aggregate signatures motivated by blockchain governance, a new cryptographic primitive that integrates privacy protection into compliance settings, enabling anonymous aggregation of signatures while maintaining compliance with public predicates.

Through these applications and the fundamental tool, this thesis demonstrates how privacy and compliance can be balanced using innovative cryptographic techniques, offering practical solutions for real-world blockchain applications.

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Name: Tian Qiu

Signature:

Date:

Authorship Attribution Statement

This thesis contains material in the following publications and manuscripts, in which the authorship is in alphabetical order and I am the lead author.

1. Ya-nan Li, **Tian Qiu**, Qiang Tang. Pisces: Private and Compliable Cryptocurrency Exchange. *The Network and Distributed System Security Symposium* 2024, **NDSS 2024**.

2. **Tian Qiu**, Qiang Tang. Predicate Aggregate Signatures and Applications. *29th Annual International Conference on the Theory and Application of Cryptology and Information Security*, **AsiaCrypt 2023**.

3. Ya-nan Li, **Tian Qiu**, Qiang Tang. Pisces: Private and Compliable Cryptocurrency Exchange. (under review by *ACM Transactions on Privacy and Security*)

4. **Tian Qiu**, Qiang Tang, Sri AravindaKrishnan Thyagarajan. Even Small, Cannot be Ignored: Fair Relay in Anonymous Zether. (*ready to submit*)

5. Quanhao Chen, Ya-nan Li, **Tian Qiu**, Qiang Tang. Charon : Private Cryptocurrencies Trading in Dark Pool. (*ready to submit*)

My particular contributions to this have been:

• Chapter 3 of this thesis is published as [1].

I designed the protocol, analyzed the security, and wrote the manuscript.

• Chapter 4 of this thesis is contained in the manuscript [5].

I designed the protocol, analyzed the security, and wrote the manuscript.

• Chapter 5 of this thesis is contained in the manuscript [4].

I designed the protocol, analyzed the security, and wrote the manuscript.

• Chapter 6 of this thesis is published as [2].

I designed the protocol, analyzed the security, and wrote the manuscript.

Name: Tian Qiu

Signature:

Date:

Attesting Authorship Attribution Statement

As the supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Supervisor Name: Prof. Qiang Tang

Signature:

Date:

Acknowledgements

My PhD journey began in the fall of 2019 at the New Jersey Institute of Technology, USA. Over these six years, both the world and I have undergone significant changes. During the challenging times of the pandemic, I transitioned from the USA to The University of Sydney, Australia. I am deeply grateful to everyone who supported and accompanied me throughout this incredible chapter of my life.

First of all, I wish to express my deepest gratitude to my supervisor and mentor, Prof. Qiang Tang. I am profoundly thankful for his time and patience during our discussions and talks. His sharp mind has cultivated my taste for valuable research, his rich experiences have broadened my view, and his enthusiasm for research has encouraged me to delve deeper into exploration. Above all, his honesty and kindness have inspired me to work toward building a better future world in a practical way.

I would like to extend my sincere thanks to the committee members, Prof. Jiaheng Zhang and Prof. Foteini Baldimtsi, as well as the committee convenor, Prof. Sri AravindaKrishnan Thyagarajan, for their wonderful support and insightful comments.

I am also grateful to the two universities, NJIT and USYD, where I have undertaken my PhD journey. They have provided me with unique experiences and supportive environments. The research reported in this thesis was supported by the award of a Faculty of Engineering Research Scholarship (USYD) to me.

Loneliness is an inherent part of life, an inescapable truth of existence. Nothingness is the most dangerous Siren, inducing people to get lost in confusion. Fortunately, friendship and love save us by alleviating the pain and filling the emptiness, and support us in living a practical life. I am very grateful to the teachers and friends I have met throughout my life for their invaluable support and encouragement. I extend my heartfelt thanks to the members of our group for their insightful discussions and to my co-authors for our fruitful and successful collaborations. An incomplete list includes Dr. Long Chen, Dr. Yuan Lu, Dr. Zhenliang

Lu, Dr. Songlin He, Dr. Erin Kenney, Dr. Bo Pang, Dr. Hanwen Feng, Ms. Ya-Nan Li, Ms. Xinrui Zhang, Mr. Sam Polgar, Mr. Tianyi Zhang, Mr. Tiancheng Mai, Mr. Yuchen Ye, Mr. Quanhao Chen, Ms. Omniyyah Ibrahim, and more. Special thanks to Dr. Guofeng Tang, my love and soulmate, although we have spent most of these years in different countries, every place I go leads me back to you.

Lastly, I am incredibly thankful to my family for their love, guidance, and unconditional support. In particular, I dedicate this work to the memory of my grandmother. When I was a child, I enjoyed sitting around the stove when she was cooking and listening to her. Now, after so many years, all that remains is the warmth of that memory. Words cannot express how deeply I long for the chance, to sit by the stove again, and translate the above English texts for you.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Background

Privacy and compliance have been fundamental to the organization and stability of societies throughout history, evolving alongside social, political, and technological advancements. Their importance lies in their ability to safeguard individual freedoms, promote trust, and ensure the fair functioning of societal systems.

Privacy as a basic right. Privacy serves as a cornerstone of personal autonomy and dignity [1, 2]. In ancient times, privacy was primarily centered on maintaining the confidentiality of personal, political, and commercial matters. While not formalized as a right, the inherent value of privacy is evident in how people resisted excessive scrutiny and safeguarded their secrets. This was achieved through methods such as physical isolation, coded communication, or reliance on trusted intermediaries. Notable examples include private communication during wartime to protect strategic information, anonymous whistleblowing to expose wrongdoing without being retaliated against, and using secret ballots to ensure fairness in voting.

In recent decades, privacy has become even more critical. As societies have transitioned to digital and interconnected environments, vast amounts of personal and organizational data are constantly being generated, stored, and analyzed, such as digital transaction information, including the sender, receiver, and transaction amount. The rise of surveillance technologies, data mining, and the commodification of personal information has heightened concerns about privacy breaches, misuse, and unauthorized access. Protecting privacy ensures individual

freedom and the ethical use of information, fostering trust between individuals, organizations, and governing bodies.

Compliance enforced by the regulation. While privacy focuses on individual autonomy, compliance upholds societal cohesion by enforcing adherence to established norms, rules, and regulations [3]. In ancient civilizations, compliance was maintained through direct surveillance, taxation systems, and conformity to social and religious norms. Governments and rulers closely monitored activities to uphold law and order. Though rudimentary, these mechanisms laid the foundation for modern compliance practices.

Today, compliance is a critical framework across domains like finance, healthcare, and digital communication, ensuring ethical and lawful operations while protecting private and public interests. Key components include Know Your Customer (KYC) [4], which collects and verifies customer information, such as name, address, and date of birth, to prevent fraud and money laundering through data collection and transaction monitoring, and Anti-Money Laundering (AML) [5], which enforces regulations to detect and report financial crimes through due diligence and monitoring systems.

Coexistence of privacy and compliance. While privacy and compliance may appear at odds – privacy seeks to limit oversight, while compliance often involves monitoring – they are deeply interconnected. Effective compliance mechanisms must respect privacy to maintain legitimacy and public trust. Conversely, privacy frameworks usually rely on compliance to enforce safeguards and hold violators accountable. Striking the right balance between these two principles is essential in creating systems that protect individual freedoms while ensuring societal order and accountability.

The importance of privacy and compliance extends beyond individuals to influence the stability and resilience of entire societies. Ensuring privacy fosters innovation and creativity, as people are more likely to express themselves and share ideas without fear of intrusion. Robust compliance systems, on the other hand, build trust in institutions, prevent corruption, and promote fair practices. Together, they create an environment where individuals and organizations can thrive while contributing to the common good.

1.2 Challenges in Blockchain Era

The raising of blockchain. Initially introduced in 2008 by Satoshi Nakamoto as the underlying technology for Bitcoin [6], blockchain emerged as a decentralized, transparent, and tamper-resistant system for managing transactions without the need for intermediaries.

At its core, a blockchain is a distributed ledger that records transactions in a series of blocks linked together. Each block is immutable once added, ensuring data integrity. It was built on cryptographic concepts such as public-key cryptography, hashing algorithms, and Merkle trees. Its development over the years has expanded far beyond cryptocurrency, finding applications in various industries such as finance, supply chain, healthcare, and governance. Blockchain's decentralized nature eliminates single points of failure and fosters transparency, as all participants in the network have access to the same information.

The launch of Ethereum in 2015 [7] marked a transformative step forward for blockchain technology. It introduced smart contracts, self-executing programs embedded in the blockchain that automatically enforce agreements when predefined conditions are met. Smart contracts expanded blockchain's potential far beyond cryptocurrency, paving the way for decentralized applications (dApps). Their transparency, tamper-resistance, and versatility have made them invaluable across industries such as finance, supply chain, real estate, and healthcare.

Given the regulatory complexities associated with these applications, implementing a compliance framework in blockchain systems is highly desirable. Such a framework ensures legal and regulatory adherence while preserving blockchain's decentralized nature. Key components include identity verification (KYC/AML) to authenticate participants, real-time transaction monitoring to prevent illicit activities, smart contract audits to ensure operational compliance, and data privacy measures to safeguard sensitive information. Together, these measures create secure, transparent, and legally compliant blockchain ecosystems.

Privacy and compliance in blockchain evolution. While blockchain technology offers groundbreaking potential, it also brings new challenges, particularly in privacy. A core feature of blockchain is its transparency, with the entire ledger replicated across a permissionless

network. Consequently, any data submitted to the blockchain becomes publicly visible, exposing transaction details and personal information. For example, the identities of the sender, the receiver, and the amount of the transaction. In contrast, centralized systems restrict access to such data, with only a few data centers having visibility into users' transaction history and sensitive information. This stark difference has raised significant privacy concerns, as many users are reluctant to have their confidential data openly accessible on the blockchain.

Many privacy-centric designs have been proposed to deal with privacy concerns, such as those found in certain cryptocurrencies [8, 9]. While these approaches effectively safeguard user privacy, they can impede regulatory oversight, sparking concerns about their potential misuse for illicit activities. Countries like Japan and South Korea have already delisted privacy coins from major exchanges [10].

Regulatory requirements in blockchain-related systems focus on identity verification, tax compliance, and sanctions enforcement. Know Your Customer (KYC) mandates identity binding at entry and exit points, such as exchanges like Binance and Coinbase [11]. Tax reporting requires users to disclose profits or losses for taxation, as enforced by agencies like the IRS in the U.S. for crypto asset disposals [12]. Sanctions compliance involves enforcing restrictions on certain individuals or entities, exemplified by OFAC's blacklist of crypto addresses [13].

In summary, the main challenge lies in balancing blockchain's inherent transparency with privacy protection. It is critical to the technology's adoption in highly regulated domains. Addressing this challenge requires innovative approaches that integrate privacy-preserving techniques with regulatory compliance mechanisms.

1.3 Balance the Privacy and Compliance

1.3.1 Previous Works

We now undertake an overview of previous works related to privacy and compliance in blockchain applications.

Private payment systems There are famous private payment systems built on a single closed blockchain, such as Zcash [8] and Monero [9]. Ben-Sasson et al. [14] introduced Zerocash, the first fully-featured and cryptographically rigorous payment system. It improves upon the efficiency and security of Zerocoin by Miers et al. [15]. Currently deployed in the cryptocurrency Zcash, Zerocash leverages succinct proofs and enables constant-time verification. However, its design requires a trusted setup and includes a constantly growing UTXO set. Monero [9] is another private cryptocurrency that enhances privacy by concealing UTXO values and obfuscating transaction inputs through the use of decoy UTXOs.

The advent of Ethereum has highlighted the remarkable versatility of smart contracts. Several studies have explored deploying private payment systems on the public Ethereum blockchain using smart contracts. Notable examples include the anonymous Zether protocol [16, 17] and Tornado Cash [18].

Many solutions have been proposed in the off-chain setting. They perform as opt-in tools that enhance privacy for existing cryptocurrencies. TumbleBit [19] is a unidirectional payment channel hub (PCH) relying on an untrusted intermediary called Tumbler and Hashed Timelock Contracts (HTLCs). The Tumbler issues anonymous payments that users can cash out to Bitcoins. It is also guaranteed that the Tumbler cannot engage in theft of Bitcoins or make payments to itself. Anonymous atomic locks (A2L) is introduced in [20] where the authors propose a payment channel hub (PCH) upon it. This PCH functions as a three-party protocol designed for conditional transactions, in which an intermediary (referred to as the hub) disburses funds to the recipient contingent upon the recipient's successful resolution of a puzzle, aided by the sender. This arrangement signifies that the sender compensates the hub. The utilization of a randomized puzzle ensures that the hub cannot establish a connection

between the sender and the recipient involved in a payment. Since the payment amount can be used to link the sender and receiver trivially, the unlinkability requires the amount to be fixed [19, 20, 21]. This restriction is removed by Qin et al. in [22]. They proposed Bindhub, which is a PCH that achieves private payments with variable amounts by concealing them.

LDSP [23] is a layer-2 cryptocurrency payment system that supports payer privacy. It is designed in the setting of shopping with cryptocurrency, where the payer is the customer and the receiver is the merchant. There is also an untrusted entity called the leader who is in charge of issuing coins for customers and merchants. Customers can transfer coins off-chain with low latency. The leader cannot link the spent coin with any customer. At the same time, the merchants are guaranteed to receive the coins.

Private payment with accountability. There are some works in studying to achieve privacy-preserving and accountability at the same time. PGC [24] is an auditable, decentralized, confidential payment system. It offers transaction confidentiality and two levels of auditability, namely, regulation compliance and global supervision at the same time. Androulaki et al. [25] present a privacy-preserving token payment system for permissioned blockchains with auditing. The content of transactions is concealed, and only some authorized parties can inspect them.

UTT [26] stands as a decentralized electronic cash payment system designed to incorporate accountable privacy measures. One of its key features is the integration of anonymous budgets, which contribute to maintaining a balance between privacy and accountability. Within the UTT framework, senders are empowered to generate payments in an anonymous manner, but this is subject to a predefined monetary limit per month. Once this limit is exceeded, the system mandates that their transactions must become visible and transparent to a governing authority. This approach ensures that while users can transact with a certain degree of privacy, their financial activities remain accountable when they surpass the specified budgetary threshold.

Platypus [27] is a payment system designed for use within the context of a central bank digital currency (CBDC) environment. It focuses on enabling transactions that are unlinkable, ensuring privacy while also accommodating regulatory requirements. The system introduces

a versatile regulatory framework, which can be applied across various scenarios, and it effectively enforces limitations on holdings and receipts as specific instances of regulatory control.

Insufficiency of existing works. In spite of the advancements in private payment designs, they cannot accommodate the full range of versatile blockchain applications. Integrating private payment systems with other mechanisms introduces new challenges that may compromise user anonymity or lead to financial loss for honest users. For example, cryptocurrency exchanges require handling multiple cryptocurrencies across different blockchains, involving complex order matching and computing the revenues of each exchange transaction, which goes beyond the scope of single-blockchain payment systems. Additionally, existing constructions face inherent limitations, such as the trade-off between trust and efficiency in MPC-based dark pool schemes, where MPC is a cryptographic technique that allows a group of parties to jointly compute a function over their private inputs without revealing those inputs to each other. Regarding the cryptographic tools, most of them only focus on either compliance or privacy. For instance, multi-signatures and aggregate signatures reveal the identities of the signers, whereas threshold signatures completely obscure them. These issues underscore the need to examine blockchain applications individually to balance privacy and compliance appropriately.

1.3.2 Categorize the Applications

Striking a balance between privacy and compliance in blockchain contexts remains a challenging open problem due to the diverse range of applications, and there is no panacea to address all scenarios efficiently. Previous works mainly focus on the payment system while overlooking some other interesting blockchain-related contexts in the current world. We also note that both privacy and compliance are crucial, yet one may often be compromised in practice. Therefore, we categorize these scenarios into the following three branches as a starting point.

- **Compliable but not private.** In certain blockchain designs, compliance requirements take precedence, often leading to the neglect of privacy. This imbalance creates significant risks, as the lack of privacy mechanisms links users' real-world identities to all their future transactions on the blockchain. For example, *centralized cryptocurrency exchange platforms* serve as online marketplaces that facilitate the exchange of digital assets, such as Bitcoin and Ethereum, for fiat currencies (e.g., USD, EUR) or other cryptocurrencies. Examples include Binance [28], Coinbase [29], etc. These platforms typically require users to provide personal information, including real identities and tax documents, to comply with KYC, AML, and tax regulations. While these requirements are essential for regulatory compliance, they compromise user privacy by linking real-world identities to blockchain transactions. This association exposes a user's entire transaction history on the blockchain, creating significant privacy concerns.
- **Privacy as a compliance mandate.** In some cases, privacy is not merely a separate feature but has become a part of compliance. Regulations like the General Data Protection Regulation (GDPR) [30] and the California Consumer Privacy Act (CCPA) [31] mandate robust privacy safeguards. Another instance is trading in the dark pool. It serves as a compelling example of an application, demonstrating the concept of privacy as a compliance mandate.

  Trading cryptocurrencies in dark pools is crucial because the prices of cryptocurrencies are highly volatile and susceptible to significant fluctuations caused by large-volume trades. In public markets, substantial buy or sell orders can lead to price slippage and market manipulation, negatively impacting both the traders executing the orders and the broader market. Dark pools mitigate these risks by enabling large trades to be executed privately without immediate exposure to the public order book. This confidentiality reduces the market impact and maintains price stability. Dark pools for trading cryptocurrencies operate in a relatively nascent and evolving regulatory landscape, necessitating innovative compliance mechanisms without compromising user privacy. The trading details like asset types, volumes, and participants' identities should be concealed while ensuring operational efficiency.

- **Privacy abuse that needs compliance.** Some blockchain designs prioritize privacy above all else, often neglecting the need for compliance. While privacy is crucial for protecting sensitive user information, an overemphasis on it can lead to potential abuses. In these cases, the very feature meant to protect users' privacy can be exploited by malicious actors, causing honest participants to suffer financial losses.

  For example, the gas fee mechanism in Ethereum is designed to prevent infinite loop attacks. To invoke smart contracts on the Ethereum network, users must pay a gas fee. One such smart contract, Anonymous Zether [16, 17], aims to facilitate anonymous payments. However, paying the gas fee when calling such a contract inadvertently reveals the sender's identity, undermining the intended privacy. To address this issue, a relay mechanism is used, where a third party (the relayer) pays the gas fee on behalf of the user, preserving anonymity. However, when there are multiple relayers without a central organization or timely synchronization, malicious users can exploit the system. In this case, a user could send requests to multiple relayers at the same time. Each relayer then pays the gas fee, but only one relayer would receive compensation, resulting in financial losses for the other relayers. Meanwhile, the malicious user cannot be identified or punished due to the anonymity guarantee.

Building on the above categorizations, examining typical applications within each category provides deeper insights into the balance issues in blockchain applications. In fact, we have achieved fruitful results, ranging from system designs to foundational tool studies, which will be highlighted in the next section.

1.4 Main Results and Thesis Structure

Striking a balance between privacy and compliance in various blockchain applications remains a challenging open problem. As shown in Figure 1.1, this thesis conducts an in-depth exploration of privacy and compliance in blockchain systems across three key branches, presenting a representative application within each as an illustrative example. Each application is accompanied by concrete and efficient constructions, demonstrating practical solutions to

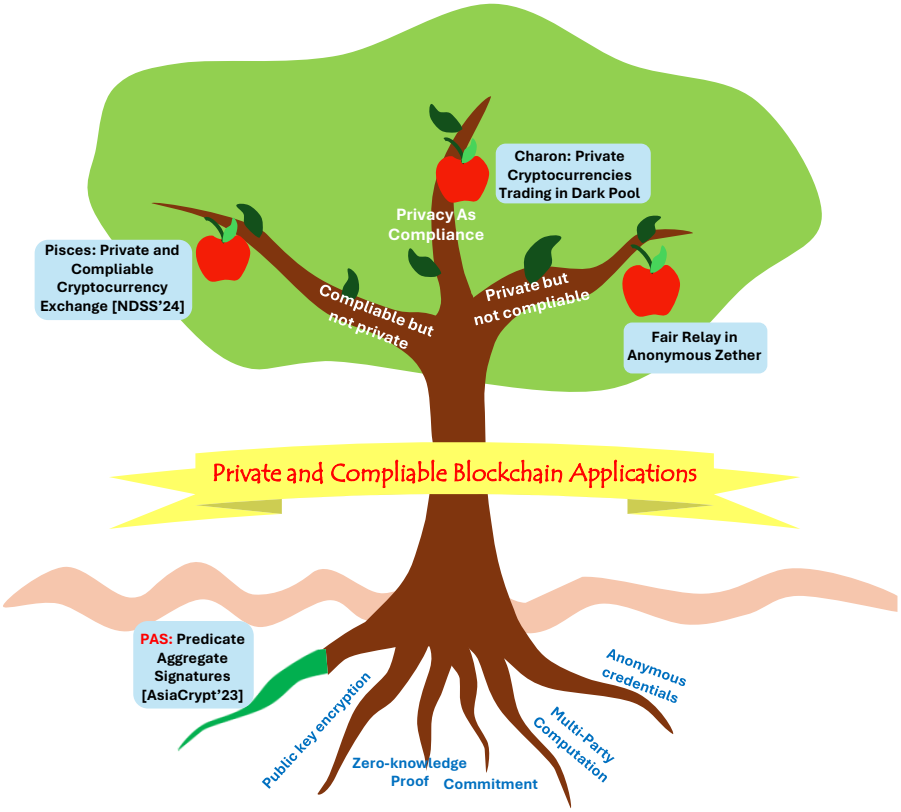

FIGURE 1.1. Main results

specific challenges. Furthermore, an innovative cryptographic tool is proposed that serves as a core building block for private and compliable blockchain applications. These contributions are outlined as follows.

- **Private and compliable cryptocurrency exchange.** We first focus on the scenario of the centralized cryptocurrency exchange platform, where stringent compliance requirements often infringe upon users' privacy. It introduces a private and compliable cryptocurrency exchange system [32] that, for the first time, restores user anonymity without compromising regulatory compliance, such as KYC and AML. The system ensures that users cannot double-spend and mandates proper reporting of *accumulated* profits for tax purposes. It features a carefully designed model and an efficient construction that achieves constant

computation and communication overhead using simple cryptographic tools and rigorous security guarantees. The implementation showcases its practicality, highlighting its feasibility for real-world applications.

- **Private cryptocurrencies trading in dark pool.** We also explore scenarios where privacy serves as the compliance mandate, with a focus on the dark pool cryptocurrency trading application. We introduce Charon, a secure and efficient cryptocurrency dark pool trading system that relies on a centralized platform acting as an intermediary. Leveraging secure multiparty computation, Charon ensures eligibility matching and trade volume matching while safeguarding critical order information, such as asset name, trading direction, and volume. This system provides robust security against semi-honest platforms and malicious traders. Two volume-matching constructions are presented: the first achieves logarithmic rounds of secure comparison w.r.t. the number of orders, while the second enhances performance through constant-round parallel comparison and stronger privacy protections. Both constructions are implemented and evaluated, showcasing their practical effectiveness in secure dark pool operations.
- **Fair relay system in anonymous Zether.** We then examine scenarios where privacy protection is exploited, highlighting the need for effective compliance mechanisms by identifying the concurrency attack within the relay system for anonymous Zether in Ethereum. In this case, a malicious user could request multiple relayers to broadcast the same transaction and pay the gas fee, but only one relayer can get paid, resulting in financial losses for the other relayers. Even worse, the malicious user cannot be identified or punished due to the anonymity guarantee. To address this issue, a novel *abuse deterring anonymous credential* (ADAC) is proposed, capable of detecting instances where a user signs the same tag multiple times, even when using credentials issued by different entities. We further introduce a concrete scheme that integrates ADAC into the relay mechanism to achieve fairness. This solution preserves the anonymity of honest participants while ensuring fair gas fee distribution: honest senders remain unaffected, and honest relayers can get reimbursed if targeted by an attack.
- **Predicate aggregate signatures.** Last but not least, we initiate the study of *predicate aggregate signatures* (PAS) [33]. It allows users to sign multiple messages, with their

individual signatures aggregated by a combiner, while maintaining the anonymity of the signers. The resulting aggregated signature reveals only a concise description of the signers for each message and ensures that both the signers and their description satisfy a specified public predicate aligned with compliance requirements. It finds various applications, such as blockchain governance and anonymous reputation systems. For example, in the on-chain voting setting, the user votes by signing for the proposal and sends it to the combiner. On receiving many users' votes, the combiner posts a final PAS signature on the chain, which hides these voters' identities and only reveals that they satisfy the vote rules. A formal definition of PAS is provided with an efficient construction that achieves a logarithmic-size signature and logarithmic verification time, enhancing efficiency while preserving privacy and compliance.

Thesis structure. This thesis conducts an in-depth exploration of balancing privacy and compliance in blockchain applications. The remaining thesis is organized as follows.

Chapter 2 introduces the foundational concepts necessary for understanding the work, including key cryptographic building blocks and blockchain-related definitions.
Chapter 3 introduces Pisces, a private and compliable cryptocurrency exchange system that restores user anonymity without compromising regulatory compliance, such as KYC, AML, and tax filing.
Chapter 4 presents Charon, a secure and efficient cryptocurrency dark pool trading system. It ensures efficient matching while safeguarding critical order information, such as asset name, trading direction, and volume.
Chapter 5 focuses on the fair relay system for anonymous Zether in Ethereum. We propose a novel abuse deterring anonymous credentials and utilize them as the building block in the relay mechanism to achieve fairness without affecting honest users' privacy.
Chapter 6 studies the predicate aggregate signatures that allow users to sign multiple messages with their individual signatures aggregated by a combiner. The resulting aggregated signature maintains the signers' anonymity and reveals only that the signers satisfy a specified public predicate.
Chapter 7 concludes this thesis by summarizing the key results presented, highlighting the

foundational methodological insights, and exploring several promising directions for future research.

CHAPTER 2

Preliminary

2.1 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 bold letter for vector, for example $\boldsymbol{a} = (a_1, ..., a_n) \in \mathbb{Z}_q^n$. We use $\circ$ to denote the Hadamard product: $\boldsymbol{a} \circ \boldsymbol{b} = (a_1 \cdot b_1, ..., a_n \cdot b_n)$ for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_q^n$ and use $\langle \boldsymbol{a}, \boldsymbol{b} \rangle := \sum_{i=1}^{n} a_i \cdot b_i$ to denote the inner product between $\boldsymbol{a}, \boldsymbol{b}$.

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.

Bilinear Map. On input the security parameter 1^λ, a group generator $\mathsf{G.Gen}(1^\lambda)$ produces public parameters $\mathsf{G.pp} = (q, \mathbb{G}, g)$, where q is a prime of length λ, and $\mathbb{G}$ is a cyclic group of order q with generator g. Similarly, a bilinear group generator $\mathsf{BG.Gen}(1^\lambda)$ produces public parameters $\mathsf{BG.pp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}, e)$ where $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle \tilde{g} \rangle, \mathbb{G}_T$ are groups of order q. The map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ define $g_T = e(g, \tilde{g})$, the map is bilinear, (for all $a, b \in \mathbb{Z}_q$, $e(g^a, \tilde{g}^b) = e(g, \tilde{g})^{ab}$) and non-degenerate (for all generators g of $\mathbb{G}_1$, $\tilde{g}$ of $\mathbb{G}_2$,

$\mathbb{G}_T = \langle e(g, \tilde{g}) \rangle$). We assume $\mathbb{G}_1 \neq \mathbb{G}_2$ and we are working on Type III groups [34] who do not have efficiently computable homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$.

We also use $[a]_1, [b]_2, [c]_T$ denotes the element $g^a, \tilde{g}^b, g_T^c$ in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively. We use $[\boldsymbol{x}]_1$ denotes the vector $(g^{x_1}, ..., g^{x_n}) \in \mathbb{G}_1^n$ for $\boldsymbol{x} = (x_1, ..., x_n) \in \mathbb{Z}_q^n$. We write all groups additively, e.g., $[a]_1 + [b]_1 = [a + b]_1$ denotes $g^a \cdot g^b = g^{a+b}$, $b \cdot [a]_1 = [ab]_1$ denotes $(g^a)^b = g^{ab}$, $[\boldsymbol{x}]_1 + [\boldsymbol{y}]_1 = [\boldsymbol{x} + \boldsymbol{y}]_1$ denotes $(g^{x_1}, ..., g^{x_n}) \circ (g^{y_1}, ..., g^{y_n}) = (g^{x_1+y_1}, ..., g^{x_n+y_n})$, $[\boldsymbol{x}]_1 \circ \boldsymbol{y} = (g^{x_1 y_1}, ..., g^{x_n y_n})$, $[\boldsymbol{x}]_1^{\boldsymbol{y}} = \sum_{i=1}^n y_i \cdot [x_i]_1 = \Pi_{i=1}^n g^{x_i y_i}$.

For $[\boldsymbol{a}]_1 \in \mathbb{G}_1^n$ and $[\boldsymbol{b}]_2 \in \mathbb{G}_2^n$, let $\langle [\boldsymbol{a}]_1, [\boldsymbol{b}]_2 \rangle := e([\boldsymbol{a}]_1, [\boldsymbol{b}]_2) = \sum_{i=1}^n e([a_i]_1, [b_i]_2)$ denote the inner pairing product between $[\boldsymbol{a}]_1, [\boldsymbol{b}]_2$. Given a vector $\boldsymbol{v} = (v_1, ..., v_n)$ with even n, we denote $\boldsymbol{v}_\ell = (v_1, ..., v_{n/2})$ and $\boldsymbol{v}_r = (v_{n/2+1}, ..., v_n)$. For $k \in \mathbb{Z}_q^*$ we use $\boldsymbol{k}^n$ to denote the vector containing the first *n* powers of *k*, i.e., $\boldsymbol{k}^n = (1, k, k^2, ..., k^{n-1})$.

2.2 Assumptions

DEFINITION 1 (DDH assumption). *Let* $(q, \mathbb{G}, g) \leftarrow \mathsf{G.Gen}(1^\lambda)$ *be a group generator. The DDH assumption holds for* G.Gen *if the following distributions are indistinguishable:* $(g, g^a, g^b, g^{ab} : a, b \leftarrow\$ \mathbb{Z}_q)$ *and* $(g, g^a, g^b, g^c : a, b, c \leftarrow\$ \mathbb{Z}_q)$

DEFINITION 2 (DLOG assumption). *Let* $(q, \mathbb{G}, g) \leftarrow \mathsf{G.Gen}(1^\lambda)$ *be a group generator. The DLOG assumption holds for* G.Gen *if for all PPT adversary* $\mathcal{A}$ *we have:* $\Pr[\mathcal{A}(q, \mathbb{G}, g, X) = x | (q, \mathbb{G}, g) \leftarrow \mathsf{G.Gen}(1^\lambda), x \leftarrow\$ \mathbb{Z}_q, X = g^x] \leq negl(\lambda)$

DEFINITION 3 (SXDH assumption [34]). *Let* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow \mathsf{BG.Gen}(1^\lambda)$ *be a bilinear group generator. The SXDH assumption holds for* BG.Gen *if DDH assumption holds for* $\mathbb{G}_1$ *and* $\mathbb{G}_2$.

DEFINITION 4 (co-CDH assumption[35]). *Let* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow \mathsf{BG.Gen}(1^\lambda)$ *be a bilinear group generator. The co-CDH assumption holds for* BG.Gen *if for all PPT* $\mathcal{A}$*, given* $[a]_1, [b]_2$ *where* $a, b \leftarrow\$ \mathbb{Z}_q$*, the probability that* $\mathcal{A}$ *can produce* $[ab]_1$ *is negligible.*

DEFINITION 5 (DPair assumption [34]). *Let* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow \mathsf{BG.Gen}(1^\lambda)$ *be a bilinear group generator,* $n = poly(\lambda)$*. The double-pairing (DPair) assumption holds for*

$\mathsf{BG.Gen}$ *if for all PPT adversary* $\mathcal{A}$*, given* $[r]_1 \leftarrow\!\$\ \mathbb{G}_1$*, the probability that* $\mathcal{A}$ *can produce* $[a]_2, [b]_2 \in \mathbb{G}_2$ *s.t.* $e([r]_1, [a]_2) + e(g, [b]_2) = [0]_T$ *and* $a, b \neq 0$ *is negligible.*

DEFINITION 6 (ML-Find-Rep assumption [36]). *Let* $(q, \mathbb{G}, g) \leftarrow \mathsf{G.Gen}(1^\lambda)$ *be a group generator,* $n = poly(\lambda)$ *which a power of 2,* $\nu = \log n$*. The ML-Find-Rep assumption holds for* $\mathsf{G.Gen}$ *if for all PPT adversary* $\mathcal{A}$ *we have:* $\Pr[\mathcal{A}(q, \mathbb{G}, [\boldsymbol{r}], X) \rightarrow \boldsymbol{a} \in \mathbb{Z}_q^n \ s.t.\ [\boldsymbol{r}]^{\boldsymbol{a}} = [0] \wedge \boldsymbol{a} \neq \boldsymbol{0} | (q, \mathbb{G}, g) \leftarrow \mathsf{G.Gen}(1^\lambda), (x_1, ..., x_\nu) \leftarrow\!\$\ \mathbb{Z}_q^\nu, \boldsymbol{r} = (1, x_1, x_2, x_1x_2, ..., x_1 \cdots x_\nu)] \leq negl(\lambda)$.

DEFINITION 7 (DPair-ML assumption). *Let* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow \mathsf{BG.Gen}(1^\lambda)$ *be a bilinear group generator,* $n = poly(\lambda)$ *which a power of 2. The DPair-ML assumption holds for* $\mathsf{BG.Gen}$ *if for all PPT adversary* $\mathcal{A}$*, given* $[\boldsymbol{r}]_1 \in \mathbb{G}_1^n$*, where* $\boldsymbol{r} = (1, x_1, x_2, x_1x_2, ..., x_1 \cdots x_\nu)$ *for* $(x_1, ..., x_\nu) \leftarrow\!\$\ \mathbb{Z}_q^\nu$*, the probability that* $\mathcal{A}$ *can produce* $[\boldsymbol{s}]_2 \in \mathbb{G}_2^n$ *s.t.* $e([\boldsymbol{r}]_1, [\boldsymbol{s}]_2) = [0]_T$ *is negligible.*

The DPair-ML assumption is implied by the SXDH assumption and ML-Find-Rep assumption.

2.3 Cryptographic Primitives

For completeness, we briefly introduce some cryptographic primitives and constructions here.

Commitment. A commitment scheme allows one to commit to a chosen value secretly, with the ability to open only to the same committed value later. A commitment scheme Π_{cmt} consists of the following PPT algorithms:
$\mathsf{Setup}(1^\lambda) \rightarrow pp$: generates the public parameter *pp*.
$\mathsf{Com}(m; r) \rightarrow com$: generates the commitment for the message m using the randomness *r*.

Hiding. A commitment scheme is said to be hiding if the commitment does not reveal any information about the committed value.
Binding. A commitment scheme is said to be binding if a commitment can only be opened to one value.

Additively homomorphic. A commitment is additively homomorphic if for any values m_1, m_2 and randomness r_1, r_2: $\mathsf{Com}(m_1; r_1) + \mathsf{Com}(m_2; r_2) = \mathsf{Com}(m_1 + m_2; r_1 + r_2)$.

Pedersen commitment. For messages $\boldsymbol{m} \in \mathbb{Z}_q^n$ and any $i \in \{1, 2, T\}$, the Pedersen commitment is defined by:
$\mathsf{Setup}(1^\lambda) \rightarrow pp$: $\boldsymbol{g} \leftarrow\!\$\ \mathbb{G}_i^n, h \leftarrow\!\$\ \mathbb{G}_i$.
$\mathsf{Com}(\boldsymbol{m}; r) \rightarrow com$: $\mathsf{Com}(\boldsymbol{m}; r) = \boldsymbol{g}^{\boldsymbol{m}} \cdot h^r \in \mathbb{G}_i$ where $r \leftarrow\!\$\ \mathbb{Z}_p$.

The Pedersen commitment is additively homomorphic, perfectly hiding and computationally binding under the DLOG assumption.

AFGHO commitment. Abe et al. [34] defined a structure-preserving commitment to group elements. In this case we have the message space $\mathbb{G}_2^n$:
$\mathsf{Setup}(1^\lambda) \rightarrow pp$: Run $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g}) \leftarrow \mathcal{G}(1^\lambda)$, the commitment key $\mathsf{ck}_1 := \boldsymbol{g} \leftarrow\!\$\ \mathbb{G}_1^n$.
$\mathsf{Com}(\boldsymbol{m}; r) \rightarrow com$: for $[\boldsymbol{m}]_2 \in \mathbb{G}_2^n$, $\mathsf{Com}([\boldsymbol{m}]_2; [r]_2) = \langle \mathsf{ck}_1, [\boldsymbol{m}]_2 \rangle + e(g, [r]_2)$ where $[r]_2 \leftarrow\!\$\ \mathbb{G}_2$.

To commit to messages in $\mathbb{G}_1$, we can just interchange the role of $\mathbb{G}_1$ and $\mathbb{G}_2$ in the above construction with $\mathsf{ck}_2 \in \mathbb{G}_2^n$.
The AFGHO commitment is additively homomorphic, perfectly hiding and computationally binding under the SXDH assumption.

Structured AFGHO. Based on the updatable common reference string technique of Daza et al. [36], we give the modified AFGHO commitment with structured commitment keys $\mathsf{ck}_1, \mathsf{ck}_2$ which are generated as below.

$$(pp, [r]_1 \in \mathbb{G}_1, \mathsf{ck}_1 = [\boldsymbol{r}]_1 \in \mathbb{G}_1^n, \mathsf{vk}_1 = [\boldsymbol{x}]_2 \in \mathbb{G}_2^\nu) \in \mathcal{L}_{\mathsf{Com}}^1 \Leftrightarrow$$
$$[r_1]_1 = [r]_1 \wedge \forall i \in [\nu], \forall j \in [2^{i-1}], [r_{2^{i-1}+j}]_1 = x_i [r_j]_1$$

$$(pp, [s]_2 \in \mathbb{G}_2, \mathsf{ck}_2 = [\boldsymbol{s}]_2 \in \mathbb{G}_2^n, \mathsf{vk}_2 = [\boldsymbol{y}]_1 \in \mathbb{G}_1^\nu) \in \mathcal{L}_{\mathsf{Com}}^2 \Leftrightarrow$$
$$[s_1]_2 = [s]_2 \wedge \forall i \in [\nu], \forall j \in [2^{i-1}], [s_{2^{i-1}+j}]_2 = y_i [s_j]_2$$

where $r, x_i \leftarrow\!\$\ \mathbb{Z}_q$ and $s, y_i \leftarrow\!\$\ \mathbb{Z}_q$ for all $i \in [\nu]$.

The structured AFGHO commitment is additively homomorphic, perfectly hiding and computationally binding under the SXDH and DPair-ML assumptions.

Hash functions. A cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$ is a function from the domain of arbitrary bit string to the domain of λ-bit strings, where λ is the security parameter. It should satisfy that:
Collision resistance. Cannot find two strings m and m' ($m \neq m'$) such that $H(m) = H(m')$ except with negligible probability in λ.
Preimage resistance. Given y, cannot find x, such that $H(x) = y$, except with negligible probability in λ.

Random oracle model. The random oracle model [37] is a theoretical framework used in cryptography to analyze the security of cryptographic protocols. In this model, a random oracle is an idealized black box that provides truly random responses to every unique query. It helps in proving the security of cryptographic schemes by assuming the existence of a perfect hash function that behaves like a random function.

Public key encryption. A public key encryption scheme Π_{PKE} for encrypting a message m has the following algorithms:
$\mathsf{KeyGen}(pp) \to (pk, sk)$: takes public parameter pp as input, outputs the public encryption key pk and secret decryption key sk.
$\mathsf{Enc}(m, pk, r) \to c$: given the message m, public key pk and randomness r, computes a ciphertext c.
$\mathsf{Dec}(c, sk) \to m$: given the ciphertext c, secret key sk, computes a message m.

The public key encryption scheme should satisfy the correctness and IND-CPA security:
Correctness. The decryption of ciphertext results in the original message provided that the correct secret key is used.
IND-CPA security. The encryption scheme must achieve the security of indistinguishability under the chosen plaintext attack (IND-CPA), which is equivalent to semantic security and essentially captures that only negligible information about the message can be feasibly extracted from the ciphertext.

Digital signatures. A digital scheme Π_{DS} for signing a message m has the following algorithms:

$\mathsf{KeyGen}(pp) \rightarrow (pk, sk)$: takes public parameter pp as input, outputs the public verification key pk and secret signing key sk.

$\mathsf{Sign}(m, sk) \rightarrow \sigma$: given the message m and secret key sk, computes a signature σ.

$\mathsf{Verify}(pk, m, \sigma) \rightarrow 0/1$: given the public key pk, message m and signature σ, output 0/1 indicating whether it is a valid signature.

The signature scheme is required to satisfy the following correctness and unforgeability:

Correctness. If a message is signed with the correct secret key, anyone can verify the signature with the corresponding public key and be confident that the signature is valid for that message.

unforgeability. Anyone cannot generate a valid signature on any message without the secret key, even if they have access to the public key and possibly some signed messages.

Zero-Knowledge Arguments of Knowledge (ZKAoK). A zero-knowledge argument of knowledge is a cryptographic protocol involving two parties: a prover and a verifier. In this protocol, the prover's objective is to provide convincing proof to the verifier that a certain statement is true, without revealing any information about the underlying witness.

It consists of three PPT algorithms Setup, $\mathcal{P}$, and $\mathcal{V}$. The setup algorithm outputs a common reference string σ on inputting a security parameter λ. The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ are interactive algorithms. As the output of this protocol, we use the notation $\langle \mathcal{P}, \mathcal{V} \rangle = b$, where $b = 1$ if $\mathcal{V}$ accepts and $b = 0$ if $\mathcal{V}$ rejects. The proof is *public coin* if an honest verifier generates his responses to $\mathcal{P}$ uniformly.

Argument of knowledge. $(\textbf{Setup}, \mathcal{P}, \mathcal{V})$ is called an argument of knowledge for the relation $\mathcal{R}$ if it satisfies the following two definitions.

Perfect completeness. The prover can persuade the verifier if it possesses a witness that attests to the truth of the statement.

Computational witness-extended emulation. Whenever an adversary produces an acceptable argument with some probability, there exists an emulator that can produce a similar argument with the same probability and provide a witness w simultaneously. It implies *soundness* which asserts that no PPT adversary can persuade the verifier when the statement is false. It

also assures *knowledge soundness* which guarantees the existence of an extractor capable of producing a valid witness for the statement.
Honest-verifier special zero-knowledge (HVSZK). Given the verifier's challenge values, it is possible to simulate the entire argument without witness efficiently.

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).
$\mathsf{Com}(\vec{m}, r) \to c$: 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 (pp, pk, sk, c) and $(pp, pk, \vec{m}, r)$ respectively. User outputs a signature σ.
$\mathsf{Vrfy}(pp, pk, \vec{m}, \sigma) \to b$: it checks $(\vec{m}, \sigma)$ pair and outputs 0/1.

Besides the correctness and unforgeability similar to the digital signature, We require a blind signature scheme to be *blind*: the signer does not learn anything about the original message during the signing process. The user blinding the message hides the content of the message from the signer, ensuring that the signer cannot link the signature to any specific message or identify the content of the signed message.

Partially blind signature. A partially blind signature is a variant of the blind signature, where the signed message is partially blind. A partially blind signature Π_{pbs} consists following three algorithms:
$\mathsf{KeyGen}(pp, 1^n) \to (pk, sk)$: generates a public and secret key pair (pk, sk).
$\langle \mathsf{PartialBlindRcv}, \mathsf{PartialBlindSign} \rangle$: it is an interactive protocol between the user and signer with inputs $(pp, pk, msg, info)$ and $(pp, pk, sk, info)$ respectively, where msg denotes the blind part of signed message, and $info$ denotes the unblind part of signed message. User outputs $\perp$ or the message signature pair $(msg, info, \sigma)$, and signer outputs $b = 0/1$.
$\mathsf{Vrfy}(pp, pk, msg, info, \sigma) \to b$: it checks $(msg, info, \sigma)$ pair and outputs 0/1.

We require a partially blind signature scheme to satisfy the correctness, unforgeability, and *partial blindness* as defined in [38]. Especially, partial Blindness refers to the property where only part of the message is hidden from the signer, while some parts of the message remain

visible. This allows the user to selectively reveal some parts of the message while keeping other parts confidential.

BLS aggregate signature. We briefly review the BLS signature scheme and its signature aggregation mechanism [35]. Given an efficiently computable non-degenerate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ in groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order q. Let g and $\tilde{g}$ be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, a hash function $H : \mathcal{M} \to \mathbb{G}_1$:
KeyGen(): the user chooses $sk \leftarrow\$ \mathbb{Z}_q$, outputs (pk, sk) for $pk \leftarrow \tilde{g}^{sk} \in \mathbb{G}_2$.
Sign(sk, m): output $\sigma \leftarrow H(m)^{sk} \in \mathbb{G}_1$.
Vrfy(pk, m, σ): output 1 if $e(\sigma, \tilde{g}) = e(H(m), pk)$, otherwise, output 0.
Signature Aggregation: Given triples (pk_i, m_i, σ_i) for $i \in [n]$, anyone can aggregate the signatures $\sigma_1, ..., \sigma_n$ into a single group element $\hat{\sigma} \leftarrow \Pi_{i\in[n]}\sigma_i \in \mathbb{G}_1$. Verification can be done by checking that if

$$e(\hat{\sigma}, \tilde{g}) = e(H(m_1), pk_1) \cdots e(H(m_n), pk_n).$$

For all same messages it just needs to check if $e(\hat{\sigma}, \tilde{g}) = e(H(m_1), \Pi_{i=1}^{n} pk_i)$.
It is *unforgeable* under the co-CDH assumption.

The rogue public-key attack and defense. Note that the aggregate public key $\Pi_{i=1}^{n} pk_i$ suffers the rogue public-key attack [39]. To prevent it, we use the Proof-of-Possession (PoP) mechanism [40] in the *registered key model*. In this approach, each party is required to provide a proof that they possess the private key corresponding to their public key. This proof can be included during the setup phase to ensure that only legitimate key owners can participate. In this paper, we implicitly assume the presence of PoP proofs for the public keys.

PS Signatures on Committed Multi-message.[41] The signature scheme for signing information-theoretically hidden messages consists of the following algorithms:
Setup(1^k): Given a security parameter k, this algorithm outputs $pp \leftarrow (p, G_1, G_2, G_T, e)$. These bilinear groups must be of type III. In the following, we denote $G_1^* = G_1 \setminus \{1_{G_1}\}$ and $G_2^* = G_2 \setminus \{1_{G_2}\}$, which are the sets of the generators.
KeyGen(pp): This algorithm selects $g \xleftarrow{\$} G_1^*$, $\tilde{g} \xleftarrow{\$} G_2^*$, and $(x, y_1, \ldots, y_r) \xleftarrow{\$} \mathbb{Z}_p^{r+1}$, computes $(X, Y_1, \ldots, Y_r) \leftarrow (g^x, g^{y_1}, \ldots, g^{y_r})$ and $(\tilde{X}, \tilde{Y_1}, \ldots, \tilde{Y_r}) \leftarrow (\tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_r})$, and

sets $sk \leftarrow X$ and $pk \leftarrow (g, \tilde{g}, \tilde{X}, Y_1, \ldots, Y_r, \tilde{Y_1}, \ldots, \tilde{Y_r})$.

Protocol: A user who wishes to obtain a signature on $(m_1, \ldots, m_r)$ first selects a random $t \xleftarrow{\$} \mathbb{Z}_p$ and computes $C \leftarrow g^t \prod_{i=1}^{r} Y_i^{m_i}$. He then sends C to the signer. They both provide proof of knowledge of the opening of the commitment. If the signer is convinced, he selects a random $u \xleftarrow{\$} \mathbb{Z}_p$ and returns $\sigma' \leftarrow (g^u, (XC)^u)$. The user can now unblind the signature by computing $\sigma \leftarrow (\sigma'_1, \sigma'_2/\sigma'^t_1)$.

Secure Multiparty Computation (MPC). We use the standard definitions of secure multiparty computation, following the stand-alone model [42]. We distinguish between two security settings: a semi-honest adversary, which follows the protocol but tries to learn additional information from the messages it receives, and a malicious adversary, which may act arbitrarily.

Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary controlling a subset of parties $I \subset [n]$. The real-world execution is denoted by $\mathsf{REAL}_{\Pi,\mathcal{A}(z),I}(x_1, ..., x_n, \lambda)$, where Π is the protocol, the parties have inputs $x_1, ..., x_n$, the adversary $\mathcal{A}$ has auxiliary input z, and λ is the security parameter. Similarly, let $\mathcal{S}$ be a non-uniform probabilistic polynomial-time adversary in the ideal world. The ideal-world execution is denoted by $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(x_1, ..., x_n, \lambda)$, where the functionality $\mathcal{F}$ replaces Π, and the trusted party performs the computation.

The security definition requires that for every real-world adversary $\mathcal{A}$, there exists an ideal-world adversary $\mathcal{S}$ such that the outputs of REAL and IDEAL are computationally indistinguishable. We refer [42] for the full definition.

Hybrid model and composition. In the hybrid model, a protocol $\Pi_{\mathcal{F}}$ is executed as in the real model, but the parties also have access to a trusted party that computes the functionality $\mathcal{G}$. The composition theorem from [42] ensures that if $\Pi_{\mathcal{F}}$ securely computes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model and there is a secure protocol $\Pi_{\mathcal{G}}$ for $\mathcal{G}$, then replacing calls to $\mathcal{G}$ in $\Pi_{\mathcal{F}}$ with executions of $\Pi_{\mathcal{G}}$ securely computes $\mathcal{F}$ in the real world.

Modeling security. For some protocols, we may want to model security under different assumptions, such as malicious P_i but semi-honest P^*. In the malicious setting, we quantify adversaries controlling corrupted parties that deviate from the protocol. In the ideal world,

the simulator receives the inputs of corrupt parties, while in the malicious case, it typically extracts the inputs used in the protocol. For semi-honest adversaries, the simulator simply forwards the inputs to the trusted party.

Basic functionalities. We introduce some useful functionalities here.

Coin tossing. We define the coin-tossing functionality, $\mathcal{F}_{\mathsf{CT}}$. The functionality assumes two parties, P_0 and P_1. Each party inputs 1^λ and some parameter $\ell > \lambda$. The functionality samples a uniform string $s \leftarrow \{0,1\}^\ell$, sends to P_0, P_1.

Three-party Secure Comparison. $(\emptyset, \emptyset, \mathsf{w}) \leftarrow \mathcal{F}_{\mathsf{3SC}}([\mathsf{v}]_0, [\mathsf{v}]_1, [\mathsf{v}]_2)$: There are three parties P_0, P_1, P_2 take $[\mathsf{v}]_0, [\mathsf{v}]_1, [\mathsf{v}]_2$ as input and send to the functionality. They are shares of a secret value v and additively homomorphic. $\mathcal{F}_{\mathsf{3SC}}$ computes one bit w. If $\mathsf{v} < 0$, then $\mathsf{w} = 1$. Otherwise, $\mathsf{w} = 0$. $\mathcal{F}_{\mathsf{3SC}}$ only sends w to P_2.

2.4 Blockchain-related Concepts

Blockchain. A blockchain functions as a public ledger, which is essentially an ever-growing transaction log collectively maintained by a network of untrusted Internet nodes. An honest node within this network, known as an honest full node, ensures its blockchain replica remains consistent with those of all other honest nodes by adhering to a set of predefined rules called the consensus protocol.

Ethereum. Ethereum is a decentralized, open-source blockchain system that features *smart contract* functionality [7]. The native cryptocurrency of the Ethereum platform is called Ether (ETH). It is used to pay for transaction fees and computational services on the Ethereum network. It allows developers to create and deploy smart contracts, which are self-executing contracts with the terms of the agreement directly written into code. These contracts automatically execute and enforce the terms when certain conditions are met. Transactions and smart contract executions on the Ethereum network require *gas*, which is a unit of measure for computational work. Users pay gas fees in ETH to incentivize miners (or validators) to process and validate transactions.

Accounts. In Ethereum, the state comprises objects called accounts. In general, there are two types of accounts: *externally owned accounts* (EOA), controlled by private keys, and *contract accounts*, controlled by their contract code. An EOA has no code, and one can send messages from an EOA by creating and signing a transaction; in a contract account, every time the contract account receives a message, its code activates, allowing it to read and write to internal storage and send other messages or create contracts in turn.

Transactions. A full transaction in Ethereum is a signed data package that includes information such as: {`Nonce, GasPrice, GasLimit, To, Value, Data, (v,r,s)`}.

`Nonce` is a counter used to ensure each transaction is only processed once. `GasPrice` is the amount users are willing to pay per unit of gas (measured in Gwei). `GasLimit` is the maximum amount of gas users are willing to spend on the transaction. `To` is the recipient's address (could be a smart contract address). `Value` is the amount of ETH to send. `Data` is where the smart contract call happens. It includes the function being called and its parameters encoded in hexadecimal. `(v, r, s)` are the signature values of the transaction.

The data field has no function by default. But it contains the data for calling a smart contract. For example, in the anonymous Zether contract, the sender generates a meta-transaction $\mathsf{meta}_{\mathsf{AZ}}$. The contract would read this data, verify it, and execute it if it is valid. To distinguish the "Ethereum transaction" and "contract meta-transaction", we explicitly call the Ethereum transaction as *full transaction* and call the contract meta-transaction as *meta-transaction*.

Anonymous Zether. Anonymous Zether [16, 17] is a protocol in which a sender may hide herself and the recipient in a larger public key ring[1] $(y_i)_{i=0}^{N-1}$. The goal is to make it impossible for an observer to determine which ring member sent or received money. Specifically, the sender chooses that public key ring, as well as indices l_0 and l_1 denote the sender and recipient, respectively. The sender publishes the list, along with a list of ciphertexts $(C_i, D)_{i=0}^{N-1}$, where (C_{l_0}, D) encrypts g^{-b^*} under y_{l_0}, (C_{l_1}, D) encrypts g^{b^*} under y_{l_1}, and (C_i, D) for $i \notin \{l_0, l_1\}$ encrypts g^0 under y_i. To apply the transfer, the contract homomorphically adds (C_i, D) to

[1]Each participant has proven knowledge of her own public key before participating in the contract (i.e., before appearing in any anonymity set) to prevent rogue public key attack.

each y_i's balance. The resulting list of new balances is denoted $(C_{Ln,i}, C_{Rn,i})_{i=0}^{N-1}$. Finally, the prover should demonstrate knowledge of:

- Indices $l_0, l_1 \in \{0, \ldots, N-1\}$ (sender and recipient's secret indices),
- sk for which $g^{sk} = y_{l_0}$ (knowledge of the secret key),
- r such that:
  - $g^r = D$ (knowledge of randomness),
  - $(y_{l_0} \cdot y_{l_1})^r = C_{l_0} \cdot C_{l_1}$ (ensuring the sender's and receiver's ciphertexts encrypt opposite balances),
  - $y_i^r = C_i$ for all $i \notin \{l_0, l_1\}$ (ensuring all other ciphertexts encrypt 0).
- Values b^* and b' in $\{0, \ldots, max\}$ such that:
  - $C_{l_0} = g^{-b^*} \cdot D$,
  - $C_{Ln,l_0} = g^{b'} \cdot C_{Rn,l_0}$ (ensuring overflow and overdraft protection).

Formally, the following relation is defined:

$$\begin{aligned}\mathcal{R}_{\mathsf{AZ}} :\{&(y_i, C_i, C_{Ln,i}, C_{Rn,i})_{i=0}^{N-1}, D, u, g_{\mathsf{epoch}}; sk, b^*, b', r, l_0, l_1 \mid \\ &g^{sk} = y_{l_0} \wedge C_{l_0} = g^{-b^*} D^{sk} \wedge C_{Ln,l_0} = g^{b'} C_{Rn,l_0} \wedge g_{\mathsf{epoch}}^{sk} = u \wedge \\ &D = g^r \wedge (y_{l_0} \cdot y_{l_1})^r = C_{l_0} \cdot C_{l_1} \wedge y_i^r = C_i \ \forall i \notin \{l_0, l_1\}, \\ &b^*, b' \in \{0, \ldots, max\}\}.\end{aligned}$$

The meta-transaction of Anonymous Zether is $\mathtt{meta}_{\mathtt{AZ}} = \{u, D, (y_i, C_i, C_{Ln,i}, C_{Rn,i})_{i=0}^{N-1}, \pi_{\mathtt{AZ}}\}$, where $\pi_{\mathtt{AZ}}$ is the non-interactive zero-knowledge proof for the relation $\mathcal{R}_{\mathsf{AZ}}$.

To defend the replay and double-spending attack, the anonymous Zether defines nonce based on the epoch. An epoch is a discrete period (usually several blocks). Each epoch is associated with a unique base value (g_{epoch}). An meta-transaction must include a nonce $u = g_{\mathsf{epoch}}^{sk}$ where sk is the sender's secret key. This nonce prevents replay and double-spending attacks by ensuring that transactions are valid only within the current epoch. Note that it differs from the `Nonce` in the full transaction.

CHAPTER 3

Pisces: Private and Compliable Cryptocurrency Exchange

3.1 Background

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 in usability and even regulatory reasons. For example, the annual trading volume of Binance was up to 9580 billion USD in 2021 [43], and Coinbase also had 1640 billion USD in 2021 [44].

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 a passport or a stamped envelope with an address, for the platform to verify. Bank information is also given for trading purposes.

[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.3.1, and Sec.3.3.1 for details.

3.1.1 Motivation

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 [45]. 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 amplifies the impacts of potential privacy breaches (in exchange for 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 account on a blockchain.

Since most of the blockchains are transparent (except very few chains such as Monero [9], 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 from the platform and 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 at least restores user anonymity/privacy so that external records are not directly linked to the real identity.

3.1.2 Challenges

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 layer-2 payment solutions [19, 21, 22, 23, 46, 47] and smart contract enabled private payment solutions [16, 17] 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 [19, 21, 22] or the leader in [23] or only outsiders [47], *not against the payer himself*. In our case, the platform is the payer and knows the payee's exact address during a coin withdrawal.

Recent works of [20, 21] even considered anonymous (k-anonymity) payment hubs against payers, assuming 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 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 systems.

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 exchanges 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 a 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) [48, 49, 50] 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 that can learn trivial information about the other.

Atomic swap across different ledgers supports the exchange between different cryptocurrencies. While atomic swap puts much effort into ensuring fairness, the only privacy-preserving atomic swap work [51] reduces the confidentiality and anonymity properties of 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 [52]. During withdrawals, the client chooses one UTXO and mixes it among k transactions. To prevent linkability by the transaction amount, each withdrawal must 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 withdrawal 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 have been

withdrawn by someone to an external address, can still reduce the anonymity set. It is hard to have reasonably stronger anonymity without full internal transaction privacy (excluding the inherent leakage during withdrawal/deposit), as it is unclear what the actual consequence of each specific leakage is. 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. 3.3.2.

Preserving major compliance functionalities. We also need to preserve all the critical functionalities that are currently provided by centralized exchange platforms, including compliance such as 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 [53], 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 each account's transaction history and 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 users' withdrawal requests.

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 does not know the actual profit or 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.

[2]There are some related works in accountable privacy (e.g., PGC [24], UTT [26], Platypus [27], [25] etc), but they only focus on the payment with a single kind of asset and enforce limits on one transaction amount or account balance or sum of all sent or received values. Note that these compliance requirements cannot cover the profit computation, which uses the specific buying price without linking to that transaction.

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 [54] may enable succinct proof size and verification time. However, proof generation incurs heavy computing costs for users. Σ-protocols may also be useful, but hiding the exchanged asset types in all n kinds 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).

Technical overview. First, we 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 platform with a real identity 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 it to an authority. The platform also generates information to check its own solvency.

We use Fig 3.1 to visualize these (simplified) procedures. Each interactive protocol can be expressed by ① ② ③ steps: ① *Transaction request*; ② *Transaction processing*: platform verifies the transaction and process it; ③ *Transaction completion*: user completes the transaction.

The user sends a compliance report to the authority who verifies it in step ④: *Compliance verification*. In step ⑤ *Compliance check*, the platform checks whether its internal state satisfies 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 outgoing transactions (withdrawals) 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

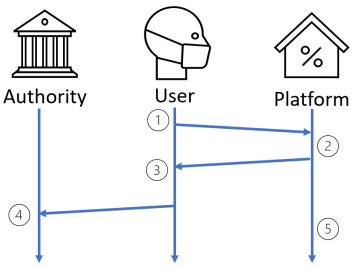

FIGURE 3.1. Overview of exchange system

(simply via a "partially" blind signature) that later can be used to make withdrawals. Users also submit a committed record containing the token's asset details 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 one-time credentials (a valid signature on a random identifier and transaction, etc), which easily prevents double-spending. Since the user balance is also hidden, all future operations, including exchange and withdrawal preparation, will involve (efficient) zero-knowledge proof 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 BTCs as a gift to any person who may not even registered with the platform. Then, the gift receiver could use the credential to do anonymous trading with the platform without revealing his real-world identity, which is not compliant with basic KYC regulations. 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, revealing this information directly to the platform would compromise user privacy when generating the compliance report. 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 the 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 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 the computation cost is 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 [55, 17].

Instead, we let the platform generate signatures on each asset name and the price and make them public, called *price credential*. To capture price fluctuations 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 newly 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. 3.4.

3.2 Related Works

Fiat to cryptocurrency (F2C) exchange. In general, the centralized F2C exchange platform does not consider user's privacy, like Coinbase, Binance. They collect user's personal information when they register to meet the KYC requirement. However, the user's accounts are transparent for the platform. It knows their asset profile, i.e., which kinds and how many assets they own. In the case of cryptocurrency, it would also know how the user spend their cryptocurrency which violates user's privacy outsides the platform.

To prevent the linkability by the transaction amount, the amount of withdrawn cryptocurrency is fixed for all transactions. For example, let all transactions be worth 1 Bitcoin. To prevent linkability by the input UTXO, the client should choose it. However, two clients may choose the same UTXO, and the conflict leads to only one of them receiving the bitcoin. Besides, it is not accountable. The users do not need to provide any compliance information otherwise their privacy cannot be preserved.

A privacy-preserving fiat-to-Bitcoin exchange scheme is proposed in [52]. In this scheme, a user can acquire a fixed quantity of cryptocurrency from an exchange platform using fiat

currency, all the while ensuring that the platform remains unaware of the connection between the user's genuine identity and the associated Bitcoin address. To achieve this, a blind signature mechanism is employed, allowing the user to receive Bitcoin from the platform without revealing the output address linked to the transaction. Subsequently, this transaction is recorded on the Bitcoin blockchain, divulging details such as the output address, transaction amount, and the Unspent Transaction Output (UTXO) utilized by the platform at that moment. To mitigate the risk of linkability through transaction amounts, a constant withdrawal amount is maintained across all transactions. For example, all transactions could be set at a fixed value of 1 Bitcoin. To counteract the potential issue of linkability through input UTXOs, clients are required to select their preferred UTXOs. However, a challenge arises when multiple clients opt for the same UTXO, potentially resulting in a conflict where only one of them receives the Bitcoin. Furthermore, this approach lacks accountability. Crucially, users are not obligated to furnish any compliance-related information. Failure to do so would compromise their privacy preservation.

Private decentralized exchange. Decentralized exchange allows users to exchange cryptocurrencies with each other directly or with smart contracts. However, it is very different from our setting. In the one hand, the private DEX focuses on the trade anonymity and trade confidentiality. It aims to keep the transaction information secret except for the trading parties. But in the CEX the platform is one of the trading party who can learn the information of the other one. On the other hand, it does not support fiat money transactions and they are generally deployed in the decentralized setting like smart contract that is unaffected by the KYC requirement. Users are free to join the DEX without providing their real identities as long as they have cryptocurrencies. It is hard to directly enforce compliance requirements since enrollment does not require real-world identities.

There are some works on the private exchange in the decentralized setting like Zexe [48], P2DEX [49] and Manta [50], but they do not consider any compliance issue. P2DEX [49] is a privacy preserving exchange system for cryptocurrency tokens cross different blockchains while preserving order privacy to avoid front-running attacks and ensuring users never lose tokens. They use MPC to privately match exchange orders and deploy smart contracts to

reimburse affected clients for the collateral deposit from the cheating server. Manta [50] is a decentralized anonymous exchange scheme based on an automated market maker (AMM). They design a mint mechanism to convert base coins to private coins, then achieve the decentralized anonymous exchange by trading private coins anonymously.

3.3 Problem Formulation

3.3.1 Syntax

In this section, we define the syntax that is abstracted from real exchange systems and is general for both plain and private centralized exchange systems. Basically, an exchange system supports users depositing multiple kinds of assets (including fiat money), exchanging assets with the platform, and withdrawing assets. To comply with regulations, the system also checks compliance with platform rules and supports users in filing their compliance documents.

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, $\mathsf{PKeyGen}$, Verify, Check as well as interactive protocols: $\langle \mathsf{Join}, \mathsf{Issue} \rangle$, $\langle \mathsf{Deposit}, \mathsf{Credit} \rangle$, $\langle \mathsf{Exchange}, \mathsf{Update} \rangle$, $\langle \mathsf{Withdraw}, \mathsf{Deduct} \rangle$, $\langle \mathsf{File}, \mathsf{Sign} \rangle$. 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 $req_{exc}.name_{in}$, $req_{exc}.amt_{in}$ are the exchange-in asset name and amount; $req_{exc}.name_{out}$, $req_{exc}.amt_{out}$ are the exchange-out asset name and 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 the join transaction, updated in the exchange and withdrawal 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. $Rd_{ast}.non$ 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 names 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 $\emptyset$.
- $\langle \mathsf{Join}, \mathsf{Issue} \rangle$: It is a register protocol. U runs the interactive algorithm $\mathsf{Join}(\boldsymbol{epp}, \boldsymbol{pk}, req_{joi})$, and P runs the interactive algorithm $\mathsf{Issue}(epp, pk, sk, st)$, 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}.
- $\langle \mathsf{Deposit}, \mathsf{Credit} \rangle$: It is a deposit transaction for users to deposit assets to P. P runs $\mathsf{Credit}(epp, pk, sk, st)$ and U runs $\mathsf{Deposit}(epp, pk, uid, Rd_{reg}, req_{dep})$. 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.
- $\langle \mathsf{Exchange}, \mathsf{Update} \rangle$: 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 $\mathsf{Exchange}(\boldsymbol{epp}, pk, uid, Rd_{reg}, Rd_{ast}, req_{exc})$, and P runs $\mathsf{Update}(\boldsymbol{epp}, pk, sk, st)$. 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 $req_{exc}.name_{out}$.
- $\langle \mathsf{Withdraw}, \mathsf{Deduct} \rangle$: 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 $\mathsf{Withdraw}(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{wit})$, and P runs $\mathsf{Deduct}(epp, pk, sk, st)$. 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}.
- $\langle \mathsf{File}, \mathsf{Sign} \rangle$: It is a two-party protocol in which U files compliance information periodically and requests P to sign it. U runs $\mathsf{File}(epp, pk, uid, Rd_{reg}, req_{fil})$ and P runs $\mathsf{Sign}(epp, pk, sk, st)$. 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. It outputs a bit b indicating a passing check or not.
- 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.

3.3.2 Security Model

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 for 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 involves four aspects. (1) we place less trust in the platform than in existing plain exchange systems where the platform is always assumed to be honest. Although 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 for the 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. 3.3.2; (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 will give a high-level description of the security requirements.

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 the user has to expose the identity, such as depositing/withdrawing fiat money from/to bank. We start discussing it from the basic anonymity where only focusing on the withdraw or deposit transactions. Although the basic anonymity scheme could be simple and not bring extra challenges to compliance (especially platform compliance), we show that the basic withdraw anonymity 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.

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 only consider privacy when trading in cryptocurrency. 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 [56, 57].

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[3] 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,

[3]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.

where oracle runs $\mathsf{Deposit}(epp, pk, uid, Rd_{reg}, req_{dep}) \rightarrow (Rd'_{reg}, Rd^{out}_{ast})$. If Deposit algorithm outputs $\perp$, then oracle outputs $\perp$; otherwise, oracle updates the map by setting $\text{MAP}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$ and adds a new tuple $(uid, ref^{\mathsf{out}}_{\mathsf{ast}}, Rd^{out}_{ast})$ to MAP. $\mathcal{A}$ gets interaction transcripts.

- $\mathcal{O}^1_{\mathsf{Exchange}}(uid, req_{exc}, ref_{\mathsf{reg}}, ref^{\mathsf{in}}_{\mathsf{ast}}, ref^{\mathsf{out}}_{\mathsf{ast}})$: oracle first gets records $Rd_{reg} = \text{MAP}(uid, ref_{\mathsf{reg}})$, $Rd_{ast} = \text{MAP}(uid, ref^{\mathsf{in}}_{\mathsf{ast}})$ from 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 Exchange algorithm outputs $\perp$, then oracle outputs $\perp$; otherwise, oracle updates the map by setting $\text{MAP}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$ and $\text{MAP}(uid, ref^{\mathsf{in}}_{\mathsf{ast}}) \leftarrow Rd'_{ast}$, and adds a new tuple $(uid, ref^{\mathsf{out}}_{\mathsf{ast}}, Rd^{out}_{ast})$ to 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 $Rd_{reg} = \text{MAP}(uid, ref_{\mathsf{reg}})$, $Rd_{ast} = \text{MAP}(uid, ref^{\mathsf{in}}_{\mathsf{ast}})$ from 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 Exchange algorithm outputs $\perp$, then oracle outputs $\perp$; otherwise, oracle updates the map by setting $\text{MAP}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$, $\text{MAP}(uid, ref^{\mathsf{in}}_{\mathsf{ast}}) \leftarrow Rd'_{ast}$. $\mathcal{A}$ gets interaction transcripts but no more output from oracle.
- $\mathcal{O}^1_{\mathsf{File}}(uid, req_{fil}, ref_{\mathsf{reg}})$: oracle first get records $Rd_{reg} = \text{MAP}(uid, ref_{\mathsf{reg}})$ from MAP per references. Then 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 $\text{MAP}(uid, ref_{\mathsf{reg}}) \leftarrow Rd'_{reg}$. $\mathcal{A}$ 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 one bit b indicating whether the operation succeeds or not. If $b = 0$, it outputs $\perp$. Otherwise, $\mathcal{A}$ gets $\{Rd_{reg}\}$.
- $\mathcal{O}^2_{\mathsf{Update}}$: $\mathcal{A}$ 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 transcript ts as external input.

It outputs a signal bit b, indicating whether the operation succeeds or not. If $b = 0$, it outputs $\perp$. Otherwise, $\mathcal{A}$ gets $\{Rd'_{reg}, Rd'_{ast_i}, Rd_{ast_j}\}$.

- $\mathcal{O}^2_{\mathsf{Credit}}$: 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.

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 Figure 3.2, 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
  $req_{wit}.amt \neq \textsc{Map}(uid_1, ref^1_{\mathsf{ast}}).amt$
  return 0
else $b \leftarrow\$ \{0, 1\}$
Interact 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 $(\hat{b} == b)$

FIGURE 3.2. Basic withdraw anonymity experiment

DEFINITION 8 (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. 3.2, it holds that* $|\Pr[\mathrm{Exp}^{\mathrm{ano-wit}}(\mathcal{A}, \lambda) = 1] - 1/2| \leq negl(\lambda)$

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 withdrawal anonymity, except that all deposit requests are achievable naturally for any user.

Limitations of basic withdraw anonymity. The basic anonymity model is simple, but we observe limited anonymity. It is well-known that the strength of anonymity 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 whose account balance exceeds the withdrawn amount.

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.

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 on it later. For other kinds of leakage, we check whether it is even worth preventing, as any protective measure comes with a 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 called interactive indistinguishability. It achieves the best possible anonymity by constraining 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 can withdraw. It is acceptable for some popular assets that many people own, but the withdrawal amount is small, such that the anonymity set is large enough. But it rules out many interesting scenarios, such as withdrawing some special assets owned by a small number of people or a comparatively large amount of assets that few people have so much. 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 withdrawal transaction consists of all registered users in the system, which is called perfect anonymity. Unfortunately, this cannot be true since the platform can always exclude some users using some public information. For example, given a withdrawal of 100 BTCs and a newly enrolled user, Alice, she is in no way the user of this withdrawal if there is no such large amount of deposit in the system after her registration.

Due to the special setting of the 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, 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 bit more, like the information (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 that 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.

Apparently, the identity of the 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 who register and do transactions, and then David and Ella join. 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 the 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 the exchange is leaked, the platform knows that David and Ella cannot withdraw 5 XRPs, and then both David and Ella are excluded from the anonymity set.
3. If the exchange content 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 that just leaking the summary of compliance information is fine. However, in this case, many users might not have generated any transactions in a 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 full anonymity via interaction indistinguishability.

Interaction indistinguishability. This property requires that the interaction between the user and platform leak nothing except public information. We design the experiment to include the interaction of all kinds of transactions. At 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 pairs. This 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. 9). 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 give the formal definition in Fig 10.

DEFINITION 9 (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 deposits/withdrawals, the users and bank accounts are the same.*

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

$epp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda))$
$(pk, st) \leftarrow \mathcal{A}(epp)$
$\mathcal{C}$ 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$(\hat{b} == b)$

FIGURE 3.3. Interaction indistinguishability experiment

DEFINITION 10 (Interaction indistinguishability). *The interaction indistinguishability is described in Fig 3.3. We say that an exchange system provides interaction indistinguishability if for all PPT* $\mathcal{A}$ *and* λ *it holds that* $|\Pr[\mathrm{Exp}^{\mathsf{IND}}(\mathcal{A}, \lambda) = 1] - 1/2| \leq negl(\lambda)$

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

REMARK 2 (**Best possible anonymity**). *We claim that the interaction indistinguishability achieves the best 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 9, we can ensure that the platform **learns nothing** about the user from their interactions, except for the unavoidable leakage. Recall 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.*

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

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 [58]. Then, the experiment is able to check if any overdraft or compliance cheating happens.

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.

DEFINITION 11 (Overdraft Prevention). *As shown in Figure 3.4, 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* $\Pr[\mathrm{Exp}^{\mathrm{od}}(\mathcal{A}, \mathcal{E}, \lambda) = 1] \leq negl(\lambda)$

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

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

$epp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda)), (1^n, st) \leftarrow \mathcal{A}(epp)$, for some $n \in \mathbb{N}$
$(pk, sk) \leftarrow \mathsf{PKeyGen}(epp, 1^n)$
Run $\mathcal{A}^{\mathcal{O}_{\text{od}}}(epp, pk, st)$
if any oracle aborts **then return** 0;
else continue until $\mathcal{A}$ halts
Run $\{h_t\} \leftarrow \mathcal{E}^{\mathcal{A}}(epp)$ // $\mathcal{E}$ controls $\mathcal{A}$'s randomness
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
$-(Rd_{ast}.amt - Rd'_{ast}.amt) \cdot pub_t.pr_{in} \neq Rd_{ast}.amt \cdot pub_t.pr_{out}$
// 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 $\mathsf{RdSet} \leftarrow \mathsf{RdSet} \setminus \{Rd_{reg}, Rd_{ast}\} \cup \{Rd'_{reg}, Rd'_{ast}\}$
return 0

FIGURE 3.4. Overdraft prevention experiment.

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 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 3.5 for details.

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

$epp \leftarrow \mathsf{Setup}(\mathcal{G}(1^\lambda)), (1^n, st) \leftarrow \mathcal{A}(epp)$, for $n \in \mathbb{N}$
$(pk, sk) \leftarrow \mathsf{PKeyGen}(epp, 1^n)$, $\mathsf{RU} \leftarrow \emptyset$
Run $doc^* := (uid,cp,mt,sig)/\perp \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{clie-comp}}}(epp,pk,st)$
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 $\mathcal{A}$'s randomness
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;
  $-uid \notin \mathsf{RU}$;
//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

FIGURE 3.5. F-Client-Compliance experiment.

DEFINITION 12 (Client Compliance). *The client compliance experiment is shown in Figure 3.5. 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 to 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 [59].

3.4 Fully Private and Compliable Exchange System

In this section, we present the generic construction of the fully private and compliable exchange system Π_{Pisces} that achieves full anonymity, overdraft prevention, and compliance, and we provide a 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. 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 [59], 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.

Regarding 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 3.6.

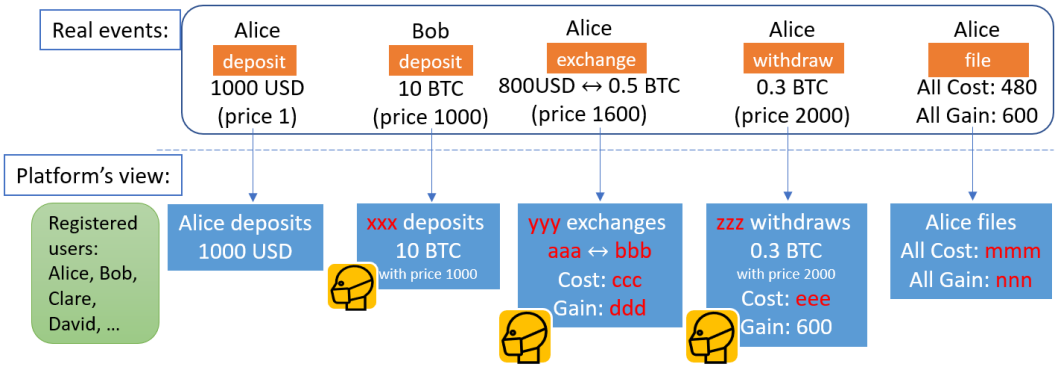

FIGURE 3.6. Platform's view in the exchange system

3.4.1 An Efficient Pisces Construction

In this section, we give an efficient Pisces construction from additive homomorphic commitment, blind signature, and zero-knowledge proof[4]. Let Com be an additively homomorphic commitment scheme, $\Pi_{\text{bs}} = (\mathsf{KeyGen}, \mathsf{Com}, \langle \mathsf{BlindSign}, \mathsf{BlindRcv} \rangle, \mathsf{Vrfy})$ be a blind signature scheme using Com to blind messages, and ZKAoK is the underlying proof system. The

[4]Note that these primitives are also used for updatable anonymous credentials and an incentive system in [60], but it does not support the exchange operation and compliance rule in our setting.

- 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: $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, pp)$
- $\langle \mathsf{Join}(epp,pk,req_{joi}), \mathsf{Issue}(epp,pk,sk) \rangle \to (uid, Rd_{reg}, b)$:
  //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 $Rd_{reg} = (uid, rid, cp_1, cp_2, \sigma_{reg})$
  $\sigma_{reg} \leftarrow \mathsf{BlindRcv}(pp, pk, \hat{\sigma}_{reg}, r)$
- $\langle \mathsf{Deposit}(epp,pk,uid, Rd_{reg}, req_{dep}), \mathsf{Credit}(epp, pk, sk) \rangle \to (Rd^{out}_{ast}, b)$:
  //$Rd_{reg} = (uid, rid, cp_1, cp_2, \sigma_{reg})$
  //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 $Rd^{out}_{ast} = (uid, aid, i, k_i, pr_i, \sigma_{ast})$
  //$\sigma_{ast} \leftarrow \mathsf{BlindRcv}(pp, pk, \hat{\sigma}_{ast}, r_2)$
- $\langle \mathsf{Exchange}(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{exc}), \mathsf{Update}(epp, pk, sk) \rangle \to (Rd'_{reg}, Rd'_{ast}, Rd^{out}_{ast}, b)$:
  //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: $rid', aid', aid^{out}, \{r_u\}_{u=1}^7 \leftarrow\!\$\ \mathbb{Z}_p$, sends $(rid, aid, \{com_u\}_{u=1}^7, \pi)$ 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 $k_i > 0$;
  – $\{com_u\}_{u=1}^5$ 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 $\hat{\sigma}_{reg}, \hat{\sigma}_{ast}, \hat{\sigma}^{out}_{ast}$
- $\langle \mathsf{Withdraw}(epp, pk, uid, Rd_{reg}, Rd_{ast}, req_{wit}), \mathsf{Deduct}(epp, pk, sk) \rangle \to (Rd'_{reg}, Rd'_{ast}, b)$:
  U: $rid', aid', \{r_u\}_{u=1}^4 \leftarrow\!\$\ \mathbb{Z}_p$, sends $(req_{wit}, rid, aid, \{com_u\}_{u=1}^4, \pi)$ 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
  $Rd'_{reg} = (uid, rid', cp'_1, cp'_2, \sigma'_{reg})$, $Rd'_{ast} = (uid, aid', i, v'_i, pr_i, \sigma'_{ast})$.
- $\langle \mathsf{File}(epp, pk, uid, Rd_{reg}, req_{fil}), \mathsf{Sign}(epp, pk, sk) \rangle \to (Rd'_{reg}, doc, b)$:
  //$Rd_{reg} = (uid, rid, cp_1, cp_2, \sigma_{reg})$, $req_{fil} = (uid, cp_1, cp_2)$
  U: $rid', \{r_u\}_{u=1}^3 \leftarrow\!\$\ \mathbb{Z}_p$, sends $(uid, rid, \{com_u\}_{u=1}^3, \pi)$ 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 $\mathsf{Vrfy}(pk, (uid, cp_1, cp_2, mt), \sigma) \to 1$, 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.

FIGURE 3.7. Our Pisces construction

platform maintains the registered user set USet and the identifier set ID which are initially empty. Formal construction is in Fig. 3.7. The concrete instantiation is presented in Sec. 3.5.

3.4.2 Security Analysis

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

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 9, 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, $Q^0 = (\mathit{uid}^0, \mathit{req}^0_{\mathit{dep}}, \mathit{ref}^0_{\mathsf{reg}}, \mathit{ref}^{\mathsf{out0}}_{\mathsf{ast}})$ and $Q^1 = (\mathit{uid}^1, \mathit{req}^1_{\mathit{dep}}, \mathit{ref}^1_{\mathsf{reg}}, \mathit{ref}^{\mathsf{out1}}_{\mathsf{ast}})$. 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, $Q^0 = (\mathit{uid}^0, \mathit{req}^0_{\mathit{exc}}, \mathit{ref}^0_{\mathsf{reg}}, \mathit{ref}^{\mathsf{in0}}_{\mathsf{ast}}, \mathit{ref}^{\mathsf{out0}}_{\mathsf{ast}})$ and $Q^1 = (\mathit{uid}^1, \mathit{req}^1_{\mathit{exc}}, \mathit{ref}^1_{\mathsf{reg}}, \mathit{ref}^{\mathsf{in1}}_{\mathsf{ast}}, \mathit{ref}^{\mathsf{out1}}_{\mathsf{ast}})$. 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, $Q^0 = (\mathit{uid}^0, \mathit{req}^0_{\mathit{wit}}, \mathit{ref}^0_{\mathsf{reg}}, \mathit{ref}^{\mathsf{in0}}_{\mathsf{ast}})$ and $Q^1 = (\mathit{uid}^1, \mathit{req}^1_{\mathit{wit}}, \mathit{ref}^1_{\mathsf{reg}}, \mathit{ref}^{\mathsf{in1}}_{\mathsf{ast}})$. 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 BlindSign and sends the blinded signatures to $\mathcal{C}$.
- For $\mathcal{O}^1_{\mathsf{File}}$ oracle, $Q^0 = (\mathit{uid}^0, \mathit{req}^0_{\mathit{fil}}, \mathit{ref}^0_{\mathsf{reg}})$, and $Q^1 = (\mathit{uid}^1, \mathit{req}^1_{\mathit{fil}}, \mathit{ref}^1_{\mathsf{reg}})$. 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 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 $\mathit{com}^0, \mathit{com}^1$ on random strings.
- For $\mathcal{O}^1_{\mathsf{Deposit}}$, to answer Q^0, Q^1, $\mathcal{C}$ behaves as in G_1 except that it produces $\{com^0_u\}^2_{u=1}$, $\{com^1_u\}^2_{u=1}$ on random strings.

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

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

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 assets 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: $Rd'_{ast}.\mathit{name} \neq Rd_{ast}.\mathit{name}$ or $Rd'_{ast}.\mathit{amt} < 0$ or $(Rd_{ast}.\mathit{amt} - Rd'_{ast}.\mathit{amt}) \cdot \mathit{pub}_t.pr_{in} \neq Rd^{out}_{ast}.amt \cdot \mathit{pub}_t.pr_{out}$;

Withdraw: $Rd_{ast}.\mathit{name} \neq ts_t.\mathit{name}$ or $Rd'_{ast}.\mathit{amt} < 0$ or $Rd_{ast}.\mathit{amt} - Rd'_{ast}.\mathit{amt} < ts_t.\mathit{amt}$.

For events 1 and 2, the input records are problematic, which are forged or stolen by $\mathcal{A}$. $\mathcal{A}$ may forge or reuse the asset records with a different identifier. In order to use the asset of others, $\mathcal{A}$ must guess the *aid* correctly. For event 3, the newly 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 on it case by case.

(1) Suppose that Pr[$\mathcal{A}$ wins and event 1 happens] is non-negligible. In this case, it leads to at least one of the following.

- 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[$\mathcal{A}$ wins and event 2 happens] is non-negligible. Here, the input records are valid records generated from previous transactions 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[$\mathcal{A}$ wins and event 3 happens] is non-negligible. In this case, $\mathcal{A}$ generates a valid transaction but gets asset records with wrong attributes. It leads to at least one of the following contradictions:

- For deposit transaction, $\mathcal{A}$ gets an asset record that is different from the deposit request. It happens only if one commitment produces two different openings that contradict 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 a new exchange-out asset record which is different from the old one or gets an exchange-in asset with more amount by breaking the fair exchange rule. It leads to at least one of the followings:
  - for the commitments of records, $\mathcal{A}$ generates a valid proof with an 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 the unforgeability of Π_{bs}.
- For the withdrawal transaction, the user should prove that he owns enough assets for the withdrawal request by committing to 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 that contradict 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 *is secure with 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, $Rd_{ast}^{out}.acp \neq pub_t.pr_{out}$; or in exchange transaction, $Rd'_{ast}.acp \neq Rd_{ast}.acp$ or $Rd_{ast}^{out}.acp \neq pub_t.pr_{out}$; 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 2,3,4, 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.

(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 him 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 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 blind signatures.
- 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.

3.4.3 Extended Compliance Supports

There are many compliance policies, such as KYC, AML, sanction list, solvency, tax filing, etc. We briefly describe how our Pisces already supports or can be directly augmented to support them.

KYC (Know Your Customer) and Sanction List. Our Pisces already supports KYC and sanction list of real-world identities. The exchange platform can get to know the customers and check customers to ensure it is not doing business with prohibited ones. Concretely, during the registration, the user must provide his real identity (passport number, birthday, nationality, bank account, etc.) to fulfill the KYC requirement. When his identity is verified and checked, which is not in sanction lists, he can join this exchange system and get the registration record.

Sanction Screening List. Platforms are required to implement sanction control for their existing customers at certain intervals after customer onboarding. In Pisces, it can be augmented simply via non-membership proof. Concretely, the platform can scan the registered user list to match with sanction lists and get the matched list. For each anonymous transaction, the user should use zero-knowledge non-membership proof to claim that he is not in the matched list. To enable this proof, additional attributes that could identify the user from the match list, should be added to the registration credential/record during the registration.

AML (Anti-Money Laundering). We mainly discuss AML regulations from the following three aspects that have been regulated and adopted by traditional finance or plain exchange platforms: (1) the wallet address for withdrawal has not been banned, for example, by OFAC (Office of Foreign Assets Control)[5]; (2) users cannot exchange or withdraw too many times in a time period[6]; (3) users cannot exchange or withdraw too many assets in one transaction, and

[5]https://ofac.treasury.gov/specially-designated-nationals-and-blocked-persons-list-sdn-human-readable-lists

[6]https://www.austrac.gov.au/business/core-guidance/amlctf-programs/transaction-monitoring

there is a limit amount for their accumulated withdrawn assets[7]. Specifically, the exchange platform verifies that the wallet address provided for each deposit and withdrawal request is not listed on OFAC's sanction list. Then, an epoch number and a transaction counter are incorporated into the registration record as additional attributes. During each exchange or withdrawal transaction, the user demonstrates that the epoch number is correct in their most recent registration record and the transaction counter is below a predetermined threshold. Subsequently, the counter is incremented by one in the newly generated record. Furthermore, an attribute for the cumulative withdrawn assets can be introduced. The user verifies that the value of the exchanged or withdrawn asset for the transaction is lower than a specified limit, and the cumulative withdrawn asset is updated accurately and remains below a predetermined threshold. At the transition to a new epoch, the user first proves that the epoch number in his latest registration record is smaller than the current one or is one of the previous ones, then its epoch number is reset to the current one, and both the counter and the accumulated withdrawn asset are reset to zero.

Solvency. Solvency requires that the platform should be able to check it has sufficient reserve [61] to avoid potential "bank run". It mainly contains two requirements: ensuring sufficient liquidity (e.g., keeping enough assets to cover the total withdraws of last month) and keeping 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 [62]; in our setting, Bitcoin). Our Pisces system with full anonymity satisfies the sufficient liquidity requirement. As the platform is still aware of the total amount of incoming/outgoing Bitcoins (and any other cryptocurrency tokens), thus the needed information could still be derived. Maintaining a minimal reserve is more challenging and intriguing for our fully anonymous scheme as exchange transactions are fully private, and users' current assets are blind to the platform. There might be some practical mitigation, e.g., actively monitoring each coin's total withdrawal amount/pattern, 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. Notably, our construction with basic anonymity can satisfy all these

[7] https://help.coinbase.com/en/exchange/funding/deposit-and-withdrawal-limits

solvency requirements as the platform knows each account's holdings (except the link between the inside and external on-chain accounts) and thus can still derive all needed information.

Enforce Tax Filing. Additionally, we can ensure compliance with tax regulations by preventing users who did not file taxes in the previous year from engaging in exchanges and withdrawals. This can be implemented by introducing a year number in the registration record, signifying the year when the user last filed taxes. During each exchange or withdrawal transaction, the user presents the year number from their most recent registration record, and this number is incremented by one when the user successfully completes his tax filing.

3.5 Performance Analysis

This section describes our instantiation, prototype implementation, and 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 [41] and Pedersen commitment [63]. 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[8] and the bilinear group provided by mcl(bn256)[9]. We run experiments on MacBook Air (1.6 GHz Dual-Core Intel Core i5, 16GB memory).

TABLE 3.1. Avg. computation cost in milliseconds.

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

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).

[8]upb.crypto: https://github.com/upbcuk.
[9]mcl: https://github.com/herumi/mcl.

Computation cost. We test the computation time cost of each party in each procedure of the anonymous exchange system. As shown in table 3.1, 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.

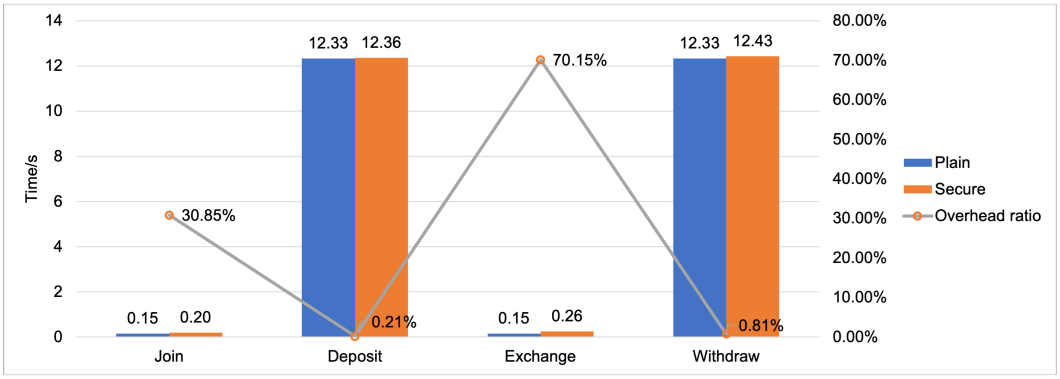

FIGURE 3.8. Comparison between plain exchange system and Pisces

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 Figure 3.8. 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)[10]. In the estimation, 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

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

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 processes 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 platform costing at least 2 RTTs, log-in procedure to get receipt address costing 1 RTT, 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 with the platform costs 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 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 a withdraw request with 1 RTT, and waiting for the on-chain confirmation with $12.21s$. The total time cost of a plain withdrawal 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 the widely used anonymous user-authentication mechanism Privacy Pass [64]. Privacy Pass published preliminary tests on consumer hardware, indicating that creating a pass in the extension takes less than 40ms[11]. 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) [60]. The results, as shown in table 3.2, demonstrate that our system is more complicated and more private, yet similarly practical to PPIS.

TABLE 3.2. Avg. computation cost of each party per procedure over 100 runs in milliseconds.

Party	Join	Earn	Exchange	Spend
PPIS [60]-user	10	8	N/A	30
PPIS [60]-provider	9	12	N/A	72

3.6 Summary

This chapter presents the first comprehensive study of a cryptocurrency exchange that balances user anonymity with compliance requirements.

For the first time, we formally define the private and compliable cryptocurrency exchange. We first give a basic version of anonymity as a warm-up, which only cares about the withdrawal operation. Hiding only part of transaction data may not provide 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

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

public withdrawal is always there). We also carefully define *soundness* properties such as *overdraft prevention*, and *compliance*.

We first give a very simple construction satisfying the basic withdrawal 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 of the number of asset types and users in the system.

We implement our Pisces system, evaluate 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. Its performance demonstrates its practicability.

CHAPTER 4

Charon: Private Cryptocurrencies Trading in Dark Pool

4.1 Introduction

Dark pools enable investors to place large orders for assets and execute trades without revealing their intentions, thereby helping to minimize significant market impact before execution. After matching, essential details are typically reported to regulators and, depending on jurisdiction, may be disclosed publicly with some delay, e.g., 15 calendar days according to CFTC Regulation 43.6 [65] in the U.S.

Currently, this confidentiality depends heavily on trusting the operator to keep trade information private. Although dark pool operators are regulated by the Securities and Exchange Commission (SEC), their credibility is often questioned, especially under temptation. Improper use of dark pool information by operators is widespread. For instance, Morgan Stanley recently paid $153 million in penalties for deceptive practices between 2018 and August 2021 [66]. In January 2022, tZERO, a U.S. dark pool operator, paid $800,000 SEC penalty for sharing order information [67].

We see that dark pool operators often prove they cannot be fully trusted in traditional finance world (or leak private data due to cyber-attacks). The situation is even worse in the less-regulated cryptocurrency space, which operates with fewer safeguards. Any leakage of trading information in the order, such as the *symbol* (name of the trading asset), trading *option* (buy or sell), *volume* (trading amount), or even the trader's identity, can be exploited by attackers to impact the market. For instance, if many traders suddenly show interest in a niche asset, it could cause significant price fluctuations. The trading option could reveal whether it's a

buyer's or seller's market. Therefore, there is an urgent need for secure dark pools that do not rely on trusted operators, where all of a trader's information remain hidden from everyone.

This issue has indeed gained increasing attention. Works from both academia and industry [68, 69, 70, 71, 72] aimed to eliminate trust in operators and make secure dark pool designs more practical. However, they all face privacy, efficiency, functionality, or deployability challenges.

Solutions with inadequate privacy. Previous dark pool designs [69, 70, 73, 71] replace the single operator with several independent servers. The user secret-shares order information across these servers, and they match orders using a multi-party computation (MPC) protocol, revealing the orders' contents only when indicated. Such a system setup does not reflect real-world application where dark pool transactions are taken place usually within one exchange.

What's worse, although the servers cannot learn the volume, they do not hide the *symbol* and *option* at the same time. Both [69, 73] focus on matching orders with hidden volumes. [69] processes orders with the number of computational operations for comparison grows linearly in the number of orders. In [73], the authors introduce a new data-independent priority queue (DIPQ) to match volumes in a dark pool. It reduces the processing complexity to polylogarithmic in the size of the order book. However, neither of them hides the symbol or option. [70] hides the symbol and volume but does not hide the option. [71] hides the option, but it leaks that fully matched orders are buy or sell orders, and it opens volumes of matched orders.

Solutions with high cost. In every dark pool design, the comparison operation is the cornerstone of matching. Given a buy order and a sell order for the same asset, also called orders on opposite sides, comparing their volumes determines the matched volume (the smaller one). If one order still has volume left, it would be compared with the next opposing order. This loop continues until one side is fully matched. Since every partial or full match requires exactly one such comparison per pairing, counting how many of these comparison operations occur directly reflects the total work the matching must do.

FHE-based solution. Asharov et al. [68] show the theoretical feasibility of fully homomorphic encryption (FHE) [74, 75] based solutions. However, they do not give concrete evaluations.

Although FHE has developed in recent years, comparison operations are not natively supported in FHE and require dedicated circuit constructions, increasing computational cost. In [76], the authors present an FHE-based inventory matching algorithm and report the performance only for volume matching: 103.2s for 128 orders, 205.3s for 256 orders, which demonstrates inefficiency, especially for large-scale order books.

MPC-related issues. Other works employ MPC protocols in their dark pool designs. Some of them [69, 70, 73, 71] were built on the honest-majority MPC protocol that emulates the dark pool operator. The small-scale servers may be controlled by the same platform that can reveal the order information and violate privacy. To get rid of the assumption of a small number of independent servers, [72] involves many users (much like a large-scale user committee) as running parties to run the MPC protocol to match orders for the dark pool. However, this large-scale MPC protocol is practically inefficient. We compare it with our results in the end of Sec. 4.5.

Polychroniadou et al. proposed Prime Match [77], an efficient server-aided MPC comparison protocol. It enables each investor to participate in the protocol and the centralized platform as an aiding server to save the user's computation. However, this protocol is tailored to match only two non-colluding traders at a time. Extending it directly to handle a set of orders, as described earlier, would require a linear number of comparison operations, making it inefficient and leaking additional information, such as the relative sizes of the matched volumes.

In summary, previous works face one or more of the following limitations.

(1) Insufficient privacy protection. Many of them only focus on hiding the volumes, but do not conceal the symbol and option at the same time, such as [71, 69, 72, 73, 77].
(2) Inefficient instantiation. Even if considering only the volume match, the number of total comparisons is linear or polylogarithmic with the number of orders, such as [69, 73, 77, 72].

We show them in Table 5.1. Thus, achieving a practically efficient dark pool with comprehensive protection of trading information is still an open challenge.

TABLE 4.1. Comparisons of relevant works.

Schemes	Symbol*	Option*	Volume *	Complexity †
DarkSide. [69]	×	×	✓	$O(N)$
Bucket. [71]	×	✗‡	✗$^{\diamond}$	$O(\log N)$
All-One. [72]	×	×	✓	$O(N)$
Prime Match [77]	×	×	✓	$O(N)$
DIPQ [73]	×	×	✓	$O(\log^2 N)$
FHE-based [68]	×	✓	✓	$O(N)$
Our Π_{PBVM} 6.	✓	✓	✓	$O(\log N)$
Our Π_{FPVM} 8.	✓	✓	✓	$O(N)^{\star\star}$

* They mean whether hiding the symbol, option, and volume. We do not include the hiding identity feature as all schemes can be upgraded to support anonymity using the techniques in private exchange, e.g., [32].

† We use the number of comparisons needed for volume match to measure the algorithm's overall computational complexity, which grows in proportion to the order book size N.

‡ It opens the options of matched orders, which also leaks the options of unmatched orders.

$^{\diamond}$ It opens the volumes of matched orders when the match finishes.

** It can be run with $O(N)$ concurrent threads, reducing the complexity to $O(1)$, while all others have to run sequentially and cannot achieve it.

4.1.1 Our Contributions

In this paper, we fill those gaps and introduce Charon, a secure dark pool system, in which traders communicate only with one semi-honest platform, jointly performing the matching computation while keeping all order details – symbol, option, and volume – hidden from the platform. The volume matching process is highly efficient, with a logarithmic number of comparisons in terms of the order book size, and ensures no compatible orders are unmatched. We formally define ideal functionalities for each process, prove the system's security, and demonstrate its practicality through implementation and performance evaluation. Furthermore, the system is versatile, supporting both cryptocurrency and traditional assets like stocks.

Hide all order information. Matching orders in a dark pool can be split into two phases: First, identify each order book by checking whether two orders are eligible to be matched (i.e., they share the same symbol but have opposite trade options). Second, determine the volume that can be matched for the orders within each order book.

In our work, the symbol, option, and volume are hidden all the time, until the orders are executed and allowed to be disclosed publicly by the regulatory authority [1]. At first, multiple order books are produced with respect to different assets by checking the eligibility of every pair of orders. We design a private equality test protocol that ensures no leakage except the eligibility results. Regarding the volume match phase, we present two private constructions. In the first private batch volume match protocol Π_{PBVM}, the platform learns nothing about the orders beyond each order's match result. Then we reduce the information leakage to a minimum. Our second fully private volume match scheme Π_{FPVM} further hides the match results from the platform. In both designs, each trader learns only their own match result and matched volume.

Reduce time cost. All previous methods of volume match require *sequentially* conducting linear or logarithmic comparisons. Our two constructions of volume match are efficient, reducing runtime without sacrificing privacy. The first one, Π_{PBVM}, requires sequential comparisons with $O(\log N)$ in the size of the order book. This matches the time and computational complexity of [71] but achieves stronger privacy protection. In the second scheme Π_{FPVM} that hides more information, each trader only conducts constant comparisons. Although the total number of comparisons is linear with the number of orders, they can be performed in parallel.

Evaluate performance. We tested all procedures and the total runtime of dark pool with 50 orders is less than 300s in single thread computation. With the same 16-thread parallel computation as in [72], the estimated total runtime for private volume match of 200 orders is under 12s, which is significantly less than that reported in [72].

4.1.2 Overview

First, we provide an overview of the workflow of a dark pool. Then, we introduce the techniques of each protocol.

[1] The regulator can reveal the order information and trader identity. But it only opens the executed ones and does not join the matching procedure according to the regulation requirements [78, 65]

System setting - single semi-honest server. In current real world practice, the platform is fully trusted. It knows the details of all orders without any cost (except the legal risk). In this work, we relax the trust in the platform. It wishes to learn as much as possible about the order information, but follows the protocol and does not collude with traders in the dark pool. It functions as an aiding server during its workflow, similar to the server-aided MPC model with a semi-honest server in [79, 77, 80].

One might wonder whether the trust assumption can be further relaxed to allow for a malicious platform. Unfortunately, this is infeasible in the current star network setup, where users communicate only with the platform, which governs all inter-user communication. As shown in [81], secure computation without authentication cannot guarantee malicious security, requiring alternative setups and advanced techniques. Our anonymity setting makes it hard. For instance, a malicious platform could pretend to be a user and fabricate an order without committing any real assets. It then compromises other users' privacy by using this fake order to match with other honest users, extract information about their orders, and still outputs "fail", even when the match succeeds. Due to the anonymity feature of the dark pool, users cannot distinguish between genuine interactions with other users and malicious interactions with the platform. Thus, we assume a semi-honest platform that adheres to the protocol, leaving the malicious platform case for future exploration, which would require extra setups.

Dark pool workflow. Given many trading parties P_i and a platform P^*, each P_i has its private order inputs. The core functionality of a dark pool is a protocol[2] where, through interactions with the platform, each trader receives their matched volume as the output. The workflow for achieving dark pool trading consists of three phases, which are depicted with a simple example in Fig. 4.1.

• *Phase 1. order placement.* The platform periodically opens the pool for a set duration, allowing traders to express their interest in trading. During this phase, traders submit their orders in a concealed form and only prove their ability to fulfill the committed order.

[2]We also call it dark pool for brevity, and the complete procedure of dark pool is described in Sec.4.4 for completeness.

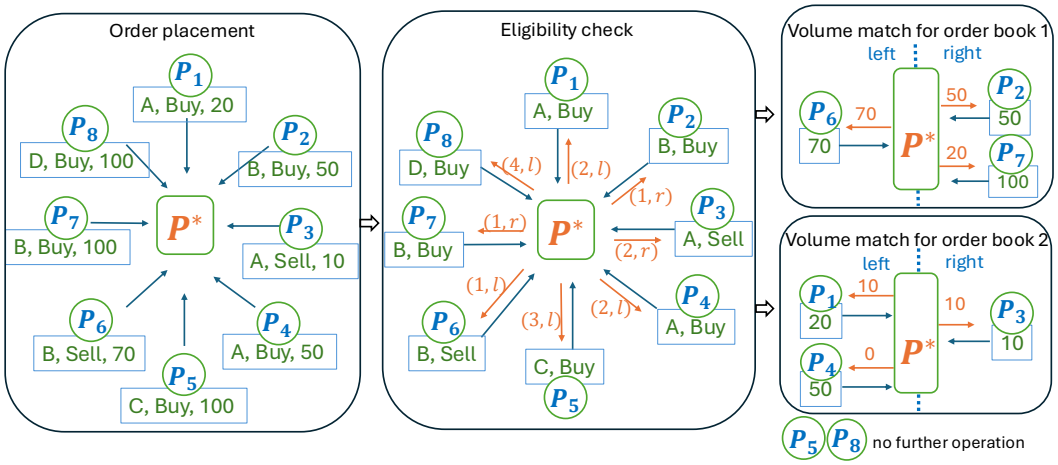

FIGURE 4.1. Simple example of dark pool workflow.

• *Phase 2. eligibility check.* In this phase, the trader interacts with the platform to determine whether there are other orders with the same symbol but different options, making them eligible to be matched. As a result, all eligible orders are partitioned into several order books. Each book corresponds to a unique symbol and contains two sets of orders with the same symbol but different options. At the end of eligibility check phase, P^* obtains two order books, while the orders from P_5 and P_8 are left alone without further operation as there are no opposite orders.

• *Phase 3. volume match.* While orders in the order book are eligible for matching, the actual match and execution depend on their volumes and submission times. In this phase, traders within the same order book interact with each other through the platform to determine the match results and the matched volume. We follow the first-come, first-served (FCFS) principle for matching. As shown in Fig.4.1, P_2, P_3, P_6 are fully matched, P_4 is unmatched, P_1, P_7 are partially matched.

Technical overview. When placing orders, all information should be hidden, which can be achieved via commitments and zero-knowledge proofs (ZKP). Similar technologies have been used in the private exchanges with the platform, like Pisces [32]. The main challenges arise in the following procedures of the eligibility check and volume match. We introduce them as follows and explain our solutions.

• *Eligibility check.* The eligibility check requires at least two orders with the same symbol and different options. During this phase, the inputs are always hidden from the platform. If the result is ineligible, it does not reveal which conditions are unsatisfied.

The eligibility check can be done via equality tests. However, conventional equality test protocols only consider the two-party computation case and can only handle at most one malicious party [82, 83]. While in the dark pool setting, the semi-honest platform can work as an aided server, and users can be malicious and jointly cheat the platform. For example, Alice has 1000 XRP to sell, and Bob wants to buy 1000 BTC. If they collude together and get matched, Bob could buy and withdraw BTC from the air. The existing server-aided equality test design, such as Prime Match [77], incurs quadratic online communication and computation costs in the input bit length, while we achieve the same security with linear communication and computation, which is more efficient.

Recall that eligibility is determined by evaluating the logical AND of two conditions: (1) an equality test on private symbols, and (2) an inequality test on private options. To support the latter, one party applies a linear transformation to its option value. To prevent information leakage during this eligibility check, we extend the scalar equality test to the vector case and propose an extended private equality test protocol. This protocol securely aggregates multiple equality checks into a single test that outputs 1 if and only if the two private vectors are equal. Crucially, it reveals nothing about individual elements or which positions, if any, are unequal.

• *Efficient private volume match.* The eligibility check produces multiple order books, each with two order sets: left (l) and right (r). A naive matching method compares orders sequentially: the smaller volume is subtracted from the larger, and the remainder is matched against the next order in the opposite set. This repeats until one set is exhausted. This approach is inefficient due to linear time complexity and also leaks information—volume dependencies across steps allow the larger-volume party to infer the smaller volume.

Private binary search. Given volumes $(\mathsf{v}_1, \ldots, \mathsf{v}_n)$ sorted by submission time, we have a monotonically increasing cumulative volume sequence $(\mathsf{cv}_1, \ldots, \mathsf{cv}_n)$ where $\mathsf{cv}_k = \sum_{i=1}^{k} \mathsf{v}_i$. In an order book, the total cumulative volumes of the left and right sets, cv_m^l and cv_n^r, are

compared. If $\mathsf{cv}_m^l < \mathsf{cv}_n^r$, all left-set orders can be fully matched. Since $(\mathsf{cv}_1^r, \ldots, \mathsf{cv}_n^r)$ is increasing, a binary search finds the smallest index $t \in [n]$ such that $\mathsf{cv}_m^l \leq \mathsf{cv}_t^r$, indicating that orders v_1^r to v_{t-1}^r are fully matched, while v_t^r is partially matched and all subsequent orders are unmatched.

This binary search idea has been used in [71], but this scheme does not hide the symbol and reveals the options and volumes of all matched orders immediately when the matching process finishes. It does not satisfy the regulatory rule that the order information can only be opened by the regulatory authority after a couple of weeks [78, 65]. If they want to hide the matched volumes further, the number of comparisons would grow linearly with the order book size. Compared with them, we retain the same computation complexity and only reveal which orders are matched without disclosing any other details.

In our design, we employ a three-party secure comparison protocol. Each participating party has one of the shares of a secret value v as its input, and this protocol determines whether this value is negative or not without revealing additional information. It serves as a foundational building block in the volume match process. Due to the linear homomorphism of secret shares, the difference of corresponding shares from two secret values can determine which value is larger via this secure comparison protocol.

In each order book, the platform knows all traders' (one-time) public encryption keys and sends the public keys of traders in one set to each trader in the other set. Then, each trader produces three shares of their volume and encrypts one of them under the public keys in the other set. It sends these ciphers and the other two shares to the platform with proof of correctness. Thus, the platform has the shares of each volume in plain and ciphertext forms. Since the encryption and secret sharing are additively homomorphic, it can compute the ciphertexts of shares for cumulative volume and send them to the traders who have decryption keys.

The first step is comparing the two total cumulative values, cv_m^l and cv_n^r. The platform selects two traders from different sets, sends them the ciphertexts of $\mathsf{cv}_m^l - \mathsf{cv}_n^r$'s shares, and it also keeps one plain share. The three parties run the secure comparison protocol to determine

which is smaller and set it as the target value, e.g., cv_m^l. Then the platform continuously runs the secure comparison protocol with the two traders for different cv_j^r in the binary search method to find the smallest one that is no less than cv_m^l, which corresponds to the last matched trader.

After that, the platform informs the fully matched traders and the unmatched traders of their match results. So, they know their matched volumes are their orders' volumes or zero immediately. It also informs the last matched trader of its matched volume in the ciphertext form. All the while, the platform never knows any volume.

• *Fully private volume match.* Although the previous matching method is efficient, the platform also learns the match result of each trader, which leaves unnecessary leakage. In an idea dark pool, even the match results are hidden from the platform, and all traders can still receive their matched volumes.

At first glance, it may feel counterintuitive. If the match results are hidden, the platform cannot identify which traders are matched, especially the last matched trader. So it has to treat them equally and cannot inform them of their (partially) matched volumes.

We propose a *fully* private match approach that addresses the dilemma between hiding the match results and informing the matched volumes. We introduce it as follows.

Decide the match result via two comparisons. Different from the previous binary search method, where each comparison depends on the result of the previous one, we adopt a new approach to eliminate this dependency. The start point is that each party (e.g., P_i^l) runs a secure comparison to determine the match result by comparing cv_{i-1}^l (the cumulative volume before its order) with cv_n^r (the total cumulative volume of the other set). If $\mathsf{cv}_{i-1}^l < \mathsf{cv}_n^r$, it can be matched. Next, they perform another secure comparison for cv_i^l and cv_n^r. The last matched party will observe two different comparison results, i.e., $\mathsf{cv}_{i-1}^l < \mathsf{cv}_n^r$, $\mathsf{cv}_i^l \geq \mathsf{cv}_n^r$. Others receiving the same results are fully matched (both less) or unmatched (both no less).

Mask comparison result with a random bit. Our method still uses the secure comparison protocol as the building block, but hides the comparison result from the platform. The trader

disguises the comparison input and result using a random bit, e. The platform only sees a masked result, which combines the real result with that random bit. Thus, only traders can figure out the true result later.

In practice, instead of running the comparison protocol with the original shares, the trader works with the platform to slightly modify the process, adding a random factor on the inputs that hides the outcome depending on e. If $e = 1$, the result would be opposite. Otherwise, it remains the same.

Inform the last matched trader obliviously. For the fully matched and unmatched traders, the two comparison results indicate their matched volumes. For the last matched trader P_t^l, computing its matched volume $\mathsf{cv}_n^r - \mathsf{cv}_{t-1}^l$ must rely on interacting with the platform. However, it cannot reveal its index to the platform. To solve this, the platform informs the last matched trader based on its unique feature: only its two comparison results are different. The platform encrypts $\mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l$ for every P_i^l and ensures only the last matched trader can get the decryption key and obtain its matched volume. We refer to Sec. 4.3.3 for a detailed explanation.

4.1.3 More related works

Secure comparison. Polychroniadou et al. [77] proposed Prime Match, an efficient server-aided two-party comparison protocol. It enables each investor participating in the protocol to match their orders with an aiding server. They only allow one party to be corrupted and do not consider the colluded user cheating the server.

Bicoptor [84] introduces a family of novel secure three-party computation (3PC) protocols, including equality test and comparison, with only two communication rounds. They require that all participants be semi-honest and that there be no collusion between any two of them.

Our designs involve trading parties in the dark pool to compute the match results. Although the platform is semi-honest, the traders may collude with others. Thus, we introduce a simple and efficient equality test protocol for eligibility checks and employ Rabbit [85] as a building

block in the volume match. It is a secure MPC supporting comparison computation in the dishonest-majority setting.

4.2 Dark Pool Framework

A centralized dark pool consists of a platform P^* and M registered users $P_1, \ldots, P_M$, where each user can place orders to trade cryptocurrencies or other assets. The market includes S asset types. Each order is represented as a tuple: $\mathsf{ord} = (\mathsf{sym}, \mathsf{opt}, \mathsf{v})$, where $\mathsf{sym} \in [S]$ denotes the asset symbol, $\mathsf{opt} \in \{1, 2\}$ specifies the operation (1 for buying, 2 for selling), and $\mathsf{v} \in [\max]$ represents the volume [3]. Prices are excluded since orders follow public prices such as the National Best Bid and Offer (NBBO) [86]. This rule has been adopted by several real-world exchanges such as Liquidnet and NYSE Euronext.

User Operations. There are six operations in the dark pool:

- **Register:** The user registers with a real identity.
- **Deposit:** Registered users deposit assets to the platform, which is responsible for their custody.
- **Place order:** The registered user places their order in the dark pool. They are also called traders.
- **Eligibility check:** All orders with the same symbol but different options are identified and included in the same order book, which consists of two sets of buy orders and sell orders.
- **Volume match:** This phase determines which orders within the order book can be matched based on their volumes and submission time. The platform informs traders of their match results and the matched volumes.
- **Withdraw:** The user withdraws assets from the platform.

REMARK 3 (**Compare with server-aided model**). *The platform functions as a semi-honest aiding server to assist users in matching their orders. This design aligns with the well-established server-aided model with a semi-honest server [79, 77, 80, 87]. The server has*

[3]The dark pool usually enforces a minimum trade volume. We omit it here for brevity and assume all volumes exceed this minimum threshold.

no private input but facilitates secure computation. It is incentivized for profits and even reputation consideration to follow the protocol while remaining curious about users' inputs.

REMARK 4 (**User participation**). *The dark pool primarily serves institutional investors with substantial funds. They want to complete large trades without triggering a price move [78]. Motivated by higher profit, they may be willing to participate in computation for matching orders privately rather than fully relying on external servers.*

REMARK 5 (**Disclose according to regulation**). *Due to the regulation requirements [78, 65], the executed orders in the dark pool would be revealed to the public in two weeks, including the users' identities and order information. This can be achieved by letting the user encrypt these messages using the regulator's public key. The regulator decrypts and discloses them publicly. It is a necessary leakage. We do not consider it here and later constructions since the regulator does not join the matching procedure.*

Features. We assume the platform is semi-honest and does not collude with any users. We hope to achieve the following features:

- **User anonymity:** The platform cannot link the order to any honest trader's identity.
- **Exchange fairness:** The trading is fair w.r.t. public prices.
- **Order privacy:** On submitted orders, the platform and traders do not learn any more information beyond the leakage allowed during order matching in the dark pool.

User anonymity and exchange fairness can be ensured using private exchange techniques, such as those in [32], where users interact with the platform via anonymous credentials. Thus, this paper focuses on preserving privacy during order matching. We explicitly specify the information leaked to different parties throughout the process and formally define the leakage using the following ideal functionalities.

Ideal functionalities and information leakage. In general, all parties in a dark pool are only allowed to get the information leakage as follows. Each user knows how many orders are in the same order book and their own matched volume. The platform knows how many order books are in total and the orders' indices in each order book. After the volume match phase,

it can know which orders in an order book are matched, but does not know their symbols, options, or volumes. To show the leakage formally, we define the ideal functionalities of the eligibility check $\mathcal{F}_{\mathsf{EC}}$, private volume match $\mathcal{F}_{\mathsf{VM}}$, and fully private volume match $\mathcal{F}_{\mathsf{FVM}}$.

Eligibility check. Given two orders, they are eligible for matching only if their symbols are the same, but the options are different. We use the function $\mathsf{Elig}(\mathsf{sym}_1, \mathsf{opt}_1; \mathsf{sym}_2, \mathsf{opt}_2)$ to denote it, where $\mathsf{sym}_{i\in\{1,2\}} \in [S]$, $\mathsf{opt}_{i\in\{1,2\}} \in \{1, 2\}$. It achieves that:

$$1 \leftarrow \mathsf{Elig}(\mathsf{sym}_1, \mathsf{opt}_1; \mathsf{sym}_2, \mathsf{opt}_2) \iff \mathsf{sym}_1 = \mathsf{sym}_2 \wedge \mathsf{opt}_1 \neq \mathsf{opt}_2$$

We define the following functionality $\mathcal{F}_{\mathsf{EC}}$ to check eligibility for all orders in a dark pool in Fig. 4.2. Given all orders' symbols and options as inputs, it outputs an empty set or some order books. Each party P_i is a dark pool trader who places an order. The symbol and option of its order are its secret inputs. The platform P^* interacts with the traders. As a result, $\mathcal{F}_{\mathsf{EC}}$ sends to P^* all the orders that are eligible for matching, and they are categorized into different books: $G_1^l, G_1^r, ..., G_N^l, G_N^r \subset [M]$, where G_k^l, G_k^r denote the two sets of indices of orders with the same symbol but do not leak their symbols or options.

Functionality $\mathcal{F}_{\mathsf{EC}}$ (Eligibility Check)
There are parties: $P_1, ..., P_M, P^*$, where $M \geq 2$.

- **Input**: Each party P_i has input $\mathsf{sym}_i, \mathsf{opt}_i$, $i \in [M]$
- **Output**: Send the number M to each P_i;
  Send P^* the empty set $\emptyset$ or non-empty index sets $G_1^l, G_1^r, ..., G_N^l, G_N^r \subseteq [M]$, such that for $\forall k \in [N]$:
  1). $\forall i \in G_k^l, j \in G_k^r$, $\mathsf{Elig}(\mathsf{sym}_i, \mathsf{opt}_i; \mathsf{sym}_j, \mathsf{opt}_j) = 1$;
  2). $\forall i \in G_k^l, j \in [M] \setminus G_k^r$, $\mathsf{Elig}(\mathsf{sym}_i, \mathsf{opt}_i; \mathsf{sym}_j, \mathsf{opt}_j) = 0$;
  3). $\forall i, j \in [M] \setminus \bigcup_{k=1}^{N} G_k^l \cup G_k^r$, $\mathsf{Elig}(\mathsf{sym}_i, \mathsf{opt}_i; \mathsf{sym}_j, \mathsf{opt}_j) = 0$.

FIGURE 4.2. The functionality of Eligibility Check

Leakage information: Each party only knows the total number of orders in the dark pool. The platform knows the checking result of each pair of orders. But it cannot get any more information about its symbol, option, or volume.

Private volume match. Given any order book G^l, G^r, where $|G^l| = m, |G^r| = n$, they correspond to two sets of parties P_1^l,..., P_m^l, $P_1^r, ..., P_n^r$. Each party has a positive volume value v as input, except for P^*, which has no input.

The system matches the orders progressively until one side runs out of orders, i.e., no more orders are available to match, and the remaining orders (on the opposite side) are unmatched. The functionality $\mathcal{F}_{\mathsf{VM}}$ decides which orders can be matched based on the volume, as shown in Fig. 4.3. As the output, the functionality sends m, n, l (or r), t to P^*. It sends the match result to each party P_i. We also call P_i matched if its order is matched.

Functionality $\mathcal{F}_{\mathsf{VM}}$ (Private Volume Match)
There are parties $P_1^l, ..., P_m^l, P_1^r, ..., P_n^r, P^*$.

- **Input**: Each party P_i^l, P_j^r has a positive volume value $\mathsf{v}_i^l, \mathsf{v}_j^r$ as its input. P^* has no input.
- **Output**: If $\mathsf{cv}_m^l = \mathsf{Min}(\mathsf{cv}_m^l, \mathsf{cv}_n^r)$ and $\exists t \in [n]$ such that $\mathsf{cv}_{t-1}^r < \mathsf{cv}_m^l \leq \mathsf{cv}_t^r$ ($\mathsf{v}_0^r = 0$ if $t = 1$), then
  Send l, t, m, n to P^*.
  Send $(\mathsf{Matched}, \mathsf{v}_i^l, m, n)$ to each $P_i^l, i \in [m]$.
  Send $(\mathsf{Matched}, \mathsf{v}_i^r, m, n)$ to each $P_i^r, i \in [t-1]$.
  Send $(\mathsf{LastMatch}, \mathsf{cv}_m^l - \mathsf{cv}_{t-1}^r, m, n)$ to P_t^r.
  Send $(\mathsf{Unmatched}, m, n)$ to $P_{t+1}^r, ..., P_n^r$ ($t < n$).
  If $\mathsf{cv}_n^r = \mathsf{Min}(\mathsf{cv}_m^l, \mathsf{cv}_n^r)$, it works similarly.

FIGURE 4.3. The functionality of Private Volume Match $\mathcal{F}_{\mathsf{VM}}$.

Leakage information: Each party learns the numbers m, n. Based on l, t, P^* learns which set has the smaller sum of volumes, where all of them can be matched, and which are the first t parties that can be matched in the other set. Each trader party learns whether they can be matched and the matched volume.

Fully private volume match. The volume match functionality can be improved further by hiding the match results from P^*. We call it the fully private volume match $\mathcal{F}_{\mathsf{FVM}}$, as displayed in Fig. 4.4, where P^* does not get l, t.

Functionality $\mathcal{F}_{\mathsf{FVM}}$ (Fully Private Volume Match)
There are parties $P_1^l, ..., P_m^l, P_1^r, ..., P_n^r, P^*$.

- **Input**: Each party P_i^l, P_j^r has a positive volume value $\mathsf{v}_i^l, \mathsf{v}_j^r$ as its input. P^* has no input.
- **Output**: If $\mathsf{cv}_m^l = \mathsf{Min}(\mathsf{cv}_m^l, \mathsf{cv}_n^r)$ and $\exists t \in [n]$ such that $\mathsf{cv}_{t-1}^r < \mathsf{cv}_m^l \leq \mathsf{cv}_t^r$ ($\mathsf{v}_0^r = 0$ if $t = 1$), then
  Send m, n to P^*.
  Send $(\mathsf{Matched}, \mathsf{v}_i^l, m, n)$ to each $P_i^l, i \in [m]$.
  Send $(\mathsf{Matched}, \mathsf{v}_i^r, m, n)$ to each $P_i^r, i \in [t-1]$.
  Send $(\mathsf{LastMatch}, \mathsf{cv}_m^l - \mathsf{cv}_{t-1}^r, m, n)$ to P_t^r.
  Send $(\mathsf{Unmatched}, m, n)$ to $P_{t+1}^r, ..., P_n^r$ $(t < n)$.
  If $\mathsf{cv}_n^r = \mathsf{Min}(\mathsf{cv}_m^l, \mathsf{cv}_n^r)$, it works similarly.

FIGURE 4.4. The functionality of Fully Private Volume Match $\mathcal{F}_{\mathsf{FVM}}$.

Leakage information: Each party learns the numbers m, n. For each party P_i, they learn whether they can be matched and the matched volume if they can. If they are matched, they also learn whether they are the last match and their matched volumes.

Summarize the above subprotocols. There are N order books $G_1^l, G_1^r, ..., G_N^l, G_N^r$. Without loss of generality, we assume that all orders in the left sets G_k^l are fully matched, and we denote the last matched order's index as t_k in each right set G_k^r. So for each $i \in G_k^r$, P_i is fully matched if $i < t_k$, P_i is unmatched if $i > t_k$.

Leakage information: Each party learns whether it is eligible, whether it is matched, and the matched volume.

In the fully private volume match case, the platform only learns the order books' information. But in the private volume match case, it also learns the matched parties' indices. We summarize them in the functionality $\mathcal{F}_{\mathsf{DP}}$ (Fig. 4.5).

Overview construction of dark pool. Fig. 4.6 gives a pictorial overview of how we construct the dark pool from the sub-functionalities $\mathcal{F}_{\mathsf{EC}}$ and $\mathcal{F}_{(\mathsf{F})\mathsf{VM}}$. All parties first run the eligibility check $\mathcal{F}_{\mathsf{EC}}$ and get the check results. Based on it, they run the private volume match functionality $\mathcal{F}_{(\mathsf{F})\mathsf{VM}}$ and get the final match results.

Functionality $\mathcal{F}_{\mathsf{DP}}$ (Dark Pool)

There are parties: $P_1, ..., P_M, P^*$, where $M \geq 2$.

- **Input**: Each party P_i has input $(\mathsf{sym}_i, \mathsf{opt}_i, \mathsf{v}_i)$, $i \in [M]$. P^* has no input. They run the functionalities $\mathcal{F}_{\mathsf{EC}}$ and $\mathcal{F}_{\mathsf{(F)VM}}$ as shown in Fig. 4.6.
- **Output**: Send the number M to each P_i;

  Send P^* the empty set $\emptyset$ or non-empty index sets $G_1^l, G_1^r, ..., G_N^l, G_N^r \subseteq [M]$, the match result l, t_k

  Send Ineligible to P_i, where $i \in [M] \setminus \bigcup_{k=1}^{N} G_k^l \cup G_k^r$

  For each $k \in [N]$, $m_k = |G_k^l|, n_k = |G_k^r|$

  Send $(\mathsf{Matched}, \mathsf{v}_i, m_k, n_k)$ to $P_i, i \in G_k^l$.

  Send $(\mathsf{Matched}, \mathsf{v}_j, m_k, n_k)$ to $P_j, j \in G_k^r$ and $j < t_k$.

  Send $(\mathsf{LastMatch}, \sum_{i \in G_k^l} \mathsf{v}_i - \sum_{j \in G_k^r \wedge j < t_k} \mathsf{v}_j, m_k, n_k)$ to P_{t_k}. // P_{t_k} is the last matched party

  Send $(\mathsf{Unmatched}, m_k, n_k)$ to $P_j, j \in G_k^r$ and $j > t_k$.

FIGURE 4.5. The functionality of the whole process of the dark pool $\mathcal{F}_{\mathsf{DP}}$.

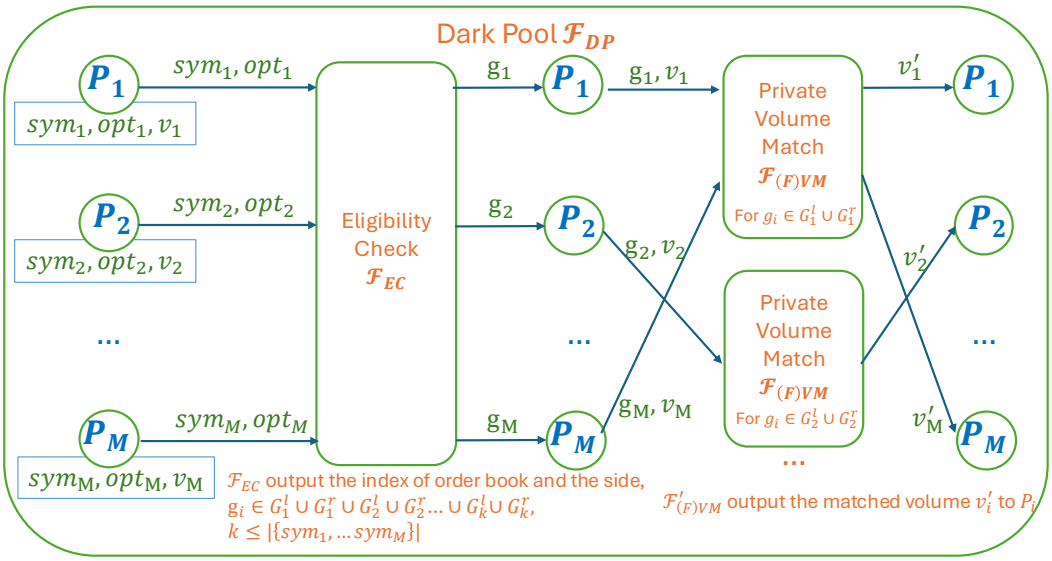

FIGURE 4.6. Overview construction of dark pool.

4.3 Constructions

The core functionality of dark pool involves each trader who has placed a private order interacting with the platform to receive their match result. In this section, we design and

prove the fulfillment of the functionality via constructing the eligibility check protocol in Sec. 4.3.1, and two private volume match protocols with different levels of privacy protection in Sec. 4.3.2 and Sec. 4.3.3.

4.3.1 Eligibility Check

The high-level idea of the eligibility check is that each user interacts with every other user, with the platform acting as an intermediary, to determine whether they are eligible to match. This check can be achieved via two (in)equality tests. First, the same symbol condition tests whether the symbols of two orders, sym_1 and sym_2, are equal. Second, the different options condition tests whether the options are different. It can be converted into testing whether opt_1 and $3 - \mathsf{opt}_2$ are equal as only two options $\mathsf{opt}_1, \mathsf{opt}_2 \in \{1, 2\}$.

We propose an efficient private 2-party equality test protocol Π_{PET} to test the equality of two private values. It supports equality tests on linear transformations of the inputs, leveraging its additively homomorphic structure. Furthermore, Π_{PET} can be naturally extended to operate on vectors, testing the equality of linear transformations element-wise. The extended protocol outputs 1 if and only if all corresponding elements in the transformed vectors are equal; otherwise, it outputs 0 without leaking which elements are different. Every two users can directly call this extension version to check their eligibility.

Before that, we first give a server-aided two-party coin tossing protocol Π_{CT} as follows, in Algo. 1, which is extended from the classical Diffie-Hellman key exchange. The aiding server is semi-honest.

Algorithm 1 Coin tossing Π_{CT}

1: Setup a group $\mathbb{G}$ over $\mathbb{Z}_p^*$ with generator g, a hash function *Hash* mapping a group element to ℓ-bit string.
2: P_0 chooses $a \leftarrow\!\!\$\ \mathbb{Z}_p^*$, P_1 chooses $b \leftarrow\!\!\$\ \mathbb{Z}_p^*$, P^* is the semi-honest server
3: P_0 sends $A = g^a$ to P^*, P_1 sends $B = g^b$ to P^*
4: P^* forwards B to P_0 and A to P_1 if neither of them equals to $1_{\mathbb{G}}$. Otherwise, it aborts.
5: P_0 computes $s = \mathit{Hash}(B^a) = \mathit{Hash}(g^{ab})$, P_1 computes $s = \mathit{Hash}(A^b) = \mathit{Hash}(g^{ab})$

LEMMA 1. *Assume the Decisional Diffie-Hellman (DDH) problem is hard and the random oracle model. The protocol in Algo.1 securely implements the coin tossing functionality $\mathcal{F}_{\mathsf{CT}}$ in Sec. 2.3.*

PROOF. According to the definition of $\mathcal{F}_{\mathsf{CT}}$, P^* learns nothing and P_0, P_1 receive the same random string. We consider (1) privacy against the semi-honest platform; (2) no party can bias or predict the output before the protocol completes.

In the real world, P_0, P_1, P^* run the above protocol, where some party can be corrupted by $\mathcal{A}$. In the ideal world, P_0, P_1 receive the same random string from $\mathcal{F}_{\mathsf{CT}}$, the simulator $\mathcal{S}$ interacts with the corrupted party and $\mathcal{F}_{\mathsf{CT}}$, and it can program and respond to hash queries.

Case 1: P^* is corrupted by a semi-honest $\mathcal{A}$. In the real world, P_0, P_1 run the protocol with $\mathcal{A}$, who sees two group elements (A, B). P_0, P_1 query Hash values for the string s, but $\mathcal{A}$ does not see it. $\mathcal{S}$ interacts with $\mathcal{A}$ and $\mathcal{F}_{\mathsf{CT}}$ as follows. $\mathcal{S}$ receives a random string s from $\mathcal{F}_{\mathsf{CT}}$. It picks two random group elements $(A^*, B^*) \leftarrow\!\$\ \mathbb{G}$ and sends them to the environment $\mathcal{E}$. Whenever the parties query $\mathit{Hash}(A^b)$ or $\mathit{Hash}(B^a)$, $\mathcal{S}$ programs the random oracle so that $\mathit{Hash}(K) = s$, where K is the "simulated" g^{ab}. $\mathcal{A}$ sees only (A^*, B^*) and never knows s. Under the DDH assumption, the pair (g^a, g^b) is computationally indistinguishable from two independent uniform elements (A^*, B^*). Programming $\mathit{Hash}(K)$ to equal s perfectly matches the real-world distribution of $\mathit{Hash}(g^{ab})$. Thus, no environment $\mathcal{E}$ can tell a real server's view or $\mathcal{S}$'s simulated view. In the simulation world, $\mathcal{A}$ learns nothing about s, and it establishes the privacy against the server. It can neither predict nor bias s.

Case 2: W.l.o.g., we assume P_0 is corrupted by the malicious $\mathcal{A}$. In the real world, the corrupted P_0 picks and runs the protocol with P_1, P^*. P_0, P_1 get s. P_0 aims to predict or force s. But it must send A to the semi-honest P^* before receiving B, who also checks if $A = 1_{\mathbb{G}}$, it aborts.

The simulator $\mathcal{S}$ works as follows. It sees P_0's A and records it. If $A = 1_{\mathbb{G}}$, it aborts. Ask $\mathcal{F}_{\mathsf{CT}}$ for a uniform string s. Choose $B^* \leftarrow\!\$\ \mathbb{G}$ and send it to P_0. Whenever P_0 queries the Hash value for an element X, check if it has been queried before. If so, return the corresponding

string s. Otherwise, answer with the fresh random string s and record it with X. In this world, P_0 cannot predict or enforce s since it is programmed by $\mathcal{S}$. $\mathcal{A}$'s view consists of (B, s) in both worlds. In the real world, B is uniform in $\mathbb{G}$. In the ideal world, B^* is also a random element. Under DDH assumption, no one can distinguish them. In both worlds, P_0 eventually queries *Hash* on a group element. Since $\mathcal{S}$ can program it to a random string, $\mathcal{A}$'s view is consistent in both worlds. Hence, no environment can distinguish its views between these worlds.

□

Private 2-party equality test Π_{PET}. We now provide a private equality test protocol for two users with a semi-honest platform, shown in Algo. 2. Each party has a private input v_i and has sent their commitment $com(v_i)$ to P^* in the first step. This protocol ensures security against malicious users attempting to manipulate the result by requiring each party to prove the correctness of their operations. To avoid extra range proof, we let P^* generate some random challenges d, h, which are also used in c_i generation. It can be made non-interactive via Fiat-Shamir transform [88] as shown in step 4. Each party proves c_i is computed correctly via an NIZK proof π_i.eq, which is generated using the zero-knowledge proof system on the NP relation $\mathcal{R}_{\mathsf{eq}}$. It proves that the knowledge of secret values in the commitments com, com_1, com_2 and c is computed from these secret values, especially, the secret value v in c equals the committed value in com. It can be achieved by classic Sigma protocol, such as Schnorr's protocol [89].

$$\begin{aligned}\mathcal{R}_{\mathsf{eq}} = \{(com, c, com_1, com_2; v, u, s, r, t, z) : com = \mathsf{Com}(v; u)\\ \wedge com_1 = \mathsf{Com}(s; r) \wedge com_2 = \mathsf{Com}(t; z)\\ \wedge (d, h) \leftarrow Hash(com_1 \| com_2) \wedge c = v(s + d) + t + h\}\end{aligned}$$

Adapt Π_{PET} to linear transformation of input. Given two public linear functions f_1, f_2, the protocol modifies a bit in step 4 by replacing v_i with $f_i(v_i)$ and proves and verifies accordingly based on the functions. In other words, the adapted protocol tests the equality of $f_1(v_1)$ and

Algorithm 2 A 2-party equality test of two private values with semi-honest platform Π_{PET}.

1: Setup a group over $\mathbb{Z}_p^*$. There are authenticated private channels between platform P^* and each party P_1, P_2. Each P_i has a private value v_i as input.
2: P_i sends $\mathsf{Com}(v_i)$ and NIZK proves knowledge of v_i to P^*, P^* verifies it.
3: P_1, P_2 share private randomness s, t, r, z via Π_{CT}
// generate *private* randomness via coin tossing
4: P_i computes $com_1 = \mathsf{Com}(s; r), com_2 = \mathsf{Com}(t; z)$
$(d, h) \leftarrow \mathit{Hash}(\mathsf{Com}(s; r) \| \mathsf{Com}(t; z))$.
// generate *public* common randomness
$c_i = v_i(s + d) + t + h$.
generates NIZK proof π_i.eq for the NP relation $\mathcal{R}_{\mathsf{eq}}$
// prove that c_i is computed from v_i which is the committed value in $\mathsf{Com}(v_i)$
5: P_i sends c_i, com_1, com_2 and π_i.eq to P^*.
6: P^* verifies the proof, if it is valid and $c_1 = c_2$, then outputs 1; otherwise, P^* outputs 0.

$f_2(v_2)$. We denote the protocol as

$$1 \leftarrow \Pi_{\mathsf{PET}}(\text{pri:}\{v_i\}_{i\in\{1,2\}}; \text{pub:}\{f_i\}_{i\in\{1,2\}}) \iff f_1(v_1) = f_2(v_2)$$

.

Private 2-party equality test extension for two vectors. The above equality test can be extended to the vector case. Each P_i's private input is a vector $\mathbf{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ and the public input are two vectors of linear functions $\mathbf{f}_i = (f_{i1}, f_{i2}, \ldots, f_{in})$. This extension protocol is denoted as $\Pi_{\mathsf{PETe}}(\text{pri:}\{\mathbf{v}_i\}_{i\in\{1,2\}}; \text{pub:}\{\mathbf{f}_i\}_{i\in\{1,2\}})$ and achieves that:

$$1 \leftarrow \Pi_{\mathsf{PETe}}(\text{pri:}\{\mathbf{v}_i\}_{i\in\{1,2\}}; \text{pub:}\{\mathbf{f}_i\}_{i\in\{1,2\}}) \iff \mathbf{f}_1(\mathbf{v}_1) = \mathbf{f}_2(\mathbf{v}_2).$$

It tests the equality of two vectors of private function values $\mathbf{f}_i(\mathbf{v}_i) = (f_{i1}(v_{i1}), f_{i2}(v_{i2}), \ldots, f_{in}(v_{in}))$, $i \in \{1, 2\}$.

We give the construction Π_{PETe} in Algo. 3. It follows similar idea as Π_{PET} that treats each linear transformation element of the input vector $\mathbf{v}$ as a private input value to commit and prove, shares a random vector $\mathbf{s}$ with the same degree as the input vector, and generates a vector $\mathbf{d}$. The final value that P_i sends to P^* is the masked random linear combination $c_i = \Sigma_{j=1}^{n} f_{ij}(v_{ij})(s_j + d_j) + t + h$, and P_i generates corresponding proofs with respect to the following equality-extension relation $\mathcal{R}_{\mathsf{eqe}}$. This protocol outputs 1 only if the two vectors

of linear function values are identical. The reason is that all elements are combined with independent randomness, so that the two sums are equal only when each element in the two vectors is equal. All values are hidden and unequal positions are hidden.

$$\begin{aligned}\mathcal{R}_{\mathsf{eqe}} = \{(\{V_j, S_j\}_{j=1}^n, T, c, (f_1, ..., f_n); \{v_j, u_j\}_{j=1}^n, \{s_j, r_j\}_{j=1}^n, t, z) :\\ \forall j \in \{1, ..., n\}, V_j = \mathsf{Com}(v_j; u_j), S_j = \mathsf{Com}(s_j; r_j) \wedge T = \mathsf{Com}(t; z)\\ \wedge (\mathbf{d}, h) \leftarrow \mathit{Hash}(\mathsf{Com}(s_1; r_1) \| \ldots \|(\mathsf{Com}(s_n; r_n) \| \mathsf{Com}(t; z))\\ \wedge c = \Sigma_{j=1}^n f_j(v_j)(s_j + d_j) + t + h\}\end{aligned}$$

Algorithm 3 A private 2-party equality test on linear transformation of two vectors with semi-honest platform $\Pi_{\mathsf{PETe}}(\text{pri:}\{\mathbf{v}_i\}_{i\in\{1,2\}}; \text{pub:}\{\mathbf{f}_i\}_{i\in\{1,2\}})$.

1: Setup a group over $\mathbb{Z}_p^*$. There are authenticated private channels between platform P^* and each party P_1, P_2. Each P_i has a vector $\mathbf{v}_i = (v_{i1}, v_{i2}, \ldots, v_{in})$ as private input. The public input includes two vectors of linear functions $\mathbf{f}_i = (f_{i1}, f_{i2}, \ldots, f_{in})$
2: For $j \in [n]$, P_i sends $\mathsf{Com}(v_{ij})$ and NIZK proves knowledge of v_{ij} to P^*. P^* verifies them. // it has been done in the place order phase
3: P_1, P_2 share private randomness $\mathbf{s}, \mathbf{r}, t, z$ via Π_{CT}.
4: P_i computes $\mathsf{Com}(s_j; r_j)$ for $j \in [n]$, $\mathsf{Com}(t; z)$,
$(\mathbf{d}, h) \leftarrow \mathit{Hash}(\mathsf{Com}(s_1; r_1) \| \ldots \|(\mathsf{Com}(s_n; r_n) \| \mathsf{Com}(t; z))$,
$c_i = \Sigma_{j=1}^n f_{ij}(v_{ij}) \cdot (s_j + d_j) + t + h$ and generates NIZK proof π_i.eqe for $\mathcal{R}_{\mathsf{eqe}}$.
// random linear combinations of functions values
5: P_i sends c_i, $\{\mathsf{Com}(s_j; r_j)\}_{j=1}^n$, $\mathsf{Com}(t; z)$ to P^*.
6: P^* verifies the proof, if it is valid and $c_1 = c_2$, then outputs 1; otherwise, P^* outputs 0.

Eligibility check for all orders. The private 2-party equality test extension protocol Π_{PETe} can be directly used to check match eligibility between two orders, where the input vector has two elements $\mathsf{sym}, \mathsf{opt}$. All linear functions are identity functions except the one for the one party's option, which turns it to the opposite: $f(\mathsf{opt}) = 3 - \mathsf{opt}$.

Given M orders, each trader runs $M - 1$ times of Π_{PETe} in parallel with others. Based on all checking results, the platform divides these orders into different groups, called order books. Each book includes all orders with the same symbol and is partitioned into two sets, G_k^l and G_k^r, with different options. This protocol is described in Algo. 4, and we prove it securely implements the functionality $\mathcal{F}_{\mathsf{EC}}$ (Fig. 4.2).

Algorithm 4 Eligibility check Π_{EC}

P^* initializes $\mathbf{T} \leftarrow \mathbf{0}_{n\times n}$
$\forall i \in [M], P_i$ has input: $\mathbf{v}_i = (v_{i1}, v_{i2}) = (\mathsf{sym}_{i1}, \mathsf{opt}_{i2})$
for $i \neq j \in [M]$ **do**
    P_i, P_j, P^* run: $b_{ij} \leftarrow \Pi_{\mathsf{PTEe}}$(pri:$(\mathbf{v}_i, \mathbf{v}_j)$; pub: $(f_{i1}(v_{i1}) = v_{i1},\ f_{i2}(v_{i2}) = 3 - v_{i2},$ $f_{j1}(v_{j1}) = v_{j1},\ f_{j2}(v_{j2}) = v_{j2})$)
    P^* stores $T_{ij} \leftarrow b_{ij}$
end for // each party runs $M - 1$ Π_{PTEe} in parallel
P^* gets $\mathbf{T} = \{T_{ij}\}_{M\times M}$, goes through every element
// detect order books from T
P^* detects the indices sets G_k^l, G_k^r such that $T_{ij} = 1$ for every $i \in G_k^l, j \in G_k^r$.
Output: empty set $\emptyset$; // no eligible matched orders
or non-empty sets $G_1^l, G_1^r, ..., G_m^l, G_m^r \subseteq [M]$.

THEOREM 4. *Assume Π_{Com} is a secure commitment scheme, and the underlying NIZK proof system is simulation-sound, and Π_{CT} is a secure coin tossing protocol. The protocol Π_{EC} described in Algo. 4 securely implements the eligibility checking functionality $\mathcal{F}_{\mathsf{EC}}$ in Fig. 4.2 in the presence of malicious users or a semi-honest platform.*

PROOF. We consider the protocol in a hybrid model where underlying ideal functionalities are $\mathcal{F}_{\mathsf{zk}}(\mathcal{R})$, $\mathcal{F}_{\mathsf{CT}}$ as introduced in Sec. 2.3. Replacing them with secure protocols follows the composition theorem.

Here, we consider two different cases. The first is the privacy against one malicious user, and the second is the privacy against the semi-honest platform.

For each user, it only receives a common random seed by running the coin-tossing protocol with the other party. Since Π_{CT} securely implements the ideal coin-tossing functionality $\mathcal{F}_{\mathsf{CT}}$, $\mathcal{S}$ can simulate it perfectly with P_i. Thus, it does not learn any other information.

The platform receives the commitments and c_i.

The simulator receives the input and output of P^* from the trusted party. Its input is empty. The output is whether the two parties are eligible for matching. $\mathcal{S}$ does the following.

Setup: It invokes the simulated setup algorithm of Π_{ZK} to generate $(\mathsf{crs}, \mathsf{td})$. It gives crs to $\mathcal{A}$. $\mathcal{S}$ keeps the trapdoor td.

W.l.o.g., we use the party P_i as an example. The case of party P_j is similar. On behalf of P_i, $\mathcal{S}$ generates the commitments $com_{v_{ij}}$, com_{s_j}, $\{com_{v_{ij}s_j}\}$ on random strings, invokes the simulated prover algorithm of Π_{ZK} to generate the proof π_i^1 using the trapdoor td, sends them to $\mathcal{A}$. The case of party P_j is similar.

On receiving the result from the trusted party, if they are eligible for matching, S samples a random value as $c_i = c_j$. Otherwise, $\mathcal{S}$ samples two different random values as c_i, c_j. It generates the proof π_i^2, π_j^2 using the trapdoor td, sends them to $\mathcal{A}$. To show the indistinguishability between the real and ideal execution, we consider the following experiments:

• Expt_0 : It is the real execution.

• Expt_1 : It is similar to Expt_0 except that $\mathcal{S}$ generates the commitments on random strings and stimulates the NIZK proof π using the trapdoor.

Since the commitment scheme is hiding and the proof system is zero-knowledge, this experiment can be distinguished from Expt_0 with only negligible probability.

• Expt_2 : It is the ideal execution experiment we defined above. It is similar to Expt_2 except that $\mathcal{S}$ receives the check result from the trusted party. If they are eligible for matching, $\mathcal{S}$ samples a random value as $c_i^* = c_j^*$. Otherwise, $\mathcal{S}$ samples two different random values as c_i^*, c_j^*. It simulates the NIZK proof π_i^2, π_j^2 using the trapdoor td, sends to P^*.

Since d_j, $t+h$ are random values, c_i^* and C_i are in the same distribution. Thus, this experiment can be distinguished from Expt_2 with only negligible probability.

In summary, the outputs of real and ideal execution experiments are indistinguishable.

□

4.3.2 Private Batch Volume Match

After eligibility checking, the platform obtains multiple order books. The orders in the same book are divided into two sets, and they are sorted by submission time. We use m, n to denote

the size of the left and right sets. The traders are denoted by $P_1^l, ..., P_m^l$ and $P_1^r, ..., P_n^r$ in the sequence of order time, and their order volumes are $\mathsf{v}_1^l, ..., \mathsf{v}_m^l$ and $\mathsf{v}_1^r, ..., \mathsf{v}_n^r$. We denote the cumulative volumes as $\mathsf{cv}_i^l = \sum_{k=1}^{i} \mathsf{v}_k^l$, $\mathsf{cv}_j^r = \sum_{k=1}^{j} \mathsf{v}_k^r$, where $i \in [m], j \in [n]$. $(\mathsf{cv}_1^l, ..., \mathsf{cv}_m^l)$, $(\mathsf{cv}_1^r, ..., \mathsf{cv}_n^r)$ are monotonically increasing sequences. Comparing cv_m^l and cv_n^r, if $\mathsf{cv}_m^l < \mathsf{cv}_n^r$, it means all volumes on the left side can be fully matched. Then, we use the binary search to find the smallest index t such that $\mathsf{cv}_m^l \leq \mathsf{cv}_t^r$. The indices of matched orders and unmatched orders are the match result of the dark pool.

The Private Batch Volume Match procedure works as outlined in Algo. 6. We explain each step as follows.

Prepare: The trader has sent $\mathsf{Com}(\mathsf{sym}), \mathsf{Com}(\mathsf{opt}), \mathsf{Com}(\mathsf{v})$ and its public encryption key pk to P^* when it placed order. P^* sends all public keys of one set to each trader in the other set. Here, the encryption scheme should be additively homomorphic, such that $m_1 + m_2 = \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m_1) + \mathsf{Enc}_{pk}(m_2))$, e.g., Paillier encryption [90].

Each trader produces three shares of its volume: $([\mathsf{v}]_0, [\mathsf{v}]_1, [\mathsf{v}]_2)$ such that $\mathsf{v} = [\mathsf{v}]_0 + [\mathsf{v}]_1 + [\mathsf{v}]_2$. It encrypts the share $[\mathsf{v}]_0$ under all public keys of the other set, proves the two shares and these ciphertexts are correct via $\pi.\mathsf{share}$, and sends them to P^*. Here, the NIZK proof $\pi.\mathsf{share}$ is generated using the zero-knowledge proof system on the NP relation:

$$\mathcal{R}_{\mathsf{share}} = \{(com, [\mathsf{v}]_1, [\mathsf{v}]_2, \{pk_j, c_j\}_{j\in[n]}; \mathsf{v}, [\mathsf{v}]_0) :$$
$$com = \mathsf{Com}(\mathsf{v}) \wedge \mathsf{v} = [\mathsf{v}]_0 + [\mathsf{v}]_1 + [\mathsf{v}]_2$$
$$\wedge \forall j \in [n], c_j = \mathsf{Enc}_{pk_j}([\mathsf{v}]_0)\}$$

P^* verifies all proofs. If any invalid, abort. Otherwise, continue. P^* obtains set $\mathcal{D} = \{\{[\mathsf{v}_i^l]_1, [\mathsf{v}_i^l]_2, \{c_{i,j}^l\}_{j\in[n]}\}_{i\in[m]}, \{[\mathsf{v}_j^r]_1, [\mathsf{v}_j^r]_2, \{c_{j,i}^r\}_{i\in[m]}\}_{j\in[n]}\}$, i.e., for each v_i^l, P^* has its two shares $[\mathsf{v}_i^l]_1, [\mathsf{v}_i^l]_2$ and the ciphertexts of its third share under the public keys of the other set: $c_{i,j}^l = \mathsf{Enc}_{pk_{i,j}^r}([\mathsf{v}_i^l]_0)$ for $\forall j \in [n]$. For the case of v_j^r, it is similar.

Compare: With the set $\mathcal{D}$, P^* helps the traders match their orders by finding the match result via a binary search method. This protocol is shown in Algo. 5. This procedure requires multiple comparisons for the cumulative volumes, e.g., cv_i^l and cv_j^r. Comparing them is

actually comparing $\Delta_{ij}^{l-r} = \mathsf{cv}_i^l - \mathsf{cv}_j^r$ or $\Delta_{ji}^{r-l} = \mathsf{cv}_j^r - \mathsf{cv}_i^l$ with 0. We utilize a share-based three-party secure comparison protocol [85], denoted as Π_{3SC} that securely realizes the ideal functionality $\mathcal{F}_{\mathsf{3SC}}$ defined in the dishonest-majority setting in Sec. 2.3.

P^* runs the Π_{3SC} protocol on it with two traders P_α^l, P_β^r chosen randomly from the left and right sets. They need the shares of Δ_{ij}^{l-r} or Δ_{ji}^{r-l} as inputs. They gain the shares via the share distribution functionality $\mathcal{F}_{\mathsf{SD}}$ as shown in Fig. 4.7. The indicator $\mathsf{x} = 0$ means they get the shares of Δ_{ij}^{l-r}. $\mathsf{x} = 1$ indicates they get the shares of Δ_{ji}^{r-l}.

Functionality $\mathcal{F}_{\mathsf{SD}}$ (Share Distribution)
There are parties: $P_\alpha^l, P_\beta^r, P^*$.

- **Input**: P^* has $\mathcal{D} = \{\{[\mathsf{v}_k^l]_1, [\mathsf{v}_k^l]_2, \{c_{k,j}^l\}_{j\in[n]}\}_{k\in[m]}, \{[\mathsf{v}_k^r]_1, [\mathsf{v}_k^r]_2, \{c_{k,i}^r\}_{i\in[m]}\}_{k\in[n]}\}$, two cumulative value indices $i \in [m], j \in [n]$, an indicator $\mathsf{x} \in \{0,1\}$, two user indices $\alpha \in [m], \beta \in [n]$.
  P_α^l has private key sk_α^l. P_β^r has private key sk_β^r.
- **Output**: If $\mathsf{x} = 0$, send the share $[\Delta_{ij}^{l-r}]_0$ to P_α^l;
  send $[\Delta_{ij}^{l-r}]_1$ to P_β^r; send $[\Delta_{ij}^{l-r}]_2$ to P^*.
  If $\mathsf{x} = 1$, send the share $[\Delta_{ji}^{r-l}]_0$ to P_α^l;
  send $[\Delta_{ji}^{r-l}]_1$ to P_β^r; send $[\Delta_{ji}^{r-l}]_2$ to P^*.

FIGURE 4.7. The functionality of share distribution

P^* has collected the shares (some in plaintext and some in ciphertext form) of each volume value in the set $\mathcal{D}$. P_α^l has private key sk_α^l. P_β^r has private key sk_β^r. Due to the additive homomorphism of the shares and ciphertexts, they can run the protocol $\Pi_{\mathsf{SD}}(sk_\alpha^l, sk_\beta^r, (\mathcal{D}, \mathsf{x}, \alpha, \beta, i, j))$ by computing as follows, which achieves the $\mathcal{F}_{\mathsf{SD}}$ functionality. We take the case of $\mathsf{x} = 0$ as an example.

- P^* computes: $[\Delta_{ij}^{l-r}]_2 = [\mathsf{cv}_i^l]_2 - [\mathsf{cv}_j^r]_2$, $[\mathsf{cv}_i^l]_1 = \sum_{k=1}^i [\mathsf{v}_k^l]_1$, $[\mathsf{cv}_i^l]_2 = \sum_{k=1}^i [\mathsf{v}_k^l]_2$, $[\mathsf{cv}_j^r]_1 = \sum_{k=1}^j [\mathsf{v}_k^r]_1$, $[\mathsf{cv}_j^r]_2 = \sum_{k=1}^j [\mathsf{v}_k^r]_2$, sends $\mathsf{Enc}_{pk_\alpha^l}([\mathsf{cv}_i^l]_1) - \sum_{k=1}^j c_{k,\alpha}^r$ to P_α^l, sends $\sum_{k=1}^i c_{k,\beta}^l - \mathsf{Enc}_{pk_\beta^r}([\mathsf{cv}_j^r]_1)$ to P_β^r.
- P_α^l decrypts to get $[\Delta_{ij}^{l-r}]_0 = [\mathsf{cv}_i^l]_1 - \sum_{k=1}^j [\mathsf{v}_k^r]_0$.
- P_β^r decrypts to get $[\Delta_{ij}^{l-r}]_1 = \sum_{k=1}^i [\mathsf{v}_k^l]_0 - [\mathsf{cv}_j^r]_1$.

Based on the share distribution and secure comparison protocols, they run the binary search procedure as shown in Π_{BSSC}, Algo. 5. It first compares $\Delta_{mn}^{l-r} = \mathsf{cv}_m^l - \mathsf{cv}_n^r$ with 0. Only P^* can get the comparison result w_{mn}. If $\mathsf{w}_{mn} = 0$, then $\Delta_{mn}^{l-r} \geq 0$ and P^* binary searches the right set to find the smallest index t such that $\mathsf{cv}_t^r \geq \mathsf{cv}_m^l$, shown in steps 4-7. Otherwise, $\Delta_{mn}^{l-r} < 0$ and P^* binary searches the left set to find the smallest index t such that $\mathsf{cv}_t^l \geq \mathsf{cv}_n^r$, shown in steps 8-11. Finally, it outputs the match result w_{mn}, t.

- If $\mathsf{w}_{mn} = 0, t \in [m]$, it means all volumes $\mathsf{v}_1^l, \ldots, \mathsf{v}_{t-1}^l$ and $\mathsf{v}_1^r, \ldots, \mathsf{v}_n^r$ are fully matched, v_t^l is the last matched volume. Other volumes are unmatched.
- If $\mathsf{w}_{mn} = 1, t \in [n]$, it means all volumes $\mathsf{v}_1^l, \ldots, \mathsf{v}_m^l, \mathsf{v}_1^r, \ldots, \mathsf{v}_{t-1}^r$ can be fully matched, and v_t^r is the last matched volume. Other volumes are unmatched.

Within Π_{BSSC}, a total of $O(\log n)$ rounds of comparison computation are performed.

Algorithm 5 Binary Search with Secure Comparison Π_{BSSC}

1: **Input**: P^* has $\mathcal{D} = \{\{[\mathsf{v}_k^l]_1, [\mathsf{v}_k^l]_2, \{c_{kj}^l\}_{j\in[n]}\}_{k\in[m]}, \{[\mathsf{v}_k^r]_1, [\mathsf{v}_k^r]_2, \{c_{ki}^r\}_{i\in[m]}\}_{k\in[n]}\}$. P^* samples $\alpha \leftarrow\$ [m]$, $\beta \leftarrow\$ [n]$. P_α^l has secret key sk_α^l. P_β^r has sk_β^r.
2: $P_\alpha^l, P_\beta^r, P^*$ run $([\Delta_{mn}^{l-r}]_0, [\Delta_{mn}^{l-r}]_1, [\Delta_{mn}^{l-r}]_2) \leftarrow \Pi_{\mathsf{SD}}(sk_\alpha^l, sk_\beta^r, (\mathcal{D}, 0, \alpha, \beta, m, n))$
// get shares of Δ_{mn}^{l-r} via share distribution protocol
3: $P_\alpha^l, P_\beta^r, P^*$ run Π_{3SC} $([\Delta_{mn}^{l-r}]_0, [\Delta_{mn}^{l-r}]_1, [\Delta_{mn}^{l-r}]_2)$, and only P^* gets w_{mn}.
4: **if** $\mathsf{w}_{mn} = 0$ **then** // left cumulative volume is larger
$L \leftarrow 0, R \leftarrow m-1, t \leftarrow m$
5: **while** $L \leq R$ **do**
$z \leftarrow \lfloor \frac{L+R}{2} \rfloor$
$P_\alpha^l, P_\beta^r, P^*$ run $([\Delta_{zn}^{l-r}]_0, [\Delta_{zn}^{l-r}]_1, [\Delta_{zn}^{l-r}]_2) \leftarrow \Pi_{\mathsf{SD}}(sk_\alpha^l, sk_\beta^r, (\mathcal{D}, 0, \alpha, \beta, z, n))$
// get shares of Δ_{zn}^{l-r} via share distribution protocol
$P_\alpha^l, P_\beta^r, P^*$ run Π_{3SC} $([\Delta_{zn}^{l-r}]_0, [\Delta_{zn}^{l-r}]_1, [\Delta_{zn}^{l-r}]_2)$ and only P^* gets w_{zn}.
6: **if** $\mathsf{w}_{zn} = 0$ **then** $t \leftarrow z$, $R \leftarrow z-1$ // change candidate, and search a smaller one
7: **else** $L \leftarrow z+1$ // search a larger one
8: **else** // right cumulative volume is larger
$L \leftarrow 0, R \leftarrow n-1, t \leftarrow n$
9: **while** $L \leq R$ **do**
$z \leftarrow \lfloor \frac{L+R}{2} \rfloor$
$P_\alpha^l, P_\beta^r, P^*$ run $([\Delta_{zm}^{r-l}]_0, [\Delta_{zm}^{r-l}]_1, [\Delta_{zm}^{r-l}]_2) \leftarrow \Pi_{\mathsf{SD}}(sk_\alpha^l, sk_\beta^r, (\mathcal{D}, 1, \alpha, \beta, z, n))$
$P_\alpha^l, P_\beta^r, P^*$ run Π_{3SC} $([\Delta_{zm}^{r-l}]_0, [\Delta_{zm}^{r-l}]_1, [\Delta_{zm}^{r-l}]_2)$ and only P^* gets w_{zm}.
10: **if** $\mathsf{w}_{zm} = 0$ **then** $t \leftarrow z$, $R \leftarrow z-1$
11: **else** $L \leftarrow z+1$
12: **Return** w_{mn}, t // obtain the match result

Inform: On the (w_{mn}, t), if $\mathsf{w}_{mn} = 1, t \in [n]$, P^* informs each party $P_1^l, \ldots, P_m^l, P_1^r, \ldots, P_{t-1}^r$ that they are fully matched, and sends the ciphertext of its matched volume to P_t^r and informs other parties that they are unmatched. Due to the additive homomorphism of the encryption, $\mathsf{Enc}_{pk_t^r}(\Delta_{m,t-1}^{l-r}) = \mathsf{Enc}_{pk_t^r}(\mathsf{cv}_m^l - \mathsf{cv}_{t-1}^r)$ is computed from

$$\begin{aligned}
&\mathsf{Enc}_{pk_t^r}(\Delta_{m,t-1}^{l-r}) = \mathsf{Enc}_{pk_t^r}(\mathsf{cv}_m^l - \mathsf{cv}_{t-1}^r) \\
&= \mathsf{Enc}_{pk_t^r}(\sum_{k=1}^{m}[\mathsf{v}_k^l]_1 + \sum_{k=1}^{m}[\mathsf{v}_k^l]_2 - \sum_{k=1}^{t-1}[\mathsf{v}_k^r]_1 - \sum_{k=1}^{t-1}[\mathsf{v}_k^r]_2) \\
&+ \sum_{k=1}^{m} c_{k,t}^l - \sum_{k=1}^{t-1} c_{k,t}^r
\end{aligned} \tag{4.1}$$

If $\mathsf{w}_{mn} = 0$, P^* notifies all parties analogously.

Algorithm 6 Private Batch Volume Match Π_{PBVM}

1: **Input**: Each party P_i^l has input v_i^l and P_j^r has input v_j^r for $i \in [m], j \in [n]$. P^* has two sets of public encryption keys $PK^l = \{pk_i^l\}_{i\in[m]}, PK^r = \{pk_j^r\}_{j\in[n]}$.

2: **Prepare**: P^* sends PK^l to each P_j^r for all $j \in [n]$, and sends PK^l to each P_i^l for all $i \in [m]$.
- For each P_i^l, $i \in [m]$, it does the following:
run $([\mathsf{v}_i^l]_0, [\mathsf{v}_i^l]_1, [\mathsf{v}_i^l]_2) \leftarrow \Pi_{\mathsf{SS}}(\mathsf{v}_i^l)$;
generate $\{c_{i,j}^l = \mathsf{Enc}_{pk_j^r}([\mathsf{v}_i^l]_0)\}_{j\in[n]}$; // encrypt one share under all public keys of other set
generate NIZK proof $\pi_i^l.\mathsf{share}$ for the relation $\mathcal{R}_{\mathsf{share}}$; // prove correctness of shares and ciphertexts
send $([\mathsf{v}_i^l]_1, [\mathsf{v}_i^l]_2, \{c_{i,j}^l\}_{j\in[n]}, \pi_i^l.\mathsf{share})$ to P^*.
- For each P_j^r, $j \in [n]$, it does the same as above and sends $([\mathsf{v}_j^r]_1, [\mathsf{v}_j^r]_2, \{c_{j,i}^r\}_{i\in[m]}, \pi_j^r.\mathsf{share})$ to P^*.
- P^* verifies all proofs. If any invalid, abort. Otherwise, continue. P^* obtains set $\mathcal{D} = \{\{[\mathsf{v}_i^l]_1, [\mathsf{v}_i^l]_2, \{c_{i,j}^l\}_{j\in[n]}\}_{i\in[m]}, \{[\mathsf{v}_j^r]_1, [\mathsf{v}_j^r]_2, \{c_{j,i}^r\}_{i\in[m]}\}_{j\in[n]}\}$.

3: **Compare**: This phase is shown in Algo. 5. That is, P^* takes all the plain and encrypted shares and each party's public key as input and interacts with two sets of parties who have their private keys as input by running the binary search with secure comparison procedure $(P^* : \mathsf{w}_{mn}, t) \leftarrow \Pi_{\mathsf{BSSC}}$. // P^* gets the match result

4: **Inform**:
- If $\mathsf{w}_{mn} = 1$, P^* informs $\{P_1^l, \ldots, P_m^l, P_1^r, \ldots, P_{t-1}^r\}$ fully matched and $\{P_{t+1}^r, \ldots, P_n^r\}$ unmatched, sends $\mathsf{Enc}_{pk_t^r}(\Delta_{m,t-1}^{l-r})$ to P_t^r as computed by equation 4.1;
- If $\mathsf{w}_{mn} = 0$, P^* informs $\{P_1^r, \ldots, P_n^r, P_1^l, \ldots, P_{t-1}^l\}$ fully matched and $\{P_{t+1}^l, \ldots, P_m^l\}$ unmatched, sends $\mathsf{Enc}_{pk_t^l}(\Delta_{n,t-1}^{r-l})$ to P_t^l.

THEOREM 5. *Assume Π_{Com} is a secure commitment scheme, Π_{Enc} is a perfectly correct, IND-CPA secure additively homomorphic encryption scheme and $\Pi_{\mathsf{zk}}(\mathcal{R}_{\mathsf{share}})$, are simulation-sound NIZK proof systems for relations $\mathcal{R}_{\mathsf{share}}$, and Π_{3SC} is a secure three-party dishonest-majority comparison protocol. The Private Batch Volume Match Π_{PBVM} protocol (Algo. 6) securely implements the private batch volume matching functionality $\mathcal{F}_{\mathsf{VM}}$ in Fig.4.3 in the presence of malicious users or a semi-honest platform.*

PROOF. We consider the protocol in a hybrid model where underlying ideal functionalities are $\mathcal{F}_{\mathsf{zk}}(\mathcal{R})$, $\mathcal{F}_{\mathsf{3SC}}$ as introduced in Sec. 2.3. Replacing them with secure protocols follows the composition theorem. Here, we consider two different cases. One involves some malicious users. The other involves a semi-honest platform.

As in standard simulation-based security proofs, we exhibit an efficient simulator interacting with the ideal functionality $\mathcal{F}_{\mathsf{PBVM}}$ that can simulate the view of any efficient adversary corrupting a subset of the parties in the protocol execution. We assume that the corrupt sets $\mathcal{C}^l \subset [m]$ and $\mathcal{C}^r \subset [n]$ are fixed before the protocol execution starts. $\mathcal{H}^l = [m] \setminus \mathcal{C}^l, \mathcal{H}^r = [n] \setminus \mathcal{C}^r$ contain honest parties.

The case of malicious users: $\mathcal{S}$ does the following

Setup: It invokes the simulated setup algorithm of Π_{share} to generate $(\mathsf{crs}_{\mathsf{share}}, \mathsf{td}_{\mathsf{share}})$, gives $\mathsf{crs}_{\mathsf{share}}$ to $\mathcal{A}$, keeps $\mathsf{td}_{\mathsf{share}}$.

Prepare: After receiving two sets of corrupted parties' indices $\mathcal{C}^l, \mathcal{C}^r$ and their encryption public keys $\{pk_i\}_{i\in\mathcal{C}^l} \cup \{pk_j\}_{j\in\mathcal{C}^r}$ from $\mathcal{A}$, $\mathcal{S}$ generates the public-secret key pairs (pk_i, sk_i) for all honest parties in $\{P_i^l\}_{i\in\mathcal{H}^l} \cup \{P_j^r\}_{j\in\mathcal{H}^r}$.

W.l.o.g., we use the party P_i^l as an example. The case of party P_j^r is similar in the prepare phase. For all $i^l \in \mathcal{C}^l$, receive from $\mathcal{A}$ the commitments $\mathsf{com}_i^l, \mathsf{com}_{i0}^l$ and shares $[\mathsf{v}_i^l]_1, [\mathsf{v}_i^l]_2$, simulate $\mathcal{F}_{\mathsf{zk}}(\mathcal{R}_{\mathsf{share}})$ with it and run the extractor algorithm with the trapdoor $\mathsf{td}_{\mathsf{share}}$ to extract witness $\mathsf{v}_i^l, \omega_i^l, [\mathsf{v}_i^l]_0, \omega_{i0}^l$. (If some corrupted party P_i^l aborts, them abort.) Check that $\mathsf{com}_i^l = \mathsf{Com}(\mathsf{v}_i^l, \omega_i^l)$, $\mathsf{com}_{i0}^l = \mathsf{Com}([\mathsf{v}_i^l]_0, \omega_{i0}^l)$, $\mathsf{v}_i^l = [\mathsf{v}_i^l]_0 + [\mathsf{v}_i^l]_1 + [\mathsf{v}_i^l]_2$. If they hold, send

accept to $\mathcal{A}$; otherwise, return reject. If all checks pass, $\mathcal{S}$ publishes all public keys $\{pk_i^l\}_{i\in[m]}$ and $\{pk_j^r\}_{j\in[n]}$ in random orders and continues. Otherwise, send abort.

For each $i \in \mathcal{C}^l$, receive from $\mathcal{A}$: $\{c_{i,j}^l\}_{j\in[n]}$, simulate $\mathcal{F}_{\mathsf{zk}}(\mathcal{R}_{\mathsf{eq}})$ with it and run the extractor algorithm with the trapdoor $\mathsf{td}_{\mathsf{eq}}$ extract witness $[\mathsf{v}_i^l]_0', \omega_{i0}^{l\,\prime}, \omega_{i,j}$. (If some corrupted party P_i^l aborts, send abort.) Check that $[\mathsf{v}_i^l]_0' = [\mathsf{v}_i^l]_0$ as extracted before from that party P_i^l and $c_{i,j}^l = \mathsf{Enc}_{pk_j}([\mathsf{v}_i^l]_0', \omega_{i0}^{l\,\prime})$ for $j \in [n]$. If they hold, send accept to $\mathcal{A}$; otherwise, return reject.

If all checks pass, continue. Otherwise, send abort.

Compare: Initialize $\mathsf{round} = 0$. W.l.o.g., we assume $m \leq n$. Sample $\alpha \leftarrow\!\$\ [m]$, $\beta \leftarrow\!\$\ [n]$, do while $\mathsf{round} \leq \lfloor \log n \rfloor$+1:

(a) if both parties are honest, send them the ciphertexts of random strings under their public keys and simulate $\mathcal{F}_{\mathsf{3SC}}$ with them. Let $\mathsf{round} = \mathsf{round} + 1$;

(b) if only one party is corrupted, send it the ciphertext of a random string under its public key, simulate $\mathcal{F}_{\mathsf{3SC}}$ with it.

(c) if both parties are corrupted, send them the other party's identity and the ciphertexts of random strings under their public keys, respectively, and simulate $\mathcal{F}_{\mathsf{3SC}}$ with them.

(d) if any corrupted party aborts, send abort to the other party and sample new $\alpha \leftarrow\!\$\ [m]$, $\beta \leftarrow\!\$\ [n]$ and repeat the above procedures again.

(e) if no party aborts and finally send accept to $\mathcal{A}$, let $\mathsf{round} = \mathsf{round} + 1$, then sample a new pair $\alpha \leftarrow\!\$\ [m]$, $\beta \leftarrow\!\$\ [n]$ and repeat until $\mathsf{round} = \lfloor \log n \rfloor + 1$.

Inform: $\mathcal{S}$ sends all $\{\mathsf{v}_i^l\}_{i\in\mathcal{C}^l}$ and $\{\mathsf{v}_j^r\}_{j\in\mathcal{C}^r}$ to the trusted party and receive in return the matched parties' indices $\mathcal{M}^l \subseteq \mathcal{C}^l, \mathcal{M}^r \subseteq \mathcal{C}^r$ and their matched values $\{\mathsf{mv}_i^l\}_{i\in\mathcal{M}^l}$ and $\{\mathsf{mv}_j^r\}_{j\in\mathcal{M}^r}$. It sends $\mathcal{M}^l, \mathcal{M}^r$, $\{\mathsf{Enc}_{pk_i^l}(\mathsf{mv}_i^l)\}_{i\in\mathcal{M}^l}$, $\{\mathsf{Enc}_{pk_j^r}(\mathsf{mv}_j^r)\}_{j\in\mathcal{M}^r}$, $\{\mathsf{Enc}_{pk_i^l}(0)\}_{i\in\mathcal{C}^l\setminus\mathcal{M}^l}$, $\{\mathsf{Enc}_{pk_j^r}(0)\}_{j\in\mathcal{C}^r\setminus\mathcal{M}^r}$ to $\mathcal{A}$.

To show the indistinguishability between the real and ideal execution, we consider the following experiments. Each experiment's output is the view of the $\mathcal{A}$.

• Expt_0 : It is the real execution.

• Expt_1 : It is almost identical to Expt_0 except that the setup for Π_{share} is replaced with the simulated setup algorithm, the knowledge extractor is run with $\mathcal{A}$ for $\mathsf{com}_i^l, \mathsf{com}_{i0}^l, com_{i0}^l, \{c_{ij}^l\}_{j\in[n]}$, $\pi_i^l.\mathsf{share}$ to obtain $v_i^l, \omega_i^l, [v_i^l]_0, \omega_{i0}^l, [v_i^l]_0'$ and the randomnesses used in the ciphertexts and check their correctness and $v_i^l = [v_i^l]_0 + [v_i^l]_1 + [v_i^l]_2$. Check $[v_i^l]_0' = [v_i^l]_0$ and the correctness of the ciphertexts. If any of them do not hold, abort. Otherwise, continue. The indistinguishability of this experiment and Expt_0 follows from the proof system's simulation soundness and the encryption scheme's correctness.

• Expt_2 : It is almost identical to Expt_1 except that $\mathcal{A}$ interacts with the functionality $\mathcal{F}_{\mathsf{3SC}}$ simulated by $\mathcal{S}$.

Since P^* is not corrupted, the secret sharing is information-theoretic secure, and Π_{3SC} is a secure protocol that implements $\mathcal{F}_{\mathsf{3SC}}$, $\mathcal{A}$'s view can be simulated perfectly, so Expt_2 is perfectly indistinguishable from Expt_1.

• Expt_3 : It is the ideal execution experiment we defined above. It is almost identical to Expt_2 except that in the **Inform** stage, $\mathcal{S}$ sends $\{\mathsf{v}_i^l\}_{i\in\mathcal{C}^l}$ and $\{\mathsf{v}_j^r\}_{j\in\mathcal{C}^r}$ to the trusted party and receive in return the matched parties' indices $\mathcal{M}^l \subseteq \mathcal{C}^l, \mathcal{M}^r \subseteq \mathcal{C}^r$ and their matched values $\{\mathsf{mv}_i^l\}_{i\in\mathcal{M}^l}$ and $\{\mathsf{mv}_j^r\}_{j\in\mathcal{M}^r}$. Then it sends $\{\mathsf{Enc}_{pk_i^l}(\mathsf{mv}_i^l)\}_{i\in\mathcal{M}^l}$, $\{\mathsf{Enc}_{pk_j^r}(\mathsf{mv}_j^r)\}_{j\in\mathcal{M}^r}$, $\{\mathsf{Enc}_{pk_i^l}(0)\}_{i\in\mathcal{C}^l\setminus\mathcal{M}^l}$, $\{\mathsf{Enc}_{pk_j^r}(0)\}_{j\in\mathcal{C}^r\setminus\mathcal{M}^r}$ to $\mathcal{A}$.

The indistinguishability of this experiment and Expt_2 follows from the encryption scheme's correctness.

In summary, the outputs of real and ideal execution experiments are indistinguishable.

The case of semi-honest platform P^*: We show the security of a corrupted semi-honest P^*. The view of P^* consists of the public keys of all parties, the commitments of their secret volumes, the random shares, the commitments and ciphertexts of shares, the NIZK proofs for the correct commitment and encryptions, and its view in Π_{3SC} run with other two parties P_i^l, P_j^r, and the match results.

$\mathcal{S}$ receives the input and output of P^* from the trusted party. Its input is empty. The output is which parties can be matched. From the matched parties' identities, it can infer the result of each three-party secure comparison.

Setup: It invokes the simulated setup algorithm of Π_{share} to generate $(\mathsf{crs}_{\mathsf{share}}, \mathsf{td}_{\mathsf{share}})$, sends $\mathsf{crs}_{\mathsf{share}}$ to $\mathcal{A}$. $\mathcal{S}$ generates the public-secret key pairs for all parties and keeps $\mathsf{td}_{\mathsf{share}}$.

Prepare: W.l.o.g., we use the party P_i^l as an example. On behalf of P_i^l, $\mathcal{S}$ generates the commitments com_i^l, com_{i0}^l on random strings, samples random values as shares, generates ciphertexts $c_{i,j}^l$ on random strings under $\{pk_j^r\}_{j\in[n]}$, invokes the simulated prover algorithm of Π_{share} to generate the proof $\pi_i^l.\mathsf{share}$ using the trapdoor $\mathsf{td}_{\mathsf{share}}$, sends them to $\mathcal{A}$.

Compare: $\mathcal{S}$ receives from the trusted party values (w_{mn}, t) indicating the comparison result of each round. If $\mathsf{w}_{mn} = 1$, it means the first round comparison result is 1, and $\mathcal{S}$ can infer the following results $\mathsf{w}_k \in \{0, 1\}$ of the next $\lfloor \log n \rfloor + 1$ rounds based on t. If $\mathsf{w}_{mn} = 0$, it means the first round comparison result is 0, and $\mathcal{S}$ can infer the following results $\mathsf{w}_k \in \{0, 1\}$ of the next $\lfloor \log m \rfloor + 1$ rounds based on t.

First, for each comparison, $\mathcal{A}$ chooses two parties from the two sides and computes the ciphertexts respectively. Then it simulates the view of P^* in the invocations of $\mathcal{F}_{\mathsf{3SC}}$ in each round such that $\mathcal{A}$ can reconstruct the comparison result $\mathsf{w}'_k = \mathsf{w}_k$. Consider the following experiments:

• Expt_0 : It is the real execution.

• Expt_1 : It is similar to Expt_0 except that $\mathcal{S}$ replaces commitments with random strings, sends the ciphertexts on random strings and stimulates the NIZK proof $\pi.\mathsf{share}$ of each party using the trapdoor. Since the commitment scheme is hiding, the encryption scheme is IND-CPA secure, and the proof system is zero-knowledge, this experiment can be distinguished from Expt_0 with only negligible probability.

• Expt_2 : It is the ideal execution experiment we defined above. It is similar to Expt_1 except that $\mathcal{S}$ simulates invocations of $\mathcal{F}_{\mathsf{3SC}}$ with $\mathcal{A}$ which result in the same output from the trusted party. Since Π_{3SC} is a secure protocol that implements $\mathcal{F}_{\mathsf{3SC}}$, $\mathcal{A}$ learns nothing else except

its input and output. Its view can be simulated perfectly. Thus, this experiment can be distinguished from Expt_1 with only negligible probability.

In summary, the outputs of real and ideal execution experiments are indistinguishable.

□

4.3.3 Fully Private Volume Match

In this section, we aim to achieve a fully private volume match that reduces the leakage to a minimum. The platform does not even know the match results that which orders are matched.[4] Only traders know whether they get matched and their own matched volume.

The dark pool needs to ensure that at least one set's orders in the order book are fully matched, find all matched orders in the other set, and decide their matched volumes. To achieve this goal, existing methods, such as the one-by-one comparison and the previous private batch match, rely on a platform that knows each comparison result and decides the next comparison values and continues. In order to hide the match result further, we need a new matching mechanism that does not require the platform to know any internal comparison result.

However, even if we can hide the match result, another issue arises as to inform the last matched party (e.g., P_i^l) of its matched volume. In the previous method, the platform–knowing P_i^l's position–sends $\mathsf{Enc}_{pk_i^l}(\mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l)$ directly, allowing P_i^l to decrypt its matched volume. But now the platform does not know who the last matched party is, and it must treat all parties equally. If it sends the ciphertext of $\mathsf{cv}_n^r - \mathsf{cv}_{j-1}^l$ to each P_j^l, then fully matched or unmatched parties may decrypt values they should not see, e.g., P_1^l decrypting cv_n^r, which reveals information beyond its matched volume.

We propose the fully private match method as outlined in Algo. 8. The trader gets its match result by running two comparisons, in which the platform learns nothing. The platform can inform each trader of its matched volume without learning any additional information. We

[4]Except the basic fact that at least one side would be fully matched, and at least the first order on the other side is matched.

show a detailed explanation with the security proof. Before that, we introduce some useful observations.

Observ. 1. Get match result from two comparison results. Firstly, we observe that one's match result can be decided by two comparison results: the cumulative volume exactly before it with that of the other complete set $\langle \mathsf{cv}^l_{i-1}, \mathsf{cv}^r_n \rangle$, and the cumulative volume till it with that of the other complete set $\langle \mathsf{cv}^l_i, \mathsf{cv}^r_n \rangle$. Especially, $\mathsf{cv}^l_0 = 0$. There are three cases:

(1) If $\mathsf{cv}^l_{i-1} < \mathsf{cv}^r_n$, $\mathsf{cv}^l_i < \mathsf{cv}^r_n$, P^l_i is fully matched;
(2) If $\mathsf{cv}^l_{i-1} < \mathsf{cv}^r_n$, $\mathsf{cv}^l_i \geq \mathsf{cv}^r_n$, P^l_i is the last matched;
(3) If $\mathsf{cv}^l_{i-1} \geq \mathsf{cv}^r_n$, $\mathsf{cv}^l_i \geq \mathsf{cv}^r_n$, P^l_i is unmatched.

Note that the case $\mathsf{cv}^l_{i-1} \geq \mathsf{cv}^r_n \wedge \mathsf{cv}^l_i < \mathsf{cv}^r_n$ is impossible, since the sequence of cv is monotonically increasing. For the case of P^r_j, it works analogously.

For cases (1) and (3), P^l_i directly knows that the matched volume is his order volume or 0. But for case (2), it is the last matched party who cannot infer its (partial) matched volume $\mathsf{cv}^r_n - \mathsf{cv}^l_{i-1}$ just based on the comparison results. Thus, some additional technique is required.

Observ. 2. Get the opposite comparison result by adding 2^ℓ. In the comparison protocol [85] for a value Δ, the result is decided by comparing $\Delta \mod p$ with 2^ℓ. If $\Delta \mod p < 2^\ell$, it outputs 0, means $\Delta \geq 0$. Otherwise, it outputs 1, means $\Delta < 0$.

We choose k, ℓ such that $2^{k-1} < 2^\ell < 2^{k-1} + 2^\ell < 2^{\ell+1} < p$. For $\Delta \in [-2^{k-1}, 0) \cup [0, 2^{k-1})$, let us check the comparison result of $\Delta' = (\Delta + 2^\ell) \mod p$.

- When $\Delta \in [-2^{k-1}, 0)$, $\Delta \mod p > 2^\ell$, comparison result is 1; $\Delta' \mod p \in [2^\ell - 2^{k-1}, 2^\ell) < 2^\ell$, result is 0.
- When $\Delta \in [0, 2^{k-1})$, $\Delta \mod p < 2^\ell$, comparison result is 0; $\Delta' \mod p \in [2^\ell, 2^\ell + 2^{k-1}) \geq 2^\ell$, result is 1.

In summary, adding 2^ℓ yields an opposite comparison result.

Overview. Based on these observations, we introduce two random mask bits for traders' comparisons. In the Prepare phase, each party commits to two random bits e_1, e_2 and

their XOR result $e = e_1 \oplus e_2 = e_1 + e_2 - 2e_1e_2$. They can be used to hide the true comparison results from the platform. Instead of comparing the values directly, they compare the values added with $2^\ell \cdot e_1$ and $2^\ell \cdot e_2$ respectively. As the comparison results, P^* gets $\mu_1 = \mathsf{w}_1 \oplus e_1, \mu_2 = \mathsf{w}_2 \oplus e_2$, where $\mathsf{w}_1, \mathsf{w}_2 \in \{0, 1\}$ denote the true comparison results. Upon receiving the two masked results μ_1, μ_2, only the trader can recover $\mathsf{w}_1, \mathsf{w}_2$.

To deliver the matched volume, P^* computes the ciphertext for each party (taking P_i^l as an example): $\mathsf{pc} = \mathsf{Enc}_{pk_i^l}(\mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l)$. This ciphertext is then encrypted again with symmetric encryption as $\mathsf{sc} = \mathsf{Enc}_{ssk}(\mathsf{pc})$. *ssk* is the unique symmetric secret key chosen by P^* for P_i^l. In the Inform phase, we ensure that after the interaction between the platform and traders, only the last matched trader can get this symmetric key and decrypt to its matched volume.

Construction. We demonstrate the fully private volume match Π_{FPVM} as follows and summarize it in Algo. 8 later.

Prepare: This phase is similar to the Prepare phase in Algo. 6 except that each party chooses two random mask bits. In addition to sending the commitment, shares, and share ciphertexts of volumes, they also send the commitment, shares, and share ciphertexts of these random mask bits.

• For each P_i^l, $i \in [m]$, it chooses $e_1, e_2 \leftarrow\$ \{0, 1\}$ (we omit the subscript i and superscript l here for brevity) and generates their secret shares $[e_1]_0$, $[e_1]_1$, $[e_1]_2$, $[e_2]_0$, $[e_2]_1$, $[e_2]_2$, commitments $\mathsf{Com}(e_1)$, $\mathsf{Com}(e_2)$. It also generates $\mathsf{ec} = \mathsf{Com}(e; \omega) = g^e h^\omega$ for $e = e_1 \oplus e_2$, and generates NIZK proof $\pi.\mathsf{mask}$ for the following relation.

$$\mathcal{R}_{\mathsf{mask}} = \{(com_1, com_2, \mathsf{ec}, [e_1]_1, [e_1]_2, [e_2]_1, [e_2]_2);$$

$$(e_1, e_2, e, [e_1]_0, [e_2]_0, \omega) : e_1 \in \{0, 1\} \wedge e_2 \in \{0, 1\}$$

$$\wedge e_1 = [e_1]_0 + [e_1]_1 + [e_1]_2 \wedge com_1 = \mathsf{Com}(e_1)$$

$$\wedge e_2 = [e_2]_0 + [e_2]_1 + [e_2]_2 \wedge com_2 = \mathsf{Com}(e_2)$$

$$\wedge e = e_1 \oplus e_2 \wedge \mathsf{ec} = \mathsf{Com}(e; \omega) = g^e h^\omega\}$$

It sends $(\mathsf{ec}, [e_1]_1, [e_1]_2, [e_2]_1, [e_2]_2, \pi.\mathsf{mask})$ to P^*.

P^* checks the proof. If it is invalid, abort. Otherwise, P^* stores $(\mathsf{ec}, [e_1]_1, [e_1]_2, [e_2]_1, [e_2]_2)$.

• For each P_j^r, $j \in [n]$ in the right set, they do the same.

P^* obtains $\mathcal{D} \cup \mathcal{E}^l \cup \mathcal{E}^r$, where $\mathcal{D}$ is the same as that in Algo. 6, $\mathcal{E}^l = \{(\mathsf{ec}_i^l, [e_{i1}^l]_1, [e_{i1}^l]_2, [e_{i2}^l]_1, [e_{i2}^l]_2)\}_{i\in[m]}$, $\mathcal{E}^r = \{(\mathsf{ec}_j^r, [e_{j1}^r]_1, [e_{j1}^r]_2, [e_{j2}^r]_1, [e_{j2}^r]_2)\}_{j\in[n]}$.

Compare: For each P_i^l, $i \in [m]$, P^* chooses $\beta \leftarrow\$ [n]$ and interacts with P_i^l and P_β^r to run the share distribution protocol such that all three get one share of $\Delta_{\mathsf{i-1,n}}^{l-r}$. It also sends $[2^\ell \cdot e_{i1}^l]_1$ to P_β^r. P_i^l keeps $[e_{i1}^l]_0$. P^* has $[e_{i1}^l]_2$. They can compute the shares of $\Delta'^{l-r}_{\mathsf{i-1,n}}$ respectively and run the comparison protocol for $\Delta'^{l-r}_{\mathsf{i-1,n}}$ and P^* gets the result μ_{i1}^l. For value $\Delta_{\mathsf{i,n}}^{l-r}$, P_i^l, P_β^r, P^* run the comparison protocol on $\Delta'^{l-r}_{\mathsf{i,n}} = \Delta_{\mathsf{i,n}}^{l-r} + 2^\ell \cdot e_{i2}^l$ same as above. P^* gets μ_{i2}^l.

For each P_j^r, P^* sets $\beta = j$ and chooses $\alpha \leftarrow\$ [m]$ and interacts with P_α^l and P_j^r to run the share distribution protocols and masked comparison protocols similar to the above. We show the whole procedure in Algo. 7.

Algorithm 7 Masked Comparison $\Pi_{\mathsf{MCP}}(\mathcal{D} \cup \mathcal{E}^l \cup \mathcal{E}^r)$

1: **Input**: P^* has the data set $\mathcal{D} \cup \mathcal{E}^l \cup \mathcal{E}^r$. Each party P_i^l has private key sk_i^l and shares $[e_{i1}^l]_0, [e_{i2}^l]_0$. Each party P_j^r has private key sk_j^r and shares $[e_{j1}^r]_0, [e_{j2}^r]_0$.
2: For each P_i^l, $i \in [m]$, P^* sets $\alpha = i$, samples $\beta \leftarrow\$ [n]$.
• For $\Delta_{\mathsf{i-1,n}}^{l-r}$: P_i^l, P_β^r, P^* run $([\Delta_{\mathsf{i-1,n}}^{l-r}]_0, [\Delta_{\mathsf{i-1,n}}^{l-r}]_1, [\Delta_{\mathsf{i-1,n}}^{l-r}]_2) \leftarrow \Pi_{\mathsf{SD}}(sk_i^l, sk_\beta^r, (\mathcal{D}, 0, \alpha = i, \beta, i-1, n))$.
//distribute shares of $\Delta_{\mathsf{i-1,n}}^{l-r}$
P^* sends $[2^\ell \cdot e_{i1}^l]_1$ to P_β^r.
P_i^l, P_β^r, P^* compute the following respectively:
$[\Delta'^{l-r}_{\mathsf{i-1,n}}]_j = [\Delta_{\mathsf{i-1,n}}^{l-r}]_j + [2^\ell \cdot e_{i1}^l]_j$, $j \in \{0, 1, 2\}$
//compute shares for the masked comparison
run Π_{3SC} $([\Delta'^{l-r}_{\mathsf{i-1,n}}]_0, \Delta'^{l-r}_{\mathsf{i-1,n}}]_1, [\Delta'^{l-r}_{\mathsf{i-1,n}}]_2)$, and only P^* gets μ_{i1}^l.
• For $\Delta_{\mathsf{i,n}}^{l-r}$: P_i^l, P_β^r, P^* get $[\Delta'^{l-r}_{\mathsf{i,n}}]_0, [\Delta'^{l-r}_{\mathsf{i,n}}]_1, [\Delta'^{l-r}_{\mathsf{i,n}}]_2$ respectively same as above. They run Π_{3SC} $([\Delta'^{l-r}_{\mathsf{i,n}}]_0, [\Delta'^{l-r}_{\mathsf{i,n}}]_1, [\Delta'^{l-r}_{\mathsf{i,n}}]_2)$ and P^* gets μ_{i2}^l.
//P^* gets the second masked comparison result
3: For each P_j^r, $j \in [n]$, P^* samples $\alpha \leftarrow\$ [m]$, sets $\beta = j$. P_α^l, P_j^r, P^* run the protocols similar to the above.
P^* gets μ_{j1}^r, μ_{j2}^r. //masked comparison results of P_j^r

Inform: In this phase, P^* informs each trader of its match result and matched volume. We also use P_i^l as the example and omit its subscript i and superscript l here for brevity. P^* computes $\mathsf{pc} = \mathsf{Enc}_{pk_i^l}(\mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l)$ as in equation (4.1). It then chooses $r^* \leftarrow\$ \mathbb{Z}_p$, computes $ssk = Hash(g^{r^*})$ and uses it as the key to encrypt pc with symmetric encryption: $\mathsf{sc} = \mathsf{Enc}_{ssk}(\mathsf{pc})$.

We require that only if P_i^l is the last matched trader can it get ssk. Note that the distinguished feature of such a party is that its two comparison results are *different*, i.e., $\mathsf{w} = \mathsf{w}_1 \oplus \mathsf{w}_2 = 1$, while all other traders' two comparison results are the same. But now P^* only has two masked comparison results μ_1, μ_2 and $\mu = \mu_1 \oplus \mu_2$.

Recall that it also has everyone's $\mathsf{ec} = g^e h^\omega$, where only the trader knows the two random bits' XOR result $e = e_1 \oplus e_2$ and the randomness ω. Based on each trader's μ and ec, P^* computes c_1, c_2 as follows to encode r^*.

- If $\mu = 1$, $c_1 = h^{r^*}$, $c_2 = (g/\mathsf{ec})^{r^*}$;
- If $\mu = 0$, $c_1 = h^{-r^*}$, $c_2 = \mathsf{ec}^{r^*}$.

Then it sends μ_1, μ_2, c_1, c_2 and sc to P_i^l.

P_i^l computes $Hash(c_2 \cdot {c_1}^\omega)$ using its secret ω. Only if $\mathsf{w} = 1$, $Hash(c_2 \cdot c_1^\omega) = Hash(g^{r^*}) = ssk$. Then, it can recover the valid ciphertext pc and decrypts it to the matched volume $\mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l$. Otherwise, it only knows its order is fully matched or unmatched based on $\mathsf{w}_1 = \mu_1 \oplus e_1, \mathsf{w}_2 = \mu_2 \oplus e_2$ according to observation 1.

THEOREM 6. *Assume Π_{Com} is a secure commitment scheme, Π_{Enc} is a perfectly correct, IND-CPA secure additively homomorphic encryption scheme and $\Pi_{\mathsf{zk}}(\mathcal{R}_{\mathsf{share}})$, $\Pi_{\mathsf{zk}}(\mathcal{R}_{\mathsf{mask}})$ are simulation-sound NIZK proof systems for relations $\mathcal{R}_{\mathsf{share}}$, $\mathcal{R}_{\mathsf{mask}}$, and Π_{3SC} is a secure three-party dishonest-majority comparison protocol. The Fully Private Volume Match protocol Π_{FPVM} (Algo. 8) securely implements the fully private volume matching functionality $\mathcal{F}_{\mathsf{FVM}}$ in Fig.4.4 in the presence of malicious users or a semi-honest platform.*

PROOF. This fully private protocol is similar to the private batch protocol. We mainly discuss the different parts.

Algorithm 8 Fully Private Volume Match Π_{FPVM}

1: **Input**: Each party P_i^l has input v_i^l and P_j^r has input v_j^r for $i \in [m], j \in [n]$. P^* has two sets of public encryption keys $PK^l = \{pk_i^l\}_{i\in[m]}, PK^r = \{pk_j^r\}_{j\in[n]}$.

2: **Prepare**: This phase is similar to the Prepare phase in Algo. 6 except that each party additionally chooses two random mask bits e_1, e_2 and a randomness ω. It keeps $(e_1, e_2, [e_1]_0, [e_2]_0, \omega)$ and sends the commitments ec, shares of its random mask bits $[e_1]_1, [e_1]_2, [e_2]_1$,$[e_2]_2$, and NIZK proof $\pi.\mathsf{mask}$) to P^*. P^* obtains $\mathcal{D} \cup \mathcal{E}^l \cup \mathcal{E}^r$.

3: **Compare**: P^* takes all the data $\mathcal{D} \cup \mathcal{E}^l \cup \mathcal{E}^r$ and each party's public key as input and interacts with two sets of parties who have their private keys as input by running the masked comparison procedure Π_{MCP} in Algo. 7.
For each party P_i^l, $i \in [m]$, P^* obtains μ_{i1}^l, μ_{i2}^l.
For each party P_j^r, $j \in [n]$, P^* obtains μ_{j1}^r, μ_{j2}^r.
// P^* gets masked comparison results of all parties

4: **Inform**: P^* does the following computations.
• For party P_i^l with μ_{i1}^l, μ_{i2}^l:
$\mathsf{pc}_i^l \leftarrow \mathsf{Enc}_{pk_i^l}(\mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l)$;
// compute ciphertext of $\mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l$, where $\mathsf{cv}_0^l = 0$
$r^* \leftarrow\$ \mathbb{Z}_p$, $ssk_i^l = \mathit{Hash}(g^{r^*})$, $\mathsf{sc}_i^l \leftarrow \mathsf{Enc}_{ssk_i^l}(\mathsf{pc}_i^l)$;
//compute ssk_i^l as symmetric key to encrypt pc_i^l
- If $\mu_i^l = \mu_{i1}^l \oplus \mu_{i2}^l = 1$:
$(c_{i1}^l, c_{i2}^l) = (h^{r^*}, (g/\mathsf{ec}_i^l)^{r^*} = g^{(1-e_i^l)\cdot r^*} h^{-\omega_i^l \cdot r^*})$.
- If $\mu_i^l = \mu_{i1}^l \oplus \mu_{i2}^l = 0$:
$(c_{i1}^l, c_{i2}^l) = (h^{-r^*}, (\mathsf{ec}_i^l)^{r^*} = g^{e_i^l \cdot r^*} h^{\omega_i^l \cdot r^*})$
sends $\mu_{i1}^l, \mu_{i2}^l, (c_{i1}^l, c_{i2}^l), \mathsf{sc}_i^l$ to P_i^l.
• For party P_j^r with μ_{j1}^r, μ_{j2}^r: computes similar to above, sends $\mu_{j1}^r, \mu_{j2}^r, (c_{j1}^l, c_{j2}^l), \mathsf{sc}_j^r$ to P_j^r.
//send each party with the encrypted ciphertext and encoding of decryption key
• Each P_i^l computes $\mathsf{w}_{i1}^l = \mu_{i1}^l \oplus e_{i1}^l$, $\mathsf{w}_{i2}^l = \mu_{i2}^l \oplus e_{i2}^l$:
- If $\mathsf{w}_{i1}^l = 1$ and $\mathsf{w}_{i2}^l = 1$, then $\mathsf{mv}_i^l = \mathsf{v}_i^l$; //full-match
- If $\mathsf{w}_{i1}^l = 1$ and $\mathsf{w}_{i2}^l = 0$, compute
$ssk_i^l = \mathit{Hash}(c_{i2}^l \cdot {c_{i1}^l}^{\omega_i^l})$, $\mathsf{pc}_i^l \leftarrow \mathsf{Dec}_{ssk_i^l}(\mathsf{sc}_i^l)$,
$\mathsf{mv}_i^l = \mathsf{cv}_n^r - \mathsf{cv}_{i-1}^l \leftarrow \mathsf{Dec}_{sk_i^l}(\mathsf{pc}_i^l)$;
//P_i^l is the last matched, decrypts matched volume
- If $\mathsf{w}_{i1}^l = 0$ and $\mathsf{w}_{i2}^l = 0$, then $\mathsf{mv}_i^l = 0$.//unmatched
• For each party P_j^r in the right set, they operate similarly and get their match results.

The case of malicious users: $\mathcal{S}$ is similar to that in the proof of Thm 5 except for the following.

Setup: It also invokes simulated setup algorithm of Π_{mask} to generate $(\mathsf{crs}_{\mathsf{mask}}, \mathsf{td}_{\mathsf{mask}})$, gives $\mathsf{crs}_{\mathsf{mask}}$ to $\mathcal{A}$, keeps $\mathsf{td}_{\mathsf{mask}}$.

Prepare: W.l.o.g., we use the party P_i^l as an example. The case of party P_j^r is similar. It also simulates the random bits commitment generation for mask. It works as follows.

For each $i \in \mathcal{C}^l$, receive from $\mathcal{A}$ the commitments $\mathsf{com}_{i1}^l, \mathsf{com}_{i2}^l$, ec_i^l, $\mathsf{c}_{i1}^l, \mathsf{c}_{i2}^l$ and shares $[e_{i1}^l]_1, [e_{i1}^l]_2$, $[e_{i2}^l]_1, [e_{i2}^l]_2$, simulate $\mathcal{F}_{\mathsf{zk}}(\mathcal{R}_{\mathsf{mask}})$ with it and run the extractor algorithm with the trapdoor $\mathsf{td}_{\mathsf{mask}}$ to extract witness $e_{i1}^l, \omega_{i1}^l, e_{i2}^l, \omega_{i2}^l, e_i^l, \omega_i^l$, $[e_{i1}^l]_0, \gamma_{i1}^l, [e_{i2}^l]_0, \gamma_{i2}^l$. (If some corrupted party P_i^l aborts, then abort.)

Check that $\mathsf{com}_{i1}^l = \mathsf{Com}(e_{i1}^l, \omega_{i1}^l)$, $\mathsf{c}_{i1}^l = \mathsf{Com}([e_{i1}^l]_0, \gamma_{i1}^l)$, $e_{i1}^l = [e_{i1}^l]_0 + [e_{i1}^l]_1 + [e_{i1}^l]_2$, $\mathsf{com}_{i2}^l = \mathsf{Com}(e_{i2}^l, \omega_{i2}^l)$, $\mathsf{c}_{i2}^l = \mathsf{Com}([e_{i2}^l]_0, \gamma_{i2}^l)$, $e_{i2}^l = [e_{i2}^l]_0 + [e_{i2}^l]_1 + [e_{i2}^l]_2$, $\mathsf{ec}_i^l = g^{e_i^l} h^{\omega_i^l}$, $e_i = e_{i1}^l \oplus e_{i2}^l$. If they hold, send accept to $\mathcal{A}$; otherwise, return abort.

$\mathcal{S}$ sends all $\{\mathsf{v}_i^l\}_{i \in \mathcal{C}^l}$ and $\{\mathsf{v}_j^r\}_{j \in \mathcal{C}^r}$ to the trusted party and receives $\mathcal{M}^l \subseteq \mathcal{C}^l, \mathcal{M}^r \subseteq \mathcal{C}^r$ and $\{\mathsf{mv}_i^l\}_{i \in \mathcal{M}^l}$, $\{\mathsf{mv}_j^r\}_{j \in \mathcal{M}^r}$ and whether he is the last matched party. For the last matched party, it has $\mathsf{w}_{i1}^l = 1, \mathsf{w}_{i2}^l = 0$. For other fully matched (but not last), it has $\mathsf{w}_{i1}^l = 1, \mathsf{w}_{i2}^l = 1$.

Compare: For the corrupted P_i^l, sample $\beta \leftarrow\!\!\$\ [n]$,

(a) if P_β^r is not corrupted, send P_i^l the ciphertext of a random string under its public key and simulate $\mathcal{F}_{\mathsf{3SC}}$ two times.

(b) if both parties are corrupted, send them the other party's identity and the ciphertexts of random strings under their public keys, respectively, and simulate $\mathcal{F}_{\mathsf{3SC}}$ two times.

(c) if anyone aborts, send abort to the other party and abort.

Inform: Based on each corrupted party's final matched result and their random bits, $\mathcal{S}$ obtains their comparison results: $\mu_{i1}^l = \mathsf{w}_{i1}^l \oplus e_{i1}^l, \mu_{i2}^l = \mathsf{w}_{i2}^l \oplus e_{i2}^l$ and sends them to $\mathcal{A}$. If P_i^l is not the last matched party, send $\mathsf{pc}_i^l, \mathsf{sc}_i^l$ and ec_i^{*l}, where $\mathsf{pc}_i^l, \mathsf{sc}_i^l$ are ciphertexts of random strings under pk_i^l and ec_i^{*l} is computed same as the protocol. If P_i^l is the last matched party, send $\mathsf{pc}_i^l, \mathsf{sc}_i^l$ and ec_i^{*l}, where $\mathsf{pc}_i^l = \mathsf{Enc}_{pk_i^l}(\mathsf{mv}_i^l)$, sc_i^l, ec_i^{*l} are computed same as the protocol. For each party P_j^r, do the same.

To show the indistinguishability between the real and ideal execution, we consider the following experiments. Each experiment's output is the view of $\mathcal{A}$.

• Expt_0 : It is the real execution.

• Expt_1 : It is almost identical to Expt_0 except that the setup for Π_{share} is replaced with the simulated setup algorithm, the knowledge extractor is run with $\mathcal{A}$ for $\mathsf{com}_i^l, \mathsf{com}_{i0}^l, [v_i^l]_1, [v_i^l]_2$ $\pi_i^l.\mathsf{share}$ to obtain $v_i^l, \omega_i^l, [v_i^l]_0, \omega_{i0}^l$ and check their correctness and $v_i^l = [v_i^l]_0 + [v_i^l]_1 + [v_i^l]_2$. If any of them does not hold, abort. Otherwise, continue.

The indistinguishability of this experiment and Expt_0 follows from the proof system's simulation soundness.

• Expt_2 : It is almost identical to Expt_1 except that the setup for Π_{mask} is replaced with the simulated setup algorithm, the knowledge extractor is run with $\mathcal{A}$ for $\mathsf{com}_{i1}^l, \mathsf{com}_{i2}^l, \mathsf{ec}_i^l, \mathsf{c}_{i1}^l, \mathsf{c}_{i2}^l$ and shares $[e_{i1}^l]_1, [e_{i1}^l]_2, [e_{i2}^l]_1, [e_{i2}^l]_2$, $\pi_i^l.\mathsf{mask}$ to obtain $e_{i1}^l, \omega_{i1}^l, e_{i2}^l, \omega_{i2}^l, e_i^l, \omega_i^l, [e_{i1}^l]_0, \gamma_{i1}^l, [e_{i2}^l]_0, \gamma_{i2}^l$ and check their correctness and $e_{i1}^l = [e_{i1}^l]_0 + [e_{i1}^l]_1 + [e_{i1}^l]_2$ and $e_{i2}^l = [e_{i2}^l]_0 + [e_{i2}^l]_1 + [e_{i2}^l]_2$. If any of them do not hold, abort. Otherwise, continue.

The indistinguishability of this experiment and Expt_0 follows from the proof system's simulation soundness.

• Expt_3 : It is almost identical to Expt_2 except that the ciphertexts sent to $\mathcal{A}$ in the Compare phase are encryptions on random strings and $\mathcal{A}$ interacts with the functionality $\mathcal{F}_{\mathsf{3SC}}$ simulated by $\mathcal{S}$. Since P^* is not corrupted, the secret sharing is information-theoretic secure, and Π_{3SC} is a secure protocol that implements $\mathcal{F}_{\mathsf{3SC}}$, $\mathcal{A}$'s view can be simulated perfectly, so Expt_3 is perfectly indistinguishable from Expt_2.

• Expt_4 : It is the ideal execution experiment we defined above. It is almost identical to Expt_3 except that in the **Inform** stage, since $\mathcal{S}$ sends $\mathcal{A}$ the comparison results of all corrupted parties computed from their final matched results and random bits. Besides, $\mathcal{S}$ sends $\mathcal{A}$: ($\mathsf{pc}_i^l, \mathsf{sc}_i^l, \mathsf{ec}_i^{*l}$) or ($\mathsf{pc}_i^r, \mathsf{sc}_i^r, \mathsf{ec}_i^{*r}$) for each corrupted party. If P_i^l is not the last matched party, let $\mathsf{pc}_i^l, \mathsf{sc}_i^l$ be the random strings and ec_i^{*l} computed same as the protocol. If P_i^l is the last matched party, let $\mathsf{pc}_i^l = \mathsf{Enc}_{pk_i^l}(\mathsf{mv}_i^l)$, and $\mathsf{sc}_i^l, \mathsf{ec}_i^{*l}$ computed same as the protocol. For the corrupted party P_i^r, do the same operation. For the last matched party, it receives the same ciphertexts as those in real execution. For other parties, since the symmetric encryption and

Paillier encryption are IND-CPA secure, $\mathcal{A}$ cannot distinguish the random strings from the valid ciphertexts. Thus, $\mathcal{A}$ can distinguish Exp_5 from Exp_4 with negligible probability.

In summary, the outputs of real and ideal execution experiments are indistinguishable.

The case of semi-honest platform P^*: $\mathcal{S}$ is similar to that in the proof of Thm 2 except for the following.

Setup: It also invokes simulated setup of Π_{mask} to generate $(\mathsf{crs}_{\mathsf{mask}}, \mathsf{td}_{\mathsf{mask}})$, gives $\mathsf{crs}_{\mathsf{mask}}$ to $\mathcal{A}$ and keeps $\mathsf{td}_{\mathsf{mask}}$.

Prepare: It is almost the same as that in Thm 2, except that on behalf of party P_i^l, $\mathcal{S}$ also produces commitments $\mathsf{com}_{i1}^l, \mathsf{com}_{i2}^l$, ec_i^l, $\mathsf{c}_{i1}^l, \mathsf{c}_{i2}^l$ on random strings and random values $[e_{i1}^l]_1, [e_{i1}^l]_2$, $[e_{i2}^l]_1, [e_{i2}^l]_2$, generates the proof $\pi_i^l.\mathsf{mask}$ using the trapdoor $\mathsf{td}_{\mathsf{mask}}$. P_j^r is similar.

Compare: For each P_i^l, $\mathcal{S}$ chooses random bits μ_{i1}^l, μ_{i2}^l, then it simulates the view of P^* in the invocations of $\mathcal{F}_{\mathsf{3SC}}$ two times such that $\mathcal{A}$ can reconstruct the comparison result ${\mu'}_{i1}^l = \mu_{i1}^l$, ${\mu'}_{i2}^l = \mu_{i2}^l$.

Inform: P^* receives nothing in this phase, so $\mathcal{S}$ does nothing either. To show the indistinguishability between the real and ideal execution, we consider the following experiments:

• Expt_0 : It is the real execution.

• Expt_1 : It is similar to Expt_0 except that $\mathcal{S}$ replaces commitments with random strings and simulates the NIZK proof $\pi.\mathsf{share}$ of each party using the trapdoor. Since the commitment scheme is hiding and the proof system is zero-knowledge, it can be distinguished from Expt_0 with only negligible probability.

• Expt_2 : It is similar to Expt_1 except that $\mathcal{S}$ replaces commitments with random strings and simulates the NIZK proof $\pi.\mathsf{mask}$ of each party using the trapdoor. Since the commitment scheme is hiding and the proof system is zero-knowledge, it can be distinguished from Expt_1 with only negligible probability.

• Expt_3 : It is the ideal execution experiment we defined above. It is similar to Expt_2 except that $\mathcal{S}$ simulates invocations of $\mathcal{F}_{\mathsf{3SC}}$ with $\mathcal{A}$ which result in the same output from the trusted party. Since Π_{3SC} is a secure protocol that implements $\mathcal{F}_{\mathsf{3SC}}$, $\mathcal{A}$ learns nothing else except its input and output. Its view can be simulated perfectly. Thus, it can be distinguished from Expt_2 with only negligible probability.

In summary, the outputs of real and ideal execution experiments are indistinguishable.

□

4.4 Cryptocurrency Dark Pool Trading System

The overall system runs in the following phases, which we describe on a high level. It can be seen as a combination of our eligibility check and private match protocols with the private exchange system, such as Pisces [32].

Registration. The user shows its real identity to the platform. The platform checks whether it has registered. If yes, aborts. Otherwise, they run the anonymous credential-issuing protocol. The user gets an identity credential.

Deposit. The user deposits cryptocurrencies into their account anonymously and gets an asset credential issued by the platform, which is also responsible for asset custody. It is the same as that in [32].

Place order. The user's order is $\mathsf{ord} = (\mathsf{sym}, \mathsf{opt}, \mathsf{v})$. It places its order by generating commitments $\mathsf{Com}(\mathsf{sym})$, $\mathsf{Com}(\mathsf{opt})$, $\mathsf{Com}(\mathsf{v})$ to these values and proves it has a valid asset credential containing enough cryptocurrencies for selling or USDT for buying. It also encrypts its identity id and order information under the regulator's public key $\mathsf{Enc}(\mathsf{id}), \mathsf{Enc}(\mathsf{ord})$ and proves its correctness. The platform verifies them. If the verification passes, the platform adds the order to the dark pool.

Eligibility check. On $\mathsf{Com}(\mathsf{sym})$, $\mathsf{Com}(\mathsf{opt})$ in its order, the trader joins the eligibility check phase and runs the Π_{EC} protocol (Algo. 4 in Sec. 4.3.1) with the platform and other traders in the dark pool. As a result, the platform outputs the non-empty order books.

Volume match. On the commitment $\mathsf{Com}(\mathsf{v})$ in its order, the trader runs the volume match protocol with the platform and some other traders in the same order book. They can run the private batch volume match protocol Π_{PBVM} (Algo. 6 in Sec. 4.3.2) or the fully private volume match protocol Π_{FPVM} (Algo. 8 in Sec. 4.3.3). As a result, the trader knows whether it is matched and the matched volume. Then, it can request the platform to issue new asset credentials anonymously.

Withdraw. The user withdraws cryptocurrencies from the platform anonymously on the asset credential issued by the platform. It is the same as that in Pisces [32].

Disclose. The regulator discloses the order information of the matched and executed orders using its secret key.

4.5 Performance

In this section, we present the performance evaluation of two phases of the dark pool system: the eligibility check and volume match, along with their time cost trends as the number of traders increases under limited concurrent computational resources. The overall evaluation results demonstrate that each trader's computation takes only a few seconds to obtain the match result, even with 200 orders in the dark pool, which confirms the practicality of our approach.

Evaluation configuration. We implemented our system using *Python 3.10.12*. We used the open-source library *MpyC* [**mpyc**] for simulating multi-party communication at all stages except the *Compare* stage. We also used *MP-SPDZ v0.3.9* [91] to specifically benchmark the three-party interactions (i.e. *Compare* in Algo. 6 and Algo. 8) with the dishonest majority security protocol *MASCOT* [92].

We conducted our experiments on a standard desktop PC equipped with a 16-core Intel i7-13700 processor and 16GB of RAM, running Ubuntu 20.04. Note that the computational power for testing is similar to that of real-world end users but much lower than that of a real-world platform with more powerful resources.

Eligibility check. We implemented the eligibility check algorithm Π_{EC} (Algo. 4) and measured the running time of the traders and the platform during this phase.

When there are only two orders in the dark pool, the two traders run a single round of the extended equality test protocol Π_{PTEe} with each other via the platform. In this case, a single run takes 0.607s to process. Theoretically, as the number of orders in the dark pool increases, the number of Π_{PTEe} rounds that each trader runs increases linearly, and the platform's rounds increase quadratically since it interacts with all traders. All rounds of Π_{PTEe} can be executed in parallel if the parties have sufficient concurrent computational resources. However, due to limited experimental resources, our experimental environment only supports a maximum of 32 threads for concurrent computation. As a result, we experimented with single thread conditions, as depicted in Fig. 4.8(A).

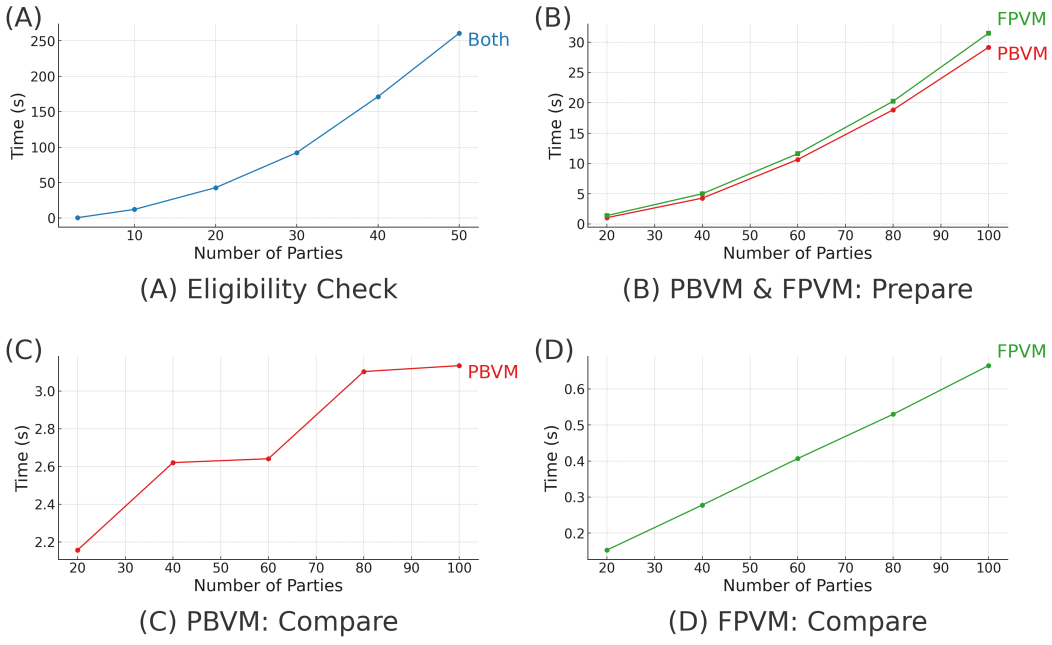

FIGURE 4.8. Parties' time cost in the Eligibility Check (Π_{EC}).

Volume match. We implemented two of our volume match algorithms Π_{PBVM} in Algo. 6 and Π_{FPVM} in Algo. 8, which consist of three main stages: *Prepare*, *Compare*, and *Inform*.

Prepare Stage. The computational performance of the *Prepare* stage is shown in Fig. 4.8(B). The cost of time increases quadratically due to the linear verification times for each user and the number of users. When the number of orders (one order per trader) increases from 20 to 100, the time cost grows as expected.

Compare Stage. In the *Compare* stage, each trader performs a constant number of rounds of either the three-party secure comparison protocol Π_{3SC} or the masked comparison protocol Π_{MCP}. We evaluated a single run of Π_{3SC} and Π_{MCP}, which takes 0.029s and 0.030s, respectively, to complete one round. Our Private Batch Volume Match requires logarithmic rounds Π_{3SC} in sequential order [5], shown in Fig. 4.8(C). On the other hand, our Fully Private Volume Match essentially runs in linear times of Π_{MCP}, as depicted in Fig. 4.8(D). However, in theory, it can be run concurrently and hence be further optimized.

Inform Stage. In this stage, the platform computes the results locally and distributes them to the traders. Traders then locally reveal the result as described in Algo. 6 and Algo. 8. The time cost on the platform side is less than 0.035s for 20-100 parties. On the trader side, it is independent of the number of parties, which is approximately 0.044s.

Overall Analysis. Our design is much faster than the state-of-the-art dark pool solution with a single server [72]. Concretely, we take the case of 200 orders with half buying and half selling as an example and run with at most 16-thread concurrent computation power. The total runtime of our private batch volume match is estimated to be 11.154s. Our fully private volume match is estimated as 8.061s. At the same time, only the online phase of the volume match of SPDZ-full threshold [72] cost around 8s according to the almost linear trend of "full th $N = M = 100$" case shown in Fig. 3 of [72] and the offline phase is too long to be shown, much more than 2500s.

[5] Due to the extra computation cost introduced by the used SPDZ framework, PBVM's compare time is higher than expected in Fig. 4.8(C). However, the trend still reflects the expected growth rate.

CHAPTER 5

Fair Relay in Anonymous Zether

5.1 Introduction

Blockchain is a distributed ledger technology that allows for secure, transparent, and tamper-proof recording of transactions across multiple participants. Decentralized consensus mechanisms, such as Proof-of-Work or Proof-of-Stake, are used to verify the authenticity of transactions and make blockchains resilient to fraud and censorship. Ethereum is a popular blockchain platform. It supports decentralized applications (dApps) through its Ethereum Virtual Machine (EVM), allowing developers to deploy programmable smart contracts. Ether (ETH), Ethereum's native cryptocurrency, fuels the network by paying the miners computational resources and transaction fees (referred to as "gas").

Smart contracts, central to Ethereum, are self-executing contracts with the terms of the agreement directly written into code, and they automatically execute once predefined conditions are met, such as transferring funds or updating records on the blockchain, eliminating the need for trusted intermediaries. Smart contracts are transparent, immutable, and decentralized, which cannot be altered once deployed on the blockchain.

Due to the transparency of Ethereum, privacy concerns have been raised. Each Ethereum user and their accounts (referred to as externally owned accounts, EOA in short) are only pseudonymous, meaning transactions can be traced back to account addresses, making it easy to track user activity.

5.1.1 Anonymous Zether for privacy

To address the privacy issue in the public blockchain, several projects [93, 94, 16, 17] have been designed to allow users to send anonymous transactions on the blockchain. One of them is anonymous Zether [16, 17], a private payment system built as a smart contract on Ethereum. It uses zero-knowledge proofs (ZKPs) to hide transaction amounts and user identities. Its deployment is currently underway [95]. The sender hides himself and the receiver in a large group with other users' public keys (the anonymity set) and encrypts transfer amounts under their public keys with ZKPs to show they are well-formed. In this way, anonymous transactions are created on Ethereum without requiring specialized blockchains like Zcash [14] or Monero [9]. Its native token is Zether (ZTH).

Inherent leakage via gas fee. In Ethereum, every transaction (initiated by an EOA) has to pay gas fees to miners in ETH. It leads to one of the main limitations[1] of anonymous Zether's anonymity: no matter what privacy guarantees it has, transactions broadcasted by the same EOA are inevitably linked via the gas fee payment. Even if a user controls multiple EOAs, he has to somehow ensure they remain unlinkable to send anonymous transactions.

This issue can be solved if miners are willing to receive transaction fees in ZTH instead of ETH, which are exchangeable by design and have identical values. However, this requires *all miners* to accept a fee in ZTH, which is a system-scale change.

Current mitigation: gas relay system. There are gas relay systems [96, 97, 98], in which users send their signed transactions to a third party, known as a *relayer*, off-chain. The relayer broadcasts the signed transaction to Ethereum and pays a gas fee for it. Therefore, users do not need to pay gas fees directly. Instead, the relayer covers the gas costs on their behalf. This is useful for applications where users may not have ETH to cover gas costs or to ensure a better user experience.

[1]The deanonymization attack in the network layer is out of the scope of our work. Users can protect themselves using an anonymous network like Tor [57].

Such relayer nodes could broadcast the anonymous transactions [16, 17]. Users send the *meta-transaction*[2] of anonymous Zether to the relayer. The relayer verifies and wraps them into standard Ethereum transactions, pays the gas fees, and propagates them. As compensation, the relayer could be rewarded in ZTH by adding their Zether account to the transaction as a receiver. The reward can be higher than the gas fee it pays, which incentivizes the relayer to provide such a service. While this approach seems promising for addressing the above tracing problem, it has brought about other issues. We will elaborate on the following.

5.1.2 Concurrency Attack

Briefly speaking, we observe a kind of concurrency attack on the relayers. It arises from the interplay between Ethereum's gas fee mechanism and anonymous Zether's double-spending prevention design. We will explain these components and demonstrate how the attack is executed.

Gas fee for failed transactions. In Ethereum, transactions that run out of gas or fail validation can still be included in a block without altering the state, but the gas fee is still charged. It is essential since it is impossible to distinguish between failed transactions caused by user error and those due to malicious intent like a denial of service. For example, attackers could spam the network with deliberately failing transactions, like creating complex contract interactions that fail at the end. Regardless of the result, the gas fee for the transaction is deducted from the sender's account [7, 99].

Double-spending prevention via nonce. The anonymous Zether defines the *epoch*, which is a discrete period (usually several blocks). Each anonymous Zether meta-transaction includes a *nonce*[3]. Given a new meta-transaction, it would be rejected if its nonce has appeared before in the same epoch. This condition prevents double-spending attacks. More details can be found in Section 2.4 or the original paper [16].

[2]We explicitly name the calling of a smart contract as a *meta-transaction*, to distinguish it from the *full* standard Ethereum transaction.

[3]Note that it differs from the `Nonce` in a full transaction.

Concurrency attack on multiple relayers. Considering the scenario where multiple relayers are available, they operate independently, and there are no direct communication channels among them. They only share the same public blockchain.

The malicious user requests many relayers with conflict meta-transactions, including the same nonce. Each relayer can verify the single request he received according to the current blockchain but does not know whether the user has requested other relayers. Once the relayers are convinced, they broadcast the transactions and pay the gas fee. However, due to the verification conditions, only the first transaction recorded on the chain would be executed successfully. Others with the same nonce are also recorded on-chain but cannot be executed. Therefore, only one relayer could be paid, while others' gas fees are wasted. Even worse, since the meta-transaction is anonymous, the malicious sender would never be identified and can repeatedly launch the attack without penalty. This unfairness would discourage people from serving as relayers and hurt the users' privacy potentially.

Beyond anonymous Zether, this attack also affects relayers supporting other privacy-preserving applications, such as anonymous auctions [100, 101]. We refer to a detailed description of Sec. 5.2.

Naive patching methods. One might question if the above attack could be easily circumvented. First of all, we do not hope to involve any trusted party or a small-scale committee that contradicts the full decentralization of blockchain. We explored several potential methods. Unfortunately, none of them works.

(1) *Pay before broadcast*: If the sender pays the relayer before the transaction is broadcast with a gas fee, the relayer could take the payment but fail to broadcast it, blaming network issues. This results in the honest sender losing money.

(2) *Pay if on-chain*: If the sender pays the relayer as long as the transaction is recorded on the chain (even if it fails), the relayer can delay the broadcast significantly until it is invalid and gets paid. The sender wastes his gas fee.

(3) *Cancel transaction*: The relayer can cancel that transaction by sending another one with the same nonce. But he still needs to pay a gas fee for the new transaction. So, it does not solve the issue.

Our goals. In summary, we aim to design a fair relay system that preserves user anonymity while ensuring fairness for users and relayers. With the proper incentive design, this system can attract sufficient relayers and foster the growth of the privacy-preserving ecosystem. All our work is motivated by the goal of protecting users' privacy.

5.1.3 Our Contributions

In this article, we begin by identifying the concurrency attack. To address this issue, we then introduce a novel type of anonymous credential, which serves as the foundation for constructing, for the first time, a fair gas relay system for anonymous Zether.

Identifying an attack. We identify the concurrency attack that happens when trivially combining a gas relay mechanism with the anonymous Zether smart contract. The user can concurrently request many relayers to broadcast conflict meta-transactions and waste their gas fees without penalization. This attack also affects other applications, such as anonymous auctions.

New primitive. We introduce a new primitive, namely *abuse deterring anonymous credential* (ADAC), and give an efficient construction with a formal security proof. Both communication cost and computation cost remain constant. It allows users to anonymously sign *message-tag* pairs with a credential. However, some secret information can be extracted if a user signs the same tag more than once, even with credentials from different issuers. Otherwise, it ensures complete user anonymity, preventing framing or unauthorized extraction of secret information. We believe it can be of independent interest.

Fair gas relay system. Finally, we propose the fair gas relay system model for anonymous payment. It ensures user anonymity and gas fee fairness. Relayers are guaranteed payment for broadcasting transactions honestly, while honest users are protected against financial loss and

privacy breaches. Additionally, we present a concrete scheme for the fair gas relay system based on ADAC without altering the anonymous Zether contract. The relayer only needs to attach an additional scalar element and one group element to the full transaction. We also analyze the additional computational cost for the users and relayers. The evaluation results indicate that the added communication and computation overheads remain minimal.

5.1.4 Technical Overview

Typically, two main parties are involved in a gas relay system: the users and the relayers. The main idea is that the relayers set their collateral smart contracts. The anonymous Zether users who want to use the relay service must register with the relayer and deposit ETH in the relayer's collateral smart contract and commit some *secret information*. If the user launches the concurrency attack on the relayers, his secret information will be extracted. On receiving the secret information, the collateral smart contract will send the user's collateral to the relayer as reimbursement. We show how the user deposits collateral and victim relayers get reimbursed via steps in Fig. 5.1 and Fig. 5.2.

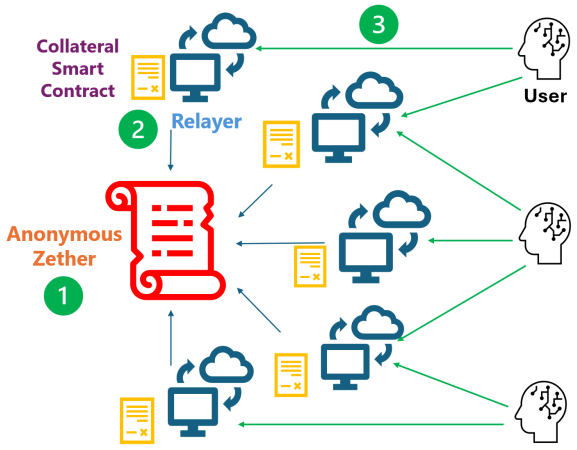

FIGURE 5.1. Relay system setup with collateral smart contracts

- Fig. 1. Step ①: The main smart contract (anonymous Zether) was deployed. Users have joined it and obtained the Zether accounts (which are linked to their Ethereum accounts).
- Fig. 1. Step ②: Each relayer sets up a *collateral smart contract* on the blockchain and embeds his Ethereum account address.

- Fig. 1. Step ③: Users register with the relayer. They deposit ETH in the relayer's collateral smart contract and commit some *secret information*. Note that each user can register to multiple relayers but must use the same secret information. This registration links to the user's account but should be unlinkable to honest users' transactions.

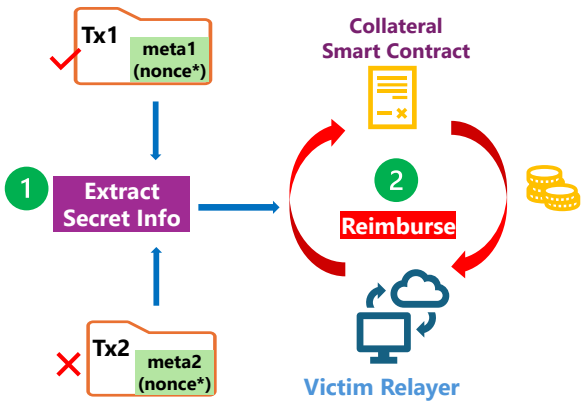

FIGURE 5.2. Reimburse the victim relayer

- Fig. 2. Step ①: The transaction (Tx2) broadcasted by the victim relayer is failed. He can extract the malicious user's secret information from the two conflict transactions (Tx1, Tx2).
- Fig. 2. Step ②: The victim relayer sends the user's secret information to the collateral smart contract. Then the smart contract sends the user's collateral to the relayer as reimbursement.

Extracting secret information. To achieve the above reimburse function, when the user sends a meta-transaction to the relayer, he should attach some other materials that can be used to extract his secret information as long as the misbehavior is detected. In this process, he only shows that he is a registered user without affecting the privacy provided by the anonymous Zether.

Note that extracting the secret information is a more stringent requirement than merely identifying the user's identity. This property is essential because we aim for the collateral smart contract to be economical. The contract should automatically and efficiently determine if an attack has occurred based on brief evidence provided by the relayer. Since invoking the smart contract incurs a gas fee for the resulting computation cost, simply identifying the user's public key is insufficient. Further complex checks, such as reading or verifying entire

historical transactions, would be necessary, resulting in high gas fees. Therefore, revealing the user's public identity alone is not adequate.

Insufficiency of existing primitives. With ring signatures [102] or anonymous credentials [103, 104], the users are anonymous and cannot be traced. In group signatures [105] and traceable signatures [106], the manager can deanonymize users arbitrarily.

Extracting public identity. There are traceable ring signatures [107] and event-oriented k-times revocable-iff-linked group signatures [108]. They managed to open the misbehaved user's identity if he signs on the same message or event more than once. However, the revealed identity is already public rather than secret information. So, it cannot be used in punishment without further check conditions.

Similarly, Chaum-style e-cash systems and certain one-show anonymous credentials [105, 109, 110] allow the user identity to be revealed in case of double-spending a coin, and these systems consider only a single-issuer model, typically involving a central bank, which is different from our setting.

Extracting secret key. Accountable assertions [111] and double authentication preventing signatures [112] are designed to prevent authenticating different messages with the same tag by enabling the extraction of the signing key in such cases. However, these schemes do not preserve signers' identities, even for honest ones.

In [113], the authors introduced the updatable anonymous credentials and utilized it to implement an offline double-spending tracing mechanism. This approach allows the extraction of a user's secret key if they sign multiple transactions using the same credential. However, each credential is single-use, requiring the issuer to issue a new credential to users after every signing, and it cannot achieve extraction across credentials issued by different issuers.

Proactive deterring. There have been some works on proactive deterring mechanisms on unauthorized leakage [114, 115, 104, 116, 117, 118], via different techniques of "witness identification/extraction". But they are different from our setting.

Non-transferable anonymous credentials [104] and tokens [116] discourage users from sharing their credentials with others by embedding some secret keys in the credentials/tokens. Sharing credentials/tokens would cause other harm, such as identity leakage, financial loss, etc. They only consider prevent *direct sharing/leakage* and do not consider extracting such secret information from usage.

Leakage-deterring primitives [114, 115] require users to embed their secret information into their public keys. This ensures that any access to even a partially functional implementation of the primitive (e.g., decryption or signing oracle) can be used to recover the embedded secret. Consequently, these mechanisms deter users from sharing their decryption or signing capabilities (via e.g., hardware token). In particular, leakage-deterring signatures put further restrictions on the distribution of messages to be signed, and the extractor needs to query signatures on the same message polynomial times to extract the secret. In our scenario, the user never shares the credential or its associated functionality.

TABLE 5.1. Comparisons of relevant primitives.

Primitives	Prv-MS[1]	Ext-Sec[2]	Mul-Iss[3]	OT-Iss[4]	Anony[5]
TR. Sig. [107]	✓	×	×	✓	✓
EO. Sig. [108]	✓	×	×	✓	✓
AA[111]/DAPS[112]	✓	✓	×	×	×
Up. Cred. [113]	✓	✓	×	×	✓
NT. Cred. [104]	×	✓	×	✓	✓
NT. Token.[116]	×	✓	×	✓	✓
LD. Sig [114]	×	✓	×	✓	×
Our ADAC.	✓	✓	✓	✓	✓

[1] Prv-MS means preventing the user from signing multiple times.
[2] Ext-Sec means some secret information can be extracted.
[3] Mul-Iss means achieving extraction across credentials from multiple issuers.
[4] OT-Iss means the issuer only needs to issue one credential to a user at one time.
[5] Anony means honest signers keep anonymous.

Formulating abuse deterring anonymous credential. Motivated based on the above discussion, we introduce the *abuse deterring anonymous credential* (ADAC) that deters users from abusing anonymity. It accommodates multiple independent issuers, such as relayers. Users can register with different issuers to obtain multiple credentials, all linked to a single identity and secret information. With these credentials, users can sign message-tag pairs anonymously.

However, if a user signs the same tag more than once, even with credentials from different issuers, their identity and secret information will be revealed. Otherwise, the system ensures complete user anonymity, preventing framing or unauthorized extraction of secret information. We compare ADAC with other related primitives in Table 5.1. Notably, our extraction feature allows secret information to be revealed if a tag is signed multiple times by the same user with many credentials issued by different issuers.

The main idea of the extraction in ADAC is similar to that of knowledge extractor. The user hides his secret information in response to the challenge, where the mask factor is determined by the tag and the user's secret key. By comparing responses across challenges for the same tag, the hidden secret can be extracted, mirroring how the rewind technique extracts information in the proof of knowledge.

We present a concrete construction based on the PS randomizable signatures [41]. It allows issuers to sign user-committed messages. The user then uses the signature as a credential. To preserve anonymity, the user rerandomizes the signature and demonstrates knowledge of the associated messages in zero-knowledge. Our design allows a user to obtain multiple credentials from different issuers, all tied to a single identity represented by the same secret seed s. Each user's credentials share the same s, which derives their secret information $\tilde{g}^s$. This value must remain private. When signing a message m and tag tag, the user computes $T = (\tilde{g}^c H(tag))^s$, where c is a public random challenge derived from m. The user also proves in zero-knowledge that s matches the secret in his credential, ensuring consistency and anonymity. However, if a tag is authenticated by the same user multiple times, the verifier obtains (tag, c_1, T_1) and (tag, c_2, T_2). Finally, the verifier extracts the user's secret information by computing $(T_1/T_2)^{\frac{1}{c_1 - c_2}} = \tilde{g}^s$.

Fair gas relay system overview. Now, let us zoom out and have a bird's eye view of the whole fair gas relay system for anonymous Zether. We will illustrate how the ADAC is applied to achieve fairness and analyze the user's anonymity set.

As introduced in the intuitive idea, each relayer establishes a collateral smart contract, requiring users to deposit ETH as collateral during registration. The smart contract encodes

withdrawal conditions that depend on specific evidence. The user receives an ADAC issued by the relayer, containing his unique secret information. This secret information serves as evidence of user misbehavior and can be immediately verified by the smart contract.

Then, the user requests the relayer to broadcast its meta-transaction, which contains a nonce. He authenticates the meta-transaction along with the nonce as the tag using its ADAC by producing the $(tag = nonce, c, T)$ tuple and proving the correctness of T. If the malicious user sends meta-transactions to different relayers with the same nonce, they wrap them with (tag, c_1, T_1) and (tag, c_2, T_2) respectively and post them on-chain. Thus, one of the transactions fails, and the secret information can be extracted. As a result, the relayer can withdraw the malicious user's collateral as reimbursement by sending the secret information to the smart contract.

Regarding honest users, they only request a single relayer and choose dummy Zether accounts that are also registered to it. It prevents the relayer from excluding unregistered users from the anonymity set. In this case, the user keeps the same anonymity level provided by anonymous Zether, and the relayer cannot identify the user on its request, which is guaranteed by the anonymity of ADAC and anonymous Zether.

More applications of ADAC. Besides the relay system, we believe our abuse deterring anonymous credential can be of independent interest and find more applications. Here are some examples.

- *Anonymous voting.* Each user possesses multiple credentials issued by independent authorities, such as passports and driver's licenses, all of which can be used for anonymous voting. A malicious user may exploit this by using different credentials to cast multiple votes. To prevent this, we aim to detect such fraudulent activity and extract their secret information to enforce penalties.
- *Anonymous concert tickets.* A concert sells tickets on many platforms, each using its own secret and public keys to issue ticket credentials. Users register on these platforms and purchase tickets anonymously. Since these platforms operate independently and do not share customer information, they cannot determine whether a user has bought tickets from

other platforms. However, when redeeming tickets, if a user attempts to redeem tickets from multiple platforms, their identity and secret information should be exposed.

5.2 Concurrency Attack on Relayers

This section shows how trivially applying the gas relay mechanism to the anonymous Zether system can be exploited. A malicious user can target multiple relayers at once, causing them to waste gas fees with minimal cost, and he will not be detected or punished.

Setting. A public blockchain, e.g., Ethereum, enables the execution of smart contracts. These contracts are triggered by submitting transactions, which incur gas fees. Notably, even failed transactions are recorded on the blockchain and still consume gas fees.

Anonymous Zether, the underlying privacy-preserving smart contract, adheres to correctness, overdraft-safety, and privacy requirements. As described in Sec. 2.4, each meta-transaction contains a unique nonce tied to the current epoch and the sender's secret key. A set is maintained for every epoch to record nonces of all valid transactions. The verification process ensures that a new transaction will fail if its nonce already exists in the set.

A gas relay system facilitates this submission process for users. Upon receiving a meta-transaction from a user, a relayer broadcasts it and pays the gas fee. Numerous independent relayers operate decentralized over the Internet, without central servers or direct coordination, using only the public blockchain as a shared reference.

Attack. A malicious user launches a concurrency attack as follows:

- **Step 1.** The user selects multiple relayers $R_1, R_2, .., R_k$.
- **Step 2.** He generates k meta-transactions $\mathsf{meta}_1, \mathsf{meta}_2, .., \mathsf{meta}_k$ for the same $epoch$, where each meta_i contains the same nonce u and includes R_i as one of the receivers to get the pre-agreed service fee in ZTH.
- **Step 3.** He requests all relayers R_i concurrently to broadcast the meta-transaction meta_i and pay the gas fee.

- **Step 4.** On receiving the request, R_i verifies meta_i independently according to the current Ethereum state and epoch. If it gets verified, he is convinced that he will be paid the service fee in ZTH as long as the transaction is executed. R_i wraps meta_i in a standard full transaction, broadcasts it, and pays the gas fee.

Impacts on fairness. Each R_i can verify meta_i, but they cannot determine whether the user requested other relayers within the same epoch since there is no direct coordination among them. While they share the blockchain, it has not recorded these transactions yet. When transactions are recorded, the gas fees have already been paid. Due to this dilemma, it does not help to check meta_i either.

Since these meta-transactions share the same nonce, only one can be executed and pay the relayer. The remaining $k-1$ failed transactions cause those relayers to lose their gas fees. As a result, the user pays only one relayer while the other $k-1$ relayers bear the cost of wasted gas fees. Although each individual gas fee is small, the total loss across multiple relayers can be significant. Furthermore, since users submit only anonymous meta-transactions, they face no charges or penalties, creating an unfair situation for relayers.

To achieve fairness in the gas fee payment without affecting the security of the underlying anonymous payment system, we propose the abuse deterring anonymous credential in the next section and describe how to use it to design the fair relay system in Section 5.4.

Beyond anonymous Zether. Except for anonymous Zether, the concurrency attack can also target relay systems supporting other anonymous smart contracts, such as anonymous auctions [101, 100]. In an anonymous auction smart contract, the auctioneer first initiates the auction. Bidders then submit their bid transactions on-chain, and the highest bidder wins. Similar to anonymous payments, neither the bid meta-transaction nor the gas fee payment should reveal the bidder's identity, necessitating the use of a relay system.

If a malicious bidder submits bid meta-transactions to multiple relayers, all will verify and submit them on-chain. However, since only one bid is valid per auction session, only one transaction executes, and only one relayer gets paid, while the rest incur gas costs without compensation.

Thus, the concurrency attack threatens relay systems across various privacy-preserving applications. In Section 5.4, we explain how our fair relay system addresses this issue.

5.3 Abuse Deterring Anonymous Credentials

In this section, we introduce our new abuse deterring anonymous credential (ADAC). It allows the user to register with different issuers, obtaining credentials tied to a single identity and secret information. While acting honestly, the user can sign tag-message pairs anonymously. However, the user's identity and secret information will be *extracted* if the same tag is signed more than once, even using credentials from different issuers.

Extractability intuition. We deter the user from abusing anonymity by extracting the malicious user's secret information. The main idea is similar to the knowledge extractor. The user hides his secret information in response to the challenge, where the mask factor is determined by the tag and the user's secret key. The tag anchors the user's responses to a single state, while the different challenges force the user to recompute masked responses. These recomputations inadvertently expose relationships tied to the user's secret information. Thus, by comparing responses for the same tag, the hidden secret can be extracted, mirroring how the rewind technique extracts information in the proof of knowledge.

5.3.1 Syntax

Our ADAC consists of algorithms: ADAC.Setup, ADAC.IssuerKeyGen, ADAC.UserKeyGen, ADAC.Request, ADAC.Issue, ADAC.Receive, ADAC.Show, ADAC.Verify, ADAC.Extract. It includes the following parties:

Issuers: We consider multiple issuers who are issuing credentials. They agree on the same algorithm and public parameters and generate their own keys.

Users: Each user binds with a unique identity. They can request credentials from many issuers, but they can only be issued once by each issuer. On a given tag and message, the user can use one credential to sign and generate a token anonymously.

Verifiers: Anyone can check whether the user has registered with the issuer by verifying the tuple of tag, message, and token. The verifier also links the tokens on the same tag. If it happens, the verifier extracts the user's secret information.

We explain the ADAC algorithms as follows:

- ADAC.Setup: generate the public parameters *pp*. For simplicity, pp will implicitly be an input of all the following algorithms.
- $(ipk, isk) \leftarrow$ ADAC.IssuerKeyGen: generate the issuer's secret and public key: *isk*, *ipk*.
- $(upk, usk, s, pid, sif) \leftarrow$ ADAC.UserKeyGen: generate the user's public key *upk* and secret key *usk*. Especially, usk contains secret seed *s*, and secret information *sif*, and *upk* contains a public identifier *pid*. There exists a deterministic one-way function that can derive *pid* from *sif*, i.e., $pid \leftarrow \mathsf{CompPid}(sif)$.
- $req \leftarrow$ ADAC.Request(ipk, upk, usk): on input issuer's public key *ipk* and user's public key upk, secret key usk, the user runs this algorithm and outputs a credential request *req*.
- $cred'/\perp \leftarrow$ ADAC.Issue(ipk, upk, req, isk): on input user's public key *upk*, issuer's public key *ipk*, issuer's secret key isk and credential request *req*, the issuer runs this algorithm and outputs $\perp$ on failure and otherwise a partial credential *cred'*.
- $cred/\perp \leftarrow$ ADAC.Receive$(ipk, req, cred', usk)$: on input user's secret key *usk*, credential request *req*, issuer's public key *ipk*, partial credential *cred'* generated by the issuer, a user runs this algorithm and outputs $\perp$ on failure and otherwise a valid credential.
- $\Sigma \leftarrow$ ADAC.Show$(ipk, m, tag, usk, cred)$: on input the issuer's public key *ipk*, user's secret key *usk*, credential *cred*, message m and tag, the user runs this algorithm and outputs a token Σ.
- $0/1 \leftarrow$ ADAC.Verify(ipk, m, tag, Σ): on input the issuer's public key *ipk*, user's show token Σ, message m and tag tag, this algorithm outputs a single bit b indicating accept (1) or reject (0).
- $sif/\perp \leftarrow$ ADAC.Extract$(ipk_1, ipk_2, tag_1, tag_2, m_1, m_2, \Sigma_1, \Sigma_2, \mathsf{PID})$: on input two issuer's public keys ipk_1, ipk_2, two tags tag_1, tag_2, two messages m_1, m_2, two tokens Σ_1, Σ_2, and a set of public identifiers $\mathsf{PID} = \{pid_1, ..., pid_n\}$, if both $(ipk_1, \Sigma_1, m_1, tag_1)$ and $(ipk_2, \Sigma_2, m_2, tag_2)$ are valid, this algorithm computes a value *sif'*. If it can derive a public

identifier of an existing user, i.e., $\mathsf{CompPid}(sif') = pid' \in \mathsf{PID}$, then it is the user's secret information $sif = sif'$ and output it; otherwise it outputs $\perp$.

5.3.2 Security Model

In this section, we briefly introduce the security properties at first. Then, we define the oracles and formal experiments.

Correctness. Honest-generated tokens can always be verified with respect to the issuer's public key, message, and tag.

Anonymity. If an honest user produces a valid token only once for a tag, no one can link it to any user's identifier or other token.

Unforgeability. Any malicious user cannot forge a valid token if the issuer has not issued a valid credential to him.

Extractability. Given multiple tokens on the same tag of the same user, everyone can extract the secret information from them.

Non-frameability. An honest user cannot be accused of having signed more than once. This means that no one can generate two tokens from which an honest user's secret information can be extracted, even with the help of issuers.

REMARK 6 (**Unforgeability**). *Unforgeability is inherently ensured by extractability and non-frameability, similar to traceable ring signatures [107]. The intuition is that if the scheme is non-frameable but not unforgeable, an adversary could create a forgery and use it to break non-frameability by framing an honest user. Conversely, if the forgery cannot frame an honest user, the scheme must not be extractable. Therefore, we do not explicitly prove unforgeability.*

Oracles. We define the following oracles to model the adversary's ability. There is an honest user table HU, a corrupted user table CU, an honest issuer table HI, a corrupted issuer table CI, and a queried tag-message table MT, a public bulletin board BB, an honest users' requests

table REQ, an honest users' partial credentials table PCred and an honest users' credentials table Cred which are empty.

- $\mathbf{AddUser}(u_i)$: Add a new user u_i to the system. If $u_i \notin$ BB, run the key generation algorithm $(upk^*, usk^*) \leftarrow$ ADAC.UserKeyGen, set $(upk_i, usk_i) = (upk^*, usk^*)$ and output upk_i. Add (u_i, upk_i) to BB and add (u_i, upk_i, usk_i) to HU. Otherwise, ignore it.
- $\mathbf{AddIssuer}(p_i)$: Add a new issuer p_i to the system. If $p_i \notin$ BB, run the key generation algorithm $(ipk^*, isk^*) \leftarrow$ ADAC.IssuerKeyGen, set $(ipk_i, isk_i) = (ipk^*, isk^*)$ and output ipk_i. Add (p_i, ipk_i) to BB and add (p_i, ipk_i, isk_i) to HI. Otherwise, ignore it.
- $\mathbf{CrptUser}(u_i)$: Corrupt an honest user in the system. If $u_i \in$ HU, output $usk_i, s_i, \mathit{sif}_i, cred_i$, delete (u_i, upk_i, usk_i) from HU and add (u_i, upk_i, usk_i) to CU. Otherwise, ignore it.
- $\mathbf{CrptIssuer}(p_i)$: Corrupt an honest issuer in the system. If $p_i \in$ HI, output isk_i, delete (p_i, ipk_i, isk_i) from HI and add (p_i, ipk_i, isk_i) to CI. Otherwise, ignore it.
- $\mathbf{Request}(u_i, p_j)$: Model an honest user u_i request to join the issuer p_j. It outputs a request req_{ij}. If $u_i \in$ HU, $p_j \in$ BB and $(u_i, p_j) \notin$ BB, it runs $\mathit{req}_{ij} \leftarrow$ ADAC.Request$(ipk_j, upk_i, pid_i, usk_i)$. Add $(u_i, p_j, \mathit{req}_{ij})$ to REQ. Otherwise, ignore it.
- $\mathbf{Issue}(u_i, p_j)$: Issue the credential for a user u_i from the issuer p_j. It checks that $u_i, p_j \in$ BB, $(u_i, p_j, \mathit{req}_{ij}) \in$ REQ and $(u_i, p_j) \notin$ BB. If yes, it outputs $\mathit{cred'}_{ij} \leftarrow$ ADAC.Issue$(ipk_j, upk_i, pid_i, req_{ij}, isk_j)$, adds (u_i, p_j) to BB, deletes $(u_i, p_j, \mathit{req}_{ij})$ from REQ and adds the partial credential $(u_i, p_j, \mathit{cred'}_{ij})$ to PCred. Otherwise, ignore it.
- $\mathbf{Receive}(u_i, ipk_j)$: Model an honest user u_i obtains a valid credential issued by the issuer j. It checks that $u_i \in$ HU, $p_j \in$ BB, $(u_i, p_j, \mathit{cred'}_{ij}) \in$ PCred. If yes, it outputs $\mathit{cred}_{ij} \leftarrow$ ADAC.Receive $(ipk_j, req_{ij}, \mathit{cred'}_{ij}, usk_i)$, deletes $(u_i, p_j, \mathit{cred'}_{ij})$ from PCred, and adds $(u_i, p_j, \mathit{cred}_{ij})$ to Cred. Otherwise, ignore it.
- $\mathbf{Show}(u_i, p_j, m, tag)$: Model an honest user shows a token on a message-tag pair using his credential. It checks that $u_i \in$ HU, $(u_i, p_j) \in$ BB and $(u_i, p_j, \mathit{cred}_{ij}) \in$ Cred. If yes, run $\Sigma \leftarrow$ ADAC.Show$(ipk_j, m, tag, usk_i, cred_{ij})$ and add (u_i, m, tag) to MT, output Σ.

Anonymity. This property ensures that honest users' identities are hidden from the public and issuers. Formally speaking, given any two valid tokens Σ_0, Σ_1 from different users with the same issuer on the same message m and tag tag, nobody can distinguish them except the

users themselves. The experiment $\mathrm{Exp}^{\mathsf{anony}}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is formalized as follows.

- $\mathcal{A}$ receives pp and has access to the oracles: $\mathbf{AddUser}$, $\mathbf{AddIssuer}$, $\mathbf{CrptUser}$, $\mathbf{CrptIssuer}$, $\mathbf{Request}$, $\mathbf{Receive}$, $\mathbf{Show}$. It can add users and issuers to the system, corrupt users and issuers and get their secret keys, invoke users to join some issuers and get their requests, query tokens of any users on any messages and tags.
- $\mathcal{A}$ chooses two challenge users with $upk_0, pid_0, upk_1, pid_1$, an issuer with ipk, the challenge message m, tag tag. It sends $(m, tag, ipk, upk_0, pid_0, upk_1, pid_1)$ to $\mathcal{C}$.
- $\mathcal{C}$ checks the following conditions and aborts if any of them are violated; otherwise, continue.
  (1) ipk has been added as a valid issuer: $ipk \in \mathsf{HI} \cup \mathsf{CI}$;
  (2) upk_0, upk_1 have registered in ipk and $upk_0, upk_1 \in \mathsf{HU}$;
  (3) $\mathcal{A}$ has never queried tokens on the tag tag for upk_0, upk_1.
- $\mathcal{C}$ randomly chooses one bit $b \leftarrow \{0,1\}$ and sends $\mathcal{A}$ a token Σ_b by running $\Sigma_b \leftarrow \mathsf{ADAC.Show}(ipk, m, tag, usk_b, cred_b)$.
- $\mathcal{A}$ outputs a guess b'.
- Outputs 1 if $b' == b$, otherwise, outputs 0.

DEFINITION 13. *The abuse deterring anonymous credential is anonymous if any PPT adversary in the* $\mathrm{Exp}^{\mathsf{anony}}$ *can only guess the bit correctly with probability negligibly close to* $1/2$*, i.e.,*
$|\Pr[\mathrm{Exp}^{\mathsf{anony}}(\mathcal{A}, \lambda) = 1] - 1/2| \leq \mathit{negl}(\lambda)$

Extractability. This property ensures that any adversary cannot generate two tokens on the same tag from the same user, but his secret information cannot be extracted from them. The extractability experiment $\mathrm{Exp}^{\mathsf{extract}}$ is formalized as follows.

- $\mathcal{A}$ receives pp and has access to the oracles: $\mathbf{AddUser}$, $\mathbf{AddIssuer}$, $\mathbf{CrptUser}$, $\mathbf{CrptIssuer}$, $\mathbf{Issue}$, $\mathbf{Show}$. It can add users and issuers to the system, corrupt users and issuers to get their secret keys, issue credentials to users on behalf of honest issuers, and it can query tokens of any users on any messages and tags.

- $\mathcal{A}$ wins if it gives $n+1$ tokens $\Sigma_1, ...\Sigma_{n+1}$, messages $m_1, ..., m_{n+1}$ and issuer public keys $ipk_1, ..., ipk_{n+1}$ and one tag tag^* such that:
  (1) There are ℓ users $u_1, ..., u_\ell$ in the system with secret information $sif_1, ..., sif_\ell$ and public identifiers $pid_1, ..., pid_\ell$, and $\mathcal{A}$ has corrupted at most n users of them;
  (2) for each $i \in [n+1]$, ipk_i corresponds to an honest issuer, $ipk_i \in \mathsf{HI}$ and some of them can be the same issuer;
  (3) for each $i \in [n+1]$, $1 \leftarrow \mathsf{ADAC.Verify}(ipk_i, m_i, tag^*, \Sigma_i)$ where all messages are different;
  (4) $\mathcal{A}$ has never queried token showing on tag^*: $tag^* \notin \mathsf{MT}$;
  (5) for $\forall i, j \in [n+1]$, $i \neq j$, $\perp \leftarrow \mathsf{ADAC.Extract}(ipk_i, ipk_j, m_i, m_j, tag^*, \Sigma_i, \Sigma_j, \mathsf{PID})$, where $\mathsf{PID} = \{pid_1, ..., pid_\ell\}$.
- Outputs 1 if $\mathcal{A}$ wins, otherwise, outputs 0.

DEFINITION 14. *The abuse deterring anonymous credential is extractable if any PPT adversary in the* $\mathrm{Exp}^{\mathsf{extract}}$ *can win with negligible probability, i.e.,* $|\Pr[\mathrm{Exp}^{\mathsf{extract}}(\mathcal{A}, \lambda) = 1]| \leq$ *negl(λ).*

Non-frameability. This property ensures that any adversary cannot frame an honest user by generating two valid tokens (no need for the same tag), such that an honest user's secret information can be extracted from them. The non-frameability experiment $\mathrm{Exp}^{\mathsf{nf}}$ is formalized as follows.

- $\mathcal{A}$ receives pp and has access to the oracles: $\mathbf{AddUser}$, $\mathbf{AddIssuer}$, $\mathbf{CrptUser}$, $\mathbf{CrptIssuer}$, $\mathbf{Request}$, $\mathbf{Receive}$, $\mathbf{Show}$. It can add users and issuers to the system, corrupt users and issuers and get their secret keys, invoke users to join honest issuers, and it can query tokens of any users on any messages and tags.
- $\mathcal{A}$ wins if gives $(ipk_1, m_1, tag_1, \Sigma_1)$, $(ipk_2, m_2, tag_2, \Sigma_2)$ such that:
  (1) There are ℓ users $u_1, ..., u_\ell$ in the system with public identifiers $pid_1, ..., pid_\ell$, and there is at least one honest user, $\mathsf{HU} \neq \emptyset$
  (2) for $i \in \{1, 2\}$, $1 \leftarrow \mathsf{ADAC.Verify}(ipk_i, m_i, tag_i, \Sigma_i)$;
  (3) *sif'* $\leftarrow \mathsf{ADAC.Extract}(ipk_1, ipk_2, m_1, m_2, tag_1, tag_2, \Sigma_1, \Sigma_2, \mathsf{PID})$, such that $\mathsf{CompPid}($*sif'*$) \in$

$\mathsf{PID} = \{pid_1, ..., pid_\ell\}$ which in HU;

(4) at least one of Σ_1, Σ_2 was not generated by querying oracles.

- Outputs 1 if $\mathcal{A}$ wins, otherwise, outputs 0.

DEFINITION 15. *The abuse deterring anonymous credential is non-frameable if any PPT adversary in the* $\mathrm{Exp}^{\mathsf{nf}}$ *can win with only negligible probability, i.e.,* $|\Pr[\mathrm{Exp}^{\mathsf{nf}}(\mathcal{A}, \lambda) = 1]| \leq$*negl(*$\lambda$*).*

5.3.3 Construction from PS Signatures

We give a concrete ADAC construction based on PS signature [41].

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter 1^λ, it produces the public parameters $pp = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e(\cdot,\cdot), H(\cdot), g, \tilde{g}, \tilde{h}, \mathsf{BB}\}$. Here $\mathbb{G}_1, \mathbb{G}_2$ are asymmetric groups, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is the Type III bilinear pairing operation, $H_1 : \{0,1\}^* \to \mathbb{Z}_p$, $H_2 : \{0,1\}^* \to \mathbb{G}_2$ are hash functions. Choose random group elements $g \leftarrow\!\!\$\ \mathbb{G}_1, \tilde{g}, \tilde{h} \leftarrow\!\!\$\ \mathbb{G}_2$, BB is a public bulletin board. For simplicity, pp will implicitly be an input of all the following algorithms.
- $(ipk, isk) \leftarrow \mathsf{ADAC.IssuerKeyGen}(pp)$: randomly choose $x, y_1, y_2 \leftarrow\!\!\$\ \mathbb{Z}_p$, generate the issuer's secret key ***isk***$= X = g^x$ and public key ***ipk***$=(Y_1 = g^{y_1}, Y_2 = g^{y_2}, \widetilde{X} = \tilde{g}^x, \widetilde{Y_1} = \tilde{g}^{y_1}, \widetilde{Y_2} = \tilde{g}^{y_2})$.
- $(usk, upk, s, sif, pid) \leftarrow \mathsf{ADAC.UserKeyGen}(pp)$: choose personal user secret key $v \leftarrow\!\!\$\ \mathbb{Z}_p$, set personal user public key $V \leftarrow g^v$, choose secret seed $s \leftarrow\!\!\$\ \mathbb{Z}_p$, set secret information ***sif*** $= \tilde{g}^s$ and public identifier ***pid*** $= e(g, \tilde{g}^s)$. Set $usk = (v, s, sif = \tilde{g}^s)$, $upk = (V = g^v, pid = e(g, \tilde{g})^s)$. ***upk*** is recorded on the bulletin board, which ensures every user binds with a single identifier and can only get one credential from one issuer.
- *req* $\leftarrow \mathsf{ADAC.Request}(ipk, usk, upk)$: on input issuer's public key ***ipk*** $= (Y_1, Y_2, \widetilde{X}, \widetilde{Y_1}, \widetilde{Y_2})$ and user's public key $upk = (V, pid)$, secret key $usk = (v, s,$ ***sif***$)$, user does the following:
(1) choose blind factor $t_1 \leftarrow\!\!\$\ \mathbb{Z}_p$, compute $Q = g^{t_1} Y_1^s Y_2^v$;
(2) generate the zero-knowledge proof π_{req} for the relation (can be achieved by Schnorr's

protocol [89]):

$$\mathcal{R}_{\mathsf{req}} = \{(ipk, Q, V, pid; s, v, t_1) : Q = g^{t_1} Y_1^s Y_2^v \wedge V = g^v \\ \wedge pid = e(g, \tilde{g})^s \wedge e(Q, \tilde{h}) = e(g, \tilde{h})^{t_1} e(Y_1, \tilde{h})^s e(Y_2, \tilde{h})^v\};$$

(3) output $req = (upk, Q, \pi_{\mathsf{req}})$.

- *cred'*$/\perp \leftarrow \mathsf{ADAC.Issue}(ipk, isk, req)$: on input issuer's public key *ipk*, issuer's secret key $isk = X = g^x$ and user's credential request *req*, the issuer does the following:

  (1) parse *req* $= (upk = (V, pid), Q, \pi_{\mathsf{req}})$, check that

  - if upk is recorded on the bulletin board BB with another valid request proof π'_{req}, then continue;
  - V has never appeared, then continue;
  - V has been recorded with another $pid' \neq pid$, output $\perp$;

  (2) verify π_{req} on (ipk, Q, upk), if it is invalid, output $\perp$; otherwise, continue, and if V has never appeared, post this req on BB;

  (3) choose $r_1 \leftarrow\$ \mathbb{Z}_p$, compute a partial credential

  cred' $\leftarrow (g^{r_1}, (XQ)^{r_1}) = (g^{r_1}, X^{r_1} Y_1^{s \cdot r_1} Y_2^{v \cdot r_1} g^{t_1 \cdot r_1})$

- *cred*$/\perp \leftarrow \mathsf{ADAC.Receive}(ipk, usk, req, cred', t_1)$: on input issuer's public key *ipk*, user's secret key *usk*, credential request *req*, partial credential *cred'*, blind factor t_1, the user does the following:

  (1) parse *cred'* $= (\sigma'_1, \sigma'_2)$;

  (2) unblind: *cred* $\leftarrow (\sigma_1, \sigma_2) = (\sigma'_1, \sigma'_2 / {\sigma'_1}^{t_1}) = (g^{r_1}, X^{r_1} Y_1^{s \cdot r_1} Y_2^{v \cdot r_1})$;

  (3) if $e(\sigma_1, \widetilde{X}_1 \widetilde{Y}_1^s \widetilde{Y}_2^v) = e(\sigma_2, \tilde{g})$, output *cred*; otherwise, output $\perp$. The credential is a valid PS signature on (s, v).

- $\Sigma \leftarrow \mathsf{ADAC.Show}(ipk, usk, cred, m, tag)$: on input the issuer's public key *ipk*, user's secret key usk, credential *cred*, message m, and tag tag, the user does the following:

  (1) parse *cred* $= (\sigma_1, \sigma_2)$;

  (2) $r_2, r_3 \leftarrow\$ \mathbb{Z}_p$, compute $\sigma^* = (\sigma_1^*, \sigma_2^*) = (\sigma_1^{r_2}, (\sigma_2 \cdot \sigma_1^{r_3})^{r_2})$;

  (3) compute $c = H_1(tag, m, ipk, \sigma^*)$, $T = (\tilde{g}^c H_2(tag))^s$;

(4) generate the zero-knowledge proof π_{show} for the relation:

$$\mathcal{R}_{\mathsf{show}} = \{(ipk, m, tag, \sigma^*, c, T; s, v, r_3) :$$
$$e(\sigma_1^*, \widetilde{X}) \cdot e(\sigma_1^*, \widetilde{Y}_1)^s \cdot e(\sigma_1^*, \widetilde{Y}_2)^v \cdot e(\sigma_1^*, \tilde{g})^{r_3} = e(\sigma_2^*, \tilde{g}) \wedge$$
$$c = H_1(tag, m, ipk, \sigma^*) \wedge e(g, T) = e(g, \tilde{g}^c H_2(tag))^s\};$$

(5) output a token $\Sigma = (\sigma^*, c, T, \pi_{\mathsf{show}})$.

- $0/1 \leftarrow \mathsf{ADAC.Verify}(ipk, tag, m, \Sigma)$: on input the issuer's public key *ipk*, message m, tag tag and showing token Σ, the verifier does the following:
  (1) parse $\Sigma = (\sigma^*, c, T, \pi_{\mathsf{show}})$, compute $c' = H_1(tag, m, ipk, \sigma^*)$, check whether $c' = c$;
  (2) verify the proof π_{show} using $(ipk, m, tag, \sigma^*, c, T)$;
  (3) output 0 if any verification fails; otherwise, output 1.
- *sif*/$\perp \leftarrow \mathsf{ADAC.Extract}(ipk_1, ipk_2, tag_1, tag_2, m_1, m_2, \Sigma_1, \Sigma_2, \mathsf{PID})$: given the tags tag_1, tag_2, messages m_1, m_2, issuer public keys $\textit{ipk}_1, \textit{ipk}_2$, and tokens Σ_1, Σ_2, a set of public identifiers $\mathsf{PID} = \{pid_1, ..., pid_n\}$, anyone can do the following if $tag_1 = tag_2$:
  (1) run $\mathsf{ADAC.Verify}$ on $(ipk_1, tag_1, m_1, \Sigma_1)$ and $(ipk_2, tag_2, m_2, \Sigma_2)$;
  (2) if any verification does not pass, output $\perp$; otherwise, parse $\Sigma_1 = (\sigma_1^*, c_1, T_1, \pi_1)$, $\Sigma_2 = (\sigma_2^*, c_2, T_2, \pi_2)$, where $c_1 \neq c_2$;
  (3) compute *sif'* $= (T_1/T_2)^{\frac{1}{c_1 - c_2}}$;
  (4) if $\exists$ *pid'* $\in \mathsf{PID}$ such that *pid'* $= \mathsf{CompPid}($*sif'*$) = e(g,$ *pid'*$)$, output *sif'*; otherwise, output $\perp$.

□

THEOREM 7. *Assuming the PS signature is unforgeable, the SXDH, DL, and DPair assumptions hold, and the underlying proof system is a zero-knowledge argument of knowledge, then our ADAC construction is correct, anonymous, extractable, and non-frameable.*

PROOF. *Correctness.* The correctness is straightforward from the correctness of PS signature and the perfect completeness of the underlying proof system.

Anonymity. In the anonymity experiment, the challenge message and tag are m^*, tag^*, and the challenge identities upk_0, upk_1 are honest registered users in the valid issuer ipk. $\mathcal{A}$ can only

contain their credential requests and tokens on other tags $\neq tag^*$. Due to the message privacy, it cannot learn any information from the requests.

We design the following hybrid games:

- $\mathsf{Game}_{\mathsf{real}}$: This game is the same as the experiment. The challenger $\mathcal{C}$ chooses $b \leftarrow\!\!\$ \{0,1\}$ and generates a token Σ_b with m, tag by running $\Sigma_b \leftarrow \mathsf{ADAC.Show}(ipk, usk_b, cred_b, m, tag)$, where $\Sigma_b = (\sigma_b^*, c_b, T_b, \pi_{\mathsf{show}}^b)$.
- G_1: This game is similar to $\mathsf{Game}_{\mathsf{real}}$ except that in the token, π_{show}^b is simulated without witness.
- Game_2: This game is similar to Game_1 except that in the token, σ_b^* consist of two random group elements in $\mathbb{G}_1$.
- Game_3: This game is similar to Game_2 except that in the token, T is a random group element in $\mathbb{G}_2$. Thus, in this game, the token is independent of b.

Compare Game_1 with $\mathsf{Game}_{\mathsf{real}}$, the only difference is that the proof is simulated. Since this proof is zero-knowledge, the probability of distinguishing Game_1 from $\mathsf{Game}_{\mathsf{real}}$ is negligible. We have that $|\Pr[\mathsf{Game}_{\mathrm{real}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{G}_1(\mathcal{A}, \lambda) = 1]| \leq negl(\lambda)$.

Compare Game_2 with Game_1, in Game_2, the random group element pair σ^* is correctly distributed because r_2 and r_3 were randomly generated in the step (2) of $\mathsf{ADAC.Show}$. Thus, they have the same distribution. We have that $|\Pr[\mathsf{Game}_1(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Game}_2(\mathcal{A}, \lambda) = 1]| = 0$.

Compare Game_3 with Game_2, in Game_3, T is a random group element in $\mathbb{G}_2$. Since H is modeled as a random oracle and the SXDH assumption, they can be distinguished with only negligible probability. We have that $|\Pr[\mathsf{Game}_2(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Game}_3(\mathcal{A}, \lambda) = 1]| \leq negl(\lambda)$.

In Game_3, $\mathcal{A}$'s view is independent of b. Thus, $\mathcal{A}$ just outputs a random guess b' in G_3, so its advantage is 0: $\Pr[\mathsf{Game}_3(\mathcal{A},\lambda)=1]-1/2=0$. In summary, we have

$$\begin{aligned}
&|\Pr[\mathrm{Exp}^{\mathsf{anony}}(\mathcal{A},\lambda)=1]-1/2| = |\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A},\lambda)=1]-1/2| \\
\leq &|\Pr[\mathsf{Game}_{\mathrm{real}}(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Game}_1(\mathcal{A},\lambda)=1]| \\
&+|\Pr[\mathsf{Game}_1(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Game}_2(\mathcal{A},\lambda)=1]| \\
&+|\Pr[\mathsf{Game}_2(\mathcal{A},\lambda)=1]-\Pr[\mathsf{Game}_3(\mathcal{A},\lambda)=1]| \\
&+|\Pr[\mathsf{Game}_3(\mathcal{A},\lambda)=1]-1/2| \leq negl(\lambda)
\end{aligned}$$

Extractability. We assume that the proof of possession has been done to prove the knowledge of the secret key for each user's public key.

If $\mathcal{A}$ wins, which means it outputs $n+1$ tokens $\Sigma_1,...\Sigma_{n+1}$, messages $m_1,...,m_{n+1}$ and issuer public keys $ipk_1,...,ipk_{n+1}$ and one tag tag^* where $1 \leftarrow \mathsf{ADAC.Verify}(ipk_i, m_i, tag^*, \Sigma_i)$, $\Sigma_i = (\sigma_i^* = (\sigma_{i1}^*, \sigma_{i2}^*), c_i, T_i, \pi_{\mathsf{show}}^i)$.

Due to the knowledge soundness of Π_{show}, there exists an extractor $\mathcal{E}$ who can extract (s_i, v_i, r_i), such that $T_i = (\tilde{g}^{c_i} H(tag^*))^{s_i}$ and compute $\sigma_i = (\sigma_{i1}, \sigma_{i2}) \leftarrow (\sigma_{i1}^*, \sigma_{i2}^* \cdot (\sigma_{i1}^*)^{-r_i})$ which is a valid signature on (s_i, v_i).

Since for $\forall i,j \in [n+1]$, $i \neq j$, $\perp \leftarrow \mathsf{ADAC.Extract}(ipk_i, ipk_j, m_i, m_j, tag^*, \Sigma_i, \Sigma_j, \mathsf{PID})$, where $\mathsf{PID} = \{pid_1,...,pid_\ell\}$. However, $\mathcal{A}$ corrupts at most n users. It means there exists at least one $\bar{s}$ in $\{s_i\}_{i\in[n+1]}$ such that $e(g,\tilde{g}^{\bar{s}}) \notin \{pid_1,...,pid_\ell\}$ and $\bar{\sigma}$ is a valid signature on $(\bar{s},\bar{v})$ and the issuers are honest.

This means a valid PS signature on $(\bar{s},\bar{v})$ can be computed from these valid tokens. It was not generated by the oracle queries, and the issuer has not been corrupted. It contradicts the unforgeability of PS signatures, which only happens with negligible probability. So, $\mathcal{A}$ wins with only negligible probability.

Non-frameability. We assume that the proof of possession has been done to prove the knowledge of the secret key for each user's public key.

Suppose $\mathcal{A}$ wins and outputs two valid pairs $(ipk_1, m_1, tag_1, \Sigma_1)$, $(ipk_2, m_2, tag_2, \Sigma_2)$, and *sif'* $\leftarrow \mathsf{ADAC.Extract}(ipk_1, ipk_2, m_1, m_2, tag_1, tag_2, \Sigma_1, \Sigma_2, \mathsf{PID})$, such that $\mathsf{CompPid}($*sif'*$) \in \mathsf{PID} = \{pid_1, ..., pid_\ell\}$ which is an honest user in HU and *sif'* $= \tilde{g}^{s'}$.

Due to the knowledge soundness of Π_{show}, there exists an extractor $\mathcal{E}$ who can extract the secret seeds $s_1, s_2 \neq 0$, such that $T_i = (\tilde{g}^{c_i} H(tag_i))^{s_i}$ for $i = 1, 2$ and at least one of s_1, s_2 does not equal to s'. We have

$$(T_1/T_2)^{\frac{1}{c_1 - c_2}} = sif' = \tilde{g}^{s'}$$

$$\tilde{g}^{c_1 s_1 - c_2 s_2} H(tag_1)^{s_1} H(tag_2)^{-s_2} = \tilde{g}^{s'(c_1 - c_2)}$$

$$\tilde{g}^{c_1 s_1 - c_2 s_2 - s'(c_1 - c_2)} H(tag_1)^{s_1} H(tag_2)^{-s_2} = 1_{\mathbb{G}_2}$$

$H(tag_1), H(tag_2)$ are random group elements in $\mathbb{G}_2$ which are outputs of random oracles.

Since $\mathcal{A}$ does not know s', $c_1 s_1 - c_2 s_2 - s'(c_1 - c_2) = 0$ holds with only negligible probability. Otherwise, it breaks the DL assumption in $\mathbb{G}_2$, which only happens also with negligible probability. Therefore, $\mathcal{A}$ wins with only negligible probability.

□

5.4 Fair Relay System for Anonymous Zether

This section presents the fair gas relay system model for anonymous Zether. Compared with the basic relay system, we additionally formalize the anonymity and fairness properties that ensure honest relayers always get paid, honest users remain anonymous and never lose money. We show a concrete scheme by integrating the ADAC with a collateral smart contract.

At a high level, a user registers with a relayer by submitting collateral and encoded secret information to a smart contract deployed by the relayer. As part of the registration process, the user receives a valid ADAC credential issued by the relayer. This credential encodes the same secret information and serves as the user's registration credential. To send a meta-transaction, the user generates a token using their credential, setting the meta-transaction as the message

and its nonce as the tag. If a malicious user causes honest relayers to lose gas fees, multiple ADAC tokens with the same tag will appear on-chain. In such cases, the secret information encoded in the tokens can be extracted and used by the relayers to withdraw the malicious user's collateral as reimbursement.

5.4.1 Model

Our relay system includes the following parties and smart contracts:

Main smart contract: It is the anonymous Zether smart contract that has been deployed on the public blockchain. It maintains a global state acc and supports anonymous payment between users.

We use $\Pi_{\mathsf{AZ}} = (\mathsf{AZ.Setup}, \mathsf{AZ.KeyGen}, \mathsf{AZ.Join}, \mathsf{AZ.Read}, \mathsf{AZ.Trans}, \mathsf{AZ.Vrfy})$ to denote the main functions of this smart contract. The user joins it by running AZ.KeyGen to produce his Zether secret and public keys (zsk, zpk) and runs AZ.Join. The user invokes $\mathsf{AZ.Trans}(\mathsf{acc}, zsk)$ to generate the meta-transaction meta, which can be checked by calling $\mathsf{AZ.Vrfy}(\mathsf{acc}, \mathsf{meta})$. The user's account balance can be revealed by $bal \leftarrow \mathsf{AZ.Read}(\mathsf{acc}, zsk)$.

User: He has joined the main smart contract and generates the anonymous meta-transaction.

Relayer: It is the entity that broadcasts the meta-transaction on behalf of the user. On receiving the meta-transaction and some other data from the user, the relayer verifies them under the current state of the blockchain. Then, he broadcasts the transaction[4] and pays the gas fee for the user.

Collateral smart contract: It is a smart contract deployed by the relayer. Users deposit ETH in it as collateral. Each relayer binds with one collateral smart contract and can get the user's collateral if and only if his misbehavior is detected. It maintains an identifier set $\mathcal{U}$ containing the identifiers of all existing registered users.

[4]The meta-transaction is stored in the data field of the transaction, as introduced in Section 2.4

We use $\Pi_{\mathsf{Colla}} = (\mathsf{Colla.Setup}, \mathsf{Colla.Join}, \mathsf{Colla.Judge})$ to denote the main functions of this smart contract. The user deposits collateral into the smart contract by invoking $\mathsf{Colla.Join}(zsk, s, sif, pid)$. His collateral will be sent to the relayer if his secret information sif is revealed and $\mathsf{Colla.Judge}(sif, pid)$ is called, and outputs 1.

The relay system consists of the following phases and algorithms:

- **Setup and key generation**: The public blockchain has been set up and is secure. The main smart contract is deployed on the blockchain. The relay system public parameter is also generated as *rpp*. For simplicity, *rpp* will implicitly be an input of all the following algorithms.

  The user runs the $\mathsf{UserKeyGen}$ algorithm and gets his personal secret and public keys usk, upk, where usk contains a secret seed s, secret information *sif*, and upk contains a public identifier pid.

  The relayer runs the $\mathsf{RelayerKeyGen}$ algorithm to get his secret key *rsk* and public key *rpk*. He publishes rpk and posts a collateral smart contract on the chain, embedding his account address.
- **Registration**: The user interacts with a relayer to become a registered member so that he can enjoy the anonymous relay service.

  The user deposits collateral locked in the relayer's collateral smart contract. He runs $\mathsf{Register}(rpk, upk, usk)$ to interact with the relayer running $\mathsf{RegRcv}(upk, rsk)$.

  The relayer checks that the collateral has been deposited and that the user's public identifier is the same as his public identifier in other relayers' collateral smart contracts, if any. Afterward, the user obtains a registration credential *cred*.
- **Broadcast**: The user generates the meta-transaction meta for the current epoch $epoch$ and requests the relayer to broadcast it in a full transaction. With the meta-transaction meta and registration credential $cred$, the user runs $\mathsf{BrdReq}(\textit{epoch}, \textit{rpk}, \textit{usk}, \textit{cred}, \mathsf{meta})$ and outputs the full transaction request *fullreq*. The relayer running $\mathsf{BrdVrfy}(\textit{epoch}, rpk, \mathsf{acc}, \textit{fullreq})$. If it is verified, the relayer outputs the full transaction tx that includes meta and broadcasts it to the chain. Otherwise, output $\perp$.

- **Reimburse**: On two full transactions tx_1, tx_2 broadcasted by different relayers, if the meta-transactions were generated by the same user for the same $epoch$ such that at least one of them fails, the relayers run the Reimburse($epoch, rpk_1, rpk_2, tx_1, tx_2, \mathcal{U}$) algorithm and get paid by the collateral smart contracts.

5.4.1.1 Threat Model and Security Requirements.

We assume the main smart contract and collateral smart contracts are secure. Most participants are random blockchain nodes and can be controlled by a computationally-bounded adversary. Users and relayers mutually distrust each other.

We assume the adversary is *rational*. If cheating is not profitable or the victims' loss does not exceed the cheating cost, they will be discouraged from cheating. Misbehaved users, as reported by the relayers, will be penalized financially from their collateral as enforced by the collateral smart contract. Under such a threat model, we expect the following computational security guarantees.

Anonymity. The honest users stay anonymous to the relayers and the public. Given two full transactions containing two honest users' anonymous meta-transactions from the same relayer in the same epoch, no one can distinguish them with non-negligible advantage.

Overdraft prevention. Any user cannot spend more money than he owns, even if it colludes with relayers.

Gas fee fairness. No adversary can cheat *two or more* honest relayers simultaneously without punishment. These relayers will be paid within an anonymous transaction or by the collateral in the smart contracts. Any honest user who only requests one relayer does not lose money. If his anonymous payment executes, he only pays the preset fee. Otherwise, he pays nothing.

REMARK 7 (**Rational adversary for fairness**). *Gas fee fairness experiment defines that an adversary wins only if it cheats at least two honest relayers simultaneously. Attacks targeting a single relayer are excluded under the assumption of a rational adversary. If the adversary targets only one honest relayer, it also incurs a cost of at least one gas fee, equivalent to*

the relayer's loss. This results in no net advantage for the adversary, making such an attack inconsistent with the behavior of a rational adversary.

5.4.1.2 Formal Definitions

We now formally define the security model of the relay system.

Oracles. We define the following oracles to model the adversary's ability. There is an honest user table HU, a corrupted user table CU, an honest relayer table HR, a corrupted relayer table CR, a queried tag-message table MT, a bulletin board BB, which are empty.

- $\mathbf{AddUser}(u_i)$: Add a new user u_i to the system. If $u_i \notin \mathsf{BB}$, run the key generation algorithm $(upk^*, usk^*, s^*, sif^*, pid^*) \leftarrow \mathsf{UserKeyGen}$, set $(upk_i, usk_i, s_i, sif_i, pid_i) = (upk^*, usk^*, s^*, sif^*, pid^*)$ and output upk_i, pid_i. Add (u_i, upk_i, pid_i) to BB and add $(u_i, upk_i, pid_i, usk_i, s_i, sif_i)$ to HU. Otherwise, ignore it.
- $\mathbf{AddRelayer}(R_i)$: Add a new relayer R_i to the system. If $R_i \notin \mathsf{BB}$, run the key generation algorithm $(rpk^*, rsk^*) \leftarrow \mathsf{RelayerKeyGen}$, set $(rpk_i, rsk_i) = (rpk^*, rsk^*)$ and output rpk_i. Add (R_i, rpk_i) to BB and add (R_i, rpk_i, rsk_i) to HR. Add the collateral smart contract to BB. Otherwise, ignore it.
- $\mathbf{CrptUser}(u_i)$: Corrupt an honest user in the system. If $u_i \in \mathsf{HU}$, output $usk_i, s_i, sif_i, cred_i$, delete $(u_i, upk_i, pid_i, usk_i, s_i, sif_i)$ from HU and add it to CU.
- $\mathbf{CrptRelayer}(R_i)$: Corrupt an honest relayer in the system. If $R_i \in \mathsf{HR}$, output rsk_i, delete (R_i, rpk_i, rsk_i) from HR and add (R_i, rpk_i, rsk_i) to CR.
- $\mathbf{Register}(u_i, R_j)$: It models that an honest user u_i requests to join the relayer R_j. It runs $\mathsf{Register}(rpk_j, upk_i, pid_i, s_i, usk_i, sif_i)$ and outputs a credential $cred_{ij}$.
- $\mathbf{RegRcv}(u_i, pid_i, R_j)$: It checks that $u_i \in \mathsf{HU} \cup \mathsf{CU}$, $R_j \in \mathsf{HR} \cup \mathsf{CR}$, but $(u_i, R_j) \notin \mathsf{BB}$. If yes, it runs $\mathsf{RegRcv}(rpk_j, upk_i, pid_i, rsk_j)$ and adds (u_i, pid_i, R_j) to BB. Otherwise, ignore it.
- $\mathbf{MetaTxGen}(u_i, R_j, epoch, k)$: If $u_i \notin \mathsf{BB}$ or $R_j \notin \mathsf{BB}$, ignore it. If $u_i \in \mathsf{HU} \wedge (u_i, epoch) \in \mathsf{MT}$, ignore it. Otherwise, run $\mathsf{AZ.Trans}(usk_i, \mathsf{acc})$ on the current epoch $epoch$ and the gas fee amount k paid to R_j, output the meta-transaction meta. Add $(u_i, R_j, \mathsf{meta}, epoch)$ to MT.

- $\mathbf{BrdReq}(u_i, R_j, \mathsf{meta})$: If $(u_i, pid_i, R_j) \notin \mathsf{BB}$, ignore it. Otherwise, run $\mathsf{BrdReq}(\mathit{sif}_i, rpk_j, \mathsf{meta})$ and add $(u_i, R_j, \mathsf{meta})$ to MT.
- $\mathbf{FullTxGen}(u_i, R_j, epoch, k_i, k_j)$: If $(u_i, pid_i, R_j) \notin \mathsf{BB}$, ignore it. If $u_i \in \mathsf{HU} \wedge (u_i, epoch) \in \mathsf{MT}$, ignore it. Otherwise, run $\mathsf{meta} \leftarrow \mathsf{AZ.Trans}(usk_i, \mathsf{acc})$, add $(u_i, R_j, \mathsf{meta}, epoch)$ to MT and run $tx \leftarrow \mathsf{FullTxGen}(rsk_j, \mathsf{meta}, k_j)$, output the full transaction tx.

Anonymity. This property ensures that honest users' identities are hidden from the public and relayers. The experiment $\mathrm{Exp}^{\mathsf{anony}}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is formalized as follows.

- $\mathcal{A}$ receives rpp and has access to the oracles: $\mathbf{AddUser}$, $\mathbf{AddRelayer}$, $\mathbf{CrptUser}$, $\mathbf{CrptRelayer}$, $\mathbf{Register}$, $\mathbf{RegRcv}$, $\mathbf{MetaTxGen}$, $\mathbf{BrdReq}$. It can add new users and relayers to the system, corrupt users and relayers to get their secret keys, invoke users to join some relayers, invoke users to generate meta transactions, and it can query the user's broadcast request.
- $\mathcal{A}$ chooses two users u_0, u_1, a relayer R^*, the gas fee k, and epoch $epoch^*$. It sends $(epoch^*, u_0, u_1, R^*, k)$ to $\mathcal{C}$.
- $\mathcal{C}$ checks the following conditions and aborts if any of them are violated; otherwise, continue.
  (1) R^* has been added as a valid relayer;
  (2) u_0, u_1 have registered in R^* and are not corrupted;
  (3) $\mathcal{A}$ has never queried meta transaction for u_0, u_1 on $epoch^*$.
  (4) $\mathbf{MetaTxGen}(u_0, R^*, epoch^*, k)$ and $\mathbf{MetaTxGen}(u_1, R^*, epoch^*, k)$ can be invoked successfully.
- $\mathcal{C}$ randomly chooses one bit $b \leftarrow \{0, 1\}$, generates $meta_b \leftarrow \mathbf{MetaTxGen}(u_b, R^*, k)$ and $fullreq_b \leftarrow \mathbf{BrdReq}(u_b, R^*, meta_b)$, sends $\mathcal{A}$ the full transaction request $fullreq_b$.
- $\mathcal{A}$ outputs a guess b'.
- Outputs 1 if $b' == b$, otherwise, outputs 0.

DEFINITION 16. *The relay system is anonymous if any PPT adversary in the* $\mathrm{Exp}^{\mathsf{anony}}$ *can only guess* b *correctly with probability negligibly close to* $1/2$*, i.e.,* $|\Pr[\mathrm{Exp}^{\mathsf{anony}}(\mathcal{A}, \lambda) = 1] - 1/2| \leq$*negl(*$\lambda$*).*

Overdraft prevention. Any user cannot spend more money than he owns in the anonymous payment, even if it colludes with relayers. The extractability experiment $\mathrm{Exp}^{\mathsf{op}}$ is formalized as follows. It is similar to the overdraft-safety experiment defined in [17] except that the adversary can corrupt the relayers.

- $\mathcal{A}$ receives rpp and has access to the oracles: $\mathbf{AddUser}$, $\mathbf{AddRelayer}$, $\mathbf{CrptUser}$, $\mathbf{CrptRelayer}$, $\mathbf{Register}$, $\mathbf{MetaTxGen}$, $\mathbf{BrdReq}$.
- $\mathcal{A}$ outputs a full transaction tx^*. It wins if tx^* is valid and any of the following conditions holds:
  (1) $\exists$ honest user $u_i \in \mathsf{HU}$ and his account balance $bal_i \leftarrow \mathsf{AZ.Read}(\mathsf{acc}, usk_i)$ decrease as a result of tx^*;
  (2) The sum of all corrupted users' balances $\sum_{u \in \mathsf{CU}} \mathsf{AZ.Read}(\mathsf{acc}, usk)$ increases as a result of tx^*.
- Outputs 1 if $\mathcal{A}$ wins, otherwise, outputs 0.

DEFINITION 17. *The relay system satisfies the overdraft prevention property if any PPT adversary in the* $\mathrm{Exp}^{\mathsf{op}}$ *wins with only negligible probability, i.e.,* $|\Pr[\mathrm{Exp}^{\mathsf{op}}(\mathcal{A}, \lambda) = 1]| \leq$*negl(*$\lambda$*).*

Gas fee fairness. This property ensures that no malicious users can cheat two or more honest relayers simultaneously without punishment, and any honest user who only requests one relayer in one epoch does not lose money. The gas fee fairness experiment $\mathrm{Exp}^{\mathsf{fair}}$ includes $\mathrm{Exp}^{\mathsf{fair-relayer}}$ and $\mathrm{Exp}^{\mathsf{fair-user}}$.
$\mathrm{Exp}^{\mathsf{fair-relayer}}$ is formalized as follows.

- $\mathcal{A}$ receives rpp and has access to the oracles: $\mathbf{AddUser}$, $\mathbf{AddRelayer}$, $\mathbf{CrptUser}$, $\mathbf{CrptRelayer}$, $\mathbf{Register}$, $\mathbf{MetaTxGen}$, $\mathbf{BrdReq}$, $\mathbf{FullTxGen}$.
- $\mathcal{A}$ outputs n full transactions $\{tx_i\}_{i=1}^n$ for the same *epoch*, each transaction tx_i corresponds to a relayer R_i. It wins if all the following conditions hold:
  (1) W.l.o.g., the first k transactions $\{tx_i\}_{i=1}^k$ are failed and the first t of them are generated

from the honest relayers: $\{R_i\}_{i=1}^{t} = \{R_i\}_{i=1}^{k} \cap \mathsf{HU}$ are the honest relayers whose transactions fail, where $2 \leq t \leq k < n$;

(2) $\exists\ tx_i \in \{tx_i\}_{i=1}^{t}$, such that $\forall\ tx_j \in \{tx_i\}_{i=1}^{n}$, $\mathsf{Reimburse}(epoch, rpk_i, rpk_j, tx_i, tx_j)$ fails.

- Outputs 1 if $\mathcal{A}$ wins, otherwise, outputs 0.

$\mathrm{Exp}^{\mathsf{fair-user}}$ is formalized as follows.

- $\mathcal{A}$ receives rpp and has access to the oracles: $\mathbf{AddUser}$, $\mathbf{AddRelayer}$, $\mathbf{CrptUser}$, $\mathbf{CrptRelayer}$, $\mathbf{Register}$, $\mathbf{MetaTxGen}$, $\mathbf{BrdReq}$, $\mathbf{FullTxGen}$.
- $\mathcal{A}$ outputs u^*, pid^* and sif'. It wins if all conditions hold:

  (1) u^* is an honest user: $u^* \in \mathsf{HU}$;

  (2) pid^* is u^*'s public identifier: $(u^*, pid^*) \in \mathsf{BB}$

  (3) u^*'s collateral can be obtained by the relayer based on sif': $\mathsf{Colla.Judge}(sif', pid^*) = 1$
- Outputs 1 if $\mathcal{A}$ wins, otherwise, outputs 0.

DEFINITION 18. *The relay system is gas fee fair if any PPT adversary in the* $\mathrm{Exp}^{\mathsf{fair}}$ *can win with only negligible probability, i.e.,* $|\Pr[\mathrm{Exp}^{\mathsf{fair}}(\mathcal{A}, \lambda) = 1]| = |\Pr[\mathrm{Exp}^{\mathsf{fair-relayer}}(\mathcal{A}, \lambda) = 1]| + |\Pr[\mathrm{Exp}^{\mathsf{fair-user}}(\mathcal{A}, \lambda) = 1]| \leq$ *negl(*λ*).*

5.4.2 Construction

This section describes the relay system for anonymous Zether via our ADAC construction in Sec. 5.3 and collateral smart contracts.

Setup and key generation: The Ethereum blockchain has been set up, and the anonymous Zether has been deployed as the main smart contract. The relay system public parameter *rpp* is generated, including $\{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e(\cdot,\cdot), H_1(\cdot), H_2(\cdot), g, \tilde{g}, \tilde{h}, \mathsf{BB}\}$. For simplicity, rpp will implicitly be an input of all the following algorithms.

The user runs ADAC.UserKeyGen algorithm and gets his personal secret and public keys $usk = (v, s, sif = \tilde{g}^s)$, $upk = (g^v, pid = e(g, \tilde{g})^s)$. He sets $zsk = v, zpk = g^v$ as the secret and public keys of his anonymous Zether account.

The relayer runs the $\mathsf{ADAC.IssuerKeyGen}$ algorithm to get his secret key $rsk = isk = X$, and the public key $rpk = ipk = (Y_1, Y_2, \widetilde{X}, \widetilde{Y_1}, \widetilde{Y_2})$. He publishes the rpk and posts a collateral smart contract on the chain, as shown in Fig. 5.3. The collateral smart contract embeds the relayer's Ethereum account address addr. Its functions include[5]:

- *Join*: On receiving the user's deposit transaction, including his public key and public identifier, it stores them in the contract state and adds the deposited ETH to his collateral balance.

- *Judge*: On receiving the user's secret information and public identifier, checks that the public identifier is stored in the contract state and *sif* is valid. If all these checks pass, send the user's collateral to the relayer's account and delete the user from the contract state.

Registration: The user runs $\mathsf{Register}(rpk, upk, usk)$ to interact with the relayer running $\mathsf{RegRcv}(upk, rsk)$.

Concretely, the user sends his Zether public key and public identifier and deposits ETH as collateral locked in the relayer's collateral smart contract. His public identifier is also added to the smart contract's identifier set $\mathcal{U}$. Then, the user requests the relayer to issue an anonymous credential by running $req \leftarrow \mathsf{ADAC.Request}(rpk, upk, usk)$ and sends req.

The relayer checks that the smart contract has received the collateral and that the user's public identifier pid is the same as his public identifiers in other relayers' collateral smart contracts if any[6]. If so, he runs $cred'/\perp \leftarrow \mathsf{ADAC.Issue}(upk, rsk, req)$. If the request req is valid, the relayer sends the partial credential $cred'$ to the user. Otherwise, abort. Then, the user runs $cred \leftarrow \mathsf{ADAC.Receive}(rpk, upk, usk, \textit{req}, \textit{cred'})$. As a result, the collateral smart contract updates $\mathcal{U}$ to $\mathcal{U} \cup \{pid\}$. The user obtains $\textit{cred} = (\sigma_1, \sigma_2) = (g^{r_1}, X^{r_1} Y_1^{s \cdot r_1} Y_2^{v \cdot r_1})$.

Broadcast: The user generates a meta-transaction $\mathsf{meta} = \{u, D, (y_i, C_i, C_{Ln,i}, C_{Rn,i})_{i=0}^{N-1}, \pi_{\mathsf{AZ}}\}$ as shown in Sec. 2.4. Especially, $u = g_{\mathsf{epoch}}^{v}$ is the nonce, and the relayer is included as one of

[5]It also allows users to quit and get a refund for their collateral by signing with their zsk. Upon doing so, they are removed from $\mathcal{U}$ without impacting their ability to use other relayers. It is straightforward, and we omit it for brevity.

[6]All collateral smart contracts and their registered users are available on the chain. Note that this check is only one time during the user's registration.

Setup
Inputs: security parameter λ, relayer account address addr
(1) $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g, \tilde{g}, e) \leftarrow \mathsf{BG.Gen}(1^\lambda)$
(2) Initialize empty account table acc: $\mathbb{G}_1 \rightarrow \mathbb{G}_1$
empty balance table bal: $\mathbb{G}_1 \rightarrow \mathbb{Z}$
empty identifier table pid: $\mathbb{G}_1 \rightarrow \mathbb{G}_T$
empty identifier set $\mathcal{U} = \emptyset$

Join
Inputs: Zether public key zpk, public identifier pid, deposit transaction tx
(1) Let $b =$ tx.value which is the deposit amount
(2) If $\mathsf{acc}[zpk] = \perp$
- Set $\mathsf{acc}[zpk] = zpk$, $\mathsf{bal}[zpk] = b$
- Set $\mathsf{pid}[zpk] = pid$, add pid to $\mathcal{U}$

(3) Else:
- require: $\mathsf{pid}[zpk] = pid$
- Set $\mathsf{bal}[zpk] = \mathsf{bal}[zpk] + b$

Judge
Inputs: public key zpk, public identifier pid, secret information *sif*
(1) require:
- $\mathsf{acc}[zpk] \neq \perp$, $\mathsf{pid}[zpk] = pid$
- Verify $e(g, \mathit{sif}) = pid$

(2) Send $\mathsf{bal}[zpk]$ ETH to addr
- Set $\mathsf{acc}[zpk] = \perp$, $\mathsf{bal}[zpk] = \perp$
- Set $\mathsf{pid}[zpk] = \perp, \mathcal{U} = \mathcal{U} \setminus \{pid\}$

FIGURE 5.3. Relayer's collateral smart contract

the receivers to get rewards in ZTH. With meta and $cred$, the user runs $\mathsf{BrdReq}(\mathit{epoch}, \mathit{rpk}, \mathit{usk}, \mathit{cred}, \mathsf{meta})$ to generate the full transaction request *fullreq* and sends it to the relayer. The relayer runs $\mathsf{BrdVrfy}$ $(\mathit{epoch}, \mathit{rpk}, \mathsf{acc}, \mathit{fullreq})$. They work as follows:

(1) Let $m = \mathsf{meta}, tag = u$, the user runs $\Sigma \leftarrow \mathsf{ADAC.Show}(rpk, usk, cred, m, tag)$ where $\Sigma = (\sigma^*, c, T, \pi_{\mathsf{show}})$.

(2) The user also generates a zero-knowledge proof π_{bind} for the following relation $\mathcal{R}_{\mathsf{bind}}$, which proves that meta and Σ were generated by the same user.

$$\mathcal{R}_{\mathsf{bind}} = \{(rpk, \sigma^*, u; s, v, r_3) : u = g_{\mathsf{epoch}}^v \wedge$$
$$e(\sigma_1^*, \widetilde{X}) \cdot e(\sigma_1^*, \widetilde{Y}_1)^s \cdot e(\sigma_1^*, \widetilde{Y}_2)^v \cdot e(\sigma_1^*, \tilde{g})^{r_3} = e(\sigma_2^*, \tilde{g})\}$$

(3) He sends the full transaction request *fullreq* $= (\mathsf{meta}, \Sigma, \pi_{\mathsf{bind}})$.

(4) On receiving *fullreq* $= (\mathsf{meta}, \Sigma, \pi_{\mathsf{bind}})$, the relayer verifies by running $\mathsf{AZ.Vrfy}(\mathsf{acc}, \mathsf{meta})$, $\mathsf{ADAC.Vrfy}(rpk, \Sigma, m, tag)$ and verifying the proof π_{bind} on (rpk, σ^*, u). If any verification fails, output $\perp$. Otherwise, it wraps (meta, c, T) in the data field of a full transaction tx and broadcasts it and pays the gas fee.

Reimburse: Given different full transactions tx_1, tx_2 on the chain sent by relayers rpk_1, rpk_2 with the same nonce u and *epoch* and an identifier set $\mathcal{U}$ of the collateral smart contract, the relayer runs $\mathsf{Reimburse}(epoch, rpk_1, rpk_2, tx_1, tx_2, \mathcal{U})$ to get reimbursed:
Run $\mathsf{ADAC.Extract}$ $(ipk_1, ipk_2, m_1, m_2, tag_1, tag_2, \Sigma_1, \Sigma_2, \mathsf{PID})$ where $ipk_1 = rpk_1$, $ipk_2 = rpk_2, tag_1 = tag_2 = u, m_1 = \mathsf{meta}_1, m_2 = \mathsf{meta}_2$, $\mathsf{PID} = \mathcal{U}$ and get $sif = \tilde{g}^s$. He computes $e(g, \tilde{g}^s)$ and locates $pid = e(g, \tilde{g}^s)$ in $\mathcal{U}$ with respective zpk. He sends zpk, pid, sif to the collateral smart contract and calls the **Judge** function.

The smart contract checks that it is an existing user and $pid = e(g, sif)$, then it sends his collateral to the relayer's address and revokes the user by deleting his *pid* from $\mathcal{U}$.

THEOREM 8. *Assuming the abuse deterring anonymous credential is anonymous, extractable, and non-frameable, the underlying anonymous Zether is ledger indistinguishable and overdraft safe. The underlying proof system is a zero-knowledge argument of knowledge and all users are rational. Then, the relayer system satisfies anonymity, overdraft prevention, and gas fee fairness.*

PROOF. *Anonymity.* The relayer receives $(\mathsf{meta}_{\mathsf{AZ}}, \Sigma, \pi_{\mathsf{bind}})$. Due to the ledger indistinguishability of the anonymous Zether meta-transaction, the anonymity of the ADAC, and the zero-knowledge property of the underlying proof system, the relayer cannot distinguish it. Given the honest full transaction, it only adds the relayer's information. So, it still cannot be linked to any users.

In the anonymity experiment, the challenge identities u_0, u_1 are honest registered users in the valid issuer R^*. The current epoch is $epoch^*$. We design the hybrid games:

- $\mathsf{G}_{\mathsf{real}}$: This game is the same as the experiment. The challenger $\mathcal{C}$ chooses $b \leftarrow\$ \{0,1\}$ and generates $meta_b$ and $fullreq_b$ by calling $\mathbf{MetaTxGen}(u_b, R^*, epoch^*\ k)$ and $\mathbf{BrdReq}(u_b, R^*, meta_b)$, where $fullreq_b = (meta_b, \Sigma_b, \pi_{\mathsf{bind}b})$.
- G_1: This game is similar to $\mathsf{G}_{\mathsf{real}}$ except that the bind proof $\pi_{\mathsf{bind}b}$ is simulated without witness.
- G_2: This game is similar to G_1 except that the meta-transaction $meta_b$ is simulated via calling the simulator in the anonymous Zether and generating the token Σ_b by calling $\mathbf{BrdReq}$ on u_b, R^* and the simulated $meta_b$.
- G_3: This game is similar to G_2 except that the token Σ is simulated by calling the ADAC simulator. Thus, in this game, the full request is independent of b.

Compare G_1 with $\mathsf{G}_{\mathsf{real}}$, the only difference is that the proof is simulated. Since this proof is zero-knowledge, the probability of distinguishing G_1 from $\mathsf{G}_{\mathsf{real}}$ is negligible. We have that $|\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{G}_1(\mathcal{A}, \lambda) = 1]| \leq negl(\lambda)$.

Compare G_2 with G_1, in G_2, the only difference is the meta-transaction is simulated. Since the anonymous Zether is ledger indistinguishable, as proven in [17], We have that $|\Pr[\mathsf{G}_1(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{G}_2(\mathcal{A}, \lambda) = 1]| \leq negl(\lambda)$.

Compare G_3 with G_2, in G_3, the only difference is the meta-transaction is simulated. Since the ADAC is anonymous, as proven in Thm. 7, we have that $|\Pr[\mathsf{G}_2(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{G}_3(\mathcal{A}, \lambda) = 1]| \leq negl(\lambda)$. In G_3, $\mathcal{A}$'s view is independent of b. Thus, $\mathcal{A}$ just outputs a random guess b' in G_3, so its advantage is 0: $\Pr[\mathsf{G}_3(\mathcal{A}, \lambda) = 1] - 1/2 = 0$. In summary, we have

$$
\begin{aligned}
&|\Pr[\mathrm{Exp}^{\mathsf{anony}}(\mathcal{A}, \lambda) = 1] - 1/2| = |\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A}, \lambda) = 1] - 1/2| \\
\leq &|\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{G}_1(\mathcal{A}, \lambda) = 1]| + |\Pr[\mathsf{G}_1(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{G}_2(\mathcal{A}, \lambda) = 1]| \\
&+ |\Pr[\mathsf{G}_2(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{G}_3(\mathcal{A}, \lambda) = 1]| + |\Pr[\mathsf{G}_3(\mathcal{A}, \lambda) = 1] - 1/2| \leq negl(\lambda)
\end{aligned}
$$

Overdraft prevention. This property can be reduced to the overdraft safety of the anonymous Zether. We do not change the anonymous Zether smart contract and let its meta-transaction be generated in a black-box way. If the user colludes with the relayer and generates a valid

payment that allows the user to spend more, they violate the overdraft safety of the underlying anonymous Zether, which happens only with negligible probability.

Gas fee fairness. We first consider the fairness for honest users and the adversary $\mathcal{A}$ in experiment $\mathrm{Exp}^{\mathsf{fair-user}}$. The user allows the relayer to be one receiver of the final anonymous payment transaction, so he pays the relayer only if that transaction has been validated on-chain and executed. We call a user is honest if he only requests at most one relayer in each epoch. The adversary wins if an honest user loses money in any of the following cases:
(1) his transaction is not executed, but he pays the transaction fee;
(2) he does not misbehave but loses his collateral.
Since the Ethereum blockchain is secure under the assumption of an honest majority, the anonymous Zether and collateral smart contract are secure, so event (1) happens only with negligible probability, and event (2) happens only if the honest user's secret information is provided. However, since the user is honest, if his secret information is revealed, it means the non-frameability of ADAC is violated, which happens with only negligible probability, as proven in Thm. 7. Therefore, $\mathcal{A}$ wins in $\mathrm{Exp}^{\mathsf{fair-user}}$ with only negligible probability.

Then, we consider the fairness for honest relayers and the adversary in $\mathrm{Exp}^{\mathsf{fair-relayer}}$. For the honest relayers, they verify the user's request $(\mathsf{meta}_{\mathsf{AZ}}, \Sigma, \pi_{\mathsf{bind}})$ at first. If the verification passes according to the current blockchain's state, it means the user has enough money in the anonymous Zether smart contract to make the payment and pay the relayer, and he is a registered user in the relayer with enough collateral due to the soundness of Π_{bind}. Then, they broadcast the full transaction and pay the gas fee when they are recorded on the chain. If the full transaction is valid and executed, the relayer gets paid in ZTH. If it is invalid due to the user's misbehavior, some other meta-transactions must exist on-chain containing the same nonce.

Since the smart contract is secure, the relayer can always receive the malicious user's collateral as long as he extracts the user's secret information and revokes him immediately. If $\mathcal{A}$ wins such that the relayer cannot get reimbursed, it means the secret information extraction fails even if given multiple full transactions with valid ADAC tokens on the same tag. Based on these, multiple valid ADAC tokens are produced that violate the extractability of ADAC,

which happens with only negligible probability. Therefore, $\mathcal{A}$ wins in $\mathrm{Exp}^{\mathsf{fair-relayer}}$ also with only negligible probability.

In summary, the gas fee fairness property holds.

□

REMARK 8 (**Registration check**). *The relayers operate on a shared public blockchain. While the blockchain cannot be used to verify whether a meta-transaction has been sent (see Sec. 5.2), it does record all collateral smart contracts that contain user accounts and public identities. This allows relayers to check whether a user has already registered with another relayer using a different pid. Additionally, the registration transaction is not anonymous but does not reveal any connection to an honest user's payment transaction.*

REMARK 9 (**Necessity of bind proof**). *The user must generate the bind proof π_{bind}; otherwise, n malicious users could collude to generate tokens with the same nonce using their own credentials and send them to n different relayers. In this case, none of their secret information can be extracted, and $n-1$ relayers would waste their gas fees without being able to identify any of the malicious users. Therefore, the relayer must verify through the bind proof that the sender of* meta *is using his own credential.*

REMARK 10 (**Anonymity set**). *Each user's Zether account is linked to their pid. When processing a meta-transaction, the relayer verifies whether the associated Zether accounts are registered with them. If an account is not registered, it cannot belong to the real sender and would be excluded from the anonymity set. Consequently, the user should select dummy Zether accounts with pids also within the same $\mathcal{U}$ to prevent the anonymity set from being reduced.*

REMARK 11 (**Efficient Revocation**). *In general, a user must prove they have not been revoked by providing a (non-)membership proof. In our scheme, the proof π_{bind} ensures the user's non-revocation. Each user's Zether account is linked to their pid. As noted in the previous remark, all accounts involved in an anonymous meta-transaction are registered*

and unrevoked users associated with the same relayer. This allows the relayer to verify non-revocation directly, eliminating the need for the user to generate additional (non-)membership proofs.

REMARK 12 (**Incentive**). *Our system incorporates a reward mechanism for relayers who receive transaction fees in* ZTH *from the sender. These fees can exceed gas fees paid in* ETH*, providing a strong incentive for relayers to offer gas relay services. It represents a reasonable cost for users seeking to maintain their privacy.*

REMARK 13 (**Punishment**). *A user can register with multiple relayers by depositing collateral into their respective collateral smart contracts. If the user attacks even just two relayers, any victimized relayer can expose the user's secret information on-chain. This enables all other relayers, regardless of whether they were attacked, to claim the user's collateral. Consequently, the malicious user forfeits their entire collateral across all relayers. This all-lose-collateral mechanism serves as a punishment for malicious behavior.*

Evaluation. In a basic relay system, relayers simply verify meta-transactions and submit them on-chain. Our construction requires both users and relayers to perform additional steps to ensure fairness. We evaluate our system by analyzing the extra communication cost between users and relayers, the increased size of the full transaction, and the additional computational overhead for both users and relayers compared to the basic relay system. The evaluation results indicate that the added overhead remains minimal.

In Broadcast step, except meta, the user sends $(\Sigma, \pi_{\mathsf{bind}})$ in the *fullreq*, which consists of $7\mathbb{G}_1 + 1\mathbb{G}_2 + 4\mathbb{Z}_p$ elements. The relayer additionally includes (c, T) in the full transaction. It consists of $1\mathbb{Z}_p + 1\mathbb{G}_2$ elements.

Extension. Our relay system can also be extended to handle other kinds of anonymous applications, such as the anonymous sealed-bid auction [101, 100]. In this scenario, bidders submit their anonymous bid meta-transactions on-chain with the help of relayers. A bidder can only place one valid bid per auction session. He can generate a tag according to the auction session number and his secret key, and the bid meta-transaction contains the relayer's

TABLE 5.2. Extra computation cost*.

	Registration	Broadcast	Reimburse
User	$5\mathsf{E}_{\mathbb{G}_1} + 1\mathsf{H}$	$8\mathsf{E}_{\mathbb{G}_1} + 2\mathsf{E}_{\mathbb{G}_2} + 3\mathsf{H}$	N/A
Relayer	$6\mathsf{P} + 2\mathsf{E}_{\mathbb{G}_1} + 4\mathsf{E}_{\mathbb{G}_T} + 1\mathsf{H}$	$14\mathsf{P} + 1\mathsf{E}_{\mathbb{G}_2} + 4\mathsf{E}_{\mathbb{G}_T} + 3\mathsf{H}$	$1\mathsf{E}_{\mathbb{G}_2} + 1\mathsf{P}$

* P refers to the cost of a pairing computation, $\mathsf{E}_{\mathbb{G}_i}$ to an exponentiation in $\mathbb{G}_i$, $(i \in \{1, 2, T\})$ and H to a hash computation.

information. Within our relay system, if a user requests many relayers to submit bids with his credentials for the same session, he would be detected, and the relayers can get his collateral as reimbursement.

Open problems. *Issuer anonymity.* An interesting direction would be to explore issuer anonymity, where the token is unlinkable to its issuer. It may require the issuers to collaborate during setup.

General anonymous smart contract. Other anonymous smart contracts offer functionalities beyond just payments. In such cases, the question arises: how should gas fees be paid to the relayers? We leave it as a future work.

CHAPTER 6

Predicate Aggregate Signatures and Applications

6.1 Background

Anonymous reputation systems are widely used in many applications. For example, on online platforms, Internet peers can jointly establish accumulated ratings on the merchants/service providers or certain products so that users who are not familiar with them can have some context to make a better choice. Since the main necessary information is the accumulated rating, ensuring anonymity plays a crucial role in allowing users to participate in the reputation systems. More specifically, in YouTube, each user registers at YouTube, and then gives his rating on each content as an "I like it" (like $+1$) or not, then there will be an accumulated content score shown in the platform. The accumulated score not only serves as a succinct representation/description, but also *hides the identities* of the voters. To reduce the reliance on fully trusting the platform, other important requirements are that the accumulated score should be *publicly verifiable*, so that users may have stronger confidence that the score is not manipulated by the platform; furthermore, in the anonymous setting, one potential threat arises when a malicious platform attempts to manipulate the ratings by repeatedly counting one user's vote for many times. Therefore, *additional measures* must be taken to assure the verifier that each voter's contribution to the accumulated score is limited to a single vote.[1]

Naturally, individual votes can be realized via digital signatures from legitimate users (e.g., registered identities). To obtain a succinct accumulated score k, say up-votes, we would like

[1]There are also other types of rating systems, such as Uber/Airbnb, that are based on accumulation on each transaction, so each user may rate on the same service provider more than once. We only consider the common version as a motivational example of our primitive.

to aggregate the corresponding identities and signatures to be a short "proof". The proof needs to ensure that indeed there are at least k signatures on "I like it" from k *distinct* identities.

The one-user-one-vote requirement above can be seen as a special policy that was put on the identities of those signatures. In broader applications, there could be more complex voting policies that could be expressed as a predicate on the voter identities. For example, in blockchain governance (e.g., Decentralized autonomous organizations (DAOs)[119]), decisions could be made by the whole community whose accounts hold a sufficient amount of tokens. The final decision needs to be attested with a short proof that the voting result indeed follows the governing policy, and the proof would be stored on-chain. Voting processes in DAOs offer a remarkable degree of flexibility and customization. These processes can be tailored and programmed to accommodate a wide range of requirements and preferences. For example, quadratic voting [120, 121] allows the voter to have budgets of credits, which are converted to counted votes according to their square root. Delegated voting [122] allows users to delegate their voting power to trusted individuals or entities. Property-based voting [123] differentiates signers based on the properties of their non-fungible tokens used in voting.

6.1.1 Motivation

Motivated by the above applications and many other relevant ones, in this chapter, we are studying a general problem for aggregating signatures and keys on multiple messages while ensuring that the signers satisfy some public predicate without disclosing their identities. We call such a cryptographic primitive *predicate aggregate signatures*, PAS for short.

More specifically, let us consider a set of users denoted as $\mathcal{U} = \{u_i\}_{i\in[n]}$ and a collection of messages $\mathcal{M} = \{m_j\}_{j\in[k]}$ drawn from a predefined message space. Users choose the messages to sign. There is also a combiner, which aggregates the corresponding signatures and signer identities/public keys into one succinct certificate/proof/signature and shows a description Δ of signers (like the number) on each message. The signature also confirms the legitimacy of both the signers and signatures, ensuring that the signers and the description

adhere to a particular public predicate P, i.e., $P(S_1, \ldots, S_k, \Delta) = 1$, where each $S_i \subseteq \mathcal{U}$ is a subset of users.

For example, in the anonymous reputation system, the rate-once policy requires that each signer can only sign once at most. It means there is no duplicate signer in each subset, and all subsets are disjoint, i.e., $S_i \cap S_j = \emptyset$ for any $i \neq j$. Another example is the on-chain voting system with a special policy. Besides showing the number of voters, the property policy [123] requires that some of the voters have special properties. By representing the property via index, the combiner can ensure the policy is satisfied by proving that some voters' indices belong to a specific range, e.g., there is at least one voter in the subset who is a senior member with an index smaller than 50.

6.1.2 Challenges

In this chapter, we formulate, construct, and analyze the new primitive of predicate aggregate signatures to address the remaining issues.

Formulating PAS. We aim to give a formal definition and security models for predicate aggregate signatures. As mentioned above, it allows registered users (public keys, identities known and made public) to sign on multiple messages and these signatures can be aggregated by a combiner who hides these signers' identities. The final signature only reveals a description of signers and guarantees that the signers and this description satisfy the public predicate [2].

We formally define the security model of predicate aggregate signatures. It enjoys the following features simultaneously, which advances existing primitives.

- *Transparent setup*: users generate pk, sk on their own, and a setup algorithm only publishes public parameters for the system.

[2] In later, we would use the *dynamic threshold* as an example of the description. It reveals the number of users who have signed on the message. We choose it as the example for three reasons: (1) For a simpler presentation that shows how we can get our final construction step by step; (2) the dynamic threshold is a natural feature of our motivated anonymous reputation system; (3) the dynamic threshold aggregate signature itself might be of independent interest, and indeed, it already advances the state of the art of several relevant signatures.

- *Signer anonymity*: the adversary (not the combiner) cannot get any information about each individual signer identity/public key (e.g., whether he signed on a particular message) except the public description from the final signature, even if the adversary corrupts *all* users, including the target himself.
- *Unforgeability*: the adversary cannot convince the verifier, if it does not collect enough signatures, or the predicate is not satisfied. To facilitate such a notion, we generalize the classical proof of knowledge and define a signer identity extractor.

Efficient constructions from standard assumptions. We proceed in several steps towards the full construction, with concrete efficient instantiations. Our starting point is the BLS aggregate signature [35]. It allows the combiner to aggregate a set of partial signatures on multiple messages.

Transparent setup. First of all, each user generates his secret-public key pair $(sk_i = x_i, pk_i = g^{x_i})$, and registers pk_i. To avoid the known rogue-key attacks [35], we first let each user run proof of knowledge of x_i during the registration. The system simply includes $pk_1, \ldots, pk_n$, and some common parameters $g_1, \ldots, g_n$ as public parameters and makes them available to everyone.

Succinct size solution. We start with the core building block of dynamic threshold aggregate signature. This can be considered a special case where the predicate only requires the threshold counting to be correct.

An intuitive idea is letting the combiner do more work: not only the partial signatures are aggregated, but their respective public keys are also compressed into a compact version. To protect signer anonymity, some blind factors are added to the compressed public keys and signatures. However, anonymity introduces a concern regarding the correctness of compressed public keys. Specifically, there is no guarantee that these compressed public keys are part of the legitimate/registered public key set.

Therefore, the combiner needs to produce an additional proof for the membership relation and duplication checks (that there are true t signatures from t *distinct* signing keys). A naïve attempt for the latter would be proving the pairwise difference on all the compressed signing

keys, which will yield a *quadratic* size proof. Some techniques in relevant primitives such as graded signatures [124] and signature of reputation [125] got around the challenge and sorted the public keys first, to do a sequential proof that $pk_i \neq pk_{i+1}$, which can push down the proof size to be *linear*. However, that is still quite cumbersome.

Alternatively, we observe that instead of proving relations among signer keys directly, we may leverage the published public keys in the public parameter. First, we can represent the included keys as a binary vector $\boldsymbol{b} = (b_1, ..., b_n)$, i.e., $b_i = 1$, if pk_i is in (has signed on the message), and 0 otherwise, and commit $\boldsymbol{b}$ in a succinct way (via vector commitment). Then, we can prove an alternative statement that the committed vector is indeed *binary*. Now, the Hamming weight of this vector will correspond to the threshold. Two remaining parts: (i) each bit value is assigned correctly; (ii) Hamming weight is correctly computed. For (i), observe that when each pk_i is directly taken in as part of the system parameter, the "aggregated public keys" $\widehat{pk} = \Pi_i pk_i^{b_i}$ can be seen as another "commitment" to the binary vector. We can establish the validity of the bit assignment by demonstrating that the previously committed binary vector is identical to the one contained in $\widehat{pk}$. While for (ii), Hamming weight can again be derived directly from inner product of the bit vector and all 1 vector, and proven using the efficient inner product argument from Bulletproofs [126].

Now we have a construction framework from the inner product argument and "binary" proof (that proves a committed vector is binary), which can be instantiated via Bulletproofs [126], yielding a signature of *logarithmic size* relative to the number of all users.

In the multiple (say k) messages setting, signers are divided into multiple sets depending on the message they have signed. A natural method is running the above proof generation for k times, so the total communication cost would have a multiplicative factor of k. Fortunately, by exploring the above technique further, we can generate a proof for k values on the knowledge of n-length binary vectors. In this way, these k proofs can be aggregated into one single proof for a $(k \cdot n)$-length binary vector. As a result, we achieve a communication cost of $O(k + \log n + \log k)$, comprising k aggregated public keys and additional proofs of size $O(\log n + \log k)$.

Reduce verification time. However, the above signature still requires a linear verification time (for example, even reading in all the public keys). To reduce verification time, we propose a new proof system for the inner product and binary relations with structured parameters that can also reduce the verification time to logarithmic.

There were previous efforts improving verification cost [36, 127], in [36], the authors achieve logarithmic size and logarithmic verification time for inner product argument and range proof using *structured* reference string with highly correlated parameters in the form of $g, g^{x_1}, g^{x_2}, \ldots, g^{x_{\log n}}, g^{x_1 \cdot x_2}, g^{x_1 \cdot x_3}, g^{x_2 \cdot x_3}, g^{x_1 \cdot x_2 \cdot x_3} \ldots$, that separates the parameter into two parts: linear proving parameter and logarithmic verification parameter.

Unfortunately, as we would like a transparent setup, public keys are generated by users themselves randomly and then included as the public parameter, which is clearly inconsistent with these structured parameters.

To work around this, we need to redesign the parameter generation and the statement for the proof. Besides the binary vector, we also commit the public keys via structure-preserving commitment of [34] (also called AFGHO commitment). Introducing structured parameters into it is still compatible with the randomly generated public keys. Given these two commitments, we can prove another element is the inner pairing product of the two committed vectors. We observe that demonstrating the well-formedness of the aggregated public key is equivalent to proving that its bilinear map is equal to the inner pairing product between a binary vector and all public keys. Now we prove the validity of $\widehat{pk}$ by directly leveraging the inner product argument between two committed vectors. One of these vectors is a binary vector, whose correctness is guaranteed by a binary proof, while the other comprises all the public keys. By adjusting the AFGHO commitment with structured parameters, these proofs achieve efficiency with logarithmic communication costs and verification times. We refer detailed description in Sec. 6.4.1.

Achieve anonymity. In the anonymous setting with a blind factor r, where $\widehat{pk} = \Pi_i pk_i^{b_i} \cdot \tilde{g}^r$, several challenges arise when applying the previous method. These challenges include proving the last position of the binary vector is 1 and handling commitments of public keys together

with the random factor $\tilde{g}^r$. These challenges are exacerbated by the anonymity requirements. See Sec. 6.4.1 for detailed discussion.

To mitigate these issues, a new approach is proposed. First, $\boldsymbol{b}$ and r are committed separately using distinct commitment keys, and proofs are generated for each. Then, by combining these two commitments, we can prove the presence of both a binary vector and a blind factor in specific positions. Subsequently, an inner pairing product argument is applied to these vectors, ensuring the well-formedness of the blinded aggregated public key.

Generic predicate. Then to lift the construction to support any arithmetic predicate on the signer identities, we observe that both techniques for the core building block is via Fiat-Shamir transformation on Σ-protocols. We can add the extra proof of predicate satisfaction similarly via Bulletproof with our optimized verification time, then use the classical And proof to bind them. The final proof is with logarithmic size and verification time, while its security can be based on the standard SXDH assumption.

Efficient instantiations for concrete predicates. We also give a concrete construction for the concrete predicate that all signer sets are also disjoint (that denotes the rate-once policy in the motivational application of the anonymous reputation system).

It is a challenging task for the combiner to demonstrate the disjoint nature of all of these subsets of signers. In general, it would require comparing every pair of them and proving that they are indeed disjoint. However, this approach would necessitate a quadratic number of comparisons, leading to additional significant communication and computation costs.

It is worth noting that the binary feature can also be utilized in this case. Specifically, each public key subset can be represented as a binary vector. The addition of two binary vectors corresponds to the union of the corresponding subsets, including duplicate elements, if any. In case the resulting sum vector remains binary, it implies that there are no duplicate elements in the union set, thereby indicating that the two sets are disjoint. By extending this approach to the k-subsets scenario, where we add all these binary vectors, we can demonstrate that all of the public key subsets are indeed disjoint.

6.2 Related Works

Despite that there is much relevant research on signature aggregation, anonymous authentication, and others, none of them gives a PAS in a satisfying way (as shown in Table 6.1). We first give a simple categorization of existing relevant primitives, briefly describe the insufficiency of each type. Besides that, most of the primitives do not support a general policy validation on the signers, and each of them lacks some other critical properties. Jumping ahead, we will show that some of the concrete instantiations of our PAS directly advance the state of the art of several of those well-studied primitives, see Table 6.2.

TABLE 6.1. Comparisons of relevant primitives.

Primitives	Trans. setup	Flexi. thld.	Agg. msgs[1]	Anony.	Signer Policy
Thld Sig. [128]	×	×	×	✓	×
Multi-Sig [129]	✓	✓	×	×	×
Agg-Sig. [35]	✓	✓	✓	×	×
Graded Sig. [124, 125]	✓	✓	×	✓	×
Compact Cert. [130]	✓	✓	×	×	×
Thld-ring Sig. [131]	✓	×	×	✓	×
Attri-based Sig. [132]	×	×	×	✓	✓[2]
Our PAS.	✓	✓	✓	✓	✓

[1] Agg. msgs means signatures can be compressed among different messages.
[2] The predicate in this setting is applied to one single user's attribute set while we consider the predicate across multiple users.

Signature aggregations. Multi-signatures [129], aggregate signatures [35], threshold signatures [128], and several other relevant ones allow one to compress signatures from different users. Besides, they usually have no anonymity guarantee, the former two have to explicitly provide the signer identities/public keys thus, the total proof size and verification cost still remain at least linear to the threshold (which is usually linear to the total number of users); threshold signature, on the other hand, can have one single public key for verification, but via a trusted setup, when its threshold is fixed, and it does not support signature aggregation across multiple messages. Multi-key homomorphic signatures [133] evaluates the messages signed by different users, but it does not protect the privacy of signers. All signers' identities are public, which is not suitable for our anonymous setting.

Anonymous primitives. Anonymity-oriented signatures, such as ring signatures [102, 134] and the linkable [135] and threshold versions [131], usually do not require the identities of the ring being aggregated, and often with a fixed threshold. Scored anonymous credential [136] is used for privacy-preserving reputation enforcement. The user's reputation is decided by some service provider. While we are considering the reputation voting setting where a set of users rate a product by signing.

Attribute-based signatures (ABS) [132] allow a user to attest that his attributes satisfy certain predicate. Anonymity can be implicitly ensured if two users have the same attribute set. However, ABS requires a trusted key generation center, it does not consider signature aggregations, or policies across multiple users. In our context, each user independently generates their own keys, and our goal is to have the flexibility to aggregate signatures and apply predicates across multiple users.

Generic constructions. Generic zk-SNARKs could certainly provide a path for feasibility. By collecting numerous signatures from signers, the combiner can create a zk-SNARK proof that guarantees the existence of sufficient valid partial signatures satisfying the public predicate, while concealing the signers' identities and revealing only the counts. However, the generation of zk-SNARK proofs remains prohibitively expensive, and it relies on trusted setups and unfalsifiable assumptions. Some recent efforts have focused on constructing *dynamic threshold* signatures[3] [137, 138] directly in the AGM model [139], whose actual security is not well-understood, and may have subtle vulnerabilities [140]. While we focus on building the PAS on classical and more standard assumptions. Additionally, in its silent setup phase, the users need to maintain some hint parameters, whose size is linear to the number of all users. While we require each user only need to keep their secret keys with constant storage.

Advancing relevant primitives. Our PAS (including the building block alone) implies many interesting primitives such as threshold signature, aggregate signature, multi-signature, ring signature, threshold ring signature, etc; more importantly, our efficient construction with

[3]They are a kind of special threshold signature that supports the *dynamic choice* of thresholds for each time of signature generation.

different instantiations of concrete predicates can improve the state of the art of all those primitives. More specifically, when using our dynamic threshold aggregate signature, we can directly yield the first multi-signature, aggregate signature, graded signature, and threshold signature with both $O(\log n)$ communication and verification cost, while the state-of-the-art construction of them (from standard assumptions except zk-SNARKs or AGM directly) are all having linear costs. See Table 6.2.

TABLE 6.2. Advancing relevant primitives.$^{\diamond}$

Primitives	Commun. Cost	Verify Cost.	Generation Cost.
Multi-Sig. [129]	$O(n)^{\dagger}$	$O(n)$	$O(n)$
Agg-Sig. [35]	$O(n)^{\ddagger}$	$O(n)$	$O(n)$
Graded Sig. [124, 125]	$O(n)$	$O(n)$	$O(n)$
Thld-ring Sig. [141]	$O(\log n)$	$O(n)$	$O(n)$
Using our PAS*	$O(\log n)$	$O(\log n)$	$O(n)$

$^{\diamond}$ In the comparison, we restrict only to the single message case in our PAS. If there are k messages to be signed, all others have a *multiplicative* factor k, while we only have an *additive* factor.

†,‡ Although their signatures can be aggregated, the signers' identities should also be transmitted, which leads to linear communication cost, except recent ones [138, 137] that rely on zk-SNARKs or AGMs directly.

* The last row means using our PAS with dynamic threshold as Δ, it implies the above primitives and advances their performance.

For multiple users and multiple messages, the combiner generates a PAS with threshold t_j for each m_j. It implies the aggregate signature and hides the signers' identities. For multiple users and one message, a PAS with threshold t implies that t different signers have signed on the message. It implies the threshold signature with transparent setup and dynamic threshold t and threshold ring signature with threshold t. It also naturally implies the multi-signature with t signers and the graded signature, which indicates there are t different signers. When there is only one signer and one message, it implies the ring signature and attribute-based signature. The signer himself works as the combiner and shows the threshold is 1 with the proof of satisfying the predicate. It is equivalent to validating the signer's attribute. More details can be found in Sec. 6.6.

6.3 Predicate Aggregate Signatures

In this section, we formalize the predicate aggregate signatures and establish the security model for this concept. Predicate aggregate signatures enable users to sign multiple messages according to some predefined public policy, and these individual signatures can be aggregated by a combiner, preserving the anonymity of the signers. The resulting aggregate signature discloses only a brief description of the involved signers (e.g., count of signers for each message, total weight of signers, etc) and provides assurance that these signers and the description satisfy the specified policy denoted by a public predicate function.

This notion addresses the need for efficient and privacy-preserving signature schemes that allow for the signing of multiple messages while ensuring adherence to a given predicate. The security model encompasses the privacy of signers and the unforgeability of the predicate aggregate signature.

6.3.1 Syntax

In general, there are three parties in the system: signers who sign on the message; the combiner, who generates a predicate aggregate signature with a public description of the involved signers and proves that these signers and the description satisfy a public predicate; the verifier who verifies the correctness of the predicate aggregate signature.

- $\mathsf{Setup}(1^\lambda)$: On the security parameter λ, the system public parameters pp are generated. The message space is set as $M = \{m_j\}_{j\in[k]}$. There is a public policy Ω which decides the computation rule of the signers description Δ and the predicate function P_Ω.

  It also includes the key generation of users. Each user u_i generates his secret key sk_i and public key pk_i pair and broadcasts the public key. The combiner (or any other parties) collects the public keys and publishes the aggregation key ak and verification key vk which contains P_Ω.
- $\mathsf{ParSign}(sk_i, m_j)$: For a message m_j chosen from M, the user i signs on it using his secret key sk_i and sends (pk_i, m_j, σ_{ij}) to the combiner.

- $\mathsf{ParVrfy}(pk, m_j, \sigma)$: On receiving (pk_i, m_j, σ), anyone can verify it.
- $\mathsf{Combine}(ak, \{S_j\}_{j\in[k]}, \{\{\sigma_{ij}\}_{i\in S_j}\}_{j\in[k]}, \{m_j\}_{j\in[k]})$: When receiving sets of signatures $\{\{\sigma_{ij}\}_{i\in S_j}\}_{j\in[k]}$ on the message m_j from different signers w.r.t. index sets $\{S_j\}_{j\in[k]}$ (called signer sets) where each $S_j \subseteq [n]$, the combiner generates a signature Σ for the message set $M = \{m_j\}_{j\in[k]}$ with corresponding description Δ of these signer sets. It also proves that the signer sets and Δ satisfy the public predicate P_Ω decided by some policy Ω, i.e., $P_\Omega(S_1, ..., S_k, \Delta) = 1$.
- $\mathsf{Verify}(vk, M, \Delta, \Sigma)$: Given the verification key vk, messages $M = \{m_j\}_{j\in[k]}$, description Δ, the PAS signature Σ, anyone can check the validness by running $\mathsf{Verify}(vk, M, \Delta, \Sigma)$ and outputs one bit $b \in \{0, 1\}$ indicating if it is valid.

REMARK 14. *Δ is the description of signers in $S_1, ..., S_k$ whose partial signatures are used to generate the final PAS signature. It is computed via a deterministic function F specified in the policy Ω from the signer sets: $\Delta = F(S_1, ..., S_k)$. Note that it cannot display the signer identities plainly, since it ruins the anonymity directly. Although it is computed deterministically, in many cases, it leaks very little information about the signers. For example, it could be the size of each signer set, the combined weight of signers within each signer set, or simply demonstrating that the number exceeds a certain minimum threshold.*

REMARK 15. *P_Ω is the predicate decided by the public policy Ω which takes the signer sets and a description as input. $P_\Omega(S_1, ..., S_k, \Delta) = 1$ indicates that $S_1, ..., S_k, \Delta$ satisfy the rule according to Ω.*

6.3.2 Model

Correctness If enough valid signatures under different public keys are used to produce the signature Σ on the messages m_j for $j \in [k]$, the description Δ and the signer sets satisfy the public predicate $P_\Omega(S_1, ..., S_k, \Omega) = 1$, then the verification for (M, Δ, Σ) always outputs 1.

Anonymity The signer identities are hidden from the public. The PAS signature only discloses a description of the signer sets and whether they satisfy the predicate according to the public policy. Given any two valid signatures Σ_0, Σ_1 with the *same* descriptions and predicates from

different signer sets on the same messages set M, they are indistinguishable even to some of these signers.

Unforgeability The adversary cannot generate a valid signature on multiple messages if it does not have enough signatures from different signers on each message, or the signer sets and the description do not satisfy the predicate.

Oracles We define the following oracles to model the adversary's ability. There is an honest user table HU, a corrupted user table CU and a queried message table QM which are initialized as empty.

- $\mathbf{add}(i)$: Add a new user u_i to the system. If i has not been queried before, run the key generation algorithm $(pk^*, sk^*) \leftarrow \mathsf{KeyGen}$, set $(pk_i, sk_i) = (pk^*, sk^*)$ and output pk_i. Add (u_i, pk_i, sk_i) to the honest user table HU.
- $\mathbf{corrupt}(pk_i)$: Corrupt an honest user in the system. If $pk_i \in \mathsf{HU}$, output sk_i, delete (u_i, pk_i, sk_i) from HU and add (u_i, pk_i, sk_i) to CU.
- $\mathbf{sign}(pk_i, m)$: If $pk_i \notin \mathsf{HU}$, ignore it. Otherwise, run $\sigma \leftarrow \mathsf{ParSign}(sk_i, m)$ and add (pk_i, m) to QM, output σ.

Anonymity This property ensures that signer identities are hidden from the public. Only the combiner knows the signer identities. Others just know the description of the signers and the signer sets with the description satisfy the public predicate. Formally speaking, given any two valid signatures Σ_0, Σ_1 from different signer sets on the same messages M with the same description Δ and both satisfy the predicate, nobody can distinguish them except the combiner. In the anonymity definition, even the signers will not be able to distinguish two PAS signatures for the maliciously chosen messages with same thresholds. The anonymity experiment $\mathrm{Exp}^{\mathsf{anony}}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is formalized as follows.

- $\mathcal{A}$ receives the public parameters including the description function F and public predicate P_Ω specified by the policy Ω.
- $\mathcal{A}$ adds users to the system, it can query signatures of any signers on any messages and even corrupt all users. The global aggregation key ak and verification key vk are setup and given to $\mathcal{A}$.

- $\mathcal{A}$ chooses a set of messages $M = \{m_j\}_{j\in[k]}$, and two sets of index sets $S^0 = \{S_j^0\}_{j\in[k]}$ and $S^1 = \{S_j^1\}_{j\in[k]}$ where each $S_j^0, S_j^1 \subseteq [n]$. Then it generates partial signatures $\sigma_{u,j} \leftarrow \mathsf{ParSign}(sk_u, m_j)$, $\sigma_{v,j} \leftarrow \mathsf{ParSign}(sk_v, m_j)$ for all $j \in [k], u \in S^0$ and $v \in S^1$ and let $D_0 = \{\{\sigma_{u,j}\}_{u\in S^0}\}_{j\in[k]}, D_1 = \{\{\sigma_{v,j}\}_{v\in S^1}\}_{j\in[k]}$. It sends (M, S^0, S^1, D_0, D_1) to $\mathcal{C}$.
- $\mathcal{C}$ computes $\Delta_0 = F(S^0)$, $\Delta_1 = F(S^1)$ and checks these partial signatures. It aborts, if there exists any invalid partial signature or $S^0 = S^1$ or $\Delta_0 \neq \Delta_1$ or $P_\Omega(S^0, \Delta_0) \neq 1$ or $P_\Omega(S^1, \Delta_1) \neq 1$. Otherwise, continue.
- $\mathcal{C}$ randomly chooses one bit $b \leftarrow \{0, 1\}$ and sends $\mathcal{A}$ a predicate aggregate signature Σ with M generated from signatures of D_b by running $\Sigma \leftarrow \mathsf{Combine}(ak, S^b, D_b, M)$.
- $\mathcal{A}$ outputs a guess b'.
- Outputs 1 if $b' == b$, otherwise, outputs 0.

DEFINITION 19. *A predicate aggregate signature is anonymous if any PPT adversary in the* $\mathrm{Exp}^{\mathsf{anony}}$ *can only guess the bit correctly with probability negligibly close to* $\frac{1}{2}$*, i.e.,* $|\Pr[\mathrm{Exp}^{\mathsf{anony}}(\mathcal{A}, \lambda) = 1] - \frac{1}{2}| \leq$*negl(*$\lambda$*).*

Unforgeability This property ensures that given the public policy, any adversary cannot generate a valid signature on multiple messages if it does not have enough signatures on the messages or the signer sets and their description do not satisfy the predicate.

It contains two properties: (1). $\mathcal{A}$ can not produce a valid PAS signature which contains an honest signer who has never signed on that message. (2). All signer sets w.r.t. the messages must adhere to the specified predicate. It is infeasible for $\mathcal{A}$ to generate a valid PAS signature with a signer sets description but the signer sets and the description are unsatisfied for the predicate.

Note that due to the anonymity requirement, the signer identities are hidden and only their description is shown. So it is hard to decide whether $\mathcal{A}$ has broken the predicate satisfaction property. To address this dilemma, we introduce an extractor $\mathcal{E}$ that has the ability to reveal the signers' identities for each message from the predicate aggregate signature. It is inspired by the knowledge extractor for the knowledge soundness in zero-knowledge proof of knowledge.

Formally speaking, the unforgeability experiment $\mathrm{Exp}^{\mathsf{unforge}}$ works as follows:

- $\mathcal{A}$ receives the public parameters and the public predicate.
- $\mathcal{A}$ can add users to the system then gets the aggregation key ak and verification key vk.
- $\mathcal{A}$ is an adaptive adversary, it can interact with $\mathcal{E}$ via querying oracles. It can then query signatures of any signers on any messages and corrupt users.
- $\mathcal{A}$ outputs a tuple (M, Δ, Σ).
- On a valid (M, Δ, Σ), $\mathcal{E}$ outputs the signers' identities $S_1, ..., S_k$.

$\mathcal{A}$ wins and the experiment outputs 1 only if (M, Δ, Σ) is valid w.r.t. *vk* and at least one of the following conditions is satisfied:

- $\exists\, i_j \in S_j$, such that $i_j \in \mathsf{HU}$ and $(pk_{i_j}, m_j) \notin \mathsf{QM}$;
- $P_\Omega(S_1, ..., S_k, \Delta) \neq 1$.

The first condition means $\mathcal{A}$ has never queried the **corrupt** oracle on pk_{i_j} or **sign** oracle on (pk_{i_j}, m_j). It models that $\mathcal{A}$ generates a valid PAS signature without enough valid partial signatures.

DEFINITION 20. *A predicate aggregate signature is unforgeable if any PPT adversary in the above experiment can only win with negligible probability, i.e.,* $\Pr[\mathrm{Exp}^{\mathsf{unforge}}(\mathcal{A}, \lambda) = 1] \leq$*negl(*$\lambda$*)*

6.4 Constructions

In this section, we give the construction of the predicate aggregate signature. Formally speaking, the combiner aims to prove the knowledge of signatures and signers satisfying the following relation:

$$\begin{aligned} R_{\mathsf{PAS}} = \{&pk_1, \ldots, pk_n, m_1, \ldots, m_k, \Omega, \Delta; \{\sigma_{i1}\}_{i \in S_1}, \ldots, \{\sigma_{ik}\}_{i \in S_k} : \\ &\mathsf{ParVrfy}(pk_i, m_j, \sigma_{ij}) = 1 \; \forall i \in S_j, j \in [k]; \\ &S_j \subseteq [n], \forall j \in [k]; P_\Omega(S_1, ..., S_k, \Delta) = 1\} \end{aligned}$$

where $pk_1, \ldots, pk_n$ are all the public keys, $m_1, \ldots, m_k$ are the candidate messages, S_j contains the indices of users who have signed on the message m_j, σ_{ij} is the signature from user i on message m_j. P_Ω is the predicate function decided by the public policy Ω.

The predicate function can be very simple that outputs 1 as long as the description Δ is correct and there is no other requirements on the policy. For any general policy that can be described by an arithmetic circuit, the predicate can be converted into a circuit and proved using the efficient zero-knowledge argument for arbitrary arithmetic circuits which have been studied in [142, 126, 36].

In this section, we mainly consider the anonymous reputation system with the rate-once policy as a non-trivial example. Here the description is the number of signers in each subset ($\Delta = \{t_j\}_{j\in[k]}$ where $t_j = |S_j|$) and all users can only sign once even for different messages. It means the signer sets for all messages are disjoint. We define the public predicate as follows:

$$P_\Omega(S_1, ..., S_k, \Delta) = \begin{cases} 1, & \text{if } \Delta = \{t_j\}_{j\in[k]}, S_j \subseteq [n], |S_j| = t_j, \forall j \in [k]; \\ & \wedge S_{j_0} \cap S_{j_1} = \emptyset,\ \forall j_0, j_1 \in [k], j_0 \neq j_1 \\ 0, & \text{otherwise.} \end{cases} \tag{6.1}$$

6.4.1 Construction Overview

Our starting point is the pairing-based BLS aggregate signature (see Sec. 2.3). Let n be the number of users in the system. On receiving a set of partial signatures σ_{ij} on each message $m_j \in M$ from signer $i \in [n]$, the combiner generates the aggregate signature $\hat{\sigma}$ and publishes it with the index set $S_j \in [n]$ of signers on message m_j for verification. The verifier computes the verification key for m_j w.r.t. S_j: $\widehat{pk}_j = \Pi_{i\in S_j} pk_i$ for $j \in [k]$. Subsequently, it verifies the validity of $\hat{\sigma}$.

An intuitive idea is letting the combiner also compress the public keys into a compact version which can be used for verifying the aggregated signature. To protect the privacy of signers, it also adds *blind factors* to the compressed public keys and respective aggregated signatures.

The crux of our construction now revolves around proving the correctness of the compressed public keys without using linear descriptions while still enabling duplication checks.

We start from the single message case. Given all the public keys $\boldsymbol{pk} = (pk_1, ..., pk_n) \in \mathbb{G}_2^n$ and a set $S \subseteq [n]$, let $\boldsymbol{pk}_S = \{pk_i\}_{i \in S} \subseteq \{pk_i\}_{i \in [n]}$ to denote a subset of all public keys w.r.t. *S*. The blinded aggregation of public keys of $\boldsymbol{pk}_S$ is: $\widehat{pk}_S = \Pi_{i \in S} pk_i \cdot \tilde{g}^{r_S}$ where r_S is the blind factor. Note that *S* determines a binary vector $\boldsymbol{b}_S = (b_1, ..., b_n)$, $b_i = 1$ if $i \in S$, otherwise, $b_i = 0$.

The combiner (or prover $\mathcal{P}$) computes a commitment B on $\boldsymbol{b}_S$ and proves that the committed $\boldsymbol{b}_S$ is a binary vector whose Hamming weight is t. Then $\mathcal{P}$ proves that $\widehat{pk}_S$ can be expressed in the form of $\boldsymbol{pk}^{\boldsymbol{v}} \cdot \tilde{g}^r$ where $\boldsymbol{v}$ is same as the $\boldsymbol{b}_S$ in B and it knows the blind factor r. It means that $\widehat{pk}_S$ contains *t* different signers. Combined with an aggregate signature $\hat{\sigma}$ and a message m, the valid tuple $(\widehat{pk}_S, m, \hat{\sigma})$, the verifier can be convinced that the message has been signed by *t* different users.

For multiple messages $m_1, ..., m_k$ case, the index set of signers who have signed on m_j is S_j and their aggregated public keys are $\widehat{pk}_j$ for $j \in [k]$. Here $\mathcal{P}$ proves the knowledge of corresponding binary vectors $\boldsymbol{b}_j$ whose Hamming weight is t_j and the knowledge of blind factor r_j and the signer sets $S = \{S_1, ..., S_k\}$ and thresholds $T = \{t_1, ..., t_k\}$ satisfy the public predicate P_Ω.

Considering the rate-once policy in the anonymous reputation system as a concrete example, the predicate is denoted by $P_\Omega(S, T) := |S_j| = t_j, \forall j \in [k] \wedge S_{j_0} \cap S_{j_1} = \emptyset,\ \forall j_0, j_1 \in [k], j_0 \neq j_1$. Due to the binary feature, we observe that proving subsets disjoint can be achieved by demonstrating that the summation of all binary vectors is still binary with a Hamming weight $t = \sum_{j \in [k]} t_j$.

In summary, we formulate this process in the following relation:

$$\begin{aligned} R_1 = \{\{\widehat{pk}_j, m_j, t_j\}_{j=1}^k, \hat{\sigma}; \boldsymbol{b}_1, \ldots, \boldsymbol{b}_k, r_1, ..., r_k : \\ e(\hat{\sigma}, \tilde{g}) = e(H(m_1), \widehat{pk}_1) \cdots e(H(m_k), \widehat{pk}_k) \\ \wedge\ \widehat{pk}_j = \boldsymbol{pk}^{\boldsymbol{b}_j} \cdot \tilde{g}^{r_j}\ \wedge\ \boldsymbol{b}_j \in \{0,1\}^n\ \wedge\ t_j = \langle \mathbf{1}^n, \boldsymbol{b}_j \rangle, \forall j \in [k] \\ \wedge\ \boldsymbol{b} = \Sigma_{j\in[k]} \boldsymbol{b}_j \in \{0,1\}^n\} \end{aligned} \tag{6.2}$$

Strawman scheme from Bulletproofs. In general, the public keys are group elements. It can be integrated with the public parameters of Bulletproofs which are also group elements. Then we use Pederson commitment to commit the binary vector and prove the correctness of the aggregated public keys. The combiner has $\widehat{pk} = \Pi_{i=1}^n pk_i^{b_i} \cdot g^r$ and generates $B = \Pi_{i=1}^n g_i^{b_i} \cdot h^r$. It generates binary proof on B with parameter $(g_1, ..., g_n, h)$ and another binary proof on $B \cdot \widehat{pk}$ with parameter $(g_1 \cdot pk_1, ..., g_n \cdot pk_n, h \cdot g)$. It implies that $\widehat{pk}$ shares the same binary vector and randomness in B, so it is well-formed. Here the binary proof can be constructed from Bulletproofs [126] with logarithmic size. The final construction comes with almost the same cost.

For multiple k messages case, instead of generating *k* individual proofs, we generate a proof for *k* values on the knowledge of *n*-length binary vectors. It aggregates *k* proofs into one proof of a kn-length binary vector ($k \cdot n$ bits) via a random challenge value z and Schwartz-Zippel lemma. It is similar with the aggregated range proofs of Sec. 4.3 in [126]. As a result, we achieve a communication cost of $O(k + \log n + \log k)$ which is just logarithmic to the parameter *n*. However, the verification time scales linearly with the total bit length, denoted as $O(k \cdot n)$. This places a significant burden on the verifier, prompting us to explore more efficient construction methods that offer sublinear verification times.

Construction with logarithmic verifier. Daza et al [36] improves Bulletproofs to achieve both logarithmic size and verification time. However, it cannot be applied to our setting in the same way as above. Since it relies on structured parameters that are incompatible with the randomly chosen public keys. Integrating the public keys with these parameters would violate their structure and render the technique useless.

Plain case without anonymity. We start from the single message case without anonymity. We assume that the proof of possession has been done to prove the knowledge of secret key for each public key. We aim to prove that a given $\widehat{pk} \in \mathbb{G}_2$ can be expressed in the form of $\boldsymbol{pk}^{\boldsymbol{b}}$ where $\boldsymbol{b}$ is a binary vector. Note that the pairing $e(g, \boldsymbol{pk}^{\boldsymbol{b}}) = \sum_{i=1}^{n} e(g, pk_i^{b_i}) = \sum_{i=1}^{n} e([b_i]_1, pk_i) = \langle [\boldsymbol{b}]_1, \boldsymbol{pk} \rangle$. Instead of directly proving the form of $\widehat{pk}$, we compute the map $e(g, \widehat{pk})$ at first. By the bilinear property, it is sufficient to prove that the map result is also an inner pairing product between $[\boldsymbol{b}]_1$ and $\boldsymbol{pk}$. To this end, we leverage an inner pairing product (IPP) argument. It asserts that a given element in $\mathbb{G}_T$ is the inner pairing product between two vectors in $\mathbb{G}_1^n$ and $\mathbb{G}_2^n$ which are committed with AFGHO commitments. By imposing a specific structure on the commitment key (similar to [36], as we introduced in Sec. 2.3), the verification time is reduced to logarithmic in relation to n.

The remaining issue is proving the form of these two committed vectors. $\boldsymbol{pk}$ are public, so the commitment can be verified publicly. For $[\boldsymbol{b}]_1$, we develop a binary proof in which the verification time is also logarithmic to n. It proves the committed vector consists of elements which is either $[0]_1$ or $[1]_1$. Thus, $\boldsymbol{b} \in \{0, 1\}^n$.

Anonymous case. Now we consider the anonymous setting with a blind factor r: $\widehat{pk} = \boldsymbol{pk}^{\boldsymbol{b}} \cdot \tilde{g}^r$ and $e(g, \widehat{pk}) = \langle ([\boldsymbol{b}]_1, g), (\boldsymbol{pk}, \tilde{g}^r) \rangle$. When we attempt to follow the above method, some issues happen. The combiner needs to take additional work on proving: (i) the last position of the binary vector is 1; (ii) the public keys are committed together with a random element $\tilde{g}^r$ and he knows r. The first task requires another invocation of binary proof and it is unclear how to prove the discrete logarithm of a committed group element without leaking r or $\tilde{g}^r$.

By the bilinear property, we also have $e(g, \widehat{pk}) = \langle ([\boldsymbol{b}]_1, g^r), (\boldsymbol{pk}, \tilde{g}) \rangle$. Even though the latter vector $(\boldsymbol{pk}, \tilde{g})$ is publicly known, the situation remains challenging because the binary proof mechanism does not inherently support proving the presence of a random value within the vector. Another observation is that $e(g, \widehat{pk}) = \langle [\boldsymbol{b}]_1, \boldsymbol{pk} \rangle + r \cdot e_T$. One may consider to extract $r \cdot e_T$ and prove the knowledge of r. But it would leak $\boldsymbol{b}$ and violate the anonymity.

To mitigate these issues, we commit $\boldsymbol{b}$ and r with different commitment keys and generate the proofs for them separately. Afterward, by combining these two commitments, we can

ascertain the presence of both a binary vector and a blind factor in designated positions. Then we can apply the inner pairing product argument on these vectors and proves the well-formedness of the blinded aggregated public key.

For k messages setting with k aggregated public keys, the aggregation technology on kn-length binary vector can also be applied as we outlined in the strawman scheme.

In the next section, we introduce our inner pairing product argument and binary proof with logarithmic communication cost and logarithmic verification time. They can be rendered non-interactive by applying the Fiat-Shamir heuristic [88]. In Sec. 6.4.3, we present our PAS signature scheme with the rate-once policy in the anonymous reputation system.

6.4.2 Succinct Proofs with Logarithmic Verifier

Inner pairing product argument. We consider an argument for inner pairing product between two vectors $\boldsymbol{v}_i \in \mathbb{G}_i^n$ committed with structured AFGHO commitments with generators $(\mathsf{ck}_{3-i}, e_H) \in \mathbb{G}_i^n \times \mathbb{G}_T$ for $i \in \{1, 2\}$ where $e_H = e(g_H, \tilde{g}), g_H \leftarrow\!\!\$\ \mathbb{G}_1$. In this section, we assume that the dimension *n* is a power of 2. If necessary, it is straightforward to add padding to the inputs to ensure this condition is met. Formally, we define a language:

$$
\begin{aligned}
(pp, P, C_1, C_2, [r]_1, \mathsf{ck}_1, \mathsf{vk}_1, [s]_2, \mathsf{ck}_2, \mathsf{vk}_2, e_H) \in \mathcal{L}_{\mathsf{IPP}} \Leftrightarrow \\
&([r]_1, \mathsf{ck}_1, \mathsf{vk}_1) \in \mathcal{L}^1_{\mathsf{Com}} \wedge ([s]_2, \mathsf{ck}_2, \mathsf{vk}_2) \in \mathcal{L}^2_{\mathsf{Com}} \wedge \\
&\exists \boldsymbol{v}_1 \in \mathbb{G}_1^n, \boldsymbol{v}_2 \in \mathbb{G}_2^n, r_{C_1}, r_{C_2}, r_P \in \mathbb{Z}_q : \\
&C_1 = \langle \boldsymbol{v}_1, \mathsf{ck}_2 \rangle + r_{C_1} \cdot e_H \wedge C_2 = \langle \mathsf{ck}_1, \boldsymbol{v}_2 \rangle + r_{C_2} \cdot e_H \wedge \\
&P = \langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle + r_P \cdot e_H
\end{aligned}
$$

Common input: $(pp, [r]_1, [s]_2, \mathsf{vk}_1, \mathsf{vk}_2, e_H), P, C_1, C_2 \in \mathbb{G}_T$

$\mathcal{P}$ input: $(\mathsf{ck}_1, \mathsf{ck}_2), \boldsymbol{v}_1 \in \mathbb{G}_1^n, \boldsymbol{v}_2 \in \mathbb{G}_2^n, r_{C_1}, r_{C_2}, r_P \in \mathbb{Z}_q$

Statement: $(pp, P, C_1, C_2, [r]_1, \mathsf{ck}_1, \mathsf{vk}_1, [s]_2, \mathsf{ck}_2, \mathsf{vk}_2, e_H) \in \mathcal{L}_{\mathsf{IPP}}$

$\mathcal{P}$ and $\mathcal{V}$ proceed the protocol Π_{IPP} as follows:

If $n = 1$:

$C_1 = e(v_1, \mathsf{ck}_2) + r_{C_1} \cdot e_H, C_2 = e(\mathsf{ck}_1, v_2) + r_{C_2} \cdot e_H, P = e(v_1, v_2) + r_P \cdot e_H$

- $\mathcal{P}$ samples $s_1 \leftarrow\!\$\, \mathbb{G}_1, s_2 \leftarrow\!\$\, \mathbb{G}_2, r_{D_1}, r_{D_2}, r_{T_1}, r_{T_2} \leftarrow\!\$\, \mathbb{Z}_q$ and computes:

$D_1 = e(s_1, \mathsf{ck}_2) + r_{D_1} \cdot e_H, D_2 = e(\mathsf{ck}_1, s_2) + r_{D_2} \cdot e_H$

$T_1 = e(s_1, v_2) + e(v_1, s_2) + r_{T_1} \cdot e_H, T_2 = e(s_1, s_2) + r_{T_2} \cdot e_H$

- $\mathcal{P}$ sends D_1, D_2, T_1, T_2 to $\mathcal{V}$

- $\mathcal{V}$ replies with $c \leftarrow\!\$\, \mathbb{Z}_q^*$

- $\mathcal{P}$ computes and sends:

$u_1 = v_1 + c \cdot s_1, u_2 = v_2 + c \cdot s_2,$

$r_1 = r_{C_1} + c \cdot r_{D_1}, r_2 = r_{C_2} + c \cdot r_{D_2}, r_3 = r_P + c \cdot r_{T_1} + c^2 \cdot r_{T_2}$

- $\mathcal{V}$ accepts if:

$C_1 + c \cdot D_1 = e(u_1, \mathsf{ck}_2) + r_1 \cdot e_H \wedge$

$C_2 + c \cdot D_2 = e(\mathsf{ck}_1, u_2) + r_2 \cdot e_H \wedge$

$P + c \cdot T_1 + c^2 \cdot T_2 = e(u_1, u_2) + r_3 \cdot e_H$

Else $n > 1$, the reduce procedure:

- $\mathcal{P}$ samples $r_{1\ell}, r_{2\ell}, r_{P\ell}, r_{1r}, r_{2r}, r_{Pr} \leftarrow\!\$\, \mathbb{Z}_q$ and computes:

$C_{1\ell} \leftarrow \langle \boldsymbol{v}_{1\ell}, \mathsf{ck}_{2r} \rangle + r_{1\ell} \cdot e_H, C_{2\ell} \leftarrow \langle \mathsf{ck}_{1r}, \boldsymbol{v}_{2\ell} \rangle + r_{2\ell} \cdot e_H,$

$P_\ell \leftarrow \langle \boldsymbol{v}_{1r}, \boldsymbol{v}_{2\ell} \rangle + r_{P\ell} \cdot e_H,$

$C_{1r} \leftarrow \langle \boldsymbol{v}_{1r}, \mathsf{ck}_{2\ell} \rangle + r_{1r} \cdot e_H, C_{2r} \leftarrow \langle \mathsf{ck}_{1\ell}, \boldsymbol{v}_{2r} \rangle + r_{2r} \cdot e_H,$

$P_r \leftarrow \langle \boldsymbol{v}_{1\ell}, \boldsymbol{v}_{2r} \rangle + r_{Pr} \cdot e_H$

- $\mathcal{P}$ sends $C_{1\ell}, C_{2\ell}, P_\ell, C_{1r}, C_{2r}, P_r$

- $\mathcal{V}$ replies with $c \leftarrow\$ \mathbb{Z}_q^*$

- $\mathcal{P}$ computes:

$\boldsymbol{v}_1' \leftarrow \boldsymbol{v}_{1\ell} c + \boldsymbol{v}_{1r} c^{-1} \in \mathbb{G}_1^{n'}$, $\boldsymbol{v}_2' \leftarrow \boldsymbol{v}_{2\ell} c^{-1} + \boldsymbol{v}_{2r} c \in \mathbb{G}_2^{n'}$,

$r'_{C_1} = r_{C_1} + r_{1\ell} \cdot c^2 + r_{1r} \cdot c^{-2}$, $r'_{C_2} = r_{C_2} + r_{2\ell} \cdot c^{-2} + r_{2r} \cdot c^2$,

$r'_P = r_P + r_{P\ell} \cdot c^{-2} + r_{Pr} \cdot c^2$

$\mathsf{ck}'_1 = c\mathsf{ck}_{1\ell} + c^{-1}\mathsf{ck}_{1r}$, $\mathsf{ck}'_2 = c^{-1}\mathsf{ck}_{2\ell} + c\mathsf{ck}_{2r}$,

$[r']_1 \leftarrow \{\mathsf{ck}'_1\}_1$, $[s']_2 \leftarrow \{\mathsf{ck}'_2\}_1$ (picks the first elements of $\mathsf{ck}'_1, \mathsf{ck}'_2$)

- $\mathcal{P}$ sends $[r']_1$, $[s']_2$ to $\mathcal{V}$.

- $\mathcal{V}$ checks the following equations and aborts if any fails:

$$e([r']_1 - c[r]_1, [1]_2) = e(c^{-1}[r]_1, [x_\nu]_2),\ e([1]_1, [s']_2 - c^{-1}[s]_2) = e([y_\nu]_1, c[s]_2)$$

Update $\mathsf{vk}'_1 = [\boldsymbol{x}']_2 \leftarrow ([x_i]_2)_{i\in[\nu-1]}$ and $\mathsf{vk}'_2 = [\boldsymbol{y}']_1 \leftarrow ([y_i]_1)_{i\in[\nu-1]}$

- Both compute

$C'_1 \leftarrow c^2 C_{1\ell} + C_1 + c^{-2} C_{1r}$, $C'_2 \leftarrow c^{-2} C_{2\ell} + C_2 + c^2 C_{2r}$,

$P' = c^{-2} P_\ell + P + c^2 P_r$,

- The reduced statement is $(pp, P', C'_1, C'_2, [r']_1, \mathsf{ck}'_1, \mathsf{vk}'_1, [s']_2, \mathsf{ck}'_2, \mathsf{vk}'_2, g_H, e_H) \in \mathcal{L}_{\mathsf{IPP}}$ with the new witnesses $(\boldsymbol{v}'_1, \boldsymbol{v}'_2, r'_1, r'_2, r'_P)$.

THEOREM 9. *The protocol presented is a Public Coin, HVSZK, interactive argument of knowledge for the relation $\mathcal{L}_{\mathsf{IPP}}$ with $O(\log n)$ round complexity, $O(n)$ prover complexity, and $O(\log n)$ communication and verification complexity under the SXDH and DPair-ML assumptions.*

PROOF. The proof follows a similar structure to those found in [126, 36, 127].

Public Coin property is immediate. *HVSZK* holds as all messages from $\mathcal{P}$ to $\mathcal{V}$ are uniformly random elements of $\mathbb{G}_T$, so can be simulated easily.

Completeness: Each reduction round results in a valid reduced statement, and this can be shown by proving that the prover and verifier compute the same key. Once this key equivalence is established, we can proceed with the same argumentation as in the case of uniform keys. Completeness holds from substituting the definition of $\mathcal{P}$'s witness into the constraints.

Witness extended emulation: For $n = 1$, the proof is similar to that of Thm 1. in [127]. For $n > 1$, to achieve witness extended emulation, it is necessary to provide a proof that for each round, the witness can be extracted, which includes the commitment key and the corresponding commitment openings. The process of extracting the commitment keys is outlined as follows. Once the commitment keys are obtained, the argumentation can proceed similarly to the approach presented in Bulletproofs: the extractor can extract a valid witness $\boldsymbol{v}_1, \boldsymbol{v}_2$ or break the DPair-ML Assumption.

If we have two accepting transcripts from the prover for different challenges c, we can demonstrate that, the unique valid commitment keys $\mathsf{ck}_1 = [\mathbf{r}]_1$ and $\mathsf{ck}_2 = [\mathbf{s}]_2$ can be extracted.

Let $[\mathbf{r}'_b]_1 = c_b[\mathbf{r}_\ell]_1 + c_b^{-1}[\mathbf{r}_r]_1, b \in \{0, 1\}$ be the new commitment keys for different challenges c_0, c_1, the $[\mathbf{r}_\ell]_1$ and $[\mathbf{r}_r]_1$ can be obtained by linear combination. We show that they are the commitment key.

Since the transcripts are accepted, we have that $[r'_{2^{i-1}+j}]_1 = x_i[r'_j]_1$ which means $[r_{2^{i-1}+j}]_1 = x_i[r_j]_1$ and $[r_{2^{\nu-1}+2^{j-1}+j}]_1 = x_i[r_{2^{\nu-1}+j}]_1$ for all $i \leq \nu - 1, j \leq 2^i$. So $[\mathbf{r}_\ell]_1$ and $[\mathbf{r}_r]_1$ are valid commitment keys w.r.t. $[x_1]_2, ..., [x_{\nu-1}]_2$. The pairing test convinces that $[r'_b]_1 = c_b[r]_1 + c_b^{-1}x_\nu[r]_1 = c_b[r_{\ell,1}]_1 + c_b^{-1}[r_{r,1}]_1$. So $[r_{\ell,1}]_1 = [r]_1$ and $[r_{r,1}]_1 = x_\nu[r]_1$, which means the extracted keys are the unique keys determined by $x_1, ..., x_\nu$. The $\mathsf{ck}_2 = [\mathbf{s}]_2$ can be argued in the same way. Thereafter, the extractor works in a similar way as in [126, 127] to extract $\boldsymbol{v}_1, \boldsymbol{v}_2, r_{C_1}, r_{C_2}, r_p$.

□

Binary proofs with logarithmic verifier. We consider an argument for a binary vector $\boldsymbol{b} = (b_1, \ldots, b_n) \in \{0,1\}^n$ with Hamming weight t. This binary vector is equivalent to a vector $\boldsymbol{v} \in \mathbb{G}_1^n$ where each element is either $[0]_1$ or $[1]_1$, and the number of $[1]_1$ is precisely t. Formally, we define a language:

$$\begin{aligned}(C, [r]_2, \mathsf{ck}_2, \mathsf{vk}_2, e_H, t) \in \mathcal{L}_{\mathsf{Bin}} \Leftrightarrow \\ ([r]_2, \mathsf{ck}_2, \mathsf{vk}_2) \in \mathcal{L}^1_{\mathsf{Com}} \wedge \\ \exists \boldsymbol{b} \in \{0,1\}^n, r_C \in \mathbb{Z}_q, s.t.: \\ C = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2 \rangle + r_C \cdot e_H \wedge \langle \mathbf{1}^n, \boldsymbol{b} \rangle = t_j,\end{aligned}$$

$\mathcal{P}$ proves that $\langle \mathbf{1}^n, \boldsymbol{b} \rangle = t \wedge \boldsymbol{b} \circ \boldsymbol{b'} = \mathbf{0}^n \wedge \boldsymbol{b'} = \boldsymbol{b} - \mathbf{1}^n$. Using random $y, \tau \in \mathbb{Z}_q^*$ from $\mathcal{V}$, these constraints can be re-written as:

$$\langle \boldsymbol{b} - \tau \cdot \mathbf{1}^n, \boldsymbol{y}^n \circ (\boldsymbol{b'} + \tau \cdot \mathbf{1}^n + \tau^2 \cdot \mathbf{1}^n) \rangle = \tau^2 \cdot t + \delta(y, \tau)$$

where $\delta(y, \tau) = (\tau - \tau^2) \cdot \langle \mathbf{1}^n, \boldsymbol{y}^n \rangle - \tau^3 \langle \mathbf{1}^n, \mathbf{1}^n \rangle \in \mathbb{Z}_q$. Thus the binary proof can be reduced to one inner pairing product argument. Concretely, $\mathcal{P}$ and $\mathcal{V}$ engage in the following protocol Π_{Bin}:

- On input $\boldsymbol{b} \in \{0,1\}^n$, $\mathcal{P}$ computes:

$\boldsymbol{b'} = \boldsymbol{b} - \mathbf{1}^n$, $[\boldsymbol{b}]_1 \in \mathbb{G}_1^n$ and $[\boldsymbol{b'}]_2 \in \mathbb{G}_2^n$, $r_{B_1}, r_{B_2} \leftarrow\!\!\$\, \mathbb{Z}_q$,

commits to $[\boldsymbol{b}]_1$ and $[\boldsymbol{b'}]_2$:

$C = B_1 = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2 \rangle + r_{B_1} \cdot e_H$, $B_2 = \langle \mathsf{ck}_1, [\boldsymbol{b'}]_2 \rangle + r_{B_2} \cdot e_H$,

chooses blinding vectors and commits them:

$\boldsymbol{u}_1, \boldsymbol{u}_2 \leftarrow\!\!\$\, \mathbb{Z}_q^n$, $r_{U_1}, r_{U_2} \leftarrow\!\!\$\, \mathbb{Z}_q$,

$U_1 = \langle [\boldsymbol{u}_1]_1, \mathsf{ck}_2 \rangle + r_{U_1} \cdot e_H$, $U_2 = \langle \mathsf{ck}_1, [\boldsymbol{u}_2]_2 \rangle + r_{U_2} \cdot e_H$,

sends B_1, B_2, U_1, U_2 to $\mathcal{V}$

- $\mathcal{V}$ sends challenges $y, \tau \leftarrow\$ \mathbb{Z}_q^*$ to $\mathcal{P}$

- $\mathcal{P}$ computes $\mathsf{ck}'_1 \leftarrow \mathsf{ck}_1 \circ \boldsymbol{y}^{-n}$,

define the following polynomials:

$$l(X) = \boldsymbol{b} - \tau \cdot \mathbf{1}^n + \boldsymbol{u}_1 \cdot X \in \mathbb{Z}_q^n[X]$$

$$r(X) = \boldsymbol{y}^n \circ (\boldsymbol{b'} + \tau \cdot \mathbf{1}^n + \boldsymbol{u}_2 \cdot X) + \tau^2 \cdot \mathbf{1}^n \in \mathbb{Z}_q^n[X]$$

$$p(X) = \langle l(X), r(X) \rangle = p_0 + p_1 \cdot X + p_2 \cdot X^2 \in \mathbb{Z}_q^n[X]$$

Next, $\mathcal{P}$ needs to convince $\mathcal{V}$ that $p_0 = t \cdot \tau^2 + \delta(y, \tau)$.

- $\mathcal{P}$ chooses $\phi_1, \phi_2 \leftarrow\$ \mathbb{Z}_q^*$ and computes:

$P_1 = p_1 \cdot e(g, \tilde{g}) + \phi_1 \cdot e_H$, $P_2 = p_2 \cdot e(g, \tilde{g}) + \phi_2 \cdot e_H$

- $\mathcal{P}$ sends $P_1, P_2 \in \mathbb{G}_T$ to $\mathcal{V}$

- $\mathcal{V}$ sends $x \leftarrow\$ \mathbb{Z}_q^*$ to $\mathcal{P}$

- $\mathcal{P}$ computes:

$[\boldsymbol{l}]_1 = [l(x)]_1 = [\boldsymbol{b} - \tau \cdot \mathbf{1}^n + \boldsymbol{u}_1 \cdot x]_1 \in \mathbb{G}_1^n$,

$[\boldsymbol{r}]_2 = [r(x)]_2 = [\boldsymbol{y}^n \circ (\boldsymbol{b'} + \tau \cdot \mathbf{1}^n + \boldsymbol{u}_2 \cdot x) + \tau^2 \cdot \mathbf{1}^n]_2 \in \mathbb{G}_2^n$

$P = \langle [\boldsymbol{l}]_1, [\boldsymbol{r}]_2 \rangle \in \mathbb{G}_T$, $\phi_x = \phi_2 \cdot x^2 + \phi_1 \cdot x \in \mathbb{Z}_q$

$\mu_1 = r_{B_1} + r_{U_1} \cdot x, \mu_2 = r_{B_2} + r_{U_2} \cdot x \in \mathbb{Z}_q$

$\mathcal{P}$ sends $[\boldsymbol{l}]_1, [\boldsymbol{r}]_2, P, \phi_x, \mu_1, \mu_2$ to $\mathcal{V}$

$\mathcal{V}$ computes $\mathsf{ck'}_1$ in the same way and computes:

$Q_1 = B_1 + x \cdot U_1 - \tau \cdot \langle [\mathbf{1}]_1, \mathsf{ck}_2 \rangle$, $Q_2 = B_2 + x \cdot U_2 + \tau \cdot \langle \mathsf{ck}'_1, [\boldsymbol{y}^n]_2 \rangle + \tau^2 \cdot \langle \mathsf{ck}'_1, [\mathbf{1}^n]_2 \rangle$

checks whether

$$P + \phi_x \cdot e_H \stackrel{?}{=} (t \cdot \tau^2 + \delta(y, \tau)) \cdot e(g, \tilde{g}) + x \cdot P_1 + x^2 \cdot P_2$$

$$Q_1 \stackrel{?}{=} \langle [\boldsymbol{l}]_1, \mathsf{ck}_2 \rangle + \mu_1 \cdot e_H,\ Q_2 \stackrel{?}{=} \langle \mathsf{ck}'_1, [\boldsymbol{r}]_2 \rangle + \mu_2 \cdot e_H$$

Note that the communication and verification cost are linear to n. To reduce them, $\mathcal{P}$ does not send $[\boldsymbol{l}]_1, [\boldsymbol{r}]_2$ directly, and $\mathcal{V}$ just computes $\mathsf{vk}'_1 \leftarrow \mathsf{vk}_1 \circ \bar{\boldsymbol{y}}$, where $\bar{\boldsymbol{y}} = (1, y^{-1}, \cdots, y^{-2^{\nu-1}})$, rather than ck'_1. To make sure $\mathcal{V}$ still can compute Q_2, $\mathcal{P}$ computes $Y = \langle \mathsf{ck}'_1, [\boldsymbol{y}^n]_2 \rangle = \langle \mathsf{ck}_1, [\mathbf{1}^n]_2 \rangle$ and $\Gamma = \langle \mathsf{ck}'_1, [\mathbf{1}^n]_2 \rangle$.

$\mathcal{P}$ sends Γ to $\mathcal{V}$ and proves its correctness as follows:

Let $\Gamma_\nu = \Gamma$, $\mathsf{ck}'_{1\nu} = \mathsf{ck}'_1$, for $i = \nu - 1$ to 1:

$\mathcal{P}$ sends $M_i = \mathsf{ck}'^{\mathbf{1}^{2^i}}_{1i} \in \mathbb{G}_1$, where $\mathsf{ck}'_{1i} \in \mathbb{G}_1^{2^i}$ is the left half of $\mathsf{ck}'_{1(i+1)}$;

$\mathcal{V}$ aborts if $e(M_i, ([1]_2 + \mathsf{vk}'_{1(i+1)})) \neq \Gamma_{i+1}$,

where $\mathsf{vk}'_{1(i+1)}$ is the $i+1$-th element of vk'_1,

otherwise let $\Gamma_i = e(M_i, [1]_2)$ and continue;

After $\nu - 1$ steps without abort, $\mathcal{V}$ can be convinced that Γ is correct:

$\Gamma = \langle \mathsf{ck}'_1, [\mathbf{1}^n]_2 \rangle$ and its computation time is $O(\nu) = O(\log n)$.

Thus, $\mathcal{V}$ can compute $Q_2 = B_2 + x \cdot U_2 + \tau \cdot Y + \tau^2 \cdot \Gamma$ in $O(\log n)$ time.

Thereafter, $\mathcal{P}$ runs the inner pairing product argument protocol Π_{IPP} with $\mathcal{V}$: $\mathcal{L}_{\mathsf{IPP}}(pp, P, Q_1, Q_2, [r]_1, \mathsf{ck}'_1, \mathsf{vk}'_1, [s]_2, \mathsf{ck}_2, \mathsf{vk}_2)$

THEOREM 10. *The binary proof has perfect completeness, HVSZK and computational witness extended emulation under the SXDH and DPair-ML assumptions.*

PROOF. The binary proof is a special case of the aggregated binary proof in Theorem 11 with $k = 1$. It can be regarded as a corollary of Theorem 11. □

Aggregated binary proofs. The prover is similar to the prover for a binary proof with $k \cdot n$ bits except for the following modifications. Without loss of generality, we assume that $k \cdot n$ is still a power of 2. It proves that the committed values are $k \cdot n$ bits, and they are the concatenation of *k* blocks. The number of 1's in the j-th block is t_j: $\boldsymbol{b} = (\boldsymbol{b}_1 || \cdots || \boldsymbol{b}_k)$ for all $j \in [k]$ where $\boldsymbol{b}_j \in \{0,1\}^n$ and $\langle \mathbf{1}^n, \boldsymbol{b}_j \rangle = t_j$. Formally, we define a language:

$$
\begin{aligned}
(C, [r]_2, \mathsf{ck}_2, \mathsf{vk}_2, e_H, \{t_j\}_{j\in[k]}) \in \mathcal{L}_{\mathsf{aBin}} \Leftrightarrow \\
&([r]_2, \mathsf{ck}_2, \mathsf{vk}_2) \in \mathcal{L}^1_{\mathsf{Com}} \wedge \\
&\exists \boldsymbol{b} \in \{0,1\}^n, r_C \in \mathbb{Z}_q, s.t.: \\
&C = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2 \rangle + r_C \cdot e_H \wedge \langle \mathbf{1}^n, \boldsymbol{b}_j \rangle = t_j, \forall j \in [k].
\end{aligned}
$$

We convert $\boldsymbol{b}$ to $[\boldsymbol{b}]_1 \in \mathbb{G}_1^{kn}$, and commit it into $B = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2 \rangle + r_B \cdot e_H$ where $r_B \leftarrow\!\!\$\ \mathbb{Z}_q$ and all t_j are public. The protocol in the former section is modified as follows:

$$l(X) = \boldsymbol{b} - \tau \cdot \mathbf{1}^{k\cdot n} + \boldsymbol{u}_1 \cdot X \in \mathbb{Z}_q^{k\cdot n}[X]$$

$$r(X) = \boldsymbol{y}^{k\cdot n} \circ (\boldsymbol{b'} + \tau \cdot \mathbf{1}^{k\cdot n} + \boldsymbol{u}_2 \cdot X) + \sum_{j=1}^{k} \tau^{j+1} \cdot (\mathbf{0}^{(j-1)\cdot n} || \mathbf{1}^n || \mathbf{0}^{(k-j)\cdot n}) \in \mathbb{Z}_q^{k\cdot n}[X]$$

$\delta(y,\tau) = (\tau - \tau^2) \cdot \langle \mathbf{1}^{k\cdot n}, \boldsymbol{y}^{k\cdot n} \rangle - \sum_{j=1}^{k} \tau^{j+2} \cdot \langle \mathbf{1}^n, \mathbf{1}^n \rangle$

The verification check needs to include each t_j:

$$P + \phi_x \cdot e_H = ((\sum_{j=1}^{k} (t_j \cdot \tau^{j+1}) + \delta(y,\tau)) \cdot e(g, \tilde{g}) + x \cdot P_1 + x^2 \cdot P_2$$

Q_2 needs to be updated to

$$Q_2 = B_2 + x \cdot U_2 + \tau \cdot Y + \sum_{j=1}^{k} \tau^{(j+1)} \cdot \langle \mathsf{ck}'_{1j}, [\mathbf{1}^n]_2 \rangle$$

where ck'_{1j} consists of the $((j-1)\cdot n + 1)$-th element to the $(j \cdot n)$-th element of ck'_1.

THEOREM 11. *The aggregated binary proof has perfect completeness, HVSZK and computational witness extended emulation under the SXDH and DPair-ML assumptions.*

PROOF. The proof is analogous to witness extraction of the range proof in [126].

Perfect completeness: The perfect completeness follows from the fact that $p_0 = \delta(y,z) + \sum_{j=1}^{k} t_j \cdot z^{j+1}$.

Perfect HVZK: We describe a simulator that, given an input to the statement and the randomness of the verifier, computes a transcript that is perfectly indistinguishable from a transcript of a real execution. It chooses all proof elements and challenges and computes them as in the protocol. Especially, V and P_1 are computed according to the verification equations, i.e.

$$P_1 = x^{-1} \cdot (P + \phi_x \cdot e_H - ((\sum_{j=1}^{k}(t_j \cdot z^{j+1}) + \delta(y,z)) \cdot e(g,\tilde{g}) - x^2 \cdot P_2))$$

$$V = x^{-1} \cdot (Q_2 - B_2 - z^n \cdot Y - \sum_{j=1}^{k} z^{(j+1)\cdot n} \cdot \langle \mathsf{ck}'_{1j}, \tilde{\boldsymbol{g}} \rangle)$$

Finally, the simulator runs the inner-product argument with the simulated witness $\boldsymbol{l}$, $\boldsymbol{r}$ and the verifier's randomness. All elements in the proof are either independently randomly distributed, or their relationship is fully defined by the verification equations. The inner product argument remains zero knowledge as we can successfully simulate the witness; thus, revealing the witness or leaking information about it does not change the zero-knowledge property of the overall protocol. The simulator runs on time and is thus efficient.

Computational witness extended emulation: we construct an extractor χ as follows. χ runs the prover with $k \cdot n$ different values of y, $(k+2)$ different values of z, and 3 different values of the challenge x. Additionally it invokes the extractor χ_{IPP} for the inner product argument on each transcript to extract $[\boldsymbol{l}]_1, [\boldsymbol{r}]_2$ such that $P = \langle [\boldsymbol{l}]_1, [\boldsymbol{r}]_2 \rangle$, $Q_1 = \langle [\boldsymbol{l}]_1, \mathsf{ck}_2 \rangle$, $Q_2 = \langle \mathsf{ck}_1, [\boldsymbol{r}]_2 \rangle$.

Using two valid transcripts and extracted inner product argument witnesses for different x challenges, we can compute $\alpha, \beta, \gamma_u, \gamma_v, [\boldsymbol{b}]_1, [\boldsymbol{b'}]_2, [\boldsymbol{u}]_1, [\boldsymbol{v}]_2$ such that $B_1 = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2 \rangle + \alpha \cdot e_H$, $B_2 = \langle \mathsf{ck}_1, [\boldsymbol{b'}]_2 \rangle + \beta \cdot e_H$, $U = \langle [\boldsymbol{u}]_1, \mathsf{ck}_2 \rangle + \gamma_u \cdot e_H$, $V = \langle \mathsf{ck}_1, [\boldsymbol{v}]_2 \rangle + \gamma_v \cdot e_H$.

If for any other set of challenges, (x, y, z) the extractor can compute a different representation of B_1, B_2, U or V, then this breaks the ML-Find-Rep assumption under the SXDH assumption.

Using these elements with $[\boldsymbol{l}]_1, [\boldsymbol{r}]_2$, the following equations hold for all challenges x, y, z:

$$[\boldsymbol{l}]_1 = [\boldsymbol{b} - z \cdot \mathbf{1}^{k \cdot n} + \boldsymbol{u} \cdot x]_1$$

$$[\boldsymbol{r}]_2 = [\boldsymbol{y}^{k \cdot n} \circ (\boldsymbol{b'} + z \cdot \mathbf{1}^{k \cdot n} + \boldsymbol{v} \cdot x) + \sum_{j=1}^{k} z^{j+1} \cdot (\mathbf{0}^{(j-1) \cdot n} || \mathbf{1}^n || \mathbf{0}^{(k-j) \cdot n})]_2$$

If the equalities do not hold for all challenges and values of $[\boldsymbol{l}]_1$ and $[\boldsymbol{r}]_2$ from the transcript, it implies that the DPair-ML assumption does not hold.

For given values of y, z, we take three transcripts with different x's and uses linear combinations of equation to compute p_1, p_2, ϕ_1, ϕ_2 such that $P_1 = p_1 \cdot e(g, \tilde{g}) + \phi_1 \cdot e_H$, $P_2 = p_2 \cdot e(g, \tilde{g}) + \phi_2 \cdot e_H$.

Furthermore, if any $[\delta(y, z) + \sum_{j=1}^{k} z^{j+2} \cdot t_j + p_1 \cdot x + p_2 \cdot x^2]_T \neq P$, it yields a violation of the binding property of the structured AFGHO commitment. If not, then for all t_j and all y, z challenges and three different challenges $X = x_1, x_2, x_3$, we have $\sum_{i=0}^{2} p_i \cdot X^i - q(X) = 0$ with $p_0 = \delta(y, z) + \sum_{j=1}^{k} z^{j+2} \cdot t_j$ and $q(X) = \langle l(X), r(X) \rangle = q_0 + q_1 \cdot X + q_2 \cdot X^2$. Since the polynomial $p(X) - q(X)$ is of degree 2 but has at least three roots, it must be the zero polynomial. In other words, $p(X) = \langle l(X), r(X) \rangle$. Since it implies $p_0 = q_0$, the following equation holds for all y, z challenges:

$$\begin{aligned} \sum_{j=1}^{k} z^{j+2} \cdot t_j + \delta(y, z) = & \langle \boldsymbol{b}, \boldsymbol{y}^{k \cdot n} \circ \boldsymbol{b'} \rangle + z \cdot \langle \boldsymbol{b} - \boldsymbol{b'}, \boldsymbol{y}^{k \cdot n} \rangle + \sum_{j=1}^{k} z^{j+1} \langle \boldsymbol{b}_j, \mathbf{1}^n \rangle \\ & - z^2 \cdot \langle \mathbf{1}^{k \cdot n}, \boldsymbol{y}^{k \cdot n} \rangle - \sum_{j=1}^{k} z^{j+2} \langle \mathbf{1}^n, \mathbf{1}^n \rangle \end{aligned}$$

If this equality holds for $k \cdot n$ distinct y challenges and $k+2$ distinct z challenges, then we can infer that

$$\boldsymbol{b} \circ \boldsymbol{b'} = \mathbf{0}^{k \cdot n}$$

$$\boldsymbol{b'} = \boldsymbol{b} - \mathbf{1}^{k \cdot n}$$

$$t_j = \langle \boldsymbol{b}_j, \mathbf{1}^n \rangle \; \forall j \in [k]$$

The first two equations imply that $\boldsymbol{b} \in \{0,1\}^{k \cdot n}$ and the last one implies that the Hamming weight of $\boldsymbol{b}_j$ is t_j for all $j \in [k]$. The extractor rewinds the prover $3(k+2)kn \cdot O(n^2)$ times, which is an efficient process. The extraction itself is efficient, and the number of transcripts generated is polynomial in λ. It is important to note that the extraction operation either yields a valid witness or breaks the DPair-ML assumption, whose probability is, at most, negligible. Consequently, we can apply the forking lemma to conclude that computational witness emulation is upheld. □

6.4.3 Efficient Construction from BLS Signatures

- **Setup**(1^λ) In this phase, the system parameters are generated, especially, the common reference string. On input the security parameter 1^λ, it produces the public parameters for the BLS scheme $pp_{bls} = \{\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e(\cdot,\cdot), H_1(\cdot)\}$. Here $\mathbb{G}_1, \mathbb{G}_2$ are asymmetric groups, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is the Type III bilinear pairing operation, and $H_1 : \{0,1\}^* \to \mathbb{G}_1$ is the hash function. $H_2 : \{0,1\}^* \to \mathbb{Z}_q$ is another hash function. Let k be the number of potential messages, n be the maximum number of users. W.l.o.g., we assume they are power of 2. The setup algorithm additionally outputs the structured common reference string $\mathsf{crs} = (pp_{\mathsf{com}}, [r]_1, \mathsf{ck}_1, \mathsf{vk}_1, [s]_2, \mathsf{ck}_2, \mathsf{vk}_2)$ as we described in Sec. 2.3. Note that $\mathsf{ck}_1 = (\mathsf{ck}_1^1 || \cdots || \mathsf{ck}_1^j || \cdots || \mathsf{ck}_1^k || \cdots || \mathsf{ck}_1^{2k}) \in \mathbb{G}_1^{2kn}$, ck_1^j denotes the $((j-1)n+1)$-th element to the jn-th element of ck_1. $\mathsf{ck}_2 = (\mathsf{ck}_2^1 || \cdots || \mathsf{ck}_2^k || \cdots || \mathsf{ck}_2^{2k}) \in \mathbb{G}_2^{2kn}$. Especially, let $\mathsf{ck}_1' = (\mathsf{ck}_1^1 || \cdots || \mathsf{ck}_1^k) \in \mathbb{G}_1^{kn}$, $\mathsf{ck}_2' = (\mathsf{ck}_2^1 || \cdots || \mathsf{ck}_2^k) \in \mathbb{G}_2^{kn}$ and we use $\mathsf{ck}_1^*, \mathsf{ck}_2^*$ to denote the first element of $\mathsf{ck}_1^{k+1}, \mathsf{ck}_2^{k+1}$ respectively. Output $pp = (pp_{bls}, \mathsf{crs}, H_2)$.

  Each user generates his secret key $sk_i \leftarrow\!\!\$\; \mathbb{Z}_q$ and public key $pk_i = [sk_i]_2 \in \mathbb{G}_2$ and broadcasts the public key with proof of knowledge of secret key.

The combiner or any other parties collects the public keys $\boldsymbol{pk} = (pk_1, ..., pk_n)$ and publishes the commitments of them as $K_1 = \langle \mathsf{ck}_1^1, \boldsymbol{pk} \rangle, ..., K_k = \langle \mathsf{ck}_1^k, \boldsymbol{pk} \rangle$.

It also publish the aggregation key $ak = (pp, \boldsymbol{pk})$ and the verification key $vk = (pp_{\mathsf{com}}, pp_{bls}, [r]_1, \mathsf{vk}_1, [s]_2, \mathsf{vk}_2, K_1, ..., K_k)$

- **ParSign**(m_j, sk_i) For a message m_j chosen from the message space $\mathcal{M}$, the user u_i signs on it using his secret key sk_i and sends (pk_i, m_j, σ_{ij}) to the combiner where $\sigma_{ij} = H(m_j)^{sk_i}$.
- **ParVrfy**(pk_i, m_j, σ_{ij}) On receiving (pk_i, m_j, σ_{ij}), the combiner verifies it. Output 1 if $e(H(m_j), pk_i) = e(\sigma_{ij}, \tilde{g})$, otherwise output 0.
- **Combine**$(ak, \{pk_i, m_j, \sigma_{ij}\}_{j\in[k]})$ When collecting a set of $\{pk_i, m_j, \sigma_i\}$, the combiner verifies them one by one. If all of them are valid, the combiner does as follows:

  - Let $S_j \subseteq [n]$ be the indices of signers who have signed on m_j, set $\boldsymbol{b}_j = (b_{1j}, ..., b_{nj}) \in \{0,1\}^n$, such that $b_{ij} = 1$ if $i \in S_j$, otherwise, $b_{ij} = 0$ and $t_j = \sum_{i=1}^n b_{ij}$;
  - Let $\boldsymbol{b} = (\boldsymbol{b}_1 || \cdots || \boldsymbol{b}_k)$, compute the commitment to $[\boldsymbol{b}]_1 \in \mathbb{G}_1^{kn}$: $B = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2' \rangle + r_B \cdot e_H$ where $r_B \leftarrow\!\!\$\ \mathbb{Z}_q$ and generate the binary proof π_{aBin} from Π_{aBin} w.r.t. the language $\mathcal{L}_{\mathsf{aBin}}(C, [r]_2, \mathsf{ck}_2', \mathsf{vk}_2', e_H, \{t_j\}_{j\in[k]})$;
  - For $j = 1$ to k, $m_j \in M$, choose $r_j \leftarrow\!\!\$\ \mathbb{Z}_q$, compute sub-aggregated public keys $\widehat{pk}_j = \Pi_{i\in S_j} pk_i \cdot \tilde{g}^{r_j} = \Pi_{i=1}^n pk_i^{b_{ij}} \cdot \tilde{g}^{r_j} = \boldsymbol{pk}^{\boldsymbol{b}_j} \cdot \tilde{g}^{r_j}$ and sub-aggregated signatures $\hat{\sigma}_j = \Pi_{i\in S_j} \sigma_i \cdot H(m_j)^{r_j}$;
  - For the k sub-aggregated public keys $\widehat{pk}_j$, compute $\widehat{PK} = \Pi_{j=1}^k \widehat{pk}_j^{z^{(j-1)}} = \boldsymbol{pk}^{\boldsymbol{b}_1} \cdot \boldsymbol{pk}^{z\boldsymbol{b}_2} \cdots \boldsymbol{pk}^{z^{k-1}\boldsymbol{b}_k} \cdot \tilde{g}^{r^*} = (\boldsymbol{pk} || \boldsymbol{pk}^z || \cdots || \boldsymbol{pk}^{z^{k-1}})^{(\boldsymbol{b}_1 || \cdots || \boldsymbol{b}_k)} \cdot \tilde{g}^{r^*}$, where $z = H_2(\{\widehat{pk}_j, \widehat{\sigma}_j\}_{j\in[k]}, B, \pi_{\mathsf{aBin}})$ and $r^* = \sum_{j=1}^k r_j \cdot z^{j-1}$.
  - Compute $X = r^* \cdot e(g, \mathsf{ck}_2^*) + r_X \cdot e_H$ and generate π_{pok} (via the Schnorr's protocol [89]) to prove the knowledge of r^* and r_X;
  - Compute $Q = B + X = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2' \rangle + e(g^{r^*}, \mathsf{ck}_2^*) + (r_B + r_X) \cdot e_H$, it means $Q = \langle ([\boldsymbol{b}]_1, g^{r^*}), (\mathsf{ck}_2', \mathsf{ck}_2^*) \rangle + (r_B + r_X) \cdot e_H \in \mathbb{G}_T$, so we know Q is the commitment of a binary vector and a randomness;
  - Compute $K = \langle (\mathsf{ck}_1', \mathsf{ck}_1^*), (\boldsymbol{pk}^{\mathbf{z}^k}, \tilde{g}) \rangle$, $E = \langle [\boldsymbol{b}]_1, \boldsymbol{pk}^{\mathbf{z}^k} \rangle + r^* \cdot e(g, \tilde{g}) \in \mathbb{G}_T$, where $\boldsymbol{pk}^{\mathbf{z}^k} = (\boldsymbol{pk} || \boldsymbol{pk}^z || ... || \boldsymbol{pk}^{z^{k-1}}) \in \mathbb{G}_2^{kn}$;

- Based on E, Q, K, generates π_{IPP} from Π_{IPP} w.r.t. the language $\mathcal{L}_{\mathsf{IPP}}(E, Q, K, \mathsf{ck}_1, \mathsf{vk}_1, \mathsf{ck}_2, \mathsf{vk}_2, e_H)$ to prove that E is the *inner pairing product* of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ which are committed in Q and K: $E = \langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle + r_E \cdot e_H$, where $\boldsymbol{v}_1 = (g^{b_1}, ..., g^{b_{kn}}, g^{r^*}, [0]_1, ..., [0]_1) \in \mathbb{G}_1^{2kn}$ and $b_i \in \{0, 1\}$, $r^* \in \mathbb{Z}_q$ which has been proved via π_{aBin} and π_{pok}, $\boldsymbol{v}_2 = (\boldsymbol{pk}^{\boldsymbol{z}^k}, \tilde{g}, [0]_2, ..., [0]_2) \in \mathbb{G}_2^{2kn}$ which is public.[4]
- Generate the proof π_{disj} to prove all signer sets are disjoint: $S_{j_0} \cap S_{j_1} = \emptyset, \forall j_0, j_1 \in [k], j_0 \neq j_1$. It requires the combiner additionally prove that $\widehat{PK}' = \prod_{j=1}^{k} \widehat{pk}_j$ can be expressed in the form of $\boldsymbol{pk}^{\boldsymbol{b}'} \cdot \tilde{g}^{r'}$ using the same method as above, where $r' = \sum_{j=1}^{k} r_j$, $\boldsymbol{b}' = \sum_{j=1}^{k} \boldsymbol{b}_j$ is also a binary vector with $\sum_{j\in[k]} t_j$ ones.
- Output the signature $\Sigma \leftarrow (\{\widehat{pk}_j, \widehat{\sigma}_j\}_{j\in[k]}, B, z, X, \pi_{\mathsf{aBin}}, \pi_{\mathsf{pok}}, \pi_{\mathsf{IPP}}, \pi_{\mathsf{disj}})$ with the thresholds $T = \{t_j\}_{j\in[k]}$ as Δ.

- **Verify**(vk, M, T, Σ) Parse $\Sigma = (\{\widehat{pk}_j, \widehat{\sigma}_j\}_{j\in[k]}, B, z, X, \pi_{\mathsf{aBin}}, \pi_{\mathsf{pok}}, \pi_{\mathsf{IPP}}, \pi_{\mathsf{disj}}), T = \{t_j\}_{j\in[k]}$, compute:
  $z' = H_2(\{\widehat{pk}_j, \widehat{\sigma}_j\}_{j\in[k]}, B, \pi_{\mathsf{aBin}}), \widehat{\sigma} = \prod_{j\in[k]} \widehat{\sigma}_j, Q = B + X,$
  $K = \boldsymbol{K}_1 + \cdots + \boldsymbol{K}_j^{z'^{j-1}} + \cdots + \boldsymbol{K}_k^{z'^{k-1}} + e(\mathsf{ck}_1^*, \tilde{g}),$
  $\widehat{PK} = \prod_{j=1}^{k} \widehat{pk}_j^{z'^{(j-1)}}, E = e(g, \widehat{PK}), \widehat{PK}' = \prod_{j=1}^{k} \widehat{pk}_j, \hat{t} = \sum_{j\in[k]} t_j.$
  Accept if all the following conditions are satisfied:
  - $z' = z$;
  - **Vrfy**$(\{\widehat{pk}_j, m_j\}_{j\in[k]}, \hat{\sigma}) = 1$;
  - π_{aBin} is valid w.r.t. B, T;
  - π_{pok} is valid w.r.t. X;
  - π_{IPP} is valid w.r.t. Q, K, E;
  - π_{disj} are valid w.r.t. $\widehat{PK}', \hat{t}$.

General Predicate Satisfiability Proofs In our specific setting, where we aim to prove the predicate regarding the relations among signer sets, we focus on the committed binary vector. It serves as the representation of signers for different messages and plays a crucial role in our proof construction.

[4] We pad 'zeros' in $\boldsymbol{v}_1, \boldsymbol{v}_2$ since the dimension of commitment keys for $\mathcal{L}_{\mathsf{IPP}}$ is $2kn$, which is a power of 2.

Recall that Daza et al. [36] proves that the committed vectors satisfy some circuits. By following the protocol of zero-knowledge SNARK for circuit satisfiability in [36] but with AFGHO commitment (which is also homomorphic same as the Pedersen commitment), we can obtain a circuit satisfiability proof that is compatible with our previous inner pairing product and binary proofs. Both communication cost and verification complexity are logarithmic to the size of the circuit.

6.5 Analysis

6.5.1 Performance Analysis

We first analyze the performance related to the inner pairing product and binary proofs. The inner pairing product and binary proof protocols require $\nu = \log(k \cdot n)$ rounds, where each round involves communication and computations. In each round, the size of the witness is halved. It leads to a communication complexity of $O(\nu)$ since the communication cost is constant in each round. The prover's computation complexity in round i is $O(2^{\nu-i+1})$. As a result, the overall prover complexity is $O(2^{\nu})$ since ν rounds are performed. On the other hand, the verifier's computation cost remains constant at $O(1)$ since it only needs to perform a fixed number of operations in each round. Consequently, the verifier's overall complexity is $O(\nu)$. For the predicate satisfaction proof with generic arithmetic circuit, the communication cost and verification complexity are logarithmic to the size of the circuit, which is $O(\log |\mathcal{C}|)$. The prover's computation complexity is $O(|\mathcal{C}|)$.

In the Combine phase, the combiner needs to verify the partial signatures, generate the sub-aggregated public keys and sub-aggregated signatures, and the computation cost is $O(n)$. The final signature contains additional aggregated public keys and signatures and thresholds w.r.t. each message. It leads to $O(k)$ communication cost. On the final signature, the verifier checks the sub-aggregated signatures with the respective message and sub-aggregated public key. It involves the $O(k)$ computation complexity. Then, it verifies the correctness of these sub-aggregated public keys by checking these proofs.

In summary, the communication cost is $O(k+\nu+\log|\mathcal{C}|) = O(k+\log k+\log n+\log|\mathcal{C}|)$, the prover complexity is $O(2^{\nu}+|\mathcal{C}|) = O(k\cdot n+|\mathcal{C}|)$, and the verifier complexity is $O(\nu+\log|\mathcal{C}|) = O(\log k+\log n+\log|\mathcal{C}|)$. For the special case of rate-once policy, the communication cost is $O(k+\log k+\log n)$, the prover complexity is $O(k\cdot n)$, and the verifier complexity is $O(k+\log k+\log n)$.

6.5.2 Security Analysis

THEOREM 12 (Anonymity). *The predicate aggregate signature is anonymous in the random oracle model under the SXDH assumption.*

PROOF. Based on the rate-once policy in the anonymous reputation system, the predicate has been explained in Equation (6.1) and the description Δ is defined as $\Delta = T = \{t_j\}_{j\in[k]}$. In the anonymous experiment, $\Delta_0 = T_0, \Delta_1 = T_1$ and it is required that $T_0 = T_1$ for the challenge sets S^0, S^1. Given the signature $\Sigma = (\{\widehat{pk}_j, \widehat{\sigma}_j\}_{j\in[k]}, B, z, X, \pi_{\mathsf{aBin}}, \pi_{\mathsf{pok}}, \pi_{\mathsf{IPP}}, \pi_{\mathsf{disj}})$, and thresholds $T = \{t_j\}_{j\in[k]}$. Note that T, K leak nothing since they are the same in both challenges with identity sets S^0, S^1. We design the hybrid games as follows:

- $\mathsf{G}_{\mathsf{real}}$: This game is the same as the experiment, challenger chooses $b \leftarrow\!\$\ \{0,1\}$ and generates the signature Σ_b honestly under the identities in S^b.
- G_1 : This game is similar with G_0 except that the proofs $\pi_{\mathsf{aBin}}, \pi_{\mathsf{pok}}, \pi_{\mathsf{IPP}}, \pi_{\mathsf{disj}}$ are simulated without witness.
- G_2 : This game is similar with G_1 except that the sub-aggregated public keys and signatures are generated randomly without using b as follows: for $j \in [k]$, choose $u_j \leftarrow\!\$\ \mathbb{Z}_q$ and set $\widehat{pk}'_j = \tilde{g}^{u_j} \in \mathbb{G}_2$ and $\hat{\sigma}'_j = H(m_j)^{u_j} \in \mathbb{G}_1$ such that $e(H(m_j), \widehat{pk}'_j) = e(\hat{\sigma}'_j, \tilde{g})$.
- G_3: This game is similar with G_2 except that B, X are also chosen randomly independently: $B', X' \leftarrow\!\$\ \mathbb{G}_T$. Note that the proofs $\pi_{\mathsf{aBin}}, \pi_{\mathsf{pok}}, \pi_{\mathsf{IPP}}, \pi_{\mathsf{disj}}$ are simulated without using witnesses. They can still be verified.

Compare G_1 with $\mathsf{G}_{\mathsf{real}}$, the only difference is that these proofs are simulated. Since these proofs are zero-knowledge under the SXDH assumptions in the random oracle model, the

probability of distinguishing G_1 from $\mathsf{G}_{\mathsf{real}}$ is negligible. We have that $|\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A},\lambda) = 1] - \Pr[\mathsf{G}_1(\mathcal{A},\lambda) = 1]| \leq negl(\lambda)$.

Compare G_2 with G_1, in G_1, $\widehat{pk}_j = \Pi_{i\in S_j^b} pk_i \cdot \tilde{g}^{r_j}$, $\widehat{\sigma}_j = \Pi_{i\in S^b}\sigma_i \cdot H(m_j)^{r_j}$ where $r_j \leftarrow\!\!\$\ \mathbb{Z}_q$, so they are uniformly random and each pair satisfies $e(H(m_j), \widehat{pk}_j) = e(\hat{\sigma}_j, \tilde{g})$. In G_1, $\widehat{pk}_j'$ and $\hat{\sigma}_j'$ are also random elements in $\mathbb{G}_1, \mathbb{G}_2$ respectively, and satisfy the same kinds of relation. $\widehat{PK}'$ is generated from these random $\widehat{pk}_j'$ and z, so is E'. Thus $(\{\widehat{pk}_j, \widehat{\sigma}_j\}_{j\in[k]}, z, \widehat{pk}, E)$ and $(\{\widehat{pk}_j', \widehat{\sigma}_j'\}_{j\in[k]}, z', \widehat{PK}', E')$ have the same distribution. The probability that they can be distinguished is 0. We have that $|\Pr[\mathsf{G}_1(\mathcal{A},\lambda) = 1] - \Pr[\mathsf{G}_2(\mathcal{A},\lambda) = 1]| = 0$.

Compare G_3 with G_2, in G_2, B is the structured AFGHO commitment of $[\boldsymbol{b}]_1$, X is the Pedersen commitment of r^*. Since these commitments are perfect hiding, they are indistinguishable from the random B', X' in G_3. We have that $|\Pr[\mathsf{G}_2(\mathcal{A},\lambda) = 1] - \Pr[\mathsf{G}_3(\mathcal{A},\lambda) = 1]| = 0$.

In G_3, $\mathcal{A}$'s view is independent of b. Thus, $\mathcal{A}$ just outputs a random guess $\hat{b}$ in G_3, so its advantage is 0: $\Pr[\mathsf{G}_3(\mathcal{A},\lambda) = 1] - 1/2 = 0$. In summary, we have

$$
\begin{aligned}
&|\Pr[\mathrm{Exp}^{\mathsf{anony}}(\mathcal{A},\lambda) = 1] - 1/2| = |\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A},\lambda) = 1] - 1/2| \\
\leq &|\Pr[\mathsf{G}_{\mathrm{real}}(\mathcal{A},\lambda) = 1] - \Pr[\mathsf{G}_1(\mathcal{A},\lambda) = 1]| + |\Pr[\mathsf{G}_1(\mathcal{A},\lambda) = 1] - \Pr[\mathsf{G}_2(\mathcal{A},\lambda) = 1]| \\
&+ |\Pr[\mathsf{G}_2(\mathcal{A},\lambda) = 1] - \Pr[\mathsf{G}_3(\mathcal{A},\lambda) = 1]| + |\Pr[\mathsf{G}_3(\mathcal{A},\lambda) = 1] - 1/2| \leq negl(\lambda)
\end{aligned}
$$

□

THEOREM 13 (Unforgeability). *The predicate aggregate signature is unforgeable in the random oracle model under the co-CDH, SXDH and DPair-ML assumptions.*

PROOF. First of all, we assume that the proof of possession has been done to prove the knowledge of secret key for each public key. Based on the rate-once policy in the anonymous reputation system, the predicate has been explained as in Equation (6.1). So in the unforgeability experiment, the adversary $\mathcal{A}$ wins if for the extracted identities sets $S_1, ..., S_k$, at least one of the following happens:

(1) $\exists\, i_j \in S_j$, such that $\mathcal{A}$ has never queried the **corrupt** oracle on pk_{i_j} or **sign** oracle on (pk_{i_j}, m_j);
(2) $\exists\, j \in [k]$, s.t. $t_j \neq |S_j|$;
(3) $\exists\, i_j \in S_j$ and it appears more than once in S_j;
(4) $\exists\, S_i$ and S_j which overlap: $S_i \cap S_j \neq \emptyset$.

We reduce the security of our scheme to the security of the underlying BLS signature which is unforgeable under co-CDH assumption, and non-interactive ZKAoK which is sound and knowledge sound under SXDH and DPair-ML assumptions. We elaborate it case by case.

Extract the identities and randomness: $\mathcal{A}$ outputs a non-trivial PAS forgery $\Sigma = (\{\widehat{pk}_j, \widehat{\sigma}_j\}_{j\in[k]}, B, z, X, \pi_{\mathsf{aBin}}, \pi_{\mathsf{pok}}, \pi_{\mathsf{IPP}}, \pi_{\mathsf{disj}})$ on message set $M = \{m_j\}_{j=1}^k$ with threshold $T = \{t_j\}_{j=1}^k$. Due to the witness extended emulation of $\Pi_{\mathsf{aBin}}, \Pi_{\mathsf{IPP}}$ and the knowledge soundness of Π_{pok}, there exists an extractor $\mathcal{E}$ who can extract the signer identities $\boldsymbol{b}_j, r_j$ such that $\widehat{pk}_j = \boldsymbol{pk}^{\boldsymbol{b}_j} \cdot \tilde{g}^{r_j}$ as follows.

$\mathcal{E}$ can run the extractor χ_{aBin} for π_{aBin} to extract the committed elements $[\boldsymbol{b}]_1 \in \mathbb{G}_1^{kn}$ and randomness $r_B \in \mathbb{Z}_q$ s.t. $B = \langle [\boldsymbol{b}]_1, \mathsf{ck}_2\rangle + r_B \cdot e_H$ and $\boldsymbol{b} \in \{0,1\}^{kn}$. $\mathcal{E}$ can rewind $\mathcal{A}$ on different z. For each z, $\mathcal{E}$ runs the extractor χ_{pok} for π_{pok} to extract the committed element $r^* \in \mathbb{Z}_q$ and randomness $r_X \in \mathbb{Z}_q$ s.t. $X = r^* \cdot e(g, \mathsf{ck}_2^*) + r_X \cdot e_H$ and runs the extractor χ_{IPP} for π_{IPP} to extract the committed elements $\boldsymbol{v}_1 \in \mathbb{G}_1^{2kn}, \boldsymbol{v}_2 \in \mathbb{G}_2^{2kn}$ and randomness $r_Q, r_K, r_E \in \mathbb{Z}_q$ s.t. $E = \langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle + r_E \cdot e_H$, $Q = \langle \boldsymbol{v}_1, (\mathsf{ck}_2, \mathsf{ck}_2^*)\rangle + r_Q \cdot e_H$, $K = \langle (\mathsf{ck}_1, \mathsf{ck}_1^*), \boldsymbol{v}_2\rangle + r_K \cdot e_H$. We know that $\boldsymbol{v}_1 = ([\boldsymbol{b}]_1, [r^*]_1, [0]_1, ..., [0]_1)$, $\boldsymbol{v}_2 = (\boldsymbol{pk}^{\boldsymbol{z}^k}, \tilde{g}, [0]_2, ..., [0]_2)$. Otherwise, it breaks the DPair-ML assumption. So $E = \langle \boldsymbol{v}_1, \boldsymbol{v}_2\rangle + r_E \cdot e_H = e(g, \widehat{G}) + e(g_H, \tilde{g}^{r_E})$, where $\widehat{G} = \boldsymbol{pk}^{\boldsymbol{z}^k \boldsymbol{b}} \cdot \tilde{g}^{r^*}$. We also have $\widehat{PK} = \prod_{j=1}^k \widehat{pk}_j^{z^{(j-1)}} \in \mathbb{G}_2$ s.t. $E = e(g, \widehat{PK})$. It means that $\widehat{PK} = \widehat{G}$ and $r_E = 0$, otherwise, it breaks the DPair assumption by finding a non-trivial pair $(N_1, N_2) = (\widehat{G}/\widehat{PK}, \tilde{g}^{r_e}) \in \mathbb{G}_2^2$ s.t. $e(g, N_1) + e(g_H, N_2) = [0]_T$. Thus $\widehat{PK}$ can be expressed in the form of $\boldsymbol{pk}^{\boldsymbol{z}^k \boldsymbol{b}} \cdot \tilde{g}^{r^*}$.

Repeating this for k different challenges z with the randomness r^*, we can compute r_j for $j \in [k]$ s.t. $r^* = \sum_{j=1}^k r_j \cdot z^{j-1}$ for each challenge and each sub-aggregated public key $\widehat{pk}_j$ is

in the form of $\boldsymbol{pk}^{\boldsymbol{b}_j} \cdot \tilde{g}^{r_j}$ by Schwartz-Zippel lemma, where $\boldsymbol{b}_j$ is the j-th block in the extracted $\boldsymbol{b}$.

(1) Suppose that P_1=Pr[$\mathcal{A}$ wins and violates condition 1] is non-negligible. We prove that if $\mathcal{A}$ wins, we can use it in a black-box manner to construct an attacker $\mathcal{B}$ to break the unforgeability of the underlying BLS signature. $\mathcal{B}$ receives the BLS parameter pp_{bls} and the target BLS public key pk^*. It can also query the BLS signing oracle $\mathbf{Sign}_{\mathrm{bls}}(\cdot)$ on pk^* and any message. $\mathcal{B}$ can emulate the experiment for $\mathcal{A}$ as follows.

Setup: $\mathcal{B}$ generates crs and sets $pp = (pp_{bls}, \mathsf{crs})$. $\mathcal{B}$ chooses an index $\hat{i} \leftarrow\!\!\$\ [n]$ and sets $pk_{\hat{i}} = pk^*$. For other $i \in [n], i \neq \hat{i}$, it generates the secret keys $sk_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and sets public keys $pk_i = [sk_i]_2 \in \mathbb{G}_2$. $\mathcal{B}$ sends pp and all public keys to $\mathcal{A}$.

Emulate Corrupt and Sign oracles: For corruption oracle **corrupt**$(\cdot)$: if $\mathcal{A}$ corrupts $pk_{\hat{i}}$, $\mathcal{B}$ aborts. Otherwise, for other identity corruption, $\mathcal{B}$ responds with the secret key sk_i. Note that $\mathcal{A}$is not allowed to corrupt all public keys.

For signing oracle **sign**$(\cdot,\cdot)$: if $\mathcal{A}$ queries on $(pk_{\hat{i}}, m)$, $\mathcal{B}$ forwards m to its **Sign**$_{bls}$ oracle and replies $\mathcal{A}$ with the signature it received. Otherwise, for other signer identities, $\mathcal{B}$ generates the signature on the message using secret key sk_i.

Breaking BLS Unforgeability: $\mathcal{A}$ outputs a non-trivial PAS forgery Σ on message set $M = \{m_j\}_{j=1}^k$ with threshold $T = \{t_j\}_{j=1}^k$. By the knowledge soundness, $\mathcal{B}$ can works like $\mathcal{E}$ as above to extract the signer identities $\boldsymbol{b}_j$, r_j such that $\widehat{pk}_j = \boldsymbol{pk}^{\boldsymbol{b}_j} \cdot \tilde{g}^{r_j}$. Based on each $\boldsymbol{b}_j$, we obtain the identity subset $S_j = \{i | b_{ji} = 1, b_{ji} \in \boldsymbol{b}_j\}$ for $j = 1$ to k and $S = \cup_{j=1}^k S_j$. Note that the non-triviality of the forgery implies that S includes at least one honest signer, who $\mathcal{A}$ did not corrupt. Otherwise, it proceeds as follows. $\mathcal{B}$ aborts if the target identity $\hat{i}$ is not included in S or $\hat{i} \in S_j$ w.r.t. a message m_j but $\mathcal{A}$ has queried **sign** oracle on $(pk_{\hat{i}}, m_j)$. Otherwise, $\mathcal{B}$ can locate the identity subset $S_{\hat{j}}$ w.r.t. the message $m_{\hat{j}}$ in which the target identity $\hat{i} \in S_{\hat{j}}$ and $\widehat{pk}_{\hat{j}} = \boldsymbol{pk}^{\boldsymbol{b}_{\hat{j}}} \cdot \tilde{g}^{r_{\hat{j}}}$. Given $\hat{\sigma}_j$, the randomness r_j and all other identities $i_j \in S_{\hat{j}}, i_j \neq \hat{i}$, $\mathcal{B}$ can computes their signatures σ_{i_j} on the message $m_{\hat{j}}$ and gets $\sigma_{i_j^*} = \hat{\sigma} / (\prod_{i \in S_j^* \setminus \{i_j^*\}} \sigma_i \cdot H(m_{\hat{j}})^{r_j})$ which is a valid signature of the target identity on $m_{\hat{j}}$ that $\mathcal{A}$ has never queried. So it is a successful BLS forgery and $\mathcal{B}$ wins.

Success Probability: Let $\epsilon_{\mathcal{A}}$ be the probability with which $\mathcal{A}$ outputs a valid forgery. It is easy to see that $\mathcal{B}$ breaks the unforgeability of BLS signature if it does not abort. We compute the lower bound of the probability with which $\mathcal{B}$ does not abort. Firstly, since $\hat{i}$ is chosen uniformly at random, $\mathcal{A}$ does not corrupt $pk_{\hat{i}}$ with probability at least $1/n$. Let δ be the probability with which $\mathcal{B}$ extracts the witness successfully. Then the probability that $\hat{i} \in I$ is at least $1/n$. Let q_H be the number of queries on the random oracle, q_S be the number of queries on the signing oracle, the probability that $\mathcal{A}$ has never queried on $(pk_{\hat{i}}, m_{\hat{j}})$ is at least $(1 - \frac{1}{n \cdot q_H}) \cdot (1 - \frac{1}{n \cdot q_H - 1}) \cdots (1 - \frac{1}{n \cdot q_H - q_S}) = \frac{n \cdot q_H - q_S - 1}{n \cdot q_H}$. Finally, we obtain the success probability of $\mathcal{B}$ is $\epsilon_{\mathcal{B}} \geq \epsilon_{\mathcal{A}} \cdot \frac{n \cdot q_H - q_S - 1}{n^3 \cdot q_H}$. Due to the unforgeability of the BLS signature, $\epsilon_{\mathcal{B}}$ is negligible, so $\epsilon_{\mathcal{A}}$ is also negligible.

(2) Suppose that P_2=Pr[$\mathcal{A}$ wins and violates condition 2] is non-negligible. It implies that $\mathcal{A}$ generates a valid binary proof for t_j with an incorrect witness whose Hamming weight is not t_j. It contradicts with the statement of π_{aBin} which breaks the soundness of the underlying ZKAoK.

(3) Suppose that P_3=Pr[$\mathcal{A}$ wins and violates condition 3] is non-negligible. It implies that $\mathcal{A}$ generates a valid binary proof with an incorrect witness which contains a number larger than 1. It also contradicts to the statement of π_{aBin}, so the soundness is broken.

(4) Suppose that P_4=Pr[$\mathcal{A}$ wins and violates condition 4] is non-negligible. It implies that $\mathcal{A}$ generates a valid binary proof for the sum of commitments with an incorrect witness which contains a number larger than 1. It contradicts to the statement of π_{disj}.

In summary, $\Pr[\mathrm{Exp}^{\mathsf{unforge}}(\mathcal{A}, \lambda) = 1] = P_1 + P_2 + P_3 + P_4 \leq negl(\lambda)$.

□

6.6 Applications and Extensions

Our PAS can be used to construct many other types of signatures by invoking the Combine algorithm as a blackbox to compress the final signature. Letting dynamic threshold be the

specific description and predicate function only requires the correctness of the threshold, our efficient scheme for single message also improves their state-of-the-art works in terms of trust model (relies on trusted party or non-standard assumptions) and efficiency as shown in Table 6.2. We explain them as follows.

(1) Our PAS implies threshold signatures with transparent setup[5]. Each signer generates their public key and secret key by themselves. Some of them sign on the same message and send to the combiner. Taking the valid partial signatures as input, the combiner runs the Combine algorithm to generate the final PAS signature with a threshold t. The verifier can be convinced that there are t different signers sign on this message.

(2) We get a multi-signature by letting everyone sign on the same message, aggregation is done via the Combine algorithm and the predicate only requires that the threshold number is correct.

(3) We get an aggregate signature which hides the signer identities from PAS, where aggregation is done via Combine algorithm and the predicate function is specified according to the concrete rule.

(4) We get a graded signature by letting everyone sign on the same message, aggregation is done via the Combine algorithm and the predicate only requires that the number of signers is correct. It ensures each of them can sign only once without leaking their identities.

(5) We get a threshold ring signature with prefixed threshold t by setting there is a single message and the predicate function always outputs 1 and modifying the verification algorithm a bit. Via the Combine algorithm, a PAS signature is produced. Now besides verifying whether it is valid, the verifier also checks whether the number of signers in PAS is larger than t. If yes, it is a valid threshold ring signature, otherwise not. When t is 1, it is a ring signature.

[5]Our dynamic threshold aggregate signature with transparent setup also offers a solution for multiverse threshold signature (MTS) [143]. For any subset of users interested in forming a universe with a specific threshold, the aggregation and verification keys can be computed from their public keys. Then run the Combine algorithm to get a PAS signature with the number of signers.

Anonymous reputation system. An anonymous reputation system enables users to rate products they have purchased. The primary security guarantee offered by such systems is privacy, allowing users to write reviews anonymously for any purchased products. However, to prevent abuse or misuse, a *rate-once policy* is implemented. This means that if a user attempts to write multiple reviews for the same product, their reviews will become publicly traceable or linked. It requires the final signature to be linear to the number of signers, and the verification time is quadratic. Recent works on the anonymous reputation systems [144, 145, 146] achieve full anonymity at the cost of linear communication cost and quadratic verification complexity.

We consider a relaxed but reasonable setting where a combiner is allowed to know the signers' identities. It can be the shopping website that knows the user's accounts when they log in. However, it cannot manipulate the final result, even if it colludes with some of the users. It cannot violate the rate-once policy and cannot generate reviews on behalf of other honest users. Malicious users and combiner cannot rate more than once or forge any other honest user's signature. The combiner produces a PAS in which the thresholds disclose the reputation states, and both the size and the verification time are logarithmic in the number of all users.

On-chain voting system. In more extensive scenarios, more intricate voting policies might exist that can be defined as conditions based on the identities of the voters. For instance, in blockchain governance, such as Decentralized Autonomous Organizations (DAOs), determinations might be reached by the entire community, provided their accounts possess a significant number of tokens. Certain policies necessitate that voters possess a particular property [123], and it can be denoted by their identity index. For instance, within an organization, senior members are associated with indices smaller than a threshold. The combiner's task is to demonstrate that among the signers, there is at least one whose index is lower than a specified threshold. In our design, relying on the binary vector, the combiner only needs to establish the existence of a single position in the vector where the value is 1, and the position is smaller than a specified threshold.

6.6.1 Extensions

Dynamic join. New users can seamlessly join the system without causing any disruptions to existing users. The process of joining is transparent and does not have any adverse effects on other users. A new user broadcasts his public key with PoP for registration. On verifying its validity, the combiner updates its aggregation key by adding this public key. Other honest verifier can also update the verification key by including the new public key in the commitment of all public keys. These updates are publicly verifiable and incur only a constant cost.

Weight aggregation. In the PAS scheme, each user can associate themselves with an additional weight. This weight represents their significance or influence within the system. The weight vector $\boldsymbol{w}$ is public, where each weight value w_i binds with a public key pk_i. In the Combine algorithm, the combiner additionally computes the sum of weights $W = \langle \boldsymbol{b}, \boldsymbol{w} \rangle$ as the description Δ. As a result, when the PAS generates a signature, it discloses the total weight of the signers involved. So our PAS also supports the weight aggregation like [130, 147, 138].

Accountability. Our PAS can be extended to support accountability by adding an extra identity encryption layer. This approach bears similarity to the method employed in TAPS [148]. In this extended system, the description pertains to the minimum threshold required for the number of signers. The combiner also encrypts both the count of signers and their identities. The predicate function ensures that they are correctly encrypted under the specified public key and that the size of the signer set surpasses the minimum threshold.

6.7 Summary

In this chapter, we introduce the concept of predicate aggregate signatures inspired by the needs of blockchain governance. It integrates privacy protection within compliance-oriented settings, enabling users to sign multiple messages while maintaining anonymity. A combiner aggregates these signatures into a single concise representation, revealing only a high-level description of the signers for each message. Importantly, the aggregated signature ensures that the signers and their descriptions satisfy a predefined public predicate aligned with compliance

requirements. We formally define PAS and propose a construction that achieves logarithmic signature size and verification time.

We demonstrate its applications to multiple settings, including many primitives, including multi-signatures, aggregate signatures, threshold signatures, (threshold) ring signatures, attribute-based signatures, etc, and advance the state of the art in all of them. Moreover, PAS finds applications in the anonymous reputation system and on-chain voting system for blockchain governance.

CHAPTER 7

Summary of This Thesis

7.1 Conclusion

Privacy and compliance are fundamental to human life, particularly in the context of blockchain applications. Striking a harmonious balance between these two aspects is crucial for fostering the growth and development of the blockchain community. This thesis focuses on safeguarding users' privacy while ensuring adherence to regulatory requirements. We uphold the belief that privacy is a basic human right that must be respected and protected. The need for privacy arises naturally from social interactions and collaborations among individuals. To enable effective collaboration and protect collective interests, societies have established universally recognized norms and regulatory frameworks across various domains. Adhering to these rules is equally important, as a fair and transparent environment encourages participation by fostering trust in a secure future where rights and properties are safeguarded. A system built on these principles would not only thrive with robust participation but also evolve continuously, improving over time while maintaining full liveness and fairness.

This thesis examines various existing blockchain applications and categorizes them into three distinct scenario settings.

We begin by exploring a scenario where compliance requirements take priority at the expense of privacy. A prominent example is the centralized cryptocurrency exchange platform, which functions as an online marketplace for exchanging digital assets, such as Bitcoin and Ethereum, with traditional fiat currencies (e.g., USD, EUR) or other cryptocurrencies, as seen in platforms like Binance [28] and Coinbase [29]. These platforms typically enforce

Know Your Customer (KYC) and Anti-Money Laundering (AML) regulations, requiring users to submit personal information, such as real identities and tax documentation. While these measures are critical for regulatory compliance, they inherently undermine user privacy by linking real-world identities to blockchain transactions. This connection exposes a user's entire transaction history on the blockchain, raising serious privacy concerns.

To address this challenge, we propose a private and compliant cryptocurrency exchange system [32] that restores user anonymity while upholding regulatory compliance. This system ensures users cannot double-spend and requires accurate reporting of accumulated profits for tax purposes, all within a privacy-preserving framework. Our design features a robust model and an efficient construction that achieves constant computational and communication overhead through the use of simple cryptographic techniques and rigorous security guarantees. The practical implementation demonstrates strong performance, affirming its feasibility for real-world deployment and its potential to resolve the privacy-compliance trade-off in cryptocurrency exchanges.

The second setting positions privacy as an integral component of compliance, reflecting its growing recognition as a fundamental regulatory requirement. A prime example is the dark pool, which illustrates how privacy is not merely a feature but a regulatory necessity. Dark pools are private financial exchanges or trading venues designed to facilitate large orders, typically from institutional investors, while maintaining confidentiality. This privacy aims to minimize market impact, ensuring that substantial trades do not influence stock prices prior to execution. Such measures align closely with regulatory mandates in many jurisdictions to promote fairness and uphold market stability.

We present Charon, a secure and efficient dark pool system built around a centralized platform acting as an intermediary. By leveraging secure multiparty computation, Charon ensures match eligibility and trade volume matching while preserving the confidentiality of critical trading information, including assets, trading direction, and volume. The system is designed to resist both semi-honest platforms and malicious traders, ensuring robust security. Two volume-matching constructions are introduced: one achieves $O(\log n)$ rounds of secure comparison with strong privacy guarantees, while the other improves performance through

constant-round parallel comparison with enhanced privacy protections. Both constructions have been implemented and rigorously evaluated, demonstrating their practical effectiveness in secure and compliant dark pool operations.

Thirdly, we explore the scenario where privacy is exploited, necessitating robust compliance measures. Certain blockchain designs prioritize privacy to such an extent that compliance is overlooked, creating opportunities for malicious actors to abuse the system. This misuse often results in financial losses or reputational harm for honest participants. One such vulnerability is the concurrency attack in the gas fee relaying system of Anonymous Zether. In this attack, a user sends a single meta-transaction to multiple relayers. Each relayer incurs the gas fee to process the transaction, but only one receives compensation, leaving the others with financial losses.

To mitigate this issue, we propose a novel mechanism called abuse deterring anonymous credential (ADAC), designed to detect when a user signs the same tag multiple times, even across credentials issued by different entities. We further present a concrete scheme that integrates ADAC into the relaying mechanism, effectively preventing gas fee abuse. This solution preserves the anonymity of honest users while ensuring fair compensation for relayers. Honest senders remain unaffected, and relayers targeted by attacks are appropriately reimbursed, fostering a secure and compliant environment without compromising privacy.

Lastly, beyond the three application-specific studies, we explore foundational cryptographic primitives to address privacy and compliance challenges. We introduce the concept of predicate aggregate signatures (PAS) [33], driven by requirements in blockchain governance. PAS seamlessly integrates privacy protections into compliance-focused environments, enabling users to sign multiple messages while preserving their anonymity. A combiner aggregates these individual signatures into a single concise representation that discloses only high-level descriptions of the signers for each message. Crucially, the aggregated signature guarantees that the signers and their descriptions adhere to a predefined public predicate aligned with compliance mandates. We provide a formal definition of PAS and propose a construction framework achieving logarithmic efficiency in both signature size and verification time.

A prominent application of PAS is in blockchain governance, where users engage in decision-making by voting, typically represented by posting signatures on the blockchain. In more complex scenarios, voting policies can involve conditions based on the identities of the voters. For example, in blockchain governance settings like Decentralized Autonomous Organizations (DAOs), decisions may require participation from the broader community, with the stipulation that accounts hold a certain number of tokens. Some policies further require that voters possess specific attributes [123], which can be represented by their identity indices. For instance, within an organization, senior members might be identified by indices below a certain threshold. The combiner's role is to verify that at least one signer satisfies this condition. In our design, leveraging a binary vector, the combiner demonstrates compliance by confirming the presence of at least one position in the vector with a value of 1 and an index below the specified threshold. This approach ensures efficient and privacy-preserving compliance with complex voting policies.

We summarize the following effective approaches that can be used to balance privacy and compliance in blockchain systems.

Data minimization and protection. This strategy emphasizes collecting and storing only the essential user data required for compliance. By leveraging advanced cryptographic techniques such as blind signatures, anonymous credentials, and zero-knowledge proofs (ZKPs), users can generate valid transactions without exposing detailed personal information. This approach minimizes the risk of privacy breaches and reduces the potential for misuse of sensitive user data while still fulfilling regulatory requirements.

Proof of compliance. This method allows users to demonstrate compliance with regulatory requirements without revealing sensitive information. Users hold cryptographically signed credentials, which regulators can verify. For example, a user may prove their tax compliance by providing zero-knowledge proof confirming their financial obligations' accuracy without disclosing detailed financial records. This approach maintains transparency with regulators while safeguarding user privacy.

Punishment with collateral. This approach ensures compliance by requiring users to lock collateral, which can be forfeited if they violate rules. The locked collateral incentivizes users to adhere to the regulations while avoiding intrusive monitoring mechanisms. For instance, users deposit assets into a smart contract as collateral before participating in the system. If a compliance violation, such as fraud or money laundering, is detected, the collateral is confiscated as a penalty. This mechanism fosters trust and accountability in decentralized systems.

7.2 Future Outlook

The interplay between privacy and compliance in blockchain applications remains a dynamic and evolving research area. While this thesis presents several concrete constructions and efficient frameworks, many open challenges and opportunities lie ahead. Below, we outline some promising directions for future work:

Auditing and accountability frameworks. Developing privacy-preserving auditing mechanisms that enable regulatory oversight without compromising user privacy is an exciting direction. For example, in privacy-preserving exchanges, all transaction details are hidden. This increases the concern of solvency issues, as users might make large, private exchanges of certain cryptocurrencies, potentially exceeding the platform's reserves and without its awareness. However, a rigorous solution to address the minimal reserve requirement remains an open challenge. Future work could focus on building efficient and scalable auditing frameworks using tools like zk-SNARKs or multi-party computation.

Enhanced privacy-compliant systems. Developing more advanced cryptographic protocols to achieve privacy-compliant systems is a key area for exploration. For example, Zcash and Monero are widely recognized as leading privacy-focused cryptocurrencies. However, their lack of regulatory oversight has led to concerns in certain countries, particularly regarding money laundering and other illicit activities. It is challenging to integrate them with compliance with financial regulations, such as KYC and AML standards. One promising direction

for making these cryptocurrencies compliant without altering their underlying protocol or consensus mechanism is through the use of layer-2 solutions.

Layer-2 solutions, which operate on top of the existing layer-1 blockchain, offer a way to introduce compliance features without changing the core functionality or privacy mechanisms of Zcash and Monero. These solutions can enable regulatory compliance, such as transaction monitoring, reporting, or identity verification, while preserving the base layer's privacy guarantees. Such solutions could provide a framework for the continued adoption of privacy-enhancing cryptocurrencies in jurisdictions with strict regulatory requirements, ensuring that privacy and compliance coexist without compromising the fundamental principles of decentralized networks.

Privacy-enhanced governance models. Blockchain governance models that incorporate privacy-preserving technologies, such as predicate aggregate signatures, are promising but still nascent. Exploring more expressive predicates, adaptive governance frameworks, and lightweight signature schemes could yield more flexible and efficient solutions. Such advancements could broaden the applicability of private governance in complex decision-making scenarios.

Interoperability and cross-chain privacy. As the blockchain ecosystem continues to expand, enabling interoperability between different chains while maintaining privacy and compliance is an urgent challenge. Future research could explore cryptographic methods for private asset transfers and governance across chains. Solutions like zero-knowledge rollups and threshold-based mechanisms could be adapted for this purpose, enabling seamless and secure cross-chain operations.

Decentralized Identity (DID) integration. Decentralized identity frameworks offer an opportunity to enhance privacy and compliance simultaneously. Research into integrating DID systems with blockchain platforms could provide users with greater control over their personal data while simplifying compliance processes. Future systems could use advanced credentials and selective disclosure protocols to meet evolving regulatory standards.

Post-quantum security. Additionally, future research will focus on integrating post-quantum cryptography (PQC) into privacy-preserving protocols. As quantum computing advances, existing cryptographic tools (e.g., elliptic curve-based zero-knowledge proofs or signatures) may become vulnerable. Investigating quantum-resistant alternatives, such as lattice-based zero-knowledge proofs, hash-based commitments, and post-quantum threshold cryptography, will ensure long-term security.

Real-world deployment. This research lays the foundation for privacy-preserving and compliant blockchain systems, but several avenues remain open for future exploration and practical enhancement. To bridge the gap between theoretical research and practical adoption, we outline a roadmap with a phased strategy for deploying our privacy-preserving and compliant solutions into existing ecosystems. Firstly, we would refine our prototypes and release open-source libraries and toolkits with clear documentation. Then, we plan to actively seek collaboration with industry partners, such as regulated financial institutions, cryptocurrency exchanges, and compliance technology providers. By establishing real-world testing environments, these collaborations could validate the practicality of privacy-preserving technologies under real regulatory mechanisms, including performance benchmarking, auditability checks, and interoperability with legacy systems. Additionally, we plan to join more and more standardization and ecosystem collaboration. For example, participating in standardization efforts (e.g., ISO, W3C, ITU) to define interoperable privacy-preserving compliance standards, and engaging with open-source communities (e.g., Ethereum Privacy groups) to encourage widespread adoption and continuous improvement.

In summary, by addressing these challenges and exploring these directions, we believe the blockchain community can build systems that strike a harmonious balance between privacy and compliance, fostering trust and innovation across a wide range of applications.

Bibliography

[1] Alan F Westin. 'Privacy and freedom'. In: *Washington and Lee Law Review* 25.1 (1968), p. 166.

[2] David J Seipp. 'The right to privacy in nineteenth century America'. In: *Harvard Law Review* 94.8 (1981), p. 1892.

[3] Nitish Singh and Thomas J Bussen. *Compliance management: a how-to guide for executives, lawyers, and other compliance professionals*. Bloomsbury Publishing USA, 2015.

[4] *Customer identification and verification*. https://www.austrac.gov.au/business/core-guidance/customer-identification-and-verification. 2024.

[5] *KYC and AML - What is the difference?* https://complyadvantage.com/insights/kyc-aml-know-your-customer-vs-anti-money-laundering/. 2024.

[6] Satoshi Nakamoto. 'Bitcoin: A peer-to-peer electronic cash system'. In: *Satoshi Nakamoto* (2008).

[7] Gavin Wood et al. 'Ethereum: A secure decentralised generalised transaction ledger'. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[8] *Zcash*. 2023. URL: %7Bhttps://z.cash%7D.

[9] Nicolas Van Saberhagen. 'CryptoNote v 2.0'. In: (2013).

[10] *With governments cracking down on privacy coins, is harsher crypto regulation in store?* https://www.disruptionbanking.com/2023/01/09/with-governments-cracking-down-on-privacy-coins-is-harsher-crypto-regulation-in-store/. 2023.

[11] *How to Complete Identity Verification for a Personal Account?* https://www.binance.com/en/support/faq/detail/360027287111?_dp=Ym5jOi8vYXBwLmJpbmFuY2UuY29tL3dlYnZpZXcvd2Vidmlldz90eXBlPWRlZmF1bHQm 2019.

[12] *Frequently asked questions on virtual currency transactions*. https://www.irs.gov/individuals/international-taxpayers/frequently-asked-questions-on-virtual-currency-transactions. 2024.

[13] *Cyber-related Designation*. https://ofac.treasury.gov/recent-actions/20220808. 2022.

[14] Eli Ben-Sasson et al. 'Zerocash: Decentralized Anonymous Payments from Bitcoin'. In: *IEEE S&P*. IEEE Computer Society, 2014, pp. 459–474.

[15] Ian Miers et al. 'Zerocoin: Anonymous distributed e-cash from bitcoin'. In: *2013 IEEE symposium on security and privacy*. IEEE. 2013, pp. 397–411.

[16] Benedikt Bünz et al. 'Zether: Towards privacy in a smart contract world'. In: *FC*. Springer. 2020, pp. 423–443.

[17] Benjamin E. Diamond. 'Many-out-of-Many Proofs and Applications to Anonymous Zether'. In: *IEEE S&P*. IEEE, 2021, pp. 1800–1817.

[18] *Introducing Private Transactions On Ethereum NOW!* https://tornado-cash.medium.com/introducing-private-transactions-on-ethereum-now-42ee915babe0. 2019.

[19] Ethan Heilman et al. 'TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub'. In: *NDSS*. The Internet Society, 2017.

[20] Erkan Tairi, Pedro Moreno-Sanchez and Matteo Maffei. 'A^{2}L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs'. In: *IEEE S&P*. IEEE, 2021, pp. 1834–1851.

[21] Noemi Glaeser et al. 'Foundations of coin mixing services'. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 1259–1273.

[22] Xianrui Qin et al. ‘BlindHub: Bitcoin-Compatible Privacy-Preserving Payment Channel Hubs Supporting Variable Amounts’. In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society. 2022, pp. 2020–2038.

[23] Lucien K. L. Ng et al. ‘LDSP: Shopping with Cryptocurrency Privately and Quickly under Leadership’. In: *ICDCS*. IEEE, 2021, pp. 261–271.

[24] Yu Chen et al. ‘PGC: Decentralized confidential payment system with auditability’. In: *ESORICS*. Springer. 2020, pp. 591–610.

[25] Elli Androulaki et al. ‘Privacy-Preserving Auditable Token Payments in a Permissioned Blockchain System’. In: *AFT*. Association for Computing Machinery, 2020, pp. 255–267.

[26] Alin Tomescu et al. ‘UTT: Decentralized Ecash with Accountable Privacy’. In: *IACR Cryptol. ePrint Arch.* (2022), p. 452.

[27] Karl Wüst, Kari Kostiainen and Srdjan Capkun. ‘Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy Preserving Regulation’. In: *IACR Cryptol. ePrint Arch.* (2021), p. 1443.

[28] *Binance*. https://www.binance.com/en.

[29] *Coinbase*. https://www.coinbase.com.

[30] *General Data Protection Regulation (GDPR)*. https://gdpr-info.eu/. 2016.

[31] *California Consumer Privacy Act (CCPA)*. https://oag.ca.gov/privacy/ccpa. 2024.

[32] Ya-Nan Li, Tian Qiu and Qiang Tang. ‘Pisces: Private and Compliable Cryptocurrency Exchange’. In: *NDSS*. The Internet Society, 2024.

[33] Tian Qiu and Qiang Tang. ‘Predicate Aggregate Signatures and Applications’. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2023, pp. 279–312.

[34] Masayuki Abe et al. ‘Structure-preserving signatures and commitments to group elements’. In: *Journal of Cryptology* 29 (2016), pp. 363–421.

[35] Dan Boneh et al. ‘Aggregate and verifiably encrypted signatures from bilinear maps’. In: *Advances in Cryptology—EUROCRYPT 2003: International Conference on the*

Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22. Springer. 2003, pp. 416–432.

[36] Vanesa Daza, Carla Ràfols and Alexandros Zacharakis. 'Updateable inner product argument with logarithmic verifier and applications'. In: *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part I 23*. Springer. 2020, pp. 527–557.

[37] Mihir Bellare and Phillip Rogaway. 'Random oracles are practical: A paradigm for designing efficient protocols'. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. 1993, pp. 62–73.

[38] Masayuki Abe and Tatsuaki Okamoto. 'Provably Secure Partially Blind Signatures'. In: *CRYPTO*. Springer, 2000, pp. 271–286.

[39] Mihir Bellare and Gregory Neven. 'Multi-signatures in the plain public-key model and a general forking lemma'. In: *Proceedings of the 13th ACM conference on Computer and communications security*. 2006, pp. 390–399.

[40] Thomas Ristenpart and Scott Yilek. 'The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks'. In: *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26*. Springer. 2007, pp. 228–245.

[41] David Pointcheval and Olivier Sanders. 'Short Randomizable Signatures'. In: *CT-RSA*. Springer, 2016, pp. 111–126.

[42] Ran Canetti. 'Security and composition of multiparty cryptographic protocols'. In: *Journal of CRYPTOLOGY* 13 (2000), pp. 143–202.

[43] *Binance Revenue and Usage Statistics (2022)*. https://www.businessofapps.com/data/binance-statistics/. Sept. 2022.

[44] *Coinbase Revenue and Usage Statistics (2022)*. https://www.businessofapps.com/data/coinbase-statistics/. Sept. 2022.

[45] *Data Breaches*. https://www.coindesk.com/tag/data-breaches/. Oct. 2022.

[46] Matthew Green and Ian Miers. 'Bolt: Anonymous Payment Channels for Decentralized Currencies'. In: *CCS*. ACM, 2017, pp. 473–489.

[47] Kristian Gjøsteen, Mayank Raikwar and Shuang Wu. 'PriBank: Confidential Blockchain Scaling Using Short Commit-and-Proof NIZK Argument'. In: *CT-RSA*. Springer, 2022, pp. 589–619.

[48] Sean Bowe et al. 'Zexe: Enabling decentralized private computation'. In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 947–964.

[49] Carsten Baum, Bernardo David and Tore Kasper Frederiksen. 'P2DEX: Privacy-Preserving Decentralized Cryptocurrency Exchange'. In: *ACNS (1)*. Springer, 2021, pp. 163–194.

[50] Shumo Chu, Qiudong Xia and Zhenfei Zhang. 'Manta: Privacy Preserving Decentralized Exchange'. In: *IACR Cryptol. ePrint Arch.* (2020), p. 1607.

[51] Apoorvaa Deshpande and Maurice Herlihy. 'Privacy-preserving cross-chain atomic swaps'. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2020, pp. 540–549.

[52] Xun Yi and Kwok-Yan Lam. 'A new blind ECDSA scheme for bitcoin transaction anonymity'. In: *AsiaCCS*. ACM, 2019, pp. 613–620.

[53] *Understanding crypto taxes*. https://www.coinbase.com/learn/crypto-basics/understanding-crypto-taxes. 2023.

[54] Nir Bitansky et al. 'Succinct non-interactive arguments via linear interactive proofs'. In: *TCC*. Springer. 2013, pp. 315–333.

[55] Jens Groth and Markulf Kohlweiss. 'One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin'. In: *EUROCRYPT (2)*. Springer, 2015, pp. 253–280.

[56] Roger Dingledine, Nick Mathewson and Paul Syverson. 'Tor: The Second-Generation Onion Router'. In: *USENIX Security*. USENIX Association, 2004.

[57] *Tor Browser*. https://www.torproject.org/. 2023.

[58] Lindell. 'Parallel coin-tossing and constant-round secure two-party computation'. In: *Journal of Cryptology* 16 (2003), pp. 143–184.

[59] *Basel Accords: Purpose, Pillars, History, and Member Countries*. https://www.investopedia.com/terms/b/basel_accord.asp. Apr. 2022.

[60] Johannes Blömer et al. 'Updatable Anonymous Credentials and Applications to Incentive Systems'. In: *CCS*. ACM, 2019, pp. 1671–1685.

[61] Gaby G. Dagher et al. 'Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges'. en. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. Denver Colorado USA: ACM, Oct. 2015, pp. 720–731. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813674. URL: https://dl.acm.org/doi/10.1145/2810103.2813674 (visited on 17/05/2022).

[62] *Currency composition of International Foreign Reserves*. https://data.imf.org/?sk=e6a5f467-c14b-4aa8-9f6d-5a09ec4e62a4. Apr. 2023.

[63] Torben P. Pedersen. 'Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing'. In: *CRYPTO*. Ed. by Joan Feigenbaum. Springer, 1991, pp. 129–140.

[64] Alex Davidson et al. 'Privacy Pass: Bypassing Internet Challenges Anonymously'. In: *Proc. Priv. Enhancing Technol.* 2018.3 (2018), pp. 164–180.

[65] *CFTC Regulation 43.6*. https://www.ecfr.gov/current/title-17/chapter-I/part-43/section-43.6. 2025.

[66] U.S. Attorney. *U.S. Attorney Announces Agreements With Morgan Stanley And Former Senior Employee, Pawan Passi, In Connection With Deceptive Practices In Block Trades Business*. https://www.justice.gov/usao-sdny/pr/us-attorney-announces-agreements-morgan-stanley-and-former-senior-employee-pawan-passi. 2024.

[67] *Overstock subsidiary tZERO fined $800,000 over dark pool rule violation*. https://coingeek.com/overstock-subsidiary-tzero-fined-800000-over-dark-pool-rule-violation/. 2022.

[68] Gilad Asharov et al. 'Privacy-Preserving Dark Pools'. In: *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 1747–1749.

[69] John Cartlidge, Nigel P. Smart and Younes Talibi Alaoui. 'MPC Joins The Dark Side'. en. In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. Auckland New Zealand: ACM, July 2019, pp. 148–159. ISBN: 978-1-4503-6752-3. DOI: 10.1145/3321705.3329809. URL: https://dl.acm.org/doi/10.1145/3321705.3329809 (visited on 30/03/2022).

[70] John Cartlidge, Nigel P. Smart and Younes Talibi Alaoui. 'Multi-party computation mechanism for anonymous equity block trading: A secure implementation of turquoise plato uncross'. In: *Intell. Syst. Account. Finance Manag.* 28.4 (2021), pp. 239–267.

[71] Mariana Botelho da Gama et al. 'Kicking-the-Bucket: Fast Privacy-Preserving Trading Using Buckets'. In: *Financial Cryptography*. Vol. 13411. Lecture Notes in Computer Science. Springer, 2022, pp. 20–37.

[72] Mariana Botelho da Gama et al. 'All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation'. In: *IACR Cryptol. ePrint Arch.* (2022), p. 923.

[73] Sahar Mazloom et al. 'An Efficient Data-Independent Priority Queue and its Application to Dark Pools'. In: *Proc. Priv. Enhancing Technol.* 2023.2 (2023), pp. 5–22.

[74] Craig Gentry. 'Fully homomorphic encryption using ideal lattices'. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 169–178.

[75] Junfeng Fan and Frederik Vercauteren. 'Somewhat practical fully homomorphic encryption'. In: *Cryptology ePrint Archive* (2012).

[76] Tucker Balch, Benjamin E. Diamond and Antigoni Polychroniadou. 'SecretMatch: inventory matching from fully homomorphic encryption'. In: *ICAIF*. ACM, 2020, 15:1–15:7.

[77] Antigoni Polychroniadou et al. 'Prime Match: A Privacy-Preserving Inventory Matching System'. In: *USENIX Security Symposium*. 2023, pp. 6417–6434.

[78] *Can You Swim in a Dark Pool?* https://www.finra.org/investors/insights/can-you-swim-dark-pool. 2023.

[79] Yulin Wu et al. 'Generic server-aided secure multi-party computation in cloud computing'. In: *Computer Standards & Interfaces* 79 (2022), p. 103552.

[80] Payman Mohassel, Ostap Orobets and Ben Riva. 'Efficient server-aided 2pc for mobile phones'. In: *Proceedings on Privacy Enhancing Technologies* (2016).

[81] Boaz Barak et al. 'Secure Computation Without Authentication'. In: *CRYPTO*. Springer, 2005, pp. 361–377.

[82] Geoffroy Couteau. 'New Protocols for Secure Equality Test and Comparison'. In: *ACNS*. Springer, 2018, pp. 303–320.

[83] Tianpei Lu et al. 'Efficient 2PC for Constant Round Secure Equality Testing and Comparison'. In: *Cryptology ePrint Archive* (2024).

[84] Lijing Zhou et al. 'Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning'. In: *SP*. IEEE. 2023, pp. 534–551.

[85] Eleftheria Makri et al. 'Rabbit: Efficient comparison for secure multi-party computation'. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2021, pp. 249–270.

[86] *What Is the National Best Bid and Offer (NBBO)? How Quote Works*. https://www.investopedia.com/terms/n/nbbo.asp. 2022.

[87] Lucjan Hanzlik and Daniel Slamanig. 'With a little help from my friends: Constructing practical anonymous credentials'. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 2004–2023.

[88] Amos Fiat and Adi Shamir. 'How to prove yourself: Practical solutions to identification and signature problems'. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194.

[89] C. P. Schnorr. 'Efficient Identification and Signatures for Smart Cards'. In: *Advances in Cryptology — CRYPTO' 89 Proceedings*. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 239–252. ISBN: 978-0-387-34805-6.

[90] Pascal Paillier. 'Public-Key Cryptosystems Based on Composite Degree Residuosity Classes'. In: *EUROCRYPT*. Springer, 1999, pp. 223–238.

[91] Marcel Keller. 'MP-SPDZ: A Versatile Framework for Multi-Party Computation'. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020.

[92] Marcel Keller, Emmanuela Orsini and Peter Scholl. 'MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer'. In: Association for Computing Machinery, 2016. ISBN: 9781450341394.

[93] *Aztec network*. 2025. URL: https://aztec.network/.

[94] *Railgun*. 2025. URL: https://www.railgun.org/.

[95] kaleido. *Zero Knowledge Token Explained*. https://docs.kaleido.io/kaleido-services/token-zkp/architecture/. 2025.

[96] *Infura*. https://www.infura.io/. 2024.

[97] *Biconomy*. https://www.biconomy.io/. 2024.

[98] *ERC-1077: Gas relay for contract calls*. https://eips.ethereum.org/EIPS/eip-1077. 2018.

[99] *Why did I pay gas fees for a failed transaction?* https://support.metamask.io/transactions-and-gas/gas-fees/why-did-i-pay-gas-fees-for-a-failed-transaction/. 2024.

[100] Gaurav Sharma et al. 'Anonymous sealed-bid auction on ethereum'. In: *Electronics* 10.19 (2021), p. 2340.

[101] Zongli Ye et al. 'An anonymous and fair auction system based on blockchain'. In: *The Journal of Supercomputing* 79.13 (2023), pp. 13909–13951.

[102] Ronald L Rivest, Adi Shamir and Yael Tauman. 'How to leak a secret: Theory and applications of ring signatures'. In: *Essays in memory of Shimon Even* 3895 (2006), pp. 164–186.

[103] David Chaum. 'Security without identification: Transaction systems to make big brother obsolete'. In: *Communications of the ACM* 28.10 (1985), pp. 1030–1044.

[104] Jan Camenisch and Anna Lysyanskaya. 'An efficient system for non-transferable anonymous credentials with optional anonymity revocation'. In: *Advances in Cryptology—EUROCRYPT 2001: International Conference on the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings 20*. Springer. 2001, pp. 93–118.

[105] David Chaum, Amos Fiat and Moni Naor. 'Untraceable electronic cash'. In: *CRYPTO*. Springer. 1988, pp. 319–327.

[106] Aggelos Kiayias, Yiannis Tsiounis and Moti Yung. 'Traceable signatures'. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2004, pp. 571–589.

[107] Eiichiro Fujisaki and Koutarou Suzuki. 'Traceable ring signature'. In: *International Workshop on Public Key Cryptography*. Springer. 2007, pp. 181–200.

[108] Man Ho Au, Willy Susilo and Siu-Ming Yiu. 'Event-oriented k-times revocable-iff-linked group signatures'. In: *Information Security and Privacy: 11th Australasian Conference, ACISP 2006, Melbourne, Australia, July 3-5, 2006. Proceedings 11*. Springer. 2006, pp. 223–234.

[109] Jan Camenisch, Susan Hohenberger and Anna Lysyanskaya. 'Compact e-cash'. In: *Lecture Notes in Computer Science* 3494 (2005), pp. 302–321. ISSN: 03029743. DOI: 10.1007/11426639{_}18.

[110] Foteini Baldimtsi and Anna Lysyanskaya. 'Anonymous credentials light'. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 1087–1098.

[111] Tim Ruffing, Aniket Kate and Dominique Schröder. 'Liar, liar, coins on fire! Penalizing equivocation by loss of bitcoins'. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 219–230.

[112] Bertram Poettering and Douglas Stebila. 'Double-authentication-preventing signatures'. In: *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I 19*. Springer. 2014, pp. 436–453.

[113] Johannes Blömer et al. 'Updatable Anonymous Credentials and Applications to Incentive Systems'. en. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. London United Kingdom: ACM, Nov. 2019, pp. 1671–1685. ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3354223. URL: https://dl.acm.org/doi/10.1145/3319535.3354223 (visited on 30/03/2022).

[114] Aggelos Kiayias and Qiang Tang. 'How to keep a secret: leakage deterring public-key cryptosystems'. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 943–954.

[115] Aggelos Kiayias and Qiang Tang. 'Traitor deterring schemes: Using bitcoin as collateral for digital content'. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 231–242.

[116] F Betül Durak et al. 'Non-Transferable Anonymous Tokens by Secret Binding'. In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 2024, pp. 2460–2474.

[117] Hanwen Feng and Qiang Tang. 'Witness authenticating NIZKs and applications'. In: *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41*. Springer. 2021, pp. 3–33.

[118] Paul Gerhart et al. 'Foundations of Adaptor Signatures'. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2024, pp. 161–189.

[119] *DAOs*. https://ethereum.org/en/dao/. 2023.

[120] *Quadratic Moloch*. https://github.com/DemocracyEarth/dao. 2019.

[121] *Quadratic voting in colorado: 2020*. https://www.radicalxchange.org/media/blog/quadratic-voting-in-colorado-2020/. Jan. 2021.

[122] *How to Delegate Votes in the Unlock DAO*. https://unlock-protocol.com/guides/delegation/. May 2023.

[123] *Erc721 Voting-Power based on some property*. https://forum.openzeppelin.com/t/erc721-voting-power-based-on-some-property/24550. Feb. 2022.

[124] Aggelos Kiayias, Murat Osmanoglu and Qiang Tang. 'Graded signatures'. In: *Information Security: 18th International Conference, ISC 2015, Trondheim, Norway, September 9-11, 2015, Proceedings 18*. Springer. 2015, pp. 61–80.

[125] John Bethencourt, Elaine Shi and Dawn Song. 'Signatures of reputation'. In: *Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14*. Springer. 2010, pp. 400–407.

[126] Benedikt Bunz et al. ‘Bulletproofs: Short Proofs for Confidential Transactions and More’. en. In: *2018 IEEE Symposium on Security and Privacy (SP)*. San Francisco, CA: IEEE, May 2018, pp. 315–334. ISBN: 978-1-5386-4353-2. DOI: 10.1109/SP.2018.00020. URL: https://ieeexplore.ieee.org/document/8418611/ (visited on 07/04/2022).

[127] Jonathan Lee. ‘Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments’. In: *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II*. Springer. 2021, pp. 1–34.

[128] Victor Shoup. ‘Practical threshold signatures’. In: *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*. Springer. 2000, pp. 207–220.

[129] Dan Boneh, Manu Drijvers and Gregory Neven. ‘Compact multi-signatures for smaller blockchains’. In: *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part II*. Springer. 2018, pp. 435–464.

[130] Silvio Micali et al. ‘Compact certificates of collective knowledge’. In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 626–641.

[131] Emmanuel Bresson, Jacques Stern and Michael Szydlo. ‘Threshold ring signatures and applications to ad-hoc groups’. In: *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings*. Springer. 2002, pp. 465–480.

[132] Hemanta K Maji, Manoj Prabhakaran and Mike Rosulek. ‘Attribute-based signatures’. In: *Topics in Cryptology–CT-RSA 2011: The Cryptographers’ Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*. Springer. 2011, pp. 376–392.

[133] Russell WF Lai et al. 'Multi-key homomorphic signatures unforgeable under insider corruption'. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2018, pp. 465–492.

[134] Jonathan Bootle et al. 'DualDory: Logarithmic-Verifier Linkable Ring Signatures Through Preprocessing'. In: *Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part II*. Springer. 2022, pp. 427–446.

[135] Joseph K Liu, Victor K Wei and Duncan S Wong. 'Linkable spontaneous anonymous group signature for ad hoc groups'. In: *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings 9*. Springer. 2004, pp. 325–335.

[136] Sherman SM Chow, Jack PK Ma and Tsz Hon Yuen. 'Scored Anonymous Credentials'. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2023, pp. 484–515.

[137] Sanjam Garg et al. 'hinTS: Threshold Signatures with Silent Setup'. In: *Cryptology ePrint Archive* (2023).

[138] Sourav Das et al. 'Threshold Signatures from Inner Product Argument: Succinct, Weighted, and Multi-threshold'. In: *Cryptology ePrint Archive* (2023).

[139] Georg Fuchsbauer, Eike Kiltz and Julian Loss. 'The algebraic group model and its applications'. In: *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II 38*. Springer. 2018, pp. 33–62.

[140] Mark Zhandry. 'To label, or not to label (in generic groups)'. In: *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part III*. Springer. 2022, pp. 66–96.

[141] Thomas Attema, Ronald Cramer and Matthieu Rambaud. 'Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold

signatures'. In: *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV*. Springer. 2021, pp. 526–556.

[142] Jonathan Bootle et al. 'Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting'. In: *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer. 2016, pp. 327–357.

[143] Leemon Baird et al. 'Threshold Signatures in the Multiverse'. In: *Cryptology ePrint Archive* (2023).

[144] Johannes Blömer, Jakob Juhnke and Christina Kolb. 'Anonymous and publicly linkable reputation systems'. In: *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*. Springer. 2015, pp. 478–488.

[145] Johannes Blömer, Jan Bobolz and Laurens Porzenheim. 'A Generic Construction of an Anonymous Reputation System and Instantiations from Lattices'. In: *Cryptology ePrint Archive* (2023).

[146] Ali El Kaafarani, Shuichi Katsumata and Ravital Solomon. 'Anonymous reputation systems achieving full dynamicity from lattices'. In: *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*. Springer. 2018, pp. 388–406.

[147] Pyrros Chaidos and Aggelos Kiayias. 'Mithril: Stake-based threshold multisignatures'. In: *Cryptology ePrint Archive* (2021).

[148] Dan Boneh and Chelsea Komlo. 'Threshold signatures with private accountability'. In: *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV*. Springer. 2022, pp. 551–581.