PGC: Decentralized Confidential Payment System with Auditability

Yu Chen [*] Xuecheng Ma [†] Cong Tang [‡] Man Ho Au [§]

Abstract

Modern cryptocurrencies such as Bitcoin and Ethereum achieve decentralization by replacing a trusted center with a distributed and append-only ledger (known as blockchain). However, removing this trusted center comes at significant cost of privacy due to the public nature of blockchain. Many existing cryptocurrencies fail to provide transaction anonymity and confidentiality, meaning that addresses of sender, receiver and transfer amount are publicly accessible. As the privacy concerns grow, a number of academic work have sought to enhance privacy by leveraging cryptographic tools. Though strong privacy is appealing, it might be abused in some cases. In decentralized payment systems, anonymity poses great challenges to system's auditability, which is a crucial property for scenarios that require regulatory compliance and dispute arbitration guarantee.

Aiming for a middle ground between privacy and auditability, we introduce the notion of *decentralized confidential payment* (DCP) system with auditability. In addition to offering transaction confidentiality, DCP supports privacy-preserving audit in which an external party can specify a set of transactions and then request the participant to prove their compliance with a large class of policies. We present a generic construction of auditable DCP system from integrated signature and encryption scheme and non-interactive zero-knowledge proof systems. We then instantiate our generic construction by carefully designing the underlying building blocks, yielding a standalone cryptocurrency called PGC. In PGC, the setup is transparent, transactions are less than 1.3KB and take under 38ms to generate and 15ms to verify.

At the core of PGC is an additively homomorphic public-key encryption scheme that we introduce, twisted ElGamal, which is not only as secure as standard exponential ElGamal, but also friendly to Sigma protocols and range proofs. This enables us to easily devise zero-knowledge proofs for basic correctness of transactions as well as various application-dependent policies in a modular fashion. Moreover, it is very efficient. Compared with the most efficient reported implementation of Paillier PKE, twisted ElGamal is an order of magnitude better in key and ciphertext size and decryption speed (for small message space), two orders of magnitude better in encryption speed. We believe twisted ElGamal is of independent interest on its own right. Along the way of designing and reasoning zero-knowledge proofs for PGC, we also obtain two interesting results. One is weak forking lemma which is a useful tool to prove computational knowledge soundness. The other is a trick to prove no-knowledge of discrete logarithm, which is a complement of standard proof of discrete logarithm knowledge.

Keywords: cryptocurrencies, decentralized payment system, confidential transactions, auditable, twisted ElGamal, weak forking lemma, proof of no-knowledge

[*] Shandong University. Email: yuchen.prc@gmail.com

[†] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences. School of Cyber Security, University of Chinese Academy of Sciences. Email: maxuecheng@iie.ac.cn

[‡] Peking University. Email: lemuria.tc@gmail.com

[§] The University of Hong Kong (HKU). Email: allenau@cs.hku.hk

Contents

1 Introduction

Unlike traditional centralized payment systems, modern cryptocurrencies such as Bitcoin [Nak08] and Ethereum [Woo14] realize decentralized peer-to-peer payment by maintaining an append-only public ledger known as blockchain, which is globally distributed and synchronized by consensus protocols. In blockchain-based payment systems, the correctness of each transaction must be verified by the miners publicly before being packed into a block. To enable efficient validation, major cryptocurrencies such as Bitcoin and Ethereum simply expose all transaction details (sender, receiver and transfer amount) to the public. Following the terminology in [BBB$^+$18], privacy for transactions consists of two aspects: (1) anonymity, hiding the identities of sender and receiver in a transaction and (2) confidentiality, hiding the transfer amount. While Bitcoin-like and Ethereum-like cryptocurrencies provide some weak anonymity through unlinkability of account addresses to real world identities, they lack confidentiality, which is of paramount importance.

1.1 Motivation

Auditability is a crucial property in all financial systems. Informally, it means that an auditor can not only check that if transactions satisfy some pre-fixed policies *before-the-fact*, but also request participants to prove his/her transactions comply with some policies *after-the-fact*. In centralized payment system where there exists a trusted center, such as bank, Paypal or Alipay, auditability is a built-in property since the center knows details of all transactions and thus be able to conduct audit. However, it is challenging to provide the same level of auditability in decentralized payment systems with strong privacy guarantee.

In our point of view, strong privacy is a double-edged sword. While confidentiality is arguably the primary concern of privacy for any payment system, anonymity might be abused or even prohibitive for applications that require auditability, because anonymity provides plausible deniability [FMMO19], which allows participants to deny their involvements in given transactions. Particularly, it seems that anonymity denies the feasibility of *after-the-fact* auditing, due to an auditor is unable to determine who is involved in which transaction. We exemplify this dilemma via the following three typical cases.

- Case 1: To prevent criminals from money laundering in a decentralized payment system, an auditor must be able to examine the details of suspicious transactions. However, if the system is anonymous, locating the participants of suspicious transactions and then force them to reveal transaction details would be very challenging.

- Case 2: Consider an employer pay salaries to employees via a decentralized payment system with strong privacy. If the employer did not receive the correct amount, how can he support his complaint? Likewise, how can the employer protect himself by demonstrating he has indeed completed payment with the correct amount.

- Case 3: Consider the employees also pay the income tax via the same payment system. How can he convince the tax office that he has paid the tax according to the correct tax rate.

Similar scenarios are ubiquitous in monetary systems that require after-the-fact auditing and in e-commerce systems that require dispute arbitration mechanism.

1.2 Our Contributions

Based on the above discussion, we conjecture that it may be impossible to construct a decentralized payment system offering the same level of auditability as centralized payment system while offering confidentiality and anonymity simultaneously without introducing some degree of centralization or trust assumption. In this work, we stick to confidentiality, but trade anonymity for auditability.

We summarize our contributions as follows.

Decentralized confidential payment system with auditability. We introduce the notion of *decentralized confidential payment* (DCP) system with auditability, and formalize a security model. For the sake of simplicity, we take an account-based approach and focus only on the *transaction layer*, and treat the network/consensus-level protocols as black-box and ignore attacks against them. Briefly, DCP should satisfy authenticity, confidentiality and soundness. The first security notion stipulates that only the owner of an account can spend his coin. The second security notion requires the account balance and transfer amount are hidden. The last security notion captures that no one is able to make validation nodes accept an incorrect transaction. In addition, we require the audit to be privacy-preserving, namely, the participant is unable to fool the auditor and the auditing results do not impact the confidentiality of other transactions.

We then present a generic construction of auditable DCP system from two building blocks, namely, integrated signature and encryption (ISE) schemes and non-interactive zero-knowledge (NIZK) proof systems. In our generic DCP construction, ISE plays an important role. First, it guarantees that each account can safely use a single keypair for both encryption and signing. This feature greatly simplifies the overall design from both conceptual and practical aspects. Second, the encryption component of ISE ensures that the resulting DCP system is *complete*, meaning that as soon as a transaction is recorded on the blockchain, the payment is finalized and takes effect immediately – receiver's balance increases with the same amount that sender's balance decreases, and the receiver can spend his coin on his will. This is in contrast to many commitment-based cryptocurrencies that require *out-of-band* transfer, which are thus not complete. NIZK not only enables a sender to prove transactions satisfy the most basic correctness policy, but also allows a user to prove that any set of confidential transactions he participated satisfy a large class of application-dependent policies, such as limit on total transfer amounts, paying tax rightly according the tax rate, and transaction opens to some exact value. In summary, our generic DCP construction is simple and complete, and supports flexible audit.

PGC: a simple and efficient instantiation. While the generic DCP construction is relatively simple and intuitive, an efficient instantiation is more technically involved. We realize our generic DCP construction by designing a new ISE and carefully devising suitable NIZK proof system. We refer to the resulting cryptocurrency as PGC (stands for Pretty Good Confidentiality). Notably, in addition to the advantages inherited from the generic construction, PGC admits transparent setup, and its security is based solely on the widely-used discrete logarithm assumption. To demonstrate the efficiency and usability of PGC, we implement it as a standalone cryptocurrency, and also deploy it as smart contracts. We report the experimental results in Section 8.

1.3 Technical Overview

We discuss our design choice in the generic construction of DCP, followed by techniques and tools towards a secure and efficient instantiation, namely, PGC.

1.3.1 Design Choice in Generic Construction of Auditable DCP

Pseudonymity vs. Anonymity. Early blockchain-based cryptocurrencies offers pseudonymity, that is, addresses are assumed to be unlinkable to their real world identities. However, a variety of de-anonymization attacks [RS13, BKP14] falsified this assumption. On the other hand, a number of cryptocurrencies such as Monero and Zcash sought to provide strong privacy guarantee, including both anonymity and confidentiality. In this work, we aim for finding a sweet balance between privacy and auditability. As indicated in [GGM16], identity is crucial to any regulatory system. Therefore, we choose to offer privacy in terms of confidentiality, and still stick to pseudonymous system. Interestingly, we view pseudonymity as a feature rather than a weakness, assuming that an auditor is able to link account addresses to real world identities. This opens the possibility to conduct after-the-fact audit. We believe this is the most promising avenue for real deployment of DCP that requires auditability.

PKE vs. Commitment. A common approach to achieve transaction confidentiality is to commit the balance and transfer amount using a global homomorphic commitment scheme (e.g., the Pedersen commitment [Ped91]), then derive a secret from blinding randomnesses to prove correctness of transaction and authorize transfer. The seminal DCP systems [Max, Poe] follow this approach.

Nevertheless, commitment-based approach suffers from several drawbacks. First, the resulting DCP systems are not *complete*. Due to lack of decryption capability, senders are required to honestly transmit the openings of outgoing commitments (includes randomness and amount) to receivers in an *out-of-band* manner. This issue makes the system much more complicated, as it must be assured that the out-of-band transfer is correct and secure. Second, users must be stateful since they have to keep track of the randomness and amount of each incoming commitment. Otherwise, failure to open a single incoming commitment will render an account totally unusable, due to either incapable of creating the NIZK proofs (lack of witness), or generating the signature (lack of signing key). This incurs extra security burden to the design of wallet (guarantee the openings must be kept in a safe and reliable way). Last but not the least, as stated by Bünz et al. [BBB+18], as Pedersen commitment is only computational binding based on discrete logarithm assumption, an adversary will be able to open a given commitment to an arbitrary value when quantum computers are available, which is a serious issue in any payment system.

Observe that homomorphic PKE can be viewed as a computationally hiding and perfectly binding commitment, in which the secret key serves as a natural trapdoor to recover message. With these factors in mind, our design equips each user with a PKE keypair rather than making all users share a global commitment.

Integrated Signature and Encryption vs. SIG+PKE. Intuitively, to secure a DCP system, we need a PKE scheme to provide confidentiality, and a signature scheme to provide authenticity. If we follow the principle of *key separation*, i.e., use different keypairs for encryption and signing operations respectively, the overall design would be complicated. In that case, each account will be associated with two keypairs, and consequently deriving account address turns out to be very tricky. If we derive the address from one public key, which one should be chosen? If we derive the address from the two public keys, then additional mechanism is needed to link the two public keys together.

A better solution is to adopt *key reuse* strategy, i.e., use the same keypair for both encryption and signing. This will greatly simplify the design of overall system. However, reusing keypairs may create new security problems. As pointed out in [PSST11], the two uses may interact with one another badly, in such a way as to undermine the security of one or both of the primitives, e.g., the case of textbook RSA encryption and signature. In this work, we propose to use integrated signature and encryption (ISE) scheme with *joint security*, wherein a single keypair is used for both signature and encryption components in a secure manner (cf. Section 2.5 for definition), to replace the naive combination of signature and encryption. To the best of our knowledge, this is the first time that ISE is used in DCP system to ensure provable security. We remark that the existing proposal Zether [BAZB20] essentially adopts the key reuse strategy, employing a signature scheme and an encryption scheme with same keypair. Nevertheless, they do not explicitly identify key reuse strategy and formally address joint security.

1.3.2 Overview of Our Generic Auditable DCP

DCP from ISE and NIZK. We present a generic construction of DCP system from ISE and NIZK. We choose the account-based model for simplicity and usability. In our generic DCP construction, user creates an account by generating a keypair of ISE, in which the public key is used as account address and secret key is used to control the account. The state of an account consists of a serial number (a counter that increments with every outgoing transaction) and an encrypted balance (encryption of plaintext balance under account public key). State changes are triggered by transactions from one account to another. A publicly accessible and append-only ledger called blockchain tracks the state of every account.

Let $\tilde{C}_s$ and $\tilde{C}_r$ be the encrypted balances of two accounts controlled by Alice and Bob respectively. Suppose Alice wishes to transfer v coins to Bob. She constructs a confidential transaction via the following steps. First, she encrypts v under her public key pk_s and Bob's public key pk_r respectively to obtain C_s and C_r, and sets $\mathsf{memo} = (pk_s, pk_r, C_s, C_r)$. Then, she produces a NIZK proof π_{correct} for the most basic correctness policy of transaction: (i) C_s and C_r are two encryptions of the same transfer amount under pk_s and pk_r; (ii) the transfer amount lies in a right range; (iii) her remaining balance is still positive. Finally, she signs serial number sn together with memo and π_{correct} under her secret key, obtaining a signature σ. The entire transaction is of the form $(\mathsf{sn}, \mathsf{memo}, \pi_{\text{correct}}, \sigma)$. In this way, validity of a transaction is publicly verifiable by checking the signature and NIZK proof. If the transaction is valid, it will be recorded on the blockchain. Accordingly, Alice's balance (resp. Bob's balance) will be updated as $\tilde{C}_s = \tilde{C}_s - C_s$ (resp. $\tilde{C}_r = \tilde{C}_r + C_r$), and Alice's serial number increments. Such balance update operation implicitly requires that the underlying PKE scheme satisfies additive homomorphism.

In summary, the signature component is used to provide authenticity (proving ownership of an account), the encryption component is used to hide the balance and transfer amount, while zero-knowledge proofs are used to prove the correctness of transactions in a privacy-preserving manner.

Auditing Policies. A trivial solution to achieve auditability is to make the participants reveal their secret keys. However, this approach will expose all the related transactions to the auditor, which is not privacy-preserving. Note that a plaintext and its ciphertext can be expressed as an $\mathcal{NP}$ relation. Auditability can thus be easily achieved by leveraging NIZK. Moreover, note that the structure of memo is symmetric. This allows either the sender or the receiver can prove transactions comply with a variety of application-dependent policies. We provide examples of several useful policies as below:

- Anti-money laundering: the sum of a collection of outgoing/incoming transactions from/to a particular account is limited.
- Tax payment: a user pays the tax according to the tax rate.
- Selectively disclosure: the transfer amount of some transaction is indeed some value.

1.3.3 PGC: a secure and efficient instantiation

A secure and efficient instantiation of the above DCP construction turns out to be more technical involved. Before proceeding, it is instructive to list the desirable features in mind:

1. transparent setup – do not require a trusted setup. This property is of utmost importance in the setting of cryptocurrencies.
2. efficient – only employ lightweight cryptographic schemes based on well-studied assumptions.
3. modular – build the whole scheme from reusable gadgets.

The above desirable features suggest us to devise efficient NIZK that admits transparent setup, rather than resorting to general-purpose zk-SNARKs, which are heavyweight or require trusted setup. Besides, the encryption component of ISE should be zero-knowledge proofs friendly.

We begin with the instantiation of ISE. A common choice is to select Schnorr signature [Sch91] as the signature component and exponential ElGamal PKE [CGS97] as the encryption component.[1] This choice brings us at least three benefits: (i) Schnorr signature and ElGamal PKE share the same discrete-logarithm (DL) keypair (i.e., $pk = g^{sk} \in \mathbb{G}, sk \in \mathbb{Z}_p$), thus we can simply use the common public key as account address. (ii) The signing operation of Schnorr's signature is largely independent to the decryption operation of ElGamal PKE, which indicates that the resulting ISE scheme could be jointly secure. (iii) ElGamal PKE is additively homomorphic. Efficient Sigma protocols can thus be employed to prove linear relations on algebraically encoded-values. For instance, proving plaintext equality: C_s and C_r encrypt the same message under pk_s and pk_r respectively.

It remains to design efficient NIZK proofs for the basic correctness policies and various application-dependent policies.

Obstacle of working with Bulletproof. Besides using Sigma protocols to prove linear relations over encrypted values, we also need range proofs to prove the encrypted values lie in the right interval. In more details, we need to prove that the value v encrypted in C_s and the value encrypted in $\tilde{C}_s - C_s$ (the current balance subtracts v) lies in the right range. State-of-the-art range proof is Bulletproof [BBB+18], which enjoys efficient proof generation/verification, logarithmic proof size, and transparent setup. As per the desirable features of our instantiation, Bulletproof is an obvious choice. Recall that Bulletproof only accepts statements of the form of Pedersen commitment $g^r h^v$. To guarantee soundness, the DL relation between commitment key (g, h) must be unknown to the prover. Note that an ElGamal ciphertext C of v under pk is of the form $(g^r, pk^r g^v)$, in which the second part ciphertext can be viewed as a commitment of v under commitment key (pk, g). To prove v lies in the right range, it seems that we can simply run the Bulletproof on $pk^r g^v$. However, this usage is insecure since the prover owns an obvious trapdoor, say sk, of such commitment key.

There are two approaches to circumvent this obstacle. The first approach is to commit v with randomness r under commitment key (g, h), then prove (v, r) in $g^v h^r$ is consistent with that in the ciphertext $(g^r, pk^r g^v)$. Similar idea was used in Quisquis [FMMO19]. The drawback of this approach is that it brings extra overhead of proof size as well as proof generation/verification. The second approach

[1] In the remainder of this paper, we simply refer to the exponential ElGamal PKE as ElGamal PKE for ease of exposition.

is due to Bünz et al. [BAZB20] used in Zether. They extend Bulletproof to Σ-Bullets, which enables the interoperability between Sigma protocols and Bulletproof. Though Σ-Bullets is flexible, it requires custom design and analysis from scratch for each new Sigma protocols.

Twisted ElGamal - our customized solution. We are motivated to directly use Bulletproof in a black-box manner, without introducing Pedersen commitment as a bridge or dissecting Bulletproof. Our idea is to twist standard ElGamal, yielding the twisted ElGamal. We sketch twisted ElGamal below. The setup algorithm picks two random generators (g, h) of $\mathbb{G}$ as global parameters, while the key generation algorithm is same as that of standard ElGamal. To encrypt a message $m \in \mathbb{Z}_p$ under pk, the encryption algorithm picks $r \xleftarrow{\text{R}} \mathbb{Z}_p$, then computes ciphertext as $C = (X = pk^r, Y = g^r h^m)$. The crucial difference to standard ElGamal is that the roles of key encapsulation and session key are switched and the message m is lifted on a new generator h.[2] Clearly, twisted ElGamal retains additive homomorphism, and is as efficient and secure as the standard exponential ElGamal (as we will rigorously prove later). More importantly, it is zero-knowledge friendly. Note that the second part of twisted ElGamal ciphertext (even encrypted under different public keys) can be viewed as Pedersen commitment under the same commitment key (g, h), whose DL relation is unknown to all users. Such structure makes twisted ElGamal compatible with all zero-knowledge proofs whose statement is of the form Pedersen commitment. In particular, one can directly employ Bulletproof to generate range proofs for values encrypted by twisted ElGamal in a black-box manner, and these proofs can be easily aggregated. Next, we abstract two distinguished cases.

Prover knows the randomness. In our generic DCP construction, to create a transaction, sender encrypts transfer amount under his public key, then proves the encrypted amount lies in the right range. This case generalizes scenarios in which the prover is the producer of ciphertexts. Concretely, consider a twisted ElGamal ciphertext $C = (X = pk^r, Y = g^r h^v)$, to prove m lies in the right range, the prover first executes a Sigma protocol on C to prove the knowledge of (r, v), then invokes a Bulletproof on Y to prove v lies in the right range. The proof of knowledge property follows since the output of Sigma protocol's extractor equals that of Bulletproof's extractor with overwhelming probability based on the hardness of DLP related to (g, h). See Section 5.2.2 for the details.

Prover knows the secret key. In our generic DCP construction, sender also needs to prove the amount encrypted by $\tilde{C}_s - C_s$ lies in the right range. Here, $\tilde{C}_s$ is the encryption of his current balance, which is the accumulation of all previous incoming and outgoing transfers. Note that the randomness beneath $\tilde{C}_s - C_s$ is generally unknown to the sender, even assuming homomorphism on randomness.[3] This case generalizes scenarios where the prover is the recipient of ciphertexts. Due to lack of randomness as witness, the prover cannot directly invoke Bulletproof as above. We solve this problem by developing *ciphertext refreshing* approach. The prover first decrypts $\tilde{C}_s - C_s$ to v using sk, then generates a new ciphertext C_s^* of v under fresh randomness r^*, and proves that $\tilde{C}_s - C_s$ and C_s^* do encrypt the same message under his public key. As we will see later, this can be efficiently done by a Sigma protocol using sk as witness. Now, the prover is able to prove that v encrypted by C_s^* lies in the right range by composing a Sigma protocol and a Bulletproof, via the same way as the first case. See Section 5.2.3 for the details.

The above range proofs provide two specialized "proof gadgets" for proving encrypted values lie in the right range. We refer to them as Gadget-1 and Gadget-2 hereafter. As we will see, all the NIZK proofs used in this work can be built from these two gadgets and simple Sigma protocols. Such modular design helps to reduce the footprint of overall cryptographic code, and have the potential to admit parallel proof generation/verification. We highlight the two "gadgets" are interesting on their own right as privacy-preserving tools, which may find applications in other domains as well, e.g., secure machine learning [TMZC17].

Additional results. Along the way of designing the zero-knowledge proofs for PGC, we obtain two interesting results. The first one is weak forking lemma, which is a relaxed version of standard forking lemma. Weak forking lemma provides a useful tool of rigorously proving argument of knowledge based on average-case hard assumption over public parameters, for example, Bulletproof and inner product

[2] As indicated in [CZ14], the essence of ElGamal is $F_{sk}(g^r) = pk^r$ forms a publicly evaluable pseudorandom functions over $\mathbb{G}$. The key insight of switching is that F_{sk} is in fact a permutation.

[3] Most encryption schemes are not randomness recovering.

argument. See Theorem 2.3 for details. The second one is a simple trick of proving no-knowledge of discrete logarithm, which is an interesting complement of the classical protocol of proving knowledge of discrete logarithm. See the discussion in Section 6.3 for details.

Enforcing more auditing policies. In the above, we have discussed how to employ Sigma protocols and Bulletproof to enforce the basic correctness policy for transactions. In fact, we are able to enforce a variety of polices that can be expressed as linear constraints over transfer amounts.

- Limit policy: let $\mathsf{ctx}_1, \dots, \mathsf{ctx}_n$ be a collection of confidential transaction sent from/to a particular account pk, and let C_i be the corresponding encryptions of transfer amounts v_i under pk. This policy stipulates that $\sum_{i=1}^n v_i \leq a_{\max}$, where $a_{\max}$ is an upper bound depending on application. The prover could be either the sender or the receiver of ctx_i, who are not required to keep track of history randomness. To enforce this policy, the prover utilizes additive homomorphism to compute to compute $C = \sum_{i=1}^n C_i$, then uses Gadget-2 to generate a proof for compliance of this policy. The auditor can enforce limit policy anti-laundering money.

- Rate policy: let ctx_1 be an incoming transaction sent to pk, ctx_2 be an transaction sent from pk, in which C_1 and C_2 are the corresponding encryptions of transfer amounts v_1 and v_2 under pk. This policy stipulates $v_1/v_2 = \rho$, where ρ is an application-dependent rate. Without much loss of generality, we assume $\rho = \alpha/\beta$, where α and β are two integers. To demonstrate this policy, the prover utilizes additive homomorphism to compute $C_1' = \beta \cdot C_1$ and $C_2' = \alpha \cdot C_2$, then prove C_1 and C_2 encrypt the same value under pk. The auditor can enforce rate policy to ensure taxpayer paid tax according to the rules.

- Open policy: let ctx be a confidential transaction from pk_s to pk_r, and C_u be the encryption of transfer amount under pk_u, where the subscript u could be either s or r. This policy stipulates that the underlying transfer amount is indeed v, where v is an application-dependent value. Either the sender or receiver of ctx prove this via a classic Sigma protocol for discrete logarithm equality, using his secret key sk as witness. This policy can be used to enforce selective-disclosure.

1.4 Related Work

The seminal cryptocurrencies such as Bitcoin and Ethereum do not provide sufficient level of privacy. In the past years privacy-enhancements have been developed along several lines of research. We provide a brief overview below.

The first direction aims to provide confidentiality. Maxwell [Max] initiates the study of *confidential transaction*. He proposes a DCP system by employing Pedersen commitment to hide transfer amount and using range proofs to prove correctness of transaction. Mimblewimble/Grin [Poe, Gri] further improve Maxwell's construction by reducing the cost of signatures. The second direction aims to enhance anonymity. A large body of works enhance anonymity via utilizing mixing mechanisms. For instance, Coinjoin [Max13], CoinShuffle [RMK14], TumbleBit [HAB$^+$17], Dash [Das], and Mixcoin [BNM$^+$14] for Bitcoin and Möbius [MM] for Ethereum. The third direction aims to attain both confidentiality and anonymity. Monero [Noe15] achieves confidentiality via similar techniques used in Maxwell's DCP system, and provides anonymity by employing linkable ring signature and stealth address. Zcash [ZCa] achieves strong privacy by equipping two types of addresses and leveraging key-private PKE and zk-SNARK.

Despite great progress in privacy-enhancement, the aforementioned schemes are not without their limitations. In terms of reliability, most of them require out-of-band transfer and thus are not complete. In terms of efficiency, some of them suffer from slow transaction generation due to the use of heavy advanced cryptographic tools. In terms of security, some of them are not proven secure based on well-studied assumption, or rely on trusted setup.

Another related work is zkLedger [NVV18], which offers strong privacy and privacy-preserving auditing. To attain anonymity, zkLedger uses a novel table-based ledger. Consequently, the size of transactions and audit efficiency are linear in the total number of participants in the system. This makes zkLedger only suitable for a small scale of participants (e.g. consortium of banks). While partial content in zkLedger transaction (audit token plus Pedersen commitment) is similar to our twisted ElGamal, its payment system is still commitment-based. In other words, the transfer amount is assumed to be transmitted through an out-of-band channel. The audit token provided by the sender is only used for the receiver to open the Pedersen commitment. In contrast, our twisted ElGamal is proposed as a homomorphic PKE scheme by tweaking the standard exponential ElGamal, with the design goal of building ISE and seamlessly working with Bulletproof. Particularly, efficient decryption algorithm is considered and implemented by design. PGC is thus encryption-based, and out-of-band transmission is not needed. As a bonus of dropping anonymity, the overall design is simple and allows fine-grained audit.

Concurrent and independent work. Fauzi et al. [FMMO19] put forward a new design of cryptocurrency with strong privacy called Quisquis in the UTXO model. They employ updatable public keys to achieve anonymity and a slight variant of standard ElGamal encryption to achieve confidentiality. Bünz et al. [BAZB20] propose a confidential payment system called Zether, which is compatible with Ethereum-like smart contract platforms. They use standard ElGamal to hide the balance and transfer amount, and use signature derived from NIZK to authenticate transactions. They also sketch how to acquire anonymity for Zether via a similar approach as zkLedger [NVV18]. Both Quisquis and Zether design accompanying zero-knowledge proofs from Sigma protocols and Bulletproof, but take different approaches to tackle the incompatibility between ElGamal encryption and Bulletproof. Quisquis introduces ElGamal commitment to bridge ElGamal encryption, and uses Sigma protocol to prove consistency of bridging. Finally, Quisquis invokes Bulletproof on the second part of ElGamal commitment, which is exactly a Pedersen commitment. Zether develops a custom ZKP called Σ-Bullets, which is a dedicated integration of Bulletproof and Sigma protocol. Given an arithmetic circuit, a Σ-Bullets ensures that a public linear combination of the circuit's wires is equal to some witness of a Sigma protocol. This enhancement in turn enables proofs on algebraically-encoded values such as ElGamal encryptions or Pedersen commitments in different groups or using different commitment keys.

We highlight the following crucial differences of our work to Quisquis and Zether: (i) We focus on confidentiality, and trade anonymity for auditability. (ii) We use jointly secure ISE, rather than ad-hoc combination of signature and encryption, to build DCP system in a provably secure way. (iii) As for instantiation, PGC employs our newly introduced twisted ElGamal rather than standard ElGamal to hide balance and transfer amount. The nice structure of twisted ElGamal enables the sender to prove the transfer amount lies in the right range by directly invoking Bulletproof in a black-box manner, without any extra bridging cost as Quisquis. The final proof for correctness of transactions in PGC is obtained by assembling small "proof gadgets" together in a simple and modular fashion, which is flexible and reusable. This is opposed to Zether's approach, in which zero-knowledge proof is produced by a Σ-Bullets as a whole, while building case-tailored Σ-Bullets requires to dissect Bulletproof and skillfully design its interface to Sigma protocol.

2 Preliminaries

Basic Notations. Throughout the paper, we denote the security parameter by $\lambda \in \mathbb{N}$. A function is negligible in λ, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ. Let $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ be two distribution ensembles indexed by λ. We say that X and Y are statistically indistinguishable, written $X \approx_s Y$, if the statistical distance between X_λ and Y_λ is negligible in λ. We say that X and Y are computationally indistinguishable, written $X \approx_c Y$, if the advantage of any PPT algorithm in distinguishing X_λ and Y_λ is $\mathsf{negl}(\lambda)$. A probabilistic polynomial time (PPT) algorithm is a randomized algorithm that runs in time $\mathsf{poly}(\lambda)$. If $\mathcal{A}$ is a randomized algorithm, we write $z \leftarrow \mathcal{A}(x_1, \ldots, x_n; r)$ to denote that $\mathcal{A}$ outputs z on inputs $(x_1, \ldots, x_n)$ and randomness r. For notational clarity we usually omit r and write $z \leftarrow \mathcal{A}(x_1, \ldots, x_n)$. For a positive integer n, we use $[n]$ to denote the set of numbers $\{1, \ldots, n\}$. For a set X, we use $|X|$ to denote its size and use $x \xleftarrow{\text{R}} X$ to denote sampling x uniformly at random from X. We use U_X to denote the uniform distribution over X.

Below, we review the cryptographic assumptions and primitives that will be used in this work.

2.1 Cryptographic Assumptions

Let $\mathsf{GroupGen}$ be a PPT algorithm that on input a security parameter 1^λ, outputs description of a cyclic group $\mathbb{G}$ of prime order $p = \Theta(2^\lambda)$, and a random generator g for $\mathbb{G}$. In what follows, we describe the discrete-logarithm based assumptions related to $(\mathbb{G}, g, p) \leftarrow \mathsf{GroupGen}(1^\lambda)$.

Definition 2.1 (Discrete Logarithm Assumption)**.** The discrete logarithm assumption holds if for any PPT adversary, we have:

$$\Pr[\mathcal{A}(g, h) = a \text{ s.t. } g^a = h] \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of $\mathsf{GroupGen}(1^\lambda)$, $\mathcal{A}$'s random tape, and the random choice of $h \xleftarrow{\text{R}} \mathbb{G}$.

Definition 2.2 (Decisional Diffie-Hellman Assumption)**.** The DDH assumption holds if for any PPT adversary, we have:

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of $\mathsf{GroupGen}(1^\lambda)$, $\mathcal{A}$'s random tape, and the random choices of $a, b, c \xleftarrow{\text{R}} \mathbb{Z}_p$.

Definition 2.3 (Divisible Decisional Diffie-Hellman Assumption)**.** The divisible DDH assumption holds if for any PPT adversary, we have:

$$\left|\Pr[\mathcal{A}(g, g^a, g^b, g^{a/b}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the randomness of $\mathsf{GroupGen}(1^\lambda)$, $\mathcal{A}$'s random tape, and the random choices of $a, b, c \xleftarrow{\text{R}} \mathbb{Z}_p$.

As proved in [BDZ03], the divisible DDH assumption is equivalent to the standard DDH assumption.

2.2 Commitments

A non-interactive commitment scheme is a two-party protocol between a sender and a receiver with two stages. At the committing stage, the sender commits to some value m by sending a commitment to the receiver. At the opening stage, the sender can open the commitment by providing m and some auxiliary information, by which the receiver can verify that the value it received is indeed the value committed by the sender during the committing stage. Formally, a commitment scheme consists of three polynomial time algorithms as below:

- $\mathsf{Setup}(1^\lambda)$: on input security parameter 1^λ, output public parameters pp. We assume that pp includes the descriptions of message space M, randomness space R, and commitment space C. pp will be used as implicit input of the following two algorithms.
- $\mathsf{Com}(m; r)$: the sender commits a message m by choosing uniform random coins r, and computing $c \leftarrow \mathsf{Com}(m; r)$, then sends c to receiver.
- $\mathsf{Open}(c, m, r)$: the sender can later decommit c by sending m, r to the receiver; the receiver outputs $\mathsf{Com}(m; r) \stackrel{?}{=} c$.

Correctness. For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, all $m \in M$ and all $r \in R$, we have $\mathsf{Open}(\mathsf{Com}(m; r), m, r) = 1$. For security, we require hiding and binding.

Hiding. A commitment $\mathsf{Com}(m; r)$ should not reveal anything about its committed value of m. Let $\mathcal{A}$ be an adversary against hiding, we define its advantage via the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta' = \beta : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (m_0, m_1) \leftarrow \mathcal{A}_1(pp); \\ \beta \xleftarrow{\text{R}} \{0, 1\}, r \xleftarrow{\text{R}} R, c \leftarrow \mathsf{Com}(m_\beta; r); \\ \beta' \leftarrow \mathcal{A}_2(c); \end{array}\right] - \frac{1}{2}.$$

A commitment scheme is perfectly hiding if $\mathsf{Adv}_{\mathcal{A}}(\lambda) = 0$ even for unbounded adversary, is statistical hiding (resp. computational hiding) if $\mathsf{Adv}_{\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ w.r.t. unbounded adversary (resp. PPT adversary).

Binding. A commitment c cannot be opened to two different messages. Let $\mathcal{A}$ be an adversary against binding, we define its advantage via the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} m_0 \neq m_1 \wedge \\ c = \mathsf{Com}(m_0; r_0) = \mathsf{Com}(m_1; r_1) \end{array} : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (c, m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(pp); \end{array}\right].$$

A commitment scheme is perfectly binding if $\mathsf{Adv}_{\mathcal{A}}(\lambda) = 0$ even for unbounded adversary (a.k.a. $\forall m_0 \neq m_1$, their commitment values are disjoint), is statistical binding (resp. computational binding) if $\mathsf{Adv}_{\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ w.r.t. unbounded adversary (resp. PPT adversary).

Pedersen Commitment. Below we recall the Pedersen commitment [Ped91]:

- $\mathsf{Setup}(1^\lambda)$: run $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$, pick $h \xleftarrow{R} \mathbb{G}^*$, then output $pp = (\mathbb{G}, p, g, h)$. Here, $M = R = \mathbb{Z}_p$, $C = \mathbb{G}$.
- $\mathsf{Com}(m; r)$: on input message $m \in \mathbb{Z}_p$ and randomness $r \xleftarrow{R} \mathbb{Z}_p$, output $c \leftarrow g^m h^r$.
- $\mathsf{Open}(c, m, r)$: output "1" if $c = g^m h^r$ and "0" otherwise.

The Pedersen commitment is perfectly hiding and computational binding under the discrete logarithm assumption.

2.3 Public-Key Encryption

A PKE scheme consists of four polynomial time algorithms as follows.

- $\mathsf{Setup}(1^\lambda)$: on input a security parameter 1^λ, output public parameters pp. We assume pp also includes the descriptions of message space M, ciphertext space C, and randomness space R.
- $\mathsf{KeyGen}(pp)$: on input pp, output a public key pk and a secret key sk.
- $\mathsf{Enc}(pk, m)$: on input a public key pk and a plaintext m, output a ciphertext c. When emphasizing the randomness r used for encryption, we denote this by $c \leftarrow \mathsf{Enc}(pk, m; r)$.
- $\mathsf{Dec}(sk, c)$: on input a secret key sk and a ciphertext c, output a plaintext m or a distinguished symbol $\perp$ indicating that c is invalid.

Correctness. For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, all $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$ and all $m \in M$ (here M is the message space), we have $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$.

Homomorphism. For any public key pk, any $(m_1, r_1), (m_2, r_2) \in M \times R$, we have $\mathsf{Enc}(pk, m_1; r_1) + \mathsf{Enc}(pk, m_2; r_2) = \mathsf{Enc}(pk, m_1 + m_2; r_1 + r_2)$. Here, we slight abuse the symbol "+" to denote componentwise operation over space $C \times C$. Additive homomorphism on message already suffices for most applications: $\mathsf{Enc}(pk, m_1) + \mathsf{Enc}(pk, m_2)$ is an encryption of $(m_1 + m_2)$ of pk under some appropriate randomness. When both ciphertext space and message space are finite cyclic groups, we can naturally define scalar multiplication, which is a special case of additive homomorphism: for all $k \in \mathbb{Z}_p$, $k \cdot \mathsf{Enc}(pk, m)$ is an encryption of km under pk.

2.4 Signature

A signature scheme consists of four polynomial time algorithms as follows.

- $\mathsf{Setup}(1^\lambda)$: on input a security parameter 1^λ, output public parameters pp. We assume pp also includes the descriptions of message space M, signature space Σ, and randomness space R.
- $\mathsf{KeyGen}(pp)$: on input pp, output a public key vk and a secret key sk.
- $\mathsf{Sign}(sk, m)$: on input sk and a message m, output a signature σ.
- $\mathsf{Verify}(pk, m, \sigma)$: on input pk, a message m, and a purported signature σ, output "1" indicating acceptance or "0" indicating rejection.

Correctness. For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, all $(vk, sk) \leftarrow \mathsf{KeyGen}(pp)$ and all $m \in M$ (here M is the message space), it holds that $\mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

2.5 Integrated Signature and Encryption Scheme

Haber and Pinkas [HP01] introduced the concept of combined public key scheme, which is a combination of a signature scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ and encryption scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$: the existing Sign, Verify, Enc, Dec algorithms are preserved, while the two sets of Setup, KeyGen algorithms are combined into a single one. Paterson et al. [PSST11] revisited this topic and gave a generic construction of combined public key scheme from identity-based encryption.

In this work, we use a special case of combined public key scheme which we refer to as integrated signature and encryption scheme (ISE). ISE requires the Setup and KeyGen algorithms of the underlying signature and encryption components are identical or are sub-routines of one another. Under this setting, the two sets of Setup and KeyGen algorithms can be completely merged to produce one compact set of public parameters and keypair.

ISE uses the same keypair for both signing and encryption, thus the two operations may interfere with one another, in such a way to undermine the security of one or both of the components. For this reason, joint security must be considered when ISE is used, which captures potentially dangerous interactions. Please refer to [PSST11] for more detailed discussion on this point.

In the context of our DCP construction, the joint security of ISE stipulates that the PKE component is IND-CPA secure in the single-plaintext/two-recipient setting even in the presence of a signing oracle, while the signature component is EUF-CMA secure even in the presence of two encryption oracles. Note that in the public key setting the adversary can always perfectly simulate encryption oracles itself, thus standard EUF-CMA security implies that the signature component is secure in the joint sense and we only need to enhance the IND-CPA security for the PKE component. Let $\mathrm{ISE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Enc}, \mathsf{Dec})$. We formally define the case-tailored joint security model used in this work.

Definition 2.4 (Joint Security for ISE)**.** We say an ISE is jointly secure (case-tailored for DCP setting) if its encryption and signature components satisfy the following security notions respectively.

IND-CPA security (1-plaintext/2-recipient) in the presence of a signing oracle. Let $\mathcal{A}$ be an adversary against the PKE component and define its advantage in the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (pk_i, sk_i) \leftarrow \mathsf{KeyGen}(pp) \text{ for } i = 1, 2; \\ (state, m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sign}}}(pp, pk_1, pk_2); \\ \beta \xleftarrow{\text{R}} \{0, 1\}; \\ C_i \leftarrow \mathsf{Enc}(pk_i, m_\beta) \text{ for } i = 1, 2; \\ \beta' \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{sign}}}(state, C_1, C_2); \end{array}\right] - \frac{1}{2}.$$

Here, $\mathcal{O}_{\mathsf{sign}}$ provides unlimited access to signing oracle with respect to sk_1 and sk_2. More precisely, $\mathcal{O}_{\mathsf{sign}}$ returns $\mathsf{Sign}(sk_i, m)$ on input $i \in \{1, 2\}$ and $m \in M$. The encryption component is IND-CPA secure in 1-plaintext/2-recipient setting if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment.

EUF-CMA security. The security requirement for the signature component is exactly the standard EUF-CMA security. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against the signature component and define its advantage in the following experiment:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \mathsf{Verify}(pk, m^*, \sigma^*) = 1 \\ \wedge\ m^* \notin \mathcal{Q} \end{array} : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (pk, sk) \leftarrow \mathsf{KeyGen}(pp); \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{sign}}}(pp, pk); \end{array}\right].$$

The set $\mathcal{Q}$ records queries to $\mathcal{O}_{\mathsf{sign}}$, which returns $\mathsf{Sign}(sk, m)$ on input m. The signature component is EUF-CMA if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment.

2.6 Zero-Knowledge Protocols

We first recall the definition of interactive proof systems.

Definition 2.5 (Interactive Proof System)**.** An interactive proof system is a two party protocol in which a prover can convince a verifier that some statement is true without revealing any knowledge about why it holds. Formally, it consists of three PPT algorithms (Setup, P, V) as below.

- The Setup algorithm on input 1^λ, outputs public parameters pp. Let $\mathsf{R}_{pp} \subseteq X \times W$ be an $\mathcal{NP}$ relation indexed by pp. We say $w \in W$ is a witness for a statement x iff $(x, w) \in \mathsf{R}_{pp}$. R_{pp} naturally defines a family of public-parameters-dependent $\mathcal{NP}$ languages:

  $$L_{pp} = \{x \mid \exists w \in W \text{ s.t. } (x, w) \in \mathsf{R}_{pp}\}$$

  From now on, we will drop the subscript pp occasionally when the context is clear.

- P and V are a pair of interactive algorithms, which both take pp as implicit input and the statement x as common input. We use the notation $tr \leftarrow \langle P(x), V(y)\rangle$ to denote the transcript of an execution between P and V, where P has input x and V has input y. We write $\langle P(x), V(y)\rangle = b$ depending on whether V accepts, $b = 1$, or rejects, $b = 0$. When the context is clear, we will also slightly abuse the notation of $\langle P(x), V(y)\rangle$ to denote V's view (View_V) in the interaction, which consists of V's input tape, random tape and incoming messages sent by P.

An interactive proof system satisfies the following two properties:

Completeness. For any $(x, w) \in \mathsf{R}_{pp}$ where $pp \leftarrow \mathsf{Setup}(1^\lambda)$, we have:

$$\Pr[\langle P(x, w), V(x)\rangle = 1] = 1$$

Statistical soundness. For any $x \notin L_{pp}$ where $pp \leftarrow \mathsf{Setup}(1^\lambda)$, any cheating prover P^*, we have:

$$\Pr[\langle P^*(x), V(x)\rangle = 1] \leq \mathsf{negl}(\lambda)$$

If we restrict P^* to be a PPT cheating prover in the above definition, we obtain an interactive argument system.

Public coin. An interactive proof/argument system is public-coin if all messages sent from V are chosen uniformly at random and independent of P's message.

We then recall several zero-knowledge extensions of interactive zero knowledge proof systems that will be used in this work.

Definition 2.6 (Zero-Knowledge Argument of Knowledge)**.** We say an interactive proof system is a zero-knowledge argument of knowledge if it satisfies standard completeness, argument of knowledge and zero knowledge.

Following [BCC+16], we use computational witness-extended emulation to define argument of knowledge. Intuitively, this definition says that whenever a malicious prover produces an accepting argument with some probability, there exists an emulator producing a similar argument with roughly the same probability together with a witness.

Computational witness-extended emulation. For all deterministic polynomial time P^* there exists an expected PPT emulator E such that for all PPT interactive adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function $\mu(\lambda)$ such that:

$$\Pr\left[\begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (x, s) \leftarrow \mathcal{A}_1(pp); \\ tr \leftarrow \langle P^*(x, s), V(x)\rangle; \\ \mathcal{A}_2(tr) = 1 \end{array}\right] - \Pr\left[\begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (x, s) \leftarrow \mathcal{A}_1(pp); \\ (tr, w) \leftarrow E^{\mathcal{O}}(x); \\ \mathcal{A}_2(tr) = 1 \wedge \\ tr \text{ is accepting} \Rightarrow (x, w) \in \mathsf{R}_{pp} \end{array}\right] \leq \mu(\lambda)$$

where the oracle $\mathcal{O} = \langle P^*(x, s), V(x)\rangle$ permits rewinding to a specific point and resuming with fresh randomness for the verifier from this point onwards. In the definition, s is interpreted as the state of P, including the randomness and auxiliary input. According to the definition, whenever P^* makes a convincing argument in state s, E can extract a witness. This is why it defines argument of knowledge.

Computational zero-knowledge. For any malicious PPT V^* there exists an expected PPT simulator $\mathcal{S}$ such that for any $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $(x, w) \in \mathsf{R}_{pp}$, we have:

$$\langle P(x, w), V^*(x)\rangle \approx_c \mathcal{S}(x)$$

Computational zero-knowledge can be strengthened to statistical (resp. perfect) zero-knowledge by requiring the views are statistically indistinguishable (resp. identical).

Definition 2.7 (Sigma Protocol (Σ-protocol) [Dam]). An interactive proof system is called a Sigma protocol if it follows the following communication pattern (3-round public-coin):

1. (Commit) P sends a first message a to V;
2. (Challenge) V sends a random challenge e to P;
3. (Response) P replies with a second message z.

and satisfies standard completeness and the variants of soundness and zero-knowledge as below:

Special soundness. For any x and any pair of accepting transcripts (a, e, z), (a, e', z') with $e \neq e'$, there exists a PPT extractor outputs a witness w for x.

Special honest-verifier zero-knowledge (SHVZK). There exists a PPT simulator $\mathcal{S}$ such that for any $(x, w) \in \mathsf{R}_{pp}$ and randomness e, we have:

$$\langle P(x, w), V(x, e)\rangle \equiv \mathcal{S}(x, e)$$

Definition 2.8 (Range Proof). For a commitment scheme over total ordering message space M and randomness space R, a range proof is a zero-knowledge argument of knowledge for the following language:

$$L = \{c \mid \exists m \in M, r \in R \text{ s.t. } c = \mathsf{Com}(m; r) \wedge x \in [a, b]\}$$

Interaction is typically expensive or impossible in some cases. Thus, removing interaction is of particular interest in practice. However, non-interactive zero-knowledge (NIZK) proofs for non-trivial languages are impossible in the plain model. We need to consider NIZK in the common reference string (CRS) model, wherein a string is generated after setup phase, and made available to everyone to prove/verify statement.

Definition 2.9 (Non-Interactive Zero-Knowledge Proof [BFM88, FLS90]). A NIZK proof system in the CRS model consists of four PPT algorithms $(\mathsf{Setup}, \mathsf{CRSGen}, P, V)$:

- $\mathsf{Setup}(1^\lambda)$: same as that of ordinary zero-knowledge proof system, which on input 1^λ, output public parameters pp.
- $\mathsf{CRSGen}(pp)$: on input pp, output a common reference string crs.
- $P(crs, x, w)$: on input crs and a statement-witness pair $(x, w) \in \mathsf{R}_{pp}$, output a proof π. We also denote this process by $\pi \leftarrow \mathsf{Prove}(crs, x, w)$.
- $V(crs, x, \pi)$: on input crs, a statement x, and a proof π, outputs "0" if rejects and "1" if accepts. We also denote this process by $b \leftarrow \mathsf{Verify}(crs, x, \pi)$.

A NIZK proof system in the CRS model satisfies the following requirements:

Completeness. For any $(x, w) \in \mathsf{R}_{pp}$ where $pp \leftarrow \mathsf{Setup}(1^\lambda)$,

$$\Pr\left[V(crs, x, \pi) = 1 : \begin{array}{c} crs \leftarrow \mathsf{CRSGen}(pp); \\ \pi \leftarrow P(crs, x, w); \end{array}\right] = 1.$$

Statistical soundness. For any cheating prover P^*,

$$\Pr = \left[\begin{array}{c} x \notin L \wedge \\ V(crs, x, \pi) = 1 \end{array} : \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ crs \leftarrow \mathsf{CRSGen}(pp); \\ (x, \pi) \leftarrow P^*(crs); \end{array}\right] \leq \mathsf{negl}(\lambda).$$

This definition is in the adaptive sense in that P^* may choose x after seeing crs. Statistical soundness can be relaxed to computational soundness by restricting P^* to be a PPT algorithm. In this case, a proof system degenerates to an argument system.

Computational zero-knowledge. For any $pp \leftarrow \mathsf{Setup}(1^\lambda)$, any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a PPT simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that:

$$\Pr\left[\begin{array}{c} crs \leftarrow \mathsf{CRSGen}(pp); \\ (x, w) \leftarrow \mathcal{A}_1(crs); \\ \pi \leftarrow P(crs, x, w); \\ \mathcal{A}_2(crs, x, \pi) = 1 \end{array}\right] - \Pr\left[\begin{array}{c} (crs, \tau) \leftarrow \mathcal{S}_1(pp); \\ (x, w) \leftarrow \mathcal{A}_1(crs); \\ \pi \leftarrow \mathcal{S}_2(crs, x, \tau); \\ \mathcal{A}_2(crs, x, \pi) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda).$$

This definition is also in the adaptive sense in that $\mathcal{A}_1$ may adaptively choose x after seeing crs. Computational zero-knowledge can be strengthened to statistical zero-knowledge by requiring the above holds even for unbounded $\mathcal{A}$.

Interestingly, in the random oracle model NIZK is possible without explicitly relying on common reference string. The celebrated Fiat-Shamir transform [FS86] shows how to compile a Sigma protocol into a NIZK by modeling cryptographic hash function as random oracle. It not only removes interaction, but also strengthens honest-verifier zero-knowledge to full zero-knowledge (against malicious verifiers). Fiat-Shamir transform actually applies to any public-coin SHVZK argument of knowledge. Formally, we have the following theorem.

Theorem 2.1 (Fiat-Shamir Transform [BR93, FKMV12])**.** *Let* (Setup, P, V) *be a* $(2k+1)$*-move public-coin SHVZK argument of knowledge,* x *be the statement,* a_i *be prover's ith round message and* e_i *be verifier's ith round challenge,* H *is a hash function whose range equal to verifier's challenge space. By setting* $e_i = \mathsf{H}(a_1, \ldots, a_i)$ *in* (Setup, P, V)*, we obtain* $(\mathsf{Setup}, P^{\mathsf{H}}, V^{\mathsf{H}})$*, which is a NIZK assuming* H *is a random oracle.*[4]

General Forking Lemma. We recall the general forking lemma of [BCC$^+$16, BBB$^+$18] that will be used in our proofs. Suppose that we have a $(2k+1)$-move public-coin argument with k challenges, $e_1, \ldots, e_k$ in sequence. Let $n_i \geq 1$ for $i \in [k]$. Consider $\Pi_{i=1}^k n_i$ accepting transcripts whose challenges satisfying the following tree format. The tree has depth k and $\Pi_{i=1}^k n_i$ leaves. The root of the tree is labeled with the statement. Each node of depth $i < k$ has exactly n_i children, each labeled with a distinct value of the ith challenge e_i. This can be referred to as an $(n_1, \ldots, n_k)$-tree of accepting transcripts, which is a natural generalization of special soundness for Sigma protocols where $k = 1$ and $n = 2$. For the simplicity in the following lemma, we assume that the challenge space is $\mathbb{Z}_p$ and $p = \Theta(2^\lambda)$.

Theorem 2.2 (Forking Lemma [BCC$^+$16, BBB$^+$18])**.** *Let* (Setup, P, V) *be a* $(2k+1)$*-move, public-coin interactive protocol. Let* E *be a PPT witness extraction algorithm that succeeds with probability* $1 - \mu(\lambda)$ *for some negligible function* $\mu(\lambda)$ *in extracting a witness from an* $(n_1, \ldots, n_k)$*-tree of accepting transcripts. If* $\Pi_{i=1}^k n_i$ *is bounded by a polynomial in* λ*, then* (Setup, P, V) *has witness-extended emulation.*

We note that the condition of standard forking lemma is required to hold independent of the distributions of public parameter and instance. It is a useful tool to prove witness-extended emulation, a.k.a. proof of knowledge. However, such a strong form of condition is impossible to achieve in the computational setting when we need to argue the output of extractor satisfying specific relation based on average-case assumptions over public parameters. For example, the cases of inner product argument and Bulletproof [BBB$^+$18]. We are thus motivated to introduce a weaker form of forking lemma by relaxing the condition, which still suffices to imply computational witness-extended emulation. In what follows, we formulate weak forking lemma by capturing the relaxed condition via an interactive game.

Theorem 2.3 (Weak Forking Lemma)**.** *Let* (Setup, P, V) *be a* $(2k+1)$*-move, public-coin interactive protocol. Let* E *be a PPT witness extraction algorithm that succeeds with probability* $1 - \mu(\lambda)$ *for some negligible function* $\mu(\lambda)$ *in extracting a witness from an* $(n_1, \ldots, n_k)$*-tree of accepting transcripts in the following game.*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda); \\ (x, s) \leftarrow \mathcal{A}(crs); \\ w \leftarrow E(crs, x, tr); \\ tr \text{ is accepting} \Rightarrow (x, w) \in \mathsf{R}_{pp} \end{array}\right] \geq 1 - \mu(\lambda)$$

If such E *exists for any PPT adversary* $\mathcal{A}$*,* $\Pi_{i=1}^k n_i$ *is bounded by a polynomial in* λ*, then* (Setup, P, V) *has computational witness-extended emulation.*

[4] To get a unifying syntax of NIZK, one can also interpret the description of H as common reference string.

3 Definition of Decentralized Confidential Payment System

We formalize the notion of *decentralized confidential payment system* in account model, adapting the notion of *decentralized anonymous payment system* [ZCa].

3.1 Data Structures

We begin by describing the data structures used by a DCP system.

Blockchain. A DCP system operates on top of a blockchain B. The blockchain is publicly accessible, i.e., at any given time t, all users have access to B_t, the ledger at time t, which is a sequence of transactions. The blockchain is also append-only, i.e., $t < t'$ implies that B_t is a prefix of $B_{t'}$.

Public parameters. A trusted party generate public parameters pp at the setup time of system, which is used by system's algorithms. We assume that pp always include an integer $v_{\max}$, which specifies the maximum possible number of coins that the system can handle. Any balance and transfer below must lie in the integer interval $\mathcal{V} = [0, v_{\max}]$.

Account. Each account is associated with a keypair (pk, sk), an encoded balance $\tilde{C}$ (which encodes plaintext balance $\tilde{m}$), as well as an incremental serial number sn (used to prevent replay attacks). Both sn, $\tilde{C}$, and pk are made public. The public key pk serves as account address, which is used to receive transactions from other accounts. The secret key sk is kept privately, which is used to direct transactions to other accounts and decodes encoded balance.

Confidential transaction. A confidential transaction ctx consists of three parts, i.e., sn, memo and aux. Here, sn is the current serial number of sender account pk_s, $\mathsf{memo} = (pk_s, pk_r, C)$ records basic information of a transaction from sender account pk_s to receiver account pk_r, where C is the encoding of transfer amount, and aux denotes the auxiliary information, which is application-dependent.

Policies. To be applicable, transactions should comply with application-dependent policies. Observe that the value-encoding tuples naturally constitute polynomially decidable relations, policies over transfer amounts and balances can thus be expressed by $\mathcal{NP}$ relations. Looking ahead, this allow participants to prove compliance with specified policies by leveraging NIZK.

We formally capture polices as predicates over public key and related transactions. Let $\{\mathsf{ctx}_i\}_{i=0}^n$ be a set of confidential transactions related to pk, i.e., for each ctx_i, pk is either the sender or the receiver, and v_i be the transfer amount of ctx_i. A policy over $\{v_i\}_{i=1}^n$ is satisfied if and only if $f(pk, \{\mathsf{ctx}_i\}_{i=0}^n) = 1$. The basic correctness policy for a single transaction in any payment system requires the transfer amount lies in the correct range and the sender account is solvent, we denote the associated predicate as $f_{\text{correct}}(pk, \mathsf{ctx})$. We list more useful policies as below: (i) limit policy – $\sum_i^n v_i \leq a_{\max}$, the associated predicate is $f_{\text{limit}}(pk, \{\mathsf{ctx}_i\}_{i=1}^n)$; (i) rate policy – $v_1/v_2 = \rho$, the associated predicate is $f_{\text{rate}}(pk, (\mathsf{ctx}_1, \mathsf{ctx}_2))$; (i) open policy – $v = v^*$, the associated predicate is $f_{\text{open}}(pk, \mathsf{ctx})$;

3.2 Decentralized Confidential Payment System with Auditability

An auditable DCP system is a tuple of polynomial-time algorithms defined as below:

- $\mathsf{Setup}(\lambda)$: on input a security parameter λ, output public parameters pp. A trusted party executes this algorithm once-for-all to setup the whole system.
- $\mathsf{CreateAccount}(\tilde{m}, \mathsf{sn})$: on input an initial balance $\tilde{m}$ and a serial number sn, output a keypair (pk, sk) and an encoded balance $\tilde{C}$. A user runs this algorithm to create an account.
- $\mathsf{RevealBalance}(sk, \tilde{C})$: on input a secret key sk and an encoded balance $\tilde{C}$, output the balance $\tilde{m}$ in plaintext. A user runs this algorithm to reveal the balance.
- $\mathsf{CreateCTx}(sk_s, pk_s, pk_r, v)$: on input a keypair (sk_s, pk_s) of sender account, a receiver account address pk_r, and a transfer amount v, output a confidential transaction ctx. A user runs this algorithm to transfer v coins from account pk_s to account pk_r.
- $\mathsf{VerifyCTx}(\mathsf{ctx})$: on input a confidential transaction ctx, output "0" denotes valid and "1" denotes invalid. Miners run this algorithm to check the validity of proposed confidential transaction ctx. If ctx is valid, it will be recorded on the blockchain B. Otherwise, it is discarded.

- $\mathsf{UpdateCTx}(\mathsf{ctx})$: for each fresh ctx appearing on the blockchain B, the corresponding sender and receiver update their encoded balances to reflect the change, i.e., the sender account decreases with v coins while the receiver account increases with v coins.

- $\mathsf{JustifyCTx}(pk, sk, \{\mathsf{ctx}\}, f)$: on input a user's keypair (pk, sk), a set of confidential transactions he participated and a policy f, output a proof π for $f(pk, \{\mathsf{ctx}\}) = 1$. A user runs this algorithm to generate a proof for auditing.

- $\mathsf{AuditCTx}(pk, \{\mathsf{ctx}\}, f, \pi)$: on input a user's public key, a set of confidential transactions he participated, a policy f and a proof, output "0" denotes accept and "1" denotes reject. An auditor runs this algorithm to check if $f(pk, \{\mathsf{ctx}\}) = 1$.

3.3 Correctness

Correctness of basic DCP functionality requires that a valid ctx will always be accepted and recorded on the blockchain, and the states of associated accounts will be updated properly, i.e., the balance of sender account decreases the same amount as the balance of receiver account increases. Correctness of auditing functionality requires honestly generated auditing proofs for transactions complying with policies will always be accept.

3.4 Security Model

Following the treatment of Quisquis [FMMO19], we focus solely on the *transaction layer* of a cryptocurrency, and assume network-level or consensus-level attacks are out of scope.

Intuitively, a DCP system should provide *authenticity*, *confidentiality* and *soundness*. Authenticity requires that the sender can only be the owner of an account, nobody else (who does not know the secret key) is able to make a transfer from this account. Confidentiality requires that other than the sender and receiver (who does not know the secret keys of sender and receiver), no one can learn the value hidden in a confidential transaction. While the former two notions address security against outsider adversary, soundness addresses security against insider adversary (e.g. the sender himself). It requires that no PPT adversary is able to generate a $\mathsf{ctx} = (\mathsf{sn}, \mathsf{memo}, \mathsf{aux})$ such that $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$ but memo does not satisfy f_{correct}, even it knows the secret key of sender.

We formalize the above intuitions into a game-based security model. Let $\mathcal{A}$ be an adversary attacking a DCP system. We assume that $\mathcal{A}$ can not only induce honest parties to perform DCP operations, but can also corrupt some honest parties. Formally, we capture attack behaviors as adversarial queries to oracles implemented by a challenger $\mathcal{CH}$. Below we describe the oracles provided to the adversary.

- $\mathcal{O}_{\mathsf{regH}}$: $\mathcal{A}$ queries this oracle for registering an honest account. $\mathcal{CH}$ keeps track of this type of queries by maintaining a list T_{honest}, which is initially empty. Upon receiving a fresh query, $\mathcal{CH}$ responds as below: picks a random balance $\tilde{m}$ and a serial number sn, runs $\mathsf{CreateAcct}(\tilde{m}, \mathsf{sn})$ to obtain (pk, sk) and $\tilde{C}$. $\mathcal{CH}$ records $(pk, sk, \tilde{C}, \tilde{m}, \mathsf{sn})$ in T_{honest}, then returns $(pk, \tilde{C}, \mathsf{sn})$ to $\mathcal{A}$. This type of oracle captures $\mathcal{A}$ can observe the public information of honest accounts in the system. Hereafter, let Q_{honest} be the maximum number of honest account registration queries that $\mathcal{A}$ makes.

- $\mathcal{O}_{\mathsf{regC}}$: $\mathcal{A}$ queries this oracle with a public key pk, an initial encoded balance $\tilde{C}$ as well as an initial serial number sn. $\mathcal{CH}$ keeps track of this type of queries by maintaining a list T_{corrupt}, which is initially empty. Upon receiving a fresh query, $\mathcal{CH}$ records $(pk, \perp, \tilde{C}, \perp, \mathsf{sn})$ in T_{corrupt}. We stress that pk and $\tilde{C}$ submitted by $\mathcal{A}$ may not be honestly generated. This type of oracle captures $\mathcal{A}$ can fully control some accounts in the system.

- $\mathcal{O}_{\mathsf{extH}}$: $\mathcal{A}$ queries this oracle with a public key pk that was registered as an honest account, a.k.a. in T_{honest}. $\mathcal{CH}$ returns the associated sk to $\mathcal{A}$ and moves this entry to T_{corrupt}. This oracle captures $\mathcal{A}$ can corrupt some honest accounts.

- $\mathcal{O}_{\mathsf{trans}}$: $\mathcal{A}$ queries this oracle with (pk_s, pk_r, v) to conduct a confidential transaction, subject to the restriction that $pk_s \in T_{\text{honest}}$. This restriction is natural because for $pk_s \in T_{\text{corrupt}}$, $\mathcal{A}$ can generate the confidential transaction itself. $\mathcal{CH}$ keeps track of this type of queries by maintaining a list T_{ctx}, which is initially empty. Upon receiving a fresh query, $\mathcal{CH}$ responds as below: If v does not constitute a correct transaction originated from pk_s with balance $\tilde{m}_s$, i.e., $v \notin \mathcal{V}$ or $(\tilde{m}_s - v) \notin \mathcal{V}$, $\mathcal{CH}$ returns $\perp$. Else, $\mathcal{CH}$ runs $\mathsf{ctx} \leftarrow \mathsf{CreateCTx}(sk_s, pk_s, pk_r, v)$, updates the state of associated accounts, records ctx on the ledger T_{ctx}, then sends ctx to $\mathcal{A}$. This oracle captures $\mathcal{A}$ can direct honest accounts to make specific transactions as its will.

- $\mathcal{O}_{\mathsf{inject}}$: $\mathcal{A}$ submits a confidential transaction ctx subject to the restriction that $pk_s \in T_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{CH}$ updates the associated account state and records ctx in T_{ctx}. Otherwise, $\mathcal{CH}$ ignores. This oracle captures $\mathcal{A}$ can generate (possibly malicious) transactions itself.

Authenticity. We define authenticity via the following security experiment between $\mathcal{A}$ and $\mathcal{CH}$.

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \mathsf{VerifyCTx}(\mathsf{ctx}^*) = 1 \wedge \\ pk_s^* \in T_{\text{honest}} \wedge \ \mathsf{ctx}^* \notin T_{\text{ctx}}(pk_s^*) \end{array} : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ \mathsf{ctx}^* \leftarrow \mathcal{A}^{\mathcal{O}}(pp); \end{array}\right].$$

Here $\mathsf{ctx}^* = (\mathsf{sn}^*, \mathsf{memo}^* = (pk_s^*, pk_r^*, C^*), \mathsf{aux}^*)$ is a confidential transaction from pk_s^*, $T_{\text{ctx}}(pk_s^*)$ denotes the set of all the confidential transactions originated from pk_s^* in T_{ctx}. A DCP system satisfies authenticity if no PPT adversary has non-negligible advantage in the above experiment.

Remark 3.1. The above definition of authenticity is essentially much stronger than its intuition. It stipulates no PPT adversary can generate a valid ctx with fresh $(\mathsf{sn}^*, \mathsf{memo}^*, \mathsf{aux}^*)$. In our DCP system, unauthorized transfers against pk_s likely diverge from sender's original intention only when the outsider adversary (without the knowledge sk_s) manages to craft a valid ctx with fresh $(\mathsf{sn}^*, \mathsf{memo}^*)$, because the two pieces encode the core information (serial number, sender and receiver's addresses, and transfer amount) of a transaction. With this insight in mind, we can weaken the definition of authenticity by only requiring freshness requirement for ctx^* over $(\mathsf{sn}^*, \mathsf{memo}^*)$. As we will see in Section 7.3, this weakening leads a more efficient instantiation of our generic DCP construction.

Confidentiality. We define confidentiality via the following security experiment between $\mathcal{A}$ and $\mathcal{CH}$.

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\beta = \beta' : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (state, pk_s^*, pk_r^*, v_0, v_1) \leftarrow \mathcal{A}_1^{\mathcal{O}}(pp); \\ \beta \xleftarrow{\text{R}} \{0,1\}; \\ \mathsf{ctx}^* \leftarrow \mathsf{CreateCTx}(sk_s^*, pk_s^*, pk_r^*, v_\beta); \\ \beta' \leftarrow \mathcal{A}_2^{\mathcal{O}}(state, \mathsf{ctx}^*); \end{array}\right] - \frac{1}{2}.$$

To prevent trivial attacks, $\mathcal{A}$ is subject to the following restrictions: (i) pk_s^*, pk_r^* chosen by $\mathcal{A}_1$ are required to be honest accounts, and $\mathcal{A}_2$ is not allowed to make corrupt queries to either pk_s^* or pk_r^*; (ii) let $\tilde{m}_s$ be the balance of pk_s^*, both v_0 and v_1 must constitute a valid transaction originated from pk_s^*; (iii) let v_{sum} be accumulation of the transfer amounts in $\mathcal{O}_{\text{trans}}$ queries related to pk_s^* after receiving ctx^*, we require that both $\tilde{m}_s - v_0 - v_{\text{sum}}$ and $\tilde{m}_s - v_1 - v_{\text{sum}}$ lie in $\mathcal{V}$. Restriction (i) prevents trivial attack by decryption, and restrictions (ii), (iii) prevent inferring β by testing whether overdraft happens. A DCP system satisfies confidentiality if no PPT adversary has non-negligible advantage in the above experiment.

Soundness. We define soundness via the following security experiment between $\mathcal{A}$ and $\mathcal{CH}$.

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} \mathsf{VerifyCTx}(\mathsf{ctx}^*) = 1 \\ \wedge\ f_{\text{correct}}(\mathsf{memo}^*) = 0 \end{array} : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ \mathsf{ctx}^* \leftarrow \mathcal{A}^{\mathcal{O}}(pp); \end{array}\right].$$

Here, $\mathsf{ctx}^* = (\mathsf{sn}^*, \mathsf{memo}^*, \mathsf{aux}^*)$.

A DCP system satisfies soundness if no PPT adversary has non-negligible advantage in the above experiment.

Secure auditing. Audit is secure if it satisfies the following two requirements: (i) a malicious user is computationally infeasible to fool an auditor to accept a false policy enforcement; (ii) the auditing proof does not reveal more information than satisfiability of the policy.

4 DCP with Auditability

4.1 A Generic Construction of Auditable DCP from ISE and NIZK

We present a generic construction of auditable DCP from ISE and NIZK. In a nutshell, we use homomorphic PKE to encode the balance and transfer amount, use NIZK to enforce senders to build confidential transactions honestly and make correctness publicly verifiable, and use digital signature to authenticate transactions. Let ISE $= (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Enc}, \mathsf{Dec})$ be an ISE scheme whose PKE component is additively homomorphic on message space $\mathbb{Z}_p$. Let NIZK $= (\mathsf{Setup}, \mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$[5] be a NIZK proof system for L_{correct} (which will be specified later). The construction is as below.

- $\mathsf{Setup}(1^\lambda)$: on input a security parameter 1^λ, runs $pp_{\text{ise}} \leftarrow \text{ISE}.\mathsf{Setup}(1^\lambda)$, $pp_{\text{nizk}} \leftarrow \text{NIZK}.\mathsf{Setup}(1^\lambda)$, $crs \leftarrow \text{NIZK}.\mathsf{CRSGen}(pp_{\text{nizk}})$, outputs $pp = (pp_{\text{ise}}, pp_{\text{nizk}}, crs)$.

- $\mathsf{CreateAccount}(\tilde{m}, \mathsf{sn})$: on input an initial balance $\tilde{m} \in \mathbb{Z}_p$ and a serial number $\mathsf{sn} \in \{0,1\}^n$ (e.g., $n = 256$), runs ISE.$\mathsf{KeyGen}(pp_{\text{ise}})$ to generate a keypair (pk, sk), computes $\tilde{C} \leftarrow \text{ISE}.\mathsf{Enc}(pk, \tilde{m}; r)$ as the initial encrypted balance, sets sn as the initial serial number[6], outputs public key pk and secret key sk. Fix the public parameters, the KeyGen algorithm naturally induces an $\mathcal{NP}$ relation $\mathsf{R}_{\text{key}} = \{(pk, sk) : \exists r \text{ s.t. } (pk, sk) = \mathsf{KeyGen}(r)\}$.

- $\mathsf{RevealBalance}(sk, \tilde{C})$: on input secret key sk and encrypted balance $\tilde{C}$, outputs $\tilde{m} \leftarrow \text{ISE}.\mathsf{Dec}(sk, \tilde{C})$.

- $\mathsf{CreateCTx}(sk_s, pk_s, v, pk_r)$: on input sender's keypair (pk_s, sk_s), the transfer amount v, and receiver's public key pk_r, the algorithm first checks if $(\tilde{m}_s - v) \in \mathcal{V}$ and $v \in \mathcal{V}$ (here $\tilde{m}_s$ is the current balance of sender account pk_s). If not, returns $\perp$. Otherwise, it creates a confidential transaction ctx via the following steps:

  1. compute $C_s \leftarrow \text{ISE}.\mathsf{Enc}(pk_s, v; r_1)$, $C_r \leftarrow \text{ISE}.\mathsf{Enc}(pk_r, v; r_2)$, set $\mathsf{memo} = (pk_s, pk_r, C_s, C_r)$, here (C_s, C_r) serve as the encoding of transfer amount;
  2. run NIZK.Prove with witness (sk_s, r_1, r_2, v) to generate a zero-knowledge proof π_{correct} for $\mathsf{memo} = (pk_s, pk_r, C_s, C_r) \in L_{\text{correct}}$, where L_{correct} is defined as below:

  $$\begin{gathered} L_{\text{correct}} = \{(pk_s, pk_r, C_s, C_r) \mid \exists sk_s, r_1, r_2, v \text{ s.t. } C_s = \mathsf{Enc}(pk_s, v; r_1) \wedge C_r = \mathsf{Enc}(pk_r, v; r_2) \\ \wedge\ v \in \mathcal{V}\ \wedge (pk_s, sk_s) \in \mathsf{R}_{\text{key}} \wedge \mathsf{Dec}(sk_s, \tilde{C}_s - C_s) \in \mathcal{V}\} \end{gathered}$$

  L_{correct} can be decomposed as $L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{solvent}}$:

  $$\begin{gathered} L_{\text{equal}} = \{(pk_s, C_s, pk_r, C_r) \mid \exists r_1, r_2, v \text{ s.t. } C_s = \mathsf{Enc}(pk_s, v; r_1) \wedge C_r = \mathsf{Enc}(pk_r, v; r_2)\} \\ L_{\text{right}} = \{(pk_s, C_s) \mid \exists r_1, v \text{ s.t. } C_s = \mathsf{Enc}(pk_s, v; r_1) \wedge v \in \mathcal{V}\} \\ L_{\text{solvent}} = \{(pk_s, \tilde{C}_s, C_s) \mid \exists sk_s \text{ s.t. } (pk_s, sk_s) \in \mathsf{R}_{\text{key}} \wedge \mathsf{Dec}(sk_s, \tilde{C}_s - C_s) \in \mathcal{V}\} \end{gathered}$$

  3. run $\sigma \leftarrow \text{ISE}.\mathsf{Sign}(sk_s, (\mathsf{sn}, \mathsf{memo}))$, here sn is the current serial number of pk_s;
  4. output the entire confidential transaction $\mathsf{ctx} = (\mathsf{sn}, \mathsf{memo}, \mathsf{aux})$, where $\mathsf{aux} = (\pi_{\text{correct}}, \sigma)$.

- $\mathsf{VerifyCTx}(\mathsf{ctx})$: on input $\mathsf{ctx} = (\mathsf{sn}, \mathsf{memo}, \mathsf{aux})$, first parses $\mathsf{memo} = (pk_s, pk_r, C_s, C_r)$, $\mathsf{aux} = (\pi_{\text{correct}}, \sigma)$, then checks its validity via the following steps:

  1. check if sn is a fresh serial number of pk_s (this can be done by inspecting the blockchain);
  2. check if ISE.$\mathsf{Verify}(pk_s, (\mathsf{sn}, \mathsf{memo}), \sigma) = 1$;
  3. check if NIZK.$\mathsf{Verify}(crs, \mathsf{memo}, \pi_{\text{correct}}) = 1$.

  If all the above tests pass, outputs "1", miners confirm that ctx is valid and record it on the blockchain via consensus protocol, sender updates his balance as $\tilde{C}_s = \tilde{C}_s - C_s$ and increments the serial number, and receiver updates his balance as $\tilde{C}_r = \tilde{C}_r + C_r$. Else, outputs "0" and miners discard ctx.

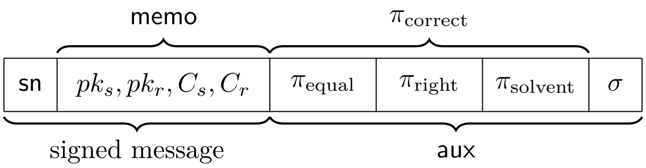

Figure 1: Data structure of confidential transaction.

On the basis that all confidential transactions on the blockchain satisfying basic correctness policy, we further describe how to enforce a variety of useful policies.

- $\mathsf{JustifyCTx}(pk_s, sk_s, \{\mathsf{ctx}_i\}_{i=1}^n, f_{\text{limit}})$: on input pk_s, sk_s, $\{\mathsf{ctx}_i\}_{i=1}^n$ and f_{limit}, first parses $\mathsf{ctx}_i = (\mathsf{sn}_i, \mathsf{memo}_i = (pk_s, pk_{r_i}, C_{s,i}, C_{r_i}), \mathsf{aux}_i)$, then runs NIZK.$\mathsf{Prove}$ with witness sk_s to generate a proof π_{limit} for the statement $(pk_s, \{C_{s,i}\}_{1\leq i\leq n}, a_{\max}) \in L_{\text{limit}}$, where L_{limit} is defined as:

  $$\{(pk, \{C_i\}_{1\leq i\leq n}, a_{\max}) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge v_i = \text{ISE}.\mathsf{Dec}(sk, C_i) \wedge \textstyle\sum_{i=1}^n v_i \leq a_{\max}\}$$

  A user runs this algorithm to prove compliance with limit policy, i.e., the sum of v_i sent from account pk_s is less than $a_{\max}$. The same algorithm can be used to proving the sum of v_i sent to the same account is less than $a_{\max}$.

- $\mathsf{AuditCTx}(pk_s, \{\mathsf{ctx}_i\}_{i=1}^n, \pi_{\text{limit}}, f_{\text{limit}})$: on input pk_s, $\{\mathsf{ctx}_i\}_{i=1}^n$, π_{limit} and f_{limit}, first parses $\mathsf{ctx}_i = (\mathsf{sn}_i, \mathsf{memo}_i = (pk_s, pk_{r_i}, C_{s,i}, C_{r_i}), \mathsf{aux}_i)$, outputs NIZK.$\mathsf{Verify}(crs, (pk_s, \{C_{s,i}\}_{1\leq i\leq n}, a_{\max}), \pi_{\text{limit}})$. The auditor runs this algorithm to check compliance with limit policy.

- $\mathsf{JustifyCTx}(pk_u, sk_u, \{\mathsf{ctx}\}_{i=1}^2, f_{\text{rate}})$: on input pk_u, sk_u, $\{\mathsf{ctx}_i\}_{i=1}^2$ and f_{rate}, the algorithm parses $\mathsf{ctx}_1 = (\mathsf{sn}_1, \mathsf{memo}_1 = (pk_1, pk_u, C_1, C_{u,1}), \mathsf{aux}_1)$ and $\mathsf{ctx}_2 = (\mathsf{sn}_2, \mathsf{memo}_2 = (pk_u, pk_2, C_{u,2}, C_2), \mathsf{aux}_2)$, then runs NIZK.$\mathsf{Prove}$ with witness sk_u to generate a zero-knowledge proof π_{rate} for the statement $(pk_u, C_{u,1}, C_{u,2}, \rho) \in L_{\text{rate}}$, where L_{rate} is defined as:

  $$\{(pk, C_1, C_2, \rho) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge v_i = \text{ISE}.\mathsf{Dec}(sk, C_i) \wedge v_1/v_2 = \rho\}$$

  A user runs this algorithm to demonstrate compliance with tax rule, i.e., proving $v_1/v_2 = \rho$.

- $\mathsf{AuditCTx}(pk_u, \{\mathsf{ctx}_i\}_{i=1}^2, \pi_{\text{rate}}, f_{\text{rate}})$: on input pk_u, $\{\mathsf{ctx}_i\}_{i=1}^2$, π_{rate} and f_{rate}, first parses $\mathsf{ctx}_1 = (\mathsf{sn}_1, \mathsf{memo}_1 = (pk_1, pk_u, C_1, C_{u,1}), \mathsf{aux}_1)$, $\mathsf{ctx}_2 = (\mathsf{sn}_2, \mathsf{memo}_2 = (pk_u, pk_2, C_{u,2}, C_2), \mathsf{aux}_2)$, outputs NIZK.$\mathsf{Verify}(crs, (pk_u, C_{u,1}, C_{u,2}, \rho), \pi_{\text{rate}})$. An auditor runs this algorithm to check compliance with rate policy.

- $\mathsf{JustifyCTx}(sk_u, \mathsf{ctx}, f_{\text{open}})$: on input pk_u, sk_u, ctx and f_{open}, parses $\mathsf{ctx} = (\mathsf{sn}, pk_s, pk_r, C_s, C_r, \mathsf{aux})$, then runs NIZK.$\mathsf{Prove}$ with witness sk_u (where the subscript u could be either s or r) to generate a proof π_{open} for the statement $(pk_u, C_u, v) \in L_{\text{open}}$, where L_{open} is defined as:

  $$\{(pk, C, v^*) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge v^* = \text{ISE}.\mathsf{Dec}(sk, C)\}$$

  A user runs this algorithm to demonstrate compliance with open policy.

- $\mathsf{AuditCTx}(pk_u, \mathsf{ctx}, \pi_{\text{open}}, f_{\text{open}})$: on input pk_u, ctx, π_{open} and f_{open}, the algorithm first parses $\mathsf{ctx} = (\mathsf{sn}, pk_s, pk_r, C_s, C_r, \mathsf{aux})$, then outputs NIZK.$\mathsf{Verify}(crs, (pk_u, C_u, v^*), \pi_{\text{open}})$, where the subscript u could be either s or r. An auditor runs this algorithm to check compliance with open policy.

Remark 4.1. We note that sender's current balance can be inferred from the blockchain. Therefore, it is not necessary to include sender's current balance in the transaction. To speed up the process, validating nodes can maintain a local database of validation state (including the set of mutable accounts), in the same way as most implementation of existing account-based cryptocurrencies (such as Nxt, Ethereum, Bitshares, NEM, Tezos).

[5] We describe our generic DCP construction using NIZK in the CRS model. The construction and security proof carries out naturally if using NIZK in the random oracle model instead.

[6] By default, $\tilde{m}$ and sn should be zero, r should be a fixed and publicly known randomness, say the zero string 0^λ. This settlement guarantees that the initial account state is publicly auditable. Here, we do not make it as an enforcement for flexibility.

4.2 Analysis of Generic DCP Construction

Correctness of our generic DCP construction follows readily from the correctness of ISE and completeness of NIZK. For the security of our DCP construction, we have the following theorem.

Theorem 4.1. *Assuming the security of ISE and NIZK, the above DCP construction is secure.*

Proof. We prove this theorem via the following three lemmas.

Lemma 4.2. *Assuming the security of ISE's signature component and the adaptive zero-knowledge property of NIZK, our DCP construction satisfies authenticity.*

Proof. We proceed via a sequence of games.

Game 0. The real experiment for authenticity. $\mathcal{CH}$ interacts with $\mathcal{A}$ as below.

1. Setup: $\mathcal{CH}$ runs $pp_{\text{ise}} \leftarrow \text{ISE}.\mathsf{Setup}(1^\lambda)$, $pp_{\text{nizk}} \leftarrow \text{NIZK}.\mathsf{Setup}(1^\lambda)$, and $crs \leftarrow \text{NIZK}.\mathsf{CRSGen}(pp_{\text{nizk}})$, then sends $pp = (pp_{\text{ise}}, pp_{\text{nizk}}, crs)$ to $\mathcal{A}$.

2. Queries: Throughout the experiment, $\mathcal{A}$ can adaptively query $\mathcal{O}_{\mathsf{regH}}$, $\mathcal{O}_{\mathsf{regC}}$, $\mathcal{O}_{\mathsf{extH}}$, $\mathcal{O}_{\mathsf{trans}}$ and $\mathcal{O}_{\mathsf{inject}}$, and $\mathcal{CH}$ answers these queries as described above.

3. Forge: $\mathcal{A}$ outputs $\mathsf{ctx}^* = (\mathsf{sn}^*, \mathsf{memo}^* = (pk_s^*, pk_r^*, C_s^*, C_r^*), \pi_{\text{valid}}^*, \sigma^*)$, and wins if $pk_s^* \in T_{\text{honest}} \wedge \mathsf{VerifyCTx}(\mathsf{ctx}^*) = 1 \wedge (\mathsf{sn}^*, \mathsf{memo}^*) \notin T_{\text{ctx}}(pk_s^*)$.

Game 1. Game 1 is same as Game 0 except $\mathcal{CH}$ makes a random guess for the index of target pk_s^* at the beginning, i.e., randomly picks an index $j \in [Q_{\text{honest}}]$. If $\mathcal{A}$ makes an extraction query of pk_j in the learning stage, or $\mathcal{A}$ picks $pk_s^* \neq pk_j$ in the challenge stage, $\mathcal{CH}$ aborts.

Let W be the event that $\mathcal{CH}$ does not abort. It is easy to see that $\Pr[W] \geq 1/Q_{\text{honest}}$. Conditioned on $\mathcal{CH}$ does not abort, $\mathcal{A}$'s view in Game 0 is identical to that in Game 1. Thereby, we have:

$$\Pr[S_1] \geq \Pr[S_0] \cdot \frac{1}{Q_{\text{honest}}}$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ generates the zero-knowledge proof via running $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ in the simulation mode. More precisely. In the Setup stage, $\mathcal{CH}$ runs $(crs, \tau) \leftarrow \mathcal{S}_1(pp_{\text{nizk}})$. When handling transaction queries, $\mathcal{CH}$ runs $\mathcal{S}_2(crs, \tau, \mathsf{memo})$ to generate π_{correct} for $\mathsf{memo} \in L_{\text{correct}}$. By a direct reduction to the adaptive zero-knowledge property of the underlying NIZK, we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{negl}(\lambda)$$

We now argue that no PPT adversary has non-negligible advantage in Game 2.

Claim 4.3. *Assuming the EUF-CMA security of ISE's signature component,* $\Pr[S_2] \leq \mathsf{negl}(\lambda)$ *for all PPT adversary* $\mathcal{A}$.

Proof. Suppose there exists a PPT $\mathcal{A}$ has non-negligible advantage in Game 2, we can build an adversary $\mathcal{B}$ breaks the EUF-CMA security of ISE with the same advantage. Given the challenge (pp_{ise}, pk^*), $\mathcal{B}$ simulates Game 2 as follows:

1. Setup: $\mathcal{B}$ runs $pp_{\text{nizk}} \leftarrow \text{NIZK}.\mathsf{Setup}(1^\lambda)$, $\mathcal{S}_1.\mathsf{Setup}(pp_{\text{nizk}}) \to (crs, \tau)$, sends $pp = (pp_{\text{ise}}, pp_{\text{nizk}}, crs)$ to $\mathcal{A}$. $\mathcal{B}$ randomly picks an index $j \in [Q_{\text{honest}}]$

2. Queries: Throughout the experiment, $\mathcal{A}$ can query the following types of oracles adaptively. $\mathcal{B}$ answers these queries by maintaining two lists T_{honest} and T_{corrupt}, which both are initially empty.

   - $\mathcal{O}_{\mathsf{regH}}$: On the i-th query, $\mathcal{B}$ picks a random balance $\tilde{m}$ and a serial number sn, then proceeds as below:
     – If $i \neq j$, $\mathcal{B}$ runs $\mathsf{CreateAcct}(\tilde{m}, \mathsf{sn})$ to obtain (pk, sk) and the initial encrypted balance $\tilde{C} \leftarrow \text{ISE}.\mathsf{Enc}(pk, \tilde{m})$, then records $(pk, sk, \tilde{C}, \tilde{m}, \mathsf{sn})$ in T_{honest}.
     – If $i = j$, $\mathcal{B}$ sets $pk = pk^*$, $\mathcal{B}$ computes $\tilde{C} \leftarrow \text{ISE}.\mathsf{Enc}(pk, \tilde{m})$, then records $(pk, \perp, \tilde{C}, \tilde{m}, \mathsf{sn})$ in T_{honest}.

Finally, $\mathcal{B}$ returns $(pk, \tilde{C}, \mathsf{sn})$ to $\mathcal{A}$.

- $\mathcal{O}_{\mathsf{regC}}$: $\mathcal{A}$ makes this query with a public key pk, a serial number sn and an initial encrypted balance $\tilde{C}$. $\mathcal{B}$ records $(pk, \bot, \tilde{C}, \bot, \mathsf{sn})$ in T_{corrupt}.
- $\mathcal{O}_{\mathsf{extH}}$: $\mathcal{A}$ makes a query with pk in T_{honest}. If $pk = pk_j$, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ returns sk to $\mathcal{A}$, then moves the corresponding entry to T_{corrupt}.
- $\mathcal{O}_{\mathsf{trans}}$: $\mathcal{A}$ makes a transaction query (pk_s, pk_r, v) subject to the restriction that $pk_s \in T_{\text{honest}}$. Let sn be the serial number, $\tilde{C}_s$ be the encrypted balance of pk_s. $\mathcal{B}$ proceeds as below:
  (a) compute $C_s \leftarrow \text{PKE}.\mathsf{Enc}(pk_s, v)$, $C_r \leftarrow \text{PKE}.\mathsf{Enc}(pk_r, v)$;
  (b) set $\mathsf{memo} = (\tilde{C}_s, pk_r, C_r, pk_s, C_s)$, compute $\pi_{\text{correct}} \leftarrow \mathcal{S}_2(crs, \tau, \mathsf{memo})$;
  (c) if $pk_s \neq pk_j$, generate a signature σ for $(\mathsf{sn}, \mathsf{memo}, \pi_{\text{correct}})$ with sk_s; else, generate such signature σ by querying the signing oracle.

  $\mathcal{B}$ updates the states of associated accounts, then returns ctx to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{inject}}$: $\mathcal{A}$ submits a confidential transaction ctx subject to the restriction that $pk_s \in T_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{B}$ updates the associated account state. Otherwise, $\mathcal{B}$ ignores.

3. Forge: Finally, $\mathcal{A}$ outputs $\mathsf{ctx}^* = (\mathsf{sn}^*, \mathsf{memo}^*, \mathsf{aux}^*)$, where $\mathsf{memo}^* = (pk_s^*, pk_r^*, C_s^*, C_r^*)$ and $\mathsf{aux}^* = (\pi^*_{\text{correct}}, \sigma^*)$. If $pk_s^* \neq pk_j$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ forwards $(\mathsf{sn}^*, \mathsf{memo}^*, \sigma^*)$ to its own challenger. Clearly, $\mathcal{B}$ wins with the same advantage as $\mathcal{A}$ wins.

It is easy to see that $\mathcal{B}$'s simulation for Game 2 is perfect. The claim immediately follows. □

This proves Lemma 4.2. □

Lemma 4.4. *Assuming the security of ISE's PKE component and the zero-knowledge property of NIZK, our DCP construction satisfies confidentiality.*

Proof. We proceed via a sequence of games.

Game 0. The real experiment for confidentiality. $\mathcal{CH}$ interacts with $\mathcal{A}$ as below.

1. Setup: $\mathcal{CH}$ runs $pp_{\text{ise}} \leftarrow \text{ISE}.\mathsf{Setup}(1^\lambda)$, $pp_{\text{nizk}} \leftarrow \text{NIZK}.\mathsf{Setup}(1^\lambda)$, and $crs \leftarrow \text{NIZK}.\mathsf{CRSGen}(pp_{\text{nizk}})$, then sends $pp = (pp_{\text{ise}}, pp_{\text{nizk}}, crs)$ to $\mathcal{A}$.

2. Pre-challenge queries: Throughout the experiment, $\mathcal{A}$ can adaptively query $\mathcal{O}_{\mathsf{regH}}$, $\mathcal{O}_{\mathsf{regC}}$, $\mathcal{O}_{\mathsf{extH}}$, $\mathcal{O}_{\mathsf{trans}}$ and $\mathcal{O}_{\mathsf{inject}}$, and $\mathcal{CH}$ answers these queries as described above.

3. Challenge: $\mathcal{A}$ picks sender pk_s^*, receiver pk_r^* and two transfer values v_0, v_1 as the challenge, subject to the restriction that $pk_s^*, pk_r^* \in T_{\text{honest}}$ and both v_0 and v_1 constitute valid transactions from pk_s^*. $\mathcal{CH}$ picks a random bit β, computes $\mathsf{ctx}^* \leftarrow \mathsf{CreateCTx}(sk_s^*, pk_s^*, pk_r^*, v_\beta)$ and sends ctx^* to $\mathcal{A}$.

4. Post-challenge queries: After receiving the challenge, $\mathcal{A}$ can continue query $\mathcal{O}_{\mathsf{regH}}$, $\mathcal{O}_{\mathsf{regC}}$, $\mathcal{O}_{\mathsf{extH}}$, $\mathcal{O}_{\mathsf{trans}}$ and $\mathcal{O}_{\mathsf{inject}}$, $\mathcal{CH}$ responds the same way as in pre-challenge stage, but with the following exceptions: (i) reject $\mathcal{A}$'s $\mathcal{O}_{\mathsf{extH}}$ query with pk_s^* or pk_r^*; (ii) reject $\mathcal{A}$'s $\mathcal{O}_{\mathsf{trans}}$ query with pk_s^* if it will make either $(\tilde{m}_s^* - v_0 - v_{\text{sum}})$ or $(\tilde{m}_s^* - v_1 - v_{\text{sum}})$ fall outside of $\mathcal{V}$. Here, v_{sum} is the accumulation of transfer amounts related to pk_s^* in post-challenge $\mathcal{O}_{\mathsf{trans}}$ queries.

5. Guess: $\mathcal{A}$ outputs a guess β' for β and wins if $\beta' = \beta$.

Game 1. Same as Game 0 except $\mathcal{CH}$ makes a random guess for index of target accounts (pk_s^*, pk_r^*) at the beginning, i.e., randomly picks two distinct indices $j, k \in [Q_{\text{honest}}]$. If $\mathcal{A}$ makes an extraction query to pk_j or pk_k in pre-challenge queries, or $\mathcal{A}$ picks $(pk_s^*, pk_r^*) \neq (pk_j, pk_k)$ in the challenge stage, $\mathcal{CH}$ aborts.

Let W be the event that $\mathcal{CH}$ does not abort. It is easy to see that $\Pr[W] \geq 1/Q_{\text{honest}}(Q_{\text{honest}} - 1)$. Conditioned on $\mathcal{CH}$ does not abort, $\mathcal{A}$'s view in Game 0 is identical to that in Game 1. Therefore, we have:

$$\Pr[S_1] \geq \Pr[S_0] \cdot \frac{1}{Q_{\text{honest}}(Q_{\text{honest}} - 1)}$$

Game 2. Same as Game 1 except that $\mathcal{CH}$ runs $(crs, \tau) \leftarrow \mathcal{S}_1(pp_{\text{nizk}})$ in the Setup stage, then generates the zero-knowledge proof in the simulation mode. More precisely, when handling transaction queries and challenge queries, $\mathcal{CH}$ runs $\mathcal{S}_2(crs, \tau, \mathsf{memo})$ to generate π_{correct} for $\mathsf{memo} \in L_{\text{correct}}$. By a direct reduction to the adaptive zero-knowledge property of the underlying NIZK, we have:

$$|\Pr[S_2] - \Pr[S_1]| \leq \mathsf{negl}(\lambda)$$

We now argue that no PPT adversary has non-negligible advantage in Game 2.

Claim 4.5. *Assuming the IND-CPA security (1-plaintext, 2-recipient) of the ISE's encryption component,* $\Pr[S_2] \leq \mathsf{negl}(\lambda)$ *for all PPT adversary* $\mathcal{A}$.

Proof. Suppose there exists a PPT adversary $\mathcal{A}$ has non-negligible advantage in Game 2, we can build an adversary $\mathcal{B}$ breaks the IND-CPA security (1-plaintext, 2-recipient) of ISE's encryption component with the same advantage. Given the challenge $(pp_{\text{ise}}, pk_a, pk_b)$, $\mathcal{B}$ simulates Game 2 as follows:

1. Setup: $\mathcal{B}$ runs $pp_{\text{nizk}} \leftarrow \text{NIZK}.\mathsf{Setup}(1^\lambda)$, $\mathcal{S}_1(pp_{\text{nizk}}) \rightarrow (crs, \tau)$, sends $pp = (pp_{\text{ise}}, pp_{\text{nizk}}, crs)$ to $\mathcal{A}$. $\mathcal{B}$ randomly picks two indices $j, k \in [Q_{\text{honest}}]$.

2. Pre-challenge queries: Throughout the experiment, $\mathcal{A}$ can query $\mathcal{O}_{\mathsf{regH}}$, $\mathcal{O}_{\mathsf{regC}}$, $\mathcal{O}_{\mathsf{extH}}$, $\mathcal{O}_{\mathsf{trans}}$ and $\mathcal{O}_{\mathsf{inject}}$ adaptively. $\mathcal{B}$ answers these queries by maintaining two lists T_{honest} and T_{corrupt}, which both are initially empty.

   - $\mathcal{O}_{\mathsf{regH}}$: On the i-th query, $\mathcal{B}$ randomly picks an initial balance $\tilde{m}$ and serial number sn and proceeds as below:
     - If $i \neq j$ and k, $\mathcal{B}$ runs $\mathsf{CreateAcct}(\tilde{m}, \mathsf{sn})$ to obtain (pk, sk) and encrypted balance $\tilde{C} \leftarrow \text{ISE}.\mathsf{Enc}(pk, \tilde{m})$, then records $(pk, sk, \tilde{C}, \tilde{m}, \mathsf{sn})$ in T_{honest}.
     - If $i = j$ or k, $\mathcal{B}$ sets $pk = pk_a$ or pk_b, $\mathcal{B}$ computes $\tilde{C} \leftarrow \text{ISE}.\mathsf{Enc}(pk, \tilde{m})$, then records $(pk, \perp, \tilde{C}, \tilde{m}, \mathsf{sn})$ in T_{honest}.

     Finally, $\mathcal{B}$ returns $(pk, \tilde{C}, \mathsf{sn})$ to $\mathcal{A}$.
   - $\mathcal{O}_{\mathsf{regC}}$: $\mathcal{A}$ makes this query with a public key pk, a serial number sn and an initial encrypted balance $\tilde{C}$. $\mathcal{B}$ records $(pk, \perp, \tilde{C}, \perp, sn)$ in T_{corrupt}.
   - $\mathcal{O}_{\mathsf{extH}}$: $\mathcal{A}$ makes this query with pk in T_{honest}. If $pk = pk_j$ or pk_k, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ returns the associated sk to $\mathcal{A}$, then moves the corresponding entry to T_{corrupt}.
   - $\mathcal{O}_{\mathsf{trans}}$: $\mathcal{A}$ makes a transaction query (pk_s, pk_r, v) subject to the restriction that $pk_s \in T_{\text{honest}}$. Let sn and $\tilde{C}_s$ be the serial number and encrypted balance of pk_s, $\mathcal{B}$ proceeds as below:
     (a) compute $C_s \leftarrow \text{ISE}.\mathsf{Enc}(pk_s, v)$, $C_r \leftarrow \text{ISE}.\mathsf{Enc}(pk_r, v)$;
     (b) set $\mathsf{memo} = (pk_r, pk_s, C_r, C_s)$, compute $\pi_{\text{correct}} \leftarrow \mathcal{S}_2(crs, \tau, \mathsf{memo})$;
     (c) if $pk_s \neq \{pk_j, pk_k\}$, generate a signature σ for $(\mathsf{sn}, \mathsf{memo}, \pi_{\text{correct}})$ with sk_s; else, generate such signature σ by querying the signing oracle of the underlying ISE.

     $\mathcal{B}$ updates the associated account state, then returns ctx to $\mathcal{A}$.
   - $\mathcal{O}_{\mathsf{inject}}$: $\mathcal{A}$ submits a confidential transaction ctx subject to the restriction that $pk_s \in T_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{B}$ updates the associated account state. Otherwise, $\mathcal{B}$ ignores.

3. Challenge: $\mathcal{A}$ picks sender pk_s^*, receiver pk_r^* and two transfer values v_0, v_1 as the challenge, subject to the restriction that $pk_s^*, pk_r^* \in T_{\text{honest}}$ and both v_0 and v_1 constitute a valid transaction from pk_s^*. If $(pk_s^*, pk_r^*) \neq (pk_j, pk_k)$, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ submits (v_0, v_1) to its own challenger and receives back $C^* = (C_s^*, C_r^*)$, which is an encryption of v_β under (pk_s^*, pk_r^*). Let sn^* and $\tilde{C}_s^*$ be the serial number and encrypted balance of pk_s^*, $\mathcal{B}$ then prepares ctx^* as follows:

   (a) set $\mathsf{memo}^* = (pk_s^*, pk_r^*, C_s^*, C_r^*)$.
   (b) generate $\pi_{\text{correct}}^* \leftarrow \mathcal{S}_2(crs, \tau, \mathsf{memo}^*)$.
   (c) query the signing oracle of ISE to obtain a signature σ^* of $(\mathsf{sn}^*, \mathsf{memo}^*)$ under sk_s^*.

   $\mathcal{B}$ updates the states of accounts pk_s^* and pk_r^*, then sends $\mathsf{ctx}^* = (\mathsf{sn}^*, \mathsf{memo}^*, \pi_{\text{correct}}^*, \sigma^*)$ to $\mathcal{A}$ as the challenge.

4. Post-challenge queries: After receiving the challenge, $\mathcal{A}$ can continue query to $\mathcal{O}_{\mathsf{regH}}$, $\mathcal{O}_{\mathsf{regC}}$, $\mathcal{O}_{\mathsf{extH}}$, $\mathcal{O}_{\mathsf{trans}}$ and $\mathcal{O}_{\mathsf{inject}}$, $\mathcal{B}$ responds the same way as in pre-challenge stage, but with the following exceptions: (i) reject $\mathcal{A}$'s $\mathcal{O}_{\mathsf{extH}}$ query with pk_s^* or pk_r^*; (ii) reject $\mathcal{A}$'s $\mathcal{O}_{\mathsf{trans}}$ query with pk_s^* if it will make either $(\tilde{m}_s^* - v_0 - v_{\text{sum}})$ or $(\tilde{m}_s^* - v_1 - v_{\text{sum}})$ fall outside of $\mathcal{V}$. Here, v_{sum} is the accumulation of transfer amounts related to pk_s^* in post-challenge $\mathcal{O}_{\mathsf{trans}}$ queries.

5. Guess: Finally, $\mathcal{A}$ outputs its guess β' for β. $\mathcal{B}$ forwards $\mathcal{A}$'s guess to its own challenger. Clearly, $\mathcal{B}$ wins with the same advantage as $\mathcal{A}$ wins.

It is easy to see that $\mathcal{B}$'s simulation for Game 2 is perfect. The claim immediately follows. □

This proves Lemma 4.4. □

Lemma 4.6. *Assuming the soundness of NIZK, our DCP construction satisfies soundness.*

Proof. Note that $\mathsf{VerifyCTx}(\mathsf{ctx}^*) = 1 \wedge f_{\text{correct}}(\mathsf{memo}^*) = 0$ implies that NIZK.$\mathsf{Verify}(crs, \mathsf{memo}^*, \pi^*_{\text{correct}}) = 1 \wedge \mathsf{memo}^* \notin L_{\text{correct}}$. The reduction is thus straightforward. We omit the details here. □

Putting Lemma 4.2, 4.4 and 4.6 together, we proves the theorem. □

The secure auditing property follows directly from the soundness and zero-knowledge properties of the used NIZK. We omit the details for its triviality.

5 PGC: an Efficient Instantiation

We now present an efficient realization of our generic DCP construction. We first instantiate ISE from our newly introduced twisted ElGamal PKE and Schnorr signature, then devise NIZK proofs from Sigma protocols and Bulletproof.

5.1 Instantiating ISE

We first describe twisted ElGamal encryption and recall Schnorr signature, which share the same setup and key generation algorithms. We then formally prove their integration constitutes ISE that satisfies joint security.

Twisted ElGamal. We propose twisted ElGamal encryption as the PKE component. Formally, twisted ElGamal consists of four algorithms as below:

- $\mathsf{Setup}(1^\lambda)$: run $(\mathbb{G}, g, p) \leftarrow \mathsf{GroupGen}(1^\lambda)$, pick $h \xleftarrow{\text{R}} \mathbb{G}^*$, set $pp = (\mathbb{G}, g, h, p)$ as global public parameters. The randomness and message spaces are $\mathbb{Z}_p$.
- $\mathsf{KeyGen}(pp)$: on input pp, choose $sk \xleftarrow{\text{R}} \mathbb{Z}_p$, set $pk = g^{sk}$.
- $\mathsf{Enc}(pk, m; r)$: compute $X = pk^r$, $Y = g^r h^m$, output $C = (X, Y)$.
- $\mathsf{Dec}(sk, C)$: parse $C = (X, Y)$, compute $h^m = Y/X^{sk^{-1}}$, recover m from h^m.

Remark 5.1. As with the standard exponential ElGamal, decryption can only be efficiently done when the message is small. However, it suffices to instantiate our generic DCP framework with small message space (say, 32-bits). In this setting, our implementation shows that decryption can be very efficient. Indeed, it is actually 30 times faster than the well-known homomorphic encryption scheme Paillier at the same security level. See Section 7.1 for details.

Correctness and additive homomorphism are obvious. The standard IND-CPA security can be proved in the standard model based on the divisible DDH assumption. We provide the proof in Appendix A for the sake of completeness.

In this work, we require twisted ElGamal is also secure in the single plaintext, two recipient setting. A natural solution is simply concatenating independently encryptions for two recipients, a.k.a., $\mathsf{Enc}(pk_1, m; r_1) || \mathsf{Enc}(pk_2, m; r_2)$. Security of this solution is implied by the results independently proved by Bellare et al. [BBM00] and Baudron et al. [BPS00]. Kurosawa [Kur02] showed that for standard ElGamal PKE, randomness can be reused in the single-plaintext, multi-recipient setting. Zether [BAZB20] utilizes Kurosawa's result to make their zero-knowledge component more efficient. Thanks to the random self-reducibility of the divisible DDH problem, our twisted ElGamal PKE is also secure in the single-plaintext, multi-recipient setting even after implementing the randomness reusing trick. This not only shortens the overall transaction size, but also improves the efficiency of the associated zero-knowledge proofs. The security proof is summarized in the following theorem.

Theorem 5.1. *Twisted ElGamal is IND-CPA secure (1-plaintext/2-recipient) based on the divisible DDH assumption.*

Proof. We proceed via two games. Let S_i be the probability that $\mathcal{A}$ wins in Game i.

Game 0. The real IND-CPA security experiment in the 1-plaintext/2-recipient setting. Challenger $\mathcal{CH}$ interacts with $\mathcal{A}$ as below:

1. Setup: $\mathcal{CH}$ runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$, runs $\mathsf{KeyGen}(pp)$ twice independently to obtain $(pk_1 = g^{sk_1}, sk_1)$ and $(pk_2 = g^{sk_2}, sk_2)$, then sends (pp, pk_1, pk_2) to $\mathcal{A}$.
2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and fresh randomness r, computes $X_1 = pk_1^r$, $X_2 = pk_2^r$, $Y = g^r h^{m_\beta}$, sends $C = (X_1, X_2, Y)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition of Game 0, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0] - 1/2.$$

Game 1. Same as Game 0 except $\mathcal{CH}$ generates the challenge ciphertext in a different way:

2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and two independent randomness r and s, computes $X_1 = pk_1^r$, $X_2 = pk_2^r$, $Y = g^s h^{m_\beta}$, sends $C = (X_1, X_2, Y)$ to $\mathcal{A}$.

In Game 1, the distribution of C is independent of β, thus the following holds even for unbounded $\mathcal{A}$:

$$\Pr[S_1] = 1/2$$

It remains to prove $\Pr[S_1]$ and $\Pr[S_0]$ are negligibly close. We prove this by showing if not so, we can build an adversary $\mathcal{B}$ breaks the divisible DDH assumption with the same advantage. Let the public parameter be $(\mathbb{G}, g, p)$, given the divisible DDH challenge instance (g, g^a, g^b, g^c), $\mathcal{B}$ is asked to decide if it is a divisible DDH tuple or a random tuple. To do so, $\mathcal{B}$ interacts with $\mathcal{A}$ by simulating $\mathcal{A}$'s challenger in the following IND-CPA experiment.

1. Setup: $\mathcal{B}$ picks $t \xleftarrow{\text{R}} \mathbb{Z}_p$, picks $h \xleftarrow{\text{R}} \mathbb{G}^*$, sends $pp = (\mathbb{G}, g, h, p)$ and $pk_1 = g^b$ and $pk_2 = g^{bt}$ to $\mathcal{A}$. Here, b and bt serve as sk_1 and sk_2, which are unknown to $\mathcal{B}$.
2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{B}$. $\mathcal{B}$ picks a random bit β and sets $X_1 = g^a$, $X_2 = g^{at}$, $Y = g^c h^{m_\beta}$, sends $C = (X_1, X_2, Y)$ to $\mathcal{A}$.
3. Guess: $\mathcal{A}$ outputs a guess β' for β. $\mathcal{B}$ outputs "1" if $\beta' = \beta$ and "0" otherwise.

If (g, g^a, g^b, g^c) is a divisible DDH tuple, $\mathcal{B}$ simulates Game 0 perfectly (with randomness $c = a/b$). Else, $\mathcal{B}$ simulates Game 1 perfectly (with two independent randomness a/b and c). Thereby, we have $\mathsf{Adv}_{\mathcal{B}} = |\Pr[S_0] - \Pr[S_1]|$, which is negligible in λ by the divisible DDH assumption.

Putting all the above together, Theorem 5.1 follows. □

Schnorr Signature. We choose Schnorr signature [Sch91] as the signature component. The choice is out of the following reasons. First, the setup and key generation algorithms of Schnorr signature are almost identical to those of twisted ElGamal. Second, the signing procedure of Schnorr signature is irrelevant to the decryption procedure of twisted ElGamal. This suggests that we are able to safely implement key reuse strategy to build ISE, as we will rigorously prove later.

Last but not the least, Schnorr signature is efficient and can be easily adapted to multi-signature scheme, which is particularly useful in cryptocurrencies setting [BN06], e.g. shrinking the size of the ledger [BDN18].

For the sake of completeness, we recall the Schnorr signature and sketch its security proof as below.

- $\mathsf{Setup}(1^\lambda)$: run $(\mathbb{G}, g, p) \leftarrow \mathsf{GroupGen}(1^\lambda)$, pick a cryptographic hash function $\mathsf{H} : M \times \mathbb{G} \to \mathbb{Z}_p$, where M denotes the message space. Finally, output $pp = (\mathbb{G}, g, p, \mathsf{H})$ as global public parameters.
- $\mathsf{KeyGen}(pp)$: on input pp, choose $sk \xleftarrow{\text{R}} \mathbb{Z}_p$, set $pk = g^{sk}$.
- $\mathsf{Sign}(sk, m)$: pick $r \xleftarrow{\text{R}} \mathbb{Z}_p$, set $A = g^r$, compute $e = \mathsf{H}(m, A)$, $z = r + sk \cdot e \bmod p$, output $\sigma = (A, z)$.
- $\mathsf{Verify}(pk, \sigma, m)$: parse $\sigma = (A, z)$, compute $e = \mathsf{H}(m, A)$, output "1" if $g^z = A \cdot pk^e$ and "0" otherwise.

Theorem 5.2 ([PS00])**.** *Assume* H *is a random oracle, Schnorr signature is EUF-CMA secure based on the discrete logarithm assumption.*

We sketch the security proof due to Pointcheval and Stern [PS00] as follows. On a high level, the reduction $\mathcal{R}$ (from DLP to the EUF-CMA security of Schnorr signature) works as follows. Given the DLP challenge instance (g, g^a), $\mathcal{R}$ embeds g^a into the public key (i.e., sets $pk = g^a$), then simulates the signing oracle by programming the random oracle H without knowledge of a and uses the oracle-replay attack to obtain two different forgeries that share the signing randomness ($A = g^r$) from the forger. This enables $\mathcal{R}$ to solve for a.

ISE from Schnorr signature and twisted ElGamal encryption. By merging the Setup and KeyGen algorithms of twisted ElGamal encryption and Schnorr signature, we obtain the ISE scheme, whose joint security is captured by the following theorem.

Theorem 5.3. *The obtained ISE scheme is jointly secure if the twisted ElGamal is IND-CPA secure (1-plaintext/2-recipient) and the Schnorr signature is EUF-CMA secure.*

Proof. As we analyzed before, in our generic DCP construction we only require the PKE component of ISE satisfying passive security, thus the security of the signature component is implied by its standalone EUF-CMA security. In what follows, we prove the security for the PKE component, namely IND-CPA security (in the 1-plaintext/2-recipient setting) in the presence of a signing oracle for Schnorr signature. We proceed via a sequence of games.

Game 0. The real security experiment for ISE's PKE component (cf. Definition 2.4). Challenger $\mathcal{CH}$ interacts with $\mathcal{A}$ as below:

1. Setup: $\mathcal{CH}$ runs $pp \leftarrow \mathsf{Setup}(1^\lambda)$, runs $\mathsf{KeyGen}(pp)$ twice independently to obtain $(pk_1 = g^{sk_1}, sk_1)$ and $(pk_2 = g^{sk_2}, sk_2)$, then sends (pp, pk_1, pk_2) to $\mathcal{A}$.
2. Queries: Throughout the experiment, $\mathcal{A}$ can make hash queries and signing queries. $\mathcal{CH}$ emulates random oracle by maintaining a list T_{hash} using standard lazy method. T_{hash} is initially empty, which stores triples $(\cdot, \cdot, \cdot) \in M \times \mathbb{G} \times \mathbb{Z}_p$, an entry (m, A, e) indicates that $e := \mathsf{H}(m, A)$. Concretely, $\mathcal{CH}$ responds $\mathcal{A}$'s queries as below:
   - Hash queries: On query (m, A), if there is an entry (m, A, e) in T_{hash}, $\mathcal{CH}$ returns e. Else, $\mathcal{CH}$ picks $e \xleftarrow{\text{R}} \mathbb{Z}_p$ and inserts (m, A, e) in T_{hash}, then returns e.
   - Signing queries for sk_1 or sk_2: $\mathcal{CH}$ responds with the corresponding secret key. On query (b, m) where $b \in \{0, 1\}$ indicating which secret key is used, $\mathcal{CH}$ picks a fresh randomness $r \xleftarrow{\text{R}} \mathbb{Z}_p$, sets $A = g^r$, obtains $e = \mathsf{H}(m, A)$ by accessing the hash oracle, then computes $z = r + sk_b \cdot e \bmod p$, returns $\sigma = (A, z)$.
3. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and fresh randomness r, computes $X_1 = pk_1^r$, $X_2 = pk_2^r$, $Y = g^r h^{m_\beta}$, sends $C = (X_1, X_2, Y)$ to $\mathcal{A}$.

4. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition of Game 0, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0] - 1/2.$$

Game 1. The same as Game 0 except that $\mathcal{CH}$ simulates signing oracle by programming random oracle H, rather than using the real secret keys. To do so, $\mathcal{CH}$ emulates random oracle by maintaining a list T^*_{hash}, which is initially empty. T^*_{hash} stores triples $(\cdot,\cdot,\cdot,\cdot,\cdot) \in M \times \mathbb{G} \times \mathbb{Z}_p \times \mathbb{Z}_p \times \{0,1\}$, an entry $(m, A, e, *, *)$ indicates that $e := \mathsf{H}(m, A)$, where $*$ is the wild-card.

2. Queries: $\mathcal{CH}$ handles hash queries and signing queries as below.
   - Hash queries: On query (m, A), if there is an entry $(m, A, e, *, *)$ in T^*_{hash}, $\mathcal{CH}$ returns e. Else, $\mathcal{CH}$ picks $e \xleftarrow{\text{R}} \mathbb{Z}_p$, inserts $(m, A, e, \perp, \perp)$ in T^*_{hash}, then returns e.
   - Signing queries for sk_1 or sk_2: On query (b, m), if there is an entry (m, A, e, z, b) (with the same value of b and m) in the T^*_{hash} list, $\mathcal{CH}$ simply returns $\sigma = (A, z)$. Otherwise, $\mathcal{CH}$ picks $z, e \xleftarrow{\text{R}} \mathbb{Z}_p$, sets $A = g^z/pk_b^e$, if there is no entry indexed by (m, A) in T^*_{hash}, then inserts (m, A, e, z, b) in T^*_{hash} and returns $\sigma = (A, z)$. Else, $\mathcal{CH}$ aborts to avoid possible inconsistence in programming.

Denote the event that $\mathcal{CH}$ aborts in Game 1 by E. Conditioned on E does not occur, $\mathcal{A}$'s view in Game 0 and Game 1 are identical. This follows from the fact that $\mathcal{CH}$ perfectly mimic the hash oracle and signing oracle. Let Q_{hash} and Q_{sign} be the maximum number of hash queries and signing queries that $\mathcal{A}$ makes during security experiment. By the union bound, we conclude that $\Pr[E] \leq (Q_{\text{hash}} Q_{\text{sign}})/p$, which is negligible in λ. In summary, we have:

$$|\Pr[S_1] - \Pr[S_0]| \leq \Pr[E] \leq \mathsf{negl}(\lambda)$$

It remains to bound $\Pr[S_1]$. We have the following claim.

Claim 5.4. *Assume the IND-CPA security (1-plaintext/2-recipient) of twisted ElGamal PKE,* $\Pr[S_1]$ *is negligible in* λ *for any PPT adversary* $\mathcal{A}$.

Proof. We prove this claim by showing that if there exists a PPT adversary $\mathcal{A}$ has non-negligible advantage in Game 1, we can build a PPT adversary $\mathcal{B}$ that breaks the IND-CPA security (single-message, two-recipient) of twisted ElGamal PKE with the same advantage. Note that according to the definition of Game 1, $\mathcal{CH}$ can simulate the signing oracles without using the secret keys. Thereby, given (pp, pk_1, pk_2), $\mathcal{B}$ can perfectly simulate Game 1 by forwarding the encryption challenge to its own challenger. This proves the claim. □

Putting all the above together, Theorem 5.3 immediately follows. □

5.2 Instantiating NIZK

Now, we design efficient NIZK proof systems for basic correctness policy (L_{correct}) and more extended policies (L_{limit}, L_{rate}, L_{open}).

As stated in Section 4.1, L_{correct} can be decomposed as $L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{solvent}}$. Let Π_{equal}, Π_{right}, Π_{solvent} be NIZK for L_{equal}, L_{right}, L_{solvent} respectively, and let $\Pi_{\text{correct}} := \Pi_{\text{equal}} \circ \Pi_{\text{right}} \circ \Pi_{\text{solvent}}$, where $\circ$ denotes sequential composition.[7] By the property of NIZK for conjunctive statements [Gol06], Π_{correct} is a NIZK proof system for L_{correct}. Now, the task breaks down to design Π_{equal}, Π_{right}, Π_{solvent}. We describe them one by one as below.

[7] In the non-interactive setting, there is no distinction between sequential and parallel composition.

5.2.1 NIZK for L_{equal}

According to our DCP construction and definition of twisted ElGamal, L_{equal} can be written as:

$$\{(pk_1, X_1, Y_1, pk_2, X_2, Y_2) \mid \exists r_1, r_2, v \text{ s.t. } X_i = pk_i^{r_i} \wedge Y_i = g^{r_i} h^v \text{ for } i = 1, 2\}.$$

As analyzed before, for twisted ElGamal randomness can be safely reused in the 1-plaintext/2-recipient setting. L_{equal} can thus be simplified to:

$$\{(pk_1, pk_2, X_1, X_2, Y) \mid \exists r, v \text{ s.t. } Y = g^r h^v \wedge X_i = pk_i^r \text{ for } i = 1, 2\}.$$

Sigma protocol for L_{equal}. To obtain a NIZK for L_{equal}, we first design a Sigma protocol $\Sigma_{\text{equal}} = (\mathsf{Setup}, P, V)$ for L_{equal}. The Setup algorithm of Σ_{equal} is same as that of the twisted ElGamal. On statement $(pk_1, pk_2, X_1, X_2, Y)$, P and V interact as below:

1. P picks $a, b \xleftarrow{R} \mathbb{Z}_p$, sends $A_1 = pk_1^a$, $A_2 = pk_2^a$, $B = g^a h^b$ to V.

2. V picks $e \xleftarrow{R} \mathbb{Z}_p$ and sends it to P as the challenge.

3. P computes $z_1 = a + er$, $z_2 = b + ev$ using witness $w = (r, v)$, then sends (z_1, z_2) to V. V accepts iff the following three equations hold simultaneously:

$$pk_1^{z_1} = A_1 X_1^e \quad (1)$$
$$pk_2^{z_1} = A_2 X_2^e \quad (2)$$
$$g^{z_1} h^{z_2} = B Y^e \quad (3)$$

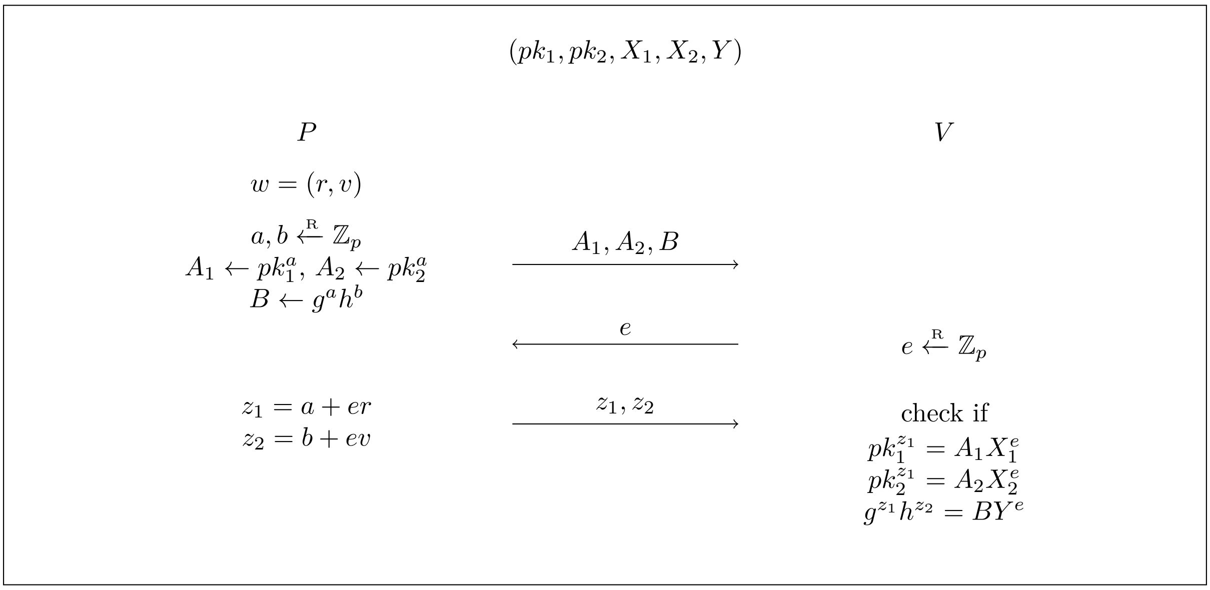
$(pk_1, pk_2, X_1, X_2, Y)$
P
V
$w = (r, v)$
$a, b \xleftarrow{R} \mathbb{Z}_p$
$A_1 \leftarrow pk_1^a$, $A_2 \leftarrow pk_2^a$
$B \leftarrow g^a h^b$
A_1, A_2, B
e
$e \xleftarrow{R} \mathbb{Z}_p$
$z_1 = a + er$
$z_2 = b + ev$
z_1, z_2
check if
$pk_1^{z_1} = A_1 X_1^e$
$pk_2^{z_1} = A_2 X_2^e$
$g^{z_1} h^{z_2} = B Y^e$

Figure 2: Σ_{equal} for L_{equal}: proof of knowledge of two twisted ElGamal ciphertexts encrypting the same value under different public keys

Lemma 5.5. *Σ_{equal} is a public-coin SHVZK proof of knowledge for L_{equal}.*

Proof. We prove that all three properties required for Sigma protocol are met.

Perfect completeness is obvious from simple calculation.

To show special soundness, fix the initial message (A_1, A_2, B), suppose there are two accepting transcripts $(e, z = (z_1, z_2))$ and $(e', z' = (z_1', z_2'))$ with $e \neq e'$, the witness can be extracted as below. From either (1) or (2), we have $z_1 = a + er$ and $z_1' = a + e'r$, which implies $r = (z_1 - z_1')/(e - e')$. Further from (3), we have $z_2 = b + ev$ and $z_2' = b + e'v$, which imply $v = (z_2 - z_2')/(e - e')$.

To show special HVZK, for a fixed challenge e, the simulator $\mathcal{S}$ works as below: picks $z_1, z_2 \xleftarrow{\text{R}} \mathbb{Z}_p$, computes $A_1 = pk_1^{z_1}/X_1^e$, $A_2 = pk_2^{z_1}/X_2^e$, $B = g^{z_1}h^{z_2}/Y^e$. Clearly, $(A_1, A_2, B, e, z_1, z_2)$ is an accepting transcript, and it is distributed as in the real protocol.

This proves Lemma 5.5. □

Applying Fiat-Shamir transform to Σ_{equal}, we obtain Π_{equal}, which is actually a NIZKPoK for L_{equal}.

5.2.2 NIZK for L_{right}

According to our DCP construction and the definition of twisted ElGamal, L_{right} can be written as:

$$\{(pk, X, Y) \mid \exists r, v \text{ s.t. } X = pk^r \wedge Y = g^r h^v \wedge v \in \mathcal{V}\}.$$

For ease of analysis, we additionally define L_{enc} and L_{range} as below:

$$L_{\text{enc}} = \{(pk, X, Y) \mid \exists r, v \text{ s.t. } X = pk^r \wedge Y = g^r h^v\}$$
$$L_{\text{range}} = \{Y \mid \exists r, v \text{ s.t. } Y = g^r h^v \wedge v \in \mathcal{V}\}$$

It is straightforward to verify that $L_{\text{right}} \subset L_{\text{enc}} \wedge L_{\text{range}}$. Observing that each instance $(pk, X, Y) \in L_{\text{right}}$ has a unique witness, while the last component Y can be viewed as a Pedersen commitment of value v under commitment key (g, h), whose discrete logarithm $\log_g h$ is unknown to any users. To prove $(pk, X, Y) \in L_{\text{right}}$, we first prove $(pk, X, Y) \in L_{\text{enc}}$ with witness (r, v) via a Sigma protocol $\Sigma_{\text{enc}} = (\mathsf{Setup}, P_1, V_1)$, then prove $Y \in L_{\text{range}}$ with witness (r, v) via a Bulletproof $\Lambda_{\text{bullet}} = (\mathsf{Setup}, P_2, V_2)$.

Sigma protocol for L_{enc}. We begin with a Sigma protocol $\Sigma_{\text{enc}} = (\mathsf{Setup}, P, V)$ for L_{enc}. The Setup algorithm is same as that of twisted ElGamal. On statement $x = (pk, X, Y)$, P and V interact as below:

1. P picks $a, b \xleftarrow{\text{R}} \mathbb{Z}_p$, sends $A = pk^a$ and $B = g^a h^b$ to V.
2. V picks $e \xleftarrow{\text{R}} \mathbb{Z}_p$ and sends it to P as the challenge.
3. P computes $z_1 = a + er$, $z_2 = b + ev$ using witness $w = (r, v)$, then sends (z_1, z_2) to V. V accepts iff the following two equations hold simultaneously:

$$pk^{z_1} = AX^e \quad (4)$$
$$g^{z_1}h^{z_2} = BY^e \quad (5)$$

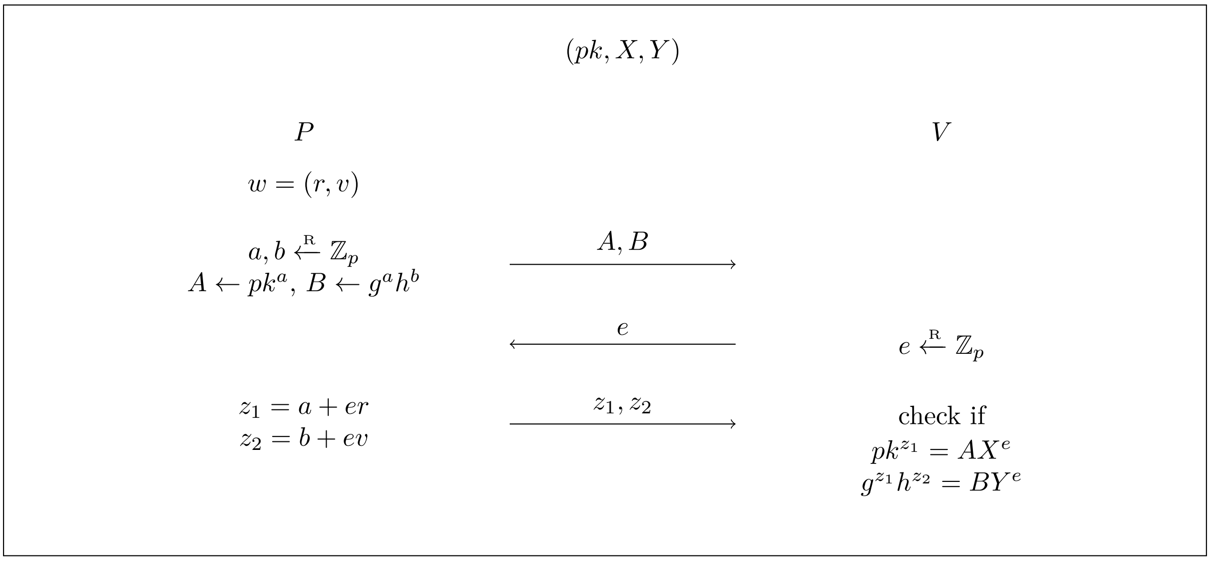

Figure 3: Σ_{enc} for L_{enc}: proof of knowledge of twisted ElGamal ciphertext

Lemma 5.6. *Σ_{enc} is a public-coin SHVZK proof of knowledge for L_{enc}.*

Proof. We prove that all three properties required for Sigma protocol are met.

Perfect completeness is obvious from simple calculation.

To show special soundness, fix the initial message (A, B), suppose there are two accepting transcripts $(e, z = (z_1, z_2))$ and $(e', z' = (z_1', z_2'))$ with $e \neq e'$, we can extract the witness as below. From (4), we have $z_1 = a + er$ and $z_1' = a + e'r$, which implies $r = (z_1 - z_1')/(e - e')$. Further from (5), we have $z_2 = b + ev$ and $z_2' = b + e'v$, which imply $v = (z_2 - z_2')/(e - e')$.

To show special HVZK, for a fixed challenge e, the simulator $\mathcal{S}$ works as below: picks $z_1, z_2 \xleftarrow{\text{R}} \mathbb{Z}_p$, computes $A = pk^{z_1}/X^e$, $B = g^{z_1}h^{z_2}/Y^e$. Clearly, (A, B, e, z_1, z_2) is an accepting transcript, and it is distributed as in the real protocol.

This proves Lemma 5.6. □

Bulletproofs for L_{range}. We employ the logarithmic size Bulletproof $\Lambda_{\text{bullet}} = (\mathsf{Setup}, P, V)$ to prove L_{range}. To avoid repetition, we refer to [BBB$^+$18, Section 4.2] for the details of the interaction between P and V.

Lemma 5.7. *[BBB$^+$18, Theorem 3] Assuming the hardness of discrete logarithm problem, Λ_{bullet} is a public-coin SHVZK argument of knowledge for L_{range}.*

Sequential composition. Let $\Gamma_{\text{right}} = \Sigma_{\text{enc}} \circ \Lambda_{\text{bullet}}$ be the sequential composition of Σ_{enc} and Λ_{bullet}. The Setup algorithm of Γ_{right} is a merge of that of Σ_{enc} and Λ_{bullet}. For range $\mathcal{V} = [0, 2^\ell - 1]$, it first generates a group $\mathbb{G}$ of prime order p together with two random generators g and h, then picks independent generators $\mathbf{g}, \mathbf{h} \in \mathbb{G}^\ell$. Let $P_1 = \Sigma_{\text{enc}}.P$, $V_1 = \Sigma_{\text{enc}}.V$, $P_2 = \Lambda_{\text{bullet}}.P$, $V_2 = \Lambda_{\text{bullet}}.V$. We have $\Gamma_{\text{right}}.P = (P_1, P_2)$, $\Gamma_{\text{right}}.V = (V_1, V_2)$.

Lemma 5.8. *Assuming the discrete logarithm assumption, $\Gamma_{\text{right}} = (\mathsf{Setup}, P, V)$ is a public-coin SHVZK argument of knowledge for L_{right}.*

Proof. The completeness and zero-knowledge properties of Γ_{right} follow from that of Σ_{enc} and Λ_{bullet}.

In order to prove argument of knowledge, it suffices to construct a PPT witness extraction algorithm E. Let E_1 and E_2 be the witness extraction algorithm for Σ_{enc} and Λ_{bullet} respectively. Note that Γ_{right} is a sequential composition of Σ_{enc} and Λ_{bullet}, thus an $(n_0, n_1, \dots, n_k)$-tree of accepting transcripts for Γ_{right} is a composition of an n_0-subtree of accepting transcripts for Σ_{enc} (where $n_0 = 2$) and an $(n_1, \dots, n_k)$-subtree of accepting transcripts for Λ_{bullet} (see [BBB$^+$18] for the concrete values of $n_1, \dots, n_k$).

E first runs E_1 on n_0-subtree of accepting transcripts to extract $w = (r, v)$, then runs E_2 on $(n_1, \dots, n_k)$-subtree of accepting transcripts to extract $w' = (r', v')$. Finally, E outputs w if $w = w'$ and aborts otherwise. Note that $\Pi_{i=0}^k n_i$ is still bounded by a polynomial of λ.

According to Lemma 5.6, E_1 outputs a witness of statement $(pk, X, Y) \in L_{\text{enc}}$ with probability 1. According to Lemma 5.7, E_2 outputs a witness of statement $Y \in L_{\text{range}}$ with overwhelming probability assuming the hardness of DLP. We argue that E_2's output $w' = (r', v')$ equals E_1's output $w = (r, v)$ with overwhelming probability, since otherwise the relation $g^v h^r = Y = g^{v'} h^{r'}$ immediately yields a solution $\log_g h$ to the DLP instance (g, h). We thus conclude that E outputs a witness for $x = (pk, X, Y) \in L_{\text{right}}$ with overwhelming probability. Thus, Γ_{right} satisfies computational witness-extended emulation.

Putting all the above together, Lemma 5.8 immediately follows. □

Applying Fiat-Shamir transform to Γ_{right}, we obtain Π_{right}, which is a NIZKAoK for L_{right}.

5.2.3 NIZK for L_{solvent}

According to our generic DCP construction, L_{solvent} is defined as:

$$\{(pk, \tilde{C}, C) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge \text{ISE}.\mathsf{Dec}(sk, \tilde{C} - C) \in \mathcal{V}\}.$$

In the above, $\tilde{C} = (\tilde{X} = pk^{\tilde{r}}, \tilde{Y} = g^{\tilde{r}} h^{\tilde{m}})$ is the encryption of current balance $\tilde{m}$ of account pk under randomness $\tilde{r}$, $C = (X = pk^r, Y = g^r h^v)$ encrypts the transfer amount v under randomness r. Let $C' = (X' = pk^{r'}, Y' = g^{r'} h^{m'}) = \tilde{C} - C$, L_{solvent} can be rewritten as:

$$\{(pk, C') \mid \exists r', m' \text{ s.t. } C' = \text{ISE}.\mathsf{Enc}(pk, m'; r') \wedge m' \in \mathcal{V}\}.$$

By the additive homomorphism of twisted ElGamal, we have $r' = \tilde{r} - r$, $m' = \tilde{m} - v$. It seems that we can prove L_{solvent} using the same protocol for L_{right}. However, while the sender (playing the role of prover) learns $\tilde{m}$ (by decrypting $\tilde{C}$ with sk), v and r, it generally does not know the randomness $\tilde{r}$. This is because $\tilde{C}$ is the sum of all the incoming and outgoing transactions of pk, whereas the randomness behind incoming transactions is unknown. The consequence is that r' (the first part of witness) is unknown, which renders us unable to directly invoke the Bulletproof on instance Y'.

Our trick is encrypting the value $m' = (\tilde{m} - v)$ under a fresh randomness r^* to obtain a new ciphertext $C^* = (X^*, Y^*)$, where $X^* = pk^{r^*}$, $Y^* = g^{r^*} h^{m'}$. C^* could be viewed as a refreshment of C'. In a nutshell, we express L_{solvent} as $L_{\text{equal}} \wedge L_{\text{right}}$, a.k.a. we have:

$$(pk, C') \in L_{\text{solvent}} \iff (pk, C', C^*) \in L_{\text{equal}} \wedge (pk, C^*) \in L_{\text{right}}$$

To prove C' and C^* encrypting the same value under public key pk, we cannot simply use a Sigma protocol like Protocol 5.2.1, in which the prover uses the message and randomness as witness, as the randomness r' behind C' is typically unknown. Luckily, we are able to prove this more efficiently by using the secret key as witness. Generally, L_{equal} can be written as:

$$\{(pk, C_1, C_2) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge \text{ISE}.\mathsf{Dec}(sk, C_1) = \text{ISE}.\mathsf{Dec}(sk, C_2)\}$$

When instantiated with twisted ElGamal, $C_1 = (X_1 = pk^{r_1}, Y_1 = g^{r_1} h^m)$ and $C_2 = (X_2 = pk^{r_2}, Y_2 = g^{r_2} h^m)$ are ciphertexts under the same public key pk, then proving membership of L_{equal} is equivalent to proving $\log_{Y_1/Y_2} X_1/X_2$ equals $\log_g pk$. This can be efficiently done by utilizing the Sigma protocol $\Sigma_{\text{ddh}} = (\mathsf{Setup}, P, V)$ for discrete logarithm equality due to Chaum and Pedersen [CP92]. For the sake of completeness, we describe it as below.

The Setup algorithm generates a cyclic group $\mathbb{G}$ of prime order p. Define the language L_{ddh} as below:

$$L_{\text{ddh}} = \{(g_1, h_1, g_2, h_2) \mid \exists w \in \mathbb{Z}_p \text{ s.t. } \log_{g_1} h_1 = w = \log_{g_2} h_2\}$$

It is easy to see that $\log_{g_1} h_1 = \log_{g_2} h_2$ iff (g_1, h_1, g_2, h_2) constitutes a DDH tuple. On statement $x = (g_1, h_1, g_2, h_2)$, P interacts with V as below:

1. P picks $a \xleftarrow{\text{R}} \mathbb{Z}_p$, sends $A_1 = g_1^a$, $A_2 = g_2^a$ to V;

2. V picks $e \xleftarrow{\text{R}} \mathbb{Z}_p$ and sends it to P as the challenge;

3. P computes $z = a + we$ with witness w and sends it to V. V accepts iff:

$$g_1^z = A_1 h_1^e \wedge g_2^z = A_2 h_2^e$$

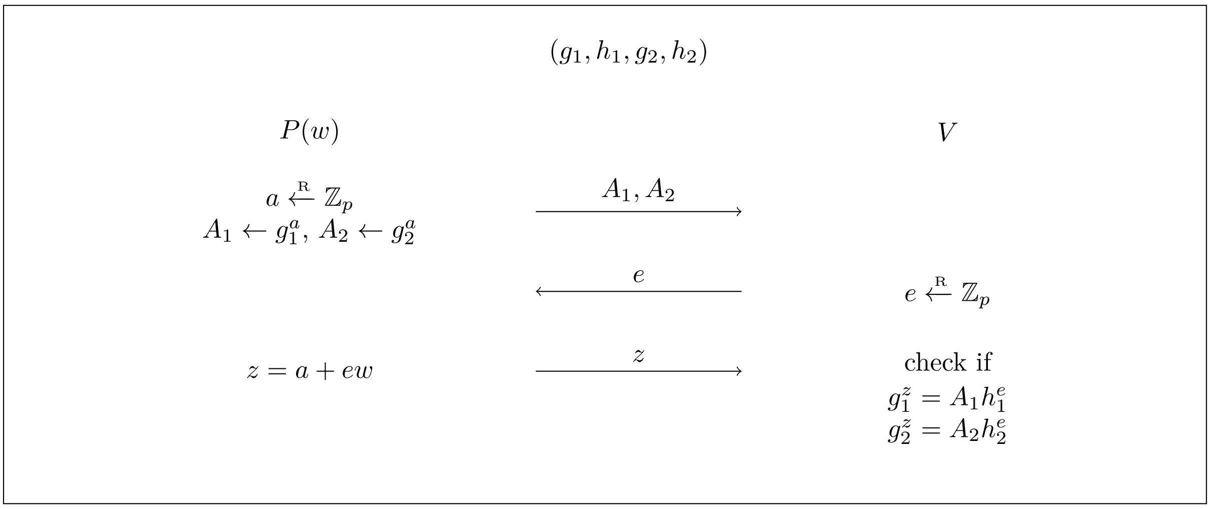
(g_1, h_1, g_2, h_2)
$P(w)$
V
$a \xleftarrow{\text{R}} \mathbb{Z}_p$
$A_1 \leftarrow g_1^a$, $A_2 \leftarrow g_2^a$
A_1, A_2
e
$e \xleftarrow{\text{R}} \mathbb{Z}_p$
$z = a + ew$
z
check if
$g_1^z = A_1 h_1^e$
$g_2^z = A_2 h_2^e$

Figure 4: Σ_{ddh} for L_{ddh}: proof of knowledge of discrete logarithm equality/DDH tuple

Lemma 5.9 ([CP92]). *Σ_{ddh} is a public-coin SHVZK proof of knowledge for L_{ddh}.*

Applying Fiat-Shamir transform to Σ_{ddh}, we obtain a NIZKPoK Π_{ddh} for L_{ddh}. We then prove $(pk, C^* = (X^*, Y^*)) \in L_{\text{right}}$ using the NIZKPoK Π_{right} as we described before. Let $\Pi_{\text{solvent}} = \Pi_{\text{ddh}} \circ \Pi_{\text{right}}$, we conclude that Π_{solvent} is a NIZKPoK for L_{solvent} by the properties of AND-proofs.

Putting all the sub-protocols described above, we obtain $\Pi_{\text{correct}} = \Pi_{\text{equal}} \circ \Pi_{\text{right}} \circ \Pi_{\text{solvent}}$.

Theorem 5.10. *Π_{correct} is a NIZKAoK for $L_{\text{correct}} = L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{solvent}}$.*

Proof. The proof of this theorem follows from the properties of AND-proofs. □

Two useful proof gadgets. The above construction contributes two useful "proof gadgets" for generating range proofs for values in encrypted form. Π_{right} constitutes Gadget-1, which admits a prover to generate range proofs with knowledge of message and randomness. Π_{right} constitutes Gadget-2, which admits a prover to generate range proofs with knowledge of secret key.

5.2.4 NIZK for L_{limit}

According to our DCP construction and the definition of twisted ElGamal, L_{limit} can be written as:

$$\{(pk, \{C_i\}_{1\leq i\leq n}, a_{\max}) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge v_i = \mathsf{ISE.Dec}(sk, C_i) \wedge \textstyle\sum_{i=1}^{n} v_i \leq a_{\max}\}$$

Let $C_i = (X_i = pk^{r_i}, Y_i = g^{r_i} h^{v_i})$. By additive homomorphism of twisted ElGamal, the prover first computes $C = \sum_{i=1}^{n} C_i = (X = pk^r, Y = g^r h^v)$, where $r = \sum_{i=1}^{n} r_i$, $v = \sum_{i=1}^{n} v_i$. As aforementioned, in PGC users are not required to maintain history state, which means that users may forget the random coins when implementing after-the-fact auditing. Nevertheless, this is not a problem. It is equivalent to prove $(pk, C) \in L_{\text{solvent}}$, which can be done by using Gadget-2.

5.2.5 NIZK for L_{rate}

According to our DCP construction and the definition of twisted ElGamal, L_{rate} can be written as:

$$\{(pk, C_1, C_2, \rho) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge v_i = \mathsf{ISE.Dec}(sk, C_i) \wedge v_1/v_2 = \rho\}$$

Without much loss of generality, we assume $\rho = \alpha/\beta$, where α and β are two positive integers that are much smaller than p. Let $C_1 = (pk^{r_1}, g^{r_1} h^{v_1})$, $C_2 = (pk^{r_2}, g^{r_2} h^{v_2})$. By additive homomorphism of twisted ElGamal, we compute $C_1' = \beta \cdot C_1 = (X_1' = pk^{\beta r_1}, Y_1' = g^{\beta r_1} h^{\beta v_1})$, $C_2' = \alpha \cdot C_2 = (X_2' = pk^{\alpha r_2}, Y_2' = g^{\alpha r_2} h^{\alpha v_2})$. Note that $v_1/v_2 = \rho = \alpha/\beta$ if and only if $h^{\beta v_1} = h^{\alpha v_2}$,[8] L_{rate} is equivalent to $(Y_1'/Y_2', X_1'/X_2', g, pk) \in L_{\text{ddh}}$, which in turn can be efficiently proved via Π_{ddh} for discrete logarithm equality using sk as witness, as already described in Protocol 5.2.3.

5.2.6 NIZK for L_{open}

According to our DCP construction and the definition of twisted ElGamal, L_{open} can be written as:

$$\{(pk, C = (X, Y), v) \mid \exists sk \text{ s.t. } X = (Y/h^v)^{sk} \wedge pk = g^{sk}\}$$

The above language is equivalent to $(Y/h^v, X, g, pk) \in L_{\text{ddh}}$, which in turn can be efficiently proved via Π_{ddh} for discrete logarithm equality, as already described in Protocol 5.2.3.

6 Further Discussions

6.1 Transparent Setup

Benefit from our dedicate design, the global parameters of PGC are exactly the public parameters of Bulletproofs, a.k.a. simply random generators $(g, h, \mathbf{g}, \mathbf{h})$ without special structure. We can thus use hash function to dismiss trusted setup. We refer to [BBB+18] for more details on this point.

[8] Since both v_1, v_2, α, β are much smaller than p, no overflow will happen.

6.2 Necessity of Composing Sigma protocol with Bulletproof

We prove the membership of L_{right} by sequentially composing a Sigma protocol for L_{enc} and a Bulletproof for L_{range}. One may wonder if we can only use the Bulletproof for L_{range} to fulfill this task. Actually, that will be problematic. Consider a malicious prover P^* generates ciphertext dishonestly as below: picks random r, r' from $\mathbb{Z}_q$ and $v' \in \mathcal{V}$, sets $X = pk^r$, $Y = g^{r'}h^{v'}$. P^* can thus use r' and v' as witness to generate a valid Bulletproof for $Y \in L_{\text{range}}$. However, the real message v encrypted by (X, Y) falls outside the range $\mathcal{V}$ with overwhelming probability, and thus $(pk, X, Y) \notin L_{\text{right}}$. This compromises proof of knowledge. It is also instructive to examine where the security proof will break. Let (pk, X, Y) be the statement and (r, v) be the corresponding randomness-message pair. To prove witness-extended emulation via weak forking lemma, we hope to argue that the output of Bulletproof's extractor (r', v') equals (r, v) overwhelmingly based on the collision resistance of hash function $\mathsf{H}(x, y) = g^x h^y$, which in turn is implied the hardness of DLP related to (g, h). However, the condition of weak forking lemma is required to hold for any PPT adversary $\mathcal{A}$, and thus there is no way to embed the collision into the input instance.

6.3 Ciphertext Refreshing vs. Key Switching

In order to safely use Bulletproof to prove the membership of L_{solvent}, we refresh $C' = (pk^{r'}, g^{r'}h^{m'})$ to $C^* = (pk^{r^*}, g^{r^*}h^{m'})$, then directly invoking the Bulletproof on the second component of C^* with witness (r^*, m'). To ensure the soundness of the overall protocol, we also need another protocol to prove that C' and C^* encrypt the same message.

One may attempt to use "key switching" trick to avoid the overhead of refreshing. Note that $pk = g^{sk}$, thus $g^{r'}h^{m'}$ can be rewritten as $(pk^{r'})^{sk^{-1}}h^{m'}$. It seems that now we are able to invoke Bulletproof on $(pk^{r'})^{sk^{-1}}h^{m'}$ by interpreting $(pk^{r'}, h)$ as Pedersen commitment key and using (sk^{-1}, m') as the witness. The soundness of the Bulletproof's requires that the prover does not know the DL relation between $(pk^{r'}, h)$. Is it possible for the prover to rigorously prove this intuition? It is well-known that one can generate a proof of knowledge of discrete logarithm using the Schnorr's protocol. But, how can the prover give a proof of no-knowledge? Interestingly, the prover can prove that he indeed does not know $\log_h pk^{r'}$ by proving his knowledge of $\log_g pk^{r'}$. This makes sense because otherwise he can solve the DL relation between (g, h), which contradicts to the assumption. Inevitably, an honest prover falls into a deadlocked setting — he needs to know r' to create a proof of knowledge of $\log_g pk^{r'}$, which is exactly the obstacle he want to overcome at the beginning. Even the soundness of the Bulletproof's part holds, additional mechanism is needed to prove that the extractor's output is consistent to the message fixed by the ciphertext, which seems difficult.

We distill the above reasoning to a trick of proving no-knowledge of discrete logarithm, which might be of independent interest. Let y, g and h be group elements, where the DL relation between (g, h) is unknown. A prover can prove that he does not know the knowledge of $\alpha = \log_h y$ by proving knowledge of $\beta = \log_g y$. The soundness follows because if the prover cheated (he knows $\alpha = \log_h y$), then he must be able to solve the discrete logarithm between g and h by computing $\beta\alpha^{-1}$.

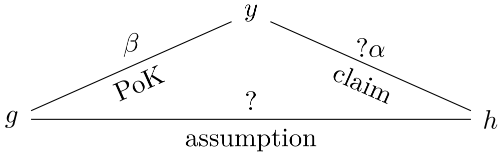

Figure 5: How to prove no-knowledge of $x = \log_h y$ by assuming the DL relation between (g, h) is intractable.

7 Optimizations

7.1 Faster Decryption for Twisted ElGamal

As in standard exponential ElGamal, to ensure additive homomorphism on message space $\mathbb{Z}_p$, message in twisted ElGamal is also encoded to the exponent. This makes decryption inefficient even with knowledge of secret key. This seems problematic because the exact balance is required to be known when generating the range proof for solvency. Similar problem also occurs in other confidential transaction systems [FMMO19, BAZB20].

Fortunately, this is not an issue. First, each user can learn its current balance by keeping track of the incoming and outgoing transfers. While the outgoing transfer amounts are always known to the senders themselves by default, the incoming transfer amounts are also known to the recipients in most times. This makes sense because senders typically communicate the transfer amount to their recipients off-chain in real-world applications. It remains to consider the worst case, i.e., the receivers do not know the exact value of incoming transfer amounts. Suppose the transfer amount is restricted to the range $[0, 2^\ell - 1]$ by protocol specification. Related works [FMMO19, BAZB20] suggested that the receiver can figure out m from g^m by brute-force enumeration with time complexity $O(2^\ell)$. Here, we recommend employing dedicated algorithms to compute a discrete logarithm in an interval more efficiently. For instance, Pollard's kangaroo method [Pol78] and Galbraith-Ruprai algorithm [GR10] run in time $O(2^{\ell/2})$ with constant memory cost, Shanks's algorithm [Sha71] runs in time $O(2^{\ell/2})$ using a table of size $O(2^{\ell/2})$, and Bernstein-Lange algorithm [BL12] runs in time $O(2^{\ell/3})$ using a table of size $O(2^{\ell/3})$.

In this work, we choose to implement the Shanks's algorithm, since it admits flexible time/space trade-off and is amenable to parallellization. When $\ell = 32$, the average decryption time is approximately 1ms by using a hash map of size 264MB. This demonstrates that Shanks's algorithm is more preferable in practice for small range. If larger range are needed, say $[0, 2^{64})$, the sender can represent a 64-bit value v by a concatenation of two 32-bit values $v_1 || v_0$, then encrypts v_1 and v_0 independently under receiver's public key, i.e., replacing $C_r = \mathsf{Enc}(pk_r, v)$ with $C_{r,1} = \mathsf{Enc}(pk_r, v_1)$ and $C_{r,0} = \mathsf{Enc}(pk_r, v_0)$.[9] In this way, time complexity of decryption only grows linearly in the size of bit-length rather than exponentially.

7.2 More Efficient Assembly of NIZK

Towards a modular design of NIZK for L_{correct}, we break this task to designing NIZK proofs for L_{equal}, L_{correct} and L_{solvent} separately. Particularly, to build NIZK for L_{right} (asserting a twisted ElGamal ciphertext encrypts a message in expected range), we compose a Sigma protocol Σ_{enc} for L_{enc} and a Bulletproof Λ_{bullet} for L_{range} sequentially, then apply the Fiat-Shamir transform. The resulting Π_{right} for L_{right} can be used as a sub-protocol when proving membership in L_{solvent}.

When comes to instantiation, we can assemble the sub-protocols in a more efficient manner. Recall that Π_{equal} is a NIZKoK for L_{equal}, $\Pi_{\text{enc}} \circ \Lambda_{\text{bullet}}$ is a NIZKPoK for L_{right}, and $\Pi_{\text{enc}} \circ \Pi_{\text{ddh}} \circ \Lambda_{\text{bullet}}$ is a NIZKPoK for L_{solvent}. The first observation is that the first Π_{enc} can be removed since Π_{equal} already proves knowledge of C_s. Moreover, benefit from the nice feature of twisted ElGamal, two Bulletproofs can be generated and verified in aggregated mode, rather than being invoked twice independently. This trick roughly reduces the size of range proof part by half.

7.3 Eliminate Explicit Signature

At a high level, our DCP construction utilizes signature to provide authenticity and uses NIZK to provide soundness. The celebrated Fiat-Shamir transform [FS86] squashes an interactive public-coin proof into a non-interactive one by setting the verifier's challenges as the hash of the prover's history messages. Particularly, the Fiat-Shamir transform can convert a three round public-coin proof of knowledge into a signature scheme [AABN02]. To generate a signature of message m, the signer runs the PoK itself (with sk as the witness) by setting the challenge e as $\mathsf{H}(I, m)$, where H is a hash function modeled as a random oracle, I is the prover's first round message. The signature is the proof, i.e., the entire transcript.

Based on this connection between signature and ZKPs, Bünz et al. [BAZB20] suggested that one can leverage ZKPs to provide signature functionality instead of employing a separate signature scheme.

[9] We remark that extra zero-knowledge proofs are needed to ensure that the values are encrypted in a correct form. Similar idea was mentioned in Zether [BAZB20] without details.

However, their construction signs both the memo part and the NIZK proof part (cf. specification of $\mathsf{CreateTransferTx}$ (line 9) in Figure 3 [BAZB20]). Thus, the signing operation and signature are still explicit.

In PGC, when building NIZK for L_{solvent} we use Σ_{ddh} for L_{ddh} as a sub-protocol, which is exactly a proof of knowledge of sender's secret key sk_s. As we highlighted in the security model of DCP, it suffices to sign only the core information, say, $(\mathsf{sn}, \mathsf{memo})$. Benefit from this relaxation, the signature component can be subsumed into the NIZK proof. More precisely, by appending $(\mathsf{sn}, \mathsf{memo})$ to the prover's first round message in Σ_{ddh}, the obtained zero-knowledge proof π_{ddh} for L_{ddh} also acts as sender's signature of $(\mathsf{sn}, \mathsf{memo})$. In this way, PGC obtains the signature functionality for free.[10]

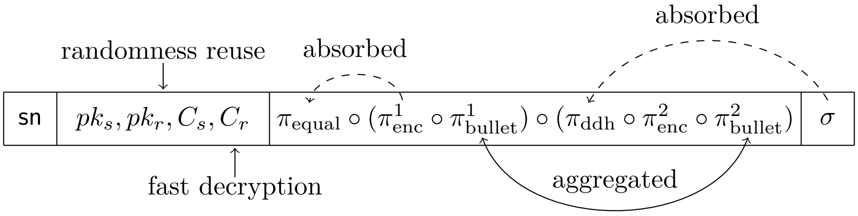
absorbed
randomness reuse
absorbed
sn
pk_s, pk_r, C_s, C_r
$\pi_{\text{equal}} \circ (\pi^1_{\text{enc}} \circ \pi^1_{\text{bullet}}) \circ (\pi_{\text{ddh}} \circ \pi^2_{\text{enc}} \circ \pi^2_{\text{bullet}})$
σ
fast decryption
aggregated

Figure 6: CTx structure of PGC and optimizations.

8 Performance

8.1 Benchmark of PGC

We first give a prototype implementation of PGC as a standalone cryptocurrency in C++ based on OpenSSL, and collect the benchmarks on a MacBook Pro with an Intel i7-4870HQ CPU (2.5GHz) and 16GB of RAM. The source code of PGC is publicly available at Github [libb]. For demo purpose only, we only focus on the transaction layer and do not explore any optimizations.[11] The experimental results are described in the table below.

Unfortunately, we cannot find implementation of any existing cryptocurrencies in the account-based model with transaction confidentiality and auditing capability for a concrete comparison. As a closely related work, Zether [BAZB20] provides transaction confidentiality but there is no reported implementation as a standalone cryptocurrency. We note that Zether can be instantiated on Ethereum smart contract platform. Thus, we also consider PGC on top of the Ethereum smart contract platform for comparison with Zether.

[10] A subtlety here is that our approach does not enable us to sign the whole information of $(\mathsf{sn}, \mathsf{memo}, \pi_{\text{correct}})$, since there seems no way to append π_{correct} (containing π_{ddh}) to the first round message of Σ_{ddh} before generating it. Nevertheless, as we discussed in Section 3.4, signing $(\mathsf{sn}, \mathsf{memo})$ is sufficient for authenticity.

[11] We expect at least 2× speedup after optimizations.

Table 1: The computation and communication complexity of PGC.

PGC	ctx size		transaction cost (ms)	
	big-$\mathcal{O}$	bytes	generation	verify
confidential transaction	$(2\log_2(\ell)+20)\lvert\mathbb{G}\rvert + 10\lvert\mathbb{Z}_p\rvert$	1310	40	14
auditing policies	proof size		auditing cost (ms)	
	big-$\mathcal{O}$	bytes	generation	verify
limit policy	$(2\log_2(\ell)+4)\lvert\mathbb{G}\rvert + 5\lvert\mathbb{Z}_p\rvert$	622	21.5	7.5
rate policy	$2\lvert\mathbb{G}\rvert + 1\lvert\mathbb{Z}_p\rvert$	98	0.55	0.69
open policy	$2\lvert\mathbb{G}\rvert + 1\lvert\mathbb{Z}_p\rvert$	98	0.26	0.42

Here we set the maximum number of coins as $v_{\max} = 2^\ell - 1$, where $\ell = 32$. PGC operates elliptic curve prime256v1 [Ope], which has 128 bit security. The elliptic points are expressed in their compressed form. Each $\mathbb{G}$ element can be stored as 33 bytes, and $\mathbb{Z}_p$ element can be stored as 32 bytes. PGC also supports other elliptic curves. The experimental result of auditing cost for limit policy is obtained by considering two transactions. We emphasize that the proof size is independent of the number of transactions, while the efficiency of proof generation and verification will not degrade much as the number of transactions grows, since homomorphic add of twisted ElGamal is pretty fast.

Similar to Zether, PGC is also compatible with Ethereum and other smart contract platforms. In order to properly compare PGC mechanism with Zether mechanism [BAZB20], we further deploy PGC as an Ethereum smart contract. The experimental results and comparison to Zether [BAZB20] are described in the table below.

Table 2: PGC vs. Zether as smart contracts.

scheme	operation	gas Cost	EC Cost	transaction size (bytes)
Zether	Burn	384k	329k	320
	Fund	260k	41k	64
	Transfer	7188k	6455k	1472
PGC	Burn	310k	161k	288
	Fund	172k	42k	64
	Transfer	8282k	6695k	2240

We measure the total gas cost which includes the basic cost for sending a transaction, the storage cost, the proof/signature verification, and transaction size. In order to be consistent with Zether's result, we do not include the basic Ethereum transaction data.

The comparison results demonstrate that the performance of PGC is similar to Zether. Besides, PGC can bring confidential payment ability to ETH and other ERC-20 tokens on Ethereum, while Zether does not support ERC-20 tokens.

8.2 Benchmark of Twisted ElGamal

Twisted ElGamal is an important building block of PGC. It is additively homomorphic over $\mathbb{Z}_p$, and enjoys nice interoperability with signature and zero-knowledge proofs. Moreover, its practical efficiency is competitive to other homomorphic PKE schemes based on number-theoretic assumptions, for example, the Paillier PKE [Pai99]. These features makes twisted ElGamal extremely useful in numerous privacy-preserving scenarios beyond cryptocurrencies.

Efficiency. We provide the benchmark of twisted ElGamal, and compare it with Paillier PKE implemented in open-source libpaillier [liba]. The experimental results are collected from the same platform as PGC.

Table 3: Benchmarks of twisted ElGamal and Paillier PKE (average of 1000 runs)

timing (ms)	Setup	KeyGen	Enc	Dec	ReRand	HomoAdd	HomoSub	Scalar
Paillier	—	1644.53	32.211	31.367	—	0.0128	—	—
twisted ElGamal	50s+6s	0.0151	0.114	1	0.157	0.0031	0.0042	0.093

size (bytes)	public parameters	public key	secret key	ciphertext
Paillier	—	384	384	768
twisted ElGamal	66	33	32	66

For a fair comparison, we choose 32 bits message space and 128 bit security level [key] (twisted ElGamal operates on elliptic curve prime256v1 and Paillier operates on 3072 bit modulus). In twisted ElGamal, the setup algorithm runs "one-and-for-all", it first generates and serializes a key-value hash map (roughly 264MB) to accelerate decryption (this step takes roughly 50s), then loads it to the memory (this step takes roughly 6s). The hash map could be included as a part of public parameters, which can be safely shared among users. The decryption time decreases linearly as the hash map size grows. In Paillier PKE, global setup is not supported, and libpaillier does not provide APIs for re-randomization, homomorphic subtraction and scalar operations.

ZKP friendliness. Next, we demonstrate the advantages of twisted ElGamal over standard ElGamal in terms of zero-knowledge proof friendliness. We compare the efficiency of two useful gadgets built from standard ElGamal and twisted ElGamal respectively. Gadget-1 allows a sender to prove the message m encrypted under $C = \mathsf{Enc}(pk, m; r)$ lies in range $[0, 2^\ell)$. In this setting, the prover knows m and r. Gadget-2 allows a receiver to prove the message m encrypted under $C = \mathsf{Enc}(pk, m)$ lies in range $[0, 2^\ell)$. In this setting, the prover knows sk and thus is able to learn m by decrypting C.

In either cases, Gadget-2 can be built from Gadget-1 by re-randomizing the ciphertext and appending a NIZK for consistency of re-randomization. Note that twisted ElGamal and its accompanying Sigma protocols are as efficient as their standard version, so the efficiency difference of gadgets lies in the Bulletproof's bridging cost. To see the comparison results more clearly, we fix baselines of two gadgets and only focus on the invoking cost. We set the baseline of Gadget-1 as a (twisted) ElGamal ciphertext, an NIZPoK for plaintext and randomness knowledge, as well as a Bulletproof. The baseline of Gadget-2 additionally includes a re-randomized (twisted) ElGamal ciphertext and a NIZK for consistency of re-randomization. When implementing Gadget-1/2 from standard ElGamal, the Bulletproof's bridging cost includes generating a Pedersen commitment $h^r g^m$ and a NIZKPoK for its consistency to the ElGamal ciphertext $(g^r, pk^r g^m)$. When implementing Gadget-1/2 from twisted ElGamal, the Bulletproof's bridging cost is zero.

Table 4: Bulletproof's bridging costs of standard ElGamal and twisted ElGamal.

bridging cost	building block	size	generation	verify
Gadgets-1/2	standard ElGamal	$n(2\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p\rvert)$	n(2Exp)	n(2Exp+1Add)
	twisted ElGamal	0	0	0

n is the number of ciphertexts that need to be processed. In many of the range proof applications like secure machine learning, a single prover needs to conduct range proofs via gadgets-1/2 for million of ciphertexts. In this case, the saving is significant.

Very recently, exponential ElGamal and Paillier are accepted as ISO standard [HEs]. Based on better ZKP friendliness over exponential ElGamal and better efficiency over Paillier, we believe twisted ElGamal has the potential to be a candidate of standardized homomorphic PKE in the future.

Acknowledgments

We thank Benny Pinkas and Jonathan Bootle for clarifications on Sigma protocols and Bulletproofs in the early stages of this research. We thank Yi Deng and Shunli Ma for helpful comments. We particularly thank Shuai Han for many enlightening discussions.

References

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the fiat-shamir transform: Minimizing assumptions for security and forward-security. In *Advances in Cryptology - EUROCRYPT 2002*, pages 418–433, 2002.

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *Financial Cryptography and Data Security - FC 2020*, 2020. https://eprint.iacr.org/2019/191.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018*, pages 315–334, 2018.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, 2000.

[BCC$^+$16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology - EUROCRYPT 2016*, pages 327–357, 2016.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology - ASIACRYPT 2018*, pages 435–464, 2018.

[BDZ03] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of diffie-hellman problem. In *Information and Communications Security, 5th International Conference, ICICS 2003*, pages 301–312, 2003.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC 1988*, pages 103–112, 1988.

[BKP14] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in bitcoin P2P network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS 2014*, pages 15–29, 2014.

[BL12] Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In *Progress in Cryptology - INDOCRYPT 2012*, pages 317–338, 2012.

[BN06] 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, CCS 2006*, pages 390–399. ACM, 2006.

[BNM$^+$14] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security - 18th International Conference, FC 2014*, pages 486–504, 2014.

[BPS00] Olivier Baudron, David Pointcheval, and Jacques Stern. Extended notions of security for multicast public key cryptosystems. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000*, pages 499–511, 2000.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT 1997*, pages 103–118, 1997.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Advances in Cryptology - CRYPTO 1992*, pages 89–105, 1992.

[CZ14] Yu Chen and Zongyang Zhang. Publicly evaluable pseudorandom functions and their applications. In *9th International Conference on Security and Cryptography for Networks, SCN 2014*, pages 115–134, 2014.

[Dam] Ivan Damgård. On sigma protocols. http://www.cs.au.dk/~ivan/Sigma.pdf.

[Das] Dash. https://www.dash.org.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the nonmalleability of the fiat-shamir transform. In *Progress in Cryptology - INDOCRYPT 2012*, pages 60–79, 2012.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, STOC 1990*, pages 308–317, 1990.

[FMMO19] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In *Advances in Cryptology - ASIACRYPT 2019*, pages 649–678, 2019.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO 1986*, pages 186–194, 1986.

[GGM16] Christina Garman, Matthew Green, and Ian Miers. Accountable privacy for decentralized anonymous payments. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016*, pages 81–98, 2016.

[Gol06] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, New York, NY, USA, 2006.

[GR10] Steven D. Galbraith and Raminder S. Ruprai. Using equivalence classes to accelerate solving the discrete logarithm problem in a short interval. In *13th International Conference on Practice and Theory in Public Key Cryptography, PKC 2010*, pages 368–383, 2010.

[Gri] Grin. https://grin-tech.org/.

[HAB$^+$17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017*, 2017.

[HEs] Homomorphic standard. ISO/IEC 18033-6:2019.

[HP01] Stuart Haber and Benny Pinkas. Securely combining public-key cryptosystems. In *Proceedings of the 8th ACM Conference on Computer and Communications Security, CCS 2001*, pages 215–224. ACM, 2001.

[key] https://www.keylength.com/.

[Kur02] Kaoru Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In *Public Key Cryptography - PKC 2002*, pages 48–63, 2002.

[liba] libPaillier. https://github.com/snipsco/paillier-libraries-benchmarks.

[libb] libPGC. https://github.com/yuchen1024/libPGC.

[Max] Greg Maxwell. Confidential transactions.

[Max13] Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world, 2013. https://people.xiph.org/~greg/confidential_values.txt.

[MM] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *PoPETs*, 2018(2):105–121.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

[Noe15] Shen Noether. Ring signature confidential transactions for monero, 2015. https://eprint.iacr.org/2015/1098.

[NVV18] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018*, pages 65–80, 2018.

[Ope] https://github.com/openssl/openssl/blob/4e6647506331fc3b3ef5b23e5dbe188279ddd575/include/openssl/obj_mac.h.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT 1999*, pages 223–238, 1999.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO 1991*, pages 129–140, 1991.

[Poe] Andrew Poelstra. Mimblewimble. https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

[Pol78] John M. Pollard. Monte carlo methods for index computation (mod p). *Mathematics of Computation*, 32, 1978.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.

[PSST11] Kenneth G. Paterson, Jacob C. N. Schuldt, Martijn Stam, and Susan Thomson. On the joint security of encryption and signature, revisited. In *Advances in Cryptology - ASIACRYPT 2011*, pages 161–178, 2011.

[RMK14] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *19th European Symposium on Research in Computer Security, ESORICS 2014*, pages 345–364, 2014.

[RS13] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, pages 6–24, 2013.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[Sha71] D. Shanks. Class number, a theory of factorization and genera. *Proc. Symposium in Pure Mathematics*, 20, 1971.

[TMZC17] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In *22nd European Symposium on Research in Computer Security, ESORICS 2017*, pages 494–512, 2017.

[Woo14] Gavin Wood. Ethereum: A secure decentralized transaction ledger. http://gavwood.com/paper.pdf, 2014. https://www.ethereum.org/.

[ZCa] Zcash: Privacy-protecting digital currency. https://z.cash/.

A Missing Proofs

Theorem A.1. *Twisted ElGamal is IND-CPA secure based on the divisible DDH assumption.*

Proof. We proceed via two games. Let S_i be the probability that $\mathcal{A}$ wins in Game i.

Game 0. The real IND-CPA security experiment. Challenger $\mathcal{CH}$ interacts with $\mathcal{A}$ as below:

1. Setup: $\mathcal{CH}$ generates pp and keypair according to the definition, then sends pp and $pk = g^{sk}$ to $\mathcal{A}$.

2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and randomness r, computes $X = pk^r$, $Y = g^r h^{m_\beta}$, sends $C = (X, Y)$ to $\mathcal{A}$.

3. Guess: $\mathcal{A}$ outputs its guess β' for β and wins if $\beta' = \beta$.

According to the definition of Game 0, we have:

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) = \Pr[S_0] - 1/2$$

Game 1. Same as Game 0 except $\mathcal{CH}$ generates the challenge ciphertext in a different way.

3. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and two independent randomness r and s, computes $X = pk^r$, $Y = g^s h^{m_\beta}$, sends $C = (X, Y)$ to $\mathcal{A}$.

In Game 1, the distribution of C is independent of β, thus we have:

$$\Pr[S_1] = 1/2$$

It remains to prove $\Pr[S_1]$ and $\Pr[S_0]$ are negligibly close. We prove this by showing if not so, one can build an adversary $\mathcal{B}$ breaks the divisible DDH assumption with the same advantage. Given (g, g^a, g^b, g^c), $\mathcal{B}$ is asked to decide if it is a divisible DDH tuple or a random tuple. To do so, $\mathcal{B}$ interacts with $\mathcal{A}$ by simulating $\mathcal{A}$'s challenger in the following IND-CPA experiment.

1. Setup: $\mathcal{B}$ sets $pp = (\mathbb{G}, g, h, p)$ as public parameters, sets g^b as the public key, where b is interpreted as the corresponding secret key $b \in \mathbb{Z}_p$ and unknown to $\mathcal{B}$. $\mathcal{B}$ sends pp and $pk = g^b$ to $\mathcal{A}$.

2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{B}$. $\mathcal{B}$ picks a random bit β and sets $X = g^a$, $Y = g^c h^{m_\beta}$, sends $C = (X, Y)$ to $\mathcal{A}$.

3. Guess: $\mathcal{A}$ outputs a guess β'. $\mathcal{B}$ outputs "1" if $\beta' = \beta$ and "0" otherwise.

If (g, g^a, g^b, g^c) is a divisible DDH tuple, $\mathcal{B}$ simulates Game 0 perfectly (with randomness $c = a/b$). Else, $\mathcal{B}$ simulates Game 1 perfectly (with two independent randomness a/b and c). Thereby, we have $\mathsf{Adv}_{\mathcal{B}} = |\Pr[S_0] - \Pr[S_1]|$, which is negligible in λ by the divisible DDH assumption.

Putting all the above together, Theorem A.1 follows. □

Remark A.1. We stress that twisted ElGamal does not require trusted setup. It remains secure even the adversary knows the discrete relation between (g, h).