PGC: Decentralized Confidential Payment System with Auditability

Yu Chen[1,2,3,4], Xuecheng Ma[5,6], Cong Tang[7], and Man Ho Au[8(✉)]

[1] School of Cyber Science and Technology,
Shandong University, Qingdao 266237, China
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] Key Laboratory of Cryptologic Technology and Information Security,
Ministry of Education, Shandong University, Qingdao 266237, China
[4] Shandong Institute of Blockchain, Jinan, China
yuchen@sdu.edu.cn
[5] State Key Laboratory of Information Security,
Institute of Information Engineering,
Chinese Academy of Sciences, Beijing 100093, China
[6] School of Cyber Security, University of Chinese Academy of Sciences,
Beijing 100049, China
maxuecheng@iie.ac.cn
[7] Beijing Temi Co., Ltd., pgc.info, Beijing, China
congtang.cn@gmail.com
[8] Department of Computer Science,
The University of Hong Kong, Pok Fu Lam, China
allenau@cs.hku.hk

Abstract. Many existing cryptocurrencies fail to provide transaction anonymity and confidentiality. As the privacy concerns grow, a number of works 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 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 1.3 KB and take under 38ms to generate and 15 ms to verify.

At the core of PGC is an additively homomorphic public-key encryption scheme that we newly introduce, twisted ElGamal, which is not only as secure as standard exponential ElGamal, but also friendly to

L. Chen et al. (Eds.): ESORICS 2020, LNCS 12308, pp. 591–610, 2020.
https://doi.org/10.1007/978-3-030-58951-6_29

Sigma protocols and Bulletproofs. 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.

Keywords: Cryptocurrencies · Decentralized payment system · Confidential transactions · Auditable · Twisted ElGamal

1 Introduction

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, 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 that 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 employee did not receive the correct amount, how can he support his complaint? Likewise, how can the employer protect himself by demonstrating that 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

The above discussions suggest that it might 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 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 Sect. 5.

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).

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, 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 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 either 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 [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

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

approach 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 $r \xleftarrow{R} \mathbb{Z}_p$, 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] Twisted ElGamal retains additive homomorphism, and is as efficient and secure as the standard exponential ElGamal. 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. 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.

Prover Knows the Secret Key. 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. 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. 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.

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

[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.

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.

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. In Sect. 4.2, we show how to enforce limit, rate and open policies using Sigma protocols and the two gadgets we have developed.

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, CoinShuffle [RMK14], 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 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, and the scale of participants is fixed at the setup stage of system.

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.

Note. Due to space limit, we refer to the full version of this work [CMTA19] for definitions of standard cryptographic primitives and all security proofs.

2 Definition of DCP System

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

2.1 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 application-dependent auxiliary information.

2.2 Decentralized Confidential Payment System with Auditability

An auditable DCP system consists of the following polynomial-time algorithms:

- $\mathsf{Setup}(\lambda)$: 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 $\tilde{m}$. 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 on the blockchain B, 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 $f(pk, \{\mathsf{ctx}\}) = 1$.

2.3 Correctness and Security Model

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.

As to security, 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). See the full version [CMTA19] for the details of security model.

3 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 $= (\mathsf{Setup}, \mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$[3] be a NIZK proof system for L_{correct} (which will be specified later). The construction is as below.

- $\mathsf{Setup}(1^\lambda)$: 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 $\text{ISE.}\mathsf{KeyGen}(pp_{\text{ise}})$ 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[4], outputs public key pk and secret key sk. Fix the public parameters, the KeyGen algorithm naturally induces an $\mathcal{NP}$ relation $\mathsf{R}_{\text{key}} = \{(pk, sk) : \exists r \text{ s.t. } (pk, sk) = \mathsf{KeyGen}(r)\}$.

[3] 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.

[4] 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.

- $\mathsf{RevealBalance}(sk, \tilde{C})$: on input secret key sk and encrypted balance $\tilde{C}$, outputs $\tilde{m} \leftarrow \mathrm{ISE.Dec}(sk, \tilde{C})$.
- $\mathsf{CreateCTx}(sk_s, pk_s, v, pk_r)$: 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 ctx via the following steps:
  1. compute $C_s \leftarrow \mathrm{ISE.Enc}(pk_s, v; r_1)$, $C_r \leftarrow \mathrm{ISE.Enc}(pk_r, v; r_2)$, 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 proof π_{correct} for $\mathsf{memo} = (pk_s, pk_r, C_s, C_r) \in L_{\text{correct}}$, where L_{correct} is defined as:

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

     L_{correct} can be decomposed as $L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{solvent}}$, where L_{equal} proves the consistency of two ciphertexts, L_{right} proves that the transfer amount lies in the right range, and L_{solvent} proves that the sender account is solvent.
  3. run $\sigma \leftarrow \mathrm{ISE.Sign}(sk_s, (\mathsf{sn}, \mathsf{memo}))$, sn is the serial number of pk_s;
  4. output $\mathsf{ctx} = (\mathsf{sn}, \mathsf{memo}, \mathsf{aux})$, where $\mathsf{aux} = (\pi_{\text{correct}}, \sigma)$.
- $\mathsf{VerifyCTx}(\mathsf{ctx})$: parses $\mathsf{ctx} = (\mathsf{sn}, \mathsf{memo}, \mathsf{aux})$, $\mathsf{memo} = (pk_s, pk_r, C_s, C_r)$, $\mathsf{aux} = (\pi_{\text{correct}}, \sigma)$, then checks its validity via the following steps:
  1. check if sn is a fresh serial number of pk_s;
  2. check if $\mathrm{ISE.Verify}(pk_s, (\mathsf{sn}, \mathsf{memo}), \sigma) = 1$;
  3. check if $\mathrm{NIZK.Verify}(crs, \mathsf{memo}, \pi_{\text{correct}}) = 1$.

  If all the above tests pass, outputs "1", miners confirm that ctx is valid and record it on the blockchain, 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.

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}, 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.Prove with witness sk_s to generate π_{limit} for $(pk_s, \{C_{s,i}\}_{1 \le i \le n}, a_{\max}) \in L_{\text{limit}}$:

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

  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 $\mathsf{ctx}_i = (\mathsf{sn}_i, \mathsf{memo}_i = (pk_s, pk_{r_i}, C_{s,i}, C_{r_i}), \mathsf{aux}_i)$, then outputs $\mathrm{NIZK.Verify}(crs, (pk_s, \{C_{s,i}\}_{1 \le i \le n}, a_{\max}), \pi_{\text{limit}})$. 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.Prove with witness sk_u to generate π_{rate} for the statement $(pk_u, C_{u,1}, C_{u,2}, \rho) \in L_{\text{rate}}$:

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

  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}, 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.Prove with witness sk_u (where the subscript u could be either s or r) to generate π_{open} for $(pk_u, C_u, v^*) \in L_{\text{open}}$:

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

  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}, parses $\mathsf{ctx} = (\mathsf{sn}, pk_s, pk_r, C_s, C_r, \mathsf{aux})$, 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.

4 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.

4.1 Instantiating ISE

We instantiate ISE from our newly introduced twisted ElGamal PKE and the classical Schnorr signature.

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 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.

Correctness and additive homomorphism are obvious. The standard IND-CPA security can be proved in standard model based on the DDH assumption.

Theorem 1. *Twisted ElGamal is IND-CPA secure in the 1-plaintext/2-recipient setting based on the DDH assumption.*

ISE from Schnorr Signature and Twisted ElGamal Encryption. We choose Schnorr signature [Sch91] as the signature component. 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 2. *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.*

4.2 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 Sect. 3, 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.[5] 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.

NIZK for L_{equal}. Recall that L_{equal} is defined as:

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

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:

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

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:

$$pk_1^{z_1} = A_1 X_1^e \wedge pk_2^{z_1} = A_2 X_2^e \wedge g^{z_1} h^{z_2} = BY^e$$

Lemma 1. *Σ_{equal} is a public-coin SHVZK proof of knowledge for L_{equal}.*

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

NIZK for L_{right}. Recall that L_{right} is defined 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 $a, b \xleftarrow{R} \mathbb{Z}_p$, sends $A = pk^a$ and $B = g^a h^b$ 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 two equations hold simultaneously:

$$pk^{z_1} = AX^e \wedge g^{z_1} h^{z_2} = BY^e$$

Lemma 2. *Σ_{enc} is a public-coin SHVZK proof of knowledge for L_{enc}.*

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

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

NIZK for L_{Solvent}. Recall that L_{solvent} is defined as:

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

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:

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

We note that 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 prover unable to directly invoke the Bulletproof on instance Y'.

Our trick is encrypting $m' = (\tilde{m} - v)$ 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'. Thus, we can express L_{solvent} as $L_{\text{equal}} \wedge L_{\text{right}}$, i.e., $(pk, C') \in L_{\text{solvent}} \iff (pk, C', C^*) \in L_{\text{equal}} \wedge (pk, C^*) \in L_{\text{right}}$.

To prove C' and C^* encrypting the same value under public key pk, we cannot simply use a Sigma protocol like Protocol for L_{equal}, 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:

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

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 encrypted by 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].

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

We then show the designs of NIZK for typical auditing policies.

NIZK for L_{limit}. Recall that $L_{\text{limit}} = \{pk, \{C_i\}_{1 \le i \le n}, a_{\max}\}$ is defined as:

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

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.

NIZK for L_{rate}. Recall that L_{rate} is defined as:

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

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 $h^{\beta v_1} = h^{\alpha v_2}$,[6] 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 for L_{solvent}.

NIZK for L_{open}. Recall that L_{open} is defined 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.

5 Performance

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 (2.5 GHz) and 16 GB of RAM. The source code of PGC is publicly available at Github [lib]. For demo purpose only, we only focus on the transaction layer and do not explore any optimizations.[7] The experimental results are described in the table below.

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

[7] 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, which has 128 bit security. The elliptic points are expressed in their compressed form. Each $\mathbb{G}$ element is stored as 33 bytes, and $\mathbb{Z}_p$ element is stored as 32 bytes.

Acknowledgments. We thank Benny Pinkas and Jonathan Bootle for clarifications on Sigma protocols and Bulletproofs in the early stages of this research. We particularly thank Shuai Han for many enlightening discussions. Yu Chen is supported by National Natural Science Foundation of China (Grant No. 61772522, No. 61932019). Man Ho Au is supported by National Natural Science Foundation of China (Grant No. 61972332).

References

[BAZB20] Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_23

[BBB+18] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, pp. 315–334 (2018)

[BKP14] Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in bitcoin P2P network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS 2014, pp. 15–29 (2014)

[BNM+14] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A., Felten, E.W.: Mixcoin: anonymity for bitcoin with accountable mixes. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 486–504. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_31

[CGS97] Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_9

[CMTA19] Chen, Y., Ma, X., Tang, C., Au, M.H.: PGC: pretty good confidential transaction system with auditability. Cryptology ePrint Archive, Report 2019/319 (2019). https://eprint.iacr.org/2019/319

[CP92] Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 89–105. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_7

[CZ14] Chen, Yu., Zhang, Z.: Publicly evaluable pseudorandom functions and their applications. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 115–134. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10879-7_8

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

[FMMO19] Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: a new design for anonymous cryptocurrencies. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 649–678. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_23

[GGM16] Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54970-4_5

[Gol06] Goldreich, O.: Foundations of Cryptography, vol. 1. Cambridge University Press, New York (2006)

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

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

[Max] Maxwell, G.: Confidential transactions (2016). https://people.xiph.org/~greg/confidential_values.txt

[MM] Meiklejohn, S., Mercer, R.: Möbius: trustless tumbling for transaction privacy. PoPETs **2**, 105–121 (2018)

[Nak08] Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf

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

[NVV18] Narula, N., Vasquez, W., Virza, M.: zkLedger: privacy-preserving auditing for distributed ledgers. In: 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, pp. 65–80 (2018)

[Ped91] Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9

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

[PSST11] Paterson, K.G., Schuldt, J.C.N., Stam, M., Thomson, S.: On the joint security of encryption and signature, revisited. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 161–178. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_9

[RMK14] Ruffing, T., Moreno-Sanchez, P., Kate, A.: CoinShuffle: practical decentralized coin mixing for bitcoin. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 345–364. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11212-1_20

[RS13] Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 6–24. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_2

[Sch91] Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). https://doi.org/10.1007/BF00196725

[Woo14] Wood, G.: Ethereum: a secure decentralized transaction ledger (2014). http://gavwood.com/paper.pdf, https://www.ethereum.org/

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