PGC: Pretty Good Confidential Transaction System with Accountability

Yu Chen * Xuecheng Ma †

Abstract

Due to the public visible nature of blockchain, the seminal cryptocurrencies such as Bitcoin and Ethereum do not provide sufficient level of privacy, i.e., the addresses of sender and receiver and the transfer amount are all stored in plaintexts on blockchain. As the privacy concerns grow, several newly emerged cryptocurrencies such as Monero and ZCash provide strong privacy guarantees (including anonymity and confidentiality) by leveraging cryptographic technique.

Despite strong privacy is promising, it might be overkilled or even could be abused in some cases. In particular, anonymity seems contradict to accountability, which is a crucial property for scenarios requiring disputes resolving mechanism, such as e-commerce.

To address the above issues, we introduce accountability to blockchain-based confidential transaction system for the first time. We first formalize a general framework of confidential transaction system with accountability from digital signature, homomorphic public-key encryption and non-interactive zero-knowledge arguments, then present a surprisingly simple and efficient realization called PGC. To avoid using general-purpose zero-knowledge proofs (such as zk-SNARK and zk-STARK), we twist the ElGamal encryption as the underlying homomorphic PKE and develop ciphertext-refreshing approach. This not only enables us to prove transaction validity/correctness by using efficient Σ protocols and zero-knowledge range proofs, but also makes PGC largely compatible with Bitcoin and Ethereum, which could be used as a drop-in to provide confidential enforcements with accountability.

Keywords: cryptocurrencies, confidential transaction, accountability, twisted ElGamal

*Institute of Information Engineering, Chinese Academy of Sciences. Email: yuchen.prc@gmail.com
†Institute of Information Engineering, Chinese Academy of Sciences. Email: maxuecheng@iie.ac.cn

Contents

1 Introduction

Unlike traditional bank systems and e-cash schemes, modern cryptocurrencies such as Bitcoin [Nak08] and Ethereum [Woo14] establish a decentralized peer-to-peer electronic cash system by maintaining an append-only ledger, known as blockchain. The ledger is global distributed and synchronized by consensus mechanisms. An important property of blockchain is public verifiability, that is, anyone can verify the validity of all transactions on the ledger. Bitcoin attains this property by simply exposing all details public: the addresses of sender and receiver as well as the transfer amount. According to [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 both Bitcoin and Ethereum provide some weak anonymity through the unlinkability of account addresses to real world identities, it lacks confidentiality, which is a serious limitation in privacy-preserving applications.

1.1 Motivation

Strong privacy is a double-edged sword. While confidentiality is arguably the primary concern of privacy for blockchain-based cryptocurrencies, anonymity might be abused or even prohibitive for applications that require accountability, since it somehow provides plausible deniability. Suppose the employer pay salaries to employees via cryptocurrencies with strong privacy (e.g., Zcash [ZCa] or Monero [Noe15]). If the employer did not receive the right salary, how can he justify this and demand justice. Vice versa, how can the employer justify that he indeed paid the right salary to the right employee to resolve dispute.

1.2 Our Contributions

To address the aforementioned limitation, in this work we still concentrate on confidentiality, but trade anonymity for accountability. We present PGC (Pretty Good Confidentiality), a simple and efficient confidential transaction system with accountability. For the sake of simplicity and portability, we take an account-based approach and focus only on the simplistic *transaction layer* of cryptocurrencies. The network-level and consensus-level protocols/attacks are put aside for the time being.

We first have to make a choice over commitment and encryption, which will be used as a core primitive to attain confidentiality for PGC.

Encryption vs. Commitment. Maxwell [Max13] first introduced the notion of *confidential transaction* in the context of UTXO model, with the purpose to provide privacy-enhancement for Bitcoin. In CT, the input and output transfer amounts are hidden in Pedersen commitments [Ped91]. To spend a UTXO a user first generates a zero-knowledge proof for validity of transaction (the sum of the committed inputs is greater than the sum of committed outputs, and all outputs are positive), then provides a signature to ensure the transaction to be approved. Recently, an improved CT called Mimblewimble [Poe] was proposed. The improvement is based on a clever observation that for a valid confidential transaction the difference between outputs and inputs value must be 0, thus the difference between input and output Pedersen commitments and the difference between corresponding randomness form an ECDSA key pair. The sender can thus sign the transaction simply with the difference of randomness. This greatly simplifies the structure of confidential transactions.

Nevertheless, commitment-based approach suffers from several drawbacks. On the first place, users must be stateful. They have to keep track of the randomness and the value of each incoming transaction. Otherwise, they are unable to spend it anymore, due to either unable to generate the zero-knowledge proof (lack of witness) or to generate the signature (lack of signing key). This problem not only renders the overall protocol more complicated (the openings of these commitments must be transferred to Bob through a separate secure channel), but also incurs extra security burden to the design of wallet (the openings must kept in safety). As indicated by Bünz et al. [BAZB19], failure to recover the randomness of a single commitment could make an UTXO isolated and render an account totally unusable. On the second place, they have to record the randomness and value of all incoming and outgoing transactions to achieve accountability. Last but not the least, as pin-pointed by Bünz et al. [BBB+18], since Pedersen commitment is only computational binding based on discrete logarithm assumption, an adversary is able to open a given commitment to an arbitrary value when quantum computers are available.

One may wonder if we can at least solve the first issue by using a computational hiding and perfectly binding commitment, say ElGamal commitment, instead of using Pedersen commitment. We note that

this patch does not help. The reason is that all users in CT share the same commitment instance (this means trapdoor is not available), and thus each user still has to be stateful. Actually, the above disadvantages seem inherent for all commitment-based confidential transaction systems [Max13, Poe].

Observing that PKE can be viewed as a computationally hiding and perfectly binding commitment in which secret key serves as a natural trapdoor to recover message, we can simply address the aforementioned issues by equipping each user with a PKE instance rather than making all users share the same global commitment.

1.3 Overview of PGC

Now, we are ready to give a brief technical overview of PGC, which is built from digital signature, public-key encryption and non-interactive zero-knowledge arguments. In PGC, each account is equipped with a key pair, which could be used for both encryption and signature. The public key is used as the address of the account. The balance of each account is kept as an encryption of the real value under individual public key. Let C_A and C_B be the encrypted balances of Alice and Bob respectively. Now, suppose Alice wants to transfer v coins to Bob. She constructs the transaction via the following steps: (1) encrypt v under her public key pk_A and Bob's public key pk_B respectively to obtain C_{out} and C_{in}; (2) prove the validity of this transaction in a zero-knowledge way: (i) C_{out} and C_{in} are two encryptions of the same message under pk_A and pk_B; (ii) her current balance deducting the amount encrypted in C_{out} is still positive; (3) sign $(C_{\text{out}}, C_{\text{in}})$ with resulting zero-knowledge proof π_{valid} using her secret key. The final transaction is roughly of the form $(pk_A, C_{\text{out}}, pk_B, C_{\text{in}}, \pi_{\text{valid}}, \sigma)$. The validity of this transaction can be publicly verifiable by checking the signature and zero-knowledge proof. If the transaction is valid, it will be posted on the blockchain. Accordingly, Alice's balance (resp. Bob's balance) will be updated as $C_A = C_A - C_{\text{out}}$ (resp. $C_B = C_B + C_{\text{in}}$). Such balance update operation implicitly requires that the underlying PKE scheme satisfies additive homomorphism. In addition, in case of dispute over some transaction recorded on the blockchain, either the sender or the receiver is able to reveal the exact amount of the transaction by simply providing a zero-knowledge proof[1], to a third party without exposing their secret keys.

In summary, signature is used to prove ownership of an account, PKE is used to hide the balance and transferred amount, while zero-knowledge argument is used to prove the validity and correctness of transactions. Though the high-level idea is pretty simple, an efficient instantiation of this framework turns out to be non-trivial. Before identifying the potential technical obstacles and introducing our approaches, it is instructive to list our design disciplines of PGC in mind in advance:

- system does not rely on a trusted setup
- users are stateless, i.e., there is no need to maintain a state – information on demand can always be computed from data on the blockchain
- only uses simple and efficient cryptographic schemes based on well-studied assumptions, desirably compatible with Bitcoin and Ethereum

1.3.1 Public-Key Cryptosystems

To be compatible with Bitcoin and Ethereum, we choose ECDSA as the signature scheme. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order p. The secret key sk is a random element in $\mathbb{Z}_p$, and the public key $pk = g^{sk}$. With the aim to use this key pair for both signature and encryption in mind, ElGamal encryption with message encoded in the exponent [CGS97] is arguably the most simple and nature candidate for PKE. This choices of ECDSA and ElGamal PKE bring us at least three benefits:

- The key pair is identical to that used in Bitcoin and similar to that used in Ethereum[2]. This makes our PGC compatible to Bitcoin and Ethereum to a large extent.

[1] Proof systems with computational soundness are called argument systems. We will use the terms proof and argument interchangeably in this work.

[2] Ethereum also uses the discrete logarithm key pair, but uses the hash of public key rather than public key itself as the account address.

- ElGamal encryption is additively homomorphic. This enables balance update operation. Moreover, users could be stateless – the account state (i.e., balance) can be computed from the data on the blockchain with corresponding secret key on demand. There is no need to keep track of the randomness and messages beneath transactions.
- ElGamal encryption is zero-knowledge protocols friendly. Its algebra structure allows us to prove C_{out} and C_{in} are two encryptions of the same message under two public keys via an efficient Σ-protocol.

1.3.2 Working with Efficient Range Proof

Besides proving C_{out} and C_{in} encrypt the same message, we still have to prove the value v encrypted in C_{out} and value encrypted in $C_A - C_{\text{out}}$ (the current balance deducts v) lie in the right range. The specific tool for this task is zero-knowledge range proof. For both efficiency and security concern, we are going to use Bulletproof [BBB+18], which is efficient and free of trusted setup. Its security is based on discrete logarithm assumption. This makes our whole PGC system based on solely the DDH assumption. Next, we identify the obstacles of making these two range proofs, and how we overcome them.

The first range proof. It turns out that the standard ElGamal encryption cannot work with Bulletproof, because Bulletproof only accepts statements of the form a Pedersen commitment. One may notice that the second component of the ElGamal encryption could be viewed as a Pedersen commitment with public parameter (pk, g), thus it seems that we can feed the Bulletproof with the second component. But, this does not make sense due to the corresponding sk is an obvious trapdoor of such commitment, with which a malicious prover can open the second component to arbitrary value (not necessarily equal the one fixed by the ciphertext) and thus compromises the desired argument of knowledge.

Our idea is *twisting ElGamal* to ensure the second component free of obvious trapdoor. In addition to g, we pick another random generator h. Now, the global public parameter is $(\mathbb{G}, g, h, p)$. We modify the standard ElGamal encryption as follows. The key pair is still of the form $(pk = g^{sk} \in \mathbb{G}, sk \in \mathbb{Z}_p)$. The encryption of v is tweaked as $(X = pk^r, Y = g^r h^v)$. Decryption can be done by computing $Y/X^{sk^{-1}}$. Now, the second component can be viewed as a Pedersen commitment for v under randomness r, where the public parameter is (g, h) and trapdoor is unknown to all users. Thus, we can safely invoke Bulletproof to generate a range proof of v hidden in statement $g^r h^v$. Luckily, the twisted ElGamal encryption is IND-CPA secure under the divisible DDH assumption, which is equivalent of the standard DDH assumption. Besides, it retains the nice algebra structure and thus still admits efficient Σ-protocol for plaintexts equality.

The second range proof. Another technical obstacle emerges when dealing with the second range proof. In Bulletproof, the prover has to use the opening of the commitment as the witness to generate the proof. This is fine when dealing with the first range proof for v encrypted by C_{out} lies in the right range, because the prover knows the corresponding randomness and value. But, it seems difficult to make a range proof for the value encrypted by $C_A - C_{\text{out}}$ via Bulletproof. The problem stems from the stateless feature of PGC – users do not have to keep track of any states including the randomness under all coming and going transactions, thus the randomness beneath $C_A - C_{\text{out}}$ is typically unknown even assuming homomorphism on randomness.[3]

We resolve this problem by developing *ciphertext refreshing* approach. Note that Alice (the prover) knows sk and thus can decrypt $C_A - C_{\text{out}}$, say v'.[4] She can thus generate a new ciphertext C' of v' under fresh randomness r', and proves $C_A - C_{\text{out}}$ and C' are two encryptions of the same message under the same public key. Then, she can invoke Bulletproof on the second component of C' with witness (r', v'). Note that $C_A - C_{\text{out}}$ and C' are encrypted under the same public key, thereby the plaintext equality argument can be reduced to discrete logarithm equality argument, which in turn can be efficiently proved by a Σ-protocol with witness sk. See Protocol **??** for the details.

1.3.3 Achieving Accountability

In our confidential transaction system the transferred amount is encrypted, and thus it seems difficult to achieve accountability. A naive solution is making the user involved in the transaction in dispute

[3] Most encryption schemes are not randomness recovering, e.g., the ElGamal encryption.

[4] In fact, as we will see shortly, ciphertext refreshing could be done efficiently by partial decryption-then-randomization.

give out his secret key. However, this will expose all the transactions related to this user in clear. Our approach again leverages the power of zero-knowledge protocols. Let $C_i = (pk_i^r, g^v h^r)$ be the ciphertext in the disputed transaction. The user with pk_i can account C_i by simply publishing a zero-knowledge proof for C_i is indeed an encryption of v under pk_i. Looking ahead, this argument can proved by the Σ-protocol for discrete logarithm equality. See Protocol **??** for the details.

1.4 Related Work

Maxwell [Max13] first introduced confidential transactions (CT), in which every transaction amount involved is hidden from public view using a commitment. To enable public validation of the blockchain, a zero-knowledge proof of validity (range proof) is appended in every transaction. Mimblewimble [Poe] further improves CT by reducing the cost of signature. To further achieve anonymity, Monero [Noe15] employs linkable ring signature and stealth address, ZCash based on Zerocash [BCG+14] utilizes shielded addresses and general purpose succinct zero-knowledge proofs. Ma et al. [MDH+17] proposed a confidential transaction scheme from the linear encryption [BBS04] and the range proof based on Boneh-Boyen signature [CCS08]. We note that the zero-knowledge proofs used in the aforementioned CT proposals either require a trusted setup (e.g. zk-SNARK [BCG+13, GGPR13] and range proof [CCS08]) or only been asymptotically efficient but practically large (e.g. zk-STARK [BBHR18]).

Concurrent work. Fauzi et al. [FMMO18] proposed a new design for anonymous cryptocurrencies called Quisquis. They mainly employed updatable public keys to achieve anonymity, and used similar cryptographic mechanism to achieve confidentiality.

Bünz et al. [BAZB19] proposed a fully-decentralized, confidential payment system called Zether, which is compatible with Ethereum and other smart contract platforms. In more details, they chosen ElGamal encryption to instantiate the underlying PKE, and developed a custom ZKP named Σ-Bullets to instantiate the ZKPoK.

Compared to their work, we focus solely on building a blockchain-based confidential transaction system, and address the accountability for the first time. Our system is insensitive of account type and network/consensus-level protocols, and thus can be used as a drop-in enhancement to provide confidentiality to many existing cryptocurrencies, such as Bitcoin and Ethereum. From technical aspect, we use the twisted ElGamal as the underlying PKE scheme rather than the standard ElGamal adopted in [FMMO18, BAZB19], and develop ciphertext refreshing approach. This design enables us not only to prove plaintext equality and account transfer amount via efficient Sigma protocol, but also to generate range proofs by directly invoking Bulletproof in a black-box manner. We believe our technique might also benefit the design of Quisquis and Zether by simple adaption.

2 Preliminaries

2.1 Basic Notations

For a set X, we use $x \xleftarrow{R} X$ to denote the operation of sampling x uniformly at random from X, and use $|X|$ to denote its size. We use U_X to denote the uniform distribution over X. For a positive integer d, we use $[d]$ to denote the set $\{1, \ldots, d\}$. We denote $\lambda \in \mathbb{N}$ as the security parameter. We say that a quantity is negligible, written $\mathsf{negl}(\lambda)$, if it vanishes faster than the inverse of any polynomial in λ. 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 indicate that $\mathcal{A}$ outputs z on inputs $(x_1, \ldots, x_n)$ and random coins r. For notational clarity we usually omit r and write $z \leftarrow \mathcal{A}(x_1, \ldots, x_n)$.

Let $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ denote two ensembles of random variables 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 $\mathsf{negl}(\lambda)$.

2.2 Cryptographic Assumptions

Let $\mathsf{GroupGen}$ be a PPT algorithm that takes as input a security parameter 1^λ and outputs a tuple $(\mathbb{G}, g, p)$, where $\mathbb{G}$ is a group of λ-bit prime order p, g is a random generator of $\mathbb{G}$. In what follows, we

review the discrete-logarithm based assumptions w.r.t. $\mathsf{GroupGen}(1^\lambda) \to (\mathbb{G}, g, p)$.

Definition 2.1 (Discrete Logarithm Assumption)**.** The discrete logarithm assumption states that for any PPT adversary it holds that:

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

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

Definition 2.2 (Decisional Diffie-Hellman Assumption)**.** The DDH assumption states that for any PPT adversary it holds that:

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

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

Definition 2.3 (Divisible Decisional Diffie-Hellman Assumption)**.** The divisible DDH assumption states that for any PPT adversary it holds that:

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

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

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

2.3 Zero-Knowledge Protocols

We first recall the general interactive proof systems.

Definition 2.4 (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 parameter pp. Let $\mathsf{R}_{pp} \subseteq X \times W$ be a $\mathcal{NP}$ 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 the public-parameter-dependent $\mathcal{NP}$ language:

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

  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 P and V interacting on input x and y and producing transcript tr. 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 in the interaction, which includes V's input tape, random tape and P's messages.

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 2.5 (Zero-Knowledge Argument of Knowledge)**.** We say an interactive argument 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. Informally, this definition says that whenever a malicious prover produces an acceptable 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:

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

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 2.6 (Sigma Protocols (Σ-protocol))**.** 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 2.7 (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 and sometime is even impossible. Therefore, removing interaction is of particular interest in practice. However, non-interactive zero-knowledge proof systems for non-trivial languages are impossible in the plain model. We need to consider NIZK in the common reference string model, in which the prover and verifier share a common random string (CRS).

Definition 2.8 (Non-Interactive Zero-Knowledge in the CRS model)**.** A non-interactive proof system in the CRS model consists of four PPT algorithms $(\mathsf{Setup}, \mathsf{Gen}, P, V)$:

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

An 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. We can relax statistical soundness to computational soundness by restricting P^* to be a PPT algorithm.

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 $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ and, we have:

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

This definition is also in the adaptive sense in that $\mathcal{A}$ may choose x after seeing crs. Computational zero-knowledge can be strengthened to statistical (resp. perfect) zero-knowledge by requiring the views are statistically indistinguishable (resp. identical).

Interestingly, in the random oracle model NIZK is possible without relying on common reference string. The celebrated Fiat-Shamir transform [FS86] shows how to convert a Sigma-protocol to a NIZK by modeling hash function as a random oracle. Their transform actually applies to any public-coin SHVZK argument of knowledge. Formally, we have the following theorem.

Theorem 2.1 (Fiat-Shamir NIZK [BR93, FKMV12])**.** *Let* (Setup, P, V) *be a* $(2k+1)$*-move public-coin SHVZK argument of knowledge,* a_i *be prover's* i*th round message and* e_i *be verifier's* i*th 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.*

3 General Framework of Accountable CT

We present the general framework of confidential transaction system with accountability from a combined public key scheme $\mathrm{CPK} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Sign}, \mathsf{Verify})$ and an non-interactive zero-knowledge proof system $\mathrm{NIZK} = (\mathsf{Setup}, \mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$.[5]

For the underlying combined public key scheme, we require: (1) its implements key-reuse strategy; (2) its PKE component is homomorphic. Fixed the public parameters, its KeyGen naturally induces a binary $\mathcal{NP}$ relation $\mathsf{R}_{\mathrm{key}} \subseteq PK \times SK$, i.e., $\mathsf{R}_{\mathrm{key}} = \{(pk, sk) : \exists r \text{ s.t. } (pk, sk) = \mathsf{KeyGen}(pp; r)\}$.

We describe the general accountable CTx framework as below.

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

- $\mathsf{Setup}(1^\lambda)$. This algorithm is used to setup the confidential transaction system. It takes as input the security parameter 1^λ, runs $pp_{\text{cpk}} \leftarrow \text{CPK.}\mathsf{Setup}(1^\lambda)$, $pp_{\text{nizk}} \leftarrow \text{NIZK.}\mathsf{Setup}(1^\lambda)$, $crs \leftarrow \text{NIZK.}\mathsf{CRSGen}(pp_{\text{nizk}})$, sets $pp = (pp_{\text{cpk}}, pp_{\text{nizk}}, crs)$ as the global public parameter that will be used by all users algorithm.

- $\mathsf{CreateAcct}(m, sn)$. This algorithm is used to create an account. It takes as input an initial balance $\tilde{m}$ and a serial number sn, runs $\text{CPK.}\mathsf{KeyGen}(pp_{\text{cpk}})$ to generate a keypair (pk, sk), sets $\tilde{C} \leftarrow \text{PKE.}\mathsf{Enc}(pk, \tilde{m}; r)$ as the encryption of initial balance, and initialize the nonce value as sn.[6] Finally, pk is set as the public key (acts as the pseudonym for a user and serves as the account address), sk is set as the secret key.

- $\mathsf{RevealBalance}(sk, \tilde{C})$. This algorithm is used to reveal the balance of an account. It takes as input the secret key sk and the current encrypted balance $\tilde{C}$, outputs $\tilde{m} \leftarrow \text{PKE.}\mathsf{Dec}(sk, \tilde{C})$.

- $\mathsf{CreateCTx}(sk_1, pk_1, v, pk_2)$. This algorithm is used to transfer v coins from account pk_1 to account pk_2. It takes as input the key pair (pk_1, pk_2) of the sender account (also fetches the current nonce sn and encrypted balance $\tilde{C}_1$), the coin value v, the public key pk_2 of the receiver account, builds a confidential transaction ctx via the following steps:

  1. compute $C_1 \leftarrow \text{PKE.}\mathsf{Enc}(pk_1, v; r_1)$, $C_2 \leftarrow \text{PKE.}\mathsf{Enc}(pk_2, v; r_2)$, set the meta information of the transaction as $\mathsf{meta} = (\tilde{C}_1, pk_1, C_1, pk_2, C_2)$;
  2. run NIZK.Prove with witness (sk_1, r_1, r_2, v) to generate a proof π_{valid} for the statement $\mathsf{meta} = (\tilde{C}_1, pk_1, C_1, pk_2, C_2) \in L_{\text{valid}}$, where L_{valid} is defined as

$$\{(\tilde{C}_1, pk_1, C_1, pk_2, C_2) \mid \exists sk_1, r_1, r_2, v \text{ s.t. } C_i = \text{PKE.}\mathsf{Enc}(pk_i, v; r_i)_{i=1,2} \wedge v \in \{0,1\}^\ell \\ (pk_1, sk_1) \in \mathsf{R}_{\text{key}} \wedge \text{PKE.}\mathsf{Dec}(sk_1, \tilde{C}_1 - C_1) \in \{0,1\}^\ell\}$$

     By the enhanced correctness of the underlying PKE component (each valid ciphertext uniquely determines the randomness and plaintext), L_{valid} can be decomposed to $L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{enough}}$:
     - $L_{\text{equal}} = \{(pk_1, C_1, pk_2, C_2) \mid \exists r_1, r_2, v \text{ s.t. } C_i = \text{PKE.}\mathsf{Enc}(pk_i, v; r_i) \text{ for } i = 1, 2\}$
       L_{equal} stipulates that the amount sent by pk_1 equals to that received by pk_2.
     - $L_{\text{right}} = \{(pk_1, C_1) \mid \exists r_1, v \text{ s.t. } C_1 = \text{PKE.}\mathsf{Enc}(pk_1, v; r_1) \wedge v \in \{0,1\}^\ell\}$
       L_{right} stipulates that the outgoing amount sent by pk_1 lies in a right range.
     - $L_{\text{enough}} = \{(pk_1, \tilde{C}_1, C_1) \mid \exists sk_1 \text{ s.t. } (pk_1, sk_1) \in \mathsf{R}_{\text{key}} \wedge \text{PKE.}\mathsf{Dec}(sk_1, \tilde{C}_1 - C_1) \in \{0,1\}^\ell\}$
       L_{enough} stipulates that the balance of pk_1 is enough for making the transfer.
  3. sets $\mathsf{memo} = (sn, \mathsf{meta}, \pi_{\text{valid}})$, runs $\text{SIG.}\mathsf{Sign}(sk_s, \mathsf{memo}) \rightarrow \sigma$, the entire confidential transaction is $\mathsf{ctx} = (\mathsf{memo}, \sigma)$.

- $\mathsf{VerifyCTx}(\mathsf{ctx})$. This algorithm is used to check if a confidential transaction ctx is valid. It takes as input $\mathsf{ctx} = (\mathsf{memo}, \sigma)$, parses memo as $(sn, \mathsf{meta} = (\tilde{C}_1, pk_1, C_1, pk_2, C_2), \pi_{\text{valid}})$, then checks its validity via the following steps:

  1. check if $\text{SIG.}\mathsf{Verify}(pk_1, \mathsf{memo}, \sigma) = 1$;
  2. check if $\text{NIZK.}\mathsf{Verify}(crs, \mathsf{meta}, \pi_{\text{valid}}) = 1$.

  If both checks pass output "1", else output "0". Eventually, if the confidential transaction is valid, the system appends ctx on the blockchain, and updates the balance of pk_1 as $\tilde{C}_1 = \tilde{C}_1 - C_1$ and the balance of pk_r as $\tilde{C}_2 = \tilde{C}_2 + C_2$.

- $\mathsf{TestifyCTx}(sk_i, \mathsf{ctx}, v)$. This algorithm is used to testify the value hidden in ctx is indeed v in a zero-knowledge manner. Here, we assume ctx is a confidential transaction recorded on the ledger. It takes as input sk_i, v and $\mathsf{ctx} = (sn, (\mathsf{meta}, \pi_{\text{valid}}), \sigma)$, where $\mathsf{meta} = (\tilde{C}_1, pk_1, C_1, pk_2, C_2)$, and sk_i is a secret key corresponding to pk_i, then runs NIZK.Prove with witness sk_i to generate a proof π_{correct} for the statement $(C_i, pk_i, v) \in L_{\text{correct}}$. Here, the language L_{correct} is defined as $\{(C, pk, v) \mid \exists sk \text{ s.t. } (pk, sk) \in \mathsf{R}_{\text{key}} \wedge v = \text{PKE.}\mathsf{Dec}(sk, C)\}$. Anyone can check if the value hidden in ctx is indeed the claimed value v by running $\text{NIZK.}\mathsf{Verify}(crs, C_i, pk_i, v, \pi_{\text{correct}})$.

[6] By default, 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 status is publicly auditable. Here, we do not make it as an enforcement for flexibility.

4 Security Model

As a necessary precursor to further development of CTX, we start by exploring security models for CTX.

Intuitively, a CTX system should provide *confidentiality*, *theft prevention* and *soundness*. Confidentiality requires that other than the sender and the receiver, no one can learn the value hidden in a confidential transaction. Theft prevention requires that an adversary is unable to transfer values from accounts for which it does not know the secret key. While the former two notions address security against outsider adversary (who does not know the secret keys of sender and receiver), soundness address security against insider adversary (who does not know the secret key of sender). It requires that an adversary is unable to generate an accepting proof π_{valid} for any invalid transaction meta, even it knows the secret key of sender.

We define the game-based security model for confidential transaction system. The model is defined via adversarial queries to oracles implemented by a challenger $\mathcal{CH}$. $\mathcal{CH}$ keeps track of all honest and corrupt registered accounts by maintaining two lists L_{honest} and L_{corrupt}, which are initially empty. We informally describe the oracles provided to the adversary attacking a confidential transaction system below.

- $\mathcal{O}_{\mathsf{regH}}$: $\mathcal{A}$ queries this oracle for registering an honest account. $\mathcal{CH}$ runs $\mathsf{CreateCTx}$ to generate an account with random balance, sends the public key pk and the encrypted balance C to the adversary. $\mathcal{CH}$ records the tuple (pk, sk, C) in the list L_{honest}. This type of oracle captures $\mathcal{A}$ can observe the public information of honest accounts in the system.
- $\mathcal{O}_{\mathsf{regC}}$: $\mathcal{A}$ queries this oracle with public key pk and an initial encrypted balance C. $\mathcal{CH}$ records the tuple $(pk, \perp, C)$ in the list L_{corrupt}. We stress that the pk and 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. $\mathcal{CH}$ looks for a tuple indexed by pk in L_{honest} and returns sk to $\mathcal{A}$. $\mathcal{CH}$ removes this tuple to L_{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) for making a confidential transaction, subject only to the restriction that $pk_s \in L_{\text{honest}}$ and v is a right value. This restriction is natural because for $pk_s \in L_{\text{corrupt}}$, $\mathcal{A}$ can generate the confidential transaction itself. $\mathcal{CH}$ runs $\mathsf{CreateCTx}(sk_s, pk_s, pk_r, v) \to \mathsf{ctx}$, updates the associated accounts status, then sends ctx to $\mathcal{A}$. $\mathcal{CH}$ records ctx in L_{ctx}. This oracle captures $\mathcal{A}$ can direct honest account to make specific transactions.
- $\mathcal{O}_{\mathsf{inject}}$: $\mathcal{A}$ submits ctx, subject only to the restriction that $pk_s \in L_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{CH}$ updates the associated accounts status and records ctx in L_{ctx}. Otherwise, $\mathcal{CH}$ ignores. This oracle captures $\mathcal{A}$ can generate (possibly malicious) transactions itself.

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

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

To prevent trivial attacks, pk_1, pk_2 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_1 or pk_2.

Theft prevention. We define theft prevention via the following security experiment between $\mathcal{A}$ and $\mathcal{CH}$.

$$\mathrm{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c} pk_s^* \in L_{\text{honest}} \wedge \\ \mathrm{SIG.Verify}(pk_s^*, \mathsf{memo}^*, \sigma^*) = 1 \\ \wedge\ \mathsf{memo}^* \notin L_{\text{trans}}(pk_s^*) \end{array} : \begin{array}{l} pp \leftarrow \mathsf{Setup}(\lambda); \\ (pk_s^*, \mathsf{ctx}^*) \leftarrow \mathcal{A}(pp); \end{array}\right].$$

Here $\mathsf{ctx}^* = (\mathsf{memo}^*, \sigma^*)$ is a confidential transaction from pk_s^*, $L_{\text{trans}}(pk_s^*)$ denotes the set of all the confidential transactions originated from pk_s^* in L_{trans}.

Remark 4.1. Our definition of theft prevention is actually much stronger than its intuition. It stipulates no PPT adversary can generate a valid signature for a fresh $(sn^*, \mathsf{meta}^*, \pi^*_{\text{valid}})$, regardless of whether $\text{NIZK.}\mathsf{Verify}(crs, \mathsf{meta}^*, \pi^*_{\text{valid}}) = 1$. In our confidential transaction system, theft against pk_s is likely to happen only when an adversary (without the knowledge sk_s) creates a valid $\mathsf{ctx}^* = (sn^*, \mathsf{meta}^*, \pi^*_{\text{valid}}, \sigma^*)$ with fresh (sn^*, meta^*), because the two components encode all the meta information of a transaction.

Based on the above analysis, we can weaken the theft prevention definition by only requiring authenticity for (sn, meta). Correspondingly, it suffices to sign only (sn, meta) rather than the entire $\mathsf{memo} = (sn, \mathsf{meta}, \pi_{\text{valid}})$. As we will see in 5.5, this weakening may lead more efficient instantiation.

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

$$\text{Adv}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c|c} \mathsf{meta} \notin L_{\text{valid}} \wedge & pp \leftarrow \mathsf{Setup}(\lambda); \\ \text{NIZK.}\mathsf{Verify}(\mathsf{meta}, \pi_{\text{valid}}) = 1 & (\mathsf{meta}, \pi_{\text{valid}}) \leftarrow \mathcal{A}(pp); \end{array}\right].$$

We then prove our CTx framework is secure based on the security of combined public key scheme and NIZK.

Theorem 4.1. *Assuming the security of combined public key scheme and NIZK, our CTx framework is secure.*

Lemma 4.2. *Assuming the security of the PKE component in combined public key scheme and the zero-knowledge property of NIZK, our CTx framework 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 $\text{CPKS.}\mathsf{Setup}(1^\lambda) \to pp$, $\text{NIZK.}\mathsf{Setup}(1^\lambda) \to crs$, then sends $gpp = (pp, crs)$ to $\mathcal{A}$.
2. Queries: Throughout the experiment, $\mathcal{A}$ can make the following types of queries adaptively. $\mathcal{CH}$ answers these queries by maintaining two lists L_{honest} and L_{corrupt}, which both are initially empty. L_{honest} and L_{corrupt} store entries of the form $(pk, sk, \tilde{C}, sn)$ to keep track of account status.
   - Register honest account: $\mathcal{CH}$ picks a random balance m and a nonce sn, runs $\mathsf{CreateAcct}(pp; m, sn)$ to obtains (pk, sk) and $\tilde{C} \leftarrow \text{PKE.}\mathsf{Enc}(pk, m)$, returns (pk, sn, C) to $\mathcal{A}$. $\mathcal{CH}$ records $(pk, sk, \tilde{C}, sn)$ in L_{honest}.
   - Register corrupted account: $\mathcal{A}$ makes this query with a public key pk, a nonce sn and an initial encrypted balance $\tilde{C}$. $\mathcal{CH}$ records $(pk, \perp, C, sn)$ in L_{corrupt}.
   - Corrupt honest account: $\mathcal{A}$ makes a query with a public key pk in L_{honest}, $\mathcal{CH}$ returns sk to $\mathcal{A}$ and move the corresponding entry to L_{corrupt}.
   - Transfer: $\mathcal{A}$ makes a transaction query (pk_s, pk_r, v) subject to the restriction that $pk_s \in L_{\text{honest}}$. $\mathcal{CH}$ generates ctx via $\mathsf{CreateCTx}(sk_s, pk_s, pk_r, v)$, updates the associated account status, then returns ctx to $\mathcal{A}$.
   - Inject: $\mathcal{A}$ submits a confidential transaction ctx subject to the restriction that $pk_s \in L_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{CH}$ updates the associated account status. Otherwise, $\mathcal{CH}$ 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 $pk^*_s, pk^*_r \in L_{\text{honest}}$ and v_0, v_1 are right amounts. $\mathcal{CH}$ 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. $\mathcal{A}$ outputs a guess β' for β and wins if $\beta' = \beta$.

Game 1. Let Q_{H} be the maximum number of honest account registration queries that $\mathcal{A}$ makes. Game 1 is same as Game 0 except $\mathcal{CH}$ makes a random guess for index of target (pk^*_s, pk^*_r) at the beginning, i.e., randomly picks two index $0 < i < j \leq Q_{\mathsf{H}}$. If $\mathcal{A}$ makes an extraction query of pk_i or pk_j in the learning stage, or $\mathcal{A}$ picks $(pk^*_s, pk^*_r) \neq (pk_i, pk_j)$ in the challenge stage, $\mathcal{CH}$ aborts. Let W be the event that $\mathcal{CH}$'s guess is correct. It is easy to see that $\Pr[W] \geq 1/Q_{\mathsf{H}}(Q_{\mathsf{H}} - 1)$.

Conditioned on $\mathcal{CH}$ does not abort, $\mathcal{A}$'s view in Game 0 is identical to that in Game 1. Therefore, we have:

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

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 $\mathcal{S}_1(1^\lambda) \to (crs, \tau)$. When handling transaction queries and challenge queries, $\mathcal{CH}$ runs $\mathcal{S}_2(crs, \tau, \mathsf{meta})$ to generate π_{valid} for $\mathsf{meta} \in L_{\text{valid}}$. 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 IND-CPA security (1-plaintext, 2-recipient) of the underlying CPKS,* $\Pr[S_2] \leq \mathsf{negl}(\lambda)$ *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 IND-CPA security (1-plaintext, 2-recipient) of CPKS with the same advantage. Given the challenge (pp, pk_a, pk_b), $\mathcal{B}$ simulates Game 2 as follows:

1. Setup: $\mathcal{B}$ runs $\mathcal{S}_1.\mathsf{Setup}(1^\lambda) \to (crs, \tau)$, sends $gpp = (pp, crs)$ to $\mathcal{A}$. $\mathcal{B}$ randomly picks $1 < j < k < Q_{\mathsf{H}}$

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

   - Register honest account: On the i-th query, $\mathcal{B}$ picks a random balance m and a nonce sn and proceeds as below:
     - If $i \neq j$ and k, $\mathcal{B}$ runs $\mathsf{CreateAcct}(pp; m, sn)$ to obtain (pk, sk) and encrypted balance $\tilde{C} \leftarrow \text{PKE.}\mathsf{Enc}(pk, m)$, then records $(pk, sk, \tilde{C}, sn)$ in L_{honest}.
     - If $i = j$ or k, $\mathcal{B}$ sets $pk = pk_j$ or pk_k, $\mathcal{B}$ computes $\tilde{C} \leftarrow \text{PKE.}\mathsf{Enc}(pk, m)$, then records $(pk, \perp, \tilde{C}, sn)$ in L_{honest}.

     Finally, $\mathcal{B}$ returns $(pk, \tilde{C}, sn)$ to $\mathcal{A}$.
   - Register corrupted account: $\mathcal{A}$ makes this query with a public key pk, a nonce sn and an initial encrypted balance $\tilde{C}$. $\mathcal{B}$ records $(pk, \perp, \tilde{C}, sn)$ in L_{corrupt}.
   - Corrupt honest account: $\mathcal{A}$ makes a query with pk in L_{honest}. If $pk = pk_j$ or pk_k, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ returns sk to $\mathcal{A}$, then moves the corresponding entry to L_{corrupt}.
   - Transfer: $\mathcal{A}$ makes a transaction query (pk_s, pk_r, v) subject to the restriction that $pk_s \in L_{\text{honest}}$. $\mathcal{B}$ proceeds as below:
     (a) Fetch the current nonce sn and balance $\tilde{C}_s$ of pk_s, computes $C_s \leftarrow \text{PKE.}\mathsf{Enc}(pk_s, v)$, $C_r \leftarrow \text{PKE.}\mathsf{Enc}(pk_r, v)$;
     (b) Set $\mathsf{meta} = (pk_r, C_r, pk_s, C_s)$, compute $\pi_{\text{valid}} \leftarrow \mathcal{S}_2(crs, \tau, \mathsf{meta})$;
     (c) Set $\mathsf{memo} = (sn, \mathsf{meta}, \pi_{\text{valid}})$. If $pk_s \neq \{pk_j, pk_k\}$, generate a signature σ for memo with sk_s. Else, generate a signature σ for memo by querying the signing oracle.

     $\mathcal{B}$ updates the associated account status, then returns ctx to $\mathcal{A}$.
   - Inject: $\mathcal{A}$ submits a confidential transaction ctx subject to the restriction that $pk_s \in L_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{B}$ updates the associated account status. 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 $pk_s^*, pk_r^* \in L_{\text{honest}}$ and v_0, v_1 are right amounts. 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^*). $\mathcal{B}$ then computes ctx^* as follows:

   (a) Fetch the current nonce sn^* and balance $\tilde{C}_s^*$ of pk_s^*, set $\mathsf{meta}^* = (\tilde{C}_s^*, pk_s^*, C_s^*, pk_r^*, C_r^*)$.
   (b) Generate $\pi_{\text{valid}}^* \leftarrow \mathcal{S}_2(crs, \tau, \mathsf{meta}^*)$.

(c) Set $\mathsf{memo}^* = (sn^*, \mathsf{meta}^*, \pi^*_{\text{valid}})$, query the signing oracle of CPKS to obtain a signature σ^* of memo^* w.r.t. sk^*_s.

$\mathcal{B}$ updates the associated account status, then returns $\mathsf{ctx}^* = (\mathsf{memo}^*, \sigma^*)$ to $\mathcal{A}$.

Finally, $\mathcal{B}$ forwards $\mathcal{A}$'s guess β' for β to its own challenger. It is easy to see that $\mathcal{B}$'s simulation for Game 2 is perfect. The claim immediately follows. □

Putting all the above together, we proves the theorem. □

Lemma 4.4. *Assuming the security of the Signature component in combined public key scheme and the adaptive zero-knowledge property of NIZK, our CTx framework satisfies theft prevention.*

Proof. We proceed via a sequence of games.

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

1. Setup: $\mathcal{CH}$ runs CPKS.$\mathsf{Setup}(1^\lambda) \to pp$, NIZK.$\mathsf{Setup}(1^\lambda) \to crs$, then sends $gpp = (pp, crs)$ to $\mathcal{A}$.
2. Queries: Throughout the experiment, $\mathcal{A}$ can make the following types of queries adaptively. $\mathcal{CH}$ answers these queries by maintaining two lists L_{honest} and L_{corrupt}, which both are initially empty. L_{honest} and L_{corrupt} store entries of the form $(pk, sk, \tilde{C}, sn)$ to keep track of account status.
   - Register honest account: $\mathcal{CH}$ picks a random balance $\tilde{m}$ and a random nonce sn, then runs $\mathsf{CreateAcct}(pp; m, sn)$ to obtains (pk, sk) and $\tilde{C} \leftarrow$ PKE.$\mathsf{Enc}(pk, m)$, returns (pk, sn, C) to $\mathcal{A}$. $\mathcal{CH}$ records $(pk, sk, \tilde{C}, sn)$ in L_{honest}.
   - Register corrupted account: $\mathcal{A}$ makes this query with a public key pk, a nonce sn and an initial encrypted balance $\tilde{C}$. $\mathcal{CH}$ records $(pk, \perp, C, sn)$ in L_{corrupt}.
   - Corrupt honest account: $\mathcal{A}$ makes a query with a public key pk in L_{honest}, $\mathcal{CH}$ returns sk to $\mathcal{A}$ and move the corresponding entry to L_{corrupt}.
   - Transfer: $\mathcal{A}$ makes a transaction query (pk_s, pk_r, v) subject to the restriction that $pk_s \in L_{\text{honest}}$. $\mathcal{CH}$ generates ctx via $\mathsf{CreateCTx}(sk_s, pk_s, pk_r, v)$, updates the associated account status, then returns ctx to $\mathcal{A}$.
   - Inject: $\mathcal{A}$ submits a confidential transaction ctx subject to the restriction that $pk_s \in L_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{CH}$ updates the associated account status. Otherwise, $\mathcal{CH}$ ignores.
3. Forge: $\mathcal{A}$ outputs $(pk^*_s, \mathsf{memo}^*, \sigma^*)$, and wins in the theft prevention experiment if $pk^*_s \in L_{\text{honest}} \wedge \mathsf{memo}^* \notin L_{\text{trans}}(pk^*_s) \wedge$ SIG.$\mathsf{Verify}(pk^*_s, \mathsf{memo}^*, \sigma^*)$.

Game 1. Let Q_{H} be the maximum number of honest account registration queries that $\mathcal{A}$ makes. 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 two index $0 < j \leq Q_{\mathsf{H}}$. 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 $\mathcal{CH}$'s guess is correct. It is easy to see that $\Pr[W] \geq 1/Q_{\mathsf{H}}$.

Conditioned on $\mathcal{CH}$ does not abort, $\mathcal{A}$'s view in Game 0 is identical to that in Game 1. Therefore, we have:

$$\Pr[S_1] \geq \Pr[S_0] \cdot \frac{1}{Q_{\mathsf{H}}}$$

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 $\mathcal{S}_1(1^\lambda) \to (crs, \tau)$. When handling transaction queries, $\mathcal{CH}$ runs $\mathcal{S}_2(crs, \tau, \mathsf{meta})$ to generate π_{valid} for $\mathsf{meta} \in L_{\text{valid}}$. 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 EUF-CMA security of the underlying signature,* $\Pr[S_2] \leq \mathsf{negl}(\lambda)$ *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 CPKS with the same advantage. Given the challenge (pp, pk^*), $\mathcal{B}$ simulates Game 2 as follows:

1. Setup: $\mathcal{B}$ runs $\mathcal{S}_1.\mathsf{Setup}(1^\lambda) \to (crs, \tau)$, sends $gpp = (pp, crs)$ to $\mathcal{A}$. $\mathcal{B}$ randomly picks $1 < j < Q_\mathsf{H}$
2. Queries: Throughout the experiment, $\mathcal{A}$ can make the following types of queries adaptively. $\mathcal{B}$ answers these queries by maintaining two lists L_{honest} and L_{corrupt}, which both are initially empty.
   - Register honest account: On the i-th query, $\mathcal{B}$ picks a random balance m and a nonce sn and proceeds as below:
     - If $i \neq j$, $\mathcal{B}$ runs $\mathsf{CreateAcct}(pp; m, sn)$ to obtain (pk, sk) and encrypted balance $\tilde{C} \leftarrow \text{PKE.}\mathsf{Enc}(pk, m)$, then records $(pk, sk, \tilde{C}, sn)$ in L_{honest}.
     - If $i = j$, $\mathcal{B}$ sets $pk = pk^*$, $\mathcal{B}$ computes $\tilde{C} \leftarrow \text{PKE.}\mathsf{Enc}(pk, m)$, then records $(pk, \perp, \tilde{C}, sn)$ in L_{honest}.

     Finally, $\mathcal{B}$ returns $(pk, \tilde{C}, sn)$ to $\mathcal{A}$.
   - Register corrupted account: $\mathcal{A}$ makes this query with a public key pk, a nonce sn and an initial encrypted balance $\tilde{C}$. $\mathcal{B}$ records $(pk, \perp, \tilde{C}, sn)$ in L_{corrupt}.
   - Corrupt honest account: $\mathcal{A}$ makes a query with pk in L_{honest}. If $pk = pk_j$, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ returns sk to $\mathcal{A}$, then moves the corresponding entry to L_{corrupt}.
   - Transfer: $\mathcal{A}$ makes a transaction query (pk_s, pk_r, v) subject to the restriction that $pk_s \in L_{\text{honest}}$. $\mathcal{B}$ proceeds as below:
     (a) Fetch the current nonce sn and balance $\tilde{C}_s$ of pk_s, computes $C_s \leftarrow \text{PKE.}\mathsf{Enc}(pk_s, v)$, $C_r \leftarrow \text{PKE.}\mathsf{Enc}(pk_r, v)$;
     (b) Set $\mathsf{meta} = (pk_r, C_r, pk_s, C_s)$, compute $\pi_{\text{valid}} \leftarrow \mathcal{S}_2(crs, \tau, \mathsf{meta})$;
     (c) Set $\mathsf{memo} = (sn, \mathsf{meta}, \pi_{\text{valid}})$. If $pk_s \neq pk_j$, generate a signature σ for memo with sk_s. Else, generate a signature σ for memo by querying the signing oracle.

     $\mathcal{B}$ updates the associated account status, then returns ctx to $\mathcal{A}$.
   - Inject: $\mathcal{A}$ submits a confidential transaction ctx subject to the restriction that $pk_s \in L_{\text{corrupt}}$. If $\mathsf{VerifyCTx}(\mathsf{ctx}) = 1$, $\mathcal{B}$ updates the associated account status. Otherwise, $\mathcal{B}$ ignores.
3. Challenge: $\mathcal{A}$ outputs $pk_s^*, \mathsf{memo}^*, \sigma^*)$ as the forgery. If $pk_s^* \neq pk^*$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ forwards $(\mathsf{memo}^*, \sigma^*)$ to its own challenger.

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

Putting all the above together, we proves the theorem. □

Lemma 4.6. *Assuming the soundness of NIZK, our CTx framework satisfies soundness.*

Proof. The reduction is straightforward. We omit the details here. □

5 PGC: an Efficient Instantiation

We now present an efficient realization of the above framework. This is the most technical part of this work. For digital signature, we choose ECDSA, which is also adopted by Bitcoin and Etherum. For PKE, we twist the classical ElGamal encryption. The underlying reason has been elaborated in the introduction part. Next, we present our twisted ElGamal encryption and prove its security in the standard model.

5.1 Instantiating PKE

We first present our twisted ElGamal PKE as follows.

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

Correctness is 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 for completeness on its own right.

In this work, we require the twisted ElGamal is secure in the single plaintext, two recipient setting. A trivial solution is simply concatenating independently encryptions for two recipients, i.e., $\mathsf{Enc}(pk_1, m_1; r_1)||\cdots||\mathsf{Enc}(pk_n, m_n; r_n)$. The security of the trivial solution is implied by the results independently proved by Bellare et al. [BBM00] and Baudron et al. [BPS00]. Kurosawa [Kur02] first showed that for standard ElGamal PKE, randomness can be reused in the single-plaintext, multi-recipient setting. Zether [BAZB19] used the same idea to make the 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 first zero-knowledge proof system. We give a formal security proof as below.

Theorem 5.1. *Twisted ElGamal is IND-CPA secure in the single-plaintext, two-recipient setting 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 single-plaintext, two-recipient setting. Challenger $\mathcal{CH}$ interacts with $\mathcal{A}$ as below:

1. Setup: $\mathcal{CH}$ runs $\mathsf{Setup}(1^\lambda) \to pp$, then runs $\mathsf{Gen}(pp)$ twice independently to obtain $(pk_1 = h^{sk_1}, sk_1)$ and $(pk_2 = h^{sk_2}, sk_2)$, 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 a randomness r, computes $X_1 = pk_1^r$, $X_2 = pk_2^r$, $Y = g^{m_\beta} h^r$, 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 $X_1 = pk_1^r$, $X_2 = pk_2^r$, $Y = g^{m_\beta} h^s$, sends $C = (X_1, X_2, Y)$ 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, we can build an adversary $\mathcal{B}$ breaks the divisible DDH assumption with the same advantage. Given $\mathbb{G}$, g and (h, h^a, h^b, h^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 $v \xleftarrow{R} \mathbb{Z}_p$, sends $pp = (\mathbb{G}, g, h)$ and $pk_1 = h^b$ and $pk_2 = h^{bv}$ to $\mathcal{A}$. Here, b and bv 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 $X_1 = h^a$, $X_2 = h^{bv}$, $Y = g^{m_\beta} h^c$, sends $C = (X_1, X_2, Y)$ to $\mathcal{A}$.

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

If (h, h^a, h^b, h^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, the theorem follows. □

We now discuss some properties of twisted ElGamal, which is of great practical interest.

Homomorphism. The proposed twisted ElGamal PKE satisfies both randomness and message additive homomorphism, i.e., for any pk, (m_1, r_1) and (m_2, r_2), 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 slightly abuse the notation "+" to denote component-wise operation over ciphertext space $\mathbb{G}^2$. Looking ahead, additive homomorphism on message suffices for our usage: $\mathsf{Enc}(pk, m_1) + \mathsf{Enc}(pk, m_2)$ is an encryption of $(m_1 + m_2)$ under some appropriate randomness.

Decryption. To ensure additive homomorphism on message space $\mathbb{Z}_p^*$, the message is encoded in the exponent. The makes decryption inefficient even with knowledge of secret key. Looking ahead, this seems to be problematic because the exact balance is required to be known when generating the range proof for enough balance. The same problem also occurs in other confidential transaction systems [MDH+17, FMMO18, BAZB19].

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 (or at least a good estimate) are also known to the recipients in most times. This make sense because senders typically communicate the transfer amount to their recipients off-chain in real-world applications.

Now, let us consider the worst case, i.e., the recipients do not know the exact value of incoming transfer amounts. Suppose the transfer amount is restricted to $[0, n]$ by protocol specification. As suggested by [MDH+17, FMMO18, BAZB19], in this case the recipients can figure out m from g^m by brute-force enumeration with time complexity $O(n)$. Here, we recommend to use the Galbraith-Ruprai algorithm [GR10] or their improved variants [ZZY+19], which is dedicated to DLP in an interval. The expected time complexity is $O(\sqrt{n})$.

In this work, we set $n = 2^{32}$ in implementation. Under this setting, the average decryption time is less than 2s. If larger range are needed, we can adopted the approach as Zether [BAZB19] to represent 64-bit value by a concatenation of two 32-bit values. The decryption complexity only grows linearly.

5.2 Instantiating Signature

After choosing twist ElGamal as the PKE component, it remains to choose the signature component. Second, the signing operation will not compromise the security of the PKE component. In this work, we choose the Schnorr signature scheme [Sch91]. Our choice is out of three reasons. First, Schnorr signature share the same key generation algorithm as twist ElGamal. Second, its signing procedure of Schnorr signature is irrelevant to the decryption procedure of twist ElGamal. This suggests that it is likely we can safely implement key reuse strategy to build a combined public-key scheme, as we will rigorously prove later. Third, Schnorr signature is obtained is derived from Schnorr identification protocol by applying Fiat-Shamir transform. Therefore, it shares common structure as the NIZKPoK derived from Σ protocols. This allows us to reduce the footprint of overall cryptographic code. 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 Bitcoin blockchain [BDN18].

For completeness, we recall the Schnorr signature as below.

- $\mathsf{Setup}(1^\lambda)$: run $\mathsf{GroupGen}(1^\lambda) \to (\mathbb{G}, g, p)$, pick $h \xleftarrow{R} \mathbb{G}^*$, pick a cryptographic hash function $\mathsf{H} : M \times \mathbb{G} \to \mathbb{Z}_p$, where M denotes the message space. Finally, set $pp = (\mathbb{G}, g, h, p, \mathsf{H})$ as global public parameters.

- $\mathsf{Gen}(pp)$: on input pp, choose $sk \xleftarrow{R} \mathbb{Z}_p$, set $pk = h^{sk}$.
- $\mathsf{Sign}(sk, m)$: pick $r \xleftarrow{R} \mathbb{Z}_p$, set $A = h^r$, compute $e = \mathsf{H}(m, A)$, $z = r + sk \cdot e$, output $\sigma = (A, z)$.
- $\mathsf{Verify}(pk, \sigma, m)$: parse $\sigma = (A, z)$, compute $e = \mathsf{H}(m, A)$, output "1" if $h^z = A \cdot pk^e$ 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. $\mathcal{R}$ embeds its DLP challenge h^a into the public key (i.e., sets $pk = h^a$), 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.

5.3 Instantiating Combined Public Key Scheme

By merging the Setup and Gen algorithms of twisted ElGamal and Schnorr Signature, we obtain the combined public key scheme. It remains to prove its security in the joint model.

Theorem 5.3. *The combined public key scheme is jointly secure if the twist ElGamal is IND-CPA secure (single-plaintext, two-recipient) and the Schnorr signature is EUF-CMA secure.*

Proof. As we analyzed before, since in the joint security model we only require the PKE component satisfies 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 (single plaintext, two recipient) in the presence of a signing oracle for Schnorr signature. We proceed via a sequence of games.

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

1. Setup: $\mathcal{CH}$ runs $\mathsf{Setup}(1^\lambda) \to pp$, then runs $\mathsf{Gen}(pp)$ twice independently to obtain $(pk_1 = h^{sk_1}, sk_1)$ and $(pk_2 = h^{sk_2}, sk_2)$, then sends (pp, pk_1, pk_2) to $\mathcal{A}$.
2. Queries: Throughout the experiment, $\mathcal{A}$ can make two types of queries:
   - Hash queries: $\mathcal{CH}$ emulates random oracle by maintaining a list L using standard lazy method. L 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 $e := \mathsf{H}(m, A)$. On query (m, A), if there is an entry (m, A, e) in L, $\mathcal{CH}$ returns e. Else, $\mathcal{CH}$ picks $e \xleftarrow{R} \mathbb{Z}_p$ and inserts (m, A, e) in L, then returns e.
   - Signing queries w.r.t. sk_1 or sk_2: $\mathcal{CH}$ responds with corresponding secret key. On query (b, m), $\mathcal{CH}$ picks a fresh randomness $r \xleftarrow{R} \mathbb{Z}_p$, sets $A = h^r$, obtains $e = \mathsf{H}(m)$ by accessing the hash oracle, then computes $z = r + sk_b \cdot e$, return $\sigma = (A, z)$.
3. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and a randomness r, computes $X_1 = pk_1^r$, $X_2 = pk_2^r$, $Y = g^{m_\beta} h^r$, 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 oracles by programming random oracle H, rather than using the real secret keys.

2. Queries: $\mathcal{CH}$ handles hash queries and signing queries as below.

- Hash queries: $\mathcal{CH}$ emulates random oracle by maintaining a list L which is initially empty. L 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 wildcard. On query (m, A), if there is an entry $(m, A, e, *, *)$ in the L list, $\mathcal{CH}$ returns e. Else, $\mathcal{CH}$ picks $e \xleftarrow{R} \mathbb{Z}_p$, inserts $(m, A, e, \perp, \perp)$ in the L list, then returns e.
- Signing queries w.r.t. sk_1 or sk_2: $\mathcal{CH}$ responds with corresponding secret key. On query (b, m), if there is an entry (m, A, e, z, b) (with the same value of b and m) in the L list, $\mathcal{CH}$ simply returns $\sigma = (A, z)$. Otherwise, $\mathcal{CH}$ picks $z, e \xleftarrow{R} \mathbb{Z}_p$, sets $A = h^z/pk_b^e$, if there is no entry indexed by (m, A) in L, then inserts (m, A, e, z, b) in L and returns $\sigma = (A, z)$. Else, $\mathcal{CH}$ aborts to avoid possible inconsistent 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_h and Q_s be the maximum number of hash queries and signing queries that $\mathcal{A}$ makes during security experiment. By the union bound, we conclude that $\Pr[E] \leq (Q_s Q_h)/q$, 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 (single-message, two 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 twist 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, the theorem immediately follows. □

5.4 Instantiating NIZK

Now, we design efficient NIZK for L_{valid} and L_{correct} respectively.

Recall that L_{valid} can be decomposed as $L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{enough}}$. Let Π_{equal}, Π_{right}, Π_{enough} are NIZK for L_{equal}, L_{right}, L_{enough} respectively, and let $\Pi_{\text{valid}} := \Pi_{\text{equal}} \circ \Pi_{\text{right}} \circ \Pi_{\text{enough}}$, where $\circ$ denotes sequential composition.[7] By the property of NIZK for conjunctive statements [Gol06], Π_{valid} is a NIZK for L_{valid}. Now, the task breaks down to design Π_{equal}, Π_{right}, Π_{enough} and Π_{correct} separately.

5.4.1 NIZK for L_{equal}.

According to the general framework of confidential transaction and the definition of twisted ElGamal, L_{equal} is written as:

$$\{(pk_s, (X_s, Y_s), pk_r, (X_r, Y_r)) \mid \exists r_1, r_2, v \text{ s.t. } X_s = pk_s^{r_1} \wedge Y_s = g^v h^{r_1} \wedge X_r = pk_r^{r_2} \wedge Y_r = g^v h^{r_2}\}.$$

As analyzed before, for twisted ElGamal the randomness can be safely reused in the single-ciphertext multi-recipient setting. Therefore, L_{equal} can be simplified to:

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

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 is same as that of the twisted ElGamal. On statement $x = (pk_1, pk_2, X_1, X_2, Y)$ whose witness is $w = (r, v)$, P and V interact as described below:

[7] 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 $A_1 = pk_1^a$, $A_2 = pk_2^a$, $B = g^b h^a$ 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 the proof iff the following three equations hold simultaneously:

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

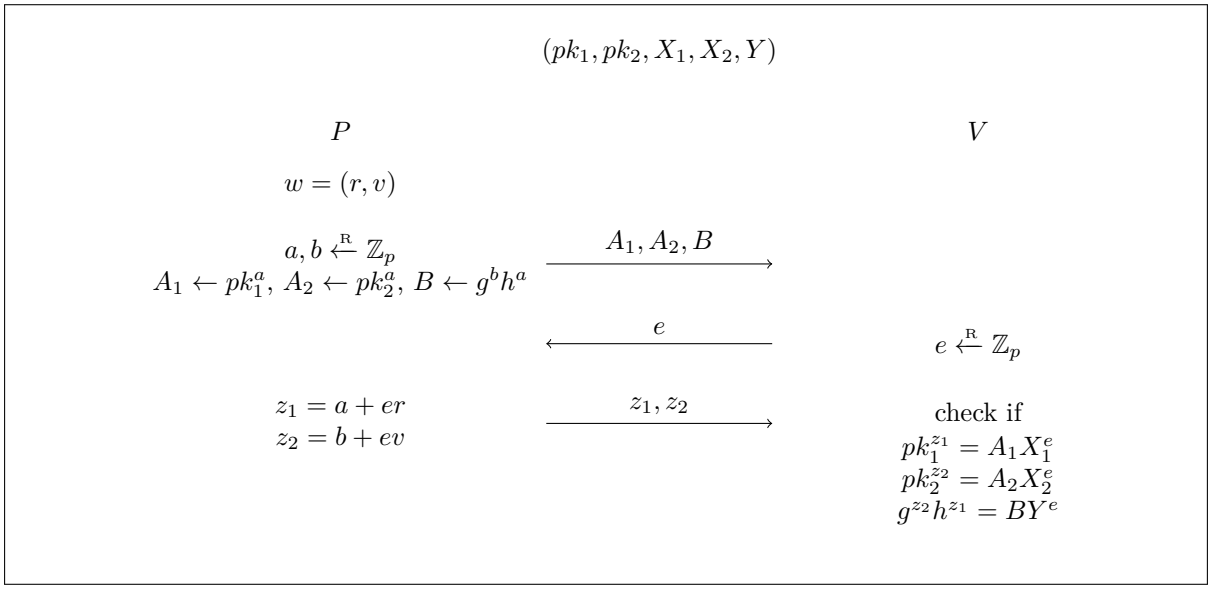

Figure 1: Σ_{equal} for L_{equal}: two twist ElGamal ciphertexts encrypt the same value

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

Proof. Perfect completeness is obvious from simple calculation.

To show special soundness, fix the initial message (A_1, A_2, B), suppose there are two accepted 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 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 $z_1, z_2 \xleftarrow{R} \mathbb{Z}_p$, computes $A_1 = pk_1^{z_1}/X_1^e$, $A_2 = pk_2^{z_1}/X_2^e$, $B = g^{z_2} h^{z_1}/Y^e$.

This finishes the proof. □

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

5.4.2 NIZK for L_{right}

According to the general framework of confidential transaction and the definition of twisted ElGamal, L_{right} is defined as:

$$\{(pk_1, X_1, Y) \mid \exists r, v \text{ s.t. } X_1 = pk_1^r \wedge Y = g^v h^r \wedge v \in [0, 2^\ell]\}.$$

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

$$L_{\text{enc}} = \{(pk_1, X_1, Y) \mid \exists r, v \text{ s.t. } X_1 = pk_1^r \wedge Y = g^v h^r\}$$
$$L_{\text{range}} = \{Y \mid \exists r, v \text{ s.t. } Y = g^v h^r \wedge v \in [0, 2^\ell]\}$$

It is straightforward to verify that $L_{\text{right}} \subset L_{\text{enc}} \wedge L_{\text{range}}$.

Observing that each instance $(pk_1, X_1, Y) \in L_{\text{right}}$ has a unique witness, while the last component Y can be viewed as a Pedersen commitment of value v under public parameters (g, h), whose discrete logarithm $\log_g h$ is unknown to any users. To prove $(pk_1, X_1, Y) \in L_{\text{right}}$, we first prove $(pk_1, X_1, Y) \in L_{\text{enc}}$ 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 the Bulletproof $\Lambda_{\text{bullet}} = (\mathsf{Setup}, P_2, V_2)$. We then show their sequential composition is a special honest verifier zero-knowledge argument of knowledge for L_{right}, assuming the hardness of discrete logarithm problem.

Sigma protocol for L_{enc}. We begin with the construction of a Sigma-protocol $\Sigma_{\text{enc}} = (\mathsf{Setup}, P, V)$ for L_{enc}. The Setup algorithm is the same as that of twisted ElGamal, a.k.a. generates a group $\mathbb{G}$ of prime order p together with two random generators g and h. On statement $x = (pk, X, Y)$ whose witness $w = (v, r)$, P and V interact as below:

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

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

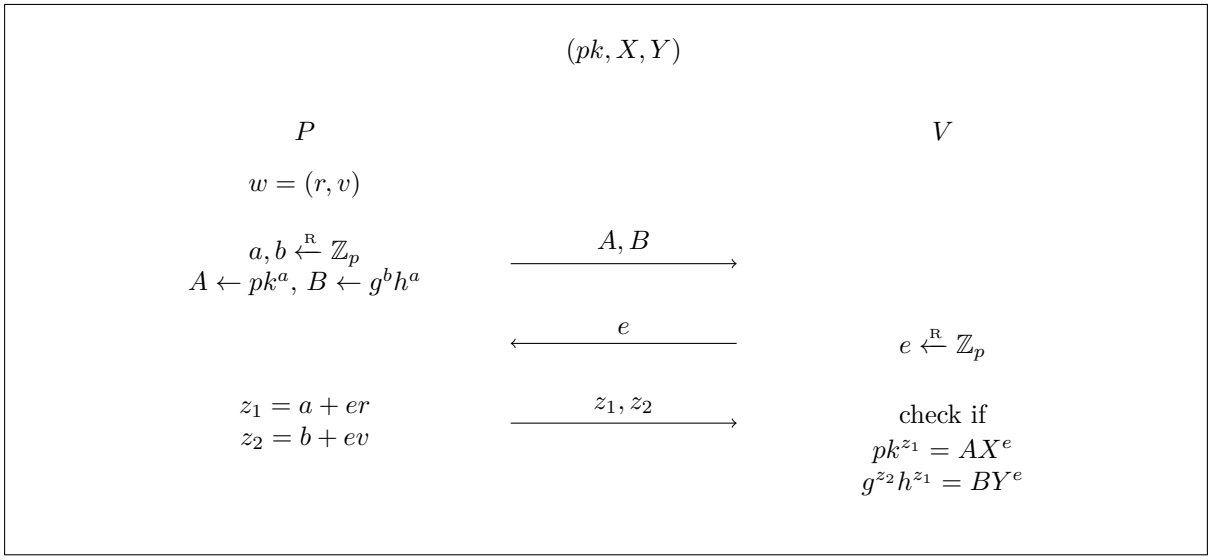

Figure 2: Σ_{enc} for L_{enc}: twisted ElGamal encryption

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

Proof. Perfect completeness is obvious from simple calculation.

To show special soundness, fix the initial message (A, B), suppose there are two accepted 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 $A = pk^{z_1}/X^e$, $B = g^{z_2} h^{z_1}/Y^e$.

This finishes the proof. □

Bulletproofs for L_{range}. We employ the logarithmic size Bulletproofs $\Lambda_{\text{bullet}} = (\mathsf{Setup}, P, V)$ to prove L_{range}. The Setup algorithm first generates a group $\mathbb{G}$ of prime order p together with two random generators g and h, then picks independent generators $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$. 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 discrete logarithm assumption, Λ_{bullet} is a public-coin SHVZK argument of knowledge for L_{range}.*

Sequential composition. Let Γ_{right} be the sequential composition of Σ_{enc} and Λ_{bullet}, i.e. $\Gamma_{\text{right}} = \Sigma_{\text{enc}} \circ \Lambda_{\text{bullet}}$. The Setup algorithm is a merge of that of Σ_{enc} and Λ_{bullet}, that is, generates a group $\mathbb{G}$ of prime order p together with random and independent generators $g, h \in \mathbb{G}$, $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$. Let $P_1 = \Sigma_{\text{enc}}.P$, $V_1 = \Sigma_{\text{enc}}.V$, $P_2 = \Lambda_{\text{bullet}}.P$, $V_2 = \Lambda_{\text{bullet}}.V$. Clearly, 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 an 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 $(n_0, n_1, \ldots, n_k)$-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 $(n_1, \ldots, n_k)$-subtree of accepting transcripts for Λ_{bullet} (see [BBB+18] for the values of $n_1, \ldots, n_k$).

E first runs E_1 on n_0-subtree of accepting transcripts to extract $w = (r, v)$, then runs E_2 on $(n_1, \ldots, n_k)$-subtree of accepting transcripts to extract $w' = (r', v')$. Finally, E outputs w if $w = w'$ and aborts otherwise.

According to Lemma 5.6, E_1 outputs a witness of statement $(pk_1, X_1, Y) \in L_{\text{enc}}$ 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 discrete logarithm assumption. We argue that E_2's output $w' = (r', v')$ equals E_1's output $w = (r, v)$ with overwhelming probability. Otherwise, the relation $g^v h^r = Y = g^{v'} h^{r'}$ immediately yields a solution $\log_g h$ to the discrete logarithm problem with respect to g and h. Therefore, E output a witness for $x = (pk_1, X_1, Y) \in L_{\text{right}}$ with overwhelming probability. Note that $\Pi_{i=0}^{k}$ is still bounded by a polynomial of λ.

Putting all the above together, Γ_{right} satisfies computational witness-extended emulation. This proves the theorem. □

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

5.4.3 NIZK for L_{enough}

According to the general framework of confidential transaction and the definition of twisted ElGamal, L_{enough} is defined as:

$$\{(pk_1, \tilde{C}_1, C_1) \mid \exists sk_1 \text{ s.t. } (pk_1, sk_1) \in \mathsf{R}_{\text{key}} \wedge \text{PKE.}\mathsf{Dec}(sk_1, \tilde{C}_1 - C_1) \in [0, 2^\ell - 1]\}.$$

In the above, $\tilde{C}_1 = (\tilde{X}_1 = pk_1^{\tilde{r}}, \tilde{Y}_1 = g^{\tilde{m}} h^{\tilde{r}})$ denotes the encrypted balance of sender, which encrypts the current balance $\tilde{m}$ of account pk_1 under randomness $\tilde{r}$. $C_1 = (X_1 = pk_1^r, Y_1 = g^v h^r)$, which encrypts the transfer value v under randomness r. Let $\tilde{C}_1 - C_1 = C_1' = (X_1' = pk_1^{r'}, Y_1' = g^{m'} h^{r'})$, L_{enough} can be rewritten as:

$$\{(pk_1, C_1') \mid \exists v', r' \text{ s.t. } C_1' = \text{PKE.}\mathsf{Enc}(pk_1, v'; r') \wedge v' \in [0, 2^\ell - 1]\}.$$

By the message and randomness homomorphism of twisted ElGamal, we have $r' = \tilde{r} - r$ and $m' = \tilde{m} - v$. It seems that we can prove L_{enough} using the same protocol as we prove L_{right}. However, while the sender (playing the role of prover) learns $\tilde{m}$ (by decrypting $\tilde{C}_1$ with sk_1), v and r, it generally does not know the randomness $\tilde{r}$. This is because $\tilde{C}_1$ is the sum of all the incoming and outgoing transactions, in which the randomness underlying incoming transactions is unknown. The consequence is that r' (the first component of witness) is unknown, which render us unable to directly invoke the Bulletproof on instance Y_1'.

Our trick is encrypting the value $m' = (\tilde{m} - v)$ under new fresh randomness r^* to obtain a new ciphertext $C_1^* = (X_1^*, Y_1^*)$, where $X_1^* = pk_1^{r^*}$, $Y_1^* = g^{m'} h^{r^*}$. C_1^* could be viewed as a refreshment of C_1'. In a nutshell, we reconstruct L_{enough} as $L_{\text{equal}} \wedge L_{\text{right}}$, a.k.a. we have:

$$(pk_1, C_1') \in L_{\text{enough}} \iff (pk_1, C_1', C_1^*) \in L_{\text{equal}} \wedge (pk_1, C_1^*) \in L_{\text{right}}$$

To prove C_1' and C_1^* encrypt the same value under public key pk_1, we can not simply use Protocol 5.4.1. The reason is same as analyzed before, the prover in Protocol 5.4.1 uses the message and randomness as witness, while the underlying randomness r' is unknown. Luckily, we are able to prove this more efficiently by using the secret key as witness. Generally, L_{equal} can be written as:

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

When instantiating with twisted ElGamal, $C_1 = (X_1 = pk^{r_1}, Y_1 = g^m h^{r_1})$ and $C_2 = (X_2 = pk^{r_2}, Y_2 = g^m h^{r_2})$ are encryptions under the same public key pk, then proving membership of L_{equal} is equivalent to prove the discrete logarithm of $\log_{Y_1/Y_2} X_1/X_2$ equals $\log_g pk$ with witness sk. This can be efficiently done by using the Sigma-protocol $\Sigma_{\text{ddh}} = (\mathsf{Setup}, P, V)$ for discrete logarithm equality [CGS97]. For 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) is a DDH tuple. On statement $x = (g_1, h_1, g_2, h_2)$ whose witness is w, 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 $z = a + we$ and sends it to V. V accepts the proof iff:

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

Lemma 5.9 ([CGS97]). *Σ_{ddh} is a public-coin SHVZK argument of knowledge for L_{ddh}.*

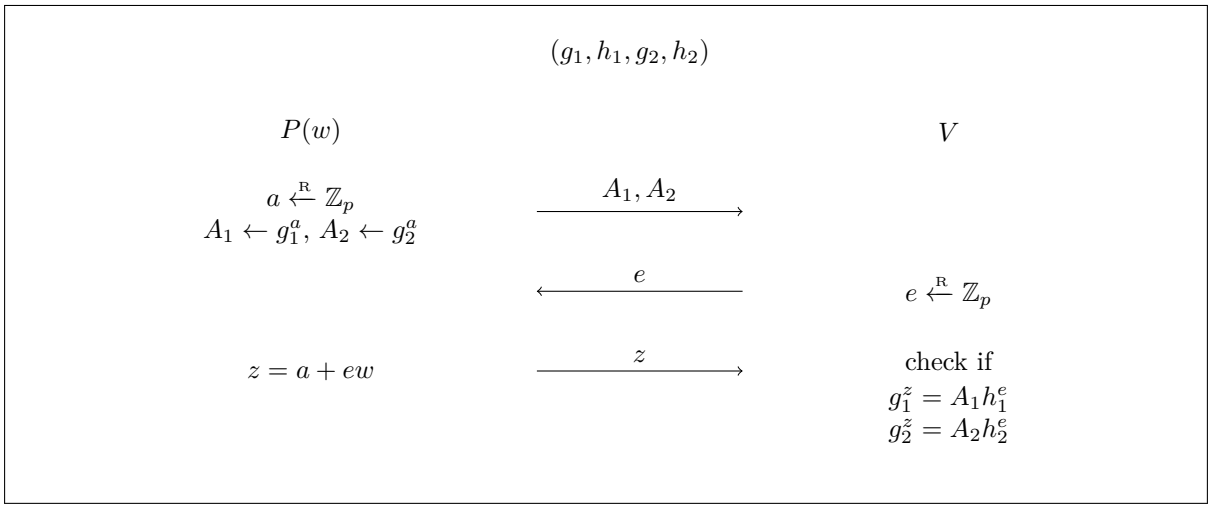

Figure 3: Σ_{ddh} for L_{ddh}: discrete logarithm equality/DDH tuple

Applying Fiat-Shamir transform to Σ_{ddh}, we obtain a NIZK Π_{ddh} for L_{ddh}. We then prove $(pk_1, C_1^* = (X_1^*, Y_1^*)) \in L_{\text{right}}$ using the NIZK Π_{right} as we described in Sec 5.4.2. Let $\Pi_{\text{enough}} = \Pi_{\text{ddh}} \circ \Pi_{\text{right}}$, we conclude that Π_{enough} is a NIZK for L_{enough} by the properties of AND-proofs.

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

Theorem 5.10. Π_{valid} *is a NIZK for* $L_{\text{valid}} = L_{\text{equal}} \wedge L_{\text{right}} \wedge L_{\text{enough}}$.

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

Remark 5.1. In Section 5.4.2, we show how to build a NIZK for L_{right} (asserting a twist ElGamal ciphertext encrypts a message in the claimed range). To do so, 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. We distill this language on its own right. Particularly, Π_{right} for L_{right} can be used as a sub-protocol when proving membership in L_{enough}.

Towards an more efficient NIZK for L_{valid} in our setting, we can simplify $L_{\text{equal}} \wedge L_{\text{right}}$ to L_{legal}, which asserts that two twist ElGamal ciphertexts encrypt the same message and the message lies in the right range. To prove membership in L_{legal}, we compose Σ_{equal} and Λ_{bullet} sequentially, then obtain a NIZK for $L_{\text{equal}} \wedge L_{\text{valid}}$ by applying Fiat-Shamir transform to $\Sigma_{\text{equal}} \circ \Lambda_{\text{bullet}}$. In this way, we save the cost of a NIZK for L_{enc} when proving L_{valid}.

5.4.4 NIZK for L_{correct}

According to the definition of twisted ElGamal, L_{correct} can be written as:

$$\{(pk_i, C_i, v) \mid \exists sk_i \text{ s.t. } X_i = (Y_i/g^v)^{sk_i} \wedge pk_i = h^{sk_i}\}$$

The above language is equivalent to $(Y_i/g^v, X_i, h, pk_i) \in L_{\text{ddh}}$. This can be efficiently proved via Π_{ddh} for discrete logarithm equality, as described in Protocol 5.4.3.

5.5 Optimization

Remove explicit signature. At a high level, confidential transaction systems use signatures to provide theft prevention and use zero-knowledge proofs 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 $H(I, m)$, 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. [BAZB19] suggested that instead of instantiating a separate signature scheme one can leverage ZKPs to provide signature functionality. In our PGC instantiation, when building NIZK for L_{enough} we use Σ_{ddh} for L_{ddh} as a sub-protocol, which is exactly a proof of knowledge of sender's secret key sk_1. By appending (sn, meta) to the prover's first round message, the obtained zero-knowledge proof π_{ddh} for L_{enough} also acts as a sender's signature of (sn, meta). In this way, we obtain the signature functionality almost for free. A subtlety here is that this approach does not enable us to sign the whole memo information $(sn, \mathsf{meta}, \pi_{\text{valid}})$, since there seems no way to append π_{ddh} to the input of H before generating it. Nevertheless, as we discussed before, authenticating (sn, meta) is sufficient for theft prevention.

6 Performance

6.1 Implementation and Performance

To evaluate the performance of PGC in practice we give a reference implementation C++. We use the elliptic curve secp256k1 which has 128 bit security.

For demo purpose, we do not explore any optimizations. All experiments were performed on MAC OS system and using a single thread. The memory used is less than XX MB. A single 64-bit range proof is 688 bytes.

7 Future Works

We defer the performance analysis as the future work.

Σ_1	Prove	Verify	Proof Size
Asymptotic	14Exp+8Add	9Exp+6Add	$4\|\mathbb{G}\|+3\|\mathbb{Z}_p\|$
Practical	XX	XX	XX

Table 1: The computation and communication complexity of the NIZKs of Σ_1

Σ_2	Prove	Verify	Proof Size
Asymptotic	14Exp+8Add	9Exp+6Add	$4\|\mathbb{G}\|+3\|\mathbb{Z}_p\|$
Practical	XX	XX	XX

Table 2: The computation and communication complexity of the NIZKs of Σ_2

Range Proof(Bulletproof)	Prove	Verify	Proof Size
Asymptotic	?	?	$(4+2\log \ell)\|\mathbb{G}\|+5\|\mathbb{Z}_p\|$
Practical	XX	XX	XX

Table 3: haha

Acknowledgments

We thank Benny Pinkas for the helpful discussions on Sigma Protocol, and thank Jonathan Bootle for the enlightening discussions on Bulletproof. We thank Yi Deng and Shunli Ma for many insightful comments and discussions on this work.

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*, volume 2332 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2002.

[BAZB19] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Report 2019/191, 2019. 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. IEEE Computer Society, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. 2018. http://eprint.iacr.org/2018/046.

[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 *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55, 2004.

[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*, volume 9666 of *Lecture Notes in Computer Science*, pages 327–357. Springer, 2016.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology - CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014*, pages 459–474. IEEE Computer Society, 2014.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology - ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464. Springer, 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*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2003.

[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.

[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*, volume 1853 of *Lecture Notes in Computer Science*, pages 499–511. Springer, 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. ACM, 1993.

[CCS08] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 234–252. Springer, 2008.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology - EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. 7668:60–79, 2012.

[FMMO18] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. Cryptology ePrint Archive, Report 2018/990, 2018. https://eprint.iacr.org/2018/990.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO 1986*, volume 263 of *LNCS*, pages 186–194, 1986.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[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 *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 368–383. Springer, 2010.

[Kur02] Kaoru Kurosawa. Multi-recipient public-key encryption with shortened ciphertext. In *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2002.

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

[MDH+17] Shunli Ma, Yi Deng, Debiao He, Jiang Zhang, and Xiang Xie. An efficient NIZK scheme for privacy-preserving transactions over account-model blockchain. 2017. https://eprint.iacr.org/2017/1239.

[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. Cryptology ePrint Archive, Report 2015/1098, 2015. https://eprint.iacr.org/2015/1098.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

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

[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*, volume 7073 of *LNCS*, pages 161–178. Springer, 2011.

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

[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/.

[ZZY+19] Yuqing Zhu, Jincheng Zhuang, Hairong Yi, Chang Lv, and Dongdai Lin. A variant of the galbraith-ruprai algorithm for discrete logarithms with improved complexity. *Des. Codes Cryptography*, 87(5):971–986, 2019.

A A General Forking Lemma

We briefly 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 $|p| = \lambda$.

Theorem A.1 (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.*

B Basic Cryptographic Schemes

B.1 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, outputs public parameters pp. We assume that pp includes the descriptions of message space M, randomness space R. 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 $\mathsf{Com}(m; r) \stackrel{?}{=} c$.

For correctness, we require that for all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, any $m \in M$ and any $r \in R$, 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:

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

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.), statistical binding (resp. computational binding) if $\mathsf{Adv}_{\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$ w.r.t. unbounded adversary (resp. PPT adversary).

Pedersen Commitment. We recall the celebrated Pedersen commitment as follows:

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

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

B.2 Public Key Encryption

A public key encryption scheme consists of four polynomial time algorithms as follows.

- $\mathsf{Setup}(1^\lambda)$: on input a security parameter 1^λ, output public parameters pp.
- $\mathsf{Gen}(pp)$: 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.
- $\mathsf{Dec}(sk, c)$: 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 $(pk, sk) \leftarrow \mathsf{Gen}(pp)$, we have $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$.

B.3 Signatures

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.
- $\mathsf{Gen}(pp)$: 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.
- $\mathsf{Vrfy}(pk, m, \sigma)$: on input pk, a message m, and a purported signature σ, output a bit b indicating acceptance or rejection.

Correctness. For all $pp \leftarrow \mathsf{Setup}(1^\lambda)$, all $(vk, sk) \leftarrow \mathsf{Gen}(pp)$ and all m in the message space, we have $\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

B.4 Combined Signature and Encryption Schemes

A combined signature and encryption scheme [PSST11] is a combination of a signature scheme and a PKE scheme that share a key generation algorithm and hence a keypair (pk, sk). It consists of a tuple of algorithms $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Enc}, \mathsf{Dec})$ such that $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ form a signature scheme and $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ form a PKE scheme.

As elaborated in [PSST11], the combined signature and encryption schemes use the same keypair for both signing and encryption. Therefore, perhaps the two uses will interacts with one another badly, in such a way to undermine the security of one or both of the primitives. For this reason, we need to consider joint security when using combined signature and encryption scheme, which captures possible dangerous interactions.

In the context of our CTX framework, the joint security stipulates that the PKE component is IND-CPA secure in the single-plaintext/2-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 extend the IND-CPA security for the PKE component. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{Enc}, \mathsf{Dec})$ be a combined signature and encryption scheme, and let $\mathcal{O}_{\mathsf{sign}}$ be the signing oracle. We formally define the case-tailored joint security model used in this work.

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:

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

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 from the message space.

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:

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

The set $\mathcal{Q}$ records queries to $\mathcal{O}_{\mathsf{sign}}(\cdot)$. A signature is EUF-CMA if no PPT adversary $\mathcal{A}$ has non-negligible advantage in the above security experiment.

C Protocols

D Missing Proofs

Theorem D.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: $\mathcal{CH}$ sends $pk = h^{sk}$ to $\mathcal{A}$ as the public key.
2. Challenge: $\mathcal{A}$ submits m_0, m_1 to $\mathcal{CH}$ as the target messages. $\mathcal{CH}$ picks a random bit β and a randomness r, computes $X = pk^r$, $Y = g^{m_\beta} h^r$, 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 $X = pk^r$, $Y = g^{m_\beta} h^s$, 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, we can build an adversary $\mathcal{B}$ breaks the divisible DDH assumption with the same advantage. Given (h, h^a, h^b, h^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: $\mathcal{B}$ sends h^b to $\mathcal{A}$ as the public key, where b is interpreted as the corresponding secret key $b \in \mathbb{Z}_p$ and 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 $X = h^a$, $Y = g^{m_\beta} h^c$, 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 (h, h^a, h^b, h^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, the theorem follows. □