Part 2: SPL ZK-Tokens Proof of Security

Solana Labs

Last Updated: December 21, 2021

1 Introduction

This document is part 2 of the SPL ZK-Token protocol speci cation. It is inteded for advanced readers who wish to verify the proofs for the ZK-Token program. We provide formal details of the ZK-Token protocol and its rigorous security proofs.[1]

1.1 Organization

We divide this document into the following main sections:

- Section 3 provides a formal description of the twisted ElGamal encryption as well as the formal correctness and security theorems.
- Section 4 provides a formal description of the zero-knowledge argument systems that are used in the ZK-Token program.
- Section 5 provides the formal de nitions of our con dential payment system abstraction.
- Section 6 provides the formal description of the ZK-Token program.

The formal proofs of the correctness and security theorems are provided in the appendices.

2 Preliminaries

Basic notation. For two integers $n < m$, we write $[n, m]$ to denote the set $\{n, n+1, \ldots, m\}$. When $n = 1$, we simply write $[m]$ to denote the set $\{1, \ldots, m\}$. For any nite set S, we use $x \leftarrow_{\mathsf{R}} S$ to denote the process of sampling an element $x \in S$ uniformly at random. Unless speci ed otherwise, we use λ to denote the security parameter. We say that an algorithm is e cient if it runs in probabilistic polynomial time in the length of its input. We say that a function $f : \mathbb{N} \to \mathbb{N}$ is negligible if $f = o(1/n^c)$ for any positive integer $c \in \mathbb{N}$. Throughout the exposition, we use $\mathsf{poly}(\cdot)$ and $\mathsf{negl}(\cdot)$ to denote any polynomial and negligible functions respectively.

[1] The proofs are currently work-in-progress.

2.1 Cryptographic Assumptions

The security of the `ZK-Token` protocol relies on two standard cryptographic assumptions on a prime order group $\mathbb{G}$. The first assumption is the discrete log relation assumption, which we use for the security of zero-knowledge proofs. It states that given a number of random group elements in $\mathbb{G}$, no efficient adversary can find a non-trivial relation on these elements.

Definition 2.1 (Discrete Log Relation)**.** Let $\mathbb{G} = \mathbb{G}(\lambda)$ be a group of prime order p. Then the *discrete log relation* assumption on $\mathbb{G}$ states that for any efficient adversary $\mathcal{A}$ and $n \geq 2$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr\left[\mathcal{A}(G_1, \ldots, G_n) \to a_1, \ldots, a_n \in \mathbb{Z}_p : \exists\ a_i \neq 0 \wedge \sum_{i \in [n]} a_i \cdot G = 0\right] = \mathsf{negl}(\lambda),$$

where $G_1, \ldots, G_n \leftarrow_{\mathsf{R}} \mathbb{G}$.

The second assumption is the standard Decision Diffie-Hellman (DDH) assumption on $\mathbb{G}$, which we use for the security of the twisted ElGamal encryption.

Definition 2.2 (Decision Diffie-Hellman)**.** Let $\mathbb{G} = \mathbb{G}(\lambda)$ be a group of prime order p. Then the *Decision Diffie-Hellman* assumption on $\mathbb{G}$ states that for any efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr[\mathcal{A}(G, a \cdot G, b \cdot G, ab \cdot G) = 1] - \Pr[\mathcal{A}(G, a \cdot G, b \cdot G, u \cdot G) = 1]\right| = \mathsf{negl}(\lambda),$$

where $a, b, u \leftarrow_{\mathsf{R}} \mathbb{Z}_p$.

2.2 Rewinding Lemma

To prove the security of the zero-knowledge sigma protocols in the `ZK-Token` program, we make use of the rewinding lemma. For the purpose of these proofs, we do not require the rewinding lemma in its full generality and therefore, we rely on the following simple variant from the work of Boneh et al. [1].

Lemma 2.3 (Rewinding Lemma)**.** *Let S, R, and T be finite, non-empty sets, and let X, Y, Y', Z, and Z' be mutually independent random variables such that*

- *X takes values in the set S,*
- *Y and Y' are each uniformly distributed over R,*
- *Z and Z' take values in the set T.*

Then for any function $f : S \times R \times T \to \{0, 1\}$, we have

$$\Pr\left[f(X, Y, Z) = 1 \wedge f(X, Y', Z') = 1 \wedge Y \neq Y'\right] \geq \varepsilon^2 - \varepsilon/N,$$

where $\varepsilon = \Pr[f(X, Y, Z) = 1]$ and $N = |R|$.

2.3 Pedersen Commitments

The `ZK-Token` program relies on encryption rather than commitments to encode transfer amounts and account balances. Although the protocol can be described entirely with respect to the twisted ElGamal encryption scheme and the corresponding zero-knowledge proofs, the concept of Pedersen commitments is nevertheless an important object that facilitate the intuition behind the `ZK-Token` protocol. Instead of formally defining the abstract concept of commitment schemes and the required security properties, we focus primarily on Pedersen commitments themselves and the properties that they satisfy.

Definition 2.4. Let $\mathbb{G}$ be a cyclic group of prime order p and let G, H be any fixed group elements in $\mathbb{G}$. Then a *Pedersen commitment* of a *message* $x \in \mathbb{Z}_p$ and an *opening* r is defined as follows:

- $\mathsf{Comm}(x, r) = x \cdot G + r \cdot H$

Pedersen commitments satisfy the following properties:

- *Computationally binding*: Suppose that the discrete log relation assumption (Definition 2.1) holds on $\mathbb{G}$. Then for any efficient adversary $\mathcal{A}$, we have

  $$\Pr\left[\mathcal{A}(G, H) \to (x, r, r') \wedge \mathsf{Comm}(x, r) = \mathsf{Comm}(x, r') \wedge r \neq r'\right] = \mathsf{negl}(\lambda),$$

  where $G, H \leftarrow_{\mathsf{R}} \mathbb{G}$.

- *Perfect hiding*: For any two elements $x, y \in \mathbb{Z}_p$, the distribution of $\mathsf{Comm}(x, r)$ and $\mathsf{Comm}(y, r')$ for $r, r' \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ are identically distributed.

2.4 Cryptographic Signatures

In this section, we provide the standard definition of a digital signature scheme.

Definition 2.5 (Signatures)**.** A signature scheme $_{\mathsf{S}}$ for a message space $\mathcal{M}$ consists of a tuple efficient algorithms $_{\mathsf{S}} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: On input the security parameter λ, the key generation algorithm returns a public key pk and secret key sk.

- $\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \to \sigma$: On input a secret key sk and a message $\mathsf{m} \in \mathcal{M}$, the signing algorithm returns a signature σ.

- $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \sigma) \to 0/1$: On input a public key pk, message m, and signature σ, the verification algorithm either accepts (returns 1) or rejects (returns 0).

The standard correctness and the security requirements for a signature scheme are defined as follows.

Definition 2.6 (Correctness)**.** Let $_{\mathsf{S}}$ be a signature scheme for a message space $\mathcal{M}$. We say that $_{\mathsf{S}}$ satisfies perfect correctness if for all security parameter $\lambda \in \mathbb{N}$ and message $\mathsf{m} \in \mathcal{M}$, we have

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, \mathsf{Sign}(\mathsf{sk}, \mathsf{m})) = 1\right] = 1,$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Definition 2.7 (Security). Let Π_{S} be a signature scheme for a message space $\mathcal{M}$. For a security parameter $\lambda \in \mathbb{N}$, an adversary $\mathcal{A}$, we define the unforgeability security experiment $\mathsf{EXP}_{\mathsf{S}}[\lambda, \mathcal{A}]$ as follows:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
2. $(\mathsf{m}^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{pk})^{\mathsf{Sign}(\mathsf{sk}, \cdot)}$
3. Output $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}^*, \sigma^*)$

We say that an adversary $\mathcal{A}$ is admissible for $\mathsf{EXP}_{\mathsf{S}}$ if it does not forge on a message m^* that it previously queried to the signing oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$. We say that a signature scheme Π_{S} is *unforgeable* if for any efficient and admissible adversary $\mathcal{A}$, we have

$$\Pr\left[\mathsf{EXP}_{\mathsf{S}}[\lambda, \mathcal{A}] = 1\right] = \mathsf{negl}(\lambda).$$

3 Twisted ElGamal Encryption

In this section, we describe the twisted ElGamal encryption [3]. We first present the correctness and security definitions of a public key encryption shceme in Section 3.1. We provide the formal specification of the twisted ElGamal encryption in Section 3.2. We present the formal correctness and security theorems in Section 3.3.

3.1 Public Key Encryption

In this section, we define the formal syntax for a public key encryption and its security requirements.

Definition 3.1 (Public Key Encryption). A public key encryption scheme Π_{PKE} for a message space $\mathcal{M}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{PKE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: On input the security parameter λ, the key generation algorithm returns a public key pk and secret key sk.

- $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{m}) \to \mathsf{ct}$: On input a public key pk and a message $\mathsf{m} \in \mathcal{M}$, the encryption algorithm returns a ciphertext ct.

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{m}/\perp$: On input a secret key sk and a ciphertext ct, the encryption algorithm returns a message m or $\perp$.

A public key encryption scheme must satisfy the following correctness requirement.

Definition 3.2 (Correctness). Let $\Pi_{\mathsf{PKE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a public key encryption scheme for a message space $\mathcal{M}$. We say that Π_{PKE} satisfies perfect correctness if for all security parameter $\lambda \in \mathbb{N}$ and message $\mathsf{m} \in \mathcal{M}$, we have

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{pk}, \mathsf{m})) = \mathsf{m}\right] = 1,$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

The `ZK-Token` program relies on a public key encryption scheme that is secure against passive adversaries. Formally, we define the standard security requirements for a public key encryption as follows.

Definition 3.3 (Security)**.** Let $_{\mathsf{PKE}}$ = (KeyGen, Encrypt, Decrypt) be a public key encryption scheme for a message space $\mathcal{M}$. For a security parameter $\lambda \in \mathbb{N}$, an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, and a bit $b \in \{0,1\}$, we define the IND-CPA security experiment $\mathsf{EXP}_{\mathsf{PKE}}[\lambda, \mathcal{A}, b]$ as follows:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
2. $(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pk})$
3. $\mathsf{ct}_b \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \mathsf{m}_b)$
4. Output $\mathcal{A}_2(\mathsf{st}, \mathsf{ct}_b)$

We say that a public key encryption scheme $_{\mathsf{PKE}}$ is *IND-CPA* secure if for any efficient adversary $\mathcal{A}$, we have

$$\left| \Pr\left[\mathsf{EXP}_{\mathsf{PKE}}[\lambda, \mathcal{A}, 0] = 1\right] - \Pr\left[\mathsf{EXP}_{\mathsf{PKE}}[\lambda, \mathcal{A}, 1] = 1\right] \right| = \mathsf{negl}(\lambda).$$

The final property that we require from a public key encryption scheme is linear homomorphism. We require that the sum of two ciphertexts that are encrypted under the same public key produces a ciphertext that encrypts the sum of the encrypted messages in each of the two ciphertexts.

Definition 3.4 (Linear Homomorphism)**.** Let $_{\mathsf{PKE}}$ = (KeyGen, Encrypt, Decrypt) be a public key encryption scheme for a message space $\mathcal{M}$. We say that $_{\mathsf{PKE}}$ is *linearly homomorphic* if for all security parameter $\lambda \in \mathbb{N}$ and messages $\mathsf{m}_0, \mathsf{m}_1$, we have

$$\Pr\left[\mathsf{Decrypt}\big(\mathsf{Encrypt}(\mathsf{pk}, \mathsf{m}_0) + \mathsf{Encrypt}(\mathsf{pk}, \mathsf{m}_1)\big) = \mathsf{m}_0 + \mathsf{m}_1\right] = 1,$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

3.2 Construction Specification

In this section, we describe the twisted ElGamal encryption. The twisted ElGamal encryption was formulated in the work of Chen et al. [3]. It has an advantage over the standard ElGamal encryption scheme in that zero-knowledge proof systems that are designed specifically for Pedersen commitments can be used directly on the ciphertexts.

A regular ElGamal encryption is defined with respect to a fixed group element $G \in \mathbb{G}$. Its ciphertext consist of two group elements $C = x \cdot G + r \cdot H$ and $D = r \cdot G$ for a message $x \in \mathbb{Z}_p$, randomness $r \in \mathbb{Z}_p$, and public key $H \in \mathbb{G}$. If group elements G and H are fixed system parameters, then proof systems such as Bulletproofs [2] that are designed for Pedersen commitments can be used directly on C. However, as H is a user generated public key component, the soundness of these proof systems can be violated if the prover knows a secret key that corresponds to H.

A twisted ElGamal encryption is defined with respect to two fixed group elements $G, H \in \mathbb{G}$. Its ciphertext consist of two group elements $C = x \cdot G + r \cdot H$ and $D = r \cdot P$ for a message x and randomness r, and public key P. In contrast to the standard ElGamal encryption scheme, the component $C = x \cdot G + r \cdot H$ is a valid Pedersen commitment over two fixed group elements $G, H \in \mathbb{G}$. Therefore, proofs that are designed specifically for Pedersen commitments can be used directly on this component of the cipehrtext. The formal specification of the twisted ElGamal encryption scheme is as follows.

Construction 3.5 (Twisted ElGamal Encryption)**.** Let $\mathbb{G}$ be a cyclic group of prime order p and let $G, H \in \mathbb{G}$ be two group elements. Then the twisted ElGamal encryption scheme for a message space $\mathcal{M} \subseteq \mathbb{Z}_p$ is specified as follows:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: The key generation algorithm samples a non-zero scalar $s \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. It computes $P = s^{-1} \cdot H$ and sets
$$\mathsf{pk} = P, \quad \mathsf{sk} = s.$$

- $\mathsf{Encrypt}(\mathsf{pk}, x) \to \mathsf{ct}$: The encryption algorithm takes in a public key $\mathsf{pk} = P \in \mathbb{G}$ and a message $x \in \mathbb{Z}_p$ to be encrypted. It samples a random scalar $r \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ and then computes the following components:

  1. *Pedersen commitment*: $C = r \cdot H + x \cdot G$,
  2. *Decryption handle*: $D = r \cdot P$.

  It returns $\mathsf{ct} = (C, D)$.

  Deterministic encryption: For the protocol speci cation in Section 6, we use $\mathsf{Encrypt}(\mathsf{pk}, x; 0)$ to denote the deterministic twisted ElGamal encryption that sets the random scalar r to always be $r = 0$.

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to x$: The decryption algorithm takes in a secret key $\mathsf{sk} = s$ and a ciphertext $\mathsf{ct} = (C, D)$ as input. It computes
$$V = C - s \cdot D \quad \in \mathbb{G},$$
  and then solves the discrete log problem to recover $x \in \mathbb{Z}_p$ for which $x \cdot G = V$.

3.3 Correctness and Security Properties

We formally state the correctness and security properties of the twisted ElGamal encryption.

Theorem 3.6 (Correctness)**.** *Let $\mathcal{M} \subseteq \mathbb{Z}_p$ be any set with cardinality $|\mathcal{M}| = \mathsf{poly}(\lambda)$. Then the twisted ElGamal encryption scheme for the message space $\mathcal{M}$ satisfies correctness as specified in Definition 3.2.*

Theorem 3.7 (Security)**.** *Suppose that $\mathbb{G}$ is a prime order group for which the decision Diffie-Hellman assumption (Definition 2.2) holds. Then the twisted ElGamal encryption scheme satisfies IND-CPA security as specified in 3.3.*

Theorem 3.8 (Linear Homomorphism)**.** *The twisted ElGamal encryption scheme in Construction 3.5 satisfies linear homomorphism as specified in Definition 3.4.*

We refer to [3] for the formal proofs of these theorems.

3.4 Randomness Re-use

A well-known property of the standard ElGamal encryption scheme is that the encryption randomness can be re-used for multiple ciphertexts of the same message. This property extends to the twisted ElGamal encryption as well. Consider two twisted ElGamal ciphertexts:

$$\mathsf{ct}_1 = (C_1 = r_1 \cdot H + x \cdot G,\ D_1 = r_1 \cdot P_1),$$

$$\mathsf{ct}_2 = (C_2 = r_2 \cdot H + x \cdot G,\ D_2 = r_2 \cdot P_2).$$

If the random scalars r_1, r_2 are generated uniformly at random from $\mathbb{Z}_p$, the decision Diffie-Hellman assumption guarantees that each of $\mathsf{ct}_1, \mathsf{ct}_2$ are computationally indistinguishable from random elements in $\mathbb{G}^2$. Namely, for random elements $H, P_1 \leftarrow_{\mathsf{R}} \mathbb{G}$ and $r_1 \leftarrow_{\mathsf{R}} \mathbb{Z}_p$, we have $(H, P_1, r_1 H, r_1 P_1) \approx_{\mathsf{c}} (H, P_1, r_1 H, V_1)$ where $V_1 \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ is a uniformly random elements in $\mathbb{Z}_p$. This shows that

$$\begin{aligned} \mathsf{ct}_1 = (C_1 = r_1 \cdot H + x \cdot G,\ D_1 = r_1 \cdot P_1) &\approx_{\mathsf{c}} (C_1 = r_1 \cdot H + x \cdot G,\ D_1 = V_1) \\ &\approx (C_1 = U_1,\ D_1 = V_1) \end{aligned}$$

where $U_1, V_1 \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. Hence, the ciphertext ct_1 is computationally indistinguishble from uniform elements in $\mathbb{G}^2$. The same argument can be applied for $\mathsf{ct}_2 = (C_2, D_2)$.

However, when generating two ciphertexts of the same message, one can optimize the size of the ciphertext. Suppose that a single random scalar $r \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ is used for the two ciphertexts ct_1 and ct_2:

$$\mathsf{ct}_1 = (C_1 = r \cdot H + x \cdot G,\ D_1 = r \cdot P_1),$$

$$\mathsf{ct}_2 = (C_2 = r \cdot H + x \cdot G,\ D_2 = r \cdot P_2).$$

Here, we have $C_1 = C_2$ and therefore, we can remove duplicate components and view the two ciphertexts as a single joint ciphertext

$$\mathsf{ct} = (C = r \cdot H + x \cdot G,\ D_1 = r \cdot P_1,\ D_2 = r \cdot P_2).$$

We claim that even when r is re-used as in the ciphertext above, the message x is secure. As before, DDH guarantees that $(H, P_1, rH, rP_1) \approx_{\mathsf{c}} (H, P_1, rH, V_1)$ for $V_1 \leftarrow_{\mathsf{R}} \mathbb{G}$ and therefore,

$$(C = r \cdot H + x \cdot G,\ D_1 = r \cdot P_1,\ D_2 = r \cdot P_2) \approx_{\mathsf{c}} \big(C = r \cdot H + x \cdot G,\ D_1 = V_1,\ D_2 = r \cdot P_2\big).$$

Now, using the DDH assumption again, $(H, P_2, rH, rP_2) \approx_{\mathsf{c}} (H, P_2, rH, V_2)$ for $V_2 \leftarrow_{\mathsf{R}} \mathbb{G}$, we can show that

$$\begin{aligned} (C = r \cdot H + x \cdot G,\ D_1 = V_1,\ D_2 = r \cdot P_2) &\approx_{\mathsf{c}} (C = r \cdot H + x \cdot G,\ D_1 = V_1,\ D_2 = V_2) \\ &\approx \big(C = U_1,\ D_1 = V_1,\ D_2 = V_2\big). \end{aligned}$$

where $U, V_1, V_2 \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. This guarantees that the joint ciphertext ct is computationally indistinguishable from uniform elements in $\mathbb{G}^3$.

In Section 6, we use this property to optimize the size of transfer instructions. The transfer instruction in the `ZK-Token` program requires that the transfer amounts be encrypted under three ElGamal public keys: the source, destination, and auditor public keys. Instead of including three independent ciphertexts in a single transfer instruction, the `ZK-Token` transfer algorithm includes only a single Pedersen commitment for the transfer amount and then generates decryption handles with respect to each of the three ElGamal public keys.

4 Zero Knowledge Arguments

In this section, we discuss zero-knowledge arguments that are used in the `ZK-Token` program. We provide the precise definitions of a zero-knowledge argument in Section 4.1. Then, in Sections 4.2, 4.3, and 4.4, we define three public-coin sigma zero-knowledge protocols that we incorporate into the `ZK-Token` program. Each of these sigma protocols can be compiled into a non-interactive argument system via the Fiat-Shamir heuristic [4]. Finally, in Section 4.5, we describe the Bulletproofs [2] range argument protocol, which we use in the `ZK-Token` program.

4.1 Zero-Knowledge Arguments of Knowledge

In full generality, zero-knowledge argument systems can be defined with respect to any class of decidable languages. However, to simplify the presentation, we define argument systems with respect to CRS-dependent languages. Specifically, let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ be an efficiently decidable ternary relation. Then a CRS-dependent language for a string $\rho \in \{0,1\}^*$ is defined as

$$\mathcal{L}_\rho = \{u \mid \exists\, w : (\rho, u, w) \in \mathcal{R}\}.$$

We generally refer to ρ as the common reference string, u as the instance of the langauge, and w as the witness for u.

For a class of CRS-dependent languages, an argument system consists of the following algorithms.

Definition 4.1 (Argument System)**.** A non-interactive argument system $_{\mathsf{AS}}$ for a CRS-dependent relation $\mathcal{R}$ consists of a tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \rho$: On input the security parameter λ, the setup algorithm returns a common reference string ρ.
- $\mathcal{P}(\sigma, u, w)$: The prover $\mathcal{P}$ is an interactive algorithm that takes in as input a common reference string σ, instance u, and witness w. It interacts with the verifier $\mathcal{V}$ according to the specification of the protocol.
- $\mathcal{V}(\sigma, u)$: The verifier $\mathcal{V}$ is an interactive algorithm that takes in as input a common reference string ρ and an instance x. It interacts with the prover $\mathcal{P}$ in the protocol and in the end, it either accepts (returns 1) or rejects (returns 0) the instance x.

We use $\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u) \rangle = 1$ to denote the event that the verifier $\mathcal{V}$ accepts the instance of the protocol. We use $\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u) \rangle \to \mathsf{tr}$ to denote the communication transacript between the prover $\mathcal{P}$ and verifier $\mathcal{V}$ during a specific execution of the protocol.

An argument system must satisfy a correctness and two security properties. The correctness property of an argument system is generally referred to as *completeness*. It states that if the prover $\mathcal{P}$ takes in as input a valid instance-witness tuple $(\rho, u, w) \in \mathcal{R}$ and follows the protocol specification, then it must be able to convince the verifier to accept.

Definition 4.2 (Completeness)**.** Let $_{\mathsf{AS}}$ be a proof system for a relation $\mathcal{R}$. Then we say that $_{\mathsf{AS}}$ satisfies perfect completeness if for any $(u, w) \in \mathcal{R}$, we have

$$\Pr\left[\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u) \rangle = 1\right] = 1,$$

where $\rho \leftarrow \mathsf{Setup}(1^\lambda)$.

The first security property that an argument system must satisfy is *soundness*, which can be defined in a number of ways. In this work, we work with *computational witness-extended emulation* as presented in Bulletproofs [2].

Definition 4.3 (Soundness [?, 5, 2])**.** Let $_{\mathsf{AS}}$ be a proof system for a relation $\mathcal{R}$. Then we say that $_{\mathsf{AS}}$ satisfies *witness-extended emulation* soundness if for all deterministic polynomial time $\mathcal{P}^*$, there exists an efficient emulator $\mathcal{E}$ such that for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left| \begin{array}{c} \Pr\left[\mathcal{A}_2(\mathsf{tr}) = 1 \,\middle|\, \begin{array}{c} \rho \leftarrow \mathsf{Setup}(1^\lambda), (u, \mathsf{st}) \leftarrow \mathcal{A}_1(\rho), \\ \mathsf{tr} \leftarrow \left\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \right\rangle \end{array} \right] - \\ \Pr\left[\mathcal{A}_2(\mathsf{tr}) = 1 \wedge (\mathsf{tr} \text{ accepting } \Rightarrow (\rho, u, w) \in \mathcal{R}) \,\middle|\, \begin{array}{c} \rho \leftarrow \mathsf{Setup}(1^\lambda), \\ (u, \mathsf{st}) \leftarrow \mathcal{A}_1(\rho), \\ (\mathsf{tr}, w) \leftarrow \mathcal{E}^{\mathcal{O}}(\rho, u) \end{array} \right] \end{array} \right| = \mathsf{negl}(\lambda),$$

where the oracle is defined as $\mathcal{O} = \left\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \right\rangle$. The oracle $\mathcal{O}$ allows the emulator $\mathcal{E}$ to rewind the protocol to a specific point and resume the protocol after reprogramming the verifier with fresh randomness.

Traditionally, the soundness condition for an argument system of knowledge requires that there exists an extractor that can use its rewinding capability to extract a valid witness from any accepting transcript of the protocol that is produced by a dishonest prover $\mathcal{P}^*$. The witness-extended emulation strengthens this traditional definition by requiring that the extractor (emulator) not only successfully extracts a valid witness, but also produces (emulates) a valid transcript of the protocol for which the verifier accepts. The value st in the definition above can be viewed as the internal state of $\mathcal{P}^*$, which can also be its randomness.

The second security property that we require from an argument system is the zero-knowledge property. All argument systems that we rely on in the `ZK-Token` program are public coin protocols that we ultimately convert into a non-interactive protocol. Therefore, we rely on the standard zero-knowledge property against honest verifiers.

Definition 4.4 (Zero-Knowledge). Let Π_{AS} be a proof system for a relation $\mathcal{R}$. Then we say that Π_{AS} satisfies *honest verifier* zero-knowledge if there exists an efficient simulator $\mathcal{S}$ such that for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we have

$$\Pr\left[(\rho, u, w) \in \mathcal{R} \wedge \mathcal{A}_1(\mathsf{tr}) = 1 \,\middle|\, \begin{array}{c} \rho \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \tau) \leftarrow \mathcal{A}_2(\rho), \\ \mathsf{tr} \leftarrow \left\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u; \tau) \right\rangle \end{array} \right]$$

$$= \Pr\left[(\rho, u, w) \in \mathcal{R} \wedge \mathcal{A}_1(\mathsf{tr}) = 1 \,\middle|\, \begin{array}{c} \rho \leftarrow \mathsf{Setup}(1^\lambda), \\ (u, w, \tau) \leftarrow \mathcal{A}_2(\rho), \\ \mathsf{tr} \leftarrow \mathcal{S}(u, \tau) \end{array} \right],$$

where ρ is the public coin randomness used by the verifier.

4.2 Zero-Balance Argument

In this section, we specify the zero-balance sigma protocol for the twisted ElGamal encryption scheme. Intuitively, the zero-balance protocol allows a prover to convince the verifier that a twisted ElGamal ciphertext encrypts the value zero under a specified public key. Formally, the zero-balance protocol captures the following language:

$$\mathcal{L}_{G,H}^{\text{zero-balance}} = \left\{ u = (P, C, D) \in \mathbb{G}^3, w = s \in \mathbb{Z}_p \,\middle|\, s \cdot P = H \wedge s \cdot D = C \right\}.$$

The language is defined with respect to two fixed generators G, H that defines the twisted ElGamal encryption scheme. The group element P corresponds to a public key in the encryption scheme and the field element s corresponds to its secret key. The elements C, D correspond to a Pedersen commitment and decryption handle that make up a single ciphertext. If the ciphertext $\mathsf{ct} = (C, D)$ is a proper encryption of zero, then its decryption must produce $\mathsf{Decrypt}(s, \mathsf{ct}) = C - s \cdot D = 0 \cdot G = 0$ and hence $s \cdot D = C$. The zero-balance argument system for the language $\mathcal{L}_{G,H}$ is specified as follows:

$$
\begin{array}{lcl}
\textbf{Prover}(x, w) & & \textbf{Verifier}(x) \\
\hline
y \leftarrow_{\mathsf{R}} \mathbb{Z}_p & & \\
Y_P \leftarrow y \cdot P & & \\
Y_D \leftarrow y \cdot D & & \\
 & \xrightarrow{\quad Y_P,\ Y_D \quad} & \\
 & & c \leftarrow_{\mathsf{R}} \mathbb{Z}_p \\
 & \xleftarrow{\quad c \quad} & \\
z \leftarrow c \cdot s + y & & \\
 & \xrightarrow{\quad z \quad} & \\
 & & z \cdot P \stackrel{?}{=} c \cdot H + Y_P \\
 & & z \cdot D \stackrel{?}{=} c \cdot C + Y_D
\end{array}
$$

The protocol follows a standard sigma protocol structure where the prover first samples a random field element $y \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. It then commits to this element by sending $Y_P = y \cdot P$ and $Y_D = y \cdot D$ to the verifier. Upon receiving a random challenge c, it provides the verifier with the masked secret key $z = c \cdot s + y$. Finally, the verifier tests the two relations $s \cdot P = H$ and $s \cdot D = C$ using the masked secret key z and the committed values Y_P and Y_D.

The zero-balance argument system above satisfies all the correctness and security properties that are specified in Section 4.1. We formally state these properties in the following theorems.

Theorem 4.5 (Completeness). *The zero-balance argument satisfies completeness 4.2.*

Theorem 4.6 (Soundness). *Suppose that $\mathbb{G}$ is a prime order group for which the discrete log relation assumption (Definition 2.1) holds. Then the zero-balance argument satisfies witness-extended emulation soundness 4.3.*

Theorem 4.7 (Zero-Knowledge). *The zero-balance argument satisfies perfect honest verifier zero-knowledge 4.4.*

We provide the formal proofs for these theorems in Section A.1.

4.3 Equality Argument

In this section, we specify the equality sigma protocol for the twisted ElGamal encryption scheme. At the start of the protocol, the prover and verifier have access to a twisted ElGamal ciphertext and a Pedersen commitment. The prover's goal is to convince the veri er that it knows a secret key and a Pedersen opening such that the ciphertext and commitment decode to the same message. Formally, the language that is captured by the protocol is speci ed as follows:

$$\mathcal{L}^{\mathsf{equality}}_{G,H} = \left\{ \begin{array}{c} u = (P_{\mathsf{EG}}, C_{\mathsf{EG}}, D_{\mathsf{EG}}, C_{\mathsf{Ped}}) \in \mathbb{G}^4, \\ w = (s, x, r) \in \mathbb{Z}_p^3 \end{array} \middle| \begin{array}{c} s \cdot P_{\mathsf{EG}} = H \wedge C_{\mathsf{EG}} - s \cdot D_{\mathsf{EG}} = x \cdot G \\ \wedge\, C_{\mathsf{Ped}} = x \cdot G + r \cdot H \end{array} \right\}.$$

The language $\mathcal{L}^{\mathsf{equality}}_{G,H}$ is speci ed by two group elements $G, H \in \mathbb{G}$ that de ne the ElGamal encryption and Pedersen commitments. The group element P_{EG} corresponds to a public key in the twisted ElGamal encryption scheme and the eld element s corresponds to its secret key. The elements $C_{\mathsf{EG}}, D_{\mathsf{EG}}$ correspond to a twisted ElGamal ciphertext and C_{Ped} corresponds to an additional Pedersen commitment. If $\mathsf{ct} = (C_{\mathsf{EG}}, D_{\mathsf{EG}})$ and C_{Ped} encode the same message x, then we must have $C_{\mathsf{EG}} - s \cdot D_{\mathsf{EG}} = x \cdot G$ and $C_{\mathsf{Ped}} = x \cdot G + r \cdot H$. The argument system for the language $\mathcal{L}^{\mathsf{equality}}_{G,H}$ is speci ed as follows:

$$\begin{array}{lcl}
\textbf{Prover}(x, w) & & \textbf{Verifier}(x) \\
\hline
y_s \leftarrow_{\mathsf{R}} \mathbb{Z}_p & & \\
y_x \leftarrow_{\mathsf{R}} \mathbb{Z}_p & & \\
y_r \leftarrow_{\mathsf{R}} \mathbb{Z}_p & & \\
 & & \\
Y_0 \leftarrow y_s \cdot P_{\mathsf{EG}} & & \\
Y_1 \leftarrow y_x \cdot G + y_s \cdot D_{\mathsf{EG}} & & \\
Y_2 \leftarrow y_x \cdot G + y_r \cdot H & & \\
 & \xrightarrow{\quad Y_0, Y_1, Y_2 \quad} & \\
 & & c \leftarrow_{\mathsf{R}} \mathbb{Z}_p \\
 & \xleftarrow{\quad\quad c \quad\quad} & \\
z_s \leftarrow c \cdot s + y_s & & \\
z_x \leftarrow c \cdot x + y_x & & \\
z_r \leftarrow c \cdot r + y_r & & \\
 & \xrightarrow{\quad z_s, z_x, z_r \quad} & \\
 & & z_s \cdot P_{\mathsf{EG}} \stackrel{?}{=} c \cdot H + Y_0 \\
 & & z_x \cdot G + z_s \cdot D_{\mathsf{EG}} \stackrel{?}{=} c \cdot C_{\mathsf{EG}} + Y_1 \\
 & & z_x \cdot G + z_r \cdot H \stackrel{?}{=} c \cdot C_{\mathsf{Ped}} + Y_2
\end{array}$$

As in the zero-balance argument protocol, the equality protocol follows a standard sigma protocol structure where the prover rst samples random eld eleemnts y_s, y_x, y_r. It then commits to these elements by sending $Y_0 = y_s \cdot P_{\mathsf{EG}}$, $Y_1 = y_x \cdot G + y_s \cdot D_{\mathsf{EG}}$, and $Y_2 = y_x \cdot G + y_r \cdot H$. Upon receiving a random challenge c, it provides the veri er with the masked secret key $z_s = c \cdot s + y$, $z_x = c \cdot x + y_x$, and $z_r = c \cdot r + y_r$. Finally, the veri er tests the three relations associated with $\mathcal{L}_{G,H}^{\mathsf{equality}}$ using the masked secret key z, and the committed values Y_0, Y_1, and Y_2.

The equality argument system above satis es all the correctness and security properties that are speci ed in Section 4.1. We formally state these properties in the following theorems.

Theorem 4.8 (Completeness)**.** *The equality argument satisfies completeness 4.2.*

Theorem 4.9 (Soundness)**.** *Suppose that $\mathbb{G}$ is a prime order group for which the discrete log relation assumption (Definition 2.1) holds. Then the equality argument satisfies witness-extended emulation soundness 4.3.*

Theorem 4.10 (Zero-Knowledge)**.** *The equality argument satisfies perfect honest verifier zero-knowledge 4.4.*

We provide the formal proofs for these theorems in Section A.2.

4.4 Ciphertext Validity Argument

In this section, we specify the ciphertext-validity sigma protocol for the twisted ElGamal encryption scheme. At the start of the protocol, the prover and veri er have access to two joint ciphertexts $\mathsf{ct}_{\mathsf{lo}} = (C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2})$ and $\mathsf{ct}_{\mathsf{hi}} = (C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2})$. The prover's goal in the protocol is to convince the veri er that it knows valid randomness and message pairs $(r_{\mathsf{lo}}, x_{\mathsf{lo}})$ and $(r_{\mathsf{hi}}, x_{\mathsf{hi}})$ that each guarantee the validity of $(C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2})$ and $\mathsf{ct}_{\mathsf{hi}} = (C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2})$. Formally, the ciphertext-validity protocol captures the following language:

$$\mathcal{L}_{G,H}^{\mathsf{ct\text{-}validity}} = \left\{ \begin{array}{c} u = (P_1, P_2, C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2}, C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2}) \in G^8, \\ w = (r_{\mathsf{lo}}, x_{\mathsf{lo}}, r_{\mathsf{hi}}, x_{\mathsf{hi}}) \in \mathbb{Z}_p^4 \end{array} \;\middle|\; \begin{array}{c} C_{\mathsf{lo}} = r_{\mathsf{lo}} \cdot H + x_{\mathsf{lo}} \cdot G \\ C_{\mathsf{hi}} = r_{\mathsf{hi}} \cdot H + x_{\mathsf{hi}} \cdot G \\ D_{\mathsf{lo},1} = r_{\mathsf{lo}} \cdot P_1 \\ D_{\mathsf{lo},2} = r_{\mathsf{lo}} \cdot P_2 \\ D_{\mathsf{hi},1} = r_{\mathsf{hi}} \cdot P_1 \\ D_{\mathsf{hi},2} = r_{\mathsf{hi}} \cdot P_2 \end{array} \right\}.$$

The formal speci cation of the protocol is given as follows:

$$
\begin{array}{lcl}
\textbf{Prover}(x, w) & & \textbf{Verifier}(x) \\
\hline
 & & w \leftarrow_{\mathsf{R}} \mathbb{Z}_p \\
 & \xleftarrow{\quad t \quad} & \\
r \leftarrow r_{\mathsf{lo}} + t \cdot r_{\mathsf{hi}} & & \\
x \leftarrow x_{\mathsf{hi}} + t \cdot x_{\mathsf{hi}} & & \\
 & & \\
y_r \leftarrow_{\mathsf{R}} \mathbb{Z}_p & & \\
y_x \leftarrow_{\mathsf{R}} \mathbb{Z}_p & & \\
 & & \\
Y_0 \leftarrow y_r \cdot H + y_x \cdot G & & \\
Y_1 \leftarrow y_r \cdot P_1 & & \\
Y_2 \leftarrow y_r \cdot P_2 & & \\
 & \xrightarrow{\quad Y_0, Y_1, Y_2 \quad} & \\
 & & c \leftarrow_{\mathsf{R}} \mathbb{Z}_p \\
 & \xleftarrow{\quad c \quad} & \\
z_r \leftarrow c \cdot r + y_r & & \\
z_x \leftarrow c \cdot x + y_x & & \\
 & \xrightarrow{\quad z_r, z_x \quad} & \\
 & & C \leftarrow C_{\mathsf{lo}} + t \cdot C_{\mathsf{hi}} \\
 & & D_1 \leftarrow D_{\mathsf{lo},1} + t \cdot D_{\mathsf{hi},1} \\
 & & D_2 \leftarrow D_{\mathsf{lo},2} + t \cdot D_{\mathsf{hi},2} \\
 & & \\
 & & z_r \cdot H + z_x \cdot G \stackrel{?}{=} c \cdot C + Y_0 \\
 & & z_r \cdot P_1 \stackrel{?}{=} c \cdot D_1 + Y_1 \\
 & & z_r \cdot P_2 \stackrel{?}{=} c \cdot D_2 + Y_2
\end{array}
$$

At the start of the protocol, the verifier sends the prover a challenge value $t \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. The prover uses t to combine its witnesses $r \leftarrow r_{\mathsf{lo}} + t \cdot r_{\mathsf{hi}}$ and $x \leftarrow x_{\mathsf{hi}} + t \cdot x_{\mathsf{hi}}$. At this point of the protocol, the prover and the verifier proceeds in a standard sigma protocol where the prover samples random scalar elements y_r, y_x and commits to them by sending $Y_0 = y_r \cdot H + y_x \cdot G$, $Y_1 = y_r \cdot P_1$, and $Y_2 = y_r \cdot P_2$ to the verifier. Upon receiving another challenge c, it provides the verifier with the masked randomness and message $z_r = c \cdot r + y_r$ and $z_x = c \cdot x + y_x$. Finally, the verifier tests the relations $z_r \cdot H + z_x \cdot G = c \cdot C + Y_0$, $z_r \cdot P_1 = c \cdot D_1 + Y_1$, and $z_r \cdot P_2 = c \cdot D_2 + Y_2$.

The ciphertext validity argument above satisfies all the correctness and security properties that are specified in Section 4.1. We formally state these properties in the following theorems.

Theorem 4.11 (Completeness). *The ciphertext validity argument satisfies completeness 4.2.*

Theorem 4.12 (Soundness). *Suppose that $\mathbb{G}$ is a prime order group for which the discrete log relation assumption (Definition 2.1) holds. Then the ciphertext valdity argument satisfies witness-extended emulation soundness 4.3.*

Theorem 4.13 (Zero-Knowledge). *The ciphertext validity argument satisfies perfect honest verifier zero-knowledge 4.4.*

We provide the formal proofs for these theorems in Section A.3.

4.5 Range Arguments

The final argument system that we require for the `ZK-Token` program is a range argument for Pedersen commitments. Such an argument system is defined with respect to the following language:

$$\mathcal{L}^{\mathsf{range}}_{G,H,\ell,u} = \left\{ \ x = C, \ w = (x, r) \ \middle| \ C = x \cdot G + r \cdot H \wedge x \in [\ell, u] \ \right\}.$$

There are a number of ways to construct a zero-knowledge argument system for the language $\mathcal{L}^{\mathsf{range}}_{G,H,\ell,u}$. In the `ZK-Token` program, we use the Bulletproof system by Boneh et al. [2] which have great scalability features. When compiled using Fiat-Shamir [4] heuristic, Bulletproofs result in a non-interactive argument system where the proof size scales logarithmically in the bit-length of the range bounds ℓ and u. Furthermore, Bulletproofs supports proof aggregation, meaning that a prover can generate a compact argument for multiple instances of the language $\mathcal{L}^{\mathsf{range}}_{G,H,\ell,u}$ at once.

To incorporate Bulletproofs in the protocol specification formally, we summarize the result of [2] in the theorem below. First, we define the following extension of the language $\mathcal{L}^{\mathsf{range}}_{G,H,\ell,u}$:

$$\mathcal{L}^{\mathsf{range\text{-}agg}}_{G,H,\{\ell\}_{i\in[N]},\{u\}_{i\in[N]},N} = \left\{ \ x = \{C_i\}_{i\in[N]}, \ w = \{(x_i, r_i)\}_{i\in[N]} \ \middle| \ \begin{array}{l} C_i = x_i \cdot G + r_i \cdot H \ \wedge \\ x_i \in [\ell_i, u_i] \quad \forall i \in [N] \end{array} \ \right\}.$$

Boneh et al. [2] proves the following:

Theorem 4.14 (Bulletproofs [2]). *There exists a non-interactive zero-knowledge argument system for the language $\mathcal{L}^{\mathsf{range}}$ that satisfies completeness (Definition 4.2), soundness (Definition 4.3), and zero-knowledge (Definition 4.4).*

In the protocol description in Section 6, we incorporate the Bulletproofs protocol in a black-box way.

5 Confidential Payment System

5.1 Algorithm Specification

In this section, we formalize confidential payment systems for smart contract platforms. A confidential payment system is defined with respect to a set of instructions that are processed by a designated smart contract that lives on a blockchain. For each of the instructions, a confidential payment protocol must specify two algorithms:

- The instruction generation algorithm that specifies how a client generates the instruction.
- The instruction processing algorithm that specifies how a contract verifies the instruction and modifies its state accordingly.

The precise list of instructions for a con dential payment system may vary for di erent protocols. To keep the de nition as simple and general as possible, we de ne a con dential payment system with respect to a minimal set of core instructions such as $\mathsf{Deposit}$, $\mathsf{Withdraw}$, and $\mathsf{Transfer}$ that contain cryptographic components most relevant for security. Therefore, our de nition excludes the following aspects of the `ZK-Token` program.

- The mint con guration instructions such as $\mathsf{ConfigureMint}$ and $\mathsf{UpdateAuditor}$ that are speci c to the Solana programming model.
- The instructions $\mathsf{ApplyPendingBalance}$, $\mathsf{EnableBalanceCredits}$, and $\mathsf{DisableBalanceCredits}$ that allows users to manage encrypted balances as *pending* and *available* ciphertexts.

We refer the readers to part 1 for more details on these instructions. Formally, we model a con dential payment system in the following de nition.

Definition 5.1 (Con dential Payment System)**.** A con dential payment system $\mathcal{CPS}$ is de ned with respect to the following:

- A public key space $\mathcal{PK}$, which speci es the address of accounts.
- A public key encryption scheme $_{\mathsf{PKE}}$ = $(\mathsf{EncKeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with an associated encryption key space $\mathcal{EK}$, message space $\mathbb{N}$, and ciphertext space $\mathcal{CT}$.
- A space of allowed balances for accounts $\mathcal{B} = [B_{\mathsf{min}}, B_{\mathsf{max}}] \subseteq \mathbb{N}$.

The algorithms for $\mathcal{CPS}$ is speci ed with respect to a set of instructions $\mathcal{I}_{\mathcal{CPS}} = \{\mathsf{OpenAccount}, \mathsf{CloseAccount}, \mathsf{Deposit}, \mathsf{Withdraw}, \mathsf{Transfer}\}$ that are processed by a contract program $\mathcal{P}_{\mathcal{CPS}}$. The contract program $\mathcal{P}_{\mathcal{CPS}}$ maintains a look-up table $\mathcal{T}_{\mathcal{CPS}} : \mathcal{PK} \to (\mathcal{EK}, \mathcal{CT})$ that maps public keys to an encryption key and ciphertext. On setup, it rst initializes $\mathcal{T}_{\mathcal{CPS}}$ by sampling $\mathsf{pk}_{\mathsf{auditor}} \leftarrow_{\mathsf{R}} \mathcal{PK}$, $(\mathsf{ek}_{\mathsf{auditor}}, \mathsf{dk}_{\mathsf{auditor}}) \leftarrow \mathsf{EncKeyGen}(1^\lambda)$ and then adding $(\mathsf{pk}_{\mathsf{auditor}} \mapsto \mathsf{ek}_{\mathsf{auditor}}, \mathsf{dk}_{\mathsf{auditor}})$ to $\mathcal{T}_{\mathcal{CPS}}$.

For each instruction in $\mathcal{I}_{\mathcal{CPS}}$, the con dential payment system $\mathcal{CPS}$ must specify an algorithm that is run by the client to generate it and an algorithm that is run by the contract program $\mathcal{P}_{\mathcal{CPS}}$ to process it.

- $\mathsf{OpenAccount}$
  - $\mathsf{GenOpenAccount}(\mathsf{pk}, \mathsf{ek}) \to \mathsf{inst}_{\mathsf{OpenAccount}}$: This is a user algorithm that takes in as input a public key pk and an encryption key ek. It returns an open account instruction $\mathsf{inst}_{\mathsf{OpenAccount}}$.
  - $\mathsf{ProcessOpenAccount}(\mathsf{inst}_{\mathsf{OpenAccount}}) \to 0/1$: This is a $\mathcal{P}_{\mathcal{CPS}}$ algorithm that takes in as input an open account instruction $\mathsf{inst}_{\mathsf{OpenAccount}}$. It either accepts and processes the instruction (returns 1), or rejects and does nothing (returns 0).
- $\mathsf{CloseAccount}$
  - $\mathsf{GenCloseAccount}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{ct}_{\mathsf{balance}}) \to \mathsf{inst}_{\mathsf{CloseAccount}}$: This is a user algorithm that takes in as input a public key pk, encryption key ek, decryption key dk, and encrypted balance $\mathsf{ct}_{\mathsf{balance}}$. It returns a close account instruction $\mathsf{inst}_{\mathsf{CloseAccount}}$.

- $\mathsf{ProcessCloseAccount}(\mathsf{inst}_{\mathsf{CloseAccount}}) \to 0/1$: This is a $\mathcal{P}_{\mathcal{CPS}}$ algorithm that takes in as input a close account instruction $\mathsf{inst}_{\mathsf{CloseAccount}}$. It either accepts and processes the instruction (returns 1) or rejects and does nothing (returns 0).

- Deposit
  - $\mathsf{GenDeposit}(\mathsf{pk}, \mathsf{ek}, \mathsf{amt}_{\mathsf{deposit}}) \to \mathsf{inst}_{\mathsf{Deposit}}$: This is a user algorithm that takes in as input a public key pk, encryption key ek, and deposit amount $\mathsf{amt}_{\mathsf{deposit}}$. It returns a deposit instruction $\mathsf{inst}_{\mathsf{Deposit}}$.
  - $\mathsf{ProcessDeposit}(\mathsf{inst}_{\mathsf{Deposit}}) \to 0/1$: This is a $\mathcal{P}_{\mathcal{CPS}}$ algorithms takes in as input a deposit instruction $\mathsf{inst}_{\mathsf{Deposit}}$. It either accepts and processes the instruction (returns 1) or rejects and does nothing (returns 0).

- Withdraw
  - $\mathsf{GenWithdraw}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{amt}_{\mathsf{withdraw}}, \mathsf{ct}_{\mathsf{balance}}) \to \mathsf{inst}_{\mathsf{Withdraw}}$: This is a user algorithm that takes in as input a public key pk, encryption key ek, decryption key dk, withdraw amount $\mathsf{amt}_{\mathsf{withdraw}}$, and account data $\mathsf{ct}_{\mathsf{balance}}$. It returns a withdraw instruction $\mathsf{inst}_{\mathsf{Withdraw}}$.
  - $\mathsf{ProcessWithdraw}(\mathsf{inst}_{\mathsf{Withdraw}}) \to 0/1$: This is a $\mathcal{P}_{\mathcal{CPS}}$ algorithm that takes in as input a withdraw instruction $\mathsf{inst}_{\mathsf{Withdraw}}$. It either accepts and processes the instruction (returns 1) or rejects and does nothing (returns 0).

- Transfer
  - $\mathsf{GenTransfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_{\mathsf{tran}}) \to \mathsf{inst}_{\mathsf{Transfer}}$: This is a user algorithm that takes in as input a source public key $\mathsf{pk}_{\mathsf{source}}$, source encryption key $\mathsf{ek}_{\mathsf{source}}$, source decryption key $\mathsf{dk}_{\mathsf{source}}$, source account balance $\mathsf{ct}_{\mathsf{source}}$, destination public key $\mathsf{pk}_{\mathsf{dest}}$, destination encryption key $\mathsf{ek}_{\mathsf{dest}}$, auditor encryption key $\mathsf{ek}_{\mathsf{auditor}}$, and transfer amount $\mathsf{amt}_{\mathsf{tran}}$. It returns a transfer instruction $\mathsf{inst}_{\mathsf{Transfer}}$.
  - $\mathsf{ProcessTransfer}(\mathsf{inst}_{\mathsf{Transfer}}) \to 0/1$: This is a $\mathcal{P}_{\mathcal{CPS}}$ algorithm that takes in as input a transfer instruction $\mathsf{inst}_{\mathsf{Transfer}}$. It either accepts and processes the instruction (returns 1) or rejects and does nothing (returns 0).

In addition to the instruction algorithms above, a con dential payment system $\mathcal{CPS}$ must additionally specify the following client decryption algorithms:

- $\mathsf{DecryptBalance}(\mathsf{dk}, \mathsf{ct}_{\mathsf{balance}}) \to \mathsf{amt}_{\mathsf{balance}}$: On input a public key pk, decryption key dk, and encrypted balance $\mathsf{ct}_{\mathsf{balance}}$, the algorithm returns a balance $\mathsf{amt}_{\mathsf{balance}}$.

- $\mathsf{DecryptTransfer}(\mathsf{dk}, \mathsf{inst}_{\mathsf{Transfer}}) \to \mathsf{amt}_{\mathsf{tran}}$: On input a public key pk, decryption key dk, and transfer instruction $\mathsf{inst}_{\mathsf{Transfer}}$, the algorithm returns a transfer amount $\mathsf{amt}_{\mathsf{tran}}$.

Discussion. We note that in the de nition above, the instructions CloseAccount, Withdraw, and Transfer instructions take in decryption keys as well as encrypted balances. As we discuss in the soundness security requirement (De nition 5.5) below, these instructions must be processed by the contract program only when certain state conditions are met. For instance, a close account instruction is legal only when the associated account contains zero balance. In the `ZK-Token` program, the three instructions CloseAccount, Withdraw, and Transfer instructions include zero-knowledge arguments that certify the legality of these instructions. Therefore, in the definition above, we make the generation algorithms for these instructions to take as input the decryption key and the currently encrypted balance associated with the relevant accounts. These components are used for generating the relevent zero-knowledge arguments for each of these instructions.

Another notable piece of the definition above is the GenTransfer algorithm. It takes as input an additional encryption key associated with the auditor. This component captures the auditability feature of the `ZK-Token` program. The decryption correctness (Definition 5.4) property below requires that users be able to recover transfer amounts from any properly generated transfer instructions using either the destination or auditor keys. The soundness condition (Definition 5.5) below requires that no adversarial user can generate a transfer instruction that is not decryptable by a designated auditor key.

5.2 Correctness

We require a confidential payment system to satisfy two correctness properties. The first property is *state* correctness. This property requires that a confidential payment system behaves essentially like a standard (non-confidential) payment system. For example, we require that if a valid GenTransfer and ProcessTransfer algorithms are used to generate and process a transfer instruction, then this transfer of funds must be reflected in the state $\mathcal{T}_{\mathcal{CPS}}$ that is maintained by $\mathcal{P}_{\mathcal{CPS}}$. Although this notion of correctness fits our most basic intuition of a confidential payment system, capturing this property formally requires some effort. To capture state correctness cleanly, we first define the notion of an ideal payment system $\mathcal{IPS}$ and use it to define correctness precisely.

Definition 5.2 (Ideal Payment Processor). An ideal payment processor $\mathcal{P}_{\mathcal{IPP}}$ for a confidential payment system $\mathcal{CPS}$ with public key space $\mathcal{PK}$ and balance space $\mathcal{B}$ is a stateful program that is defined with respect to the same public key space $\mathcal{PK}$ and instructions $\mathcal{I}_{\mathcal{CPS}}$ of $\mathcal{CPS}$. It maintains a look-up table $\mathcal{T}_{\mathcal{IPP}} : \mathcal{PK} \to \mathcal{B}$ that maps public keys to account balances. $\mathcal{P}_{\mathcal{IPP}}$ processes each instruction in $\mathcal{I}_{\mathcal{CPS}}$ as follows:

- $\mathsf{OpenAccount}(\mathsf{pk})$: On input a public key pk, the program $\mathcal{P}_{\mathcal{IPP}}$ adds an entry $(\mathsf{pk} \mapsto 0)$ to $\mathcal{T}_{\mathcal{IPP}}$.

- $\mathsf{CloseAccount}(\mathsf{pk})$: On input a public key pk, the program $\mathcal{P}_{\mathcal{IPP}}$ checks if an entry $(\mathsf{pk} \mapsto 0)$ exists in $\mathcal{T}_{\mathcal{IPP}}$. If so, it removes $(\mathsf{pk} \mapsto 0)$ in $\mathcal{T}_{\mathcal{IPP}}$.

- $\mathsf{Deposit}(\mathsf{pk}, \mathsf{amt}_{\mathsf{deposit}})$: On input a public key pk and deposit amount $\mathsf{amt}_{\mathsf{deposit}}$, the program $\mathcal{P}_{\mathcal{IPP}}$ first checks if an entry $(\mathsf{pk}, \mathsf{amt}_{\mathsf{balance}})$ exists in $\mathcal{T}$ and that $\mathsf{amt}_{\mathsf{balance}} + \mathsf{amt}_{\mathsf{deposit}} \in \mathcal{B}$. If so, it replaces the entry with $(\mathsf{pk}, \mathsf{amt}_{\mathsf{balance}} + \mathsf{amt}_{\mathsf{deposit}})$.

- $\mathsf{Withdraw}(\mathsf{pk}, \mathsf{amt}_{\mathsf{withdraw}})$: On input a public key pk and withdraw amount $\mathsf{amt}_{\mathsf{withdraw}}$, the program $\mathcal{P}_{\mathcal{IPP}}$ first checks if an entry $(\mathsf{pk}, \mathsf{amt}_{\mathsf{balance}})$ exists in $\mathcal{T}$ and that $\mathsf{amt}_{\mathsf{balance}} \geq \mathsf{amt}_{\mathsf{withdraw}}$. If so, it replaces the entry with $(\mathsf{pk}, \mathsf{amt}_{\mathsf{balance}} - \mathsf{amt}_{\mathsf{withdraw}})$.

- $\mathsf{Transfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{amt}_{\mathsf{tran}})$: On input a source public key $\mathsf{pk}_{\mathsf{source}}$, destination public key $\mathsf{pk}_{\mathsf{dest}}$, and transfer amount $\mathsf{amt}_{\mathsf{tran}}$, the program $\mathcal{P}_{\mathcal{IPP}}$ first checks if entries $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{amt}_{\mathsf{balance,source}})$, $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{amt}_{\mathsf{balance,dest}})$ exist in $\mathcal{T}$ and that $\mathsf{amt}_{\mathsf{balance,source}} \geq \mathsf{amt}_{\mathsf{tran}}$. If so, the program replaces each of these entries with $(\mathsf{pk}_{\mathsf{source}}, \mathsf{amt}_{\mathsf{balance,source}} - \mathsf{amt}_{\mathsf{tran}})$ and $(\mathsf{pk}_{\mathsf{dest}}, \mathsf{amt}_{\mathsf{balance,dest}} + \mathsf{amt}_{\mathsf{tran}})$.

To de ne state correctness, we de ne an experiment between an adversary and challenger. The adversary is given access to the generation oracles for each instruction in $\mathcal{I}_{\mathcal{CPS}}$. For each call to one of these oracles, the challenger submits corresponding instructions to both the $\mathcal{CPS}$ contract program $\mathcal{P}_{\mathcal{CPS}}$ and the ideal payment processor $\mathcal{P}_{\mathcal{IPP}}$. At the end of the experiment, the challenger compares the state of $\mathcal{P}_{\mathcal{CPS}}$ and $\mathcal{P}_{\mathcal{IPP}}$. If there exists an account (public key) for which the stored balances are di erent, then the adversary breaks correctness and wins in the correctness experiment. We say that a con dential payment system is correct if no adversary wins in the correctness experiment.

Definition 5.3 (State Correctness)**.** Let $\mathcal{CPS}$ be a con dential payment system and let $\mathcal{P}_{\mathcal{IPP}}$ be a corresponding ideal payment system. For a security parameter $\lambda \in \mathbb{N}$ and an adversary $\mathcal{A}$, we de ne the correctness experiment $\mathsf{EXP}_{\mathsf{correctness}}[\lambda, \mathcal{A}]$ as follows:

1. Throughout the experiment, the adversary $\mathcal{A}$ is provided oracle access to a key generation oracle:

   - $\mathcal{O}_{\mathsf{KeyGen}}()$: On its invocation, the challenger sample $\mathsf{pk} \leftarrow_{\mathsf{R}} \mathcal{PK}$ and $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{EncKeyGen}(1^\lambda)$. It stores the mapping $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{dk})$ in $\mathcal{T}_{\mathsf{keys}}$ and returns $(\mathsf{pk}, \mathsf{ek}, \mathsf{dk})$ to $\mathcal{A}$.

   In addition, $\mathcal{A}$ is provided oracle access to each of the instruction generation algorithms of $\mathcal{CPS}$ as speci ed in De nition 5.1. For each of $\mathcal{A}$'s queries to these oracles, the challenger responds as follows:

   - $\mathsf{GenOpenAccount}(\mathsf{pk}, \mathsf{ek})$:
     (a) If an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{dk})$ does not exist in $\mathcal{T}_{\mathsf{keys}}$, then the challenger returns $\perp$. Otherwise, it computes $\mathsf{inst}_{\mathsf{OpenAccount}} \leftarrow \mathsf{GenOpenAccount}(\mathsf{pk}, \mathsf{ek})$ and feeds $\mathsf{inst}_{\mathsf{OpenAccount}}$ to $\mathcal{P}_{\mathcal{CPS}}$.
     (b) It then submits $\mathsf{OpenAccount}(\mathsf{pk})$ to $\mathcal{P}_{\mathcal{IPP}}$.
   - $\mathsf{GenCloseAccount}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{ct}_{\mathsf{balance}})$:
     (a) If an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{dk})$ does not exist in $\mathcal{T}_{\mathsf{keys}}$ or $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ does not exist in $\mathcal{T}_{\mathcal{CPS}}$, then the challenger returns $\perp$. Otherwise, it computes $\mathsf{inst}_{\mathsf{CloseAccount}} \leftarrow \mathsf{GenCloseAccount}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{ct}_{\mathsf{balance}})$ and feeds $\mathsf{inst}_{\mathsf{CloseAccount}}$ to $\mathcal{P}_{\mathcal{CPS}}$.
     (b) It then submits $\mathsf{CloseAccount}(\mathsf{pk})$ to $\mathcal{P}_{\mathcal{IPP}}$.
   - $\mathsf{GenDeposit}(\mathsf{pk}, \mathsf{amt}_{\mathsf{deposit}}, \mathsf{ek})$
     (a) If an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{dk})$ does not exist in $\mathcal{T}_{\mathsf{keys}}$, then the challenger returns $\perp$. Otherwise, it computes $\mathsf{inst}_{\mathsf{Deposit}} \leftarrow \mathsf{GenDeposit}(\mathsf{pk}, \mathsf{amt}_{\mathsf{deposit}}, \mathsf{ek})$ and feeds $\mathsf{inst}_{\mathsf{Deposit}}$ to $\mathcal{P}_{\mathcal{CPS}}$.
     (b) It then submits $\mathsf{GenDeposit}(\mathsf{pk}, \mathsf{amt}_{\mathsf{deposit}})$ to $\mathcal{P}_{\mathcal{IPP}}$.
   - $\mathsf{GenWithdraw}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{amt}_{\mathsf{withdraw}}, \mathsf{ct}_{\mathsf{balance}})$
     (a) If an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{dk})$ does not exist in $\mathcal{T}_{\mathsf{keys}}$ or $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ in $\mathcal{T}_{\mathcal{CPS}}$, then the challenger returns $\perp$. Otherwise, it computes $\mathsf{inst}_{\mathsf{Withdraw}} \leftarrow \mathsf{GenWithdraw}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{amt}_{\mathsf{withdraw}}, \mathsf{ct}_{\mathsf{balance}})$ and feeds $\mathsf{inst}_{\mathsf{Withdraw}}$ to $\mathcal{P}_{\mathcal{CPS}}$.
     (b) It then submits $\mathsf{GenWithdraw}(\mathsf{pk}, \mathsf{amt}_{\mathsf{withdraw}})$ to $\mathcal{P}_{\mathcal{IPP}}$.
   - $\mathsf{GenTransfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_{\mathsf{tran}})$

(a) If any of the source and destination keys are not consistent with the previous outputs of $\mathcal{O}_{\mathsf{KeyGen}}$ or $\mathsf{ct}_{\mathsf{source}}$ is not consistent with $\mathcal{T}_{\mathcal{CPS}}$, the challenger returns $\perp$. Otherwise, it computes $\mathsf{inst}_{\mathsf{Transfer}} \leftarrow \mathsf{GenTransfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_{\mathsf{tran}})$ and feeds $\mathsf{inst}_{\mathsf{Transfer}}$ to $\mathcal{P}_{\mathcal{CPS}}$.

(b) It then submits $\mathsf{Transfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{amt}_{\mathsf{tran}})$ to $\mathcal{P}_{\mathcal{IPP}}$.

2. At the end of the experiment, the challenger compares the state of $\mathcal{P}_{\mathcal{CPS}}$ and $\mathcal{P}_{\mathcal{IPP}}$. Namely, for each entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}}) \in \mathcal{T}_{\mathcal{CPS}}$, it looks up the corresponding decryption key dk in $\mathcal{T}_{\mathsf{keys}}$, computes $\mathsf{amt}_{\mathsf{balance}} \leftarrow \mathsf{DecryptBalance}(\mathsf{dk}, \mathsf{ct}_{\mathsf{balance}})$, and veri es that $(\mathsf{pk} \mapsto \mathsf{amt}_{\mathsf{balance}}) \in \mathcal{T}_{\mathcal{IPP}}$. If there exists an entry in $\mathcal{T}_{\mathcal{CPS}}$ for which this condition does not hold, then it returns 1. Otherwise, it returns 0.

We say that a con dential payment system $\mathcal{CPS}$ is correct if for any λ and adversary $\mathcal{A}$, we have

$$\Pr\left[\mathsf{EXP}_{\mathsf{correctness}}[\lambda, \mathcal{A}] = 1\right] = \mathsf{negl}(\lambda).$$

The second correctness property that we require is the transfer decryption correctness. This property simply requires that any properly generated transfer instruction via GenTransfer decrypts to a correct transfer amount via DecryptTransfer. This property, for instance, is important for auditability of the `ZK-Token` program.

Definition 5.4 (Decryption Correctness)**.** Let $\mathcal{CPS}$ be a con dential payment system with respect to a public key space $\mathcal{PK}$, public key encryption $_{\mathsf{PKE}}$, and balance space $\mathcal{B}$. We say that $\mathcal{CPS}$ satis es decryption correctness if for any $\mathsf{amt}_{\mathsf{tran}} \in \mathcal{B}$, we have

$$\Pr\left[\mathsf{DecryptTransfer}(\mathsf{dk}_{\mathsf{dest}}, \mathsf{inst}_{\mathsf{Transfer}}) = \mathsf{DecryptTransfer}(\mathsf{dk}_{\mathsf{auditor}}, \mathsf{inst}_{\mathsf{Transfer}}) = \mathsf{amt}_{\mathsf{tran}}\right] = 1,$$

where $\mathsf{pk}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}} \leftarrow_{\mathsf{R}} \mathcal{PK}$, $(\mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}) \leftarrow \mathsf{EncKeyGen}(1^\lambda)$, $(\mathsf{ek}_{\mathsf{dest}}, \mathsf{dk}_{\mathsf{dest}}) \leftarrow \mathsf{EncKeyGen}(1^\lambda)$, $(\mathsf{ek}_{\mathsf{auditor}}, \mathsf{dk}_{\mathsf{auditor}}) \leftarrow \mathsf{EncKeyGen}(1^\lambda)$, and $\mathsf{inst}_{\mathsf{Transfer}} \leftarrow \mathsf{GenTransfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_{\mathsf{tran}})$.

5.3 Security

For security, we require that a con dential payment system satisfy two security properties. The rst property is soundness, which prevents the contract program $\mathcal{P}_{\mathcal{CPS}}$ from accepting CloseAccount, Withdraw, or Transfer instructions that are generated illegally. The disallowed scenarios that are captured by the soundness condition includes the following:

- An owner of an account must not be able to close the account unless the associated encrypted balance is zero.
- An owner of an account must not be able to withdraw or transfer an amount that is greater than the encrypted balance in the account.
- A user must not be able to generate a transfer instruction that cannot be decrypted by the owners of the destination account or the auditor.

We capture soundness using a security experiment between an adversary and challenger. Throughout the experiment, the adversary is provided access to a number of oracles that allow the adversary to open new accounts, submit instructions of its choosing, and read the state $\mathcal{T}_{\mathcal{CPS}}$ of the contract program. At the end of the experiment, the adversary outputs an instruction that applies to one of the disallowed scenarios above. The adversary wins in the experiment if the instruction that it outputs is accepted by the contract program $\mathcal{P}_{\mathcal{CPS}}$.

Definition 5.5 (Soundness)**.** Let $\mathcal{CPS}$ be a con dential payment system with an associated public key space $\mathcal{PK}$ and public key encryption scheme $\ _{\mathsf{PKE}} = (\mathsf{EncKeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with $\mathcal{EK}$, $\mathcal{M}$, and $\mathcal{CT}$. For a security parameter λ and an adversary $\mathcal{A}$, we de ne the soundness security experiment $\mathsf{EXP}_{\mathsf{soundness}}[\lambda, \mathcal{A}]$ for $\mathcal{CPS}$ as follows:

- Throughout the experiment, the challenger maintains a look-up table $\mathcal{T}_{\mathsf{dk}}$. Using $\mathcal{P}_{\mathcal{CPS}}$ that it executes internally, the challenger provides $\mathcal{A}$ access to the following set of oracles:
  - $\mathcal{O}_{\mathsf{OpenAccount}}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk})$: If an entry with pk already exists in $\mathcal{T}_{\mathcal{CPS}}$ or the encryption-decryption key pair $(\mathsf{ek}, \mathsf{dk})$ is not a valid pair for $\ _{\mathsf{PKE}}$, then the challenger returns $\perp$ and does nothing. Otherwise, it records the mapping $(\mathsf{ek}, \mapsto \mathsf{dk})$ in $\mathcal{T}_{\mathsf{dk}}$, computes $\mathsf{inst}_{\mathsf{OpenAccount}} \leftarrow \mathsf{GenOpenAccount}(\mathsf{pk}, \mathsf{ek})$, and submits $\mathsf{inst}_{\mathsf{OpenAccount}}$ to $\mathcal{P}_{\mathcal{CPS}}$.
  - $\mathcal{O}_{\mathsf{Instruction}}(\mathsf{inst})$: If inst is an open account instruction, then the challenger returns $\perp$ and does nothing. Otherwise, it submits inst to $\mathcal{P}_{\mathcal{CPS}}$ and relays the output to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{Read}}()$: The challenger provides $\mathcal{A}$ the entire state $\mathcal{T}_{\mathcal{CPS}}$ that is maintained by $\mathcal{P}_{\mathcal{CPS}}$.
- At the end of the experiment, the adversary $\mathcal{A}$ returns one of the following:
  - *Close account forgery*: The adversary $\mathcal{A}$ returns a public key pk and close account instruction $\mathsf{inst}_{\mathsf{CloseAccount}}$. If an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct})$ does not exist in $\mathcal{T}_{\mathcal{CPS}}$ or an entry $(\mathsf{pk} \mapsto \mathsf{dk})$ does not exist in $\mathcal{T}_{\mathsf{dk}}$, the challenger returns 0 as the output of the experiment. Otherwise, it checks whether $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{ct}) > 0$ and $\mathsf{ProcessCloseAccount}(\mathsf{inst}_{\mathsf{CloseAccount}}) = 1$. If this is the case, then it returns 1 as the output of the experiment. Otherwise, it returns 0.
  - *Invalid account*: The adversary $\mathcal{A}$ speci es a public key pk. If an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct})$ does not exist in $\mathcal{T}_{\mathcal{CPS}}$ or an entry $(\mathsf{pk} \mapsto \mathsf{dk})$ does not exist in $\mathcal{T}_{\mathsf{dk}}$, the challenger returns 0 as the output of the experiment. Otherwise, it checks whether $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{ct}) < 0$ or $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{ct}) = \perp$. If this is the case, then it returns 1 as the output of the experiment. Otherwise, it returns 0.
  - *Non-decryptable transfer instruction*: The adversary $\mathcal{A}$ speci ed a transfer instruction $\mathsf{inst}_{\mathsf{Transfer}}$ and three public keys $\mathsf{pk}_{\mathsf{source}}$, $\mathsf{pk}_{\mathsf{dest}}$ and $\mathsf{pk}_{\mathsf{auditor}}$. If entries $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{ek}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}})$, $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{ek}_{\mathsf{dest}}, \mathsf{ct}_{\mathsf{dest}})$ or $(\mathsf{pk}_{\mathsf{auditor}} \mapsto \mathsf{ek}_{\mathsf{auditor}}, \mathsf{ct}_{\mathsf{auditor}})$ do not exist in $\mathcal{T}_{\mathsf{dk}}$, then the challenger returns 0 as the output of the experiment. Otherwise, it checks the following:
    - The challenger submits $\mathsf{inst}_{\mathsf{Transfer}}$ to $\mathcal{P}_{\mathcal{CPS}}$ and veri es that $\mathcal{P}_{\mathcal{CPS}}$ does process $\mathsf{inst}_{\mathsf{Transfer}}$.
    - It calculates the change of balances in $\mathsf{pk}_{\mathsf{source}}$ and $\mathsf{pk}_{\mathsf{dest}}$ accounts and veri es that they are equal.
    - Let $\mathsf{amt}_{\mathsf{tran}}$ be the change of balance amount in the source and destination accounts. Then, the challenger veri es that $\mathsf{DecryptTransfer}(\mathsf{dk}_{\mathsf{dest}}, \mathsf{inst}_{\mathsf{Transfer}}) = \mathsf{DecryptTransfer}(\mathsf{dk}_{\mathsf{auditor}}, \mathsf{inst}_{\mathsf{Transfer}}) = \mathsf{amt}_{\mathsf{tran}}$.

If any one of the conditions above fail, then the challenger returns 1 as the output of the experiment. Otherwise, it outputs 0.

We say that a con dential payment system $\mathcal{CPS}$ satis es soundness if for all e cient adversaries $\mathcal{A}$, we have

$$\mathsf{EXP}_{\mathsf{soundness}}[\lambda, \mathcal{A}] = \mathsf{negl}(\lambda).$$

The second security property that we require from $\mathcal{CPS}$ is con dentiality. Intuitively, con dentiality requires that a transfer instruction does not reveal any information about the transfer amount. We capture con dentiality using an experiment between an adversary and a challenger. As in the soundness security experiment, the adversary may interact with a number of oracles that are provided by the challenger. The main conceptual distinction between the con dentiality and the soundness experiment is related to the adversary's access to decryption keys. The soundness experiment captures security even against adversarial *owners* of accounts. The owner of an account, for instance, must not be able to transfer more tokens than what is allowed by its current balance. The con dentiality security experiment captures security against adversaries that are not directly involved in a transfer. As long as an adversary does not have decryption keys pertaining to the source, destination, or auditor accounts, it must not learn the precise amount associated with a transfer instruction.

Therefore, in the con dentiality experiment, the challenger maintains a list of \honest" user accounts $\mathcal{T}_{\mathsf{honest}}$ that the adversary does not know the corresponding decryption keys for. After interacting with the oracles that it is provided by the challenger, the adversary outputs two transfer amounts amt_0, amt_1 as well as a source, destination, and auditor accounts from $\mathcal{T}_{\mathsf{honest}}$. The challenger generates a transfer instruction using one of these amounts and the speci ed source, destination, and auditor keys, and provides the instruction to the adversary. The adversary wins in the security experient if it correctly guesses which amount was used to generate the transfer instruction.

Definition 5.6 (Con dentiality). Let $\mathcal{CPS}$ be a con dential payment system with an assocaited public key space $\mathcal{PK}$ and public key encryption scheme $_{\mathsf{PKE}}$ = ($\mathsf{EncKeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}$) with $\mathcal{EK}$, $\mathcal{M}$, and $\mathcal{CT}$. For a security parameter λ, adversary $\mathcal{A}$, and distinguishing bit $b \in \{0, 1\}$, we de ne the con dentiality security experiment $\mathsf{EXP}_{\mathsf{confidentiality}}[\lambda, \mathcal{A}, b]$ for $\mathcal{CPS}$ as follows:

- Throughout the experiment, the challenger maintains a look-up table $\mathcal{T}_{\mathsf{honest}}$. Using $\mathcal{P}_{\mathcal{CPS}}$ that it executes internally, the challenger provides $\mathcal{A}$ access to the following set of oracles:
  - $\mathcal{O}_{\mathsf{KeyGen}}()$: On its invocation, the challenger samples $\mathsf{pk} \leftarrow_{\mathsf{R}} \mathcal{PK}$, $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{EncKeyGen}(1^{\lambda})$, and computes $\mathsf{inst}_{\mathsf{OpenAccount}} \leftarrow \mathsf{GenOpenAccount}(\mathsf{pk}, \mathsf{ek})$. It keeps record of the decryption keys $(\mathsf{pk} \mapsto \mathsf{dk})$ in $\mathcal{T}_{\mathsf{honest}}$ and submits $\mathsf{inst}_{\mathsf{OpenAccount}}$ to $\mathcal{P}_{\mathcal{CPS}}$. It relays the output to $\mathcal{A}$ and also returns $(\mathsf{pk}, \mathsf{ek})$ to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{Corrupt}}(\mathsf{pk})$: On input a public key pk, the challenger checks if an entry $(\mathsf{pk} \mapsto \mathsf{dk})$ exists in $\mathcal{T}_{\mathsf{honest}}$. If so, then it removes the entry from $\mathcal{T}_{\mathsf{honest}}$ and returns dk to $\mathcal{A}$.
  - $\mathcal{O}_{\mathsf{CloseAccount}}(\mathsf{pk})$: On input a public key pk, the challenger checks if an entry $(\mathsf{pk} \mapsto \mathsf{dk})$ exists in $\mathcal{T}_{\mathsf{honest}}$ and $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ in $\mathcal{T}_{\mathcal{CPS}}$. If so, then it computes $\mathsf{inst}_{\mathsf{CloseAccount}} \leftarrow \mathsf{GenCloseAccount}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{ct}_{\mathsf{balance}})$, and submits $\mathsf{inst}_{\mathsf{CloseAccount}}$ to $\mathcal{P}_{\mathcal{CPS}}$. It relays the output to $\mathcal{A}$ along with $\mathsf{inst}_{\mathsf{CloseAccount}}$

- $\mathcal{O}_{\mathsf{Withdraw}}(\mathsf{pk}, \mathsf{amt}_{\mathsf{withdraw}})$: On input a public key pk, the challenger checks if an entry $(\mathsf{pk} \mapsto \mathsf{dk})$ exists in $\mathcal{T}_{\mathsf{honest}}$ and $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ in $\mathcal{T}_{\mathcal{CPS}}$. If so, then it computes $\mathsf{inst}_{\mathsf{Withdraw}} \leftarrow \mathsf{GenWithdraw}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{amt}_{\mathsf{withdraw}})$ and submits $\mathsf{inst}_{\mathsf{Withdraw}}$ to $\mathcal{P}_{\mathcal{CPS}}$. It relays the output to $\mathcal{A}$ along with $\mathsf{inst}_{\mathsf{Withdraw}}$.
- $\mathcal{O}_{\mathsf{Transfer}}(\mathsf{pk}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_{\mathsf{tran}})$: On input a source public key $\mathsf{pk}_{\mathsf{source}}$, destination public key $\mathsf{pk}_{\mathsf{dest}}$, auditor encryption key $\mathsf{ek}_{\mathsf{auditor}}$, and transfer amount $\mathsf{amt}_{\mathsf{tran}}$, the challenger checks the following:
  * an entry $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{dk}_{\mathsf{source}})$ exists in $\mathcal{T}_{\mathsf{honest}}$,
  * an entry $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{ek}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}})$ exists in $\mathcal{T}_{\mathcal{CPS}}$,
  * an entry $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{ek}_{\mathsf{dest}}, \mathsf{ct}_{\mathsf{dest}})$ exists in $\mathcal{T}_{\mathcal{CPS}}$.

  If so, then it computes $\mathsf{inst}_{\mathsf{Transfer}} \leftarrow \mathsf{GenTransfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_{\mathsf{tran}})$ and submits $\mathsf{inst}_{\mathsf{Transfer}}$ to $\mathcal{P}_{\mathcal{CPS}}$. It relays the output to $\mathcal{A}$ along with $\mathsf{inst}_{\mathsf{Transfer}}$.
- $\mathcal{O}_{\mathsf{Instruction}}(\mathsf{inst})$: On input an instruction, the challenger submits inst to $\mathcal{P}_{\mathcal{CPS}}$ and relays the output to $\mathcal{A}$.
- $\mathcal{O}_{\mathsf{Read}}()$: The challenger provides $\mathcal{A}$ the entire state $\mathcal{T}_{\mathcal{CPS}}$ that is maintained by $\mathcal{P}_{\mathcal{CPS}}$.

• At one point in the experiment, the adversary $\mathcal{A}$ speci es a challenge query: a source, destination, and auditor public keys $\mathsf{pk}_{\mathsf{source}}$, $\mathsf{pk}_{\mathsf{dest}}$, $\mathsf{pk}_{\mathsf{auditor}}$, and two transfer amounts amt_0 and amt_1. The challenger veri es that the keys $\mathsf{pk}_{\mathsf{source}}$, $\mathsf{pk}_{\mathsf{dest}}$, and $\mathsf{pk}_{\mathsf{auditor}}$ pertain to honest user accounts:

  - an entry $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{dk}_{\mathsf{source}})$ exists in $\mathcal{T}_{\mathsf{honest}}$,
  - an entry $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{ek}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}})$ exists in $\mathcal{T}_{\mathcal{CPS}}$,
  - an entry $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{dk}_{\mathsf{dest}})$ exists in $\mathcal{T}_{\mathsf{honest}}$,
  - an entry $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{ek}_{\mathsf{dest}}, \mathsf{ct}_{\mathsf{dest}})$ exists in $\mathcal{T}_{\mathcal{CPS}}$,
  - an entry $(\mathsf{pk}_{\mathsf{auditor}} \mapsto \mathsf{dk}_{\mathsf{auditor}})$ exists in $\mathcal{T}_{\mathsf{honest}}$,
  - an entry $(\mathsf{pk}_{\mathsf{auditor}} \mapsto \mathsf{ek}_{\mathsf{auditor}}, \mathsf{ct}_{\mathsf{auditor}})$ exists in $\mathcal{T}_{\mathcal{CPS}}$.

  Additionally, it decrypts $\mathsf{amt}_{\mathsf{source}} \leftarrow \mathsf{Decrypt}(\mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}})$ and $\mathsf{amt}_{\mathsf{dest}} \leftarrow \mathsf{Decrypt}(\mathsf{dk}_{\mathsf{dest}}, \mathsf{ct}_{\mathsf{dest}})$ and checks that the following values are contained in $\mathcal{B}$:

  - $\mathsf{amt}_{\mathsf{source}} - \mathsf{amt}_0$,
  - $\mathsf{amt}_{\mathsf{source}} - \mathsf{amt}_1$,
  - $\mathsf{amt}_{\mathsf{dest}} + \mathsf{amt}_0$,
  - $\mathsf{amt}_{\mathsf{dest}} + \mathsf{amt}_1$.

  If these conditions are not true, then the challenger aborts the experiment and returns 0. Otherwise, it computes $\mathsf{inst}_b \leftarrow \mathsf{GenTransfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_b)$, submits inst_b to $\mathcal{P}_{\mathcal{CPS}}$, and relays the result along with inst_b.

• Throughout the rest of the experiment, challenger continues to provide $\mathcal{A}$ with the same set of oracles speci ed above with one additional global check on the adversary's deposit, withdraw, and transfer oracle queries:

– Let $\mathsf{amt}_0, \mathsf{amt}_1$ be two amounts associated with the adversary's challenge query. Let $\mathsf{amt}_{\mathsf{source}}, \mathsf{amt}_{\mathsf{dest}}$ be amounts associated with $\mathsf{pk}_{\mathsf{source}}$ and $\mathsf{pk}_{\mathsf{dest}}$ at the time the adversary outputs the challenge query.
– Let v_{source} and v_{dest} be the net sum of amounts that are deposited, withdrawn, transferred out, and transferred into accounts pertaining to $\mathsf{pk}_{\mathsf{source}}$ and $\mathsf{pk}_{\mathsf{dest}}$.
– The challenger verifies that the following values are contained in the range $\mathcal{B}$:
  * $\mathsf{amt}_{\mathsf{source}} - \mathsf{amt}_0 + v_{\mathsf{source}}$,
  * $\mathsf{amt}_{\mathsf{source}} - \mathsf{amt}_1 + v_{\mathsf{source}}$,
  * $\mathsf{amt}_{\mathsf{dest}} + \mathsf{amt}_0 + v_{\mathsf{dest}}$,
  * $\mathsf{amt}_{\mathsf{dest}} + \mathsf{amt}_1 + v_{\mathsf{dest}}$.

If the condition above does not hold at any point in the experiment since the adversary outputs its challenge query, the challenger terminates the experiment and returns 0.

- Finally, at the end of the experiment, the adversary $\mathcal{A}$ outputs a distinguishing bit b', which becomes the output of the experiment.

We say that a confidential payment system $\mathcal{CPS}$ satisfies confidentiality if for all efficient adversaries $\mathcal{A}$, we have

$$\left| \Pr\left[\mathsf{EXP}_{\mathsf{confidentiality}}[\lambda, \mathcal{A}, 0] = 1\right] - \Pr\left[\mathsf{EXP}_{\mathsf{confidentiality}}[\lambda, \mathcal{A}, 1] = 1\right] \right| = \mathsf{negl}(\lambda).$$

Discussion. One property that is not captured by the two security requirements above is instruction authorization. In a confidential payment system, users mut not be able to generate a valid $\mathsf{CloseAccount}$, $\mathsf{Withdraw}$ or $\mathsf{Transfer}$ instruction for accounts that they are not the owners of. For instance, a user must not be able to withdraw funds from another user's account. In the actual implementation of the `ZK-Token` program, the contract program processes these instructions only if it is additionally signed by the owners of relevant accounts and hence, this security property is satisfied straightforwardly. As this property is not unique to confidential payment systems, but rather a general requirement for any (non-private) payment system and smart contracts in general, we exclude it from the formal requirements to keep the definition as minimal and simple as possible.

6 Protocol Specification

In this section, we formally specify the `ZK-Token` protocol. The protocol is defined over the public key space $\mathcal{PK} = \{0,1\}^{32}$, Twisted ElGamal encryption scheme from Section 3, and balance space $\mathcal{B} = [0, 2^{64}]$. We refer to the construction overview in part 1 of the document for the main intuition behind the construction.

Construction 6.1. The `ZK-Token` protocol is defined with respect to the following:

- $\mathcal{PK} = \{0,1\}^{64}$,
- Twisted ElGamal encryption $\Pi_{\mathsf{PKE}} = (\mathsf{EncKeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ from Construction 3.5. The encryption scheme has the encryption key space $\mathcal{EK} = \mathbb{G}$, message space $\mathcal{M} = \{0,1\}^{64}$, and ciphertext space $\mathcal{CT} = \mathbb{G}^2$. As we describe in the specification of Construction 3.5, we use $\mathsf{Encrypt}(\cdot, \cdot; 0)$ to denote the deterministic version of encryption where the encryption randomness is always fixed to be $0 \in \mathbb{Z}_p$.

- Balance space $\mathcal{B} = [0, 2^{64}] \subset \mathbb{N}$.

In the construction description, we additionally rely on the following cryptographic building blocks:

- The Pedersen commitment scheme Commit of Definition 2.4.
- Non-interactive argument systems for the languages $\mathcal{L}_{G,H}^{\text{zero-balance}}$, $\mathcal{L}_{G,H}^{\text{equality}}$, and $\mathcal{L}_{G,H}^{\text{ct-validity}}$ where the group elements G and H correspond to the fixed parameter elements of the twisted ElGamal encryption scheme $_{\mathsf{PKE}}$. These arguments systems correspond to the sigma protocols defined in Sections 4.2, 4.3 and 4.4 that are compiled via the Fiat-Shamir heuristic [4]. For language $\tau \in \{\text{zero-balance}, \text{eq}, \text{ct-validity}\}$, we use $(\mathsf{Prove}_\tau, \mathsf{Verify}_\tau)$ to denote the non-interactive prover and verifier algorithms.
- Non-interactive argument system for the langauge $\mathcal{L}_{G,H,\ell,u,N}^{\text{range-agg}}$ in Theorem 4.14. The group elements G and H correspond to the fixed parameter elements of the twisted ElGamal encryption scheme $_{\mathsf{PKE}}$. We use $(\mathsf{Prove}_{\text{range-agg}}, \mathsf{Verify}_{\text{range-agg}})$ to denote the prover and verifier algorithms.

With these set of primitives, we define a confidential payment system $\mathcal{CPS}$ as follows:

- **Instruction Algorithms.** For each instruction in $\mathcal{I}_{\mathcal{CPS}}$, we define the following generation and processing algorithms:
  - OpenAccount
    - $\mathsf{GenOpenAccount}(\mathsf{pk}, \mathsf{ek}) \to \mathsf{inst}_{\mathsf{OpenAccount}}$: On input a public key pk and an encryption key ek, the algorithm defines $\mathsf{inst}_{\mathsf{OpenAccount}} = (\mathsf{pk}, \mathsf{ek})$ and returns $\mathsf{inst}_{\mathsf{OpenAccount}}$.
    - $\mathsf{ProcessOpenAccount}(\mathsf{inst}_{\mathsf{OpenAccount}}) \to 0/1$: On input an open account instruction $\mathsf{inst}_{\mathsf{OpenAccount}} = (\mathsf{pk}, \mathsf{ek})$, the instruction processor encrypts $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, 0; 0)$. Then, it adds $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct})$ in $\mathcal{T}_{\mathcal{CPS}}$ and returns 1.
  - CloseAccount
    - $\mathsf{GenCloseAccount}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{ct}_{\mathsf{balance}}) \to \mathsf{inst}_{\mathsf{CloseAccount}}$: On input a public key pk, encryption key ek, decryption key dk, and encrypted balance $\mathsf{ct}_{\mathsf{balance}}$, the algorithm generates a zero-balance proof $\pi_{\text{zero-balance}} \leftarrow \mathsf{Prove}_{\text{zero-balance}}\big((\mathsf{ek}, \mathsf{ct}_{\mathsf{balance}}), \mathsf{dk}\big)$, and returns $\mathsf{inst}_{\mathsf{CloseAccount}} = (\mathsf{pk}, \mathsf{ek}, \pi_{\text{zero-balance}})$.
    - $\mathsf{ProcessCloseAccount}(\mathsf{inst}_{\mathsf{CloseAccount}}) \to 0/1$: On input a close account instruction $\mathsf{inst}_{\mathsf{CloseAccount}} = (\mathsf{pk}, \mathsf{ek}, \pi_{\text{zero-balance}})$, the instruction processor first checks if $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ exists in $\mathcal{T}_{\mathcal{CPS}}$. If this is not the case, then it returns 0. Otherwise, it verifies the zero-balance proof by computing $\mathsf{Verify}_{\text{zero-balance}}\big((\mathsf{ek}, \mathsf{ct}_{\mathsf{balance}}), \pi_{\text{zero-balance}}\big)$. If the verification fails, it returns 0. Otherwise, it removes the entry $(\mathsf{pk} \mapsto , \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ from $\mathcal{T}_{\mathcal{CPS}}$ and returns 1.
  - Deposit
    - $\mathsf{GenDeposit}(\mathsf{pk}, \mathsf{ek}, \mathsf{amt}_{\mathsf{deposit}}) \to \mathsf{inst}_{\mathsf{Deposit}}$: On input a public key pk, encryption key ek, and deposit amount $\mathsf{amt}_{\mathsf{deposit}} \in [0, 2^{64}]$, the algorithm returns $\mathsf{inst}_{\mathsf{Deposit}} = (\mathsf{pk}, \mathsf{ek}, \mathsf{amt}_{\mathsf{deposit}})$.
    - $\mathsf{ProcessDeposit}(\mathsf{inst}_{\mathsf{Deposit}}) \to 0/1$: On input a deposit instruction $\mathsf{inst}_{\mathsf{Deposit}} = (\mathsf{pk}, \mathsf{ek}, \mathsf{amt}_{\mathsf{deposit}})$, the instruction processor first checks if an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$

exists in $\mathcal{T}_{\mathcal{CPS}}$. If this is not the case, then it returns 0. Otherwise, it computes $\mathsf{ct}_{\mathsf{deposit}} \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \mathsf{amt}_{\mathsf{deposit}}; 0)$ and replaces the entry in $\mathcal{T}_{\mathcal{CPS}}$ with $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}} + \mathsf{ct}_{\mathsf{deposit}})$.

– Withdraw

  * $\mathsf{GenWithdraw}(\mathsf{pk}, \mathsf{ek}, \mathsf{dk}, \mathsf{amt}_{\mathsf{withdraw}}, \mathsf{ct}_{\mathsf{balance}}) \to \mathsf{inst}_{\mathsf{Withdraw}}$: On input a public key pk, encryption key ek, withdraw amount $\mathsf{amt}_{\mathsf{withdraw}} \in [0, 2^{64}]$, and encrypted balance $\mathsf{ct}_{\mathsf{balance}}$, the algorithm proceeds as follows:
    1. It encrypts $\mathsf{ct}_{\mathsf{withdraw}} \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \mathsf{amt}_{\mathsf{withdraw}}; 0)$ and computes the ciphertext $\mathsf{ct}_{\mathsf{rem}} \leftarrow \mathsf{ct}_{\mathsf{balance}} - \mathsf{ct}_{\mathsf{withdraw}}$.
    2. It decrypts the account balance $\mathsf{amt}_{\mathsf{balance}} \leftarrow \mathsf{Decrypt}(\mathsf{dk}, \mathsf{ct}_{\mathsf{balance}})$ and computes the remaining balance $\mathsf{amt}_{\mathsf{rem}} \leftarrow \mathsf{amt}_{\mathsf{balance}} - \mathsf{amt}_{\mathsf{withdraw}}$.
    3. It samples a random opening $r \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ and creates a Pedersen commitment $\mathsf{comm}_{\mathsf{rem}} \leftarrow \mathsf{Commit}(\mathsf{amt}_{\mathsf{rem}}, r)$. It then generates a range proof $\pi_{\text{range-agg}} \leftarrow \mathsf{Prove}_{\text{range-agg}}\big((\mathsf{ek}, \mathsf{comm}_{\mathsf{rem}}), (\mathsf{dk}, \mathsf{amt}_{\mathsf{rem}}, r)\big)$.

    Finally, it sets $\mathsf{inst}_{\mathsf{Withdraw}} = (\mathsf{pk}, \mathsf{ek}, \mathsf{amt}_{\mathsf{withdraw}}, \mathsf{comm}_{\mathsf{rem}}, \pi_{\text{range-agg}})$ and returns $\mathsf{inst}_{\mathsf{Withdraw}}$.
  * $\mathsf{ProcessWithdraw}(\mathsf{inst}_{\mathsf{Withdraw}}) \to 0/1$: On input a withdraw $\mathsf{inst}_{\mathsf{Withdraw}} = (\mathsf{pk}, \mathsf{ek}, \mathsf{amt}_{\mathsf{withdraw}}, \mathsf{comm}_{\mathsf{rem}}, \pi_{\text{range-agg}}, \sigma)$, the instruction processor rst checks if an entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ exists in $\mathcal{T}_{\mathcal{CPS}}$. If this is not the case, then it returns 0. Otherwise, it encrypts $\mathsf{ct}_{\mathsf{withdraw}} \leftarrow \mathsf{Encrypt}(\mathsf{ek}, \mathsf{amt}_{\mathsf{withdraw}}; 0)$ and computes the ciphertext $\mathsf{ct}_{\mathsf{rem}} \leftarrow \mathsf{ct}_{\mathsf{balance}} - \mathsf{ct}_{\mathsf{withdraw}}$. It veri es the range proof $\mathsf{Verify}_{\text{range-agg}}\big((\mathsf{ek}, \mathsf{comm}_{\mathsf{rem}}), \pi_{\text{range-agg}}\big)$ and returns 0 if it fails. Otherwise, it replaces the entry $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{balance}})$ to $(\mathsf{pk} \mapsto \mathsf{ek}, \mathsf{ct}_{\mathsf{rem}})$ in $\mathcal{T}_{\mathcal{CPS}}$.

– Transfer

  * $\mathsf{GenTransfer}(\mathsf{pk}_{\mathsf{source}}, \mathsf{ek}_{\mathsf{source}}, \mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}}, \mathsf{pk}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{dest}}, \mathsf{ek}_{\mathsf{auditor}}, \mathsf{amt}_{\mathsf{tran}}) \to \mathsf{inst}_{\mathsf{Transfer}}$: On input a source public key $\mathsf{pk}_{\mathsf{source}}$, source encryption key $\mathsf{ek}_{\mathsf{source}}$, source decryption key $\mathsf{dk}_{\mathsf{source}}$, source encrypted balance $\mathsf{ct}_{\mathsf{source}}$, destination public key $\mathsf{pk}_{\mathsf{dest}}$, destination encryption key $\mathsf{ek}_{\mathsf{dest}}$, auditor encryption key $\mathsf{ek}_{\mathsf{auditor}}$, and transfer amount $\mathsf{amt}_{\mathsf{tran}} \in [0, 2^{64}]$, the algorithm proceeds as follows:
    1. It divides the transfer amount into two 32-bit numbers $\mathsf{amt}_{\mathsf{lo}}, \mathsf{amt}_{\mathsf{hi}}$ such that $\mathsf{amt}_{\mathsf{tran}} = \mathsf{amt}_{\mathsf{lo}} + 2^{32} \cdot \mathsf{amt}_{\mathsf{hi}}$.
    2. It samples random scalars $r_{\mathsf{lo}}, r_{\mathsf{hi}} \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ and creates Pedersen commitments of each amounts $\mathsf{comm}_{\mathsf{lo}} \leftarrow \mathsf{Commit}(\mathsf{amt}_{\mathsf{lo}}; r_{\mathsf{lo}})$ and $\mathsf{comm}_{\mathsf{hi}} \leftarrow \mathsf{Commit}(\mathsf{amt}_{\mathsf{hi}}; r_{\mathsf{hi}})$.
    3. It generates decryption handles for $\mathsf{comm}_{\mathsf{lo}}$ under the three encryption keys
       · $\mathsf{dh}_{\mathsf{lo,source}} \leftarrow \mathsf{GenHandle}(\mathsf{ek}_{\mathsf{source}}, r_{\mathsf{lo}})$,
       · $\mathsf{dh}_{\mathsf{lo,dest}} \leftarrow \mathsf{GenHandle}(\mathsf{ek}_{\mathsf{dest}}, r_{\mathsf{lo}})$,
       · $\mathsf{dh}_{\mathsf{lo,auditor}} \leftarrow \mathsf{GenHandle}(\mathsf{ek}_{\mathsf{auditor}}, r_{\mathsf{lo}})$.
    4. It generates decryption handles for $\mathsf{comm}_{\mathsf{hi}}$ under the three encryption keys
       · $\mathsf{dh}_{\mathsf{hi,source}} \leftarrow \mathsf{GenHandle}(\mathsf{ek}_{\mathsf{source}}, r_{\mathsf{hi}})$,
       · $\mathsf{dh}_{\mathsf{hi,dest}} \leftarrow \mathsf{GenHandle}(\mathsf{ek}_{\mathsf{dest}}, r_{\mathsf{hi}})$,
       · $\mathsf{dh}_{\mathsf{hi,auditor}} \leftarrow \mathsf{GenHandle}(\mathsf{ek}_{\mathsf{auditor}}, r_{\mathsf{hi}})$.

5. It decrypts $\mathsf{amt}_{\mathsf{balance}} \leftarrow \mathsf{Decrypt}(\mathsf{dk}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{balance}})$ and computes $\mathsf{amt}_{\mathsf{rem}} \leftarrow \mathsf{amt}_{\mathsf{balance}} - \mathsf{amt}_{\mathsf{tran}}$. Then it samples a random scalar $r_{\mathsf{rem}} \leftarrow_{\mathsf{R}} \mathbb{Z}_p$ and creates a Pedersen commitment $\mathsf{comm}_{\mathsf{rem}} \leftarrow \mathsf{Commit}(\mathsf{amt}_{\mathsf{rem}}, r_{\mathsf{rem}})$.
6. It sets $\mathsf{ct}_{\mathsf{lo,source}} = (\mathsf{comm}_{\mathsf{lo}}, \mathsf{dh}_{\mathsf{lo,source}})$, $\mathsf{ct}_{\mathsf{hi}} = (\mathsf{comm}_{\mathsf{hi}}, \mathsf{dh}_{\mathsf{hi,source}})$, and computes

$$\mathsf{ct}_{\mathsf{rem}} = \mathsf{ct}_{\mathsf{balance}} - (\mathsf{ct}_{\mathsf{lo,source}} + 2^{32} \cdot \mathsf{ct}_{\mathsf{hi,source}}).$$

Then, it creates an equality proof $\pi_{\mathsf{eq}} \leftarrow \mathsf{Prove}_{\mathsf{eq}}\big((\mathsf{ct}_{\mathsf{rem}}, \mathsf{comm}_{\mathsf{rem}}), (\mathsf{dk}_{\mathsf{source}}, r_{\mathsf{rem}})\big)$.
7. It creates a range proof $\pi_{\mathsf{range\text{-}agg}} \leftarrow \mathsf{Prove}_{\mathsf{range\text{-}agg}}\big(\{\mathsf{comm}_{\mathsf{rem}}, \mathsf{comm}_{\mathsf{lo}}, \mathsf{comm}_{\mathsf{hi}}\}, \{r_{\mathsf{rem}}, r_{\mathsf{lo}}, r_{\mathsf{hi}}\}\big)$.
8. It sets $\mathsf{ct}_{\mathsf{lo}} = (\mathsf{comm}_{\mathsf{lo}}, \mathsf{dh}_{\mathsf{lo,dest}}, \mathsf{dh}_{\mathsf{lo,auditor}})$, $\mathsf{ct}_{\mathsf{hi}} = (\mathsf{comm}_{\mathsf{hi}}, \mathsf{dh}_{\mathsf{hi,dest}}, \mathsf{dh}_{\mathsf{hi,auditor}})$, and generates a ciphertext validity proof $\pi_{\mathsf{ct\text{-}validity}} \leftarrow \mathsf{Prove}_{\mathsf{ct\text{-}validity}}\big((\mathsf{ct}_{\mathsf{lo}}, \mathsf{ct}_{\mathsf{hi}}), (r_{\mathsf{lo}}, \mathsf{amt}_{\mathsf{lo}}, r_{\mathsf{hi}}, \mathsf{amt}_{\mathsf{hi}})\big)$.

The algorithm returns $\mathsf{inst}_{\mathsf{Transfer}} = (\{\mathsf{comm}_{\mathsf{lo}}, \mathsf{comm}_{\mathsf{hi}}\}, \{\mathsf{dh}_{\mathsf{lo,source}}, \mathsf{dh}_{\mathsf{lo,dest}}, \mathsf{dh}_{\mathsf{lo,auditor}}\}, \{\mathsf{dh}_{\mathsf{hi,source}}, \mathsf{dh}_{\mathsf{hi,dest}}, \mathsf{dh}_{\mathsf{hi,auditor}}\}, \mathsf{comm}_{\mathsf{rem}}, \pi_{\mathsf{eq}}, \pi_{\mathsf{range\text{-}agg}}, \pi_{\mathsf{ct\text{-}validity}})$.

∗ $\mathsf{ProcessTransfer}(\mathsf{inst}_{\mathsf{Transfer}}) \rightarrow 0/1$: On input $\mathsf{inst}_{\mathsf{Transfer}} = (\{\mathsf{comm}_{\mathsf{lo}}, \mathsf{comm}_{\mathsf{hi}}\}, \{\mathsf{dh}_{\mathsf{lo,source}}, \mathsf{dh}_{\mathsf{lo,dest}}, \mathsf{dh}_{\mathsf{lo,auditor}}\}, \{\mathsf{dh}_{\mathsf{hi,source}}, \mathsf{dh}_{\mathsf{hi,dest}}, \mathsf{dh}_{\mathsf{hi,auditor}}\}, \mathsf{comm}_{\mathsf{rem}}, \pi_{\mathsf{eq}}, \pi_{\mathsf{range\text{-}agg}}, \pi_{\mathsf{ct\text{-}validity}}, \sigma)$, the instruction processor rst checks if entries $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{ek}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}})$ and $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{ek}_{\mathsf{dest}}, \mathsf{ct}_{\mathsf{dest}})$ exist in $\mathcal{T}_{\mathcal{CPS}}$. If this is not the case, then it returns 0. Otherwise, it veri es the following:

1. It sets $\mathsf{ct}_{\mathsf{lo,source}} = (\mathsf{comm}_{\mathsf{lo}}, \mathsf{dh}_{\mathsf{lo,source}})$, $\mathsf{ct}_{\mathsf{hi,source}} = (\mathsf{comm}_{\mathsf{hi}}, \mathsf{dh}_{\mathsf{hi,source}})$, and computes $\mathsf{ct}_{\mathsf{rem}} = \mathsf{ct}_{\mathsf{source}} - (\mathsf{ct}_{\mathsf{lo,source}} + 2^{32} \cdot \mathsf{ct}_{\mathsf{hi,source}})$. Then, it veri es $\mathsf{Verify}_{\mathsf{eq}}\big((\mathsf{ct}_{\mathsf{rem}}, \mathsf{comm}_{\mathsf{rem}}), \pi_{\mathsf{eq}}\big)$.
2. It veri es range proof $\mathsf{Verify}_{\mathsf{range\text{-}agg}}\big(\{\mathsf{comm}_{\mathsf{rem}}, \mathsf{comm}_{\mathsf{lo}}, \mathsf{comm}_{\mathsf{hi}}\}, \pi_{\mathsf{range\text{-}agg}}\big)$.
3. It sets $\mathsf{ct}_{\mathsf{lo}} = (\mathsf{comm}_{\mathsf{lo}}, \mathsf{dh}_{\mathsf{lo,dest}}, \mathsf{dh}_{\mathsf{lo,auditor}})$, $\mathsf{ct}_{\mathsf{hi}} = (\mathsf{comm}_{\mathsf{hi}}, \mathsf{dh}_{\mathsf{hi,dest}}, \mathsf{dh}_{\mathsf{hi,auditor}})$, and veri es $\mathsf{Verify}_{\mathsf{ct\text{-}validity}}\big((\mathsf{ct}_{\mathsf{lo}}, \mathsf{ct}_{\mathsf{hi}}), \pi_{\mathsf{ct\text{-}validity}}\big)$.

If any of these conditions do not verify, then the processor returns 0. Otherwise, it replaces the entries $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{ek}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{source}})$ and $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{ek}_{\mathsf{dest}}, \mathsf{ct}_{\mathsf{dest}})$ in $\mathcal{T}_{\mathcal{CPS}}$ with $(\mathsf{pk}_{\mathsf{source}} \mapsto \mathsf{ek}_{\mathsf{source}}, \mathsf{ct}_{\mathsf{rem}})$ and $(\mathsf{pk}_{\mathsf{dest}} \mapsto \mathsf{ek}_{\mathsf{dest}}, \mathsf{ct}_{\mathsf{dest}} + (\mathsf{ct}_{\mathsf{lo}} + 2^{32} \cdot \mathsf{ct}_{\mathsf{hi}}))$.

- **Client Algorithms**
  - $\mathsf{SignKeyGen}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{sk})$: On input the security parameter λ, the key generation algorithm computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{S.KeyGen}(1^\lambda)$ and returns $(\mathsf{pk}, \mathsf{sk})$.
  - $\mathsf{EncKeyGen}(1^\lambda) \rightarrow (\mathsf{ek}, \mathsf{dk})$: On input the security parameter λ, the key generation algorithm computes $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{E.KeyGen}(1^\lambda)$ and returns $(\mathsf{ek}, \mathsf{dk})$.
  - $\mathsf{DecryptTransfer}(\mathsf{dk}, \mathsf{inst}_{\mathsf{Transfer}}) \rightarrow \mathsf{amt}_{\mathsf{tran}}$: On input a decryption key dk and a transfer instruction $\mathsf{inst}_{\mathsf{Transfer}} = (\{\mathsf{comm}_{\mathsf{lo}}, \mathsf{comm}_{\mathsf{hi}}\}, \{\mathsf{dh}_{\mathsf{lo,source}}, \mathsf{dh}_{\mathsf{lo,dest}}, \mathsf{dh}_{\mathsf{lo,auditor}}\}, \{\mathsf{dh}_{\mathsf{hi,source}}, \mathsf{dh}_{\mathsf{hi,dest}}, \mathsf{dh}_{\mathsf{hi,auditor}}\}, \mathsf{comm}_{\mathsf{rem}}, \pi_{\mathsf{eq}}, \pi_{\mathsf{range\text{-}agg}}, \pi_{\mathsf{ct\text{-}validity}})$, the decryption algorithm decrypts $\mathsf{amt}_{\mathsf{lo}} \leftarrow \mathsf{Decrypt}\big(\mathsf{dk}, (\mathsf{comm}_{\mathsf{lo}}, \mathsf{dh}_{\mathsf{lo}})\big)$, $\mathsf{amt}_{\mathsf{hi}} \leftarrow \mathsf{Decrypt}\big(\mathsf{dk}, (\mathsf{comm}_{\mathsf{hi}}, \mathsf{dh}_{\mathsf{hi}})\big)$, and returns $\mathsf{amt} = \mathsf{amt}_{\mathsf{lo}} + 2^{32} \cdot \mathsf{amt}_{\mathsf{hi}}$.
  - $\mathsf{DecryptBalance}(\mathsf{dk}, \mathsf{data}_{\mathsf{balance}}) \rightarrow \mathsf{amt}_{\mathsf{balance}}$: On input a decryption key dk and an account data $\mathsf{data}_{\mathsf{balance}} = (\mathsf{ek}, \mathsf{ct})$, the decrytion algorithm returns the output of $\mathsf{Decrypt}(\mathsf{dk}, \mathsf{ct})$.

References

[1] Boneh, D., Drijvers, M., and Neven, G. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security* (2018), Springer, pp. 435{464.

[2] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. Bulletproofs: Short proofs for con dential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 315{334.

[3] Chen, Y., Ma, X., Tang, C., and Au, M. H. Pgc: Decentralized con dential payment system with auditability. In *European Symposium on Research in Computer Security* (2020), Springer, pp. 591{610.

[4] Fiat, A., and Shamir, A. How to prove yourself: Practical solutions to identi cation and signature problems. In *Conference on the theory and application of cryptographic techniques* (1986), Springer, pp. 186{194.

[5] Lindell, Y. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology 16*, 3 (2003).

A Proofs in Section 4

In this section, we provide the missing proofs from Section 4.

A.1 Zero-Balance Argument

A.1.1 Proof of Theorem 4.5

To prove completeness, let us x any valid instance and witness for $\mathcal{L}_{G,H}^{\text{zero-balance}}$: $P, C, D \in \mathbb{G}$ and $s \in \mathbb{Z}_p$ such that $s \cdot P = H$ and $s \cdot D = C$. It su ces to show that after an honest execution of the protocol by the prover, the veri er always returns 1 at the end of the protocol. Let y and c be any elements in $\mathbb{Z}_p$ and let $Y_P = y \cdot P$, $Y_D = y \cdot D$, and $z = c \cdot s + y$ in an execution of the protocol. Then we have

$$z \cdot P = (c \cdot s + y)P = c \cdot (s \cdot P) + y \cdot P = c \cdot H + Y_P,$$

$$z \cdot D = (c \cdot s + y)D = c \cdot (s \cdot D) + y \cdot D = c \cdot C + Y_C.$$

As both of the algebraic relations that the veri er checks at the end of the protocol hold, the proof is always accepted. Completeness follows.

A.1.2 Proof of Theorem 4.6

To prove soundness, we construct an emulator $\mathcal{E}$ that has oracle access to any malicious prover $\mathcal{P}^*$ and extracts a valid witness by rewinding $\mathcal{P}^*$ and simulating two execution of the zero-balance protocol with an honest veri er $\mathcal{V}$. By the work of Lindell [?], this su ces to prove witness-extended emulation soundness.

Let (P, C, D) be an instance of the language $\mathcal{L}_{G,H}^{\text{zero-balance}}$. We construct an emulator $\mathcal{E}$ that uses $\mathcal{P}^*$ to extract a valid witness as follows:

- The emulator $\mathcal{E}$ rst executes $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$ to produce a transcript $\mathsf{tr} = (Y_P, Y_D, c, z)$.
- Then, it rewinds the protocol to the point where the veri er $\mathcal{V}$ samples a random $c \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. It programs $\mathcal{V}$ with fresh randomness such that $\mathcal{V}$ generates a new $c' \leftarrow \mathbb{Z}_p$ independently of the previous execution of the protocol.
- The emulator completes the second execution of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$, producing a new transcript $\mathsf{tr} = (Y_P, Y_D, c', z')$.
- If $c - c' = 0$, then the emulator aborts and returns $\perp$. Otherwise, it computes $s \leftarrow (z - z')/(c - c')$ and returns s as the extracted witness.

To complete the proof, we rst bound the probability that $\mathcal{E}$ does not abort at the end of the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$. Then, we show that if $\mathcal{E}$ does not abort, then the extracted witness $s = (z - z')/(c - c')$ is a valid witness.

Abort probability. The emulator $\mathcal{E}$ aborts only when $c = c'$, which is dependent on the probability that $\mathcal{P}^*$ successfully convinces $\mathcal{V}$ at the end of the protocol. Let ε_{P^*} be the probability that $\mathcal{P}^*$ successfully convinces $\mathcal{V}$ in $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$. We bound the probability that $c = c'$ with ε_{P^*} using the rewinding lemma 2.3. Speci cally, let us de ne the following random variables:

- Let X be the elements (Y_P, Y_D) in the transcript of an execution of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.

- Let Y and Y' be the values c and c' respectively in the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let Z and Z' be the values z and z' respectively in the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let $f(\mathsf{tr}) \to \{0, 1\}$ be the protocol veri cation function that returns 1 if tr is an accepting transcript and 0 otherwise.

Then, the rewinding lemma states that

$$\Pr\left[f(X, Y, Z) = 1 \wedge f(X, Y', Z') = 1 \wedge Y \neq Y'\right] \geq \varepsilon_{P^*}^2 - \varepsilon_{P^*}/p.$$

By assumption, we have $1/p = \mathsf{negl}(\lambda)$. Therefore, if ε_{P^*} is non-negligible, then the probability that $\mathcal{E}$ aborts at the end of the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$ is non-negligible.

Witness validity. Now assume that the two executions of $\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u) \rangle$ returns two accepting transcripts $\mathsf{tr} = (Y_P, Y_D, c, z)$, $\mathsf{tr}' = (Y_P, Y_D, c', z')$, and that $\mathcal{E}$ does not abort and returns $s = (z - z')/(c - c')$. Since tr and tr' are accepting transcripts, we have

$$z \cdot P = c \cdot H + Y_P,$$
$$z' \cdot P = c' \cdot H + Y_P.$$

This means that $(z - z') \cdot P = (c - c') = H$ and hence, $s \cdot P = H$. Similarly, we have

$$z \cdot D = c \cdot C + Y_D,$$
$$z' \cdot D = c' \cdot C + Y_D.$$

Therefore, the relation $(z - z') \cdot D = (c - c') \cdot C$ holds, which means that $s \cdot D = C$.

We have shown that if $\mathcal{P}^*$ successfully convinces the veri er $\mathcal{V}$ for an instance $x = (P, C, D)$ with non-negligible probability, then the emulator $\mathcal{E}$ successfully extracts a valid witness s. This completes the proof of soundness.

A.1.3 Proof of Theorem 4.7

Fix any elements $P, C, D \in \mathbb{G}$ and $s \in \mathbb{Z}_p$ such that $s \cdot P = H$ and $s \cdot D = C$. Let $\mathsf{tr}^* = (Y_P^*, Y_D^*, c^*, z^*)$ be any accepting transcript. By the speci cation of the protocol, the probability that an honest execution of the protocol by the prover and the veri er results in the transcript tr^* is as follows:

$$\Pr\left[\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u) \rangle \to \mathsf{tr} \wedge \mathsf{tr} = \mathsf{tr}^*\right] = 1/p^2.$$

To prove zero-knowledge, we de ne a simulator $\mathcal{S}$ that produces such distribution without knowledge of a valid witness s.

$\mathcal{S}(P, C, D)$:

1. Sample $c, z \leftarrow_{\mathsf{R}} \mathbb{Z}_p$.
2. Set $Y_P = z \cdot P - c \cdot D$.
3. Set $Y_D = z \cdot D - c \cdot C$.
4. Return $\mathsf{tr} = (Y_P, Y_D, c, z)$.

The simulator $\mathcal{S}$ returns a transcript that is uniformly random under the condition that $z \cdot P = Y_P + c \cdot D$ and $z \cdot D = Y_D + c \cdot C$. As the variables Y_P, Y_D are completely determined by c, z, we have

$$\Pr[\mathcal{S}(P, C, D) \to \mathsf{tr} \wedge \mathsf{tr} = \mathsf{tr}^*] = 1/p^2,$$

for any xed transcript tr^*. Zero-knowledge follows.

A.2 Equality Argument

A.2.1 Proof of Theorem 4.8

to prove completeness, let us x any valid instance and witness for $\mathcal{L}_{G,H}^{\mathsf{equality}}$: $P_{\mathsf{EG}}, C_{\mathsf{EG}}, D_{\mathsf{EG}}, C_{\mathsf{Ped}} \in \mathbb{G}$ and $s, x, r \in \mathbb{Z}_p$ such that

- $s \cdot P_{\mathsf{EG}} = H$
- $C_{\mathsf{EG}} - s \cdot D_{\mathsf{EG}} = x \cdot G$
- $C_{\mathsf{Ped}} = x \cdot G + r \cdot H$

Let y_s, y_x, y_r and c be any elements in $\mathbb{Z}_p$, and let

- $Y_0 = y_s \cdot P$, $Y_1 = y_x \cdot G + y_s \cdot D_{\mathsf{EG}}$, $Y_2 = y_x \cdot G + y_r \cdot H$
- $z_s = c \cdot s + y_s$, $z_x = c \cdot x + y_x$, $z_r = c \cdot r + y_r$

in an execution of the protocol. Then we have

$$\begin{aligned} z_s \cdot P &= (c \cdot s + y_s) \cdot P \\ &= c \cdot (s \cdot P) + y_s \cdot P \\ &= c \cdot H + Y_0 \end{aligned}$$

$$\begin{aligned} z_x \cdot G + z_s \cdot D_{\mathsf{EG}} &= (c \cdot x + y_x) \cdot G + (c \cdot s + y_s) \cdot D_{\mathsf{EG}} \\ &= c \cdot (x \cdot G + s \cdot D_{\mathsf{EG}}) + (y_x \cdot G + y_s \cdot D_{\mathsf{EG}}) \\ &= c \cdot C_{\mathsf{EG}} + Y_1 \end{aligned}$$

$$\begin{aligned} z_x \cdot G + z_r \cdot D_{\mathsf{EG}} &= (c \cdot x + y_x) \cdot G + (c \cdot r + y_r) \cdot H \\ &= c \cdot (x \cdot G + r \cdot H) + (y_x \cdot G + y_r \cdot H) \\ &= c \cdot C_{\mathsf{Ped}} + Y_2 \end{aligned}$$

As all the algebraic relations that the veri er checks hold, the proof is always accepted. Completeness follows.

A.2.2 Proof of Theorem 4.9

To prove soundness, we construct an emulator $\mathcal{E}$ that has oracle access to any malicious prover $\mathcal{P}^*$ and extracts a valid witness by rewinding $\mathcal{P}^*$ and simulating two execution of the zero-balance protocol with an honest veri er $\mathcal{V}$.

Let $(P_{\mathsf{EG}}, C_{\mathsf{EG}}, D_{\mathsf{EG}}, C_{\mathsf{Ped}})$ be an instance of the language $\mathcal{L}_{G,H}^{\mathsf{equality}}$. We construct an emulator $\mathcal{E}$ that uses $\mathcal{P}^*$ to extract a valid witness as follows:

- The emulator $\mathcal{E}$ rst executes $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$ to produce a transcript $\mathsf{tr} = (Y_0, Y_1, Y_2, c, z_s, z_x, z_r)$.
- Then, it rewinds the protocol to the point where the veri er $\mathcal{V}$ samples a random $c \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. It programs $\mathcal{V}$ with fresh randomness such that $\mathcal{V}$ generates a new $c' \leftarrow \mathbb{Z}_p$ independently of the previous execution of the protocol.

- The emulator completes the second execution of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$, producing a new transcript $\mathsf{tr} = (Y_0, Y_1, Y_2, c, z_s', z_x', z_r')$.
- If $c - c' = 0$, then the emulator aborts and returns $\perp$. Otherwise, it computes
  - $s \leftarrow (z_s - z_s')/(c - c')$
  - $x \leftarrow (z_x - z_x')/(c - c')$
  - $r \leftarrow (z_r - z_r')/(c - c')$

  and returns (s, x, r) as the witness.

To complete the proof, we first bound the probability that $\mathcal{E}$ does not abort at the end of the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$. Then, we show that if $\mathcal{E}$ does not abort, then the extracted witness (s, x, r) is valid.

Abort probability. The emulator $\mathcal{E}$ aborts only when $c = c'$, which is dependent on the probability that $\mathcal{P}^*$ successfully convinces $\mathcal{V}$ at the end of the protocol. Let $\varepsilon_{\mathcal{P}^*}$ be the probability that $\mathcal{P}^*$ successfully convinces $\mathcal{V}$ in $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$. We bound the probability that $c = c'$ with $\varepsilon_{\mathcal{P}^*}$ using the rewinding lemma 2.3. Specifically, let us define the following random variables:

- Let X be the elements (Y_0, Y_1, Y_2) in the transcript of an execution of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let Y and Y' be the values c and c' respectively in the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let Z and Z' be the values (z_s, z_x, z_r) and (z_s', z_x', z_r') respectively in the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let $f(\mathsf{tr}) \to \{0, 1\}$ be the protocol verification function that returns 1 if tr is an accepting transcript and 0 otherwise.

Then, the rewiding lemma states that

$$\Pr\left[f(X, Y, Z) = 1 \wedge f(X, Y', Z') = 1 \wedge Y \neq Y' \right] \geq \varepsilon_{\mathcal{P}^*}^2 - \varepsilon_{\mathcal{P}^*}/p.$$

By assumption, we have $1/p = \mathsf{negl}(\lambda)$. Therefore, if $\varepsilon_{\mathcal{P}^*}$ is non-negligible, then the probability that $\mathcal{E}$ aborts at the end of the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$ is non-negligible.

Witness validity. Now assume that the two executions of $\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u) \rangle$ returns two accepting transcripts $\mathsf{tr} = (Y_0, Y_1, Y_2, c, z_s, z_x, z_r)$, $\mathsf{tr}' = (Y_0, Y_1, Y_2, c', z_s', z_x', z_r')$, and that $\mathcal{E}$ does not abort and returns

- $s \leftarrow (z_s - z_s')/(c - c')$
- $x \leftarrow (z_x - z_x')/(c - c')$
- $r \leftarrow (z_r - z_r')/(c - c')$

Since tr and tr' are accepting transcripts, we have

$$z_x \cdot P_{\mathsf{EG}} = c \cdot H + Y_0,$$

$$z_x' \cdot P_{\mathsf{EG}} = c' \cdot H + Y_0,$$

This means that $(z_x - z'_x) \cdot P_{\mathsf{EG}} = (c - c') \cdot H$ and hence, $s \cdot P_{\mathsf{EG}} = H$. Similarly, we have

$$z_x \cdot G + z_s \cdot D_{\mathsf{EG}} = c \cdot C_{\mathsf{EG}} + Y_1,$$

$$z'_x \cdot G + z'_s \cdot D_{\mathsf{EG}} = c' \cdot C_{\mathsf{EG}} + Y_1,$$

This means that $(z_x - z'_x) \cdot G + (z_s - z'_s) \cdot D_{\mathsf{EG}} = (c - c') \cdot C_{\mathsf{EG}}$ and hence, $x \cdot G + s \cdot D_{\mathsf{EG}} = C_{\mathsf{EG}}$. Finally, we have

$$z_x \cdot G + z_r \cdot H = c \cdot C_{\mathsf{Ped}} + Y_2,$$

$$z'_x \cdot G + z'_r \cdot H = c' \cdot C_{\mathsf{Ped}} + Y_2,$$

which means that $(z_x - z'_x) \cdot G + (z_r - z'_r) \cdot H = (c - c') \cdot C_{\mathsf{Ped}}$ and hence, $x \cdot G + r \cdot H = C_{\mathsf{Ped}}$.

We have shown that if $\mathcal{P}^*$ successfully convinces the veri er $\mathcal{V}$ for an instance $x = (P_{\mathsf{EG}}, C_{\mathsf{EG}}, D_{\mathsf{EG}}, C_{\mathsf{Ped}})$ with non-negligible probability, then the emulator $\mathcal{E}$ successfully extracts a valid witness (s, x, r). This completes the proof of soundness.

A.2.3 Proof of Theorem 4.10

Fix any elements $P_{\mathsf{EG}}, C_{\mathsf{EG}}, D_{\mathsf{EG}}, C_{\mathsf{Ped}} \in \mathbb{G}$ and $s, x, r \in \mathbb{Z}_p$ such that $C_{\mathsf{EG}} - s \cdot D_{\mathsf{EG}} = x \cdot G$ and $C_{\mathsf{Ped}} = x \cdot G + r \cdot H$. Let $\mathsf{tr}^* = (Y_0^*, Y_1^*, Y_2^*, c^*, z_s^*, z_x^*, z_r^*)$ be any accepting transcript. By the speci cation of the protocol, the probability that an honest execution of the protocol by the prover and the veri er results in the transcript tr^* is as follows:

$$\Pr\left[\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u) \rangle \to \mathsf{tr} \wedge \mathsf{tr} = \mathsf{tr}^*\right] = 1/p^4.$$

To prove zero-knowledge, we de ne a simulator $\mathcal{S}$ that produces such distribution without knowledge of a valid witness s, x, and r.

$\mathcal{S}(P_{\mathsf{EG}}, C_{\mathsf{EG}}, D_{\mathsf{EG}}, C_{\mathsf{Ped}})$:

1. Sample $c, z_s, z_x, z_r \leftarrow_{\mathsf{R}} \mathbb{Z}_p$
2. Set $Y_0 = z_x \cdot P_{\mathsf{EG}} - c \cdot H$
3. Set $Y_1 = z_x \cdot G + z_s \cdot D_{\mathsf{EG}} - c \cdot C_{\mathsf{EG}}$
4. Set $Y_2 = z_x \cdot G + z_r \cdot H - c \cdot C_{\mathsf{Ped}}$
5. Return $\mathsf{tr} = (Y_0, Y_1, Y_2, c, z_s, z_x, z_r)$

The simulator $\mathcal{S}$ returns a transcript that is uniformly random given that

- $z_x \cdot P_{\mathsf{EG}} = c \cdot H + Y_0$,
- $z_x \cdot G + z_s \cdot D_{\mathsf{EG}} = c \cdot C_{\mathsf{EG}} + Y_1$,
- $z_x \cdot G + z_r \cdot H = c \cdot C_{\mathsf{Ped}} + Y_2$.

As the variables Y_0, Y_1 and Y_2 are completely determined by c, z_s, z_x, z_r, we have

$$\Pr\left[\mathcal{S}(C_{\mathsf{EG}}, D_{\mathsf{EG}}, C_{\mathsf{Ped}}) \to \mathsf{tr} \wedge \mathsf{tr} = \mathsf{tr}^*\right] = 1/p^4,$$

for any xed transcript tr^*. Zero-knowledge follows.

A.3 Ciphertext Validity Argument

A.3.1 Proof of Theorem 4.11

To prove completeness, let us x any valid instance and witness for $\mathcal{L}_{G,H}^{\mathsf{ct-validity}}$: $P_1, P_2, C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2}, C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2} \in \mathbb{G}$ and $r_{\mathsf{lo}}, x_{\mathsf{lo}}, r_{\mathsf{hi}}, x_{\mathsf{hi}} \in \mathbb{Z}_p$ such that

- $C_{\mathsf{lo}} = r_{\mathsf{lo}} \cdot H + x_{\mathsf{lo}} \cdot G$,
- $C_{\mathsf{hi}} = r_{\mathsf{hi}} \cdot H + x_{\mathsf{hi}} \cdot G$,
- $D_{\mathsf{lo},1} = r_{\mathsf{lo}} \cdot P_1$,
- $D_{\mathsf{lo},2} = r_{\mathsf{lo}} \cdot P_2$,
- $D_{\mathsf{hi},1} = r_{\mathsf{hi}} \cdot P_1$.
- $D_{\mathsf{hi},2} = r_{\mathsf{hi}} \cdot P_2$.

Let t, y_r, y_x, z_r, z_x be any elements in $\mathbb{Z}_p$ and let

- $Y_0 = y_r \cdot H + y_x \cdot G$,
- $Y_1 = y_r \cdot P_1$,
- $Y_2 = y_r \cdot P_2$,
- $z_r = c \cdot r + y_r$,
- $z_x = c \cdot x + y_x$,

in an execution of the protocol. Then we have

$$\begin{aligned}
z_r \cdot H + z_x \cdot G &= (c \cdot r + y_r) \cdot H + (c \cdot x + y_x) \cdot G \\
&= c \cdot (r \cdot H + x \cdot G) + (y_r \cdot H + y_x \cdot G) \\
&= c \cdot \big((r_{\mathsf{lo}} + t \cdot r_{\mathsf{hi}}) \cdot H + (x_{\mathsf{lo}} + t \cdot x_{\mathsf{hi}}) \cdot G\big) + Y_0 \\
&= c \cdot \big(C_{\mathsf{lo}} + t \cdot C_{\mathsf{hi}}\big) \\
&= c \cdot C + Y_0
\end{aligned}$$

$$\begin{aligned}
z_r \cdot P_1 &= (c \cdot r + y_r) \cdot P_1 \\
&= c \cdot (r \cdot P_1) + y_r \cdot P_1 \\
&= c \cdot \big((r_{\mathsf{lo}} + t \cdot r_{\mathsf{hi}}) \cdot P_1\big) + y_r \cdot P_1 \\
&= c \cdot (D_{\mathsf{lo},1} + t \cdot D_{\mathsf{hi},1}) + Y_1 \\
&= c \cdot D_1 + Y_1
\end{aligned}$$

$$\begin{aligned}
z_r \cdot P_2 &= (c \cdot r + y_r) \cdot P_2 \\
&= c \cdot (r \cdot P_2) + y_r \cdot P_2 \\
&= c \cdot \big((r_{\mathsf{lo}} + t \cdot r_{\mathsf{hi}}) \cdot P_2\big) + y_r \cdot P_2 \\
&= c \cdot (D_{\mathsf{lo},2} + t \cdot D_{\mathsf{hi},2}) + Y_2 \\
&= c \cdot D_2 + Y_2
\end{aligned}$$

As all of the algebraic relations that the veri er checks at the end of the protocol hold, the proof is always accepted. Completeness follows.

A.3.2 Proof of Theorem 4.12

To prove soundness, we construct an emulator $\mathcal{E}$ that has oracle access to any malicious prover $\mathcal{P}^*$ and extracts a valid witness by rewinding $\mathcal{P}^*$ and simulating four executions of the zero-balance protocol with an honest veri er $\mathcal{V}$.

Let $(P, C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2}, C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2})$ be an instance of the language $\mathcal{L}_{G,H}^{\mathsf{ct-validity}}$. We construct an emulator $\mathcal{E}$ that uses $\mathcal{P}^*$ to extract a valid witness. The emulator $\mathcal{E}$ rewinds the protocol at di erent stages. To simplify the presentation, we de ne a sub-emulator $\mathcal{E}_{\mathsf{inner}}$ that $\mathcal{E}$ uses as a subroutine to extract a valid witness. The sub-emulator $\mathcal{E}_{\mathsf{inner}}$ works as follows:

- The emulator $\mathcal{E}_{\mathsf{inner}}$ rst executes $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$ to produce a transcript $\mathsf{tr} = (w, Y_0, Y_1, Y_2, c, z_r, z_x)$.
- Then, it rewinds the protocol to the point where the veri er $\mathcal{V}$ samples a random $c \leftarrow_{\mathsf{R}} \mathbb{Z}_p$. It programs $\mathcal{V}$ with fresh randomness such that $\mathcal{V}$ generates a new $c' \leftarrow \mathbb{Z}_p$ independently of the previous execution of the protocol.
- The emulator completes the second execution of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$, producing a new transcript $\mathsf{tr} = (t, Y_0, Y_1, c', z_r', z_x')$.
- If $c - c' = 0$, then the emulator aborts and returns $\perp$. Otherwise, it computes
  - $r \leftarrow (z_r - z_r')/(c - c')$
  - $x \leftarrow (z_x - z_x')/(c - c')$

  and returns (r, x).

We rst bound the probability that $\mathcal{E}_{\mathsf{inner}}$ does not abort at the end of the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$. Then, we show that if $\mathcal{E}_{\mathsf{inner}}$ does not abort, then its output (r, x) satis es

- $C = r \cdot H + x \cdot G$,
- $D_1 = r \cdot P_1$,
- $D_2 = r \cdot P_2$,

where $C = C_{\mathsf{lo}} + t \cdot C_{\mathsf{hi}}$, $D_1 = D_{\mathsf{lo},1} + t \cdot D_{\mathsf{hi},1}$, and $D_2 = D_{\mathsf{lo},2} + t \cdot D_{\mathsf{hi},2}$ in an execution of the protocol.

Abort probability of the sub-emulator. The emulator $\mathcal{E}_{\mathsf{inner}}$ aborts only when $c = c'$, which is dependent on the probability that $\mathcal{P}^*$ successfully convinces $\mathcal{V}$ at the end of the protocol. Let $\varepsilon_{\mathcal{P}^*}$ be the probability that $\mathcal{P}^*$ successfully convinces $\mathcal{V}$ in $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$. We bound the probability that $c = c'$ with $\varepsilon_{\mathcal{P}^*}$ using the rewinding lemma 2.3. Speci cally, let us de ne the following random variables:

- Let X be the elements (w, Y_0, Y_1, Y_2) in the transcript of an execution of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let Y and Y' be the values c and c' respectively in the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let Z and Z' be the values (z_r, z_x) and (z_r', z_x') respectively in the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let $f(\mathsf{tr}) \to \{0, 1\}$ be the protocol veri cation function that returns 1 if tr is an accepting transcript and 0 otherwise.

Then, the rewiding lemma states that

$$\Pr\left[f(X,Y,Z)=1 \wedge f(X,Y',Z')=1 \wedge Y \neq Y'\right] \geq \varepsilon^2 - \varepsilon/p.$$

By assumption, we have $1/p = \mathsf{negl}(\lambda)$. Therefore, if $\varepsilon_{\mathcal{P}^*}$ is non-negligible, then the probability that $\mathcal{E}$ aborts at the end of the two executions of $\langle \mathcal{P}^*(\rho,u,\mathsf{st}), \mathcal{V}(\rho,u)\rangle$ is non-negligible.

Output validity of sub-emulator. Now assume that the two executions of $\langle \mathcal{P}(\rho,u,w), \mathcal{V}(\rho,u)\rangle$ returns two accepting transcripts $\mathsf{tr} = (t, Y_0, Y_1, Y_2, c, z_r, z_x)$, $\mathsf{tr}' = (t, Y_0, Y_1, Y_2, c', z_r', z_x')$, and that $\mathcal{E}_{\mathsf{inner}}$ does not abort and returns

- $r \leftarrow (z_r - z_r')/(c - c')$
- $x \leftarrow (z_x - z_x')/(c - c')$

Since tr and tr' are accepting transcripts, we have

$$z_r \cdot H + z_x \cdot G = c \cdot C + Y_0,$$

$$z_r' \cdot H + z_x' \cdot G = c' \cdot C + Y_0,$$

This means that $(z_r - z_r') \cdot H + (z_x - z_x') \cdot G = (c - c') \cdot C$ and hence, $r \cdot H + x \cdot G = C$. Similarly, we have

$$z_r \cdot P_1 = c \cdot D + Y_1,$$

$$z_r' \cdot P_1 = c' \cdot D + Y_1,$$

This means that $(z_r - z_r') \cdot P_1 = (c - c') \cdot D_1$, which means that $r \cdot P_1 = D_1$. The argument can be used to show that $r \cdot P_2 = D_2$.

Main emulator. For a language instance $u = (P, C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2}, C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2})$, the main emulator $\mathcal{E}$ executes two instances of the sub-emulator $\mathcal{E}_{\mathsf{inner}}$ to obtain two outputs

- Let t be the veri er's rst message in the protocol on the rst execution of $\mathcal{E}_{\mathsf{inner}}$. The sub-emulator returns r and x such that
  - $C = r \cdot H + x \cdot G$,
  - $D_1 = r \cdot P_1$,
  - $D_2 = r \cdot P_2$,

  where $C = C_{\mathsf{lo}} + t \cdot C_{\mathsf{hi}}$, $D_1 = D_{\mathsf{lo},1} + t \cdot D_{\mathsf{hi},1}$, and $D_2 = D_{\mathsf{lo},2} + t \cdot D_{\mathsf{hi},2}$.

- Let t' be the veri er's rst message in the protocol on the rst execution of $\mathcal{E}_{\mathsf{inner}}$. The sub-emulator returns r' and x' such that
  - $C = r' \cdot H + x' \cdot G$,
  - $D_1 = r' \cdot P_1$,
  - $D_2 = r' \cdot P_2$,

  where $C' = C_{\mathsf{lo}} + t' \cdot C_{\mathsf{hi}}$, $D_1 = D_{\mathsf{lo},1} + t' \cdot D_{\mathsf{hi},1}$, and $D_2 = D_{\mathsf{lo},2} + t' \cdot D_{\mathsf{hi},2}$.

If $t = t'$ in the two executions, $\mathcal{E}$ aborts and returns $\perp$. Otherwise, the emulator returns the following:

- $r_{\mathsf{lo}} = (rt' - r't)/(t' - t)$ and $x_{\mathsf{lo}} = (xt' - x't)/(t' - t)$,
- $r_{\mathsf{hi}} = (r - r')/(t - t')$ and $x_{\mathsf{hi}} = (x - x')/(t - t')$.

To nish the proof, we bound the probability that $\mathcal{E}$ does not abort at the end of the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$. Then, we show that if $\mathcal{E}$ does not abort, then its output (r, x) is a valid witness.

Abort probability of the main emulator. The emulator $\mathcal{E}$ aborts only when $t = t'$, which is dependent on the probability that $\mathcal{E}_{\mathsf{inner}}$ successfully returns an output (r, x). Let $\varepsilon_{\mathcal{E}_{\mathsf{inner}}}$ be the probability that $\mathcal{E}_{\mathsf{inner}}$ successfully returns an output (r, x). We bound the probability that $t = t'$ with $\varepsilon_{\mathcal{E}_{\mathsf{inner}}}$ using the rewinding lemma. Speci cally, let us de ne the following random variables:

- The variable $X = \varepsilon$ is an empty variable.
- Let Y and Y' be the values t and t' respectively in the two executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let Z and Z' be the values in the two pairs of transcripts $\mathsf{tr} = (\mathsf{tr}_0, \mathsf{tr}_1)$ and $\mathsf{tr}' = (\mathsf{tr}_0', \mathsf{tr}_1')$ during $\mathcal{E}_{\mathsf{inner}}$'s executions of $\langle \mathcal{P}^*(\rho, u, \mathsf{st}), \mathcal{V}(\rho, u) \rangle$.
- Let $f(\mathsf{tr}) \to \{0, 1\}$ be the function that output 1 if $\mathcal{E}_{\mathsf{inner}}$ can successfully extract (r, x) from tr and 0 otherwise.

Then, the rewinding lemma states that

$$\Pr\left[f(X, Y, Z) = 1 \wedge f(X, Y', Z') = 1 \wedge Y \neq Y'\right] \geq \varepsilon^2 - \varepsilon/p.$$

By assumption, we have $1/p = \mathsf{negl}(\lambda)$. Therefore, if $\varepsilon_{\mathcal{E}_{\mathsf{inner}}}$ is non-negligible, then the probability that $\mathcal{E}$ aborts at the end of the two executions of $\mathcal{E}_{\mathsf{inner}}$ is non-negligible.

Witness validity. Now assume that $\mathcal{E}$ does not abort after two executions of the protocol. Then it returns we have $t \neq t'$ and $\mathcal{E}$ returns

- $r_{\mathsf{lo}} = (rt' - r't)/(t' - t)$ and $x_{\mathsf{lo}} = (xt' - x't)/(t' - t)$,
- $r_{\mathsf{hi}} = (r - r')/(t - t')$ and $x_{\mathsf{hi}} = (x - x')/(t - t')$.

We show that $r_{\mathsf{lo}}, x_{\mathsf{lo}}, r_{\mathsf{hi}}, x_{\mathsf{hi}}$ make a valid witness for the ciphertext validity relation. By assumption on $\mathcal{E}_{\mathsf{inner}}$, the values r, x, r', x' satisfy the following relations:

$$r \cdot H + x \cdot G = C = C_{\mathsf{lo}} + t \cdot C_{\mathsf{hi}},$$

$$r' \cdot H + x' \cdot G = C = C_{\mathsf{lo}} + t' \cdot C_{\mathsf{hi}}.$$

Subtracting the two relations above, we have

$$(r - r') \cdot H + (x - x') \cdot G = (t - t') \cdot C_{\mathsf{hi}},$$

and hence, we have $(r - r')/(t - t') \cdot H + (x - x')/(t - t') \cdot G = C_{\mathsf{hi}}$.

Likewise, by assumption on $r_{\mathsf{lo}}, x_{\mathsf{lo}}, r_{\mathsf{hi}}, x_{\mathsf{hi}}$, we have

$$r \cdot P_1 = D_1 = D_{\mathsf{lo},1} + t \cdot D_{\mathsf{hi},1},$$

$$r' \cdot P_1 = D_1 = D_{\mathsf{lo},1} + t' \cdot D_{\mathsf{hi},1}.$$

Subtracting the two relations, we have

$$(r - r') \cdot P_1 = (t - t') \cdot D_{\mathsf{hi},1},$$

and hence, we have $(r - r')/(t - t') \cdot P_1 = D_{\mathsf{hi},1}$. Similar arguments shows that $r_{\mathsf{lo}} \cdot H + x_{\mathsf{lo}} \cdot G = C_{\mathsf{lo}}$, $r_{\mathsf{lo}} \cdot P_1 = D_{\mathsf{lo},1}$, $r_{\mathsf{lo}} \cdot P_2 = D_{\mathsf{lo},2}$, and $r_{\mathsf{hi}} \cdot P_2 = D_{\mathsf{hi},2}$. Soundness follows.

A.3.3 Proof of Theorem 4.13

Fix any elements $P, C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2}, C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2} \in \mathbb{G}$ and $r_{\mathsf{lo}}, x_{\mathsf{lo}}, r_{\mathsf{hi}}, x_{\mathsf{hi}} \in \mathbb{Z}_p$ such that the ciphertext validity relation hold. Let $\mathsf{tr}^* = (t^*, Y_0^*, Y_1^*, Y_2^*, c^*, z_r^*, z_x^*)$ be any accepting transcript. By the speci cation of the protocol, the probability that an honest execution of the protocol by the prover and the veri er results in the transcript tr^* is given by

$$\Pr\left[\langle \mathcal{P}(\rho, u, w), \mathcal{V}(\rho, u)\rangle \to \mathsf{tr} \wedge \mathsf{tr} = \mathsf{tr}^*\right] = 1/p^4.$$

To prove zero-knowledge, we de ne a simulator $\mathcal{S}$ that produces such distribution without knowledge of a valid witness r_{lo}, x_{lo}, r_{hi}, and x_{hi}.

$\mathcal{S}(P, C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2}, C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2})$:

1. Sample $t, c, z_r, z_x, \leftarrow_{\mathsf{R}} \mathbb{Z}_p$
2. Let $C = C_{\mathsf{lo}} + t \cdot C_{\mathsf{hi}}$, $D_1 = D_{\mathsf{lo},1} + t \cdot D_{\mathsf{hi},1}$, and $D_2 = D_{\mathsf{lo},2} + t \cdot D_{\mathsf{hi},2}$
3. Set $Y_0 = z_r \cdot H + z_x \cdot G - c \cdot C$
4. Set $Y_1 = z_r \cdot P - c \cdot D_1$
5. Set $Y_2 = z_r \cdot P - c \cdot D_2$
6. Return $\mathsf{tr} = (w, Y_0, Y_1, c, z_r, z_x)$

The simulator $\mathcal{S}$ returns a transcript that is uniformly random given that

- $z_r \cdot H + z_x \cdot G = c \cdot C + Y_0$,
- $z_r \cdot P_1 = c \cdot D_1 + Y_1$,
- $z_r \cdot P_2 = c \cdot D_2 + Y_1$,

where $C = C_{\mathsf{lo}} + w \cdot C_{\mathsf{hi}}$, $D_1 = D_{\mathsf{lo},1} + w \cdot D_{\mathsf{hi},1}$, and $D_2 = D_{\mathsf{lo},2} + w \cdot D_{\mathsf{hi},2}$. As the variables Y_0, Y_1, and Y_2 are completely determined by t, c, z_r, z_x, we have

$$\Pr[\mathcal{S}(P, C_{\mathsf{lo}}, D_{\mathsf{lo},1}, D_{\mathsf{lo},2}, C_{\mathsf{hi}}, D_{\mathsf{hi},1}, D_{\mathsf{hi},2}) \to \mathsf{tr} \wedge \mathsf{tr} = \mathsf{tr}^*] = 1/p^4,$$

for any xed transcript tr^*. Zero-knowledge now follows.