Confidential Transactions
Theory Justification

Iftach Haitner*

August 14, 2025

Abstract

We describe a and analyze the security of a variant of the confidential transactions scheme introduced by Bünz, Agrawal, Zamani, and Boneh [FC '20].

Contents

*Stellar Development Foundation. E-mail: `iftach.haitner@stellar.org`..

1 Introduction

We describe a and analyze the security of a variant of the confidential transactions scheme introduced by Bünz, Agrawal, Zamani, and Boneh [BAZB20], which supports transactions over a block-chain without revealing the accounts value and the transferred amounts. Our scheme follows rather closely the implementation of the scheme of Solana (*Confidential Transfer*) and Aptos (*Confidential Assets*). [**Iftach's Note: is it accurate?**]

Paper organization.

Security notions and some basic building blocks used in our protocol are given in Section 2. In Section 3 we define the confidential transaction scheme, and prove its security. In **??**, we define the *Chunk-ElGamal encryption scheme* that can take the role of the additive-homomorphic scheme required for the confidential transaction scheme.

2 Preliminaries

2.1 Notation

We use calligraphic letters to denote sets, uppercase for random variables, and lowercase for integers and functions. Let $\mathbb{N}$ denote the set of natural numbers. For $n \in \mathbb{N}$, let $[n] := \{1, \ldots, n\}$ and $(n) := \{0, \ldots, n\}$. For a relation $\mathcal{R}$, let $\mathcal{L}(\mathcal{R})$ denote its underlying language, i.e., $\mathcal{L}(\mathcal{R}) := \{x \colon \exists w \colon (x, w) \in \mathcal{R}\}$. We number the element of a vector starting from 0, i.e., $v_0, v_1, \ldots,$

2.2 Homomorphic Encryption

An homomorphic encryption over $\mathbb{Z}_q$ is a triplet $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ of efficient algorithms, with the standard correctness and semantic security properties. In addition, there exist an efficient addition operation denote $+$ such that for uniformly generated public key pk, and any two messages $x_0, x_1 \in \mathbb{Z}_q$, it holds that $\mathsf{Enc}_{pk}(x_0) + \mathsf{Enc}_{pk}(x_1)$ are computationally indistinguishable from $\mathsf{Enc}_{pk}(x_0 + x_1 \bmod q)$.

2.3 Security Model

[**Iftach's Note: UC**]

[**Iftach's Note: Parties are aware of ideal functionality calls**]

[**Iftach's Note: sid**]

3 The Confidential Transactions Protocol

In this section we define the confidential transactions scheme, and prove its security. The ideal functionality for the scheme is define in Section 3.1, the protocol itself in Section 3.2, and its security is proved in Protocol 3.3.

Remark 3.1 (sid)**.** *Recall that all ideal functionalities operations below and the protocol execution get* $\mathsf{sid} \in \{0,1\}^*$ *as common input. The* sid *is stored in the log and passed as common input to the proofs. To keep the text simpler, we omit it from the following text.*

3.1 The Ideal Functionality

In this section we define the ideal functionality for the confidential transactions scheme. The functionality captures the relevant parts of the actual scheme, which typically invoked using smart contracts over a block chain. Specifically, we model the the chain as a single (honest) entity, the *chain holder*, assume money flows into the system by a single (honest) party, the *mint*, and assume fixed number of *users*. We also assume an a initial starting phase, *init*.

Functionality 3.2 ($\mathcal{F}_{\mathsf{ConfTrans}}$: Confidential transactions)**.**

Parties: Mint M, chain holder C and users $\mathsf{U}_1, \ldots, \mathsf{U}_n$.

Parameters: $p_{\mathsf{pcount}}, p_{\mathsf{size}}, q \in \mathbb{N}$.

Init. Upon receiving init from all parties: for each $i \in [n]$: set $\mathsf{avlBlance}_i \leftarrow 0, \mathsf{pndBalance}_i \leftarrow 0, \mathsf{tcount}_i \leftarrow 0$.

Mint. Upon receiving (mint, d, x) from C and M:

1. Assert $(x \in (p_{\mathsf{size}}) \wedge \mathsf{tcount}_d \leq p_{\mathsf{pcount}})$.
2. tcount_d^{++}.
3. $\mathsf{pndBalance}_d \mathrel{+}= x$.
4. Send (mint, d, x) to all parties.

Transfer. Upon receiving $(\mathsf{transfer}, d)$ from C and U_s, with U_s using private input x.

1. Assert $(x \in (p_{\mathsf{size}}) \wedge \mathsf{tcount} \leq p_{\mathsf{pcount}} \wedge \mathsf{avlBlance}_s \geq x)$.
2. tcount^{++}.
3. $\mathsf{avlBlance}_s \mathrel{-}= x$.
4. $\mathsf{pndBalance}_d \mathrel{+}= x$.
5. Send $(\mathsf{transfer}, s, d)$ to all parties, and send x to U_d.

Rollover. Upon receiving $\mathsf{rollover}$ from party U_i and C, party C

1. $\mathsf{tcount} \leftarrow 0$.
2. $\mathsf{avlBlance}_i \mathrel{+}= \mathsf{pndBalance}_i$.
3. $\mathsf{pndBalance}_i \leftarrow 0$.
4. Send $(\mathsf{rollover}, i)$ to all parties.

Withraw. Upon receiving $(\mathsf{withraw}, x)$ from party U_i and C, party C

1. Assert $(x \in (q) \wedge \mathsf{avlBlance}_i \geq x)$.

2. $\mathsf{avlBlance}_i \mathrel{-}= x$.
3. Send $(\mathsf{withraw}, i, x)$ to all parties,

Audit. [**Iftach's Note: TODO**]

3.2 The Protocol

Throughout, we fix a security parameter κ and omit is from the notation. We also fix an homomorphic encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ over $\mathbb{Z}_q$ with randomness domain $\mathcal{D}$, and denote the ciphertexts of the scheme using overlined capital letters.

We split the protocol into several sub-protocols defined below, and use the following environment to define the common part the different sub-protocols share, e.g., global parameter.

Protocol 3.3 ($\Pi_{\mathsf{ConfTrans}}$: Confidential transactions)**.**

Parties: Mint M, chain-holder C and users $\mathsf{U}_1, \ldots, \mathsf{U}_n$.

Parameters: $p_{\mathsf{pcount}}, p_{\mathsf{size}}, q \in \mathbb{N}$.

Subprotocols: See below.

3.2.1 Init

This sub-protocol is were the encryption key are sampled and shared, and the chain manager C set the initial values of the chain. The protocol uses ZKPOK proof for the relation:

Key generation: $\mathcal{R}_{\mathsf{KeyGen}} = \{(pk, w)) \colon \mathsf{KeyGen}(w) = (\cdot, pk)\}$.

Protocol 3.4 ($\Pi_{\mathsf{ConfTrans}}$.Init)**.**

Participating parties. All parties.

Proofs: $\Pi^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{KeyGen}}}$.

Operation:

1. U_i, for all $i \in [n]$:
   (a) $(pk_i, sk_i) \xleftarrow{\text{R}} \mathsf{KeyGen}(r_i)$ for $r_i \xleftarrow{\text{R}} \mathcal{D}$.
   (b) $\pi_i \xleftarrow{\text{R}} \mathsf{P}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{KeyGen}}}(pk_i, r_i)$.
   (c) Send (pk_i, π_i) to C.
2. C:
   (a) For all $i \in [n]$:
      i. $\mathsf{V}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{KeyGen}}}(pk_i, \pi_i)$.[a]
      ii. $\overline{P}_i \xleftarrow{\text{R}} \mathsf{Enc}_{pk_i}(0), \overline{A}_i \leftarrow \mathsf{Enc}_{pk_i}(0), \mathsf{tcount}_i \leftarrow 0$.

(b) Broadcast $(\mathsf{init}, \{pk_i, \overline{A}_i, \overline{P}_i\}_{i\in[n]})$

[a]Here and after, C aborts and publish the prover identity if the proof is not verified.

3.2.2 Mint

Protocol 3.5 ($\Pi_{\mathsf{ConfTrans}}$.Mint)**.**

Parties: M and C.

Common input: $d \in [n]$ and $x \in (p_{\mathsf{size}})$.

Operation: C

1. Assert $(d \in [n] \wedge \mathsf{tcount}_d \leq p_{\mathsf{pcount}} \wedge x \in (p_{\mathsf{size}}))$
2. $\overline{P}_d \mathrel{+}= \mathsf{Enc}_{pk_d}(x)$.
3. Broadcast $(\mathsf{mint}, d, x, \overline{P}_d)$.

3.2.3 Transfer

The protocol uses ZK and ZKPOK proofs for the following relations:

In range. $\mathcal{R}_{\mathsf{Rp}} = \left\{((pk, \overline{A}, b), (a, r)) \colon \mathsf{Enc}_{pk}(a; r) = \overline{A} \wedge a \in (b)\right\}$, i.e., encryption of values in $[p_{\mathsf{size}}]$.

Equality. $\mathcal{R}_{\mathsf{Eq}} = \left\{((pk_0, pk_1, \overline{A}_0, \overline{A}_1), (w_0, r_1)) \colon \mathsf{KeyGen}(w_0) = (sk_0, pk_0) \wedge \mathsf{Enc}_{pk_1}(\mathsf{Dec}_{sk_0}(\overline{A}_0); r_1) = \mathrm{A}_1\right\}$, i.e., encryptions of the same value, witness is the secret key for the pk_0 and the randomness of $\overline{A}_1$.

Protocol 3.6 ($\Pi_{\mathsf{ConfTrans}}$.Transfer)**.**

Parties: U_s and C.

Proofs: $\Pi^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Rp}}}$,$\Pi^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Eq}}}$,$\Pi^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{LrgerEq}}}$

Common input: $d \in [n]$.

U_s's private input: $x \in (p_{\mathsf{size}})$.

Operation:

1. U_s:
   (a) $\overline{X}_d \overset{\mathrm{R}}{\leftarrow} \mathsf{Enc}_{pk_d}(x; r_d)$ for $r_d \overset{\mathrm{R}}{\leftarrow} \mathcal{D}$.
   (b) $\pi^{\mathsf{Rp}} \overset{\mathrm{R}}{\leftarrow} \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Rp}}}((pk_d, \overline{X}_d, p_{\mathsf{size}}), (x, r_d))$.
   (c) $a \leftarrow \mathsf{Dec}_{sk_s}(\overline{A}_s)$.
   (d) $\overline{A}'_s \overset{\mathrm{R}}{\leftarrow} \mathsf{Enc}_{pk_s}(a - x)$.

(e) $\pi^{\mathsf{Eq}} \xleftarrow{\text{R}} \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Eq}}}((pk_s, pk_d, \overline{A}_s - \overline{A}'_s, \overline{X}_d), (x, sk_s, r_d))$.

(f) Send $(\overline{A}'_s, \overline{X}_d, \pi^{\mathsf{Rp}}, \pi^{\mathsf{Eq}})$ to C.

2. C:

(a) $\mathsf{Assert}(\mathsf{tcount}_d \leq p_{\mathsf{pcount}})$.

(b) $\mathsf{V}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{Rp}}}((pk_d, \overline{X}_d, p_{\mathsf{size}}), \pi^{\mathsf{Rp}})$ and $\mathsf{V}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Eq}}}((pk_s, pk_d, \overline{A}_s - \overline{A}'_s, \overline{X}_d), \pi^{\mathsf{Eq}})$.

(c) $\overline{A}_s \leftarrow \overline{A}'_s$.

(d) $\overline{P}_d \mathrel{+}= X_d$.

(e) tcount_d^{++}.

(f) Broadcast $(\mathsf{transfer}, s, d, \overline{P}_d)$.

3.2.4 Rollover

Protocol 3.7 ($\Pi_{\mathsf{ConfTrans}}$.Rollover)**.**

Parties. U_i and C.

Operation: C:

1. $\overline{A}_i \mathrel{+}= \overline{P}_i$.
2. $\overline{P}_i \mathrel{-}= \overline{P}_i$.
3. $\mathsf{tcount}_i \leftarrow 0$.
4. Broadcast $(\mathsf{rollover}, i)$

3.2.5 Audit

[**Iftach's Note: TODO**]

3.3 Security of Protocol 3.3

Theorem 3.8 (Security of Protocol 3.3)**.** *Assuming* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is CPA secure, then Protocol 3.3 UC-realizes (with static security*[**Iftach's Note: ?**]*) Functionality 3.2 against semi-honest chain holder and mint.*

Proof. [**Iftach's Note: TODO**] □

4 The Chunk-ElGamal Encryption Scheme

In this section, we define the (almost) additive homomorphic encryption scheme based on ElGamal multiplicative homomorphic encryption scheme. [**Iftach's Note: give citations**] The idea is to bootstrap the so-called *ElGamal in-the exponent* additive homomorphic encryption/commitment scheme,[1] which in turn is based on the ElGamal multiplicative homomorphic encryption scheme, that lacks efficient decryption algorithm, by splitting the plain text into small "chunks". That is, we present a message $a \in Z_t$ as $\sum_{i \in (t/c)} 2^{ic} \cdot a_i$, where c, the chunk size, is some inEgEr dividing t, and encrypt each of the a_i using additive homomorphic EG. To decry $\overline{A} = (A_0, \ldots, A_{t/c})$, one

1. Decrypt each A_i to get $a_i \cdot G$.
2. Use brute force to find a.[2]
3. Reconstruct a.

In Section 4.1, we formally define the ElGamal in-the-exponent scheme, and a few ZK proofs for the NP-relations the scheme induces. Actually, we use a "twisted" variant of this scheme, that supports somewhat more efficient proofs in our settings. The chunk ElGamal scheme is defined in Section 4.2, and the related ZK proofs are defined in Section 4.3. Finally, in Section 4.4 we explain how to adjust Protocol 3.3 to work with this almost homomorphic scheme.

4.1 Twisted ElGamal In-the-Exponent Encryption Scheme

Throughout we fix a cyclic additive q-size group $\mathcal{G}$ with generator G. The twisted ElGamal in-the-exponent encryption scheme $(\mathsf{EgGen}, \mathsf{EgEnc}, \mathsf{EgDec})$ is defined below. Note that it gets $H \in \mathcal{G}$ as an additional parameter.

Algorithm 4.1 (($\mathsf{EgGen}, \mathsf{EgEnc}, \mathsf{EgDec}$): Twisted ElGamal in-the-exponent encryption)**.**

Key generation: $\mathsf{EgGen}(1^b, H)$ *samples* $e \xleftarrow{R} \mathbb{Z}_q$*, and outputs* $(sk \leftarrow e, pk \leftarrow (1^b, H, E \leftarrow e^{-1} \cdot H))$.

Encryption: $\mathsf{EgEnc}_{(H,E)}(a)$ *samples* $r \xleftarrow{R} \mathbb{Z}_q$*, and outputs* $\widetilde{A} \leftarrow (r \cdot E, \mathsf{Ped}_H(a; r))$*, for* $\mathsf{Ped}_H(a; r) := r \cdot H + a \cdot G$.

Decryption: $\mathsf{EgDec}_{(1^b,H,e)}(\widetilde{A})$,

1. *Let* $M \leftarrow \widetilde{A}_1 - e \cdot \widetilde{A}_0$.
2. *Find (using brute force)* $m \in (-b, b) \in \mathbb{Z}_q$ *so that* $m \cdot G = M$*. Abort if no such* m *exists.*
3. *Output* m.

Addition: Addition over $\mathcal{G}^2$.[a]

Minus: The inverse in $\mathcal{G}^2$.

[a] For $\widetilde{A}, \widetilde{B} \in \mathcal{G}^2$: $\widetilde{A} + \widetilde{B} := (\widetilde{A}_0 + \widetilde{B}_0, \widetilde{A}_1 + \widetilde{B}_1)$.

When clear from the context, will omit the parameters 1^b from the public key of the scheme.

Namely, the right hand side if a twisted ElGamal ciphertext is just a Pedersen commitment [Ped91] (of the same palaintext). This change enable using proofs that support Pedersen commitment on the ciphertext without changing it, but doing that should be done with care.

[1] It is called ElGamal "in-the-exponent" due to typical multiplicative group notation. Here use additive group notation, but keep the name for historical reason.

[2] One can use processing to speed-up this part from c group operations to $\sqrt{c}$ operations, or even [**Iftach's Note: cite**] to $\sqrt[3]{c}$.

1. The parameter H should be chosen so that the prover does not know that discrete log of H with respect to G. (Otherwise, a proof of the Pedersen part means nothing).

2. When using a proof of the Pedersen part (which should always be POK, since Pedersen is perfectly hiding), it should *always* be accompanied with a POK of the palaintext for the whole EG encryption (otherwise, the palaintext and randomness extracted by the Pedersen POK, might be inconsistent with EG publicj key)

Theorem 4.2 (Security of twisted-ElGamal in-the-exponent)**.** *Assuming DDH is hard over $\mathcal{G}$, then* **??** *is a perfectly binding, semantically secure additively homomorphic scheme over $\mathbb{Z}_q$, with the following caveat: the description only guaranteed to work on encryptions of explain in* (b)*, for* 1^b *being the parameter of the key generation algorithm:*

4.1.1 Zero-Knowledge Proofs

In Section 4.2 we make use of ZK proofs for the following relations regrading the above scheme.

Knowledge of secret key.

Task: ZKPOK for $\mathcal{R}_{\mathsf{EgKG}} = \{(E, e)) \colon e \cdot G = E\}$.

Protocol: The standard Schnorr proof for discrete log [Sho00].

Knowledge of plain text.

Task: ZKPOK for $\mathcal{R}_{\mathsf{EgEnc}} = \left\{((H, E, \widetilde{A}), (a, r)) \colon \mathsf{EgEnc}_{(H,E)}(a; r) = \widetilde{A}\right\}$.

Protocol: [HLNR23, Protocol A.2].

Equality.

Task: ZKP for $\mathcal{R}_{\mathsf{EgEq}} = \left\{((H, E_0, E_1, \widetilde{A}_0, \widetilde{A}_1), (e_0, r_1)) \colon e_0 \cdot G = E_0 \wedge \widehat{\mathsf{EgEnc}}_{H,E_1}(\widehat{\mathsf{EgDec}}_{e_0}(\overline{A}_0); r_1) = \mathrm{A}_1\right\}$.

letting $\widehat{\mathsf{EgEnc}}_{H,E}(A; r) := (r \cdot E, r \cdot H + A)$ and $\widehat{\mathsf{EgDec}}_e(\widetilde{A}) := \widetilde{A}_1 - e \cdot \widetilde{A}_0$. Namely, encryption and decryption of group elements.

Letting $\widehat{\mathsf{EgEnc}}_E(A; r) := (r \cdot G, r \cdot E + A)$ and $\widehat{\mathsf{EgDec}}_e(\widetilde{A}) := \widetilde{A}_1 - e \cdot \widetilde{A}_0$.

Protocol:

Protocol 4.3 (Equality proof)**.**

1. P:
   (a) Sample $e_0', r_1' \xleftarrow{\text{R}} \mathbb{Z}_q$.
   (b) Let $E_0' \leftarrow e_0' \cdot G$.
   (c) Let $A \leftarrow \widehat{\mathsf{EgDec}}_{e_0}(\overline{A}_0)$.
   (d) Let $(L_0, R_0) \leftarrow \overline{A}_0$.
   (e) Let $R_0' \leftarrow e_0' \cdot L_0 + A$ and $\overline{A}_1' \xleftarrow{\text{R}} \widehat{\mathsf{EgEnc}}_{(H,E_1)}(A; r_1')$.
   (f) Send $(E_0', R_0', \overline{A}_1')$ to V.

2. V: Send $t \xleftarrow{R} \mathbb{Z}_q$ to P.
3. P:
   (a) Send $(e_0'' \leftarrow t \cdot e_0 + e_0', r_1'' \leftarrow r_1 + r_1')$ to V.
4. V: Verify
   (a) $e_0'' \cdot G = t \cdot E_0 + E_0'$.
   (b) $\widehat{\mathsf{EgEnc}}_{E_1}(t \cdot R_0 + R_0' - e_0'' \cdot L_0; r_1'')) = t \cdot \overline{A}_1 + \overline{A}_1'$.

Proof. **Correctness** clear.

Special soudness: clear [**Iftach's Note: verify**].

ZK:

1. Sample $e_0'', r_1'', t \xleftarrow{R} \mathbb{Z}_q$.
2. Sample $R_0'', E_0'', \overline{A}_1'' \xleftarrow{R} \mathcal{G}$.
3. Let $E_0' \leftarrow E_0'' - t \cdot E_0$.

□

In range.

Task: ZKP for $\mathcal{R}_{\mathsf{EgRp}} = \left\{((H, E, \widetilde{A}, b), (a, r)) \colon \mathsf{EgEnc}_{(H,E)}(a; r) = \widetilde{A} \wedge a \in (b)\right\}$.

Protocol: The proof consists of two parts:

1. *Bullet proofs* [BBBPWM18] (which is ZKPOK) on $\widetilde{A}_1$ (i.e., right hand side of $\widetilde{A}$).
2. $\mathcal{R}_{\mathsf{EgEnc}}$ ZKPOK for the whole $\widetilde{A}$.

Remark 4.4 (Efficiency)**.** *The saving in the above proofs comparing to using non twisted EG, is in not sending two additional group elements in the In-range proofs; we do not have to provide the Pedersen commitment (one group element) for the plain text and prove it is consistent with the EG encryption (three group elements). Yet, we still have to preform an EG POK proof (two group elements).*

4.2 The Chunk-ElGamal Scheme

In the following we fix $t, c \in \mathbb{N}$ with $t \leq q$ and $\ell \leftarrow t/c \in \mathbb{N}$. The chunk ElGamal encryption scheme is defined as follows:

Definition 4.5 (Base factorization)**.** *For $a \in \mathbb{Z}_q$ let $a_0, \ldots, a_{\ell-1}$ so that $a = \sum_{i \in (\ell)} 2^{ic} \cdot a_i$.*

Algorithm 4.6 (($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$): Chunk ElGamal adaptively homomorphic encryption)**.**

Key generation: $\mathsf{KeyGen}(1^b, H)$*: act as* $\mathsf{EgGen}(1^b, H)$.

Encryiption: $\mathsf{Enc}_{pk}(a)$

1. *Compute* $(a_0, \ldots, a_{\ell-1}) \leftarrow \mathsf{baseFcts}(a)$.
2. *For each* $i \in (\ell)$*: let* $\widetilde{A}_i \xleftarrow{R} \mathsf{EgEnc}_{pk}(a_i)$.

3. *Output* $\overline{A} \leftarrow (\widetilde{A}_0, \ldots, \widetilde{A}_{\ell-1})$.

Decription: $\mathsf{Dec}_{sk}(\overline{M}, b)$

1. *For each* $i \in (\ell)$*: let* $m_i \overset{R}{\leftarrow} \mathsf{EgDec}_{sk}(\overline{M}_i, b)$.
2. *Let* $m \leftarrow \sum_{i \in (\ell)} 2^{ic} \cdot m_i$.
3. *Output* m.

Addition: Vector addition.[a]

Minus: Vector negation.

[a]For $\overline{A}, \overline{B} \in (\mathcal{G}^2)^\ell$, $\overline{A} + \overline{B} := (\widetilde{A}_0 + \widetilde{B}_0, \ldots, \widetilde{A}_{\ell-1} + \widetilde{B}_{\ell-1})$.

Theorem 4.7 (Security of Chunk ElGamal)**.** *Assuming* DDH *is hard over* $\mathcal{G}$*, then Algorithm 4.6 is a perfectly binding, semantically secure additively homophobic scheme over* $\mathbb{Z}_q$*, with the following caveat work on encryptions of plaintext a so that* $\mathsf{baseFcts}(a) \in (-b, b)^\ell$ *(*1^b *being the input of the key generation algorithm).*

4.3 Zero-Knowledge Proofs

In this section, we define the ZK and POK proofs used in Section 3. In the following, we omit the parameter b from the input list of Dec. We will address its value in Section 4.4.

Knowledge of secret key.

Task: ZKPOK for $\mathcal{R}_{\mathsf{KeyGen}} = \{(pk, w)) \colon \mathsf{KeyGen}(w) = (\cdot, pk)\}$.
Protocol: Same as $\Pi^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{EgKG}}}$.

Knowledge of plain text.

Task: ZKPOK for $\mathcal{R}_{\mathsf{Enc}} = \{((pk, A), (a, r)) \colon \mathsf{Enc}_{pk}(a; r) = A\}$.
Protocol:

P: On input $((pk, \overline{A}), (\overline{a}, \overline{r})$.
1. For each $i \in (\ell)$: let $\pi_i \leftarrow \Pi^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{EgEnc}}}((pk, \overline{A}_i), (\overline{a}_i, \overline{r}_i))$.
2. Output $\pi \leftarrow (\pi_0, \ldots, \pi_{\ell-1})$.

V: On input $((pk, \overline{A}), \pi = (\pi_0, \ldots, \pi_{\ell-1}))$: Accept iff $\mathsf{V}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgEnc}}}((pk, \widetilde{A}_i), \pi_i)$ for all $i \in (\ell)$.

Proof. Immediate. □

Equality.

Task: ZKP for $\mathcal{R}_{\mathsf{Eq}} = \big\{((pk_0, pk_1, \overline{A}_0, \overline{A}_1), (w_0, r_1)) \colon \mathsf{KeyGen}(w_0) = (sk_0, pk_0) \wedge \mathsf{Enc}_{pk_1}(\mathsf{Dec}_{sk_0}(\overline{A}_0); r_1) = \mathrm{A}$
Protocol:

P: On input $((pk_0, pk_1, \overline{A}_0, \overline{A}_1), (e_0, \overline{r}_1))$:

1. Let $a \leftarrow \sum_{i \in (\ell)} 2^c \cdot \overline{a}_i$.
2. For both $j \in \{0,1\}$: $\widetilde{A}_j \leftarrow \sum_i 2^c \cdot (\overline{A}_j)_i$.
3. $r_1 \leftarrow \sum_{i \in (\ell)} 2^c \cdot (\overline{r}_1)_i$.
4. Output $\pi \leftarrow \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgEq}}}((pk, \widetilde{A}_0, \widetilde{A}_1), (e_0, r_1))$.

V**:** On input $((pk_0, pk_1, \overline{A}_0, \overline{A}_1), \pi)$:

1. Generate $\widetilde{A}_0$ and $\widetilde{A}_1$ as done by P.
2. Apply $\mathsf{V}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgEq}}}((pk, \widetilde{A}_0, \widetilde{A}_1), \pi)$.

Proof. [**Iftach's Note: TODO**] ☐

In range.

Task: ZK for $\mathcal{R}_{\mathsf{Rp}} = \big\{((pk, \overline{A}, b), (a, r)) \colon \mathsf{Enc}_{pk}(a; r) = \overline{A} \wedge a \in (b)\big\}$.

Protocol:

P**:** On input $((pk, \overline{A}, b), (\overline{a}, \overline{r}))$:

1. $a \leftarrow \sum_{i \in (\ell)} 2^c \cdot \overline{a}_i$.
2. $\widetilde{A} \leftarrow \sum_{i \in (\ell)} 2^c \cdot \overline{A}_i$.
3. $r \leftarrow \sum_{i \in (\ell)} 2^c \cdot \overline{r}_i$.
4. Output $\pi \leftarrow \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgRp}}}((pk, \widetilde{A}, b), (a, r))$.

V**:** On input $((pk, \overline{A}, b), \pi)$:

1. Generate $\widetilde{A}$ as by P.
2. Output $\mathsf{V}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgRp}}}((pk, \widetilde{A}, b), \pi)$.

Proof. [**Iftach's Note: TODO**] ☐

Freshness. Proving that each of the entries of the ciphertext are small.

Task: ZKP for $\mathcal{R}_{\mathsf{EgFsh}} = \big\{((pk, \overline{A}, b), (\overline{a}, \overline{r})) \colon \forall i \in (\ell) \colon \mathsf{EgEnc}_{pk}(\overline{a}_i; \overline{r}_i) = \overline{A}_i \wedge \overline{a}_i \in (b)\big\}$.

Protocol:

P**:** On input $((pk, \overline{A}, b), (\overline{a}, \overline{r})$:

1. For each $i \in (\ell)$: $\pi_i \leftarrow \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgRp}}}((pk, \widetilde{A}_i, b), (\overline{a}_i, \overline{r}_i))$.
2. Output $\pi \leftarrow (\pi_0, \ldots, \pi_{\ell-1})$.

V**:** On input $((pk, \overline{A}, b), \pi = (\pi_0, \ldots, \pi_{\ell-1}))$: Accept iff $\mathsf{V}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgRp}}}((pk, \widetilde{A}_i, b), \pi_i)$ for all $i \in (\ell)$.

Proof. [**Iftach's Note: TODO**] ☐

4.4 Adjusting Protocol 3.3

Since the decryption procedure of of chunk-ElGamal encryption scheme only guarantees to work on certain type of ciphertexts, see Theorem 4.7 and take a group element, i.e., H, as an additional parameter, instantiating Protocol 3.3 with this scheme requires some adjustments.

Init: (a) The parties call an ideal functionality that returns $H \overset{R}{\leftarrow} \mathcal{G}$.

(b) Each U_i sets the parameters of the encryption key generation algorithm to $(1^b, H)$, for $b \leftarrow 2^c \cdot p_{\mathsf{pcount}}$.

Transfer. The sender also provide proofs that X_d is fresh (i.e., using $\mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgFsh}}}$ with parameter $b \leftarrow 2^c$).

Rollover: The rollover over operation should be updated to allow the account holder to "normalize" its active balance: to make it fresh. Specifically

(a) U_i:

i. Decrypt $\overline{P}_i$ and $\overline{B}_i$ to get value (p_i, r_i) and (b_i, w_i) respectively.

ii. Generate a fresh encryption $\overline{B}'_i$ of $(p_i + b_i)$ and $\overline{P}'_i$ of 0.

iii. Generate a proof π_{Eq} (i.e., using $\mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Eq}}}$) that $\overline{P}_i + \overline{B}_i = \overline{P}'_i + \overline{B}'_i$.

iv. Send $(\overline{P}'_i, \overline{B}'_i, \pi_{\mathsf{Eq}}, \pi^P_{\mathsf{Fsh}}, \pi^B_{\mathsf{Fsh}})$ to C.

v. Send $(\overline{P}'_i, \overline{B}'_i, \pi_{\mathsf{Eq}})$ to C.

(b) C:

i. Verify π_{Eq}.

ii. Set $\overline{P}_i \leftarrow \overline{P}'_i$ and $\overline{B}_i \leftarrow \overline{B}'_i$.

iii. Continue as in the original protocol.

4.5 Efficient Improvements

1. The bullet proofs used in the In range and the freshness proofs can be batched.
2. The Schnorr proofs can be batched. In particular, the one performed in the different In-range proofs.

References

[BAZB20] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. "Zether: Towards Privacy in a Smart Contract World". In: *Financial Cryptography and Data Security*. 2020, pp. 423–443 (cit. on pp. 1, 2).

[BBBPWM18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020 (cit. on p. 9).

[2] Can be implemented using a proper protocol, ot sampled by a trusted setup.

[HLNR23] Iftach Haitner, Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. *Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody*. Tech. rep. 2018/987. Cryptology ePrint Archive, 2023 (cit. on p. 8).

[Ped91] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: 1991, pp. 129–140 (cit. on p. 7).

[Sho00] Victor Shoup. "A Composition Theorem for Universal One-Way Hash Functions". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. 2000 (cit. on p. 8).