Confidential Transactions
Theory Justification

Iftach Haitner[*]

September 2, 2025

Abstract

We describe a and analyze the security of a variant of the confidential transactions scheme introduced by **BunzAZB20** [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 **BunzAZB20**, 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 $\mathsf{Enc}_{pk}(x_0 + x_1 \bmod q)$.

Remark 2.1 (Implicit randomness)**.** *When calling* KeyGen *or* Enc*, or any other randomized algorithms, we sometimes explicitly provide the random coins. When we do not, it means that the algorithm sample them by itself.*

2.3 Security Model

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

[**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. Assert$(\mathsf{avlBlance}_i \leq q/4)$
2. $\mathsf{tcount} \leftarrow 0$.
3. $\mathsf{avlBlance}_i \mathrel{+}= \mathsf{pndBalance}_i$.
4. $\mathsf{pndBalance}_i \leftarrow 0$.
5. 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) $(pk_i, sk_i) \overset{\text{R}}{\leftarrow} \mathsf{KeyGen}(r_i)$ for $r_i \overset{\text{R}}{\leftarrow} \mathcal{D}$.
   (b) $\pi_i \overset{\text{R}}{\leftarrow} \mathsf{P}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{KeyGen}}}(pk_i, r_i)$. [**Iftach's Note: Probably needed for the security proof.**
   ]
   (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. $\overline{P}_i \overset{\mathrm{R}}{\leftarrow} \mathsf{Enc}_{pk_i}(0;0), \overline{A}_i \overset{\mathrm{R}}{\leftarrow} \mathsf{Enc}_{pk_i}(0;0), \mathsf{tcount}_i \leftarrow 0$.

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

In range using secret key. $\mathcal{R}_{\mathsf{RpSk}} = \{((pk, \overline{A}, b), w) \colon \mathsf{KeyGen}(w) = (sk, pk) \wedge \mathsf{Dec}_{sk}(\overline{A}) \in (b)\}$, i.e., encryption of values in $[p_{\mathsf{size}}]$, witness is secret key.

Remark 3.6. *To prove this relation, see Section 4.2.1, the prover decrypts the ciphertext, encrypt it again using fresh randomness, and use this randomness as the witness for the proof above. By storing the random coins used to generate the original ciphertext, one significantly reduce the proof's cost.*

Equality. $\mathcal{R}_{\mathsf{Eq}} = \big\{((pk_0, pk_1, \overline{A}_0, \overline{A}_1), (a, r_0, r_1)) \colon \forall i \in \{0,1\}\ \mathsf{Enc}_{pk_i}(a; r_i) = \mathrm{A}_i\big\}$, i.e., encryptions of the same value, witness is the secret key for the pk_0 and the randomness of $\overline{A}_1$.

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

Parties: U_s and C.

Proofs: $\Pi^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{Rp}}}$,$\Pi^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{RpSk}}}$,$\Pi^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Eq}}}$

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

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

Operation:

1. U_s:

   (a) $\overline{X}_s \overset{\mathrm{R}}{\leftarrow} \mathsf{Enc}_{pk_d}(x; r_s)$ for $r_s \overset{\mathrm{R}}{\leftarrow} \mathcal{D}$.

   (b) $\pi^{\mathsf{Rp}} \overset{\mathrm{R}}{\leftarrow} \mathsf{P}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{Rp}}}((pk_s, \overline{X}_s, p_{\mathsf{size}}), (x, r_s))$.

   (c) $\overline{X}_d \overset{\mathrm{R}}{\leftarrow} \mathsf{Enc}_{pk_d}(x; r_d)$ for $r_d \overset{\mathrm{R}}{\leftarrow} \mathcal{D}$.

   (d) $\pi^{\mathsf{Eq}} \overset{\mathrm{R}}{\leftarrow} \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Eq}}}((pk_s, pk_d, \overline{X}_s, \overline{X}_d), (x, r_s, r_d))$.

   (e) $\pi^{\mathsf{RpSk}} \overset{\mathrm{R}}{\leftarrow} \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{RpSk}}}((pk_s, \overline{A}_s - \overline{X}_s, q/2), sk_s)$.[a]

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

2. C:

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

   (b) $\mathsf{V}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{Rp}}}((pk_d, \overline{X}_d, p_{\mathsf{size}}), \pi^{\mathsf{Rp}})$,
   $\mathsf{V}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{Eq}}}((pk_s, pk_d, \overline{X}_s, \overline{X}_d), \pi^{\mathsf{Eq}})$ and
   $\mathsf{V}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{RpSk}}}((pk_s, \overline{A}_s - \overline{X}_s, q/2), \pi^{\mathsf{RpSk}})$.

   (c) $\overline{A}_s \mathrel{-}= \overline{X}_s$.

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

   (e) tcount_d^{++}.

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

[a]Assuming there are never at most q/p_{size} increments of $\overline{A}_s$, it holds that $\overline{A}_s$ is smaller than $q/2$. Since $p_{\mathsf{size}} \leq q/2$ as well, this check is equivalent to "$\overline{A}_s - \overline{X}_s$ encrypts a positive value". [**Iftach's Note: Rephrase**]

3.2.4 Rollover

Protocol 3.8 ($\Pi_{\mathsf{ConfTrans}}$.Rollover)**.**

Parties. U_i and C.

Operation: C:

1. $\overline{A}_i \mathrel{+}= \overline{P}_i$.

2. $\overline{P}_i \leftarrow \mathsf{Enc}_{pk_i}(0; 0)$.

3. $\mathsf{tcount}_i \leftarrow 0$.

4. Broadcast $(\mathsf{rollover}, i)$.[a]

[a]No need to publish the new $\overline{P}_i$, since everyone can compute it.

3.2.5 Audit

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

3.3 Security of Protocol 3.3

Theorem 3.9 (Security of Protocol 3.3)**.** *Assuming* (KeyGen, Enc, 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 4.2, 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 4.3, and the related ZK proofs are defined in Section 4.3.1. Finally, in Section 4.4 we explain how to adjust Protocol 3.3 to work with this almost homomorphic scheme.

4.1 Additional Preliminaries

We will use access to an ideal functionality RanElm that returns a random element of $\mathcal{G}$.[3] We will also use *Pedersen commitments* defined by $\mathsf{Ped}_H(a; r) := r \cdot H + a \cdot G$.

Zero-knowledge proofs. We will also use zero-knowledge proofs for the following relations.

Knowledge of discrete log.

Task: ZKPOK for $\mathcal{R}_{\mathsf{DL}} = \{(A, a)) \colon A = a \cdot G\}$.

[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 $\sqrt[3]{c}$.

[3] Can be implemented using a proper protocol, sampled by a trusted setup or using a random oracle.

Protocol: The standard Schnorr proof for discrete log, e.g., [**Shoup00b**].

Knowledge of plaintext of Pedersen commitent.

Task: ZKPOK for $\mathcal{R}_{\mathsf{Ped}} = \{((H, A), (a, r))) \colon A = \mathsf{Ped}_H(a; r)\}$.

Protocol: See [**HaitnerLNR23**].

Pedersen equality.

Task: ZKPOK for $\mathcal{R}_{\mathsf{PedEq}} = \{((H, A_0, A_1), (a, r_0, r_1))) \colon \forall j \in \{0, 1\} \mathsf{Ped}_H(a; r_j) = A_j\}$.

Protocol: This is just the discrete log protocol for $A_1 - A_0$ with witness and $r_1 - r_0$.

Pedersen and group equality.

Task: ZKPOK for $\mathcal{R}_{\mathsf{PedGrEq}} = \{((H, A, B), (a, r)) \colon \mathsf{Ped}_H(a; r) = A \wedge a \cdot G = B\}$.

Protocol: Concatenation of the $\mathcal{R}_{\mathsf{Ped}}$ and $\mathcal{R}_{\mathsf{DL}}$ with the same challenge.

4.2 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. The ciphertext of the encryption scheme are elements of $\mathcal{G} \times \mathcal{G}$. We will mark such ciphertexts using tilde, and address the left-hand side and right-hand side of such a ciphertext $\widetilde{A}$, by $\widetilde{A}_L$ and $\widetilde{A}_R$, respectively.

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

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

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

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

1. *Let* $M \leftarrow \widetilde{A}_R - e \cdot \widetilde{A}_L$.
2. *Find (using brute force)* $m \in (b) \in \mathbb{Z}_q$ *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, i.e., minus, in $\mathcal{G}^2$.

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

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 [**Pedersen91**]. This change enable using proofs that support Pedersen commitment on the ciphertext without changing it, but doing that should be done with care.

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 Algorithm 4.1 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:*

4.2.1 Zero-Knowledge Proofs

[**Iftach's Note: Remove unused proofs**]

In Section 4.3 we make use of ZK proofs for the following relations regrading the above scheme. We use the following notation:

Notation 4.3. *Let* $\widehat{\mathsf{EgDec}}_e(\widetilde{A}) := \widetilde{A}_1 - e \cdot \widetilde{A}_0$.

Namely, decryption without finding discrete log.

Knowledge of secret key.

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

Protocol: Same protocol as for $\mathcal{R}_{\mathsf{DL}}$ (see Section 4.1).

Knowledge of plain text.

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

Protocol: [**HaitnerLNR23**].

Conssitency with plaintext using secret key.

Task: ZKPOK for $\mathcal{R}_{\mathsf{EgCons}} = \{((H, E, \widetilde{A}, a), e) \colon e \cdot E \wedge \widehat{\mathsf{EgDec}}_e(\widetilde{A}) = a \cdot G\}$.

Protocol: P proves that $\widetilde{B} \leftarrow (\widetilde{A}_L, \widetilde{A}_R - a \cdot G)$ is an encryption of 0 under E. Specifically, that it knows e so that $e \cdot \widetilde{B}_R = \widetilde{B}_L$. This is just the standard Schnorr proof for discrete log.

Equality.

Task: ZKP for $\mathcal{R}_{\mathsf{EgEq}} = \{(((H, E_0, E_1, \widetilde{A}_0, \widetilde{A}_1), (a, r_0, r_1)) \colon \forall i \in \{0,1\} \colon \mathsf{EgEnc}_{(H,E_i)}(a; r_i) = \widetilde{A}_i\}$.

Namely, the relation is of ciphertexts that encrypt the same value under the different public key, where the witness is the plaintext and the two randomness used by the encryption algorithm.

Protocol: The Sigma protocol for this relation concatenates, with the same challenge t, two proofs of $\mathcal{R}_{\mathsf{EgEnc}}$.

Protocol for the case $E_0 = E_1$: For this case, the proof is somewhat simpler. P proves that $\widetilde{B} \leftarrow \widetilde{A}_1 - \widetilde{A}_0$ is an encryption of 0 under E. Specifically, that it knows r (i.e., $r \leftarrow r_1 - r_0$) so that $\widetilde{B} = r \cdot (E, H)$. The Sigma protocol for this relation concatenates, with the same challenge t, two discrete log proofs. (Thus, prover only sends two group elements, vs four above.)

Equality using secret key.

Task: ZKP for $\mathcal{R}_{\mathsf{EgEqSk}} = \{((H, E, \widetilde{A}_0, \widetilde{A}_1), e) \colon e \cdot G = E \wedge \widehat{\mathsf{EgDec}}_e(\overline{A}_0) = \widehat{\mathsf{EgDec}}_e(\overline{A}_1)\}$. Namely, the relation is of ciphertexts that encrypt the same value under the same public key. The witness is the secret key.

Protocol: P proves that $\widetilde{B} \leftarrow \widetilde{A}_1 - \widetilde{A}_0$ is an encryption of 0 under E. Specifically, that it knows e so that $e \cdot \widetilde{B}_R = \widetilde{B}_L$. This is just the standard Schnorr proof for discrete log.

Equality with Pedersen commitment.

Task: ZKP for $\mathcal{R}_{\mathsf{EgEqPed}} = \{((H, E, \widetilde{A}, A), (a, r, r')) \colon \mathsf{EgEnc}_{H,E}(a; r) = \widetilde{A} \wedge \mathsf{Ped}_H(a; r') = A\}$.

Protocol: The Sigma protocol for this relation concatenates, with the same challenge t, a POK of $\mathcal{R}_{\mathsf{EgEnc}}$ and of $\mathcal{R}_{\mathsf{Ped}}$.

In range.

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

Protocol: The proof consists of two parts:

1. *Bullet proofs* [**SP: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 $\widetilde{A}$.

Efficiency. The saving in the above protocol comparing to using non-twisted EG, is in two group elements in the proof size; the prover does not have to provide a new Pedersen commitment (one group element) for the plain text and prove it is consistent with the EG encryption (three group elements). Yet, he still has to preform an EG POK proof (two group elements).

4.3 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 4.4 (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 4.5 (($\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$): 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* $\widetilde{A}_i \overset{R}{\leftarrow} \mathsf{EgEnc}_{pk}(a_i)$.
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 4.6 (Security of Chunk ElGamal)**.** *Assuming* DDH *is hard over* $\mathcal{G}$*, then Algorithm 4.5 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).*

4.3.1 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 $\mathcal{R}_{\mathsf{Enc}} = \{((pk, \overline{A}), (a, r)) \colon \mathsf{Enc}_{pk}(a; r) = \overline{A}\}$.
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 $\mathcal{R}_{\mathsf{Eq}} = \big\{((pk_0, pk_1, \overline{A}_0, \overline{A}_1), (a, r_0, r_1)) \colon \forall i \in \{0, 1\}\ \mathsf{Enc}_{pk_i}(a; r_i) = \mathrm{A}_i\big\}$.

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**] □

Equality using secret key.

Task: ZKP for $\mathcal{R}_{\mathsf{EqSk}} = \{((pk, \overline{A}_0, \overline{A}_1), w) \colon \mathsf{KeyGen}(w) = (sk, pk) \wedge \mathsf{Dec}_{sk}(\overline{A}_0) = \mathsf{Dec}_{sk}(\overline{A}_1)\}$. Namely, the relation is of ciphertexts that encrypt the same value under the same public key. The witness is the secret key.

Protocol:

- P: On input $((E, \overline{A}_0, \overline{A}_1), e)$:
  1. For both $j \in \{0,1\}$: $\widetilde{A}_j \leftarrow \sum_{i\in(\ell)} 2^c \cdot (\overline{A}_j)_i$.
  2. Let $\pi^{\mathsf{EgEqSk}} \overset{R}{\leftarrow} \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgEqSk}}}((E, \widetilde{A}_0, \widetilde{A}_1), e)$.
  3. Send π^{EgEqSk} to V.
- V: On input $((E, \overline{A}_0, \overline{A}_1), \pi^{\mathsf{EgEqSk}})$:
  1. Generate $\widetilde{A}_0, \widetilde{A}_1$ as by P.
  2. Call $\mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgEqSk}}}((E, \widetilde{A}_0, \widetilde{A}_1), \pi^{\mathsf{EgEqSk}})$.

In range.

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

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**] □

In-range proof using sceret key.

Task: POK for $\mathcal{R}_{\mathsf{RpSk}} = \{((pk, \overline{A}, b), w) \colon \mathsf{KeyGen}(w) = (sk, pk) \wedge \mathsf{Dec}_{sk}(\overline{A}) \in (b)\}$.

Protocol: Similar line to the protocol for $\mathcal{R}_{\mathsf{Rp}}$, but the prover decrypt $\widetilde{A}$, encrypt the plaintext sing fresh randomness, and continues as above.

P: On input $((E, \overline{A}, b), e)$:

1. $\widetilde{A} \leftarrow \sum_{i \in (\ell)} 2^c \cdot \overline{A}_i$.
2. $a \leftarrow \mathsf{Dec}_e(\widetilde{A})$.
3. $\widetilde{A}' \leftarrow \mathsf{Enc}_E(a; r)$ for $r \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$.
4. Let $\pi^{\mathsf{EgEqSk}} \overset{\mathrm{R}}{\leftarrow} \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgEqSk}}}((E, \widetilde{A}, \widetilde{A}'), e)$.
5. Let $\pi^{\mathsf{EgRp}} \overset{\mathrm{R}}{\leftarrow} \mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgRp}}}((E, \widetilde{A}', b), (a, r))$.
6. Send $(\pi^{\mathsf{EgEqSk}}, \pi^{\mathsf{EgRp}})$ to V.

V: On input $((pk, \widetilde{A}', b), \pi^{\mathsf{EgEqSk}}, \pi^{\mathsf{EgRp}})$:

1. Generate $\widetilde{A}$ as by P.
2. Call $\mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgEqSk}}}((E, \widetilde{A}, \widetilde{A}'), \pi^{\mathsf{EgEqSk}})$ and $\mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgRp}}}((E, \widetilde{A}', b), (a, r))$.

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

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

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 4.6 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 the ideal functionality RanElm that returns $H \overset{\mathrm{R}}{\leftarrow} \mathcal{G}$.

(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. In addition to the original protocol, U_s

(a) Provides a proof that X_d is fresh, i.e., using $\mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EgFsh}}}$ with parameter $b \leftarrow 2^c$.

(b) Refreshes its active balance: it sends $\overline{A}'_s \leftarrow \overline{A}_s - \overline{X}_s$ to C and a proof that $\overline{A}'_s + \overline{X}_s = \overline{A}_s$, using $\mathsf{P}^{\mathsf{ZK}}_{\mathcal{R}_{\mathsf{EqSk}}}$.

[**Iftach's Note: So currently the U_s sends 3 encryptions, $\overline{X}_s, \overline{X}_d, \overline{A}'_s$, and 4 proofs. Can we do better?**]

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.

4.6 Threshold ElGammal

In this section, we present a threshold variant of the ElGammal encryption scheme. It will be used for the auditing capability of the confidential translation protocol. In the following, fix the number of public keys to $n < q$ and a threshold $t \in [m]$.

4.6.1 Additional Preliminaries

Notation 4.7 (Lagrange basis functions)**.** *For $\mathcal{S}$, $z \in \mathcal{S}$ and $x \in \mathbb{Z}_q$, let* $\lambda^{\mathcal{S}}_y(x) := \prod_{z \in \mathcal{S} \setminus \{y\}} \frac{x-z}{y-z}$.

Fact 4.8. *Let p be a degree $t-1$ polynomial over $\mathbb{Z}_q$ and let $\mathcal{S} \subseteq \mathbb{Z}_q$ be of size t. Then for any $x \in \mathbb{Z}_q$: $p(x) = \sum_{y \in \mathcal{S}} \lambda^{\mathcal{S}}_y(x) \cdot p(y)$.*

4.6.2 The Scheme

The threshold scheme $(\mathsf{TshGen}, \mathsf{TshEnc}, \mathsf{TshDec})$ is defined as follows:

Algorithm 4.9 ($(\mathsf{TshGen}, \mathsf{TshEnc}, \mathsf{TshDec})$: The ElGamal threshold Encryption Scheme)**.**

Paramters: $1 \leq t \leq n < q$.

Key generation (standalone): $\mathsf{TshGen}(1^b, H)$*:*

1. *Sample uniformly a degree* $t-1$ *polynomial* p *over* Z_q.
2. *For each* $i \in (n+1)$*: Let* $e_i \leftarrow p(i)$.
3. *Output public key* $(1^b, H, E \leftarrow e_0^{-1} \cdot G)$*, verification keys* $\{E_i \leftarrow e_i \cdot G\}_{i \in [n]}$*, and secret keys* $\{e_i\}_{i \in [n]}$.

Encryption: TshEnc*: Same like* EgEnc.

Decryption: Protocol $\mathsf{TshDec}(\widetilde{A})$ *between parties* $\{\mathsf{P}_i\}_{i \in \mathcal{S}}$ *for a* t*-size* $\mathcal{S} \subseteq [n]$*, party* P_i *holding secret keys* e_i*, and combiner* C.

1. *Party* P_i*, for all* $i \in \mathcal{S}$*: Send* $A_i \leftarrow e_i \cdot \widetilde{A}_L$ *to* C.[a]
2. C*:*
   (a) $M \leftarrow \widetilde{A}_R - \sum_{i \in [t]} \lambda_i^{\mathcal{S}}(0) \cdot A_i$.
   (b) *Find (using brute force)* $m \in (b)$ *so that* $m \cdot G = M$*. Abort if no such* m *exists.*[b]
   (c) *Output* m.

[a]Depending on the context, we might ask P_i to prove that A_i is the correct value. This can be done using the verification keys and $\Pi^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{EgCons}}}$.

[b]This part is identical to EgDec.

Note that the public key and the encryption algorithm of the above scheme are identical to that Algorithm 4.1, the twisted ElGamal in-the-exponent encryption scheme. So almost all the machinery developed on top of Algorithm 4.1 is applicable to the above scheme without any change, i.e., the zero-knowledge protocols and chunk EG scheme. The only exception are the protocols that require knowledge of the secrete key, but those are irrelevant for are use case of threshold encryption.

4.6.3 Distributed Key Generation

In this part we present a variant of the standard distributed key-generation protocol for the key generation algorithm TshGen described above. We make use of the following two-party multiplication functionality.

Functionality 4.10 (Mult: two-party multiplication)**.**

Parties: $\mathsf{P}_0, \mathsf{P}_1$.

P_i's private input: $a_i \in \mathbb{Z}_q$. A corrupt P_0 can provide $o_0 \in \mathbb{Z}_q$.

Operation:

1. If o_o is not set, sample $o_o \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$.

2. Send o_0 to P_0 and $o_1 \leftarrow a_0 \cdot a_1 - o_0$ to P_1.

Functionality Mult is just the well-known OLE functionality, see [**HaitnerMRT22**] for an OT-base implementation.[4]

Protocol 4.11 (Distributed key generation)**.**

Paramters: $1 \leq t \leq n < q$.

Oracles: RanElm, Mult.

Parties: $\mathsf{P}_1, \ldots, \mathsf{P}_n$.

Oracles: Mult.

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

Common input: $1^b, H$.

Operation: Party P_i acts as follows:

1. Call RanElm, Let D be the common output. // Sample a common ephemeral: Pedersen public key.[a]
2. // Sample random polynomial and commit to its coefficients.
   (a) Sample a uniform degree $t-1$ polynomial p_i.
   Let $\{c_{i,j}\}_{j \in (t-1)}$ be the coefficients of p_i.
   (b) For each $j \in (t-1)$: Sample $r^C_{i,j} \xleftarrow{\text{R}} \mathbb{Z}_q$.
   (c) Send $\{C_{i,j} \leftarrow \mathsf{Ped}_D(c_{i,j}; r^C_{i,j})\}_{j \in (t-1)}$ to all parties.

- For $k, \ell \in [n]$, let $B_k(\ell) := \sum_{j \in (t-1)} C_{k,j} \cdot \ell^j$ and let $B(\ell) := \sum_{k \in [n]} B_k(\ell)$.

3. // Send opening of $B_i(\ell)$ to P_ℓ.
   (a) For all $\ell \in [n]$: Set $r_{i,\ell} \leftarrow \sum_{j \in (t-1)} r^C_{i,j} \cdot \ell^j$.
   (b) For all $\ell \in [n] \backslash \{i\}$: Send $(p_i(\ell), r_{i,\ell})$ to P_ℓ
4. // Generates the secret and verification keys.
   (a) For all $\ell \in [n] \backslash \{i\}$: Verify $B_i(\ell) = \mathsf{Ped}_D(p_\ell(i); r_{\ell,i})$.
   (b) Let $e_i \leftarrow \sum_{\ell \in [n]} p_\ell(i)$, $E_i \leftarrow e_i \cdot G$, and $r_i \leftarrow \sum_{\ell \in [n]} r_{\ell,i}$.
   (c) Let $\pi_i \leftarrow \mathsf{P}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{PedGrEq}}}((H, B(i), E_i), (e_i, r_i))$.
   (d) Send (E_i, π_i) to all parties.
5. For all $\ell \in [n] \backslash \{i\}$: Call $\mathsf{V}^{\mathsf{ZK\text{-}POK}}_{\mathcal{R}_{\mathsf{PedGrEq}}}((H, B(\ell), E_\ell), \pi_\ell)$.

[4]If we are fine with using non-twisted threshold ElGamal, then the Mult functionality is not needed, and the whole protocol gets simpler.

6. // Generates additive shares of public key $E = e^{-1} \cdot H$.

   [**Iftach's Note: The following sub-protocol allows a corrupt party to induce wrong E. This should be possible to handled by verifying that the decryption works fine**]

   (a) Let $a_i \leftarrow p_i(0)$ and $b_i \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$.

   (b) For each $\ell \in [n] \backslash \{i\}$, call Mult jointly with P_ℓ, twice. If $\ell < i$, use a_i in the first call and b_i in the second; otherwise, use b_i in the first call and a_i in the second. Let $c_{i,\ell}$ and $c'_{i,\ell}$ be the private outputs of these calls.

   (c) Output $c_i = a_i b_i + \sum_{\ell \in [n] \backslash \{i\}} c_{i,\ell} + c'_{i,\ell}$.

   (d) Let $c \leftarrow \sum_{\ell \in c_i}$.

   (e) Send $E'_i \leftarrow b_i \cdot c^{-1} \cdot H$ to all parties.

7. // Construct and verify public key.

   (a) Set $E \leftarrow \sum_{\ell \in [n]} E_\ell$.

8. Output $(1^b, H, E)$ as the public key, $\{E_\ell\}_{\ell \in [n]}$ as the verification keys, and e_i as the secret key.

[a]In practice, we can use H.

Theorem 4.12 (Security of key-generation protocol)**.** *Assuming* DDH *is hard over* $\mathcal{G}$*, then Protocol 4.11 UC-realizes* TshGen *in the* $\{$RanElm,Mult$\}$*-hybrid model.*

Proof. [**Iftach's Note: TODO**] □

4.6.4 The Chunk Variant

Since the encryption algorithm Algorithm 4.9 is exactly like that of the non-threshold case, its chunk variant is identical to the non-threshold case. The only difference are the zero-knowledge protocols that requires the use of the secrete key, which in the threshold case is shared. Fortunately, it seems that we do not need this protocols in out application.