A Cryptographic Analysis of the UTT system

Abstract

We analyze the UTT system from [1] in the framework of secure multiparty computation (MPC). We present an ideal functionality $\mathcal{F}_{\mathsf{utt}}$ that captures the task of decentralized anonymous payment, and show that the UTT system securely realizes this functionality against an active adversary that may corrupt an arbitrary number of clients that are chosen at the beginning of the execution. We limit our analysis to a fully synchronous setting. Along the way, we present a new model of stand-alone MPC for reactive (aka stateful) functionalities which captures the ability of the adversary to adaptively select the inputs of the parties, without resorting to the full-fledged UC model of [9]. We reduce the security of the protocol to the security of the underlying cryptographic building blocks. Our analysis holds in the standard model. Though the protocol inherits some ideal model assumptions (RO and AGM) from the concrete instantiations of some of the underlying cryptographic building blocks.

Organization. In Section 1 we present the security model and the ideal $\mathcal{F}_{\mathsf{utt}}$ functionality. Section 2 presents the cryptographic ingredients needed for our protocol and their instantiations. (Some details are deferred to the Appendices). While most of the material is standard, the reader is advised to read about our somewhat non-standard notion of committed signature schemes (Section 2.2). Our protocol appears in Section 3 and some extensions are presented in Section 4. In Sections 5 and 6 we describe the simulator and analyze it.

1 The Ideal Functionality and the MPC model

In this Section, we describe the ideal $\mathcal{F}_{\mathsf{utt}}$ functionality and describe some aspects of the MPC model and our security proofs.

1.1 The ideal functionality

Notation The functionality interacts with n banks, a minter M, a set of N clients C and an adversary Adv. Each client is identified by a unique public identifier $\mathsf{pid} \in \{0,1\}^*$ and we let $\mathsf{C}_{\mathsf{pid}}$ denote the client whose identifier is pid. The functionality is parameterized with a set of legal coin values V which, by default, is taken to be integers in the range $[0, \mathsf{V}_{\mathsf{max}}]$ where $\mathsf{V}_{\mathsf{max}}$ is some positive integer. The functionality initializes the counter $\mathsf{T} = 0$ that keeps track of the number of coins that were generated so far. That is, whenever a coin is generated (either via a successful Mint operation or a successful Pay operation) the counter is increased, and we use this value as the (ideal) identifier of the generated coin. The functionality also maintains a dictionary coins that maps a coin id t to the coin's value and owner. Specifically, $\mathsf{coins} : \mathbb{N} \to (V \times \{0,1\}^*) \cup \{\perp\}$. We denote by $\mathsf{val}(t) \in V$ and $\mathsf{owner}(t) \in \{0,1\}^*$ the value and owner associated with a coin id t and view these values as undefined if $\mathsf{coins}(t) = \perp$. Our payment mechanism supports 2-to-2 coin transaction, where the two "input" coins are both owned by Sender and the one "output" coin is delivered to a client pid_1 and the second output coin is delivered to a (possibly different) client pid_2.

Our specification of the $\mathcal{F}_{\mathsf{utt}}$ functionality mainly strives for simplicity. Naturally, one can consider several other variants of the functionality that may suit concrete applications. Below we elaborate on our choices and list some natural variants that can be easily supported by applying minor modifications to our protocol.

FUNCTIONALITY 1.1. (Functionality $\mathcal{F}_{\mathsf{utt}}$)

The functionality supports two types of operations Mint and Pay.

- **Mint.** Upon receiving the message $(\texttt{mint}, v, \mathsf{pid})$ from the minter, where $v \in V$ is the desired value and pid is the public identifier of the client to which the coin is destined, set $\mathsf{T} \leftarrow \mathsf{T}+1$ and assign $\mathsf{coins}[\mathsf{T}] = (v, \mathsf{pid})$. Send $(\texttt{minted}, v, \mathsf{pid}, \mathsf{T})$ to the client $\mathsf{C}_{\mathsf{pid}}$ and send $(\texttt{minted}, \mathsf{T})$ to everyone else.
- **Pay.** Upon receiving the message $(\texttt{pay}, t_1, t_2, \mathsf{pid}_1, v_1, \mathsf{pid}_2, v_2)$ from a client Sender identified by pid, initialize all error flags to false, and verify that the following conditions hold:
  - (legal incoming coins) Sender owns input coins. Namely, for $i \in \{1,2\}$, if $\mathsf{owner}(t_i) \neq \mathsf{pid}$ set the error flag $\mathsf{err\text{-}in}_i$ to true.
  - (legal values) The values of the output coins are legal. Namely, for $j \in \{1,2\}$, if $v_j \notin V$ set the error flag $\mathsf{err\text{-}val}_j$ to true.
  - (legal sum) The sum of the input values and the output values is equal. Namely, if $\mathsf{val}(t_1) + \mathsf{val}(t_2) \neq v_1 + v_2$ set the error flag err-sum to true. (By convention, if $\mathsf{val}(t_1)$ or $\mathsf{val}(t_2)$ are undefined, we set err-sum to true.)

  (Report error if needed:) If at least one of the error flags is on, broadcast to all parties the error flag $\mathsf{err} = \mathsf{true}$ together with the current value of T and terminate the current operation.
  Otherwise (verification succeeds), "burn" the incoming coins by setting $\mathsf{coins}[t_1]$ and $\mathsf{coins}[t_2]$ to $\perp$, and do the following for each $j \in \{1,2\}$:
  - Increase $\mathsf{T} \leftarrow \mathsf{T}+1$, set $\mathsf{owner}(\mathsf{T}) = \mathsf{pid}_j$ and $\mathsf{val}(\mathsf{T}) = v_j$. If there exists a client whose public identifier is pid_j, send her the message $(\texttt{paid}, \mathsf{T}, v_j)$. Send $(\texttt{paid}, \mathsf{T})$ to all the other parties.

1. (The Minter) The use of a minter models the fact that money is injected to the system via some external process (e.g., central Bank.) Jumping ahead, the Minter will always be assumed to be honest, though, as usual in MPC, the adversary has the power to select the inputs of the Minter and this way to inject money to the system.

2. (Error handling) We release a single error flag if either the incoming coins are not owned by the sender or the outgoing coins have illegal values or if the sum of the values of the incoming coins do not match the sum of the values of the outgoing coins.[1] This single error flag is being sent to all the parties for the sake of transparency. But could be sent, in principle, only to the Banks and the Sender. Also, one could send a more detailed error report by sending the vector of error flags $(\mathsf{err\text{-}in}_1, \mathsf{err\text{-}in}_2, \mathsf{err\text{-}val}_1, \mathsf{err\text{-}val}_2, \mathsf{err\text{-}sum})$ either to all the parties or only to the Banks. One may further decide to "burn" the ith incoming coin if it is legal (i.e., $\mathsf{err\text{-}in}_i = \mathsf{false}$) regardless of the validity of the whole transaction.

3. (Other coin identifiers) The use of the counter T as a coin identifier is somewhat arbitrary and can be replaced by any other reference mechanism. (e.g., random identifiers). It should be noted that the counter T counts the number of "generated coins" as opposed to the number of "valid" coins. (Indeed, we count coins that are designated for "invalid receivers" and do not decrease the counter when a coin is transferred to a new use). Consequently, the current value of T can be always inferred based on the history of successful transactions. Still, we find it convenient to deliver the current value of T as part of the output.

4. (Static vs dynamic set of parties) We assume that the functionality is implicitly parameterized by a fixed set of clients. One can consider a dynamic version in which parties are being added on the fly via a special registration command.

[1] Note that We do allow a payer to pay her money to a non-existing payee without raising an error flag, which is analogous to the act of "throwing away" money.

5. (k-to-ℓ transactions) For simplicity, we support 2-to-2 transactions which are essentially universal. One can naturally extend the functionality (and the protocol) to deal with a more rich family of k-to-ℓ transactions.

Remark 1.2 (The generalized $\mathcal{F}_{\mathsf{utt}}$ functionality)**.** In some cases it may be useful to assume that coins carry additional "type" information that can be taken to be a vector of attributes. One can then define appropriate rules that determine whether a set of incoming coins is allowed to be converted into a set of outgoing rules as a function of the coins owners, values, and types. The $\mathcal{F}_{\mathsf{utt}}$ functionality can be naturally extended to support this more general notion by changing the error checks accordingly. Indeed, the anonymity budget mechanism can be captured under this abstraction. In Section 4.2 we briefly explain how to adopt the protocol to this more general setting.

1.2 The MPC Model

Modeling adaptive inputs. Following the standard REAL/IDEAL paradigm we would like to say that a protocol Π securely realizes an ideal functionality $\mathcal{F}$ if any efficient adversary Adv that attacks the protocol Π can be translated into an efficient adversary $\mathcal{S}$ (simulator) that attacks the ideal implementation in which parties have an access to the ideal functionality. However, since our ideal functionality is *reactive* (i.e., it maintains a state), a special care is needed in order to define security. Specifically, an important aspect that should be captured is the ability of the adversary to adaptively inject "inputs" to the system based on the view that was gathered so far. Here "inputs" refers to the actions of the corrupted parties and to the inputs of the *honest* parties.[2] In a non-reactive setting, this concern is easily taken care of by quantifying security over all possible inputs. For reactive functionalities, such a universal quantification fails to capture the adaptive power of the adversary.[3] Following the UC model of [9], we capture such an adaptive choice of inputs via the use of an external environment Env, and present a limited version of the UC definition that, in our opinion, provides a sound model for "standalone MPC security" for reactive functionalities. In a nutshell, we assume a synchronous setting (like [19, 18, 17]), and assume that the protocol is invoked in "phases", in the beginning of each phase, the adversary (Adv or $\mathcal{S}$) and the honest parties receive inputs from the environments Env, participate in the protocol, and at the end of the phase send their outputs to the environment. We emphasize that, while our definition is inspired by UC, composability is not our central concern, rather our main goal is to capture the adaptive choice of inputs. Indeed, since the adversary does not communicate with the environment *during a phase*, we can, for example, rewind it to the beginning of the phase (though we do not exploit such rewinding in our proof). We proceed with a formal definition.

Definition 1 (stand-alone MPC for reactive functionalities)**.** *For an environment* Env*, real adversary* Adv*, ideal adversary* $\mathcal{S}$ *(aka the simulator), and a collection of subsets of parties* $\mathcal{M}$ *(aka adversary structure), we define the ideal execution of the functionality* $\mathcal{F}$ *and the real execution of the protocol* Π *as follows.*

At the beginning, the environment Env*, that is given the security parameter,* 1^λ*, and an auxiliary input* z*, chooses which subset* $M \in \mathcal{M}$ *of the parties to corrupt. The game now proceeds in "phases". In each phase, based on the information gathered so far,* Env *sends inputs to the honest parties and to the adversary who controls the parties in* M*.*[4] *The corresponding parties then execute the current phase, either by running the protocol* Π *with their inputs, or by making a single call to the ideal functionality* $\mathcal{F}$ *which delivers outputs*

[2] Indeed, it is now widely accepted both in practice and theory, that the possibility of adversarial influence on the inputs of honest parties is a real concern and typical cryptographic definitions (e.g., CPA or CCA security) are tailored to cope with such scenarios.

[3] To illustrate this point, consider a (contrived) system that leaks to the adversary, after the first call, a sequence of N random operations, $R_1, \ldots, R_N$. The system operates securely, but if the next N calls follow the pattern R (i.e., the ith client makes the ith call with input R_i), the system completely breaks down (e.g., reveals all secrets and deliver the "money" to the adversary). Since the probability of failure is tiny for any fixed predetermined sequence of inputs, such a protocol is secure with respect to a static choice of inputs. Of course, security is violated when the inputs are chosen adaptively. While this example is somewhat contrived, we note that properly dealing with such scenarios leads to complications both in the proofs and in the definitions.

[4] This implicitly means that when the functionality receives an input from a single party at a time (like in our case) Env chooses which party speaks in the current phase.

and updates its state. Of course, when the adversary is active (as is the case in our setting) she is allowed to arbitrarily deviate from the protocol's instructions and to submit arbitrary values to the ideal functionality. At the end of the phase, the honest parties deliver their outputs to Env*, and the adversary delivers to* Env *its output, which, wlog, contains its entire view. At the end of the execution the environment terminates with an output that, wlog, can be taken to be a single bit attempting to distinguish whether the real execution takes place or the ideal execution. We denote by* $\mathsf{Exec}_{\Pi,\mathsf{Adv},\mathsf{Env}}(1^\lambda, z)$ *the random variable that describes the output of* Env *in the real execution, and by* $\mathsf{Exec}_{\mathcal{F},\mathcal{S},\mathsf{Env}}(1^\lambda, z)$ *the random variable that describes the output of* Env *in the ideal execution.*

We say that a protocol Π *securely realizes the functionality* $\mathcal{F}$ *with respect to the collection* $\mathcal{M}$*, if for every polynomial-time adversary* Adv*, there exists a polynomial-time simulator* $\mathcal{S}$*, such that for every computationally-bounded environment* Env *the ensemble*

$$\left\{\mathsf{Exec}_{\Pi,\mathsf{Adv},\mathsf{Env}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \tag{1}$$

is indistinguishable from the ensemble of random variables

$$\left\{\mathsf{Exec}_{\mathcal{F}_{\mathrm{utt}},\mathcal{S},\mathsf{Env}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}. \tag{2}$$

The use of idealized oracles. Our standalone model can be easily extended to work with ideal oracles such as Random Oracles (RO) [3] or the algebraic group model (AGM) [12], that are available only to the adversary but not to the environment. Indeed, some of our building blocks are proved to be secure in the RO or AGM model, and so these models are carried to our protocol. We emphasize that the analysis of the protocol does not make a direct use of these models. Accordingly, the protocol's "standard-model" security follows from a "standard-model" of the underlying building blocks.

The adversarial model. We consider an active adversary that corrupts any number of clients that are selected non-adaptively at the beginning of the execution. The main protocol is described with respect to a *single Bank* that is assumed to be honest, though the adversary can listen to all the incoming/outgoing communication from the Bank. In Section 4.1, we explain how to extend the protocol and its analysis to a threshold setting, in which there are multiple Banks, and the adversary can actively corrupt up to 1/3 of them. (For more details about the adversarial model and the network setting, see 3.)

2 Cryptographic Primitives

Global Setup. We present here the definition of the cryptographic primitives used by our construction: commitment, committed signature and anonymous encryption schemes. All these algorithms will make use of some global public parameters pp (e.g., a description of a bilinear group, a CRS, etc.) that are generated by a PPT algorithm Setup that takes as input the security parameter 1^λ (and some randomness). The public parameters pp will be given as inputs to all cryptographic algorithms and to the adversary. We further assume (WLOG) that pp implicitly contains the security parameter 1^λ and therefore there is no need to explicitly send the security parameter to the following cryptographic algorithms. We further assume that the message space, randomness space and output space, of all the cryptographic algorithms are determined by the public parameters.[5] In later sections, we will typically omit the dependency in pp from the cryptographic algorithms for ease of notation.

Instantiation. Our setup algorithm samples the description $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ of a bilinear map of type III [15], together with a random generator $h \xleftarrow{\$} \mathbb{G}_1$ (to be used later as a public bases for the Dodis-Yampolskiy PRF [10], see Section 2.4).

[5] This convention is taken mainly for the sake of simplicity and since all our instantiations satisfy it.

2.1 Homomorphic Commitment Scheme

We will consider non-interactive homomorphic commitments in which the hiding property holds information theoretically.

Definition 2 (Homomorphic Commitments)**.** *A commitment scheme consists of the following algorithms:*

1. $\mathsf{CM.Setup}(\mathsf{pp}) \to \mathsf{ck}$. *Given public parameters* pp *and random coins the algorithm outputs a commitment key* ck.
2. $\mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m; r) \to \mathsf{cm}$. *Given public parameters* pp*, a commitment key* ck*, a message* $m \in \mathcal{M}$ *and randomness* $r \xleftarrow{\$} \mathcal{R}$*, outputs the commitment* $\mathsf{cm} \in \mathcal{C}$*. Recall that* pp *determines the message space* $\mathcal{M}$*, the randomness space* $\mathcal{R}$ *and the commitment space* $\mathcal{C}$*.*

The algorithms should satisfy the following properties:

- **Perfect hiding** *For every possible* pp, ck *and every pair of messages* $m_0, m_1 \in \mathcal{M}$*, it holds that the pair* $(\mathsf{pp}, \mathsf{ck}, \mathsf{cm}_0 = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m_0; r))$ *and* $(\mathsf{pp}, \mathsf{ck}, \mathsf{cm}_1 = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m_1; r))$ *are identically distributed where* r *is uniformly sampled from* $\mathcal{R}$*.*
- **Strong binding** *For every efficient adversary* Adv*, the probability, over random choice of the public parameters* $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *and the commitment key* $\mathsf{ck} \xleftarrow{\$} \mathsf{CM.Setup}(\mathsf{pp})$*, that* $\mathsf{Adv}(\mathsf{pp}, \mathsf{ck})$ *outputs a tuple* (m_0, r_0, m_1, r_1) *for which*

$$\mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m_0; r_0) = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m_1; r_1) \quad \textit{and} \quad (m_0, r_0) \neq (m_1, r_1)$$

  is negligible in λ*.*

We say the scheme is homomorphic *if the message space* $\mathcal{M}$ *and the randomness space* $\mathcal{R}$ *are abelian groups and there is a special operation that given* pp, ck *and* $\mathsf{cm}_1, \mathsf{cm}_2 \in \mathcal{C}$ *outputs a commitment* cm *with the following guarantee: If there exists* m_0, r_0, m_1, r_1 *for which*

$$\mathsf{cm}_0 = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m_0; r_0) \quad \textit{and} \quad \mathsf{cm}_1 = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m_1; r_1)$$

then

$$\mathsf{cm} = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m_0 + m_1; r_0 + r_1)$$

where "+" stands for the group operation. By abuse of notation, we sometimes omit the dependency in pp, cm *and denote the homomorphic combination by* $\mathsf{cm}_0 \boxplus \mathsf{cm}_1$*.*

Remark 2.1 (Opening a commitment)**.** We adopt the convention that in order to "open" a commitment cm with respect to public parameters pp and a commitment key ck, the committer sends the message m and the randomness r and the verifier checks that $\mathsf{CM.Commit}(\mathsf{ck}, m; r) = \mathsf{cm}$.

Remark 2.2 (Rearandomizing a commitment)**.** A homomorphic commitment can be perfectly rerandomized as follows. Given public parameters pp, a commitment key ck, a commitment cm and a random shift $r_\Delta \in \mathcal{R}$, define the procedure $\mathsf{CM.Rerand}(\mathsf{pp}, \mathsf{ck}, \mathsf{cm}; r_\Delta) \to \mathsf{cm}'$ by setting

$$\mathsf{cm}' = \mathsf{cm} \boxplus \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, 0; r_\Delta).$$

Observe that if $\mathsf{cm} = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m; r_0)$ for some message m and randomizer r_0, then $\mathsf{cm}' = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m; r_0 + r_\Delta)$. Since we will be employing only homomorphic commitments, we will always assume the availability of such $\mathsf{CM.Rerand}$ procedure.

Remark 2.3 (Vector commitments)**.** Our message space will always be of the from $\mathbb{Z}_p^4$. That is, our messages are quadruplets where each entry represents a different data item. The homomorphic operation is the standard component-wise addition over $\mathbb{Z}_p$, and therefore one can add a value to a single entry without changing the other entries (by adding zeroes in all other locations).

Instantiation. We use a dual "bilinear" version of the well-known Pedersen commitment [20]. The binding property holds under the SDL assumption of [5] which follows from 1-SDH of [6]. (See Section A for details.) As already mentioned, the hiding property hold information theoretically.

2.2 Committed Signature Scheme

Our protocol makes extensive use of signatures over commitments schemes. The signatures should be re-randomizable and, in addition, one should be able to re-randomize a commitment while maintaining the validity of the signature. This means that the signature should be somewhat non-malleable – a property that inherently contradicts the unforgeability property. The following definition formalizes the desired notion of unforgeability for such signatures. Roughly speaking, we count a forgery as valid if the forger can generate a signed commitment together with a corresponding opening so that the corresponding message did not appear before as a query.

Definition 3 (Rerandomizable Signatures over Commitments)**.** *A rerandomizable signature-commitment scheme is defined with respect to a given Commitment scheme* $(\mathsf{CM.Setup}, \mathsf{CM.Commit})$ *that is equipped with a rerandomization procedure* $\mathsf{CM.Rerand}$ *(see Definition 2 and Remark 2.2) and consists of the following additional algorithms:*

1. $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \to (\mathsf{sk}, \mathsf{pk} = (\mathsf{pp}, \mathsf{vk}, \mathsf{ck}))$. *Given public parameters* pp*, a commitment key* ck*, and randomness, the key-generation algorithm computes a private signing key,* sk*, a public verification key* vk *and outputs* $(\mathsf{sk}, \mathsf{pk} = (\mathsf{pp}, \mathsf{vk}, \mathsf{ck}))$.

2. $\mathsf{RS.Sign}(\mathsf{sk}, \mathsf{cm}; u) \to \sigma$. *Given a signing key,* sk*, a commitment* cm *taken from the commitment space* $\mathcal{C}$*, and random coins* u *sampled uniformly from the randomness space of the signature scheme, output the signature* σ.

3. $\mathsf{RS.Rerand}(\mathsf{pk}, \sigma; r_\Delta, u_\Delta) \to \sigma'$. *Given a public key* $\mathsf{pk} = (\mathsf{ck}, \mathsf{vk})$ *a signature* σ*, and randomizers* r_Δ, u_Δ *the algorithm outputs a new signature* σ' *such that if* $\sigma = \mathsf{RS.Sign}(\mathsf{sk}, \mathsf{CM.Commit}(\mathsf{ck}, m; r); u)$ *for some message* m *and randomizer* r *then* $\sigma' = \mathsf{RS.Sign}(\mathsf{sk}, \mathsf{CM.Commit}(\mathsf{ck}, m; r + r_\Delta); u + u_\Delta)$ *where we assume that the randomness space of the signature and the randomness space of the commitment can be viewed as groups with efficiently computable group operation "+".*

4. $\mathsf{RS.Ver}(\mathsf{pk}, \mathsf{cm}, \sigma) \to \{0, 1\}$. *The algorithm is given the verification key,* vk*, the commitment* $\mathsf{cm} \in \mathcal{C}$ *and the signature* σ *and outputs 1 if verification is successful (and 0 otherwise). The algorithm should satisfy the* perfect correctness *property, i.e., for every keys* $(\mathsf{sk}, \mathsf{pk})$*, every commitment* $\mathsf{cm} \in \mathcal{C}$ *and every randomness* u *it holds that* $\mathsf{RS.Ver}(\mathsf{pk}, \mathsf{cm}, \mathsf{RS.Sign}(\mathsf{sk}, \mathsf{cm}; u)) = 1$.

The scheme should satisfy **Existential unforgeability under chosen commitment attack**. *That is, every efficient adversary* Adv *cannot win in the following game,* $\mathsf{GAME}_{\mathsf{EU-CCA},\mathsf{Adv}}$*, with more than negligible probability in* λ*:*

1. *The Challenger samples* $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*,* $\mathsf{CM.Setup}(\mathsf{pp}) \to \mathsf{ck}$ *and* $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \to (\mathsf{sk}, \mathsf{pk})$ *and sends* pk *to the adversary* Adv.

2. *The adversary has a signing oracle that given a message* m *and commitment randomness* r*, samples a fresh* u *and returns a signature* $\sigma = \mathsf{RS.Sign}(\mathsf{sk}, \mathsf{cm}; u)$ *where* $\mathsf{cm} = \mathsf{CM.Commit}(\mathsf{ck}, m; r)$*. Let* M *denote the set of all messages that were sent to the oracle.*

3. *The adversary outputs a tuple* (m^*, r^*, σ^*) *and wins if* $\mathsf{RS.Ver}(\mathsf{pk}, \mathsf{CM.Commit}(\mathsf{ck}, m^*; r^*), \sigma^*) = 1$ *and* $m^* \notin M$.

Instantiation. We use a variant of the Pointcheval-Sanders signature scheme [21] over dual-Pedersen commitments. In Section B we describe the construction (Figure 2) and prove the following lemma.

Lemma 2.4. Under Assumption 1 of [21], the scheme presented in Figure 2 is a rerandomizable signature-commitment scheme.

2.3 Anonymous Identity-Based Encryption Scheme

We need an identity-based encryption (IBE) scheme that satisfies the standard notion of ciphertext-indistinguishability. In addition, we will need a less standard *key indistinguishability* (KI) property [2] that asserts that it is hard to relate a ciphertext to the corresponding public key. Both properties should hold under Chosen-Ciphertext Attacks (CCA). This combined notion of security is nicely captured via the notion of IND-RA-CCA security [22] defined below.

Definition 4 (IND-RA-CCA IBE)**.** *An IBE scheme consists of the following algorithms:*

1. $(\mathsf{msk}, \mathsf{mpk}) \xleftarrow{\$} \mathsf{IBE.Setup}(\mathsf{pp})$. *Given public parameters* pp*, and randomness, the IBE-setup algorithm,* $\mathsf{IBE.Setup}$*, computes a master secret key,* msk*, and a master public key* mpk.
2. $\mathsf{sk}_{\mathsf{id}} = \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$. *Given an identifier* id *and a master secret key* msk *the deterministic extraction algorithm,* $\mathsf{IBE.Extract}$*, computes a secret key* $\mathsf{sk}_{\mathsf{id}}$ *that is asscosiated with the identifier* sk.
3. $c \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{id}, m)$. *Given an identifier* id *and a message* m*, the randomized encryption algorithm,* $\mathsf{IBE.Enc}$*, outputs a ciphertext* c.
4. $m = \mathsf{IBE.Dec}(\mathsf{sk}_{\mathsf{id}}, c)$. *Given a secret-key* $\mathsf{sk}_{\mathsf{id}}$ *and a ciphertext* c*, the decryption algorithm,* $\mathsf{IBE.Dec}$*, outputs a plaintext* m.

We require correctness *namely, for every* pp *in the support of* Setup*, every* $(\mathsf{msk}, \mathsf{mpk})$ *in the support of* $\mathsf{IBE.Setup}(\mathsf{pp})$*, every identifier* id *and every message* m *it holds that*

$$\Pr[m = \mathsf{IBE.Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{IBE.Enc}(\mathsf{id}, m))] = 1,$$

where $\mathsf{sk}_{\mathsf{id}} = \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$.[6]

An IBE scheme is IND-RA-CCA secure [22, Section 2.8.3] if every efficient adversary Adv *cannot* win *in the following game with probability better than* $0.5 + \mathsf{negl}(\lambda)$*:*

1. *The challenger samples public parameters* $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ *and a pair of master public/secret keys* $(\mathsf{msk}, \mathsf{mpk}) \xleftarrow{\$} \mathsf{IBE.Setup}(\mathsf{pp})$ *and sends* $(\mathsf{pp}, \mathsf{mpk})$ *to the adversary* Adv.
2. *The adversary is given an access to two oracles: (1) Key-extraction oracle that given an identity* id *returns the corresponding secret key* $\mathsf{sk}_{\mathsf{id}} = \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$*; (2) Decryption oracle that given* (id, c) *returns* $m = \mathsf{IBE.Dec}(\mathsf{sk}_{\mathsf{id}}, c)$ *where* $\mathsf{sk}_{\mathsf{id}} = \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$.
3. *The adversary can send a single challenge query of the form* $(\mathsf{id}_0, m_0), (\mathsf{id}_1, m_1)$*. In response, the challenger tosses a coin* $b \xleftarrow{\$} \{0, 1\}$ *and returns a fresh challenge ciphertext* $c^* \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{id}_b, m_b)$.
4. *At the end, the adversary outputs* b'.

The adversary wins in the game if and only if (1) neither id_0 *nor* id_1 *were sent as queries to the Key-extraction oracle; AND (2) neither* (id_0, c^*) *nor* (id_1, c^*) *were sent as queries to the decryption oracle* after *the challenge phase; AND (3)* $b' = b$.

Remark 2.5. We note that, for our static version of the protocol (where all parties register at the beginning) it suffices to consider a weaker game in which the adversary makes all the key-extraction queries at the beginning of the protocol in a non-adaptive way. That is, the adversary sends a single vector of identities $(\mathsf{id}_1, \ldots, \mathsf{id}_m)$ and gets the answers once and for all. Furthermore, these identities are chosen before seeing the mpk and pp.

[6] One can also consider a variant in which decryption error may occur with negligible probability over the randomness of the encryption, and/or over a random choice of pp and $(\mathsf{msk}, \mathsf{mpk})$. Such a variant also suffices for our purposes.

Instantiation. It is shown in [22, Chapter 4] that if one applies a variant of the Fujisaki-Okamoto transform [14, 13] to the basic (CPA-secure) variant of the Boneh-Franklin IBE [8] the resulting IBE is IND-RA-CCA under the BDH assumption in the Radnom-Oracle model. We use this instantiation in our protocols.

2.4 Pseudorandom Functions

We make use of standard pseudorandom functions. In the following, we say that an infinite sequence of sets $\mathcal{X} = \{\mathcal{X}_{\mathsf{pp}}\}_{\mathsf{pp}\in\{0,1\}^*}$ is efficiently indexed by pp, if given pp one can efficiently sample a uniform element from $\mathcal{X}_{\mathsf{pp}}$ and efficiently decide membership in $\mathcal{X}_{\mathsf{pp}}$.

Definition 5. *A PRF over key space $\mathcal{K} = \{\mathcal{K}_{\mathsf{pp}}\}_{\mathsf{pp}\in\{0,1\}^*}$, input space $\mathcal{X} = \{\mathcal{X}_{\mathsf{pp}}\}_{\mathsf{pp}\in\{0,1\}^*}$ and output space $\mathcal{Y} = \{\mathcal{Y}_{\mathsf{pp}}\}_{\mathsf{pp}\in\{0,1\}^*}$ which are all efficiently indexed by pp, is an efficiently computable function $\mathsf{PRF}(\mathsf{pp}, k, x)$ that maps public parameters pp, a key $k \in \mathcal{K}_{\mathsf{pp}}$, and input $x \in \mathcal{X}_{\mathsf{pp}}$ to an output $y \in \mathcal{Y}_{\mathsf{pp}}$ such that for every efficient adversary Adv, it holds that*

$$\left|\Pr[\mathsf{Adv}^{\mathsf{PRF}(\mathsf{pp},k,\cdot)}(\mathsf{pp}) = 1] - \Pr[\mathsf{Adv}^{\mathcal{F}(\cdot)}(\mathsf{pp}) = 1]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda), k \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$ and $\mathcal{F}$ is a random function from $\mathcal{X}_{\mathsf{pp}}$ to $\mathcal{Y}_{\mathsf{pp}}$.

We typically abuse notation and write $\mathsf{PRF}_k(x)$ to denote $\mathsf{PRF}(\mathsf{pp}, k, x)$.

Regularity properties. For technical reasons we further require that the PRF has no self-collisions nor pairwise collisions. The former property asserts that for every $\mathsf{pp} \in \mathsf{Setup}(1^\lambda), k \in \mathcal{K}_{\mathsf{pp}}$, the function $\mathsf{PRF}(\mathsf{pp}, k, \cdot)$ is injective, and the latter property asserts that for every $\mathsf{pp} \in \mathsf{Setup}(1^\lambda), k \neq k' \in \mathcal{K}_{\mathsf{pp}}$ and input $x \in \mathcal{X}_{\mathsf{pp}}$, it holds that $\mathsf{PRF}(\mathsf{pp}, k, x) \neq \mathsf{PRF}(\mathsf{pp}, k', x)$. Both requirements can be significantly relaxed[7] or even completely removed (at the expense of slightly modifying the protocol). Since our concrete instantiation satisfies these properties, we keep them for simplicity.

Instantiation. We use a well-known PRF by Dodis and Yampolskiy [10] that takes retrieves from the public parameters a public element $h \xleftarrow{\$} \mathbb{G}_1$ and maps a key $k \in \mathbb{Z}_p$ and an input $x \in \mathbb{Z}_p$ to the group element $h^{1/(k+x)} \in \mathbb{G}_1$. Observe that this PRF has no self-collisions nor pairwise collisions.

2.5 Zero-Knowledge Proofs

We employ Zero-Knowledge Proofs of Knowledge (ZKPOK) which are, by default, non-interactive.

Definition 6 (non-interactive ZKPOK (Syntax and Completeness))**.** *For an NP-relation $\mathcal{R}_{\mathsf{pp}}(\mathbb{x}, \mathbb{w})$, a non-interactive ZKPOK proof system consists of the following PPT algorithms which may be oracle aided:*

- $\mathsf{ZK.Setup}(\mathsf{pp}) \to \mathsf{zkpp}$. *Given the global public parameters the randomized algorithm outputs the ZK public parameters zkpp.*
- $\mathsf{ZK.Prove}(\mathsf{pp}, \mathsf{zkpp}, \mathbb{x}, \mathbb{w}; \rho) \to \Pi$. *Given public parameters $\mathsf{pp}, \mathsf{zkpp}$, a statement $\mathbb{x}$ and a valid witness $\mathbb{w}$ that satisfy the relation $\mathcal{R}$, and random coins ρ sampled from the randomness space, the algorithm outputs a proof Π.*
- $\mathsf{ZK.Ver}(\mathsf{pp}, \mathsf{zkpp}, \mathbb{x}, \Pi) \to v$. *Given public parameters $\mathsf{pp}, \mathsf{zkpp}$, a statement $\mathbb{x}$, a proof $\Pi$the (deterministic) verification algorithm outputs a Boolean flag that asserts whether the the proof Π is accepted or rejected.*

[7] E.g., by relaxing the requirement to hold whp over a random choice of (pp, k, k') and by allowing some bounded number T of self collisions and pairwise collisions. (Furthermore, it suffices to assume that finding more than T self/pairwise collisions is computationally intractable.) The protocol and the proof essentially remain the same and we suffer from a loss in the distinguishing advantage that is linear in T.

We require perfect completeness,[8] *namely for every* $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{pp}}$,

$$\Pr_{\mathsf{pp},\mathsf{zkpp},\rho}[\mathsf{ZK.Ver}(\mathsf{pp}, \mathsf{zkpp}, \mathbb{x}, \mathsf{ZK.Prove}(\mathsf{pp}, \mathsf{zkpp}, \mathbb{x}, \mathbb{w}; \rho)) = \mathsf{true}] = 1.$$

If the parties make calls to an oracle, then the probability is taken over the internal randomness of the oracle as well.

Zero-knowledge. We require the existence of an efficient probabilistic (stateful) simulator, $\mathsf{ZK.Sim}$, such that no efficient adversary Adv can win in the following distinguishing game with more than negligible probability in λ:

- Sample $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ and a secret challenge bit $b \xleftarrow{\$} \{0,1\}$. If $b = 0$ let $\mathsf{zkpp} \xleftarrow{\$} \mathsf{ZK.Setup}(\mathsf{pp})$, otherwise let $\mathsf{zkpp} \xleftarrow{\$} \mathsf{ZK.Sim}(\mathsf{pp})$.
  Send $(\mathsf{pp}, \mathsf{zkpp})$ to Adv.
- Adv sends a query of the form $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathsf{pp}}$ and receives back a proof Π where $\Pi \xleftarrow{\$} \mathsf{ZK.Prove}(\mathsf{pp}, \mathsf{zkpp}, \mathbb{x}, \mathbb{w})$ if $b = 0$, and $\Pi \xleftarrow{\$} \mathsf{ZK.Sim}(\mathbb{x})$ if $b = 1$.
- Adv terminates with an output b' and wins if $b' = b$.

If the scheme assumes an access to an ideal oracle $\mathcal{O}$ (e.g., RO) then the adversary is allowed to query the oracle $\mathcal{O}$. If $b = 1$ the same oracle is being used by the prover $\mathsf{ZK.Prove}$, and if $b = 0$ the simulator answers the oracle queries of Adv, i.e., to "program the oracle". (See the discussion below on the actual use of this convention in our proof.) Note that the above definition does not allow the simulator to "rewind" the adversary.[9]

Knowledge extraction. The soundness follows from the following (stronger) notion of *straight-line proof of knowledge* relative to an oracle which is adopted from [4, Def. 3]. Specifically, we require that for every prover Adv there exists an efficient knowledge-extractor $\mathsf{ZK.KE}$ such that the probability that Adv wins in the following game is negligible in λ:

- Sample $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, $\mathsf{zkpp} \xleftarrow{\$} \mathsf{ZK.Setup}(\mathsf{pp})$, and send $(\mathsf{pp}, \mathsf{zkpp})$ to Adv.
- Adv may send queries to the oracle $\mathcal{O}$, and terminates with $(\mathbb{x}, \Pi)$.
- $\mathsf{ZK.KE}$ gets tuple $(\mathsf{pp}, \mathsf{zkpp}, \Pi)$ together with the randomness ρ that was used in order to sample zkpp, the list of queries that were sent by Adv to her oracle $\mathcal{O}$, and an oracle access to $\mathcal{O}$. The extractor $\mathsf{ZK.KE}$ outputs $\mathbb{w}$.
- The adversary wins if the proof Π pass verification but $\mathbb{w}$ is not a valid witness. That is, if

$$\mathsf{ZK.Ver}(\mathsf{pp}, \mathsf{zkpp}, \mathbb{x}, \Pi) = \mathsf{true} \quad \wedge \quad (\mathbb{x}, \mathbb{w}) \notin \mathcal{R}_{\mathsf{pp}}.$$

We say that the knowledge extractor is *universal* if the same extractor works for every adversary Adv. Our instantiations achieve this notion.

[8] Though we can tolerate negligible errors as well.

[9] Recall that our MPC model allow some limited form of rewinding and so the current definition is, in a sense, too restrictive than needed. Furthermore, due to the use of trapdoor perfectly hiding commitments (i.e., Pedersen's commitments), we can use witness-hiding proofs. Regardless, we keep the current formulation for the sake of simplicity.

Instantiation. We employ non-interactive ZKPOK for several natural algebraic relations. Our constructions are obtained by taking an interactive proofs of knowledge that has a straight-line knowledge extractor in the Algebraic Group Model (AGM), and collapsing them to non-interactive zero-knowledge proofs via the Fiat-Shamir transform [11]. The resulting proof systems still have a straight-line knowledge extractor (by keeping track of the AGM queries) and a straight-line simulator (that programs the random oracle).[10]

Let us further mention that our MPC simulator does not call the ZK-simulator, and that the latter algorithm is only employed as part of the analysis. Hence, it seems likely that the analysis can be carried to a "non-programmable" random oracle model. Furthermore, one can trade AGM with appropriate discrete-log related knowledge assumptions. In fact, most of the underlying protocols (with the exception of the range proofs [7]) are Schnorr-like Sigma protocols that satisfy the special soundness property. (See Section C.) It seems likely that, at least for these proofs, one can completely get rid of knowledge assumptions/AGM and rely solely on random oracles.

3 The Protocol

Network model. For the sake of initialization, we assume that each client is connected to the Bank via an authenticated (private or public) channel that is associated with its public identifier pid. (This models the registration procedure in which parties should identify themselves, e.g., via physical means.) After initialization, we assume that each client (and the minter) is connected to the Bank via an untamperable anonymous channel. For the sake of transparency, we further assume that this channel is public and that everyone can listen to it.[11] We assume, for simplicity, a fully synchronous model and that in each round only a single client, that is chosen by the environment on-the-fly, can send her message. This is essentially equivalent to allowing the environments to drop/delay unwanted messages *without looking at their content*. This in particular, means that the adversary's message in a given round can depend only on messages that were sent *before this round*. (In particular, "front running" is prevented.) We mention that the actual protocol from [1] deals with front-running scenarios via the use of additional layer of "signatures-of-knowledge". The Formalization of this additional layer is left for future works.

We model the ledger as a public bulletin board whose content can be (anonymously) accessed by all the parties, but only the Bank has a write access to it. (Again, this is mainly a feature, and the protocol can be adopted to the case where all parties can write on the ledger; just ask the Bank to authenticate its ledger messages by signing them.) The current description assumes a single incorruptible Bank (and an arbitrary number of possibly corrupted clients). We will later explain in Section 4.1 how to extend the protocol to the threshold setting in which there are n Banks, and where the adversary may actively corrupt at most t of them.

In the following subsections, we provide a formal description of the protocol. For a high-level intuitive explanation the reader is referred to the main paper [1]. It may be useful to keep in mind that coins are represented by signed-commitments whose underlying message can parsed as a quadruple $(\mathsf{pid}, \mathsf{sn}, \mathsf{val}, \mathsf{s})$ where pid is the public identifier of the "owner" of the coin, sn is the coin's serial number, val is the value of the coin, and s is a private PRF-key that is associated by the owner. We often use commitments in which only some of these elements are defined, and in this case the "empty fields" are filled with zeroes. (See also Remark 2.3.)

3.1 Initialization

Trusted setup. We assume a trusted setup, which consists of generating the public parameters pp and and placing them in a public repository (i.e., the ledger). Formally, a trusted party calls $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$.

[10] To avoid rewinding, we let the prover sample a random NONCE (aka a "session identifier") in each invocation, and use it as part of the input to the random oracle. Consequently, the RO queries of an honest prover are unpredictable, and the adversary is unlikely to guess them ahead of time.

[11] The ability to use a public channel is a feature and we do not rely on it. In particular, the protocol can be used over a private channel without any modification.

In addition, the trusted set-up samples a pair of master secret-key/public-key $(\mathsf{msk}, \mathsf{mpk})$ for an Identity-Based Anonymous Encryption Scheme, and privately sends to each client who holds a public identifier pid a corresponding secret key $\mathsf{sk}^{\mathsf{ibe}}$. The master public-key, mpk, is being added to the ledger together with pp. If the underlying ZK protocols use some setup parameters, then these strings are also generated and published as part of the setup. (In an actual realization this step is implemented via a one-time MPC protocol over several trusted authorities.)

The Bank. After the trusted setup, the Bank calls sample $\mathsf{ck} \xleftarrow{\$} \mathsf{CM.Setup}(\mathsf{pp})$ where ck is a commitment key for a homomorphic commitment (as per Definition 2) whose message space consists of quadruple of elements in $\mathbb{Z}_p^4$ where p is a large prime that is determined by pp. We denote by $\mathcal{R}$ the randomness space of the commitment scheme. In addition, the Bank calls to $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck})$ twice and generates two pairs of signing/verification keys $(\mathsf{bsk}, \mathsf{bvk})$ and $(\mathsf{rsk}, \mathsf{rvk})$ where the former keys (referred to as "Bank's secret key") will be used for signing standard coins and the latter keys (referred to as "registration keys") will be used only for registration (i.e., for binding together the user's public identifier with its public identifier). The signing keys are being added to the Bank's private state, and the verification keys and the commitment key ck are being added to the ledger. A global counter T is initialized to zero and is also published in ledger.

Clients. Each client C_i with public identifier pid, gets her IBE secret-key $\mathsf{sk}^{\mathsf{ibe}}$ from the trusted set-up. In addition, she samples a random PRF key $\mathsf{s}_0 \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$, and computes a "temporary" registration commitment

$$\mathsf{rcm}_0 = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0); a)$$

where $a \xleftarrow{\$} \mathcal{R}$ is fresh randomizer for the commitment scheme. The client sends to the Bank the values $(\mathsf{pid}, \mathsf{rcm}_0)$ together with a non-interactive zero-knowledge proof of knowledge Π_{Init} for the knowledge of (a, s_0) that satisfies the relation $\mathsf{rcm}_0 = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0); a)$. The Bank then verifies the proof.

- If verification passes, the Bank samples a random shift $\Delta \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$, computes a registration commitment $\mathsf{rcm} = \mathsf{rcm}_0 + \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, 0, \Delta); 0)$ (supposedly $\mathsf{rcm} = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0 + \Delta); a))$, and sends back the tuple
$$(\Delta, \quad \mathsf{rcm}, \quad \mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b))$$
where b is chosen at random. The client sets her PRF-key to $\mathsf{s} = \mathsf{s}_0 + \Delta$, and records her "address secret key" as $\mathsf{ask} = (\mathsf{s}, \mathsf{sk}^{\mathsf{ibe}}, \mathsf{rcm}, \mathsf{rs}, a)$.
- If verification fails, the Bank chooses the corresponding values for the client. Formally, the Bank samples fresh values $\mathsf{s} \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$, $a \xleftarrow{\$} \mathcal{R}$, and sends back the tuple
$$(\mathsf{s}, \quad a, \quad \mathsf{rcm} = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a), \quad \mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b)),$$
where b is chosen at random. The client records her "address secret key" as $\mathsf{ask} = (\mathsf{s}, \mathsf{sk}^{\mathsf{ibe}}, \mathsf{rcm}, \mathsf{rs}, a)$.

3.2 The client's state

Throughout the protocol, each client with public identifier pid maintains a state that consists of ask as defined by the initialization process, and a list $\mathcal{L}$ of tuples. Specifically, for each unspent coin that is owned by the client, the client holds

$$(\mathsf{ccm}_i, \sigma_i, \mathsf{sn}_i, r_i, \mathsf{val}_i, t_i),$$

where

$$\mathsf{ccm}_i = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}_i, \mathsf{val}_i, 0); r) \quad \text{and} \quad \sigma_i = \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm}_i; u_i)) \tag{3}$$

for some (unknown but re-randomized) u_i, and t_i is the "ideal identifier" of the coin, which is the value of the global counter as recorded in the round in which the coin was obtained. We sometimes refer to the tuple $(\mathsf{sn}_i, r_i, \mathsf{val}_i, t_i)$ as the *handles* of the signed coin $(\mathsf{ccm}_i, \sigma_i)$ since one has to know these values in order to be able to spend the coin. In addition, the client has an access to the ledger including the current value of the global counter T.

3.3 The payment sub-protocol

A client Sender who wishes to invoke a "pay" operation of the form $(\mathsf{pay}, t_1, t_2, \mathsf{pid}_B, v_B, \mathsf{pid}_C, v_C)$ does the followings.[12]

1. (Check validity and nullify) For $i \in \{1,2\}$: If the client's list $\mathcal{L}$ contains a tuple whose time identifier is t_i, denote the tuple by $h_i = (\mathsf{ccm}_i, \sigma_i, \mathsf{sn}_i, r_i, \mathsf{val}_i, t_i)$. If this condition fails, set $\mathsf{err\text{-}in}_i = \mathsf{true}$. For $j \in \{B, C\}$, if $v_j \notin [0, \mathsf{V}_{\mathsf{max}}]$ set $\mathsf{err\text{-}val}_j = \mathsf{true}$. If $\mathsf{err\text{-}in}_1 = \mathsf{err\text{-}in}_2 = \mathsf{false}$ verify that $\mathsf{val}_1 + \mathsf{val}_2 = v_B + v_C$, and turn $\mathsf{err\text{-}sum} = \mathsf{true}$ if verification fails. If any of the error flags is on, send the error flag $\mathsf{err} = \mathsf{true}$ to the Bank and terminate. Else, remove from $\mathcal{L}$ the tuples h_1 and h_2, and continue.

2. (Split incoming coins) The client splits each of these two coins into a value-commitment and a nullifier. That is, for $i \in \{1,2\}$, she generates the values

$$\mathsf{vcm}_i = \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, \mathsf{val}_i, 0); z_i) \quad \text{and} \quad \mathsf{nullif}_i = \mathsf{PRF}_{\mathsf{s}_{\mathsf{Sender}}}(\mathsf{sn}_i),$$

   where z_i is a fresh randomizer and $\mathsf{s}_{\mathsf{Sender}}$ is the PRF key of Sender.

3. (Randomize registration signatures) In addition, the client fully re-randomizes the registration signature $(\mathsf{rcm}, \mathsf{rs})$ into $(\mathsf{rcm}', \mathsf{rs}')$. This is done, by letting

$$\mathsf{rcm}' = \mathsf{CM.Rerand}(\mathsf{ck}, \mathsf{rcm}; a_\Delta) \quad \text{and} \quad \mathsf{rs}' = \mathsf{RS.Rerand}(\mathsf{pk}, \sigma; a_\Delta, u_\Delta)$$

   where $\mathsf{pk} = (\mathsf{pp}, \mathsf{rvk}, \mathsf{ck})$ and a_Δ, u_Δ are freshly chosen randomizers. Note that the randomizer of rcm' is simply $a' = a + a_\Delta$.

4. (Prove consistency of splitting) For each $i \in \{1,2\}$, the client generates ZKPOK Π_{Split_i} for the split relation $\mathcal{R}_{\mathsf{split}}$ that contains

$$\mathbb{x} = (\mathsf{ccm}_i, \mathsf{vcm}_i, \mathsf{rcm}', \mathsf{nullif}_i), \quad \mathbb{w} = (\mathsf{s}_{\mathsf{Sender}}, \mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, r_i, z_i, a')$$

   for which the following conditions hold

$$\begin{aligned} \mathsf{ccm}_i &= \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, 0); r_i) \\ \mathsf{vcm}_i &= \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, \mathsf{val}_i, 0); z_i) \\ \mathsf{rcm}' &= \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_{\mathsf{Sender}}, 0, 0, \mathsf{s}_{\mathsf{Sender}}); a') \\ \mathsf{nullif}_i &= \mathsf{PRF}_{\mathsf{s}_{\mathsf{Sender}}}(\mathsf{sn}_i). \end{aligned} \tag{4}$$

5. (Generate coin requests) The client prepares two coin requests for the payees (identified by) pid_B and pid_C with values v_B and v_C, respectively. That is, for $j \in \{B, C\}$, she samples an *identity commitment* to the identity pid_j, and a value commitment to v_j as follows

$$\mathsf{icm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_j, 0, 0, 0); t_j) \quad \text{and} \quad \mathsf{vcm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, v_j, 0); \rho_j), \tag{5}$$

   where t_B, t_C and ρ_B, ρ_C are fresh randomizers.

6. (Prove validity of coin requests) For each coin request $j \in \{B, C\}$, the client generates a ZKPOK of opening for icm_j denoted by $\Pi_{\mathsf{icm}\,j}$, and ZKPOK Π_{Range_j} for the knowledge of (v_j, ρ_j) that satisfies the *range* relation

$$\mathcal{R}_{\mathsf{range}} = \{(\mathbb{x} = \mathsf{vcm}_j; \mathbb{w} = (v_j, \rho_j)) : \quad \mathsf{vcm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, v_j, 0); \rho_j) \wedge v_j \in [0, \mathsf{V}_{\mathsf{max}}]\}. \tag{6}$$

   In addition, the client computes a ZKPOK Π_{Sum} that shows that the sum of values in the incoming coins is equal to the sum of values of the outgoing coins, i.e., for the relation $\mathcal{R}_{\mathsf{sum}}$ that contains all

$$\mathbb{x} = (\mathsf{vcm}_1, \mathsf{vcm}_2, \mathsf{vcm}_B, \mathsf{vcm}_C), \quad \mathbb{w} = (\mathsf{val}_1, z_1, \mathsf{val}_2, z_2, v_B, \rho_B, v_C, \rho_C)$$

[12] By convention, we index the outgoing coins by B and C.

for which the following conditions hold

$$\begin{aligned} \mathsf{vcm}_i &= \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, \mathsf{val}_i, 0); z_i), \quad i \in \{1, 2\} \\ \mathsf{vcm}_j &= \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, v_j, 0); \rho_j), \quad j \in \{B, C\} \\ \mathsf{val}_1 + \mathsf{val}_2 &= v_B + v_C. \end{aligned} \tag{7}$$

The sum can be computed either over the integers or modulo some (public) prime p which is larger than $\mathsf{V}_{\mathsf{max}}$. (In our instantiation, we will take p to be the order of the underlying bilinear group.)

7. (Append coin handles for the payee and send to Bank) The client prepares a ciphertext $\mathsf{ctxt}_j \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}_j, (v_j, \rho_j + t_j))$ for each recipient $j \in \{B, C\}$ and passes to the Bank all the above information, i.e.,

$$((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),\ (\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}},\ \Pi_{\mathsf{Sum}} \tag{8}$$

(Note that the first tuple corresponds to the incoming coins, and the second tuple corresponds to the outgoing coins.)

The Bank proceeds as follows:

1. If the Bank receives from the client an error flag $\mathsf{err} = \mathsf{true}$, the Bank broadcasts it and terminates. Otherwise, the Bank initializes all the error flags to false, and verifies the following conditions:

   - (legal incoming coins + nullification) For $i \in \{1, 2\}$: If (a) σ_i is a valid signatures over the committed coin ccm_i with respect to the Bank's verification key bvk; and (b) the split-proof Π_{Split_i} (together with the relevant tuples) passes verification and (c) rs' is a valid signature on rcm' with respect to the registration key rvk; and (d) nullif_i is not in the in the list of spent coins; Else, set the error-flag $\mathsf{err\text{-}in}_i$ to true and $\mathsf{err\text{-}sum} = \mathsf{true}$.
   - (legal values) For $j \in \{B, C\}$: If either the range proof Π_{Range_j} or the knowledge-of-committed-identity proof $\Pi_{\mathsf{icm}\,j}$ do not pass verification, set the error-flag $\mathsf{err\text{-}val}_j$ to true.
   - (legal sum) If the sum-proof Π_{Sum} does not pass verification set the error-flag $\mathsf{err\text{-}sum}$ to true.

   If at least one of the error flags is on, broadcast (e.g., via the ledger) to all parties the error flag $\mathsf{err} = \mathsf{true}$ and terminate the operation. Else, the Bank appends the nullifier nullif_i to the public list of used coins and continue.

2. (Validating the new coins) The Bank chooses two new serial numbers sn_B and sn_C by setting $\mathsf{sn}_B = H(\mathsf{nullif}_1, \mathsf{nullif}_2, 1)$ and $\mathsf{sn}_C = H(\mathsf{nullif}_1, \mathsf{nullif}_2, 2)$ where H is a hash function that is modeled, for simplicity, as a random oracle. (Alternatively, we can use a correlation-robust hash function; see footnote 22.). For $j \in \{B, C\}$, the Bank approves the jth coin request as follows:

   (a) The Bank Homomorphically computes the commitment

   $$\mathsf{ccm}_j = \mathsf{icm}_j \boxplus \mathsf{sncm}_j \boxplus \mathsf{vcm}_j, \quad \text{where } \mathsf{sncm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (0, \mathsf{sn}_j, 0, 0); 0).$$

   (Supposedly, $\mathsf{ccm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_j, \mathsf{sn}_j, v_j, 0); \rho_j + t_j)$.)

   (b) The Bank signs the commitment ccm_j using the Bank's signing key bsk with fresh private randomness u_j. Let $\sigma_j = \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm}_j; u_j)$ denote the signature.

   The Bank appends to the ledger the entries

   $$(t, \mathsf{nullif}_1, \mathsf{nullif}_2), \quad (\mathsf{ccm}_j, \sigma_j, \mathsf{ctxt}_j) \qquad j \in \{B, C\}, \tag{9}$$

   where t is the current value of the counter T. In addition, the Bank increases the global counter T by 2 and publishes its state on the ledger. The Bank terminates this operation with the output `paid`.

Finally, each client Rec with public identifier pid, retrieves the new entry from the ledger. If this is an err message, then the client outputs err. Otherwise, the client parses the ledger's new entry as

$$(t, \mathsf{nullif}_1, \mathsf{nullif}_2), \quad (\mathsf{ccm}_j, \sigma_j, \mathsf{ctxt}_j)_{j\in\{1,2\}}.$$

For each $j \in \{1, 2\}$, the client retrieves the ith tuple, $(\mathsf{ccm}_j, \sigma_j, \mathsf{ctxt}_j)$, in this entry and applies the following *Claim* operation:

1. (Recovering the coin handles) Check that ctxt_j did not appear in any previous entry of the ledger, and if this is the case try to decrypt the ciphertext ctxt_j by using the client's private key sk. If decryption succeeds, parse the plaintext as (v, r) and check that

   $$\mathsf{ccm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}, v, 0); r)$$

   and $\mathsf{sn} = H(\mathsf{nullif}_1, \mathsf{nullif}_2, j)$. Also, verify that the signature σ is a valid Bank's signature on ccm_j. If any of the above tests fail, abort the claim procedure for this entry with an output $(\mathtt{paid}, t + j)$.

2. (Rerandomizing the coin) Sample a new commitment randomizer r', and a signature randomizer u', re-randomize the signed committed coin $(\mathsf{ccm}_j, \sigma_j)$ into (ccm', σ') where

   $$\mathsf{ccm}' = \mathsf{CM.Rerand}(\mathsf{pp}, \mathsf{ck}, \mathsf{ccm}_j; r'), \quad \sigma' = \mathsf{RS.Rerand}((\mathsf{pp}, \mathsf{bvk}, \mathsf{ck}), \sigma_j; r', u').$$

3. Append the coin $(\mathsf{ccm}', \sigma', \mathsf{sn}, r + r', v, t + j)$ to the private state and output $(\mathtt{paid}, t + j, v_j)$.

3.4 The minting sub-protocol

The minting protocol can viewed as a degenerate version of the payment protocol.

The Minter: Given an input $(\mathtt{mint}, v, \mathsf{pid})$, where $v \in V$, the minter prepares a coin commitment for the payee $\mathsf{C}_{\mathsf{pid}}$ with value v:

$$\mathsf{ccm} = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}, v, 0); r),$$

where r is a fresh randomizer and $\mathsf{sn} = H(\mathsf{T}, z)$ were z is a random nonce. The minter also computes a ciphertext $\mathsf{ctxt} \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}, (v, r))$ for the payee, and sends the tuple $(\mathsf{sn}, \mathsf{ccm}, \mathsf{ctxt})$ to the Bank.

The Bank: The Bank signs the commitment ccm using the signing key bsk with fresh private randomness u. Let $\sigma = \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm}; u)$ denote the signature. The Bank appends to the ledger the entry

$$t, \quad (\mathsf{sn}, \mathsf{ccm}, \sigma, \mathsf{ctxt}) \tag{10}$$

where t is the current value of the counter T. In addition, the Bank increases the global counter T by 1 and publishes its state on the ledger.

Each potential Payee Rec**:** Retrieves the last tuple from the ledger, and claim the coin according to the "Claim" procedure defined in the payment protocol. (With the modification that sn is recovered from the ledger and in the output $\mathtt{paid}$ is replaced with $\mathtt{minted}$.)

4 Extending the Protocol

4.1 The Threshold Setting

Let us briefly explain how to extend the protocol to the threshold setting in which there are n Banks, $\mathsf{bank}_1, \ldots, \mathsf{bank}_n$, and where the adversary may actively corrupt at most t of them. Intuitively, the idea is

to replace each operation of the "single Bank" in the protocol by a corresponding MPC sub-protocol that is distributively executed by the group of n Banks. Formally, let us think of the Bank in the above protocol as a trusted-party Virtual Bank, and view the protocol as operating in a hybrid model. We analyze the protocol in this hybrid model (Sections 5 and 6), and so, by using proper MPC composition theorems (e.g., [9, 16]), one can conclude that when the trusted party is replaced by a t-out-of-n secure protocol over the actual banks, $\mathsf{bank}_1, \ldots, \mathsf{bank}_n$, the resulting protocol remains secure.

Realizing Virtual Bank. Let us take a closer look at the Virtual Bank. During Initialization, the Virtual Bank samples two sets of signature/verification keys by calling $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck})$ and keeps the verification keys as part of the secret state of the Virtual Bank. This is the only secret state that is kept by the Virtual Bank. All the other operations of the Virtual Bank take the following simple form: The Virtual Bank gets some public input (by receiving a message from a client and/or reading some part of the public ledger), checks that the input satisfies some predefined public condition, applies some deterministic public computation, and, in some cases (depending on the public information) issues a signature on some committed value, and publishes the final results.

Consequently, in order to securely realize this process by the banks, $\mathsf{bank}_1, \ldots, \mathsf{bank}_n$, all that is needed is an appropriate protocol for threshold signing a commitment, and some form of broadcast channel from clients to Banks that guarantees that all Banks receive the same message. The latter mechanism is instantiated by a BFT system. Let us briefly expand on the notion of threshold signatures that suffices for our needs.

Fix a committed signature scheme (as per Definition 3), and consider the initialization functionality that given pp samples a commitment key $\mathsf{CM.Setup}(\mathsf{pp}) \to \mathsf{ck}$ and 2 pairs of signature/verification keys $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \to (\mathsf{bsk}, \mathsf{bvk})$ and $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \to (\mathsf{rsk}, \mathsf{rvk})$, and delivers t-out-of-n secret sharing of the secret keys bsk and rsk to the Banks, $\mathsf{bank}_1, \ldots, \mathsf{bank}_n$, and broadcasts the values $(\mathsf{ck}, \mathsf{bvk}, \mathsf{rvk})$ to everyone. In addition, we need a signing functionality that takes from the Banks n shares of the secret-signing key out of which at least t are valid, and takes from a client a commitment cm together with its opening m, r such that $\mathsf{cm} = \mathsf{CM.Commit}(\mathsf{pp}, \mathsf{ck}, m; r)$, delivers to the client a valid signature σ over cm. In the paper [1], we present information-theoretic protocols that realize these functionalities with respect to a variant of the PS signatures [21] over dual "dual" Pedersen commitments. The protocol has 3 rounds where at the first round, the client sends some message $a = A(m, r; \eta)$ where ρ is some private randomness, the Banks respond with some public values $b = (b_1, \ldots, b_n)$ that are computed based on their private shares and on a, and finally, the client computes the signature σ by applying some procedure $C(m, r, \eta, b)$. This sub-protocol can be integrated into the protocol without increasing the round complexity as follows. Upon payment, the client appends a to its transaction and publicly sends it to the Banks in addition the client appends the private randomness η to the corresponding ciphertext ctxt that is addressed to the payee. The Banks compute their response just like Virtual Bank and if signature is required they place their "sub-signature" b_i on the ledger. The payee can then apply an extended-Claim operation in which the values m, r, η are recovered from the ciphertext and the signature σ is obtained by completing the C-step of the sub-protocol. [13]

4.2 Realizing the generalized $\mathcal{F}_{\mathsf{utt}}$ functionality

Recall that in Remark 1.2 we mentioned a generalized form of $\mathcal{F}_{\mathsf{utt}}$ that supports coins that carry with them a "type" (vector of attributes) and where the notion of a legal payment depends on rules that take into account the types. We note that our protocol can be naturally extended to work in such a setting by using vector commitments with more "slots" (say by adding more public generators to Pedersen's commitments) and by designing ZKPOK that can verify the validity of the generalized payment rules. Indeed, this is exactly the approach taken in [1] in order to realize "privacy budgets". It can be verified that our security proof (Sections 5 and 6) naturally extends to this generalized version of the protocol provided that the corresponding ZK building blocks are sound.

[13] Additionally, in [1] the Banks are in charge of generating the IBE secret keys. Here too, an appropriate (simpler) information-theoretic MPC protocol is being employed (taken from the Boneh-Franklin paper [8]).

5 Simulation

Fix an adversary Adv that actively corrupts a subset $M \subset \mathsf{C}$ (for malicious) of the clients and recall that Adv also passively listens to all the communication of the Bank which includes all incoming messages and all the outgoing messages (that are directed to the ledger anyway). We define a corresponding simulator $\mathcal{S}$ that treats Adv in a black-box straight-line manner and interacts with the $\mathcal{F}_{\mathsf{utt}}$ functionality while taking the role of parties corrupted by the adversary in the real execution. Recall that the inputs are injected to the system in an online manner by the environment. That is, in each round the environment sends either a mint operation to the minter, or a payment operation to an honest client, or some message to the adversary that results in a payment operation by some corrupted client. Each of the following subsections is devoted to one of these case. Following the discussion in Section 4.1, the simulator treats the Virtual Bank as an ideal functionality. This functionality can be trivially instantiated given a single honest Bank and can be realized with an appropriate MPC protocol among n banks out of which t are honest. The simulation will run the adversary Adv internally while maintaining a simulated ledger that will be always available to Adv.

The simulator $\mathcal{S}$ is described in the following subsections.

5.1 Initialization

The simulator proceeds as follows:

1. Samples public parameters $\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$, and a pair of master secret-key/public-key $(\mathsf{msk}, \mathsf{mpk})$ for an Identity-Based Anonymous Encryption Scheme, and sends to each corrupted client a secret key $\mathsf{sk}^{\mathsf{ibe}}$ that corresponds to her public identifier pid. The tuple $(\mathsf{pp}, \mathsf{mpk})$ is being added to the ledger and msk is being kept as part of the simulator's private state.

2. The Virtual Bank initializes its state just like in the protocol. That is, it samples $\mathsf{ck} \xleftarrow{\$} \mathsf{CM.Setup}(\mathsf{pp})$, and calls $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck})$ twice generating two pairs of signing/verification keys $(\mathsf{bsk}, \mathsf{bvk})$ and $(\mathsf{rsk}, \mathsf{rvk})$. The private signing keys are being kept as part of its private state and the verification keys bvk and rvk are appended to the ledger together with the commitment key ck, and a counter $\mathsf{T} = 0$.

3. The simulator mimics the behavior of every honest party $\mathsf{C}_{\mathsf{pid}}$ in the initialization step. That is, for every honest client $\mathsf{C}_{\mathsf{pid}} \notin M$, whose public identifier is pid, the simulator samples an initial random PRF key $\mathsf{s}_0 \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$, and computes a commitment

$$\mathsf{rcm}_0 = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0); a)$$

   where $a \xleftarrow{\$} \mathcal{R}$ is fresh randomizer for the commitment scheme. The simulator sends to the Virtual Bank the values $(\mathsf{pid}, \mathsf{rcm}_0)$ together with a non-interactive zero-knowledge proof of knowledge Π_{Init} for the knowledge of (a, s_0) that satisfies the relation $\mathsf{rcm}_0 = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0); a)$. The Virtual Bank proceeds as in the protocol, i.e., sends

$$(\Delta, \quad \mathsf{rcm}, \quad \mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b)),$$

   where $\Delta \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$, $\mathsf{rcm} = \mathsf{rcm}_0 + \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, 0, \Delta); 0)$ and b is chosen at random. The simulator sets the address secret key of this client to be $\mathsf{ask}_{\mathsf{pid}} = (\mathsf{s} = \mathsf{s}_0 + \Delta, \mathsf{sk}^{\mathsf{ibe}}, \mathsf{rcm}, \mathsf{rs}, a)$ and appends it to its private state.[14]

4. For each corrupted client $\mathsf{C}_{\mathsf{pid}} \in M$, the adversary responds with an initialization information $(\mathsf{pid}, \mathsf{rcm}_0, \Pi_{\mathsf{Init}})$. The simulator uses msk to compute the secret decryption key, $\mathsf{sk}^{\mathsf{ibe}}$, that is associated with pid. The simulator checks if the ZK verification passes.

[14] We note that in principle rcm and rs can be public. Indeed, these values are transferred over a public channel and so they are available to all parties.

(a) If verification succeeds, $\mathcal{S}$ extracts from Π_{Init} the secret PRF key s_0 and the commitment randomizer a. Given $(\mathsf{pid}, \mathsf{rcm}_0, \Pi_{\mathsf{Init}})$, the Virtual Bank, who follows the protocol, sends back to the simulator the tuple

$$(\Delta, \quad \mathsf{rcm}, \quad \mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b)),$$

where $\Delta \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$, $\mathsf{rcm} = \mathsf{rcm}_0 + \mathsf{CM.Commit}(\mathsf{ck}, (0,0,0,\Delta); 0)$ and b is chosen at random. The simulator passes the tuple to the adversary, sets the address secret key of $\mathsf{C}_{\mathsf{pid}}$ to be $\mathsf{ask}_{\mathsf{pid}} = (\mathsf{s} = \mathsf{s}_0 + \Delta, \mathsf{sk}^{\mathsf{ibe}}, \mathsf{rcm}, \mathsf{rs}, a)$ and appends it to its private state.

(b) If verification fails, the Virtual Bank follows the protocol and sends back to $\mathcal{S}$ the tuple

$$(\mathsf{s}, \quad a, \quad \mathsf{rcm} = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a), \quad \mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b)),$$

where s, a, and b are sampled uniformly. The simulator passes the tuple to the adversary, sets the address secret key of $\mathsf{C}_{\mathsf{pid}}$ to be $\mathsf{ask}_{\mathsf{pid}} = (\mathsf{s}, \mathsf{sk}^{\mathsf{ibe}}, \mathsf{rcm}, \mathsf{rs}, a)$ and appends it to its private state.

5.2 Minting operation

Assume that the environment sends the input $(\mathtt{mint}, v, \mathsf{pid})$ to the Minter who passes this input to $\mathcal{F}_{\mathsf{utt}}$. The simulator gets from $\mathcal{F}_{\mathsf{utt}}$ the value $(\mathtt{minted}, \mathsf{T})$.

1. If the payee is corrupted ($\mathsf{C}_{\mathsf{pid}} \in M$), the $\mathcal{F}_{\mathsf{utt}}$ functionality sends to the simulator also the identity pid of the payee and the value v. Set $\mathsf{pid}_0 = \mathsf{pid}$ and $v_0 = v$.

2. If the payee is honest ($\mathsf{C}_{\mathsf{pid}} \notin M$), then the simulator arbitrarily selects an identity pid_0 of some honest user $\mathsf{C}_{\mathsf{pid}_0} \notin M$, e.g., the lexicographically first honest user, and sets v_0 to some arbitrary value, e.g., $v_0 = 1$.

Next, the simulator performs the minting operation exactly as in the protocol by playing the role of the Minter with respect to the value v_0 and receiver pid_0. That is, the simulator sends to the Virtual Bank the values

$$\mathsf{ccm} = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_0, \mathsf{sn}, v_0, 0); r),$$

where r is a fresh randomizer and $\mathsf{sn} = H(\mathsf{T}, z)$ were z is a random nonce. The minter also computes a ciphertext $\mathsf{ctxt} \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}_0, (v_0, r))$ for the receiver, and sends the tuple $(\mathsf{sn}, \mathsf{ccm}, \mathsf{ctxt})$ to the Virtual Bank. The Virtual Bank continues just like in the protocol. That is, the Virtual Bank appends to the ledger the entry

$$t, \quad (\mathsf{sn}, \mathsf{ccm}, \sigma, \mathsf{ctxt})$$

where t is the current value of the counter according to the ledger, and $\sigma = \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm}; u)$ for some randomly chosen u. In addition, the Virtual Bank increases the ledger's counter by 1 and publishes its state on the ledger.

5.3 Payment operation by an honest client

Assume that the environment sends the input $(\mathsf{pay}, t_1, t_2, \mathsf{pid}_B, v_B, \mathsf{pid}_C, v_C)$ to some honest client $\mathsf{Sender} \notin M$ who passes this command to $\mathcal{F}_{\mathsf{utt}}$. We may assume WLOG that the honest party performs a legal operation and so the simulator gets from $\mathcal{F}_{\mathsf{utt}}$ a "success" $\mathtt{paid}$ message. (If this is not the case, and the simulator receives an error message from $\mathcal{F}_{\mathsf{utt}}$, the simulator simply passes it to the Virtual Bank who passes it forward to the adversary.)

- For $j \in \{B, C\}$:
  If the payee (identified by) pid_j is corrupted, the $\mathcal{F}_{\mathsf{utt}}$ functionality sends to the simulator also the identity pid_j and the value v_j together with the corresponding state of the counter T_j. Set $v'_j = v_j$, and $\mathsf{pid}'_j = \mathsf{pid}_j$.

Else (i.e., the payee pid_j is honest) set pid'_j to be the identifier of some arbitrary honest user (say the lexicographically first honest user) and set $v'_j = 1$. (The value 1 can be replaced with some other default value in $[0, \mathsf{V_{max}}]$.)

- The simulator then sends to the Virtual Bank a "fake" payment request:

$$((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),\ (\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}},\ \Pi_{\mathsf{Sum}} \quad (11)$$

which is computed as follows.

1. (Generate a fake honest sender) Select a random identity pid_0 (that does not belong to any existing client). Sample a random PRF key $\mathsf{s}_0 \xleftarrow{\$} \mathcal{K}_{\mathsf{pp}}$ for this fake user and compute a fresh commitment $\mathsf{rcm} = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a)$. The adversary locally generates a fresh signature $\mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b)$ where b is chosen at random.[15]
2. (Generate fake incoming coins) Set $(v_1, v_2) = (v'_B, v'_C)$. For $i \in \{1, 2\}$ generate a fake coin, by computing
$$\mathsf{ccm}_i = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_0, \mathsf{sn}_i, v_i, 0); r_i),$$
where r_i is a fresh randomizer and sn_i is uniformly distributed over the range of the PRF. Then locally generate a fresh signature σ_i over ccm_i under the key bsk. (See Footnote 15.)
3. (Split incoming coins) For $i \in \{1, 2\}$, set
$$\mathsf{vcm}_i = \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, v_i, 0); z_i) \quad \text{and} \quad \mathsf{nullif}_i = \mathsf{PRF}_{\mathsf{s}_0}(\mathsf{sn}_i),$$
where z_i is a fresh randomizer.
4. (Randomize registration signatures) Rerandomize the registration signature $(\mathsf{rcm}, \mathsf{rs})$ into $(\mathsf{rcm}', \mathsf{rs}')$.[16]
5. (Prove consistency of splitting) For each $i \in \{1, 2\}$, the simulator honestly generates a proof Π_{Split_i} for the statement "$\mathbb{x} = (\mathsf{ccm}_i, \mathsf{vcm}_i, \mathsf{rcm}', \mathsf{nullif}_i)$ is in the split-relation" which is defined in Eq. (4). Note that this can be done efficiently since the simulator holds the corresponding witnesses.
6. (Generate coin requests) Prepare two coin requests for the payees (identified by) pid'_B and pid'_C with values v'_B and v'_C, respectively, by following Step 5 of the real protocol. That is, for $j \in \{B, C\}$, set an identity and value commitments via
$$\mathsf{icm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}'_j, 0, 0, 0); t_j) \quad \text{and} \quad \mathsf{vcm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, v'_j, 0); \rho_j), \quad (12)$$
where t_B, t_C and ρ_B, ρ_C are fresh randomizers.
7. (Prove validity of coin requests) For each coin request $j \in \{B, C\}$, the simulator uses the honest prover algorithms to generate the POK of opening for icm_j denoted by $\Pi_{\mathsf{icm}\,j}$, and to generate a ZKPOK Π_{Range_j} for the statement "vcm_j satisfies the range relation", as defined in Eq. (6). In addition, the simulator uses the honest prover algorithm to generate the sum-proof Π_{Sum} for the statement that asserts that the sum of values in the incoming coins is equal to the sum of values of the outgoing coins, i.e., that
$$\mathbb{x} = (\mathsf{vcm}_1, \mathsf{vcm}_2, \mathsf{vcm}_B, \mathsf{vcm}_C), \quad \mathbb{w} = (\mathsf{val}'_1, z_1, \mathsf{val}'_2, z_2, v'_B, \rho_B, v'_C, \rho_C)$$
satisfies the relation $\mathcal{R}_{\mathsf{sum}}$ as defined in Eq. 7. (Note that $\Pi_{\mathsf{icm}\,B}, \Pi_{\mathsf{icm}\,C}, \Pi_{\mathsf{Range}\,B}, \Pi_{\mathsf{Range}_C}, \Pi_{\mathsf{Sum}}$ are all proofs for "valid" statements that are being generated by running the honest prover's algorithm. Indeed, the simulator holds the witnesses for all these assertions.)
8. (Append coin handles for the payee) The client prepares a ciphertext $\mathsf{ctxt}_j \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}'_j, (v'_j, \rho_j + t_j))$ for each recipient $j \in \{B, C\}$.

Next, the Virtual Bank processes the request exactly as in the real protocol and updates the ledger accordingly. By construction, the Virtual Bank approves the transaction.

[15] Recall that during initialization the simulator receives from the ideal Virtual Bank all the signing-key shares that belong to the honest Banks. Since the secret-sharing threshold t is smaller than the number of honest parties, the simulator holds the signing keys and can locally sign messages.

[16] This step is actually redundant since $(\mathsf{rcm}, \mathsf{rs})$ were generated as fresh values, but we add it here for the sake of clarity.

5.4 Payment operation by a corrupted client

Assume that the environment sends the input χ to the adversary. The simulator passes this command to the adversary Adv who responds on behalf of some corrupted client with some (possibly malformed) message to the Virtual Bank

$$((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),\ (\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}},\ \Pi_{\mathsf{Sum}}. \tag{13}$$

The simulator passes (13) to the Virtual Bank and translates this command to an $\mathcal{F}_{\mathsf{utt}}$ payment command $(\mathsf{pay}, t'_1, t'_2, \mathsf{pid}'_B, v'_B, \mathsf{pid}'_C, v'_C)$ on behalf of some corrupt client $\mathsf{Sender}' \in M$ defined as follows.

1. (legal incoming coins) For $i \in \{1, 2\}$ do:

   - Check if (a) σ_i is a valid signatures over the committed coin ccm_i with respect to the Virtual Bank's verification key bvk; and (b) the split-proof Π_{Split_i} (together with the relevant tuples) passes verification and (c) rs' is a valid signature on rcm' with respect to the registration key rvk; and (d) nullif_i is not in the in the list of spent coins that is maintained in the ledger.
   - If any of the conditions (a–d) fails: The simulator sets t'_i to some illegal value (e.g., -1), , and sets Sender' to some arbitrary (e.g., the lexicographically first) corrupted party.
   - If conditions (a–d) pass: the Simulator searches the ledger for a (successful) transaction whose outgoing coin has a serial number[17] of sn for which there exists a client C whose PRF key s satisfies

     $$\mathsf{nullif}_i = \mathsf{PRF}_{\mathsf{s}}(\mathsf{sn}).$$

     If such an entry is found, set t'_i to be the corresponding counter's state, $\mathsf{sn}_i := \mathsf{sn}$, and $\mathsf{Sender}'_i := \mathsf{C}$. Else, the simulation aborts with "simulation failure" symbol.

   If $\mathsf{Sender}'_1 \neq \mathsf{Sender}'_2$ or $\mathsf{Sender}'_1 \notin M$, the simulation aborts with an error. Else, set $\mathsf{Sender}' := \mathsf{Sender}'_1$.

2. (legal values) For $j \in \{B, C\}$ check that the range proof Π_{Range_j} and the knowledge-of-committed-identity proof $\Pi_{\mathsf{icm}\,j}$ pass verification.

   - If any of these conditions fail: The simulator sets v'_j to some illegal value (e.g., -1), , and sets pid'_j to some arbitrary (possibly illegal) value.
   - If both conditions pass, use the knowledge extractor to extract from Π_{Range_j} the value v and from $\Pi_{\mathsf{icm}\,j}$ the identifier pid. Set $v'_j = v$ and $\mathsf{pid}'_j = \mathsf{pid}$.

3. (legal sum) Check if the sum-proof Π_{Sum} passes verification.

   - If verification fails update v'_1 to some illegal value

4. (Validating honest payees) If conditions (1–3) are satisfied, the simulator tests whether, in a real execution, the outgoing coins can be claimed successfully by an honest payee. Specifically, for each $j \in \{B, C\}$ such that $\mathsf{C}_{\mathsf{pid}'_j}$ is honest, we do the following: Verify that ctxt_j does not appear in any previous entry on the ledger and if this is the case, try to decrypt this ciphertext by using the private key sk associated with $\mathsf{C}_{\mathsf{pid}'_j}$. If decryption succeeds, parse the plaintext as (v, r) and check that $\mathsf{ccm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}'_j, \mathsf{sn}_j, v, 0); r)$ where $\mathsf{sn}_B = H(\mathsf{nullif}_1, \mathsf{nullif}_2, 1)$ and $\mathsf{sn}_C = H(\mathsf{nullif}_1, \mathsf{nullif}_2, 2)$. If the test fails, change pid'_j to some arbitrary string that does not match any of the existing identifiers.

The simulator sends the payment command

$$(\mathsf{pay}, t'_1, t'_2, \mathsf{pid}'_B, v'_B, \mathsf{pid}'_C, v'_C)$$

to the ideal functionality $\mathcal{F}_{\mathsf{utt}}$ on behalf of Sender'. In addition, the Virtual Bank processes the request (13) as in the real protocol and updates the ledger accordingly.

[17] Recall that the serial number of the outgoing coins either appear explicitly on the ledger in a mint operation, or can be computed publicly based on the incoming nullifiers.

6 Analysis of the Simulator

Reminder: Fix a computationally-bounded environment Env and let 1^λ and z denote its inputs. Recall that in each round the environment sends either a mint/payment command to the (honest) minter/some honest client, or an arbitrary message to the adversary who translates it into a message of some corrupted client. After each such round, the environment Env gets back the view of all the corrupted parties which includes (1) their private state, (2) the (public) state of the ledger and (3) the content of all messages that are being sent to the Bank by the honest parties. The environment also receives the outputs that are generated by all the honest parties. Based on all this information the environment chooses the content and recipient of the next command. At the end, the environment outputs a single bit, and halts.

Our goal is to prove that the ensemble of binary random variables defined in (1) and (2) are indistinguishable.

Fix a sequence $\{z_\lambda\}_{\lambda\in\mathbb{N}}$ of inputs for the environment. Let $\Delta(\lambda)$ denote the statistical distance between (1) and (2), and let $\mu := \mu(\lambda)$ be a negligible function that upper-bounds the success probability of any polynomial-time adversary in breaking each of the underlying primitives.[18] We further assume that μ upper-bounds the quantities $N^2Q^2/|\mathcal{X}_{\mathsf{pp}}|$ and $N^2/|\mathcal{K}_{\mathsf{pp}}|$ where $\mathcal{K}_{\mathsf{pp}}$ is the key-space of the PRF, $\mathcal{X}_{\mathsf{pp}}$ and is the domain of the PRF, N is the number of clients, and Q upper-bounds the number of queries that the adversary makes to the random oracle.[19]

We show that $\Delta(\lambda)$ is upper-bounded by the negligible function $(\mu(\lambda)\cdot p(\lambda))^c$ where $c \geq 1$ is some universal constant, and $p(\lambda)$ upper-bounds the number of commands that is sent by $\mathsf{Env}(1^\lambda, z_\lambda)$. The proof is based on a hybrid argument.

Hybrid experiment. Fix some security parameter λ, let $z = z_\lambda$, $\Delta = \Delta(\lambda)$, and let $p = p(\lambda)$ denote the maximal number of commands that $\mathsf{Env}(1^\lambda, z)$ issues. For $\ell \in [p]$, we define a hybrid experiment $\mathcal{H}_\ell$, in which we run in parallel the real execution (attacked by Adv) and an idea execution (attacked by the simulator), where in the first phase (that consists of the first ℓ commands) the environment $\mathsf{Env}(1^\lambda, z)$ interacts with the real execution and in the second phase environment communicates with the ideal execution. Details follow.

1. (Initialization) Given a list of the identifiers of the participating parties (that is given as part of the specification of the functionality and may also be chosen adversarially), the environment declares chooses which parties to corrupt. We run the initialization procedure of the real protocol. We also initialize the simulator consistently with the same public parameters $(\mathsf{pp}, \mathsf{ck}, \mathsf{mpk}, \mathsf{bvk}, \mathsf{rvk})$, and the same private initialization values (i.e., the same $\mathsf{msk}, \mathsf{bsk}, \mathsf{rsk}$ and the same address secret keys of all clients). We initialize the $\mathcal{F}_{\mathsf{utt}}$ functionality with the same set of clients.

2. (First Phase: Ideal execution) We pass the view of Adv (after initialization) to the environment. Then, each of the first ℓ commands generated by the environment is handled as in the real execution attacked by Adv. In parallel, each of these command is also executed in the ideal execution attacked by the simulator. That is, a command that is issued by an honest party is forwarded to $\mathcal{F}_{\mathsf{utt}}$ and to the simulator, and a command that is issued by a corrupted party controlled by Adv, is translated into an $\mathcal{F}_{\mathsf{utt}}$ command via the help of the simulator. In this phase, we do not let the simulator write on the ledger and whenever the simulator wishes to read the ledger (i.e., when checking the nullifier list) it uses the ledger that is maintained by the real execution. After the command terminates, the view of Adv and the output of the honest parties in the *real execution* is being forwarded to the environment who may use it to select its next command. (Note that, in this phase, the ideal execution has no effect on the environment's view.)

3. (Second Phase: Ideal execution) The remaining $p - \ell$ commands are handled in the ideal protocol. That is, at the beginning of this phase, we pass to the simulator the ledger as defined by the real

[18] In fact, it suffices to consider adversaries whose running time is a fixed polynomial in the complexity of Env and Adv and the complexity of all the underlying primitives.

[19] As usual, Q is upper-bounded by the running time of the adversary and is therefore polynomially bounded. We mention that we did not try to optimize the concrete bound on the distinguishing advantage, and we believe that it can be tighten.

execution in the first phase, and continue as before except that now after each command, we let the simulator update the ledger, and pass to the environment the simulated view of the corrupted parties, and the output of the honest parties in the *ideal execution*.

4. (Output) The final output of the game is the output if the environment.

By definition, $\mathcal{H}_p$ is is identical to the real execution $\mathsf{Exec}_{\Pi,\mathsf{Adv},\mathsf{Env}}(1^\lambda, z)$. Also, it is not hard to see that the game $\mathcal{H}_0$ is identical to an ideal execution $\mathsf{Exec}_{\mathcal{F}_{\mathsf{utt}},\mathcal{S},\mathsf{Env}}(1^\lambda, z)$. (Indeed, this follows by noting that the initialization procedure in an ideal execution and real execution is performed identically.) Therefore it suffices to prove that $|\Pr[\mathcal{H}_{\ell+1} = 1] - \Pr[\mathcal{H}_\ell = 1]|$ is upper-bounded by $O(\ell\mu)$. In fact, it will be convenient to embed the two experiments in the same probability space, and assume that the same random tape R is being used for both experiments. That is, we prove the following lemma.

Lemma 6.1. For every ℓ, the gap $\Delta_\ell := \Pr_R[\mathcal{H}_{\ell+1}(R) \neq \mathcal{H}_\ell(R)]$ is upper-bounded by $O((\ell+1)\mu)$.

The use of a common random tape R ensures that all the internal variables computed during the first ℓ iterations are identical in both experiments. (During the $(\ell+1)$th iteration, different parts of the tapes are being read.) We can naturally define the intersecting probability of a "successful distinguishing" AND an event E, via

$$\Delta_{\ell,E} := \Pr_R[\mathcal{H}_{\ell+1}(R) \neq \mathcal{H}_\ell(R) \quad \wedge \quad E(R)].$$

Specifically, we will prove Lemma 6.1 by induction on ℓ and analyze $\Delta_{\ell,E}$ seperately for the event E_m in which the $(\ell+1)$th command is a mint (Section 6.1), the event E_h in which the $(\ell+1)$th command is a payment of an honest party (Section 6.2), and the event E_c in which the the $(\ell+1)$th command is a payment of a corrupted party (Section 6.3).

6.1 Mint command

Fix some ℓ. We begin with the following observation that will also be useful in the subsequent sections.

Observation 6.1. *Consider the first ℓ iterations of $\mathcal{H}_\ell$ and let T denote the number of iterations in which the public output of the protocol (as generated by the Bank) is successful. Let T' denote the number of iterations in which the simulated output (as computed the ideal functionality) is successful. Then, except with probability $O(\ell\mu)$, it holds that $\mathsf{T}' = \mathsf{T}$.*

Proof. Indeed, when $\ell = 0$, this is true since $\mathsf{T} = \mathsf{T}' = 0$, and for $\ell > 0$, this follows from the induction hypothesis. Specifically, T and T' are available to the environment in $\mathcal{H}_\ell$ and $\mathcal{H}_{\ell-1}$, respectively, and therefore, by Lemma 6.1 for $\ell' = \ell - 1$, these values can differ with probability at most $O(\ell\mu)$. □

Consider the event E_m (for minting) that at the $(\ell+1)$th round the environment sends to the minter a minting command, denoted by $(\texttt{mint}, v, \mathsf{pid})$. Our goal is to upper-bound Δ_{ℓ,E_m} by $O((\ell+1)\mu)$. By Observation 6.1, it suffices to show that, conditioned on the event that $\mathsf{T}' = \mathsf{T}$, the probability Δ_{ℓ,E_m} is upper-bounded by $2\mu_{\mathrm{IBE}} + \mu_{\mathrm{COM}}$ where μ_{IBE} upper-bounds the probability that an efficient IBE adversary wins at the IND-RA-CCA game, and μ_{COM} upper-bounds the probability that an efficient adversary breaks the hiding property of the commitment.

Since the minting operation always succeeds, the output of the honest party both in the $\mathcal{H}_\ell$ and $\mathcal{H}_{\ell+1}$ is `paid`. Hence, the only difference in the view of the environment is in the message $(\mathsf{ccm}, \mathsf{ctxt})$ that the simulator/minter sends to the Bank and in the Bank's response as written on the ledger. Recall that the Bank's response consists of the value of the counter T, and a signature on the Minter's message. Therefore, it suffices to show that the Minter's message in $\mathcal{H}_\ell$ is indistinguishable from its message in $\mathcal{H}_{\ell+1}$, even with respect to an adversary who holds the Bank's private signing key. We show that this is indeed the case.

If the payee is corrupted then the simulator computes this message exactly as in the real protocol and in this case the view is identically distributed and $\Delta_{\ell,E_m} = 0$. We move on to the case where the payee is honest. In $\mathcal{H}_\ell$ the message $(\mathsf{sn}, \mathsf{ccm}, \mathsf{ctxt})$ is generated by

$$\mathsf{sn} = H(\mathsf{T}+1, z), \qquad \mathsf{ccm} = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_0, \mathsf{sn}, v_0, 0); r), \qquad \mathsf{ctxt} \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}_0, (v_0, r, \mathsf{sn})), \quad (14)$$

where pid_0 is some fixed honest identity, v_0 is some default value, r is a fresh randomizer, and T is the number of coins that were successfully generated so far. In $\mathcal{H}_{\ell+1}$ the pair $(\mathsf{ccm}, \mathsf{ctxt})$ is generated by

$$\mathsf{sn} = H(\mathsf{T}' + 1, z), \qquad \mathsf{ccm} = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}, v, 0); r), \qquad \mathsf{ctxt} \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}, (v, r, \mathsf{sn})), \tag{15}$$

where, again, r is a fresh randomizer and sn is defined as above. By assumption, $\mathsf{T} = \mathsf{T}'$. Let us further condition on the event that the same public randomizer z is chosen in both cases and so the serial number sn is also equal. We can therefore focus on the marginal distribution of $(\mathsf{ccm}, \mathsf{ctxt})$, and show that (14) is indistinguishable from (15) even with respect to an adversary who holds all the private signing keys of the Bank(s).

Consider the sub-hybrid $\mathcal{H}'_\ell$ in which ccm is computed as in (14) but $\mathsf{ctxt} \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}, (v, r', \mathsf{sn}))$ where r' is an independently chosen randomizer.

We begin by proving that $\mathcal{H}'_\ell$ is indistinguishable from $\mathcal{H}_\ell$ by describing an efficient adversary $\mathcal{B}$ that breaks the security of the IBE scheme with advantage $\delta = |\Pr[\mathcal{H}_\ell = 1] - \Pr[\mathcal{H}'_\ell = 1]|$. The adversary $\mathcal{B}$ plays the IND-RA-CCA game as follows. First, $\mathcal{B}$ initializes $\mathcal{H}_\ell$ where the IBE public-key mpk is taken to be the one that is given by the IND-RA-CCA game. Moreover, $\mathcal{B}$ uses the key-extraction oracle to learn the private keys of all the parties that are controlled by the adversary. The other initialization values are sampled locally. Then, $\mathcal{B}$ emulates $\mathcal{H}_\ell$ for the first i steps. During these iterations whenever an encryption of an honest party is performed, $\mathcal{B}$ just uses the encryption algorithm with the corresponding identity, and whenever a decryption is performed $\mathcal{B}$ uses the decryption oracle. At the $(\ell+1)$th step, $\mathcal{B}$ makes a challenge query

$$(\mathsf{pid}_0, m_0 = (v_0, r, \mathsf{sn})), \quad \text{and} \quad (\mathsf{pid}, m_1 = (v, r', \mathsf{sn}))$$

in the IND-RA-CCA game. Given the oracle's response, ctxt, the adversary $\mathcal{B}$ sets $(\mathsf{ccm}, \mathsf{ctxt})$ as the message from the minter to the Bank, where ccm is computed as in (14). Then, $\mathcal{B}$ proceeds with the rest of the emulation as in $\mathcal{H}_{\ell+1}$. Again, decryption queries are needed in order to emulate the simulator's behavior for a payment operation by a corrupted client. We claim that $\mathcal{B}$ wins with advantage δ. Indeed, we never extract the keys of sn_0 and sn, and, by design, we never issue the challenge ctxt as a decryption query during the second phase. (Since the simulator never decrypts a ciphertext that has already appeared on the ledger.) We conclude that $\delta \leq \mu_{\mathrm{IBE}}$.

Next, consider the sub-hybrid $\mathcal{H}'_{\ell+1}$ which is identical to (15) except that $\mathsf{ctxt} \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}, (v, r', \mathsf{sn}))$ where r' is a fresh randomizer that is chosen *independently* of the randomizer r. By repeating the previous argument (this time with $m_0 = (v, r, \mathsf{sn})$) we conclude that $\delta' = |\Pr[\mathcal{H}_{\ell+1} = 1] - \Pr[\mathcal{H}'_{\ell+1} = 1]|$ is upper bounded by μ_{IBE} as well.

Finally, it is left to show that $\mathcal{H}'_{\ell+1}$ and $\mathcal{H}'_\ell$ are indistinguishable. Since the difference boils down to distinguishing between

$$\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_0, \mathsf{sn}, v_0, 0); r) \quad \text{and} \quad \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}, v, 0); r),$$

where r is a fresh randomizer that is not used anywhere else, this statement can be reduced to the hiding property of the commitment in a straightforward way. It follows that in this case Δ_{ℓ, E_m} is at most

$$2\mu_{\mathrm{IBE}} + \mu_{\mathrm{COM}} \leq 3\mu,$$

which completes the proof of Lemma 6.1 for the case where the $(\ell+1)$th command is a minting command. □

6.2 Payment by an honest client

Consider the event E_h (for honest) that at the $(\ell+1)$th round the environment sends the input

$$(\mathsf{pay}, t_1, t_2, \mathsf{pid}_B, v_B, \mathsf{pid}_C, v_C)$$

to some honest client $\mathsf{Sender} \notin M$. We show that $\Delta_{\ell, E_h} = O((\ell+1)\mu)$.

From now on, we refer to the $(\ell+1)$th command as the *current* command. We begin by showing that the public output of the current command (hereafter referred to as the *current public output*) as computed in $\mathcal{H}_\ell$ (by the ideal functionality) is indistinguishable from the output that is generated in $\mathcal{H}_{\ell+1}$ (as computed by the real protocol). Recall that the output is either an error message err or a payment notification $\mathtt{paid}$. It will be convenient to treat the latter case as false error message. Denote by err_ℓ and by $\mathsf{err}_{\ell+1}$ the error message that is computed in $\mathcal{H}_\ell$ and $\mathcal{H}_{\ell+1}$, respectively. We prove the following claim.

Claim 6.1 (public error message)**.** $\Pr[\mathsf{err}_\ell \neq \mathsf{err}_{\ell+1} \wedge E_h] \leq O((\ell+1)\mu)$.

Proof of Claim 6.1. We prove a stronger statement: Except with negligible probability, the error flags $(\mathsf{err\text{-}in}_1, \mathsf{err\text{-}in}_2, \mathsf{err\text{-}val}_1, \mathsf{err\text{-}val}_2, \mathsf{err\text{-}sum})$ as defined in $\mathcal{H}_\ell$ agree with the error flags in $\mathcal{H}_{\ell+1}$. (The latter flags are defined by taking the disjunction of the local flags that are computed by the honest payer and the flags computed by the Bank.)

$\mathsf{err\text{-}val}_j$. If $v_j \notin V$ then both in $\mathcal{H}_\ell$ (where the current command is handled as in the ideal protocol) and in $\mathcal{H}_{\ell+1}$ (where the current command is handled as in the real protocol) the flag $\mathsf{err\text{-}val}_j$ is being raised (by the functionality or by the honest payer). If $v_j \in V$ then in $\mathcal{H}_\ell$ the flag $\mathsf{err\text{-}val}_j = \mathsf{false}$, and in $\mathcal{H}_{\ell+1}$ the payer does not raise the flag $\mathsf{err\text{-}val}_j$. Moreover, the Bank raises this flag only if the range proof or the knowledge-of-committed-identity proof are being rejected. Since both proofs are computed honestly for valid statements, perfect completeness guarantees that such an event never happens.

$\mathsf{err\text{-}in}_i$. We move on and analyze the flag $\mathsf{err\text{-}in}_i$ for $i \in \{1,2\}$. When the current command is handled as in the real protocol (in $\mathcal{H}_{\ell+1}$), the flag $\mathsf{err\text{-}in}_i$ is false if and only if an entry of the form $(\mathsf{ccm}_i, \sigma_i, \mathsf{sn}_i, r_i, \mathsf{val}_i, t_i)$ appears in the Sender's private list of coins $\mathcal{L}$. By the definition of the protocol, this means that (1) In the iteration $k(t_i) < \ell+1$ in which the t_ith coin was generated, a minting/payment command whose corresponding payee is Sender was processed by the Bank and was claimed successfully by Sender; and (2) the honest Sender did not issue another payment operation with incoming coin t_i during any of the subsequent iterations in the period $(k, \ell+1)$.

When the current command is handled by the ideal protocol (in $\mathcal{H}_\ell$), the flag $\mathsf{err\text{-}in}_i$ is false iff the state of $\mathcal{F}_{\mathsf{utt}}$ after the first ℓ iterations satisfies (*) $\mathsf{owner}(t_i) = \mathsf{Sender}$. We show that, except with a negligible error probability of $O(\ell\mu)$, (*) happens iff conditions (1) and (2) hold.

For $\ell > 0$, this equivalence follows from the induction hypothesis. Specifically, let us bound the probability of the event that (1) and (2) hold but (*) does not hold. (The other direction is handled similarly.) By the correctness of Lemma 6.1 for $\ell' = k(t_i) - 1$ (which is smaller than ℓ), it follows that, except with probability $O((\ell'+1)\mu)$, the coin that was generated in the $k(t_i)$th iteration and was received by the honest Sender in the real execution was also received by the same honest party in the ideal execution. (Since the output of Sender in this iteration was delivered to the environment.) Now let us condition on the event that at end of the $k(t_i)$th iteration the state of $\mathcal{F}_{\mathsf{utt}}$ satisfied (*). If (*) does not hold at the current iteration, there exist an iteration $k' \in (k(t_i), \ell+1)$ in which Sender issued a payment command with incoming coin t_i to $\mathcal{F}_{\mathsf{utt}}$. Since this command is also processed in the real execution, this contradicts (2).

When $\ell = 0$, the equivalence holds since in both cases (the ideal and real executions) the error flag will always be raised when a payment command is issued as the first command. (Recall that after initialization, the Sender's list of coins and the state of $\mathcal{F}_{\mathsf{utt}}$ are both empty.)

$\mathsf{err\text{-}sum}$. Finally, we move on to the flag $\mathsf{err\text{-}sum}$. Recall that both in the real and ideal executions if any of the flags $\mathsf{err\text{-}in}_1$ or $\mathsf{err\text{-}in}_2$ are true, then $\mathsf{err\text{-}sum}$ is also taken to be true. Hence, we may condition on the event that $\mathsf{err\text{-}in}_1 = \mathsf{err\text{-}in}_2 = \mathsf{false}$ in both executions. Let us further condition on the event that, for $i \in \{1,2\}$, the value val_i defined above equals to the value $\mathsf{val}(t_i)$ as recorded in the state of $\mathcal{F}_{\mathsf{utt}}$ before the current command. Observe that the above argument shows that this is the case, except with probability $O(\ell\mu)$. In the ideal execution, the flag $\mathsf{err\text{-}sum}$ is being raised iff $\mathsf{val}_1 + \mathsf{val}_2 \neq v_B + v_C$, which is also the case in the real execution (due to the perfect completeness of the proof system). This completes the proof of the claim. □

From now on we condition on the event that $\mathsf{err}_\ell = \mathsf{err}_{\ell+1}$. Recall that if these flags are true, all the honest parties output the error message. Moreover, the proof of Claim 6.1 shows that in this case, except with negligible probability, in $\mathcal{H}_{\ell+1}$ the message sent from Sender to the bank is an error message. Thus, in this case the view of the corrupted parties consists of an error message which is perfectly simulated by the simulator. We conclude that the lemma holds in this case.

From now on, we condition on the event that no error flags are being raised both in the ideal and in the real executions. Under this assumption, let us record the values $(\mathsf{ccm}_i, \sigma_i, \mathsf{sn}_i, r_i, \mathsf{val}_i, t_i), \forall i \in \{1,2\}$ as defined in the proof of the above claim. By Observation 6.1, we may further assume that $\mathsf{T} = \mathsf{T}'$ where T and T' denote the number of coins generated during the first ℓ iterations in the ideal execution and real execution, respectively.

The private output of an honest payee. For $j = B$, consider the case that the client $\mathsf{C}_{\mathsf{pid}_j}$ is honest. (The case of $j = C$ is proved analogously.) In the ideal execution ($\mathcal{H}_\ell$) the current private output of $\mathsf{C}_{\mathsf{pid}_B}$ is $(\mathtt{paid}, \mathsf{T}+1, v_B)$. We claim that the output in the real execution ($\mathcal{H}_{\ell+1}$) is identical. Indeed, since the honest payer generates an honest transaction, this transaction is being signed by the bank, added to the ledger, and successfully claimed by the honest payee. It follows that the output of the payee is $(\mathtt{paid}, \mathsf{T}'+1, v_B)$. Since $\mathsf{T} = \mathsf{T}'$, the claim follows.

The view of the corrupted parties. We show that the view of the corrupted parties in $\mathcal{H}_{\ell+1}$ is $O(\mu)$-computationally indistinguishable from the simulated view of the corrupted parties in $\mathcal{H}_\ell$. We begin by showing that the message sent by Sender to the Bank in $\mathcal{H}_\ell$ is indistinguishable from the corresponding message in $\mathcal{H}_{\ell+1}$. In both cases, this message is of the form

$$((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),\ (\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}},\ \Pi_{\mathsf{Sum}}$$

where in the real protocol it is distributed as in (8), and in the ideal protocol it is distributed as in (11). We show that the difference is indistinguishable even for an adversary who holds the Bank's private signing key.We gradually move from $\mathcal{H}_\ell$ to $\mathcal{H}_{\ell+1}$ via the following sequence of hybrids.

The hybrid $\mathcal{H}'_0$ is defined just like the simulated distribution $\mathcal{H}_\ell$ except that all the zero-knowledge proofs are generated by using the simulators. That is, for $i \in \{1,2\}$, the split proofs Π_{Split_i} is generated by running the ZK-Simulator over the $\mathcal{R}_{\mathsf{split}}$ statement $(\mathsf{ccm}_i, \mathsf{vcm}_i, \mathsf{rcm}', \mathsf{nullif}_i)$. The sum-proof Π_{Sum} is generated by applying the ZK-simulator over the $\mathcal{R}_{\mathsf{sum}}$ statement $(\mathsf{vcm}_1, \mathsf{vcm}_2, \mathsf{vcm}_B, \mathsf{vcm}_C)$, and for $j \in \{B, C\}$, the proofs $\Pi_{\mathsf{icm}\,j}$ and Π_{Range_j} are sampled by applying the corresponding simulators on icm_j and on vcm_j. Since in $\mathcal{H}_\ell$ these proofs are generated honestly, $\mathcal{H}'_0$ is $7\mu_{\mathrm{ZK}}$-indistinguishable from $\mathcal{H}_\ell$ where μ_{ZK} upper-bounds the probability that an efficient adversary breaks the zero-knowledge property of the simulator.

The hybrid $\mathcal{H}'_1$ is defined just like in $\mathcal{H}'_0$ except that the commitments $\mathsf{ccm}_i, i \in \{1,2\}$ are computed like in the real execution, i.e.,

$$\mathsf{ccm}_i = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}_i, \mathsf{val}_i, 0); r_i)$$

where pid is the Sender's public identifier and r_i is a fresh randomizer.[20] By the hiding property of the commitment, $\mathcal{H}'_1$ is $2\mu_{\mathrm{COM}}$-indistinguishable from $\mathcal{H}_\ell$ where μ_{COM} upper-bounds the probability that an efficient adversary breaks the hiding property of the commitment.

[20] We further assume that all the values that are computed based on the commitments is computed based on the new commitments. (This convention applies to all the subsequent hybrids.) Specifically, in $\mathcal{H}'_1$, the simulators of the zero-knowledge proofs are applied to the new commitments, and the signature σ_i is computed by generating a fresh signature over ccm_i with the Bank's signing key bsk.

The hybrid $\mathcal{H}'_2$ is defined just like $\mathcal{H}'_1$ except that the commitments $\mathsf{vcm}_i, i \in \{1,2\}$ are computed like in the real execution, i.e.,

$$\mathsf{vcm}_i = \mathsf{CM.Commit}(\mathsf{ck}, (0,0,\mathsf{val}_i, 0); z_i),$$

where z_i is fresh randomizer. Again, by the hiding property of the commitment, $\mathcal{H}'_2$ is $2\mu_{\mathrm{COM}}$-indistinguishable from $\mathcal{H}'_1$.

The hybrid $\mathcal{H}'_3$ is defined just like $\mathcal{H}'_2$ except that the registration commitment rcm' by

$$\mathsf{rcm}' = (\mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a)$$

where a is a fresh randomizer and s is the PRF key of the Sender. (Correspondingly, rs' is computed by generating a fresh signature over rcm' with the Bank's registration signing key rsk.) Again, by the hiding property of the commitment, $\mathcal{H}'_3$ is μ_{COM}-indistinguishable from $\mathcal{H}'_2$.

The hybrid $\mathcal{H}'_4$ is defined just like $\mathcal{H}'_3$ except that the registration commitment rcm' and its signature rs', are computed just like in the real execution, i.e., by refreshing the registration entry $(\mathsf{rcm}, \mathsf{rs})$ of Sender. By the rerandomization property of the rerandomizable signatures over commitments, $\mathcal{H}'_4$ is identically distributed to $\mathcal{H}'_3$.

The hybrid $\mathcal{H}'_5$ is defined just like $\mathcal{H}'_4$ except that we sample the nullifiers $\mathsf{nullif}_i, i \in \{1,2\}$ uniformly at random from the range of the PRF. Recall that in the previous hybrids, it holds that $\mathsf{nullif}_i = \mathsf{PRF}_{\mathsf{s}_{\mathsf{Sender}_0}}(\mathsf{sn}'_i)$ where sn'_i is uniformly distributed and Sender_0 is a fresh random PRF key. Therefore, a distinguisher between $\mathcal{H}'_5$ and $\mathcal{H}'_4$ can be easily translated into a distinguisher that breaks the pseudorandomness of $\mathsf{PRF}_{\mathsf{s}_{\mathsf{Sender}_0}}(\cdot)$ (crucially all other values in $\mathcal{H}'_5$ and $\mathcal{H}'_4$ do not depend on $\mathsf{s}_{\mathsf{Sender}_0}$). Overall, $\mathcal{H}'_5$ is μ_{PRF}-indistinguishable from $\mathcal{H}'_4$, where μ_{PRF} upper-bounds the probability that an efficient adversary breaks the pseudorandom function.

The hybrid $\mathcal{H}'_6$ is defined just like $\mathcal{H}'_5$ except that the nullifiers $\mathsf{nullif}_i, i \in \{1,2\}$ are computed just like in the real execution, i.e.,

$$\mathsf{nullif}_i = \mathsf{PRF}_{\mathsf{s}_{\mathsf{Sender}}}(\mathsf{sn}_i),$$

where $\mathsf{s}_{\mathsf{Sender}}$ is the PRF key of the Sender. By the pseudorandomness of the PRF, $\mathcal{H}'_6$ is μ_{PRF}-indistinguishable from $\mathcal{H}'_5$. (Crucially all other values in $\mathcal{H}'_6$ and $\mathcal{H}'_5$ can be sampled given an oracle access to $\mathsf{s}_{\mathsf{Sender}_0}$.)

The hybrid $\mathcal{H}'_7$ is defined just like $\mathcal{H}'_6$ except that, for $j \in \{B, C\}$, the commitment. icm_j, to the identifier of the jth payee and the commitment vcm_j to the corresponding paid value, are computed just like in the real execution, i.e.,

$$\mathsf{icm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_j, 0, 0, 0); t_j) \quad \text{and} \quad \mathsf{vcm}_j = \mathsf{CM.Commit}(\mathsf{ck}, (0,0,v_j,0); \rho_j), \tag{16}$$

where t_B, t_C and ρ_B, ρ_C are fresh randomizers. Recall that if pid_j is corrupted then this new value of $(\mathsf{icm}_j, \mathsf{vcm}_j)$ is distributed identically to the corresponding entry in $\mathcal{H}'_6$ (conditioned on all other values). If this is not the case, then these values are $2\mu_{\mathrm{COM}}$-indistinguishable. It follows that $\mathcal{H}'_7$ is $4\mu_{\mathrm{COM}}$-indistinguishable from $\mathcal{H}'_6$.

The hybrid $\mathcal{H}'_8$ is defined just like $\mathcal{H}'_6$ except that, for $j \in \{B, C\}$, we compute the ciphertext ctxt_j just like in the real execution, i.e.,

$$\mathsf{ctxt}_j \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}_j, (v_j, \rho_j + t_j)).$$

Recall that if pid_j is corrupted then ctxt_j is distributed identically to the corresponding entry in $\mathcal{H}'_7$ (conditioned on all other values). If this is not the case, then in $\mathcal{H}'_7$, it holds that $\mathsf{ctxt}_j \xleftarrow{\$} \mathsf{IBE.Enc}(\mathsf{pid}'_j, (v'_j, \rho_j + t_j))$

where $v'_j = 1$ and pid'_j is an identifier of some arbitrary honest user. It follows that these two versions of ctxt_j are μ_{IBE}-indistinguishable. Indeed, a distinguisher can be translated into an adversary $\mathcal{B}$ that wins the IND-RA-CCA game with similar advantage. The reduction is similar to the one that is explained in Section 6.1. Overall, $\mathcal{H}'_8$ is $2\mu_{\mathrm{IBE}}$-indistinguishable from $\mathcal{H}'_7$.

The hybrid $\mathcal{H}'_9$ is similar to $\mathcal{H}'_8$ except that all the zero-knowledge proofs, $\Pi_{\mathsf{Split}_1}, \Pi_{\mathsf{Split}_2}, \Pi_{\mathsf{icm}\,B}, \Pi_{\mathsf{Range}_B}$, $\Pi_{\mathsf{icm}\,C}, \Pi_{\mathsf{Range}_C}$ and Π_{Sum}, are generated just like in the real execution by running the honest prover's algorithms on the corresponding statements and their witnesses. The zero-knowledge property implies that this hybrid is $7\mu_{\mathrm{ZK}}$-indistinguishable from $\mathcal{H}'_9$. Moreover, the hybrid $\mathcal{H}'_9$ is identically distributed to $\mathcal{H}_{\ell+1}$. Overall, we conclude that $\mathcal{H}_{\ell+1}$ cannot be efficiently distinguished from $\mathcal{H}_\ell$ with advantage better than

$$O(\mu_{\mathrm{ZK}} + \mu_{\mathrm{COM}} + \mu_{\mathrm{PRF}} + \mu_{\mathrm{IBE}}) = O(\mu).$$

The Bank's response. Finally, we mention that, in addition to the Sender's message to the the Bank, the adversary also sees the Bank's response (appended to the ledger) but this response can be computed efficiently based on the Sender's message and given an oracle access to the Bank's signing algorithm. Since (8) and (11) remain indistinguishable even given the signature's key, the entire view of the adversary remains indistinguishable. This completes the proof of Lemma 6.1 for the case where the $(\ell+1)$th command is an honest payment command. □

6.3 Payment by a corrupted client

Denote by E_c (c for "corrupt") the event that at the $(\ell+1)$th round the environment sends some instruction χ to the adversary who responds by sending, on behalf of some corrupted client, the following message to the Bank

$$((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),\ (\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}},\ \Pi_{\mathsf{Sum}}. \tag{17}$$

We show that Δ_{ℓ,E_c} is upper-bounded by $O(\ell\mu)$. As in Section 6.2, we refer to the $(\ell+1)$th command as the *current* command. We begin by showing that the public error message, err_ℓ, of the current command as computed in $\mathcal{H}_\ell$ (by the ideal functionality) is computationally indistinguishable from the public error message, $\mathsf{err}_{\ell+1}$, that is generated in $\mathcal{H}_{\ell+1}$ (by the real protocol). Specifically, we prove the following claims.

Claim 6.2. *Conditioned on E_c, if any of the error flags in $\mathcal{H}_{\ell+1}$ is set to* true *then the corresponding flag in $\mathcal{H}_\ell$ is also set to* true*.*

Proof. By design of the simulator, if the Bank raises a flag in response to the message (17), then the simulator sends to $\mathcal{F}_{\mathsf{utt}}$ a command that issues the same error. □

The other (less trivial) direction follows from the next lemma that makes use of the following notation.

Notation 6.1. *For every iteration $k \le \ell+1$ denote by tx_k the message sent to the Bank and let P_k denote the party that issues this command. If the kth command is a corrupted payment command that is accepted by the Bank, we define the values*

$$(\mathsf{s}_{k,i}, \mathsf{pid}_{k,i}, \mathsf{sn}_{k,i}, \mathsf{val}_{k,i}, r_{k,i}, z_{k,i}, a'_{k,i})$$

to be the witness that is extracted by the knowledge extractor from the ith split proof $\Pi_{\mathsf{Split}_{k,i}}$ that appears in tx_k. Similarly, for $j \in \{B, C\}$, let

$$(\mathsf{pid}_{k,j}, \mathsf{sn}_{k,j}, v_{k,j}, \rho_{k,j}, t_{k,j})$$

be the witnesses extracted by the knowledge-extractor from the proofs $\Pi_{\mathsf{Range}_{k,j}}$ and $\Pi_{\mathsf{icm}\,k,j}$ that that appears in tx_k. Let $r_{k,j} = \rho_{k,j} + t_{k,j}$. (If some of the extractions fail to find a value that satisfy the corresponding relation, set the corresponding values to $\perp$.)

If the kth command is an honest payment command then all the above values are defined by taking the corresponding values as computed by the corresponding honest party.

If the kth command is a minting command with $\mathsf{tx}_k = (\mathsf{sn}_{k,B}, \mathsf{ccm}_{k,B}, \mathsf{ctxt}_{k,B})$ then let $(\mathsf{pid}_{k,B}, \mathsf{sn}_{k,B}, v_{k,B}, r_{k,B})$ denote the values for which $\mathsf{ccm}_{k,B} = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_{k,B}, \mathsf{sn}_{k,B}, v_{k,B}, 0); r_{k,B})$. Observe that these values are well defined since we explicitly compute them during the experiment when we emulate the minter.[21]

We prove the following key lemma in Section 6.4.

Lemma 6.2 (key lemma)**.** Except with probability of $O((\ell+1)\mu)$, the event G (for "good") holds, where G asserts that for every iteration $k \leq \ell+1$ in which a payment command is processed successfully the followings hold.

1. For $i \in \{1,2\}$, $\mathsf{s}_{k,i}$ is the PRF key associated with $\mathsf{pid}_{k,i}$ (i.e., appears as the first entry in $\mathsf{ask}_{\mathsf{pid}}$ as defined by the simulator) and $(\mathsf{pid}_{k,1}, \mathsf{s}_{k,1}) = (\mathsf{pid}_{k,2}, \mathsf{s}_{k,2})$.

2. For $i \in \{1,2\}$, there exists an iteration $k'(k,i) < k$ and an index $j(k,i) \in \{B,C\}$ such that the ith incoming coin in the (successful) kth transaction is consistent with the jth outgoing coin in k'th transaction, formally,
$$(\mathsf{pid}_{k,i}, \mathsf{sn}_{k,i}, \mathsf{val}_{k,i}) = (\mathsf{pid}_{k',j}, \mathsf{sn}_{k',j}, v_{k',j}),$$
where $k' = k'(k,i)$ and $j = j(k,i)$.

3. The simulator's kth command to the ideal functionality $\mathtt{cmd}_k$ is a payment command that is sent on behalf of the client whose identifier is $\mathsf{pid}_{k,1}$ and the addresses of the incoming coins are $\mathsf{T}_{k,1} := \mathsf{T}[k'(k,1), j(k,1)]$ and $\mathsf{T}_{2,k} := \mathsf{T}[k'(k,2), j(k,2)]$ where $\mathsf{T}[x,y]$ denotes the number of coins generated by the ideal functionality until the generation of the coin y in iteration x.

4. In $\mathtt{cmd}_k$ the payees and the paid values are $(\mathsf{pid}'_{k,j}, v_{k,j})_{j\in\{B,C\}}$ where $\mathsf{pid}'_{k,j} = \mathsf{pid}_{k,j}$, except for the case where the simulator modifies this value during Step 4 which may happen only if the kth operation is performed by a corrupted payer and $\mathsf{pid}_{k,j}$ is an honest party.

5. (a) Before the kth iteration, the state of the ideal functionality satisfies $\mathsf{coins}[\mathsf{T}_{k,1}] = (\mathsf{val}_{k,1}, \mathsf{pid}_{k,1})$ and $\mathsf{coins}[\mathsf{T}_{k,2}] = (\mathsf{val}_{k,2}, \mathsf{pid}_{k,1})$. (b) Moreover, at the end of the kth iteration, the ideal functionality outputs $\mathtt{paid}$, and its state satisfies $\mathsf{coins}[\mathsf{T}_k + 1] = (v_{k,B}, \mathsf{pid}'_{k,B})$ and $\mathsf{coins}[\mathsf{T}_k + 2] = (v_{k,2}, \mathsf{pid}'_{k,C})$ where T_k is the number of coins that were generated till the kth iteration.

The last item (applied to $k = \ell+1$) implies that if the current transaction is approved by the Bank as a successful transaction, then, except with negligible probability, it is also accepted by the ideal functionality. By combining this with Claim 6.2, we conclude that $\Pr[\mathsf{err}_\ell \neq \mathsf{err}_{\ell+1} \wedge E_c] \leq O((\ell+1)\mu)$. From now on, let us condition on the event G (and therefore $\mathsf{err}_\ell = \mathsf{err}_{\ell+1}$) and on event that $\mathsf{T} = \mathsf{T}'$, as defined in Observation 6.1.

The private output of an honest payee. If an error flag is issued then all honest parties output an error both in $\mathcal{H}_{\ell+1}$ and in $\mathcal{H}_\ell$. We can therefore focus on the event that the current command does not issue an error flag. Fix some honest client $\mathsf{C}_{\mathsf{pid}}$. We analyze the output of an honest client $\mathsf{C}_{\mathsf{pid}}$ with respect to the first outgoing coin. (The case of the second coin is similar.) It suffices to show that, except with negligible probability, $\mathsf{C}_{\mathsf{pid}}$ outputs a message of the form $(\mathtt{paid}, \mathsf{T}+1, v)$ in $\mathcal{H}_{\ell+1}$ if and only if it outputs the same message in $\mathcal{H}_\ell$.

We begin with the "only if direction". Assume that $\mathsf{C}_{\mathsf{pid}}$ successfully claims the first outgoing coin and outputs $(\mathtt{paid}, \mathsf{T}+1, v'_j)$. (The case of the second coin is similar.) This means that client recovers from the Bank's outputs $(\mathsf{ccm}_B, \mathsf{ctxt}_B)$ the values $(\mathsf{pid}, \mathsf{sn}, v, 0); r$ for which
$$\mathsf{ccm}_B = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}, v, 0); r), \qquad \text{where } \mathsf{sn} = H(\mathsf{nullif}_1, \mathsf{nullif}_2, j).$$

[21] the subscript "B" in a minting command is redundant (since it generates a single coin) and we add it in order to unify the treatment of minting and payment.

By the construction of ccm_B, it also holds that

$$\mathsf{ccm}_B = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_{\ell+1,B}, \mathsf{sn}_{\ell+1,B}, v_{\ell+1,B}, 0); r_{\ell+1,B}).$$

We conclude that

$$(\mathsf{pid}, \mathsf{sn}, v) = (\mathsf{pid}_{\ell+1,B}, \mathsf{sn}_{\ell+1,B}, v_{\ell+1,B}), \tag{18}$$

unless we can break the binding property of the commitment which can happen with probability at most $\mu_{\text{BCOM}} \leq \mu$. Conditioned on (18), the first payee/paid-value in the simulator's command to the ideal functionality is $(\mathsf{pid}_{\ell+1,B}, v_{\ell+1,B}) = (\mathsf{pid}, \mathsf{sn})$. This follows from G and from the fact that the simulator keeps the payee unchanged in Step 4 (since the "Claim" operation on behalf of $\mathsf{C}_{\mathsf{pid}_{\ell+1,B}} = \mathsf{C}_{\mathsf{pid}}$ succeeds). We conclude that in $\mathcal{H}_\ell$ the current output of the honest party $\mathsf{C}_{\mathsf{pid}}$ is also $(\mathtt{paid}, \mathsf{T}+1, v'_j)$, as required.

We move on to prove the other direction. Suppose that $\mathsf{C}_{\mathsf{pid}}$ receives from the ideal functionality the message $(\mathtt{paid}, \mathsf{T}+1, v)$. Let $\mathsf{pid}'_{\ell+1,1}$ denote the identifier of the first payee in the simulator's submitted command. By the definition of the simulator, $\mathsf{pid}'_{\ell+1,1}$ has not been changed in Step 4. (Since it belongs to an existing honest client.) By relying on G with $\mathsf{pid}'_{\ell+1,1} = \mathsf{pid}_{\ell+1,1}$, we conclude that (18) holds. Consequently, when $\mathsf{C}_{\mathsf{pid}}$ applies the claim operation to the message sent by the Bank in $\mathcal{H}_{\ell+1}$, the operation succeeds and the output is $(\mathtt{paid}, \mathsf{T}+1, v)$, as required.

The view of the corrupted parties. The view of the corrupted parties consists of the Bank's message which is identically distributed in both experiments. This completes the proof of Lemma 6.1 for the case where the $(\ell+1)$th command is a corrupted payment command. □

6.4 Proof of Lemma 6.2

Throughout the proof, we condition on the event that during the first $\ell+1$ iteration, whenever a zero-knowledge proof passes verification, the corresponding knowledge extractor extracts witnesses. This event happens, except with probability $(\ell+1)\mu_{\text{KE}} \leq O((\ell+1)\mu)$

Item 1. We show that if (1) does not hold then there exists an adversary $\mathcal{B}$ that either breaks the EU-CCA security of the registration signature scheme or breaks the binding property of the commitment. The adversary $\mathcal{B}$ is given $\mathsf{pk} = (\mathsf{pp}, \mathsf{vk}, \mathsf{ck})$ sampled using the global setup algorithm, commitment set-up algorithm and the signature key-generation algorithm (as defined in Definition 3), and places these values as the initialization values in the hybrid $\mathcal{H}_\ell$ where vk takes the role of the Bank's registration verification key rvk. The other initialization values (e.g., the IBE msk and the Bank's signature keys $(\mathsf{bsk}, \mathsf{bvk})$) are sampled locally based on pp and ck. The adversary can now emulate the initialization phase with the aid of the commitment-signing oracle that given a message m and a randomizer r, signs the commitment $\mathsf{CM.Commit}(\mathsf{ck}, m; r)$ under the key rsk. Indeed, whenever a client registers with a message $(\mathsf{pid}, \mathsf{rcm}, \Pi_{\mathsf{Init}})$ the adversary extracts from Π_{Init} the values $(\mathsf{pid}, \mathsf{s}, a)$ for which $\mathsf{rcm} = \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a)$, queries the oracle with $m = (\mathsf{pid}, 0, 0, \mathsf{s})$ and a, and receives back the signature $\mathsf{rs} \xleftarrow{\$} \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm})$. Next, $\mathcal{B}$ perfectly emulates the first $\ell+1$ iterations of hybrid $\mathcal{H}_\ell$, and in each iteration $k \leq \ell+1$ in which a successful payment command is performed, the adversary $\mathcal{B}$ computes the values $(\mathsf{s}_{k,i}, \mathsf{pid}_{k,i}, \mathsf{sn}_{k,i}, \mathsf{val}_{k,i}, r_{k,i}, z_{k,i}, a'_{k,i})$ for $i \in \{1,2\}$ (either directly by the honest payer or via the knowledge extractor). If (a) $\mathsf{s}_{k,i}$ is not the PRF key associated with $\mathsf{pid}_{k,i}$, the adversary $\mathcal{B}$ outputs the forgery $(m^*, a'_{k,i}, \xleftarrow{\$}'_k)$ where

$$m^* = (\mathsf{pid}_{k,i}, 0, 0, \mathsf{s}_{k,i}).$$

Note that in this case, this is indeed a valid forgery since the message m^* was not queried before (at the registration phase) and $\xleftarrow{\$}'_k$ is a valid rsk-signature over $\mathsf{CM.Commit}(\mathsf{ck}, m^*; a'_{k,i})$. If (a) does not hold, but $(\mathsf{pid}_{k,1}, \mathsf{s}_{k,1}) \neq (\mathsf{pid}_{k,2}, \mathsf{s}_{k,2})$ the adversary outputs the commitment collision

$$[m_1 = (\mathsf{pid}_{k,1}, 0, 0, \mathsf{s}_{k,1}), \quad a'_{k,1})] \quad \neq \quad [m_2 = (\mathsf{pid}_{k,2}, 0, 0, \mathsf{s}_{k,2}), \quad a'_{k,2}],$$

for which $\mathsf{CM.Commit}(\mathsf{ck}, m_1; a'_{k,1}) = \mathsf{CM.Commit}(\mathsf{ck}, m_2; a'_{k,2})$. Assuming that the knowledge extractor does not fail in the first $\ell + 1$ iterations, if (1) does not hold then either (a) or (b) happen. Therefore, (1) holds except with probability $\mu_{\mathrm{SIG}} + \mu_{\mathrm{BCOM}} \leq O(\mu)$, where μ_{BCOM} (resp., μ_{SIG}) denote an upper-bounds on the probability of breaking the binding of the commitment (resp., breaking the EU-CCA security of the signature scheme).

Item 2. Next, we show that if (2) does not hold then there exists an adversary $\mathcal{B}$ that breaks the EU-CCA security of the Bank's signature scheme. The forger $\mathcal{B}$ is given $\mathsf{pk} = (\mathsf{pp}, \mathsf{vk}, \mathsf{ck})$ sampled using the global setup algorithm, commitment set-up algorithm and the signature key-generation algorithm (as defined in Definition 3), and places these values as the initialization values in the hybrid $\mathcal{H}_\ell$ where vk takes the role of the Bank's verification key bvk. The other initialization values (e.g., the IBE msk and the registration keys $(\mathsf{rsk}, \mathsf{rvk})$) are sampled locally based on pp and ck. The adversary can now emulate the first ℓ steps of the hybrid $\mathcal{H}_\ell$ with the aid of a commitment-signing oracle that given a message m and a randomizer r, signs the commitment $\mathsf{CM.Commit}(\mathsf{ck}, m; r)$ under the key bsk. Specifically, for every iteration $k \leq \ell$ that is approved, the adversary $\mathcal{B}$ computes all the transaction entries as defined in Notation 6.1. If these entries satisfy condition (2), then the adversary $\mathcal{B}$ generates the Bank's response where the signature on

$$\mathsf{ccm}_{k,j} = \mathsf{icm}_{k,j} \boxplus \mathsf{CM.Commit}(\mathsf{ck}, (0, \mathsf{sn}_{k,j}, 0, 0); 0) \boxplus \mathsf{vcm}_{k,j}, \qquad j \in \{B, C\}$$

is computed by making a call to the signing oracle with $m_{k,j} := (\mathsf{pid}_{k,j}, \mathsf{sn}_{k,j}, \mathsf{val}_{k,j}, 0)$ and $r_{k,j} = \rho_{k,j} + t_{k,j}$. If the kth transaction does not satisfy (2) with respect to the ith incoming coin for some $i \in \{1, 2\}$, then $\mathcal{B}$ outputs the forgery $(m_{k,i}, r_{k,i}, \sigma_i)$ where

$$m_{k,i} = (\mathsf{pid}_{k,i}, \mathsf{sn}_{k,i}, \mathsf{val}_{k,i}, 0).$$

Finally, if we reached to the $(\ell+1)$th iteration and the transaction satisfies the split relations but (2) does hold with respect to the ith incoming coin for some $i \in \{1, 2\}$, then $\mathcal{B}$ outputs the forgery $(m^*_{\ell+1,i}, r_{\ell+1,i}, \sigma_{\ell+1})$. Otherwise, $\mathcal{B}$ terminates with failure.

To analyze $\mathcal{B}$, first observe that assuming that the knowledge-extractor does not fail, as long as $\mathcal{B}$ does not halt, it perfectly emulates the hybrid $\mathcal{H}_\ell$. Moreover, if (2) does not happen, then $\mathcal{B}$ terminates with a forgery $m_{k,i}$ that, by definition, differs from all the previous queries $\{m_{k',j} : k' < k, j \in \{B, C\}\}$. It follows that in this case $\mathcal{B}$ wins the EU-CCA game, which means that (2) fails with probability of at most $\mu_{\mathrm{SIG}} \leq \mu$.

Item 3. Let us assume that (1) and (2) hold, and upper-bound the probability that in some iteration k item (3) fails to hold. By the definition of the simulator, it must be the case that in the kth iteration, a corrupted payment command is issued. Recall that in this case the simulator searches the ledger for transactions $q_1, q_2 < k$, indices $g_1, g_2 \in \{B, C\}$ and PRF keys $\mathsf{PRF}_1, \mathsf{PRF}_2$ associated with some public identifiers $\mathsf{pid}_1, \mathsf{pid}_2$ such that, for $i \in \{1, 2\}$ the ith nullifier of the kth transaction, $\mathsf{nullif}_{k,i}$, equals to $\mathsf{PRF}_{\mathsf{s}_i}(\mathsf{sn}_i)$ where sn_i is the serial number associated with the g_ith outgoing entry of the q_ith transaction. Since (1) and (2) hold, we conclude that the simulator finds such entries. Recall that if $\mathsf{pid}_1 = \mathsf{pid}_2$, the simulator submits to $\mathcal{F}_{\mathsf{utt}}$ a payment command on behalf of pid_1 whose incoming coin identifiers are $\mathsf{T}[q_1, g_1]$ and $\mathsf{T}[q_2, g_2]$. Therefore, assuming that (1) holds for this iteration, it suffices to show that for $i \in \{1, 2\}$ it holds that

$$q_i = k'(k, i) \qquad g_i = j(k, i) \tag{19}$$

and that

$$\mathsf{pid}_i = \mathsf{pid}_{k,i}. \tag{20}$$

If (19) does not hold, then $\mathsf{nullif}_{k,i}$ can be written both as $\mathsf{PRF}_{\mathsf{s}}(\mathsf{sn})$ and as $\mathsf{PRF}_{\mathsf{s}'}(\mathsf{sn}')$ where sn and sn' are obtained by hashing two *different* inputs via the random oracle (RO) H and s and s' are the PRF keys associated with pid_i and $\mathsf{pid}_{k,i}$ (which may be equal or not). We show that such an event is unlikely to happen due to the *correlation robustness* of the RO H. Indeed, assuming that $\mathsf{PRF}_{\mathsf{s}}(\cdot)$ and $\mathsf{PRF}_{\mathsf{s}'}(\cdot)$ are injective (have no self collisions), the probability of finding a pair of Hash inputs that map into different sn

and sn' that satisfy $\mathsf{PRF}_{\mathsf{s}}(\mathsf{sn}) = \mathsf{PRF}_{\mathsf{s}'}(\mathsf{sn}')$ is at most $O(Q_{\ell+1}^2/|\mathcal{X}_{\mathsf{pp}}|)$ where $\mathcal{X}_{\mathsf{pp}}$ is the domain of the PRF and $Q_{\ell+1}$ is the number of RO queries that the adversary makes during the first $\ell+1$ iterations. Taking a union-bound over all pairs of registered PRF-keys, we get an upper-bound of $O(N^2 Q_{\ell+1}^2/|\mathcal{X}_{\mathsf{pp}}|) \leq \mu$. We move on to establish (20) conditioned on (19). Suppose that (20) does not hold. Then, for a pair of registered PRF keys s and s' (associated with $\mathsf{pid}_i \neq \mathsf{pid}_{k,i}$), the adversary finds a hash input x that hashes by H to sn for which $\mathsf{PRF}_{\mathsf{s}}(\mathsf{sn}) = \mathsf{PRF}_{\mathsf{s}'}(\mathsf{sn})$. Since our collection has no pairwise collisions this can happen only if 2 of the registered PRF keys agree. Since we randomize the PRF keys on registration, this event happens except with probability $O(N^2/|\mathcal{K}_{\mathsf{pp}}|) \leq \mu$ where $\mathcal{K}_{\mathsf{pp}}$ is the key-space of the PRF.[22]

Item 4. If the kth command is an honest payment, the statement trivially holds (by the definition of the simulator). If (4) is violated in some iteration for which the payer is corrupt, we break the binding of the commitment (as indicated in the error message of the simulator). This can happen with probability of at most $\mu_{\mathrm{BCOM}} \leq \mu$.

Item 5 Let k denote the first iteration for which the statement does not hold. If k is an honest "payment iteration" the statement follows from the upper-bound on Δ_{k,E_h}. (For $k = \ell+1$ this is established in Section 6.2 and for $k < \ell+1$ this is established as part of the induction hypothesis, i.e., Lemma 6.1 for k.) We can therefore focus on the case where a corrupted payer performs the kth iteration. Specifically, let us first consider the case where (a) fails. Namely, before the ideal functionality processes the kth command, its state does not satisfy $\mathsf{coins}[\mathsf{T}_{k,1}] = (\mathsf{val}_{k,1}, \mathsf{pid}_{k,1})$ and $\mathsf{coins}[\mathsf{T}_{k,2}] = (\mathsf{val}_{k,2}, \mathsf{pid}_{k,1})$. Without loss of generality, let us assume that the first equality does not hold. (The other case is proved similarly.) Consider the iteration $k' := k'(k,1) < k$. We first claim that at the end of iteration k' it must hold that

$$\mathsf{coins}[\mathsf{T}_{k,1}] = (\mathsf{val}_{k,1}, \mathsf{pid}_{k,1}). \tag{21}$$

Indeed, by item (2), it holds that the corresponding transaction $\mathsf{tx}_{k'}$ is successful and its $j = j(k,1)$th outgoing entry satisfies

$$(\mathsf{pid}_{k,1}, \mathsf{sn}_{k,1}, \mathsf{val}_{k,1}) = (\mathsf{pid}_{k',j}, \mathsf{sn}_{k',j}, v_{k',j}).$$

We conclude that in the corresponding command of the simulator $\mathtt{cmd}_{k'}$ the jth outgoing coin has the entries $(\mathsf{val}_{k,1}, \mathsf{pid}_{k,1})$. If the iteration k' is an "honest" iteration, this follows directly from the definition of the simulator, and otherwise, this follows from item (4). Note that, by assumption, the payee $\mathsf{pid}_{k',j} = \mathsf{pid}_{k,1}$ is corrupted, and therefore the exception in (4) does not apply. Finally, by the induction hypothesis (Lemma 6.1 for k'), the corresponding command of the simulator is processed successfully by the ideal functionality. We can therefore conclude that at the end of iteration k', Eq. 21 holds. Next, assume towards a contradiction that Eq. 21 is violated in some iteration $k'' \in (k', k)$. This may happen only if, on iteration k'' the simulator issues a payment command with address $\mathsf{T}_{k,1}$ on behalf of the corrupted payer $\mathsf{pid}_{k,1}$. By the design of the simulator, this happens only of the corresponding transaction $\mathsf{tx}_{k''}$ is accepted. By items (1)–(3), it must be the case that $\mathsf{tx}_{k''}$ contains the nullifier $\mathsf{PRF}_{\mathsf{s}_{k,1}}(\mathsf{sn}_{k,1})$, and therefore kth transaction cannot be approved – a contradiction.

Item (5b) follows directly from items (5a) and (4). This completes the proof of the lemma. □

References

[1] UTT: Decentralized, anonymous and accountable digital cash infrastructure. Under (Anonymous) submission, 2021.

[22] This is the only case where we directly exploit the random oracle assumption on H. We mention that the above analysis can be extended beyond the random oracle model, by assuming that the hash function H is correlation robust with respect to a concrete efficiently commutable relation that is induced by the above proof (and depends on the underlying PRF). Detailed omitted.

[2] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582. Springer, 2001.

[3] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

[4] David Bernhard, Marc Fischlin, and Bogdan Warinschi. Adaptive proofs of knowledge in the random oracle model. *IET Inf. Secur.*, 10(6):319–331, 2016.

[5] Patrik Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Get shorty via group signatures without encryption. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 381–398. Springer, 2010.

[6] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptol.*, 21(2):149–177, 2008.

[7] Dan Boneh, Ben Fisch, Ariel Gabizon, and Zac Williamson. A Simple Range Proof From Polynomial Commitments, 2020. https://hackmd.io/@dabo/B1U4kx8XI.

[8] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[10] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings*, pages 416–431, 2005.

[11] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[12] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62. Springer, 2018.

[13] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*, volume 1560 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 1999.

[14] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[15] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discret. Appl. Math.*, 156(16):3113–3121, 2008.

[16] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[17] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. *IACR Cryptol. ePrint Arch.*, page 325, 2015.

[18] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. *IACR Cryptol. ePrint Arch.*, page 310, 2011.

[19] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM J. Comput.*, 39(5):2090–2112, 2010.

[20] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[21] David Pointcheval and Olivier Sanders. Short Randomizable Signatures. In Kazue Sako, editor, *CT-RSA 2016*, pages 111–126, Cham, 2016. Springer International Publishing.

[22] Sriramkrishnan Srinivasan. *New Security Notions for Identity Based Encryption*. Phd, Royal Holloway, University of London, 2010.

A Dual Pedersen Commitments over Bilinear Groups

For the sake of self-containment, we describe and analyze the "dual" version of Dual Pedersen Commitments. For simplicity, we focus on the version in which the message space is a vector of length 2. The proof naturally generalizes to longer vectors.

The commitment key ck (output from $\mathsf{CM.Setup}(1^\lambda)$) consist of two sub-keys, over two different groups: the first sub-key (g_1, g_2, g) is over $\mathbb{G}_1$ and the sub-key $(\tilde{g}_1, \tilde{g}_2, \tilde{g})$ is over $\mathbb{G}_2$. Note that the sub-keys are correlated, that is, $\log_g g_1 = \log_{\tilde{g}} \tilde{g}_1 = k_1$ and $\log_g g_2 = \log_{\tilde{g}} \tilde{g}_2 = k_2$. The commitment algorithm, $\mathsf{CM.Commit}$ consists of two instances of the Pedersen (vector) commitment algorithm, using each of the sub-keys. in (See Figure 1.)

$\underline{\mathsf{CM.Setup}(1^\lambda) \to \mathsf{ck}}$	$\underline{\mathsf{CM.Commit}(\mathsf{ck}, (m_1, m_2); r) \to \mathsf{cm}}$	$\underline{\mathsf{CM.Rerand}(\mathsf{ck}, \mathsf{cm}; r) \to \mathsf{cm}'}$
$(g, \tilde{g}) \xleftarrow{\$} \mathbb{G}_1 \times \mathbb{G}_2$	Parse ck as $((g_1, g_2, g), (\tilde{g}_1, \tilde{g}_2, \tilde{g}))$	Parse ck as $(\cdot, \cdot, g)$, $(\cdot, \cdot, \tilde{g})$
$(k_1, k_2) \xleftarrow{\$} \mathbb{Z}_p^2$	$\mathsf{cm}_{\mathbb{G}_1} \leftarrow g_1^{m_1} g_2^{m_2} g^r$	Parse cm as $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2})$
$g_1 \leftarrow g^{k_1} \quad g_2 \leftarrow g^{k_2}$	$\mathsf{cm}_{\mathbb{G}_2} \leftarrow \tilde{g}_1^{m_1} \tilde{g}_2^{m_2} \tilde{g}^r$	**return** $(\mathsf{cm}_{\mathbb{G}_1} \cdot g^r, \mathsf{cm}_{\mathbb{G}_2} \cdot \tilde{g}^r)$
$\tilde{g}_1 \leftarrow \tilde{g}^{k_1} \quad \tilde{g}_2 \leftarrow \tilde{g}^{k_2}$	**return** $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2})$	
return $((g_1, g_2, g), (\tilde{g}_1, \tilde{g}_2, \tilde{g}))$		

Figure 1: The commitment scheme.

It is not hard to verify that the scheme is perfectly hiding and homomorphic, just like standard Pedersen commitments. We reduce the binding of the scheme to the symmetric discrete log assumption (SDLP) [5, Assumption 2] that asserts that, given

$$g, g^\mu \in \mathbb{G}_1, \quad \text{and} \quad \tilde{g}, \tilde{g}^\mu \in \mathbb{G}_2, \qquad \text{where } \mu \xleftarrow{\$} \mathbb{Z}_p,$$

an efficient adversary cannot recover μ with more than negligible probability in the security parameter.

The reduction proceeds as follows. Let Adv be an adversary that on input $\mathsf{ck} = ((g_1, g_2, g), (\tilde{g}_1, \tilde{g}_2, \tilde{g}))$ outputs $((m_1, m_2, r), (m_1', m_2', r'))$ such that $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2}) = (\mathsf{cm}'_{\mathbb{G}_1}, \mathsf{cm}'_{\mathbb{G}_2})$ with probability ε, where

$$\begin{aligned} \mathsf{cm}_{\mathbb{G}_1} &= g_1^{m_1} g_2^{m_2} g^r \quad , \quad \mathsf{cm}'_{\mathbb{G}_1} = g_1^{m_1'} g_2^{m_2'} g^{r'}, \text{ and} \\ \mathsf{cm}_{\mathbb{G}_2} &= \tilde{g}_1^{m_1} \tilde{g}_2^{m_2} \tilde{g}^r \quad , \quad \mathsf{cm}'_{\mathbb{G}_2} = \tilde{g}_1^{m_1'} \tilde{g}_2^{m_2'} \tilde{g}^{r'}. \end{aligned}$$

We construct an adversary Adv' that solves the SDLP: Adv' is given $(g, \tilde{g}, h, \tilde{h})$, where $h = g^\mu$ and $\tilde{h} = \tilde{g}^\mu$, and do as follows:

1. Pick $i \in \{1, 2\}$ uniformly and let $j = 3 - i$. Set $g_i = g$ and $\tilde{g}_i = \tilde{g}$. In addition, set $g_j = h^\alpha$ and $\tilde{g}_j = \tilde{h}^\alpha$, where α is chosen uniformly from $\mathbb{Z}_p$.

2. Sends Adv the commitment key $\mathsf{ck} = \Big((h, g_1, g_2), (\tilde{h}, \tilde{g}_1, \tilde{g}_2)\Big)$. Observe that the commitment key ck and the commitment key output by $\mathsf{CM.Setup}(1^\lambda)$ are identically distributed.

3. With probability ε the adversary Adv responds with $((m_1, m_2, r), (m_1', m_2', r'))$ such that $(m_1, m_2) \neq (m_1', m_2')$ and $\mathsf{cm}_{\mathbb{G}_1} = \mathsf{cm}'_{\mathbb{G}_1}$, that is, $g_1^{m_1} g_2^{m_2} h^r = g_1^{m_1'} g_2^{m_2'} h^{r'}$ (in fact, with probability ε it holds that $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2}) = (\mathsf{cm}'_{\mathbb{G}_1}, \mathsf{cm}'_{\mathbb{G}_2})$, but we focus only on the first entry of the commitment). If the above does not hold (i.e. the commitments are not equal) or $m_i \neq m_i'$ then halt. Note that since i was chosen uniformly and is secret from Adv, we have that $m_i \neq m_i'$ with probability $\varepsilon/2$.

4. Let $\beta = \alpha m_j$ and $\beta' = \alpha m_j'$. We have

$$\begin{aligned} & g_1^{m_1} g_2^{m_2} h^r = g_1^{m_1'} g_2^{m_2'} h^{r'} \\ \implies \quad & g_i^{m_i} g_j^{m_j} h^r = g_i^{m_i'} g_j^{m_j'} h^{r'} \\ \implies \quad & g_i^{m_i} h^\beta h^r = g_i^{m_i'} h^{\beta'} h^{r'} \\ \implies \quad & g_i^{m_i - m_i'} = h^{\beta' - \beta + r' - r} \\ \implies \quad & g_i^{(m_i - m_i')(\beta' - \beta + r' - r)^{-1}} = h \end{aligned}$$

   where $g_i = g$,

5. Output $\mu = (m_i - m_i')(\beta' - \beta + r' - r)^{-1}$.

It follows that Adv' solves SDLP with probability $\varepsilon/2$.

B Security of PS-based rerandomizable signature-commitment scheme

The PS-based rerandomizable signature-commitment scheme is given in Figure 2.

Proof of Lemma 2.4 We prove Lemma 2.4. We begin by recalling Assumption 1 of [21] (hereafter referred to as the PS assumption). For a Type III bilinear pairings $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ over groups of prime order p with generators of $(g, \tilde{g}) \in \mathbb{G}_1 \times \mathbb{G}_2$, Assumption 1 of [21] asserts that every efficient algorithm Adv cannot win the following "PS" game with more than negligible probability in the security parameter:

1. The Challenger samples $x, y \xleftarrow{\$} \mathbb{Z}_p$ and publishes $\tilde{X} = \tilde{g}^x$, $Y = g^y$ and $\tilde{Y} = \tilde{g}^y$. (As usual we also assume that the adversary is also given the public parameters $(p, g, \tilde{g}, e)$ as an auxiliary input.)

2. The adversary is allowed to send queries of the form $m \in \mathbb{Z}_p$ on which the Challenger replies with the pair $(g^u, X^u Y^{mu})$ for a randomly chosen $u \xleftarrow{\$} \mathbb{Z}_p$. The adversary wins if it terminates with an output of the form $g^u, X^u Y^{m^* u}$ for some non-zero u and a new m^* that was not queried before.

$\underline{\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \to (\mathsf{sk}, \mathsf{pk})}$
Parse ck as $((g_1, g_2, g), (\tilde{g}_1, \tilde{g}_2, \tilde{g}))$
$x \xleftarrow{\$} \mathbb{Z}_p$
$X \leftarrow g^x, \tilde{X} = \tilde{g}^x$
return $\Big((g, X), (\mathsf{ck}, \tilde{X})\Big)$

$\underline{\mathsf{RS.Sign}(\mathsf{sk}, \mathsf{cm}; u) \to \sigma}$
Parse sk as (g, X)
Parse cm as $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2})$
$u \xleftarrow{\$} \mathbb{Z}_p$
return $(g^u, (X \cdot \mathsf{cm}_{\mathbb{G}_1})^u)$

$\underline{\mathsf{RS.Ver}(\mathsf{pk}, \mathsf{cm}, \sigma) \to \{0, 1\}}$
Parse pk as $(\mathsf{ck}, \tilde{X})$
Parse ck as $(\cdot, \cdot, g), (\cdot, \cdot, \tilde{g})$
Parse cm as $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2})$
Parse σ as (σ_1, σ_2)
assert $e(\sigma_2, \tilde{g}) = e(\sigma_1, \tilde{X} \cdot \mathsf{cm}_{\mathbb{G}_2})$

$\underline{\mathsf{RS.Rerand}(\mathsf{pk}, \sigma; r_\Delta, u_\Delta) \to \sigma'}$
Parse σ as (σ_1, σ_2)
return $(\sigma_1^{u_\Delta}, (\sigma_2 \cdot \sigma_1^{r_\Delta})^{u_\Delta})$

Figure 2: The rerandomizable signatures over the commitment scheme in Figure 1.

We prove Lemma 2.4 by showing that the scheme from Figure 2 is a rerandomizable signature over the commitment scheme from Figure 1, based on the above assumption. First observe that the underlying commitment scheme is a standard (2-slot) generalization of Pedersen commitments which is known to be perfectly hiding, perfectly rerandomizable and computationally-binding under the DLOG assumption (which must hold in both groups as otherwise the PS assumption does not hold). The correctness of the signature scheme can be easily verified and so it remains to be seen that the scheme is unforgeable.

Given an adversary Adv that breaks the signature scheme with probability ϵ, we construct two adversaries Adv_0 and Adv_1 with similar complexity such that Adv_0 wins the PS game with probability ϵ_0 and Adv_1 breaks the binding property of Pedersen's commitment with probability ϵ_1 where $\epsilon_1 + \epsilon_2 \geq \epsilon$.

For the sake of analysis, it will be convenient to describe Adv_0 and Adv_1 together, although Adv_0 plays the SP game whereas Adv_1 plays the binding game. On a first reading, the reader may want to skip Adv_1 parts of the description.

1. (a) Adv_0: Given the initial input in the PS game, $\mathsf{pp} = (p, g, \tilde{g}, e)$, $\tilde{X} = \tilde{g}^x$, $Y = g^y$ and $\tilde{Y} = \tilde{g}^y$, we sample, for each slot $i \in \{1, 2\}$ of the commitment a pair of random elements $(\alpha_i, \beta_i) \xleftarrow{\$} \mathbb{Z}_p^2$, and set $g_i = Y^{\alpha_i} g^{\beta_i}$ and $\tilde{g}_i = \tilde{Y}^{\alpha_i} g^{\beta_i}$ for $i \in \{1, 2\}$.

   (b) Adv_1: Given the public parameters $\mathsf{pp} = (p, g, \tilde{g}, e)$ and the parameters to (extended) Pedersen scheme $(g_1, g_2, \tilde{g}_1, \tilde{g}_2)$ we sample $x \xleftarrow{\$} \mathbb{Z}_p$, and set $X \leftarrow g^x, \tilde{X} = \tilde{g}^x$.

   Send to the signature adversary Adv the elements $\tilde{X}$, $g_1, g_2, \tilde{g}_1, \tilde{g}_2$ and forward the public parameters pp.

2. When the adversary Adv issues a signature query (m_1, m_2, r), we proceed as follows.

   (a) Adv_0: Send the query $m = \alpha_1 m_1 + \alpha_2 m_2$ to the PS-oracle. Given the answer $(g^u, X^u Y^{um})$ we return to the adversary the pair

   $$(g^u, X^u Y^{um} \cdot (g^u)^{\beta_1 m_1 + \beta_2 m_2 + r}) = (g^u, X^u (g_1^{m_1} g_2^{m_2} g^r)^u).$$

   Note that the second entry can be computed by raising the first entry to the power of $\beta_1 m_1 + \beta_2 m_2 + r$, and by multiplying the result by $X^u Y^{um}$.

   (b) Adv_1: Compute the (first part of the) commitment $\mathsf{cm}_{\mathbb{G}_1} = (g_1^{m_1} g_2^{m_2} g^r)$, sample $u \xleftarrow{\$} \mathbb{Z}_p$, sign the commitment by $\sigma = (g^u, (X \cdot \mathsf{cm}_{\mathbb{G}_1})^u)$ and send σ to Adv.

3. When the adversary terminates with a pair (m_1^*, m_2^*), commitment randomizer r^* and a "candidate signature" (U, B) we proceed as follows:

(a) Adv_0: output the pair $(U, B/U^{\beta_1 m_1^* + \beta_2 m_2^* + r^*})$.

(b) Adv_1: If there exists a previous query $(m_1, m_2) \neq (m_1^*, m_2^*)$ for which $(g_1^{m_1} g_2^{m_2}) = (g_1^{m_1^*} g_2^{m_2^*})$, then the adversary terminates with (m_1, m_2, r) and (m_1^*, m_2^*, r) for some arbitrary r.

Analysis. Observe that, both for Adv_0 and Adv_1, the signature's public key $(\tilde{X}, g_1, \tilde{g}_1, g_2, \tilde{g}_2)$ is distributed properly. Moreover, for every valid fixing of the public key, both Adv_0 and Adv_1 answer the queries of Adv perfectly according to the distribution of the real forgery game. Fix a public key and randomness for the adversary and let us consider the case that Adv wins the forgery game with the values $(m_1^*, m_2^*, r^*, U, B)$. We will show that at least one of the adversaries, Adv_0 and Adv_1, wins as well.

Recall that when Adv wins the forgery game, the pair (m_1^*, m_2^*) must be a new pair that was not queried before. We distinguish between two cases.

- The value $m^* = \alpha_1 m_1^* + \beta_1 m_2^*$ is "new" in the sense that for every previous query (m_1, m_2) it holds that $m^* \neq \alpha_1 m_1 + \alpha_1 m_2$. In this case, we show that the adversary Adv_0 wins the PS game. Indeed, since Adv wins the forgery game it holds that $e(B, \tilde{g}) = e(U, \tilde{X} \cdot \tilde{g}_1^{m_1^*} \tilde{g}_2^{m_2^*} \tilde{g}^{r^*})$. Letting $B = g^b$ and $U = g^u$, it follows that
$$b = u(x + m_1^*(y\alpha_1 + \beta_1) + m_2^*(y\alpha_2 + \beta_2) + r^*),$$
and therefore
$$(U, B/U^{\beta_1 m_1^* + \beta_2 m_2^* + r^*}) = (g^u, g^{b - u(\beta_1 m_1^* + \beta_2 m_2^* + r^*)}) = (g^u, X^u Y^{m^* u}).$$
We conclude that Adv_0 generated a "valid" pair $(g^u, X^u Y^{m^* u})$ with respect to a new value of m^* on which the PS oracle was not queried, and so Adv_0 wins the PS game.

- The value $\alpha_1 m_1^* + \alpha_1 m_2^*$ equals to $\alpha_1 m_1 + \alpha_1 m_2$ for some previous query (m_1, m_2). Since (m_1^*, m_2^*) is a new pair, it holds that $(m_1, m_2) \neq (m_1^*, m_2^*)$ and so the adversary Adv_1 finds a collision. Indeed, for every $r \in \mathbb{Z}_p$, we have that
$$(g_1^{m_1} g_2^{m_2} g^r) = (g_1^{m_1^*} g_2^{m_2^*} g^r) \quad \text{and} \quad (\tilde{g}_1^{m_1} \tilde{g}_2^{m_2} \tilde{g}^r) = (\tilde{g}_1^{m_1^*} \tilde{g}_2^{m_2^*} \tilde{g}^r),$$
in contradiction to the binding property of the commitment.

Lemma 2.4 follows. □

C Zero-Knowledge Proofs

In this section, we describe some concrete instantations of our zero-knowledge proofs. Recall that these proofs are compiled to the non-interactive setting via the Fiat-Shamir transform as described in Section 2.5. For compatibility with previous versions, we have that the commitment basis is denoted by (g_1, g_2, g_3, g_6, g).

C.1 Consistency of Splitting

For each $i \in \{1, 2\}$, the client generates ZKPOK Π_{Split_i} for the split relation $\mathcal{R}_{\mathsf{split}}$ that contains
$$\mathbb{x} = (\mathsf{ccm}_i, \mathsf{vcm}_i, \mathsf{rcm}', \mathsf{nullif}_i), \quad \mathbb{w} = (\mathsf{s}_{\mathsf{Sender}}, \mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, r_i, z_i, a')$$
for which the following conditions hold

$$\begin{aligned}
\mathsf{ccm}_i &= \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, 0); r_i) &&= g_1^{\mathsf{pid}_{\mathsf{Sender}}} g_2^{\mathsf{sn}_i} g_3^{\mathsf{val}_i} g^{r_i} \\
\mathsf{vcm}_i &= \mathsf{CM.Commit}(\mathsf{ck}, (0, 0, \mathsf{val}_i, 0); z_i) &&= g_3^{\mathsf{val}_i} g^{z_i} \\
\mathsf{rcm}' &= \mathsf{CM.Commit}(\mathsf{ck}, (\mathsf{pid}_{\mathsf{Sender}}, 0, 0, \mathsf{s}_{\mathsf{Sender}}); a') &&= g_1^{\mathsf{pid}_{\mathsf{Sender}}} g_6^{\mathsf{s}_{\mathsf{Sender}}} g^{a'} \\
\mathsf{nullif}_i &= \mathsf{PRF}_{\mathsf{s}_{\mathsf{Sender}}}(\mathsf{sn}_i) &&= h^{1/(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i)}
\end{aligned}$$

The proof is a Σ-protocol of the form $(\mathsf{init}, \mathsf{challenge}, \mathsf{response})$ where $\mathsf{challenge} = c \leftarrow \mathbb{Z}_p$ (where p is the groups order) is chosen by the verifier.

C.1.1 Prover

Message init. The prover picks random $\{x_j\}_{j\in[5]}$ and $\{t_i\}_{i\in[2]}$ from $\mathbb{Z}_p$, and computes:

$$\begin{aligned}
\mathsf{vk}_i &= \tilde{h}^{(\mathsf{s}_{\mathsf{Sender}}+\mathsf{sn}_i)}\tilde{w}^{t_i} \\
y_i &= e(\mathsf{nullif}_i, \tilde{w})^{t_i} \\
X_1 &= g_1^{x_1} g_2^{x_2} g_3^{x_3} g^{x_4} \\
X_2 &= g_3^{x_3} g^{x_5} \\
X_3 &= g_1^{x_1} g_6^{x_6} g^{x_7} \\
X_4 &= \tilde{h}^{x_6} \tilde{h}^{x_2} \tilde{w}^{x_8} \\
X_5 &= q_i^{x_8} \qquad\qquad //q_i \triangleq e(\mathsf{nullif}_i, \tilde{w}) \in
\end{aligned}$$

Here, t_i is secret randomness from $\mathbb{Z}_p$ and $(\tilde{h}, \tilde{w}) \in \mathbb{G}_2^2$ are part of the public parameters. The values y_i and vk_i are used in order to prove that nullif_i is consistent (i.e. that it uses $\mathsf{s}_{\mathsf{Sender}}$ and sn_i as in ccm_i and rcm'). However, before we can use y_i and vk_i to prove consistency of nullif, it has to be verified that y_i and vk_i are consistent on their own, that is, that both use the same randomizer t_i and that vk_i is a commitment to $\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}$ using the commitment key $(\tilde{h}, \tilde{w})$. This verification is done using the values X_4, X_5. Finally, we verify that the rest of the conditions in the statement using the values X_1, X_2, X_3.

Message response. Upon receiving the challenge e from the verifier, the prover sends $\{\alpha_j\}_{j\in[8]}$ as follows:

$$\begin{aligned}
\alpha_1 &= x_1 + c \cdot \mathsf{pid}_{\mathsf{Sender}} \\
\alpha_2 &= x_2 + c \cdot \mathsf{sn}_i \\
\alpha_3 &= x_3 + c \cdot \mathsf{val}_i \\
\alpha_4 &= x_4 + c \cdot r_i \\
\alpha_5 &= x_5 + c \cdot z_i \\
\alpha_6 &= x_7 + c \cdot a' \\
\alpha_7 &= x_6 + c \cdot \mathsf{s}_{\mathsf{Sender}} \\
\alpha_8 &= x_8 + c \cdot t_i
\end{aligned}$$

C.1.2 Verifier

The verifier outputs 'accept' if *all* the below statements hold, and 'reject' otherwise.

$$e(\mathsf{nullif}_i, \mathsf{vk}_i) = e(h, \tilde{h}) \cdot y_i \tag{22}$$

$$\mathsf{ccm}_i^c \cdot X_1 = g_1^{\alpha_1} g_2^{\alpha_2} g_3^{\alpha_3} g^{\alpha_4} \tag{23}$$

$$\mathsf{vcm}_i^c \cdot X_2 = g_3^{\alpha_3} g^{\alpha_5} \tag{24}$$

$$\mathsf{rcm}'^c \cdot X_3 = g_1^{\alpha_1} g_6^{\alpha_7} g^{\alpha_6} \tag{25}$$

$$\mathsf{vk}_i^c \cdot X_4 = \tilde{h}^{\alpha_7} \tilde{h}^{\alpha_2} \tilde{w}^{\alpha_8} \tag{26}$$

$$y_i^c \cdot X_5 = q_i^{\alpha_8} \tag{27}$$

C.1.3 Completeness

For a honest prover, all statements in Eq. (22) hold:

$$\begin{aligned}
\mathsf{ccm}_i^c \cdot X_1 &= (g_1^{\mathsf{pid}_{\mathsf{Sender}}} g_2^{\mathsf{sn}_i} g_3^{\mathsf{val}_i} g^{r_i})^c \cdot g_1^{x_1} g_2^{x_2} g_3^{x_3} g^{x_4} &&= g_1^{x_1 + c\cdot \mathsf{pid}_{\mathsf{Sender}}} g_2^{x_2 + c\cdot \mathsf{sn}_i} g_3^{x_3 + c\cdot \mathsf{val}_i} g^{x_4 + c\cdot r_i} &&= g_1^{\alpha_1} g_2^{\alpha_2} g_3^{\alpha_3} g^{\alpha_4} \\
\mathsf{vcm}_i^c \cdot X_2 &= (g_3^{\mathsf{val}_i} g^{z_i})^c \cdot g_3^{x_3} g^{x_5} &&= g_3^{x_3 + c\cdot \mathsf{val}_i} g^{x_5 + c\cdot z_i} &&= g_3^{\alpha_3} g^{\alpha_5} \\
\mathsf{rcm}'^e \cdot X_3 &= (g_1^{\mathsf{pid}_{\mathsf{Sender}}} g_6^{\mathsf{s}_{\mathsf{Sender}}} g^{a'})^c \cdot g_1^{x_1} g_6^{x_6} g^{x_7} &&= g_1^{x_1 + c\cdot \mathsf{pid}_{\mathsf{Sender}}} g_6^{x_6 + c\cdot \mathsf{s}_{\mathsf{Sender}}} g^{x_7 + c\cdot a'} &&= g_1^{\alpha_1} g_6^{\alpha_7} g^{\alpha_6} \\
\mathsf{vk}_1^c \cdot X_4 &= (\tilde{h}^{(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i)} \tilde{w}^{t_i})^c \cdot \tilde{h}^{x_6} \tilde{h}^{x_2} \tilde{w}^{x_8} &&= \tilde{h}^{x_6 + c\cdot \mathsf{s}_{\mathsf{Sender}}} \tilde{h}^{x_2 + c\cdot \mathsf{sn}_i} \tilde{w}^{x_8 + c\cdot t_i} &&= \tilde{h}^{\alpha_7} \tilde{h}^{\alpha_2} \tilde{w}^{\alpha_8} \\
y_i^c \cdot X_5 &= q_i^{c\cdot t_i} \cdot q_i^{x_8} &&= q_i^{x_8 + c\cdot t_i} &&= q_i^{\alpha_8}
\end{aligned}$$

And,

$$\begin{aligned}
e(h, \tilde{h}) \cdot y_i &= e(h, \tilde{h}) \cdot e(\mathsf{nullif}_i, \tilde{w})^{t_i} \\
&= e(h^{1/(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i)}, \tilde{h}^{(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i)}) \cdot e(\mathsf{nullif}_i, \tilde{w})^{t_i} \\
&= e(\mathsf{nullif}_i, \tilde{h}^{(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i)}) \cdot e(\mathsf{nullif}_i, \tilde{w}^{t_i}) \\
&= e(\mathsf{nullif}_i, \tilde{h}^{(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i)} \tilde{w}^{t_i}) \\
&= e(\mathsf{nullif}_i, \mathsf{vk}_i)
\end{aligned}$$

C.1.4 Soundness

Consider two accepting executions of the protocol above: $(\mathsf{init}, \mathsf{challenge}_1, \mathsf{response}_1)$ and $(\mathsf{init}, \mathsf{challenge}_2, \mathsf{response}_2)$. Denote by $\mathsf{init} = (\mathsf{vk}_i, y_i, \{X_j\}_{j\in[5]})$, $\mathsf{challenge}_1 = c$, $\mathsf{challenge}_2 = c'$, $\mathsf{response}_1 = \{\alpha_j\}_{j\in[8]}$ and $\mathsf{response}_2 = \{\alpha'_j\}_{j\in[8]}$. Then, we first show knowledge extraction of the values $\mathbb{w} = (\mathsf{s}_{\mathsf{Sender}}, \mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, r_i, z_i, a')$ and that these values are consistent in $\mathsf{ccm}_i, \mathsf{vcm}_i, \mathsf{rcm}', \mathsf{vk}_i$ and y_i. Later we show that these values are consistent within nullif_i as well.

From the verification procedure it follows that:

$$\mathsf{ccm}_i^{c-c'} = g_1^{\alpha_1 - \alpha'_1} g_2^{\alpha_2 - \alpha'_2} g_3^{\alpha_3 - \alpha'_3} g^{\alpha_4 - \alpha'_4} \tag{28}$$

$$\mathsf{vcm}_i^{c-c'} = g_3^{\alpha_3 - \alpha'_3} g_5^{\alpha_5 - \alpha'_5} \tag{29}$$

$$\mathsf{rcm}'^{c-c'} = g_1^{\alpha_1 - \alpha'_1} g_6^{\alpha_7 - \alpha'_7} g^{\alpha_6 - \alpha'_6} \tag{30}$$

$$y_1^{c-c'} = q_1^{\alpha_8 - \alpha'_8} \tag{31}$$

$$\mathsf{vk}_i^{c-c'} = \tilde{h}^{\alpha_7 - \alpha'_7} \tilde{h}^{\alpha_2 - \alpha'_2} \tilde{w}^{\alpha_8 - \alpha'_8} \tag{32}$$

Define $\mathsf{exp}(g_k, C = \prod_j g_j^{\mathsf{ex}_j}) \triangleq \mathsf{ex}_k$, where $\{\mathsf{ex}_j\}_j$ are known by the prover, then,

$$\begin{aligned}
(28, 30) &\implies \mathsf{exp}(g_1, \mathsf{ccm}_i) = \mathsf{exp}(g_1, \mathsf{rcm}') = (\alpha_1 - \alpha'_1)/(e - e') = \mathsf{pid} \\
(28, 29) &\implies \mathsf{exp}(g_3, \mathsf{ccm}_i) = \mathsf{exp}(g_3, \mathsf{vcm}_i) = (\alpha_3 - \alpha'_3)/(e - e') = \mathsf{val}_i \\
(28, 30, 32) &\implies \mathsf{exp}(g_2, \mathsf{ccm}_i) + \mathsf{exp}(g_6, \mathsf{rcm}') = \mathsf{exp}(\tilde{h}, \mathsf{vk}_i) = (\alpha_2 + \alpha_7 - \alpha'_2 - \alpha'_7)/(e - e') = \mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i \\
(32, 31) &\implies \mathsf{exp}(q_i, y_i) = \mathsf{exp}(\tilde{w}, \mathsf{vk}_i) = t
\end{aligned}$$

It is shown that ccm_i and rcm' use the same pid, that ccm_i and vcm_i use the same value v_i, that the sum $\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i$ is indeed the sum of the values sn_i and $\mathsf{s}_{\mathsf{Sender}}$ in ccm_i and rcm', respectively, and that the value t is used in both y_i and vk_i.

It remains to show that nullif_i is computed correctly. Assume $\mathsf{nullif}_i = h^a$. Then, since the verification

succeeds, we have:

$$\begin{aligned} e(\mathsf{nullif}_i, \mathsf{vk}_i) &= e(h, \tilde{h}) \cdot y_i && \Leftrightarrow \\ e(h^a, \tilde{h}^{\mathsf{s}_{\mathsf{Sender}}+\mathsf{sn}_i}) \cdot e(h^a, \tilde{w}^{t_i}) &= e(h, \tilde{h}) \cdot e(h^a, \tilde{w}^{t_i}) && \Leftrightarrow \\ e(h, \tilde{h})^{a(\mathsf{s}_{\mathsf{Sender}}+\mathsf{sn}_i)} &= e(h, \tilde{h}) && \Leftrightarrow \\ a(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i) &= 1 && \Leftrightarrow \\ a &= 1/(\mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i) \end{aligned}$$

Thus, $\mathsf{nullif}_i = h^a = h^{1/(\mathsf{s}_{\mathsf{Sender}}+\mathsf{sn}_i)}$ as desired.

C.1.5 Zero-Knowledge

Given the statement $\mathbb{x} = (\mathsf{ccm}_i, \mathsf{vcm}_i, \mathsf{rcm}', \mathsf{nullif}_i)$, the simulator picks a random challenge $\hat{c}$, random responses $\{\hat{\alpha}_j\}_{j\in[8]}$ and a random $\hat{\mathsf{vk}}_i$ and computes:

$$\hat{y}_i = e(\mathsf{nullif}_i, \hat{\mathsf{vk}}_i)/e(h, \tilde{h})$$

and

$$\begin{aligned} \hat{X}_1 &= g_1^{\hat{\alpha}_1} g_2^{\hat{\alpha}_2} g_3^{\hat{\alpha}_3} g^{\hat{\alpha}_4} \cdot \mathsf{ccm}_i^{-\hat{c}} \\ \hat{X}_2 &= g_3^{\hat{\alpha}_3} g^{\hat{\alpha}_5} \cdot \mathsf{vcm}^{-\hat{c}} \\ \hat{X}_3 &= g_1^{\hat{\alpha}_1} g_6^{\hat{\alpha}_6} g^{\hat{\alpha}_7} \cdot \mathsf{rcm}'^{-\hat{c}} \\ \hat{X}_4 &= \tilde{h}^{\hat{\alpha}_7} \tilde{h}^{\hat{\alpha}_2} \tilde{w}^{\hat{\alpha}_8} \cdot \hat{\mathsf{vk}}_i^{-\hat{c}} \\ \hat{X}_5 &= q_i^{\hat{\alpha}_8} \cdot \hat{y}_i^{-\hat{c}} \end{aligned}$$

The simulator outputs the transcript $(\hat{\mathsf{init}}, \hat{\mathsf{challenge}}, \hat{\mathsf{response}})$ where

$$\begin{aligned} \hat{\mathsf{init}} &= (\hat{\mathsf{vk}}_i, \hat{y}_i, \{\hat{X}_j\}_{j\in[5]}) \\ \hat{\mathsf{challenge}} &= \hat{c} \\ \hat{\mathsf{response}} &= \{\hat{\alpha}_j\}_{j\in[8]} \end{aligned}$$

We argue that $(\hat{\mathsf{init}}, \hat{\mathsf{challenge}}, \hat{\mathsf{response}})$ is distributed identically as $(\mathsf{init}, \mathsf{challenge}, \mathsf{response})$ in the real execution. Note that in the real execution vk_i is uniform in $\mathbb{G}_2$ (since t_i is uniform) and $c, \{\alpha_j\}$ are uniform in $\mathbb{Z}_p$ (since $\{x_j\}$ are uniform). Given that, the values $y_i, \{X_j\}$ are fully determined, as a function of $\mathsf{vk}_i, c, \{x_j\}$ (and the commitment basis). We observe that $(\hat{\mathsf{init}}, \hat{\mathsf{challenge}}, \hat{\mathsf{response}})$ is distributed exactly the same. $\hat{\mathsf{vk}}_i, \hat{c}$ and $\{\hat{\alpha}_j\}$ are uniform in $\mathbb{G}_2$ and $\mathbb{Z}_p$ respectively. Then, the values $y_i, \{X_j\}$ adhere the same conditions as in the real execution, as the verification outputs 'accept' on $(\hat{\mathsf{init}}, \hat{\mathsf{challenge}}, \hat{\mathsf{response}})$:

$$\begin{aligned} e(h, \tilde{h}) \cdot \hat{y}_i &= e(h, \tilde{h}) \cdot e(\mathsf{nullif}_i, \hat{\mathsf{vk}}_i)/e(h, \tilde{h}) && = e(\mathsf{nullif}_i, \hat{\mathsf{vk}}_i) \\ \mathsf{ccm}_i^{\hat{c}} \cdot \hat{X}_1 &= \mathsf{ccm}_i^{\hat{c}} \cdot g_1^{\hat{\alpha}_1} g_2^{\hat{\alpha}_2} g_3^{\hat{\alpha}_3} g^{\hat{\alpha}_4} \cdot \mathsf{ccm}_i^{-\hat{c}} && = g_1^{\hat{\alpha}_1} g_2^{\hat{\alpha}_2} g_3^{\hat{\alpha}_3} g^{\hat{\alpha}_4} \\ \mathsf{vcm}_i^{\hat{c}} \cdot \hat{X}_2 &= \mathsf{vcm}_i^{\hat{c}} \cdot g_3^{\hat{\alpha}_3} g^{\hat{\alpha}_5} \cdot \mathsf{vcm}^{-\hat{c}} && = g_3^{\hat{\alpha}_3} g^{\hat{\alpha}_5} \\ \mathsf{rcm}'^{\hat{c}} \cdot \hat{X}_3 &= \mathsf{rcm}'^{\hat{c}} \cdot g_1^{\hat{\alpha}_1} g_6^{\hat{\alpha}_6} g^{\hat{\alpha}_7} \cdot \mathsf{rcm}'^{-\hat{c}} && = g_1^{\hat{\alpha}_1} g_6^{\hat{\alpha}_7} g^{\hat{\alpha}_6} \\ \hat{\mathsf{vk}}_i^{\hat{c}} \cdot \hat{X}_4 &= \hat{\mathsf{vk}}_i^{\hat{c}} \cdot \tilde{h}^{\hat{\alpha}_7} \tilde{h}^{\hat{\alpha}_2} \tilde{w}^{\hat{\alpha}_8} \cdot \hat{\mathsf{vk}}_i^{-\hat{c}} && = \tilde{h}^{\hat{\alpha}_7} \tilde{h}^{\hat{\alpha}_2} \tilde{w}^{\hat{\alpha}_8} \\ \hat{y}_i^{\hat{c}} \cdot \hat{X}_5 &= \hat{y}_i^{\hat{c}} \cdot q_i^{\hat{\alpha}_8} \cdot \hat{y}_i^{-\hat{c}} && = \hat{q}_i^{\alpha_8} \end{aligned}$$

■

C.2 KZG-Pedersen agreement

The prover wants to prove the following statement:

$$\mathbb{x} = (c, c', \mathsf{cm}, Y) \tag{33}$$

$$\mathbb{w} = [r, z, a] \tag{34}$$

$$\mathcal{R}(\mathbb{x}, \mathbb{w}) = \begin{pmatrix} c' = & c^r \wedge & \\ \mathsf{cm} = & g_3^a g^z \wedge & \\ Y = & q^{ar} & //q = e(G, \tilde{G}) \end{pmatrix} \tag{35}$$

Where $r \leftarrow \mathbb{Z}_p$, c is a KZG commitment, cm is a Pedersen commitment (with a randomizer z) and $a = \phi(i)$ where $\phi(x)$ is a polynomial.

C.2.1 Prover

The prover picks uniformly random $\{x_i\}_{i\in[5]}$ from $\mathbb{Z}_p$ and sends init:

$$\begin{aligned} \mathsf{cm}' &= \mathsf{cm}^r = (g_3^a g^z)^r = g_3^{ar} g^{zr} \\ X_1 &= c^{x_1} \\ X_2 &= g_3^{x_2} g^{x_3} \\ X_3 &= q^{x_4} \\ X_4 &= g_3^{x_4} g^{x_5} \\ X_5 &= \mathsf{cm}^{x_1} \end{aligned}$$

Upon receiving the challenge e, the prover sends $\mathsf{response}$:

$$\begin{aligned} \alpha_1 &= x_1 + e \cdot r \\ \alpha_2 &= x_2 + e \cdot a \\ \alpha_3 &= x_3 + e \cdot z \\ \alpha_4 &= x_4 + e \cdot ar \\ \alpha_5 &= x_5 + e \cdot zr \end{aligned}$$

C.2.2 Verifier

$$c'^e \cdot X_1 = c^{\alpha_1} \tag{36}$$

$$\mathsf{cm}^e \cdot X_2 = g_3^{\alpha_2} g^{\alpha_3} \tag{37}$$

$$Y^e \cdot X_3 = q^{\alpha_4} \tag{38}$$

$$\mathsf{cm}'^e \cdot X_4 = g_3^{\alpha_4} g^{\alpha_5} \tag{39}$$

$$\mathsf{cm}'^e \cdot X_5 = \mathsf{cm}^{\alpha_1} \tag{40}$$

C.2.3 Completeness

$$(36) : c'^e \cdot X_1 = c^{e\cdot r} \cdot c^{x_1} = c^{x_1 + e\cdot r} = c^{\alpha_1}$$

$$(37) : \mathsf{cm}^e \cdot X_2 = (g_3^a g^z)^e \cdot g_3^{x_2} g^{x_3} = g_3^{x_2 + e\cdot a} g^{x_3 + e\cdot z} = g_3^{\alpha_2} g^{\alpha_3}$$

$$(38) : Y^e \cdot X_3 = q^{e\cdot ar} q^{x_4} = q^{\alpha_4}$$

$$(39) : \mathsf{cm}'^e \cdot X_4 = g_3^{e\cdot ar} g^{e\cdot zr} \cdot g_3^{x_4} g^{x_5} = g_3^{x_4 + e\cdot ar} g^{x_5 + e\cdot zr} = g_3^{\alpha_4} g^{\alpha_5}$$

$$(40) : \mathsf{cm}'^e \cdot X_5 = g_3^{e\cdot ar} g^{e\cdot zr} \cdot \mathsf{cm}^{x_1} = g_3^{e\cdot ar} g^{e\cdot zr} \cdot g_3^{ax_1} g^{zx_1} = (g_3^a)^{x_1 + er} (g^z)^{x_1 + er} = (g_3^a g^z)^{x_1 + er} = \mathsf{cm}^{\alpha_1}$$

C.2.4 Soundness

Consider equations (41)-(45), which are implied by equations (36)-(40) above.

- From equation (41) we can extract $\log_c(c') = (\alpha_1 - \alpha'_1)/(e - e')$.
- From equation (42) we can extract $\log_{\mathsf{cm}}(\mathsf{cm}') = (\alpha_1 - \alpha'_1)/(e - e')$. Notice that $\log_c(c') = \log_{\mathsf{cm}}(\mathsf{cm}')$ as required, denote this value by r.
- From equation (43) we can extract $[a, z] \triangleq \log_{(g_3,g)}(\mathsf{cm}) = [(\alpha_2 - \alpha'_2)/(e - e'), (\alpha_3 - \alpha'_3)/(e - e')]$.
- From equation (44) we get that $m \triangleq \log_q(Y) = (\alpha_4 - \alpha'_4)/(e - e')$. We show later that $m = a \cdot r$.
- Finally, from equation (45) we get $[a', z'] \triangleq \log_{(g_3,g)}(\mathsf{cm}') = [(\alpha_4 - \alpha'_4)/(e - e'), (\alpha_5 - \alpha'_5)/(e - e')]$. The fact that $r = \log_{\mathsf{cm}}(\mathsf{cm}')$ implies that $[a', z'] = r \cdot [a, z] = [ra, rz]$.
- Notice that $m = a = ra$ as required.

$$(36) \implies c'^{e-e'} = c^{\alpha_1 - \alpha'_1} \tag{41}$$
$$(40) \implies \mathsf{cm}'^{e-e'} = \mathsf{cm}^{\alpha_1 - \alpha'_1} \tag{42}$$
$$(37) \implies \mathsf{cm}^{e-e'} = g_3^{\alpha_2 - \alpha'_2} g^{\alpha_3 - \alpha'_3} \tag{43}$$
$$(38) \implies Y^{e-e'} = q^{\alpha_4 - \alpha'_4} \tag{44}$$
$$(39) \implies \mathsf{cm}'^{e-e'} = g_3^{\alpha_4 - \alpha'_4} g^{\alpha_5 - \alpha'_5} \tag{45}$$

C.2.5 Zero-Knowledge

First, notice that in the real execution the values $\{x_i\}_{i\in[5]}$ and e are uniformly random and the values cm' and $\{X_i\}_{i\in[5]}$ are correlated by $(a, r, z, \{x_i\}_{i\in[5]})$. We show a simulator that outputs a distribution that is indistinguishable from the above.

The simulator picks $\mathsf{challenge}' = e$ and $\mathsf{response}' = \{\alpha_i\}_{i\in[5]}$ at random from $\mathbb{Z}_p$. Then, it computes init' as:

$$\begin{aligned}
\mathsf{cm}' &\leftarrow_R \mathbb{G}_1 \\
X_1 &= c^{\alpha_1} \cdot c'^{-e} \\
X_2 &= g_3^{\alpha_2} g^{\alpha_3} \cdot \mathsf{cm}^{-e} \\
X_3 &= q^{\alpha_4} \cdot Y^{-e} \\
X_4 &= g_3^{\alpha_4} g^{\alpha_5} \cdot \mathsf{cm}'^{-e} \\
X_5 &= \mathsf{cm}^{\alpha_1} \cdot \mathsf{cm}'^{-e}
\end{aligned}$$

All verification condition hold, as:

$$\begin{aligned}
&(36): c'^{e} \cdot X_1 = c'^{e} \cdot c^{\alpha_1} \cdot c'^{-e} = c^{\alpha_1} \\
&(37): \mathsf{cm}^{e} \cdot X_2 = \mathsf{cm}^{e} \cdot g_3^{\alpha_2} g^{\alpha_3} \cdot \mathsf{cm}^{-e} = g_3^{\alpha_2} g^{\alpha_3} \\
&(38): Y^{e} \cdot X_3 = Y^{e} \cdot q^{\alpha_4} \cdot Y^{-e} = q^{\alpha_4} \\
&(39): \mathsf{cm}'^{e} \cdot X_4 = \mathsf{cm}'^{e} \cdot g_3^{\alpha_4} g^{\alpha_5} \cdot \mathsf{cm}'^{-e} = g_3^{\alpha_4} g^{\alpha_5} \\
&(39): \mathsf{cm}'^{e} \cdot X_5 = \mathsf{cm}'^{e} \cdot \mathsf{cm}^{\alpha_1} \cdot \mathsf{cm}'^{-e} = \mathsf{cm}^{\alpha_1}
\end{aligned}$$