UTT: Decentralized Ecash with Accountable Privacy

Alin Tomescu* Adithya Bhat† Benny Applebaum‡ Ittai Abraham§
Guy Gueta§ Benny Pinkas¶ Avishay Yanai§

April 9, 2022

Abstract

We present *UnTraceable Transactions (UTT)*, a system for *decentralized ecash with accountable privacy*. UTT is the first ecash system that obtains three critical properties: (1) it provides *decentralized trust* by implementing the ledger, bank, auditor, and registration authorities via threshold cryptography and Byzantine Fault Tolerant infrastructure; (2) it *balances accountability and privacy* by implementing *anonymity budgets*: users can anonymously send payments, but only up to a limited amount of currency per month. Past this point, transactions can either be made public or subjected to customizable auditing rules; (3) by carefully choosing cryptographic building blocks and co-designing the cryptography and decentralization, UTT is *tailored for high throughput and low latency*. With a combination of optimized cryptographic building blocks and vertical scaling (optimistic concurrency control), UTT can provide almost 1,000 payments with accountable privacy per second, with latencies of around 100 milliseconds and less. Through horizontal scaling (multiple shards), UTT can scale to tens of thousands of such transactions per second. With 60 shards we measure over 10,000 transactions with accountable privacy per second.

We formally define and prove the security of UTT using an MPC-style ideal functionality. Along the way, we define a new MPC framework that captures the security of reactive functionalities in a standalone setting, thus filling an important gap in the MPC literature. Our new framework is compatible with practical instantiations of cryptographic primitives and provides a trade-off between concrete efficiency and provable security that may also be useful for future work.

*Aptos, work done while at VMware
†VMware and Purdue
‡VMware and TAU
§VMware
¶VMware and BIU

1 Introduction

From Chaum's groundbreaking work on *Electronic cash (ecash)* and blind signatures [1], anonymous ecash schemes have received significant attention over the last 40 years [2–14]. Such ecash systems require an online *bank* entity (sometimes called a *ledger*) that records transactions and often has control over the minting of new anonymous tokens. One prominent materialization of this interest is the deployment of Zcash [15], a decentralized anonymous payment scheme that hides all transaction details, including *senders*, *recipients* and amounts [16].

There are serious concerns that systems that allow payments with *unconditional* anonymity could make illicit activities worse [17–19], and calls for future regulation may make unconditionally-anonymous payment systems illegal [20]. This has led to a line of research around *accountable privacy* that aims to strike a balance between privacy and *accountability*. Research on responsible privacy originates from the pioneering work of Sander and Ta-Shma [5,8] which has recently reflourished [21–31].

The emergence of *central bank digital currencies (CBDC)* has further increased demand for balancing *privacy* with *accountability*. Most relevant to our work is the idea of *anonymity budgets*, originally suggested and briefly sketched in [5], that allows users to anonymously send payments, but only as long as the total sum of the payments is smaller than some limited amount. This anonymity budget can be set and replenished, say once a month, by an *auditing authority*.

A digital analogue of physical cash

Motivated by these emerging use cases, the main research question addressed in this paper can be informally stated as follows.

> *Can digital cash systems provide a digital analogue to physical cash?*

Towards answering this question we identify three key characteristics of physical cash that are crucial for a digital analogue.

1. **Trust**: Physical cash is issued by trusted authorities and highly regulated entities. It embeds physical protections that make it hard to double spend and counterfeit. In fact, one may argue that the *value* of cash strongly relies on it being issued by a *trustworthy* authority; see, e.g., [32].

2. **Accountable Privacy**: With physical cash, small transactions are anonymous and private. In many jurisdictions, large cash payments are regulated, require auditor approval, are time delayed, or might even be illegal [33].

3. **Scalability**: Transacting with physical cash is very quick and can trivially scale horizontally (cash transactions can trivially be executed in parallel by different pairs of users).

Unfortunately, physical cash requires the payer and the payee to physically transfer a note between them; this makes physical cash rather useless in a digital setting.

In this paper, we consider digital ecash systems that are operated by licensed, supervised, established, and highly regulated parties. To provide a digital analogue of physical cash, ecash systems have the following properties.

1. **Trust**: To obtain fault tolerance and decentralized trust, the *logically-centralized* ecash entities (e.g., the bank) need to be *decentralized to multiple fault-isolated servers* so that the security and privacy of the system is maintained even if a threshold of the servers are compromised by a malicious adversary.

2. **Accountable Privacy**: To limit the total sum of private payments, users of the system need to receive a digital anonymity budget from an *auditing authority* at a regular time interval (say monthly). All transactions that stay within this anonymity budget need to remain completely private. That is, all the information about a transaction (e.g., the payer, payee, the amount of money and its relation to other transactions) is hidden from everyone including the bank. Digital payments beyond the anonymity budget need to go through a real-time auditor approval.

3. **Scalability**: For an ecash system to be useful, it needs to support low latency transactions and be able to scale (both vertically and horizontally) to handle (tens of) thousands of transactions with accountable privacy per second.

Contributions

- The main contribution of this work is an ecash system named UTT that combines decentralized trust, accountable privacy, and scalability.

- UTT obtains **decentralized trust** by implementing the bank (and other authorities) via $3f + 1$ servers using threshold cryptography and Byzantine Fault Tolerance such that the security and privacy guarantees are maintained despite any f malicious servers and any number of colluding users. Furthermore, we co-designed the fault tolerance mechanisms along with the cryptography to allow parallel processing, vertical scaling, and sharding.

- UTT obtains **accountable privacy** via a novel scheme that implements anonymity budgets [5]. Our implementation is optimized for performance: we use efficient building blocks like Σ-protocols and rerandomizable signatures [34]. The main novelty of our scheme is not in any building block, but in the non-trivial way we combine them in order to get accountable privacy using budget coins in an efficient manner. Having said that, to the best of our knowledge, our use of rerandomizable signatures over rerandomizable commitments and our Σ-protocol for proving correctness of "nullifiers" of spent coins (see §4.3) are both novel building blocks that provide concrete efficiency gains relative to previous solutions.

- We present a novel and arguably clean formulation of UTT as an *ideal functionality*, and **prove that our protocol realizes it**. Along the way, we present a new MPC framework that formalizes, for the first time, the notion of stand-alone

security for reactive functionalities. Our definition relaxes UC security [35] but still accounts for *adaptively* and *adversarily* selected inputs to honest parties. Being compatible with practical building blocks, our framework provides a useful trade-off between practical efficiency and provable security.

- We provide extensive experimental evidence that UTT obtains good **scalability and performance**. We experiment with a decentralized deployment showing that UTT scales well both when implemented as a Byzantine Fault Tolerance State Machine Replication and when implemented using Byzantine Consistent Broadcast. We show scalability in terms of (1) minimal overhead when increasing the number of faults (for $f = 1, 3, 5, 10$); (2) near-linear vertical scalability in terms of increasing the number of cores per server; (3) near-linear vertical scalability in terms of increasing the number of shards. For a non-sharded system using $n = 4$ servers each with 96 cores we obtain a throughput of 903 transactions with accountable privacy per second, with a latency of $39ms$. For a sharded system using $s = 60$ shards, where each shard has a 16 core server we obtain a throughput of $11,351$ transactions with accountable privacy per second.

- We leverage the decentralized threshold-based nature of UTT to also use threshold-based *identity-based cryptography*. This enables us to (1) allow users to use natural names for recipients; (2) reduce onboarding friction (by allowing users to send payment to other users that have not registered yet); and (3) avoid looking up public keys of payees in a directory, which, done naively, would break anonymity (or require a heavy PIR operation).

1.1 Design goals and choices

We design UTT with several goals in mind. Our design choices come with trade-offs, but they help us achieve an efficient, decentralized system, which we evaluate in §6.

"Cash-like" payments. We seek to emulate physical cash, which offers complete anonymity for small payments and auditability for large payments (in many jurisdictions). In line with this philosophy, our transactions are fully anonymous, hiding senders, recipients and amounts. However, we balance this with *accountability*, through so-called *anonymity budgets*, as proposed by Sander and Ta-Shma [5]. Specifically, we limit each user to only send up to $\mathcal{B}$[1] coins anonymously per month. We believe that this design reasonably balances privacy with accountability. In §8, we discuss how our design can also be changed to support other balances such as limits on the amounts of payments that users can receive.

Decentralization. Our ecash system has several actors: banks, auditors and registration authorities (discussed in §4). To avoid compromise and bootstrap trust, we decentralize these actors using a set of servers via Byzantine fault tolerance (BFT) in a partially synchronous environment [36]. We explore decentralization both via totally ordered BFT state machine replication (SMR), and via a Byzantine-consistent broadcast (BCB) primitive [37], which previous work applied to non-private payments [38–40]. In addition, we implement *horizontal scaling* via a two phase *sharding* scheme via Byzantine-consistent broadcast (BCB). In this work, the servers that implement the actors do so via a *permissioned* proof-of-membership mechanism. Our system could be adopted to a setting where the servers are dynamically changing and do so via a *permissionless* proof-of-stake mechanism (see §8).

Fast, modular cryptography. The abstract UTT design allows modularity of the cryptographic building blocks. For example, one can change the *zero-knowledge range proofs* without changing the rest of the scheme. When implementing UTT, we meticulously consider each building block and use the most efficient version we could find. We use Σ*-protocols* in most places. We implement the nullifier using a *pseudo random function* and a novel Σ-protocol. We use PS *rerandomizable signatures over commitments* to sign coins. We discuss alternative designs based on algebraic message authentication codes [41] in §2.

Detecting majority corruptions by users. UTT's security is based on the assumption that at most f servers are compromised. In this work we do not address the case that there is an adversary that controls more than f servers. One desirable property that our current scheme does not support is the ability of users to detect minting violations. This property, first defined by [4] and later implemented in [15],

[1] We often highlight newly-defined symbols in blue.

protects validating users from an adversary that controls any number of servers. We consider adding this property as future work (see discussion in §8).

Setup assumptions. UTT requires a trusted setup. Our system requires a distributed key generation (DKG) protocol [42] for several threshold signature systems, and a "powers-of-tau" ceremony [43] for our range proofs. We view the implementation of the setup assumptions (DKG and "powers-of-tau") as future work. A range proof with public setup [44] avoids this ceremony (see §8).

User-friendly payment addresses. Current schemes do not make it easy for senders to look up the *payment address* of a recipient and pay them. This often results in funds being sent to non-existing addresses and lost. (Indeed, a company was started on this premise [45].) While a public-key directory (PKD) such as Keybase [46] could be used to look up the recipient's address, this would leak who is being paid to the directory. As a result, we find it interesting and natural to explore user-friendly addresses, such as phone numbers, email addresses, or national identification numbers, using *identity-based cryptography (IBC)*.

Beyond being user-friendly and anonymous, IBC makes it possible to recover lost keys and to retroactively audit, which we leave as future work. (Checks and balances would be needed for this.) Furthermore, *threshold-issuance* IBC removes single points of failure (see §3.1). On the other hand, dealing with key-theft in IBC can be challenging, requiring us to maintain a non-anonymous list of stolen identities (see §8) Should this be considered unacceptable, we can fall back to normal public-key encryption, but only at the cost of reintroducing a public-key distribution problem, and thus an anonymity problem.

Modeling and analysis UTT was designed in conjunction with a security definition of its desired properties, as well as a proof that it complies with this definition. Instead of using a game-based definition, the security definition is similar to those used for MPC protocols: the system is compared with an ideal functionality, and the security proof argues indistinguishability from this ideal functionality. To support ecach, the ideal functionality must be reactive and keep a state.

1.2 Non-goals

Identity verification. For anonymity budgets to work, as proposed in [5], it must be hard to create fake user identities (i.e., *Sybil attacks* must be difficult). Otherwise, malicious users can artificially increase their anonymity budget (see §4.2). Therefore, like other accountable systems [31, 47], we assume an *identity verification process* that prevents attackers from registering multiple times.

Network level anonymity. While an individual anonymous transaction leaks nothing about its sender, recipient or amount, the collective timing and network traffic pattern of multiple transactions can be used, to of deanonymize users. Like in previous work [16], we must assume that Alice can forward her transactions to the BFT system via anonymous networks such as Tor [48], Blinder [49] or HoneyBadgerMPC [50].

1.3 Overview of techniques

The high-level structure of our UTT protocol follows the footsteps of Chaum's seminal work on ecash [1], and of more recent work based on ledgers [4, 12, 16, 47, 51]. UTT uses some standard building blocks such as standard Σ-protocols for equality, and also some new building blocks such as a Σ-protocol for proving correctness of "nullifiers" of spent coins, and a novel randomizing of both the commitment and its signature to obtain efficient proofs of knowledge (see §4.3). UTT generalizes ecash to allow multiple types of tokens and anonymity budgets. UTT's main novelty is the careful and novel combination of these building blocks to achieve a scalable generalized ecash solution.

Coins as signed commitments. We represent a *coin* as a *cryptographic commitment* ccm to three values: the owner's *public identifier*, denoted by pid (e.g., a phone number), the value of the coin v and a *serial number* sn used to prevent double spending: *i.e.,* $\mathsf{ccm} = \mathsf{CM.Com}(\mathsf{pid}, \mathsf{sn}, v)$. (We abuse notation here and do not specify the randomness of the commitment as part of the input.) To issue the coin, the bank digitally signs its commitment as $\sigma = \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm})$, where bsk is the bank's secret key for a *rerandomizable signature scheme*. In practice, the bank is a BFT system and bsk is secret-shared amongst the BFT *replicas*. (We more carefully define $\mathsf{CM.Com}$ and $\mathsf{RS.Sign}$ in §3.1.)

Anonymously spending a coin. When spending her coin, a user *Alice* does *not* reveal her original ccm with signature σ, which the bank issued and would recognize. Instead, Alice picks randomizers Δr and Δu and *rerandomizes* her coin **and** its signature as as $\mathsf{ccm}' = \mathsf{CM.Rerand}(\mathsf{ccm}; \Delta r)$ and $\sigma' = \mathsf{RS.Rerand}(\sigma; \Delta r, \Delta u)$. As a result, the bank cannot link the old coin with this newly rerandomized one, which guarantees anonymity. Nonetheless, the rerandomized signature will verify the rerandomized commitment!

At the same time, Alice needs to anonymously indicate in the transaction that she is sending her coin to Bob, who has a public identifier pid_B. For this, Alice includes a commitment $\widehat{\mathsf{ccm}}_B = \mathsf{CM.Com}(\mathsf{pid}_B, 0, v)$ to Bob's future coin, excluding its serial number. The bank will homomorphically add in a random serial number sn_B, obtaining $\mathsf{ccm}_B = \widehat{\mathsf{ccm}}_B \cdot \mathsf{CM.Com}(0, \mathsf{sn}_B, 0)$, which it will sign assuming the transaction passes various checks such as *no double spends*, *authorization* and *value preservation*. Importantly, Alice's transaction encrypts the value v for Bob using identity-based encryption (IBE). Although Bob's sn_B is included publicly in the transaction, Bob will never have to reveal it again, which preserves anonymity.

No double spends. To prevent double spends, Alice will derive a unique *nullifier* for her coin. The bank can then check a *nullifier list* for which nullifiers are spent. Note that Alice cannot anonymously reveal her coin's serial number (SN) as the nullifier since the SN is public, having been previously picked by the bank. Instead, Alice will compute the nullifier as the output of a *pseudo-random function (PRF)* over the SN, similar to previous work [16,25,52]. Our key contribution here is a faster mechanism for Alice to prove (and for the bank to verify) the correctness of her nullifier.

Authorizing spends. Clearly, only Alice should be able to spend her coin. However, since coin serial numbers are bank-picked and Alice's pid and coin value are guessable, this begs the question: *"What secret information can Alice use to authorize the transfer of her coin?"* Here, Alice can use her PRF key to implicitly sign her transaction. Specifically, Alice implicitly signs while proving the correctness of her nullifier using Σ-protocols (via the Fiat-Shamir transform [53]).

Lastly, when sending her coin of value v to Bob, Alice could maliciously create a coin commitment for Bob with value $v' > v$, thereby creating money out of thin air. We tackle this next.

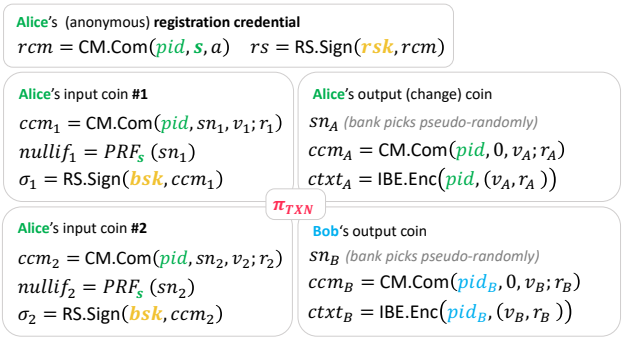

Figure 1: Overview of a UTT transaction from Alice, who has coins of value v_1 and v_2. Alice sends v_B coins to Bob and gets v_A coins back as change. A coin is a commitment ccm_i to its owner's pid, its serial number sn_i and its value v_i. This commitment has a rerandomizable signature σ_i under the bank's bsk. Each input coin ccm_i is uniquely identified by its *nullifier* nullif_i, which is used to mark it as spent. nullif_i is computed as a PRF under Alice's PRF secret key s on the coin's serial number sn_i. Alice's registration credential $(\mathsf{rcm}, \mathsf{rs})$, signed under a registration authority's rsk, proves that the correct s was used to derive nullif_i. Alice creates coin commitments ccm_A (to her change coin) and ccm_B (to Bob's coin), while the bank homomorphically adds in a random serial number to these commitments. Here, (a, r_1, r_2, r_A, r_B) are commitment randomizers.

Combining multiple coins. If Alice has coins of value v_1 and v_2, she should be able to send Bob v_B coins and give herself a change of $v_A = (v_1 + v_2) - v_B$ coins. This *2-to-2 transaction* type, where Alice spends two *input coins* and creates two new *output coins*, is central to our design; see Fig. 1. However, Alice must now prove *value preservation*, or that $v_1 + v_2 = v_A + v_B$. Otherwise, as hinted before, Alice can maliciously create coins out of thin air. Furthermore, as shown in previous work [16, 54], when values are encoded as elements of $\mathbb{Z}_p$, proving that the above equality holds modulo p is not sufficient, so Alice gives an additional *zero-knowledge range proof (ZKRP)* on the (committed) output values v_A and v_B.

Anonymity budgets. To ensure that each user can only send up to $\mathcal{B}$ coins per month, we issue a single *budget coin* of denomination $\mathcal{B}$ to each user every month [5]. We change the transaction to additionally take this budget coin as its third input and give budget coin change as its third output. Crucially, the transaction proves (also via a range proof) that

the amount transferred to Bob does not exceeded the input budget amount.

Preventing Sybils using registration. To prevent Sybil attacks, we use a *registration authority (RA)* to allow new users into the system after they pass an identity verification process as per §1.2. Specifically, each user is given an anonymous *registration credential* rcm signed under the RA's secret key rsk, also using a rerandomizable signature schemes. This credential anonymously links that user's pid with their PRF key s. In addition to anonymously proving that a coin's owning user is registered, the credential also helps prove correctness of nullifiers by anonymous linking a coin's pid with the PRF key s used to compute said nullifier.

2 Related work

We look at related work from three angles. First, we survey *theoretical foundations* of anonymous payments, which are typically not concerned with implementation efficiency. Then, we survey *cryptographic implementations* of anonymous payments, which typically microbenchmark performance but do not build or macrobenchmark a decentralized BFT system. Lastly, we survey *decentralized BFT implementations* of anonymous payment systems. Our work falls into this latter category and we believe it to be a novel design point in this space. Below, we often distinguish between systems which offer k-anonymity for senders and/or recipients and fully-anonymous systems.

Pointcheval-Sanders (PS) signatures. Our work builds upon PS signatures [34] to blindly sign coins. An alternative approach worth exploring would rely upon algebraic MACs and zero-knowledge proofs-of-knowledge (ZKPoKs) [41]. However, it appears difficult to make public verification of such MACs via ZKPoKs as fast as PS verification. Initially, PS signatures based their security on a new assumption proved secure in the generic group model (GGM). However, recent work shows PS security reduces to simple variants of the Strong Diffie-Hellman assumption [55].

Theoretical foundations. Chaum introduced blind signatures and showed how they give rise to ecash [1]. Sander and Ta-Shma first proposed anonymity budgets for limiting the amounts sent [5]. They are also first to propose a ledger-based design with zero-knowledge membership proofs of valid coins as an alternative to blind signatures [4]. Maxwell proposed using homomorphic commitments with correlated randomness to hide transferred amounts [54]. Camenisch et al. [8] propose having each merchant specify a per-user spending limit, but they do not hide recipient identities nor support arbitrary-denomination coins. Groth and Kohlweiss [56] propose *one-out-of-many proofs* and use it to build an ecash scheme. Garman et al. [21] coin the notion of an accountable DAP and give various accountability notions for anonymous payments, including anonymity budgets, and sketch how they can be achieved using zkSNARKs. They also give a new simulation-based security definition that improves over [16] and sketch how to handle accountability in their definition. Mimblewimble [57] shows how to "compress" a UTXO-based ledger such as Bitcoin into a single *super transaction*. This gives full anonymity, but only against attackers who see the ledger post-compression, and k-anonymity for attackers who are sniffing the network or getting multiple snapshots of the ledger. Damgård et al. [28] propose an identity layer for ledger-based payment systems that is amenable to auditability but only sketch how it integrates with anonymous payments. Barki and Gouget [26] explore trade-offs between privacy and accountability in traceable ecash schemes, but do not investigate anonymity budgets.

Cryptographic implementations. Solidus [22] uses publicly-verifiable oblivious RAMs (PVORAMs) for anonymity and has the advantage of only storing state linearly-sized in the number of users rather than transactions. It also enables auditors to decrypt user's balances. However, in Solidus each user's balance is visible to their associated bank. Also, the PVORAM primitive is expensive: in a setting of 12 banks with 2^{15} users each, it supports only 9 transactions per second. PRcash [23] is partially-anonymous, with validators having some "limited ability to link transactions with the same recipient issued within a short period of time." PRcash also provides accountability by limiting *incoming* payments (whereas our work limits outgoing payments). PRcash does not have a formal security analysis. zkLedger [24] gives anonymous payments from Σ-protocols and supports certain third-party auditing functionality, but only handles a small numbers of users who pay each other. MiniLedger [30] improves upon the storage overheads of zkLedger, but not the number of users. Coconut [58] gives a threshold variant of Pointcheval-Sanders (PS) signatures, which this work slightly

modifies, and sketches a tumbler for anonymous payments, but does not investigate accountability. Androulaki et al. [25] also use PS signatures to build anonymous payments, but do not appear to leverage re-randomizability when creating a transaction and instead rely on proofs-of-knowledge of a signature. They only microbenchmark their scheme which, when compared to this work, has 31× slower transaction creation and 90× slower validation. Without transaction auditability, these slowdowns would be 10× and 33×, respectively. Quisquis [59] addresses the unbounded growth of the state in systems like Monero and Zerocash [16]. However, Quisquis only offers k-anonymity for senders and recipients. Bogatov et al. [29] investigate using delegatable anonymous credentials for registering users inside an auditable payment scheme. Zether [60] allows users in an Ethereum smart contract to pay each other while fully hiding the amounts and partially-hiding the involved parties using k-anonymity. Zexe [61] proposes a new *Decentralized Private Computation (DPC)* notion, which they use to build privacy-preserving smart contracts, including a scheme for transferring user-defined assets. However, Zexe does not tackle accountability for anonymous payments and is not lightweight due to zkSNARKs. PGC [27] hides denominations but not user identities in a transaction, which are pseudonymous like in Bitcoin. However, PGC achieves three accountability notions: limiting amounts sent or received, paying a certain tax rate and disclosing a transaction's amount. Veksel [62] builds anonymous payments from commit-and-prove SNARKs for RSA accumulators, Bulletproofs [44] and SNARK-friendly elliptic curves. Veksel does not provide auditability. Concomitant with our work, Platypus [31] also proposes anonymous payments with anonymity budgets for incoming payments. Additionally Platypus can limit the total balance of users. However, their payment protocol is *interactive* as it requires the recipient to send their account information to the sender. Platypus microbenchmarks a zkSNARK-based and a Σ-protocol-based instantiation of their protocol. Compared to the SNARK one, UTT is 8× faster when creating TXNs and 27× slower when validating them, which we believe can be improved (see §8). The Σ-protocol instantiation also uses PS signatures, but it does not seem to support accountability and it is not actually described in the paper.

BFT implementations. Zerocoin [12] extends Bitcoin with ecash-like privacy using RSA accumulators. Monero [63, 64] extends Maxwell's confidential transactions [54] by fully-hiding recipients via *stealth addresses* and partially-hiding senders (k-anonymity) via *ring signatures*. Zerocash [16] builds decentralized anonymous payments using zkSNARKs. None of these systems provide accountability and all use proof-of-work consensus which results in low throughput.

3 Preliminaries

Notation. We assume Type III bilinear pairings $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ over groups of prime order p. We use the multiplicative notation for $\mathbb{G}_1$ and $\mathbb{G}_2$ throughout the paper. Let $[n] = \{1, 2, \ldots, n\}$. Let $a \leftarrow\$ S$ denote uniformly sampling a from a set S. We often use bolded variables to denote vectors as $\mathbf{m} = [m_1, m_2, \ldots, m_\ell]$. Let $\mathbf{g} \in \mathbb{G}^\ell$ and $(\mathbf{x}, \mathbf{y}) \in (\mathbb{Z}_p^\ell)^2$. Then, $\mathbf{g}^{\mathbf{x}} = g_1^{x_1} g_2^{x_2} \cdots g_\ell^{x_\ell}$ and $\mathbf{x} + \mathbf{y} = [x_1 + y_1, \ldots, x_\ell + y_\ell]$.

3.1 Cryptographic building blocks

We use *homomorphic commitments* to vectors $\mathbf{m}$ computed as $\mathsf{cm} \leftarrow \mathsf{CM.Com}([m_1, \ldots, m_\ell]; r)$ where r denotes the randomness. These commitments can be *rerandomized* via $\mathsf{cm}' \leftarrow \mathsf{CM.Rerand}(\mathsf{cm}; \Delta r)$, where $\Delta r \leftarrow\$ \mathbb{Z}_p$, which prevents an attacker from linking an old, known cm with a fresh, rerandomized cm'. Homomorphism means that $\mathsf{CM.Com}(\mathbf{m}_1; r_1) \cdot \mathsf{CM.Com}(\mathbf{m}_2; r_2) = \mathsf{CM.Com}(\mathbf{m}_1 + \mathbf{m}_2; r_1 + r_2)$. We assume familiarity with notions of *binding* and *hiding* for commitments, which we do not repeat here.

Pedersen commitments. In particular, we use "dual" *Pedersen commitments* [65] computed over both $\mathbb{G}_1$ and $\mathbb{G}_2$ using a public *commitment key* $\mathsf{ck} \stackrel{\mathsf{def}}{=} (\mathsf{ck}_1, \mathsf{ck}_2) = ((g, \mathbf{g}), (\tilde{g}, \tilde{\mathbf{g}}))$ as:

$$(\mathsf{cm}, \widetilde{\mathsf{cm}}) = \mathsf{CM.Com}_{\mathsf{ck}}(\mathbf{m}; r) = (\mathbf{g}^{\mathbf{m}} g^r, \tilde{\mathbf{g}}^{\mathbf{m}} \tilde{g}^r), r \leftarrow\$ \mathbb{Z}_p$$

We can rerandomize any dual commitment $(\mathsf{cm}, \widetilde{\mathsf{cm}})$, as $(\mathsf{cm} \cdot g^{\Delta r}, \widetilde{\mathsf{cm}} \cdot \tilde{g}^{\Delta r})$, where $\Delta r \leftarrow\$ \mathbb{Z}_p$. We often abuse notation and let $\mathsf{cm} = \mathsf{CM.Com}_{\mathsf{ck}_1}(\mathbf{m}, r) = \mathbf{g}^{\mathbf{m}} g^r \in \mathbb{G}_1$. (We proceed analogously for $\mathbb{G}_2$.) We give full algorithms in Fig. 3 in the appendix.

Pointcheval-Sanders (PS) signatures. We often sign dual Pedersen commitments $(\mathsf{cm}, \widetilde{\mathsf{cm}})$ using a *rerandomizable signature (RS) scheme* by Pointcheval and Sanders [34] as:

$$\sigma = \mathsf{RS.Sign}_{\mathsf{ck}}(\mathsf{sk}, \mathsf{cm}; u) = (g^u, (\mathsf{sk} \cdot \mathsf{cm})^u), u \leftarrow\$ \mathbb{Z}_p \quad (1)$$

Note that the signing algorithm only needs the $\mathsf{cm} \in \mathbb{G}_1$ part of the dual commitment. Here, $\mathsf{sk} \in \mathbb{G}_1$ is the *secret key*, u is the signing randomness and ck is the commitment key. However, the verification algorithm $\mathsf{RS.Ver}_{\mathsf{ck}}(\mathsf{vk}, (\mathsf{cm}, \widetilde{\mathsf{cm}}), \sigma)$ needs $\widetilde{\mathsf{cm}} \in \mathbb{G}_2$ and, of course, the *verification key* vk for sk. Lastly, it is crucial not to reuse u for two different signatures, which would lead to a forgery. We detail (single-signer) PS signatures in Fig. 4 in the appendix.

A signature σ on cm can be rerandomized into a signature σ' on a rerandomized $\mathsf{cm}' = \mathsf{CM.Rerand}(\mathsf{cm}; \Delta r)$ via $\sigma' = \mathsf{RS.Rerand}(\sigma, \Delta r, \Delta u)$, where $\Delta u \leftarrow\$ \mathbb{Z}_p$. As a consequence, a signer cannot link the old, known signed commitment (cm, σ) with the fresh, rerandomized (cm', σ').

In §4, we abuse the notation and use $\mathsf{RS.Sign}(\mathsf{sk}, \mathsf{cm}; u)$ and $\mathsf{RS.Ver}(\mathsf{vk}, \mathsf{cm}, \sigma)$ when describing our high-level design. Later, in §5, we use the precise notation that accounts for ck and $\widetilde{\mathsf{cm}} \in \mathbb{G}_2$ when describing low-level details.

Threshold PS. To thresholdize PS, the key pair $(\mathsf{sk}, \mathsf{vk})$ is (t, n)-secret-shared amongst the *signers*, such that each signer i has a *share secret key* sk_i and a *share verification key* vk_i. As a result, only subsets of $\geq t+1$ signers can collaborate to produce a signature that verifies under vk. A key challenge is generating the per-signature private randomness u from Eq. 1 in a distributed manner so that no single signer learns u. Here, we slightly modify previous work [58], which gives a fast approach that avoids expensive *distributed key generation (DKG)* protocols for each u.

Our threshold PS variant works as follows. First, each signature request on a committed vector $\mathbf{m}$ is associated with a pre-agreed upon, random $h \in \mathbb{G}_1$ such that nobody knows the discrete log between h and g. (We detail later how pre-agreement on h happens in our application.) Second, a *client* sends (to each signer) individual commitments $\mathsf{cm}_k = h^{m_k} g^{r_k}, r_k \leftarrow\$ \mathbb{Z}_p$ to the ℓ messages in $\mathbf{m}$ with a *zero-knowledge proof-of-knowledge (ZKPoK)* of opening π_k^{zkpok} (see App. B.1). Third, each signer produces their *signature share*:

$$[\sigma^*]_i = \mathsf{RS.ShareSign}_{\mathsf{ck}}(\mathsf{sk}_i, ((\mathsf{cm}_k, \pi_k^{\mathsf{zkpok}})_{k\in[\ell]}); h) \tag{2}$$

Fourth, the client verifies each signature share via $\mathsf{RS.ShareVer}_{\mathsf{ck}}(\mathsf{vk}_i, (\mathsf{cm}_k, \pi_k^{\mathsf{zkpok}})_{k\in[\ell]}, [\sigma^*]_i; h)$ and identifies a set S of $t+1$ valid ones. Fifth, the client aggregates them as follows:

$$\sigma \leftarrow \mathsf{RS.Aggregate}_{\mathsf{ck}}(([\sigma^*]_i)_{i\in S}, [r_1, \ldots, r_k]) \tag{3}$$

Importantly, the final threshold signature verifies as $\mathsf{RS.Ver}(\mathsf{vk}, (\mathsf{cm}, \widetilde{\mathsf{cm}}), \sigma)$, where $(\mathsf{cm}, \widetilde{\mathsf{cm}}) = \mathsf{CM.Com}_{\mathsf{ck}}(\mathbf{m}; 0)$ has randomness set to zero (but can be rerandomized after). We give full details in Fig. 5 in the appendix.

Identity-based encryption (IBE). We use a key-private, CCA-secure, threshold-issuance variant of the Boneh-Franklin IBE [66, 67], fully described in Fig. 6 in the appendix. In IBE, every user j has a public identifier pid_j (e.g., their phone number). The IBE authority has an *IBE key-pair* $(\mathsf{msk}, \mathsf{mpk})$ and issues to each user j their *decryption secret key* $\mathsf{dsk}_j \leftarrow \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{pid}_j)$. To deal with a malicious authority, *threshold-issuance* IBE secret shares the msk amongst n *cothoroties* [68]. This way, $t+1 \leq n$ or more cothorities must collaborate to issue user j's dsk_j and no subset of $\leq t$ malicious cothorities can learn dsk_j. To encrypt m for a user with pid_j, one computes $\mathsf{ctxt} \leftarrow \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{pid}_j, m)$. To decrypt, user j computes $m \leftarrow \mathsf{IBE.Dec}(\mathsf{dsk}_j, \mathsf{ctxt})$. When using CCA-secure IBE, this returns $\perp$ if the ciphertext is invalid.

Range proofs. We use a pairing-based range proof protocol by Boneh et. al [69] based on constant-sized polynomial commitments, which we adapt to work for Pedersen commitments in App. B. A range proof for a committed $v \in [0, 2^N)$ is computed as $\pi^{\mathsf{range}} \leftarrow \mathsf{ZKRP.Prove}(\mathsf{rpp}, v, z)$, where rpp are public parameters for proving $[0, 2^N)$ ranges. The proof can be verified against the commitment $\mathsf{cm} = \mathsf{CM.Com}(v; z)$ as $\mathsf{ZKRP.Ver}(\mathsf{rpp}, \mathsf{cm}, \pi^{\mathsf{range}})$. The scheme is fully described in Fig. 7 in the appendix.

Discrete-log-based ZK proofs. We often use *zero-knowledge arguments of knowledge (ZKAoKs)* for discrete logs (DLs), where a computationally bounded *prover* convinces a *verifier* that a certain *NP relation* $\mathcal{R}(\mathbb{x}; \mathbb{w})$ holds between the DLs of group elements. Here, $\mathbb{x}$ is referred to as the *statement* and $\mathbb{w}$ as the *witness*. Importantly, the prover proves that it *knows* a $\mathbb{w}$ such that $\mathcal{R}(\mathbb{x}; \mathbb{w}) = 1$. Lastly, secret knowledge of the witness $\mathbb{w}$ can actually be used to *implicitly* sign a message m as part of the proof for $\mathcal{R}(\mathbb{x}; \mathbb{w})$. If so, we refer to such an argument as a *zero-knowledge signature of knowledge (ZKSoK)* [70]. For brevity, we refer the reader to [56, 59, 70] for well-known definitions of ZKAoKs/ZKSoKs.

Example: A relation could be that $\mathsf{cm}_1 = g_1^a g_2^b g_3^c g^r$ and $\mathsf{cm}_2 = h^a g^z$ both commit to the same a. The statement would be $\mathbb{x} = (\mathsf{cm}_1, \mathsf{cm}_2)$, while the openings would be the witness $\mathbb{w} = (a, b, c, r, z)$. Crucially,

zero-knowledge guarantees the verifier learns nothing about $\mathbb{w}$ (beyond cm_1 and cm_2 being related as described above).

Notation. When proving a particular relation $\mathcal{R}_{\mathsf{rel}}(\mathbb{x}; \mathbb{w})$ holds, we often use $\mathsf{ZK.Prove}_{\mathsf{rel}}(\mathbb{x}; \mathbb{w})$ and $\mathsf{ZK.Ver}_{\mathsf{rel}}(\mathbb{x})$ as notation for the prover and verifier algorithms.

4 UTT design

In this section, we break down our high-level design from Fig. 1 into a lower-level one built from commitments, rerandomizable signatures, Σ-protocols and range proofs; see Fig. 2. For ease of exposition, we focus on how these building blocks fit together and defer *efficiently* instantiating them to §5.

4.1 Actors

Our system consists of: (1) *users*, who want to transact, (2) a *bank*, which issues new coins and blindly validates transactions, (3) an *auditor* who grants users anonymity budgets and audits transactions when users run out of budget, and (4) a *registration authority (RA)*, which is responsible for registering users to prevent Sybil attacks on anonymity budgets. For ease of exposition, we describe the bank, auditor and RA as single, honest-but-curious servers, and we decentralize them in §5.

Bank. The bank has a rerandomizable signature key-pair (bsk, bvk) which it uses to sign new coins. The bank's *state* consists of a *ledger* of valid transactions and a *nullifier list* which keeps track of spent or "nullified" coins.

Users. Users store coins in their *wallet*. Every user has a *public identifier* pid (e.g., phone number, email address or national identity number) and two secrets: a *PRF key* s, which allows them to spend their coins, and a *decryption secret key* dsk, which allows them to receive coins.

Registration authority (RA). The RA prevents attackers from arbitrarily creating fake user identities in our system (i.e., sybils) and artificially increasing their anonymity budget. The RA registers new users who passed an identity verification process as per §1.2. Specifically, it issues anonymous credentials using a rerandomizable signature key-pair (rsk, rvk). Furthermore, the RA manages an IBE key-pair (msk, mpk) and issues each user their dsk.

Auditor. Each user receives their anonymity budget from the *auditor*, who signs so-called *budget coins*, discussed in §4.6, using a rerandomizable signature key-pair (ask, apk), which the bank also knows. When a user "runs out of anonymity budget", they can still create transactions that are anonymous to the bank, but not to the auditor, who must clear them based on a customizable auditing policy, which is outside the scope of this work. In future work the auditor may have a governance mechanism that allows it to change its anonymity budget policy.

4.2 Anonymous registration credentials

To register, a user with pid must first pass an identity verification process (see §1.2). Then, the RA can issue them a decryption secret key dsk and a *registration credential* consisting of a signature rs on a *registration commitment* rcm to their pid and PRF key $s = s_1 + s_2$, where the user picks s_1 and the server picks s_2. First, the user lets $\mathsf{rcm}_1 \leftarrow \mathsf{CM.Com}([0, s_1]; a)$, where $(a, s_1) \leftarrow\$ \mathbb{Z}_p^2$ and sends $(\mathsf{pid}, \mathsf{rcm}_1)$ to the RA, together with a ZKPoK of the opening $[0, s_1]$ (see App. B.1). Second, the RA picks $s_2 \leftarrow\$ \mathbb{Z}_p$, and lets:

$$\mathsf{rcm}_2 \leftarrow \mathsf{CM.Com}([\mathsf{pid}, s_2]; 0) \tag{4}$$

$$\mathsf{rcm} \leftarrow \mathsf{rcm}_1 \cdot \mathsf{rcm}_2 = \mathsf{CM.Com}([\mathsf{pid}, s], a) \tag{5}$$

$$\mathsf{rs} \leftarrow \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b), \text{ where } b \leftarrow\$ \mathbb{Z}_p \tag{6}$$

$$\mathsf{dsk} \leftarrow \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{pid}) \tag{7}$$

Third, the RA sends $(\mathsf{rcm}, \mathsf{rs}, \mathsf{dsk}, s_2)$ to the user who computes his PRF key as $s = s_1 + s_2$. Of course, dsk and s_2 are privately-sent (e.g., via TLS). Note that while the RA learns who is registering, it does not learn their PRF key s.

Why should users have to register? Whenever a user asks for their monthly budget, they must present their registration credential $(\mathsf{rcm}, \mathsf{rs})$ and provably-reveal their committed pid (see App. B.1), which allows the auditor to check if that pid has already been issued budget. This prevents Sybil attacks on anonymity budgets. Even though users reveal their pid's during budget issuance, anonymity could be preserved if users instead proved (in ZK) that their pid has not already been issued budget. (We leave this to future work.)

4.3 Rerandomizable coins

Signing coins. We commit to a coin owner's pid, its value v and serial number sn as:

$$\mathsf{ccm} = \mathsf{CM.Com}([\mathsf{pid}, \mathsf{sn}, v]; r), \text{ where } r \leftarrow\$ \mathbb{Z}_p \quad (8)$$

Since the bank will pick the sn pseudo-randomly (see §4.5), transactions (TXNs) maintain privacy by revealing a *nullifier* (i.e., a PRF over the sn) instead of the publicly-known sn. The bank computes a *coin signature* σ directly over the commitment ccm, which hides all coin details from the bank:

$$\sigma = \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm}; u), \text{ where } u \leftarrow\$ \mathbb{Z}_p \quad (9)$$

This way, anyone can verify if a committed coin was validly-issued by the bank via $\mathsf{RS.Ver}(\mathsf{bvk}, \mathsf{ccm}, \sigma)$ (see Fig. 4).

Rerandomizing coins. A key ingredient of our privacy-preserving design is that a user can take an old coin (ccm, σ), which the bank knows of, and rerandomize it as a new coin (ccm', σ'). This way, if a transaction spends the rerandomized coin (ccm', σ'), the bank cannot tell that this coin is actually spending the old coin. Specifically, the user randomly picks $(\Delta r, \Delta u) \leftarrow\$ \mathbb{Z}_p^2$ and rerandomizes as:

$$\begin{aligned} \mathsf{ccm}' &\leftarrow \mathsf{CM.Rerand}(\mathsf{ccm}; \Delta r) \\ \sigma' &\leftarrow \mathsf{RS.Rerand}(\sigma; \Delta r, \Delta u) \end{aligned} \quad (10)$$

Importantly, the new σ' signature still verifies against the rerandomized ccm.

Nullifiers, or how to mark coins as spent. Since serial numbers are picked by (and known to) the bank, coins cannot be anonymously marked as spent by their serial number. Instead, when including a coin (ccm, σ) in a transaction, the owner reveals a *nullifier* computed as a PRF over the coin's serial number: $\mathsf{nullif} = \mathsf{PRF}_s(\mathsf{sn})$. It will be useful to additionally reveal a separate *value commitment* $\mathsf{vcm} = \mathsf{CM.Com}([0, 0, v]; z)$, where $z \leftarrow\$ \mathbb{Z}_p$ (we explain why in §4.4). Importantly, the owner argues correctness of nullif and vcm by proving, in zero-knowledge, that the following relation $\mathcal{R}_{\mathsf{split}}(\mathsf{ccm}, \mathsf{rcm}, \mathsf{nullif}, \mathsf{vcm}; \mathsf{pid}, \mathsf{sn}, v, r, s, a, z)$ holds:

$$\left(\begin{array}{ll} \mathsf{ccm} = \mathsf{CM.Com}([\mathsf{pid}, \mathsf{sn}, v]; r) & \wedge \\ \mathsf{rcm} = \mathsf{CM.Com}([\mathsf{pid}, s]; a) & \wedge \\ \mathsf{nullif} = \mathsf{PRF}_s(\mathsf{sn}) & \wedge \\ \mathsf{vcm} = \mathsf{CM.Com}([0, 0, v]; z) & \end{array}\right) \quad (11)$$

We call this a *splitproof*, denote it by π^{split} and instantiate it *efficiently* in §5.

4.4 Anonymous-but-unaccountable transactions

As a warm-up, we first explain our anonymous-but-*unaccountable* transactions format from Fig. 2 (i.e., *no* anonymity budgets). Throughout the rest of this section, we consider *Alice* with pid_A, PRF key s_A and registration credential $(\mathsf{rcm}, \mathsf{rs})$, who has two coins $(\mathsf{pid}_A, \mathsf{sn}_i, v_i)_{i\in\{1,2\}}$, committed and signed as $(\mathsf{ccm}_i, \sigma_i)_{i\in\{1,2\}}$. Alice wants to pay v_B coins to *Bob* with pid_B and get v_A coins back as change, of course subject to $v_1+v_2 = v_A+v_B$. We *incrementally* design an anonymous-but-unaccountable transaction format for this task below. (We deal with transaction fees in App. A.)

Step 1: Input commitments. First, Alice includes her anonymous registration credential in a transaction tx. Second, she includes her signed *input coin commitments* ccm_i, their nullifiers, and separate value commitments $\mathsf{vcm}_i = \mathsf{CM.Com}([0, 0, v_i]; z_i)$ with randomness z_i she picked. So far, the transaction is:

$$\mathsf{tx} = (\mathsf{rcm}, \mathsf{rs}), \langle(\mathsf{ccm}_i, \sigma_i), \mathsf{vcm}_i, \mathsf{nullif}_i, \pi_i^{\mathsf{split}}\rangle_{i\in\{1,2\}} \quad (12)$$

Importantly, Alice *rerandomized* her anonymous credential, as well as her coins and their signatures (via $\mathsf{CM.Rerand}$ and $\mathsf{RS.Rerand}$ as per §4.3) so that neither the bank nor the RA recognizes them from when they issued them in the past. Alice will also need to *anonymously* prove that *she* owns her coins, which she will actually do by implicitly signing tx via her π^{split} (see Step 4 below).

Step 2: Output commitments. Next, Alice needs to anonymously indicate she is sending v_B coins to Bob with pid_B and v_A coins to herself back as change. For this, Alice picks randomness (t'_j, z'_j) and creates separate *output value commitments* and *identity commitments* to the amount and identity of each recipient $j \in \{A, B\}$:

$$\mathsf{icm}_j = \mathsf{CM.Com}([\mathsf{pid}_j, 0, 0]; t'_j) \quad (13)$$

$$\mathsf{vcm}_j = \mathsf{CM.Com}([0, 0, v_j]; z'_j) \quad (14)$$

Alice also uses IBE to encrypt recipient j's coin value and randomness, which they will need when receiving her payment:

$$\mathsf{ctxt}_j = \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{pid}_j, (v_j, z'_j, t'_j)) \quad (15)$$

Importantly, the anonymous IBE scheme guarantees that ctxt_j does not leak anything about the recipient

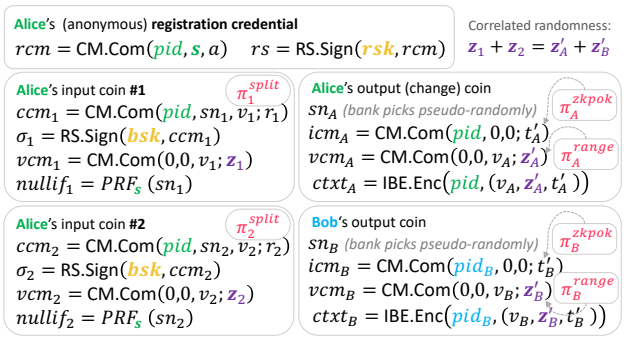

Figure 2: More fine-grained view of an anonymous-but-unaccountable UTT transaction. Alice is sending two coins of values v_1 and v_2, giving v_B coins to Bob and v_A coins back to herself. Alice provably "splits" each input coin i into a nullif_i and a *value commitment* vcm_i via a *splitproof* π_i^{split}. For each recipient $j \in \{A, B\}$, Alice creates output value commitments vcm_j with range proofs and uses *correlated randomness* to argue that $v_1 + v_2 = v_A + v_B$. Alice also creates *identity commitments* icm_j (with ZKPoKs of opening), which the bank combines with their corresponding vcm_j and serial number sn_j to obtain an output coin commitment and sign it (see §4.5).

pid_j. Next, Alice computes a ZKPoK π_j^{zkpok} of the opening $[\mathsf{pid}_j, 0, 0]$ of each icm_j and a ZK range proof $\pi_j^{\mathsf{range}} = \mathsf{ZKRP.Prove}(v_j, z'_j)$, for each vcm_j. These will be used to ensure Alice does not create coins out of thin air, which we discuss next. The transaction is now extended as:

$$\mathsf{tx} = \begin{cases} (\mathsf{rcm}, \mathsf{rs}), \langle (\mathsf{ccm}_i, \sigma_i), \mathsf{vcm}_i, \mathsf{nullif}_i, \pi_i^{\mathsf{split}} \rangle_{i \in [2]} \\ \langle \mathsf{icm}_j, \pi_j^{\mathsf{zkpok}}, \mathsf{vcm}_j, \pi_j^{\mathsf{range}}, \mathsf{ctxt}_j \rangle_{j \in \{A,B\}} \end{cases} \tag{16}$$

Step 3: Value preservation. Alice must prove that $v_1 + v_2 = v_A + v_B$, or *value preservation*; otherwise, she could create coins out of thin air by inflating her change amount v_A! For this, we use a well-known, efficient approach based on homomorphic commitments with correlated randomness. Specifically, since Alice is the one who picks the randomness of the value commitments, she can correlate it so that $z_1 + z_2 = z'_A + z'_B$. As a result, the homomorphic properties of the commitment can be used to verify value preservation:

$$\mathsf{vcm}_1 \cdot \mathsf{vcm}_2 = \mathsf{vcm}_A \cdot \mathsf{vcm}_B \Leftrightarrow \tag{17}$$

$$\mathsf{CM.Com}([0, 0, v_1 + v_2]; z_1 + z_2) = \tag{18}$$

$$= \mathsf{CM.Com}([0, 0, v_A + v_B]; z'_A + z'_B) \Rightarrow$$

$$v_1 + v_2 \equiv v_A + v_B \pmod{p} \tag{19}$$

Here, p is the prime order of our bilinear groups (see §3). However, note that the equality above holding (mod p) is actually not enough: a malicious Alice could set $v_B = (v_1 + v_2) + 1$ and $v_A = p - 1$. This gives $v_A + v_B = (v_1 + v_2) + p \equiv (v_1 + v_2) \pmod{p}$, yet Alice has maliciously created p additional coins out of thin air! This is why each output value commitment vcm_j comes with a range proof π_j^{range}, which proves that, when viewed as a positive integer, $v_j < 2^N$ (i.e., a predefined maximum number of coins in the system).

Step 4: Authorization (and TXN non-malleability). Clearly, only Alice, who knows the PRF key s_A associated with her pid_A should be able to spend her coins. Furthermore, an adversary should not be able to *maul* Alice's TXN and change the recipients or amounts. We ensure this by using a *zero-knowledge signature of knowledge (ZKSoK)* as our splitproof rather than a plain ZKAoK (see §3.1). As a result, each input's splitproof π_i^{split} implicitly signs *all* the outputs, which proves that Alice is authorized to spend that input's coin and prevents TXN mauling. The signing key can be thought of as Alice's PRF key s_A (and all other secrets in Eq. 11), which only she knows.

4.5 Validating and processing transactions

In this section, we explain how the bank (1) validates our anonymous-but-unaccountable transactions from §4.4 (2) issues new coins and (3) how recipients fetch them. In §4.6, we add support for accountability.

Validating TXNs. Given Alice's tx from Eq. 16, the bank first checks that the sender's registration credential is valid via $\mathsf{RS.Ver}(\mathsf{rvk}, \mathsf{rcm}, \mathsf{rs})$. Importantly, since the bank verifies this signature against the (rerandomized) registration commitment, the bank learns nothing about Alice's identity. Second, the bank checks that each coin was validly issued via $\mathsf{RS.Ver}(\mathsf{bvk}, \mathsf{ccm}_i, \sigma_i), \forall i \in [2]$. Third, the bank checks that the split relation $\mathcal{R}_{\mathsf{split}}(\mathsf{ccm}_i, \mathsf{rcm}, \mathsf{nullif}_i, \mathsf{vcm}_i)$ from Eq. 11 holds via π_i^{split} for each $i \in [2]$. This implicitly verifies Alice's signature on the whole transaction, without leaking Alice's identity as the sender.

Fourth, the bank parses the tx's input $(\mathsf{vcm}_i)_{i \in [2]}$ and output $(\mathsf{vcm}_j)_{j \in \{A,B\}}$ and checks *value preservation* holds modulo p as per Eq. 17. Then, the bank checks each output's range proof, which implicitly proves knowledge of an opening $[0, 0, v_j]$ for vcm_j.

(Note that input vcm_i's do not need a range proof: since input coins were validly-issued, they can safely be assumed to be "in range".) Fifth, the bank checks, via π_j^{zkpok}, that each icm_j is of the form $[\mathsf{pid}_j, 0, 0]$ for some secret pid_j. Lastly, the bank ensures all nullif_i's are *not* in its nullifier list. If so, the bank adds the nullifiers to the set and adds tx to the ledger of valid TXNs.

Issuing new coins. The bank is now ready to *blindly* issue new coins for the recipients, without learning the amounts or identities of the recipients. First, the bank pseudo-randomly picks serial numbers $\mathsf{sn}_j = \mathcal{H}_{\mathsf{sn}}(\mathsf{tx}||j)$ for each output coin $j \in \{A, B\}$, where $\mathcal{H}_{\mathsf{sn}}$ is a collision-resistant hash function (CRHF). Since the bank verified each $(\pi_j^{\mathsf{zkpok}}, \pi_j^{\mathsf{range}})$, it can safely compute the jth *output coin commitment* as:

$$\mathsf{ccm}_j = \mathsf{icm}_j \cdot \mathsf{CM.Com}([0, \mathsf{sn}_j, 0]; 0) \cdot \mathsf{vcm}_j \tag{20}$$
$$= \mathsf{CM.Com}([\mathsf{pid}_j, \mathsf{sn}_j, v_j]; t'_j + z'_j) \tag{21}$$

Specifically, π_j^{zkpok} prevents icm_j from maliciously committing to $(\mathsf{pid}_j, 0, \delta_j)$ with $\delta_j \neq 0$ which, when combining it with vcm_j, would yield a malicious ccm_j with a larger value $v_j + \delta_j$ than what the range proof was over.

The bank learns nothing about the recipient's (pid_j, v_j). Furthermore, even though the bank knows the sn_j's of these new coins, it will not be able to link them to their nullifiers when the coins are later spent (see §4.3). Lastly, the bank signs each coin as $\sigma_j \leftarrow \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm}_j; u_j)$, where $u_j \leftarrow\$ \mathbb{Z}_p$ and updates tx on the ledger with its σ_j's.

Receiving payments. Similar to previous work [16], to receive payments, Bob has to scan the bank's transaction list for new transactions that he missed and check if those transactions paid him. For each new transaction tx (as per Eq. 16) and for each output j in tx, Bob checks if ctxt_j is encrypted for him via $\mathsf{IBE.Dec}(\mathsf{dsk}, \mathsf{ctxt}_j) \neq \perp$, where dsk is his decryption secret key. If so, Bob decrypts his (v_j, t'_j, z'_j) from Eq. 15, recomputes his output's $\mathsf{sn}_j \leftarrow \mathcal{H}_{\mathsf{sn}}(\mathsf{tx}||j)$ and reconstructs his coin commitment $\mathsf{ccm}_j \leftarrow \mathsf{CM.Com}([\mathsf{pid}_j, \mathsf{sn}_j, v_j]; t'_j + z'_j)$. Next, he fetches the coin signature σ_j from the TXN, rerandomizes $(\mathsf{ccm}_j, \sigma_j)$ as per §4.3 and adds it to his wallet. We discuss how users can delegate this linear scan to an untrusted third party in §8.

4.6 Anonymous-and-accountable transactions

Budget coins. Each user gets $\mathcal{B}$ *budget coins* per month, which allow them to send up to $\mathcal{B}$ normal coins every month blindly, without revealing any transaction details to the bank or the auditor. A budget coin resembles a normal coin from §4.3, except it has an *expiration date* expi and is signed using the auditor's ask. For example, at the beginning of the month, a user provably reveals their pid to the auditor using their registration credential (and a ZKPoK; see App. B.1). The auditor then issues them their monthly budget coin:

$$\mathsf{ccm} = \mathsf{CM.Com}([\mathsf{pid}, \mathsf{sn}, \mathcal{B}, \mathsf{expi}]; r), \text{ where } (\mathsf{sn}, r) \leftarrow\$ \mathbb{Z}_p^2$$
$$\sigma = \mathsf{RS.Sign}(\mathsf{ask}, \mathsf{ccm}) \tag{22}$$

Note that the auditor can easily encrypt the coin secrets for the user via $\mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{pid}, (\mathsf{sn}, r))$. For example, if the pid is an email address (or a phone number), this ciphertext can be emailed (or texted) to the user. By default, normal coins are now committed with expiration date set to 0.

(Anonymous) accountable transactions. Assume Alice has her budget coin $\mathsf{ccm}_3 = \mathsf{CM.Com}([\mathsf{pid}_A, \mathsf{sn}_3, v_3, \mathsf{expi}]; r_3)$ and its signature σ_3, as per Eq. 22. We modify our "2-to-2" transactions from §4.4 as follows. First, Alice provides her budget coin $(\mathsf{ccm}_3, \sigma_3)$ as a third input to the TXN and argues in zero-knowledge that (1) its value v_3 exceeds the value v_B sent to Bob and (2) it is not expired. Second, Alice creates a third output (denoted by C) for the bank to issue her budget change $v_C = v_3 - v_B$. To validate the transaction, the bank proceeds as per §4.4, nullifying all input coins, including the budget, and issuing coins as before, including Alice's budget change. (Recall that the bank also knows ask.) Our new "3-to-3" anonymous-**and**-accountable transaction format is:

$$\mathsf{tx}_{\mathsf{acc}} = \begin{cases} (\mathsf{rcm}, \mathsf{rs}), \pi^{\mathsf{budget}}, \mathsf{expi} \\ \langle (\mathsf{ccm}_i, \sigma_i), \mathsf{vcm}_i, \mathsf{nullif}_i, \pi_i^{\mathsf{split}} \rangle_{i \in [3]} \\ \langle \mathsf{icm}_j, \pi_j^{\mathsf{zkpok}}, \mathsf{vcm}_j, \pi_j^{\mathsf{range}}, \mathsf{ctxt}_j \rangle_{j \in \{A,B,C\}} \end{cases} \tag{23}$$

Step 1: Proving budget suffices. Alice must prove in zero-knowledge that (1) she gave enough budget $v_3 \geq v_B$ and (2) that her budget change is exactly $v_C = v_3 - v_B$. First, Alice correlates the randomness of the value commitments to satisfy not only $z_1 + z_2 =$

$z'_A + z'_B$, but also $z_3 = z'_C + z'_B$. This way, similar to Eq. 17, the bank can verify correctness of the budget change via $\mathsf{vcm}_3 = \mathsf{vcm}_C \cdot \mathsf{vcm}_B \Leftrightarrow v_3 \equiv v_C + v_B \pmod p$. Second, Alice includes a range proof π_C^{range} which argues that $v_C \in [0, 2^N)$. As a result, it follows that $v_3 \geq v_B$.

Step 2: Verifying expiration date. To verify the expi of the budget coin, we do the following. Alice actually computes ccm_3 by using 0 as the expiration date **and** proves this using a Σ-protocol (see App. B.1). Note this is compatible with the split relation from Eq. 11, which implicitly only accepts coins with expiration date 0. Next, the bank can homomorphically add in the actual expiration date from the transaction and check the signature verifies via $\mathsf{RS.Ver}(\mathsf{apk}, \mathsf{ccm}_3 \cdot \mathsf{CM.Com}([0,0,0,\mathsf{expi}]; 0))$.

Step 3: Proving single recipient. The above assumes Alice correctly generates the TXN to (1) pay Bob and (2) give herself normal change v_A as well as budget change v_C. However, a malicious Alice might craft the transaction to send the budget change to another user altogether, or to use her change output, which is not subject to budgeting, to pay yet another user. We prevent this using a *budget proof* π^{budget} which argues that Alice used her own pid_A (from rcm) in her icm_A and icm_C identity commitments, which should only be used to give herself change. In other words, Alice proves that the following relation $\mathcal{R}_{\mathsf{budget}}(\mathsf{rcm}, \mathsf{icm}_A, \mathsf{icm}_C; \mathsf{pid}_A, s_A, a, t'_A, t'_C)$ holds:

$$\left(\begin{array}{rl} \mathsf{rcm} = \mathsf{CM.Com}([\mathsf{pid}_A, s_A]; a) & \wedge \\ \mathsf{icm}_A = \mathsf{CM.Com}([\mathsf{pid}_A, 0, 0, 0]; t'_A) & \wedge \\ \mathsf{icm}_C = \mathsf{CM.Com}([\mathsf{pid}_A, 0, 0, 0]; t'_C) & \end{array}\right) \quad (24)$$

Splitting coins. Alice should still be able to split her own coins of value v_1 and v_2 into new coins of value v'_1 and v'_2, without spending any of her anonymity budget. For this, we still allow 2-to-2 transactions as per Eq. 16, but only if they come with a budget proof π^{budget} that argues all inputs and outputs have the same pid: i.e., that $\mathcal{R}_{\mathsf{budget}}(\mathsf{rcm}, \mathsf{icm}_A, \mathsf{icm}_B)$ holds! Note that an adversary can distinguish between transactions that split coins and transaction that actually spend them using budget.

Running out of budget? In the rare case that Alice runs out of budget, she creates a 2-to-2 unaccountable transaction as per Eq. 16 which she *clears* with the auditor. Based on a customizable auditing policy, the auditor can ask Alice to reveal various transaction details. This could range anywhere from provably-revealing everything to the auditor or revealing nothing but the fact that the payment is a donation to a charity. We leave exploring practical policies to future work and for now assume Alice reveals everything to the auditor. The auditor can simply sign the transaction to signal to the bank that it has been audited and can be accepted on the ledger.

5 UTT building blocks

Byzantine fault tolerance and threshold cryptography. To deal with an actively-malicious bank, registration authority or auditor, we decentralize them as a Byzantine fault-tolerant (BFT) state machine replication (SMR) system of $n = 3f+1$ *servers*, which guarantees safety in a partially-synchronous model [36, 71]. (Separate BFT systems for each actor should be used, but we avoid this for ease of exposition.) Every server $i \in [n]$ will store shares $(\mathsf{bsk}_i, \mathsf{rsk}_i, \mathsf{ask}_i, \mathsf{msk}_i)$ of the bsk, rsk, ask and msk, respectively. Furthermore, honest servers will have a consistent view of the nullifier list and the valid transaction ledger. Servers must now collaborate to produce an $f+1$ out of n threshold signature on a coin (or registration commitment), or to issue an encryption SK to a user. This way, no subset of $\leq f$ servers can maliciously issue new coins, register fake users or decrypt user's transactions. We explain below how we must augment our transactions to support threshold signing coins.

Dual Pedersen commitments. We *dual commit* to coins using Pedersen (see §3.1) as:

$$(\mathsf{ccm}, \widetilde{\mathsf{ccm}}) = \mathsf{CM.Com}_{\mathsf{cck}}([\mathsf{pid}, \mathsf{sn}, v, \mathsf{expi}]; r) \quad (25)$$

$$\stackrel{\mathsf{def}}{=} (g_1^{\mathsf{pid}} g_2^{\mathsf{sn}} g_3^{v} g_4^{\mathsf{expi}} g^r, \tilde{g}_1^{\mathsf{pid}} \tilde{g}_2^{\mathsf{sn}} \tilde{g}_3^{v} \tilde{g}_4^{\mathsf{expi}} \tilde{g}^r) \quad (26)$$

Here, $\mathsf{cck} \stackrel{\mathsf{def}}{=} (\mathsf{cck}_1, \mathsf{cck}_2) = (g, (g_1, \ldots, g_4), \tilde{g}, (\tilde{g}_1, \ldots, \tilde{g}_4))$ is our *coin commitment key*. Registration commitments $(\mathsf{rcm}, \widetilde{\mathsf{rcm}})$ are also dual, but under a related *registration commitment key* $\mathsf{rck} \stackrel{\mathsf{def}}{=} (\mathsf{rck}_1, \mathsf{rck}_2) = (g, (g_1, g_5), \tilde{g}, (\tilde{g}_1, \tilde{g}_5))$. Input value commitments in a TXN are computed only in $\mathbb{G}_1$ as $\mathsf{vcm}_i = \mathsf{CM.Com}_{\mathsf{cck}_1}([0, 0, v_i, 0]; z_i)$. We defer discussion of output identity and value commitments, which will be closely tied to our threshold PS signatures discussed below.

Threshold Pointcheval-Sanders signatures. The bank will sign coin and registration commitments using the threshold PS scheme from §3.1. This slightly affects our TXN format, as illustrated in Eq. 32, as well as how the bank validates and processes TXNs.

Dual commitments. For PS verification to work, any signed commitment in our TXN from Eq. 23 must be dual. Thus, we include $\widetilde{\mathsf{rcm}} \in \mathbb{G}_2$ next to rcm and, for each input $\mathsf{ccm}_i \in \mathbb{G}_1$, we include its corresponding $\widetilde{\mathsf{ccm}}_i \in \mathbb{G}_2$. When receiving a payment (as per §4.4), users must now fetch individual *signature shares* from the bank and aggregate them via $\mathsf{RS.Aggregate}$ (see §3.1). (Alternatively, the bank can aggregate the signature for the user and the user need only unblind it as per the last step in $\mathsf{RS.Aggregate}$ from Fig. 5.)

Agreeing on signing randomness. Recall that our threshold PS variant from §3.1 requires pre-agreed randomness h_j for threshold signing the jth output coin commitment ccm_j from Eq. 20. For this, both Alice and the bank pseudo-randomly compute a unique $h_j \leftarrow \mathcal{H}_{\mathsf{ps}}(j||(\mathsf{nullif}_i)_{i\in[3]})$ based on the input coin nullifiers in the transaction, where $\mathcal{H}_{\mathsf{ps}}$ is a CRHF.

Output identity and value commitments. For the bank to threshold sign, Alice must separately commit to the fields of each output coin j under a *signature-specific commitment key* $\mathsf{tck}_j = (h_j, g)$ (see §3.1). Since the bank already knows the sn_j's and the budget coin's expiration date, this could be as simple as Alice recomputing icm_j and vcm_j under tck_j. While Alice can do so for icm_j, using a different commitment key for vcm_j would break the value preservation checks from Eq. 17.

Instead, Alice leaves vcm_j as per Eq. 14 but computes an additional $\mathsf{vcm}^*_j \leftarrow \mathsf{CM.Com}_{\mathsf{tck}_j}(v_j; d'_j), d'_j \leftarrow\$ \mathbb{Z}_p$ and proves using a ZK proof π^{pedeq}_j that vcm^*_j commits to the same value as vcm_j (see App. B.1). Importantly, Alice now encrypts d'_j for each recipient in $\mathsf{ctxt}_j \leftarrow \mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{pid}, (v_j, d'_j, t'_j))$, where t'_j is the randomness of icm_j.

Thresholdizing registration and budgets. To threshold sign registration credentials for a user with pid, the user's rcm_1 commitment from Eq. 5 must also be computed under a signature-specific commitment key (h_{pid}, g), where h_{pid} is pre-agreed upon between the user and the RA. This is easy to achieve by letting h_{pid} be the hash of pid and the current time of the day. Then, the RA can threshold sign rcm from Eq. 5 as explained in §3.1. Similarly, when issuing each user's monthly budget coin, the auditor needs to threshold sign it as well. Once again, the randomness for this signature can be agreed to be the hash of pid and the current month.

Dodis-Yampolskiy PRFs. We use a well-known pseudo-random function by Dodis and Yampolskiy [72] to compute nullifiers as $\mathsf{nullif} = \mathsf{PRF}_s(\mathsf{sn}) = h^{1/(s+\mathsf{sn})}$, where $h \leftarrow\$ \mathbb{G}_1$ is a fixed PRF public parameter. Our main challenge is to efficiently prove correctness of nullif in ZK as part of the $\mathcal{R}_{\mathsf{split}}$ relation from Eq. 11. For this, we include some auxiliary information:

$$\mathsf{vk} = \tilde{h}^{s+\mathsf{sn}}\tilde{w}^t \qquad y = e(\mathsf{nullif}, \tilde{w})^t \tag{27}$$

Here, t is secret randomness from $\mathbb{Z}_p$ and $(\tilde{h}, \tilde{w}) \leftarrow\$ \mathbb{G}_2^2$ are also fixed PRF public parameters. Assuming correctness of vk and y, the bank can check the correctness of the PRF as:

$$e(\mathsf{nullif}, \mathsf{vk}) \stackrel{?}{=} e(\mathsf{nullif}, \tilde{h}^{s+\mathsf{sn}}\tilde{w}^t) = \tag{28}$$

$$= e(\mathsf{nullif}, \tilde{h}^{s+\mathsf{sn}}) \cdot e(\mathsf{nullif}, \tilde{w}^t) \tag{29}$$

$$= e(h^{\frac{1}{s+\mathsf{sn}}}, \tilde{h}^{s+\mathsf{sn}}) \cdot y \stackrel{?}{=} e(h, \tilde{h}) \cdot y \tag{30}$$

We argue in §B.2 in the appendix why the check above proves correctness of the nullifier in zero-knowledge. Of course, since correctness of vk and y cannot be assumed, we prove it as part of an *inner* split relation, discussed next.

Splitproof. To prove the original $\mathcal{R}_{\mathsf{split}}$ holds as per Eq. 11, the TXN creator first computes vk and y as per Eq. 27 and uses a Σ-protocol to prove the following *inner* split relation $\mathcal{R}^*_{\mathsf{split}}(\mathsf{ccm}, \mathsf{rcm}, \mathsf{nullif}, \mathsf{vcm}, \mathsf{vk}, y; \mathsf{pid}, \mathsf{sn}, v, r, s, a, z, t)$ holds:

$$\left(\begin{array}{ll} \mathsf{ccm} = \mathsf{CM.Com}_{\mathsf{cck}_1}([\mathsf{pid}, \mathsf{sn}, v]; r) = g_1^{\mathsf{pid}} g_2^{\mathsf{sn}} g_3^v g^r & \wedge \\ \mathsf{rcm} = \mathsf{CM.Com}_{\mathsf{rck}_1}([\mathsf{pid}, s]; a) = g_1^{\mathsf{pid}} g_5^s g^a & \wedge \\ \cancel{\mathsf{nullif} = \mathsf{PRF}_s(\mathsf{sn})} & \wedge \\ \mathsf{vcm} = \mathsf{CM.Com}_{\mathsf{cck}_1}([0, 0, v]; z) = g_3^v g^z & \wedge \\ \mathsf{vk} = \tilde{h}^{s+\mathsf{sn}}\tilde{w}^t \wedge y = e(\mathsf{nullif}, \tilde{w})^t & \end{array}\right) \tag{31}$$

The splitproof π^{split} for the original $\mathcal{R}_{\mathsf{split}}$ will consist of (vk, y) and the proof for this inner $\mathcal{R}^*_{\mathsf{split}}$, which no longer (directly) argues about correctness of nullif (crossed out in red above). The $\mathcal{R}_{\mathsf{split}}$ verifier will first check $(\mathsf{nullif}, \mathsf{vk}, y)$ as per Eq. 28 and then verify the proof for the inner $\mathcal{R}^*_{\mathsf{split}}$ relation above.

ZKSoK. Our Σ-protocol for the inner $\mathcal{R}^*_{\mathsf{split}}$ is made non-interactive via the Fiat-Shamir transform. This allows the TXN creator to implicitly sign the TXN when computing splitproofs, by simply hashing the TXN outputs when computing the challenge in the Fiat-Shamir transform (see App. B.1).

Sigma protocols. We often have to prove knowledge of commitment openings or equality of certain committed values, which we do using well-known Σ-protocols. (We give all details in App. B.1.) Our final transaction format is:

$$\mathsf{tx}_{\mathsf{acc}} = \begin{cases} (\mathsf{rcm}, \widetilde{\mathsf{rcm}}, \mathsf{rs}), \pi^{\mathsf{budget}}, \mathsf{expi} \\ \langle (\mathsf{ccm}_i, \widetilde{\mathsf{ccm}}_i, \sigma_i), \mathsf{vcm}_i, \mathsf{nullif}_i, \pi_i^{\mathsf{split}} \rangle_{i \in [3]} \\ \langle \mathsf{icm}_j, \pi_j^{\mathsf{zkpok}}, \mathsf{vcm}_j, \mathsf{vcm}^*_j, \pi_j^{\mathsf{pedeq}}, \pi_j^{\mathsf{range}}, \mathsf{ctxt}_j \rangle_{j \in \{A,B,C\}} \end{cases} \tag{32}$$

5.1 Decentralizing the Bank via BFT SMR

A natural approach to decentralize the bank is via generic *Secure Multiparty Computation* (MPC), however this generic approach would be rather costly. We use a highly optimized approach by observing that the UTT transaction can be split into 3 parts: (1) verification of the transaction (this is pure function); (2) atomic verification that the nullifiers are new and then adding them to the nullifier list; (3) signing the outgoing coins.

Parts one and two require no secrets and hence can be implemented as a public ledger (essentially a *Byzantine Fault Tolerant* storage). Only Part three requires secrets to securely sign the out going coins (after the transaction is validated and verified atomically for no double spend). We design this part via an efficient threshold cryptography scheme. Using a BFT and threshold cryptography is a fairly common architecture. When designed naively, it requires to *serially* execute all transactions and hence does not scale well. In order to make better use of multi-core servers, we use methods from *Optimistic Concurrency Control* approaches that we tailor to the malicious setting. In particular we observe that it is possibly to optimistically run part one (the verification of the transaction) in parallel, before consensus, and only execution part two (atomic verification of no double spend of nullifiers) after consensus.

Vertical scalability via Pre-execution and Optimistic Concurrency Control. In order to better utilize modern multi-core architectures and the inherent parallelism of disjoint UTT transactions, we use a method for Byzantine Fault Tolerant Optimistic Concurrency Control which we call *pre-execution*.

Instead of sending UTT transaction directly for consensus, the primary sends them to the servers for pre-execution. Each server (optimistically) *pre-executes* each UTT transaction by checking its cryptographic validity and (optimistically) checking that its nullifiers are not already present in the nullifier list. If the check succeeds, the server creates a read set and a write set that equals to the new nullifiers, and sends a signed message back to the primary with this read write set and the hash of the transaction. Once the primary receives $f+1$ signed messages of the same read and write set and the hash of the transaction, it adds this cryptographic proof to the block that goes for consensus. After a block passes consensus, it gets sent for execution. During execution each server serially checks each read-write set in its block for conflicts (so there is no double spends) and checks that there are $f+1$ signatures for this hash of the transaction (so the transaction is valid). If there are no conflicts and the transaction is valid then the server signs its part of transaction's outgoing coins and adds the nullifiers to its nullifier list.

This Byzantine Fault Tolerant Optimistic Concurrency Control via Pre-execution provides important benefits: (i) it allows UTT transactions to be verified in parallel (allowing to take advantage of modern multi-core servers), and (ii) it removes expensive cryptography verification from the critical path of the serial execution stage after consensus .

5.2 Decentralizing the Bank via Byzantine Consistent Broadcast

We also explore an alternative design where the bank is decentralized using *Byzantine Consistent Broadcast* and threshold cryptography. The main observation is that much like other token based payment schemes [38–40], the consensus number of a UTT coin is one (only the payer can make operations on its coin) and there is no need to totally order transactions from different payers.

In this design, the payer sends the transaction directly to all the servers. Each server simply process the transactions it receives *locally*: it checks that the transaction is valid and that the coins are not spent (their nullifiers are not in the nullifier list). If these checks pass, it sends the partial threshold signature of the new coins to the clients and adds the new nullifiers to the nullifier list. Reading and writing to the

nullifier list is done atomically. The threshold signature is set to require at least $n - f$ signatures.

Once the client accumulate $n - f$ signatures is has a cryptographic proof that indicates that the incoming coins are spent by at least $n - 2t$ honest servers hence no other way of spending these coins can obtain $n - f$ signatures (due to quorum intersection). In addition, since the outgoing coins are signed, the payee can immediately use them. We also want to persist the outgoing coins at the servers in order to provide *data availability*, which is the ability of any party to read the signed outgoing coins. The user sends the signed transaction back to the servers and waits for $n - f$ confirmations that the servers stored it. When a server received a signed transaction, it locally stores it and returns and acknowledgment back. Once the user receives $n - f$ acknowledgements, it has a cryptographic proof that the signed transaction is *available*: anyone can query the servers and by waiting for just $n - f$ responses will receive at least one honest server with the signed transaction (due to quorum intersection).

Note that Byzantine Consistent Broadcast is inherently parallel. The only sequential (atomic) part is when checking the nullifier list for double spend. In terms of latency, the BFT SMR approach requires waiting for a block of commands and needs additional round trips for consensus. Hence we expect the BCB approach to have better latency. On the other had, BCB does not provide total ordering, this may add challenges to cross-server backup-and-restore procedures, and in the future for smart contracts that need total ordering.

Fast path for Byzantine Consistent Broadcast

As in many other fault tolerant protocols [73], it is possible to have a *fast path* which reduces latency and message complexity in the *optimistic case* where all servers are non-faulty and the system is synchronous. For Byzantine Consistent Broadcast we observe that obtaining n-out-of-n signatures from the first round (instead of just $n-f$-out-of-n) guarantees data availability without needing an additional round: any later read will eventually see $n - f$ responses for the same signed transaction. Similar to SBFT, we use a fast path single-round BCB, wait for n responses with a timeout to revert to the regular two round protocol that waits for $n - f$ responses each round.

5.3 Horizontal scaling via bank sharding

In the previous two subsections we described two ways to decentralize a single logical bank. In many cases one would want to scale beyond just one bank, by partitioning the bank into several *bank shards*. The high level idea is that *logically* each bank shard will responsible for a different part of the nullifier space. This will allow to increase throughput by adding more bank shards. *Operationally*, to obtain decentralized trust, each bank shard will be decentralized via a separate sub-system of multiple servers (each such sub-system has a disjoint set of servers implementing it).

Sharding the bank The key observation is that we can split the UTT payment into two separate phases:

(i) *burn phase*, where the nullifiers are marked as spent and the transaction is verified, and

(ii) *mint phase*, where new coins are signed.

Given a set of logical bank shards, we use a standard *consistent hashing* approach that deterministically map the space of a hash function on the nullifiers to the bank shards. Each bank shard is decentralized via a separate BCB system of $3f + 1$ servers, hence a system with k bank shards will need a total of $k(3f+1)$ servers. All bank shards use the same secret signing key, but each sub-system secret shares this secret key independently in such a way that secrets shares from different sub-systems are completely independent. Thus, the security of the system is maintained as long as for any sub-system the adversary does not control more than f severs out of the $3f + 1$ servers that are used to decentralize each sub-system.

Logically, a user first executes the *burn phase* by sending the transaction to all the bank shards that are responsible for the nullifiers in the transaction. Each such bank shard verifies the transaction and that the nullifiers it is responsible for have not been double spent and sends a signed hash of the transaction back to the user. Since UTT transaction have three incoming coins that need to be burned, this means the user needs to communicate with at most three bank shards. The user can then execute the *mint phase* by sending the signed hash from all the bank shards responsible to the nullifiers in the transaction to any bank shard. The receiving bank shard checks that the signatures are valid and from the correct bank shards and signs the new outgoing coins. To improve load balancing, we force the mint phase

to happen only on a designated shard (as a function of the hash of the transaction).

5.4 Decentralizing the bank shards via Crusader Broadcast and Byzantine Consistent Broadcast

In order to decentralize the burn phase and mint phase of a UTT transaction we make the observation that while the mint phase may need the full two-round Byzantine Consistent Broadcast for non-equivocation and data availability, the burn phase only needs a proof of non-equivocation - there is no need to have a proof that the certificate of burn is available on each of the bank shards responsible for the burn, because this certificate will be made available on the shard that executes the mint phase during the mint phase.

Therefore we can optimize the decentralized implementation as follows. We implement the mint phase using Byzantine Consistent Broadcast with a fast path: for the first round of BCB each server in the minting bank shard locally checks that $n - f$ servers from each of the sub-systems responsible for burning the incoming coins have indeed burned them, and then signs the outgoing coins with its threshold signature. If n responses are returned then the protocol is complete (fast path BCB). Otherwise the $n - f$ signatures are persisted in the second round of BCB.

For the burn phase, users run a single round protocol (which we call *Crusader Broadcast*) waiting for $n - f$ signatures. A server locally checks that the incoming coin's nullifier is indeed in its responsibility and is not in its nullifier list. It then sends back a signature of the transaction. So $n - f$ such signatures guarantees that the coins that are in the responsibility of this sub-system cannot participate in any other transaction (because that requires $n - f$ signatures which is impossible due to quorum intersection).

For the burn phase, the user sends a one round trip (Crusader Broadcast) to at most three sub-systems (in parallel) and once each of these sub-systems returns $n - f$ signatures. Then, for the mint phase the user runs Byzantine Consistent Broadcast with a fast path, so requires two round trips (just one in the optimistic case). In total, each transaction requires three round trips (just two in the optimistic case) of latency and interacting with at most 4 different bank shards.

6 Implementation and Experimental Evaluation

We evaluate our open-source implementation[2] of our accountable transaction from §4.6. We first conduct microbenchmarks of our cryptographic implementation on a single machine. The remaining of our experiments are performed on a real cloud-based distributed system (AWS). Unless otherwise noted, we use separate machines for users and a separate machine for each server. machines (c5.4xlarge running Ubuntu 18.04) use 16 cores and deployed to the same AWS region (us-east2). Our default setup is for $n = 4,\ f = 1$ (but see later for higher sizes of n, f, and more cores). All transactions use the accountable privacy payments described in the previous section.

Under these experimental conditions, we formulate the following hypothesis and experimental questions.

1. What is the performance profile of a UTT transaction in terms of its size and cryptographic latency overhead. We conduct micro benchmarks.

2. What is the latency throughout trade-off obtained when decentralizing the system using BFT SMR? What component is bottlenecking the throughout?

3. What is the latency throughout trade-off obtained then decentralizing the system using BCB? The theory leads us to conjecture that the latency should be better than SMR BFT.

4. How does the performance scale when we increase the fault tolerance to $f \in \{1, 3, 5, 10\}$? The theory leads us to conjecture that both throughput and latency should not be effected drastically since the main bottleneck is the local execution (cryptographic verification and signature generation).

5. How does the performance improve due to vertical scaling (adding more cores to the servers)? Our conjecture is that CPU is the bottleneck for throughput, hence increasing the number of cores should near-linearly increase throughput.

6. How does the performance improve when the Bank is sharded and each bank shard is decentralize via a separate distributed set of servers? The theory leads us to conjecture that we pay a

[2] https://github.com/definitelyNotFBI/utt

constant factor overhead for moving to a shaded protocol, and increase latency by another round trip, but after that the throughput should scale near-linearly as a function of the number of shards.

6.1 Micro-benchmarks

We implemented our accountable transaction in a C++ library called `libutt` based on BN254 elliptic curve groups [74] implemented in `libff` [75]. Here, we microbenchmark `libutt` on our default AWS machines and summarize our results in Table 1.

We measured the size of transactions, the time to create a TXN for Alice, the time to validate a TXN by the BCB servers (i.e., "BCB TXN validation" column), the time for a bank server to fully validate a TXN, the time for Bob to check if a transaction is paying him (i.e., "Check TXN is mine" column), and the time for Bob to actually decrypt his coin details from the TXN and aggregate his coin signature (i.e., "Claim TXN" column).

Analysis. Not surprisingly, we observe that TXN validation is the biggest bottleneck that will affect the throughput and latency of the system. Our prototype implementation as well as our cryptographic design could be further accelerated (see §8).

Table 1: Accountable transaction microbenchmarks (on the default AWS machines).

Name	Measurement
TXN size	14.4 KiB
TXN creation	60.45 ms
BCB Quick TXN validation	14.56 ms
Bank TXN validation	34.11 ms
Bank Sign	2.70 ms
Check TXN is mine	0.65 ms
Claim TXN	4.53 ms

6.2 BFT-SMR Performance

We experiment with a BFT SMR bank of $n = 4$ servers as per §5. We evaluated *throughput* in terms of the number of transactions processed by the system (measured on the primary server) and *latency* in terms of the end-to-end transaction confirmation time as measured by the sender and recipient (clients of the BFT-SMR).

Experimental setting. We run our macrobenchmark on the default AWS machines (16 core, 16 GiB RAM). We start with the default $f = 1$ and $n = 3f + 1 = 4$ servers for the BFT SMR bank, with one machine per server connected, all in the same AWS region.

BFT implementation. For a BFT SMR based Bank we use concord-bft [76], an open-source production implementation, based on SBFT [73], which implemented the pre-execution protocol in January 2020 [77]. The servers use RocksDB to persistently store nullifer sets [78]. The pre-execution is used to validate (or reject) transactions that are not cryptographically valid and transactions that contains coins that are already spent. After a batch of transactions is committed, the servers execute each transaction in the batch serially. In each transaction execution, the servers verify (or reject) that the nullifiers are not in the nullifier list. If this is the case, then and add them to the nullifier list and issue partial signatures to every client.

Methodology. Each client sends a UTT accountable transaction to the servers and waits for $n - f$ servers to respond. At least $n - 2f$ responses are guaranteed to be valid and can be used to recover the newly minted coins. Every $7s$ we measure the average throughput in a 1s interval at the primary. We then report the *throughput* as the median of these averages. For end-to-end latency, we maintain a histogram at each client and report the median end-to-end latency over 400 operations. We then report the *latency* as the median over the client reported median latency. The BFT SMR uses an automated batching policy, and we observe an average batch sizes of 20 transactions per consensus unit.

Table 2: Throughtput and latency of payments on a BFT SMR bank with $n = 4$ servers on AWS c5.4xlarge.

# Clients	Throughput	Latency
4	56.00 tx/s	110.49 ms
8	96.98 tx/s	125.44 ms
16	129.16 tx/s	162.88 ms
32	165.28 tx/s	231.04 ms
64	173.93 tx/s	460.00 ms

Analysis. Table 2 shows that a BFT bank can process about 10 transactions per second per core (extracted from the second to the last row with

32 clients). However, when measuring the average times to handle a transaction inside a BFT server for pre-execution and execution after consensus, we get 62.23 ± 4.44 ms and 7.2 ± 4.44 ms leading to theoretical maximum throughput of ≈ 230 tx/s. Instead, we observe an actual throughput of 174 tx/s which arises due to the serial execution of the transactions after consensus. We believe further tuning and optimizations like parallelizing the post-execution will improve the BFT throughput. Indeed, removing only the signature generation in the execution after consensus for the last row in Table 2 gives a median throughput of 272.14 tx/s and latency of 289.23 ms.

6.3 BCB Performance

For a BCB based Bank we implemented the BCB with fast path (See Section 5.3), based on concord-bft codebase in C++ and use the same RocksDB database [78] to store the nullifier sets. In this set of experiments, we use the default $n = 4$ servers, and the same experimental settings as §6.2.

Methodology. Each client sends a UTT accountable transaction to the servers using the BCB with fast path protocol. For *throughput*, we measure the average throughput every second on every server. We then take the mediation on each server and then median over all servers. For end-to-end *latency*, as in §6.2, we maintain a histogram at each client during 400 operations and report the median between all clients of the medians at each client as our measurement.

Table 3: Throughput and latency of payments on a BCB-based bank with $n = 4$ servers on AWS c5.4xlarge.

# Clients	Throughput	Latency
4	64, 40 tx/s	48.04 ms
8	111.60 tx/s	58.00 ms
16	233.09 tx/s	65.23 ms
32	235.39 tx/s	129.90 ms
64	235.36 tx/s	274.81 ms

Analysis. Table 3 shows that a BCB based bank can process 15 transactions per core. The lowest median latency observed among all the clients was 46.53 ms. The throughput of the BCB based bank is 1.35 times more than the BFT SMR, and the median latency observed by the clients are also 67.4 % lower than the latency observed by the clients using BFT SMR based banks. When disabling the post-execution bottleneck in the BFT-SMR, we observed that BCB still provides 5% better latency. These experiments clearly show that BCB based solutions can significantly improve the payment latency experienced by the users.

6.4 Server Scalability

In this section, we evaluate the scalability of our payment systems based on BFT SMR §6.2 and BCB §6.3 as n increases.

Methodology. We use $f = 1, 3, 5, 10$ and the corresponding values for $n = 4, 10, 16, 31$. As we observed in the stand-alone measurements for BFT-SMR and BCB, the latency generally increase with increasing number of clients before saturating the throughput. Therefore, we report the saturated throughput as *sat. throughput* with 64 clients and report *low latency* as the median latency with 4 clients.

Table 4: Scalability of payments on a BFT SMR bank with $n = 4$ servers on AWS c5.4xlarge.

# servers	Sat. Throughput	Low Latency
4	172.86 tx/s	107.84 ms
10	170.71 tx/s	107.71 ms
16	168.93 tx/s	110.00 ms
31	168.57 tx/s	110.05 ms

Table 5: Scalability of payments on a BCB based bank with $n = 4$ servers on AWS c5.4xlarge.

# servers	Sat. Throughput	Low Latency
4	235.98 tx/s	40.71 ms
10	235.42 tx/s	39.46 ms
16	234.40 tx/s	49.83 ms
31	229.28 tx/s	53.71 ms

Analysis. Table 4 and Table 5 show the scalability of the BFT SMR based system and the BCB based system as n increases. In general, the throughput and latency remain nearly constant. This is because the system is mainly bottlenecked by cryptographic operations.

6.5 Vertical scalability

We evaluate the vertical scalability of the BCB systems as we increase the number of cores of the individual servers.

Methodology. For this we use the following AWS machines: c5.4xlarge (16 cores), c5.9xlarge (36 cores), c5.12xlarge (48 cores), c5.16xlarge (64 cores), c5.18xlarge (72 cores), and c5.24xlarge (96 cores). For the rest of the setup we use the same methodology as §6.4.

Table 6: Vertical scalability of payments on a BCB based bank with $n = 4$ servers on AWS c5 machines

# cores	Sat. Tput.	Low Lat.	Tput./core
16	235.36 tx/s	43.01 ms	14.75
36	498.32 tx/s	44.06 ms	11.58
48	636.60 tx/s	38.77 ms	13.26
72	788.00 tx/s	43.49 ms	10.94
96	974.54 tx/s	38.6 ms	10.15

Analysis. Table 6 shows the scalability of the BCB system, as we increase the number of cores. We observe that the median saturated throughput scales nearly linearly as the number of cores increase. Comparing the throughput of the 16 core machine and the 96 core machine, we expect a 6 time performance improvement. However, we find that the throughput increases only by ≈ 4.1 times. This arises due to synchronization overheads across the threads. Further optimizations such as lock-free and wait-free algorithms can improve the performance of the system and we leave it as future work.

6.6 Horizontal Scaling using Sharding

We evaluate the UTT sharding as described in §5.3.

Implementation. We implement each shard as $n = 4$ BCB servers, and use a total of $s \in \{1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 60\}$ shards. Clients create a *burn request* that contains the transaction and a target shard where the coin will be minted. For every nullifier in the transaction, the client derives the responsible shard (modulo number of shards) and sends the *burn request* to all n servers in the shard. Upon collecting the required signatures from all the responsible shards (upto $3n$ signatures), the client sends a *mint request* to the target shard specified in the *burn request*. The servers validate the embedded transaction TXN and verify all the signatures, and then send partial coin signatures to the client. The client collects at most n partial coin signatures before terminating.

The BCB servers do not need to perform a full UTT payment validation when validating the *burn request* as it will be performed by the shard that executes the *mint request*. We use RSA signatures as responses to the *burn request* which is also validated when executing the *mint request*.

6.6.1 Microbenchmarks

In this section, we evaluate the performance of sharding with a non-fault tolerant $f = 0$ setting using one server ($n = 1$) per shard and compare that to the $f = 1$ fault tolerant case with $n = 4$ servers per shard. In both configurations, we vary the number of shards $s \in \{1, 2, .., 5\}$ and report throughput and latency in Table 7.

Methodology. Clients generate transactions as in our previous SMR macrobenchmark from §6.2, and measure the end-to-end latency until the client collects the last response for the *mint request*. We measure latency as described in §6.5. We use 4 clients to measure latency under low load. We use $160 - 320$ clients to measure throughput. For load balancing, each client is deterministically assigned a target shard to get its coins minted.

The BCB servers track the number of *mint requests* processed as throughput. We use the median throughput across all the servers in a shard as the throughput per shard and compute the sum of these throughput measurements as the total throughput of the system.

Table 7: Microbenchmarks of sharding performance with $n = 1, 4$, and number of shards $s = 1, 2, .., 5$ on c5.4xlarge machines.

s	n=1		n=4	
	Tput.	Lat.	Tput.	Lat.
1	221.59 tx/s	43.68 ms	206.92 tx/s	53.68 ms
2	417.09 tx/s	42.87 ms	406.92 tx/s	47.71 ms
3	605.82 tx/s	43.79 ms	609.90 tx/s	46.90 ms
4	811.08 tx/s	44.31 ms	789.60 tx/s	43.55 ms
5	999.13 tx/s	42.58 ms	984.56 tx/s	42.58 ms

Analysis. Table 7 shows the throughput and latency between $n = 1$ and $n = 4$ servers per shard, as the

number of shards increase slowly.

The latency observed by the clients is close to the latency of the Bank validation (see Table 1) along with an additional $\approx 2-3$ ms overhead (from other computations). Importantly, the latency remains constant as the number of shards increases. This confirms that the bottleneck is computational and that the communication cost is constant.

The throughput scales almost linearly with the number of shards, since the load is uniform and there is no cross-shard communication. Compared to the performance measurements observed in §6.3, the throughput is slightly lower. This is because the servers need to process *burn requests* that are not counted toward the throughput measurements, decreasing the amount of CPU available to process the *mint requests* which are measured for throughput.

We also observe that the latency measured at the clients during the throughput experiments drops as the number of shards increases. For example, for $n = 4$, when $s = 2$ with 160 clients, the median latency was 750.13 ms, which drops to 322.05 ms when $s = 5$ with 320 clients, further proving the benefits of sharding in reducing payment latency. We believe additional tuning may reduce latency under load and leave this as future work.

6.6.2 Large scale horizontal scaling

Our main conclusion from observing Table 7 is that the $n = 1$ based shard can act as a reasonable performance proxy for the fault tolerant $n = 4$ shard in terms of both throughput and latency Said differently, the slowdown of $n = 4$ compared to $n = 1$ is less than %5 for throughput.

Methodology. Our AWS account allows only 1920 vCPUs. In order to evaluate higher numbers of shards, we use 1 server per shard as a proxy, for the large scale sharding experiments of this section. We measure the median throughput per shard and report the median among all medians as *Throughput per shard* (Tput./shard). We sum the medians and report them as *Total throughput* (Tot. Tput.).

We start with 640 clients and increase the number of clients as the number of shards until we observe a saturation in the throughput in every shard.

Analysis. Table 8 shows the throughput as a function of the number of shards. As expected from the share nothing design, we observe that the throughput of a single shard is nearly constant and the throughput of the entire system scales almost linearly with the number of shards.

Table 8: Macrobenchmarks of sharding performance with $n = 1$, and number of shards $s = 10, 20, 40, 60$ on c5.4xlarge machines.

# Shards	Tput./shard	Tot. Tput.
10	200.89 tx/s	1,996.90 tx/s
20	196.77 tx/s	3,726.34 tx/s
30	196.05 tx/s	5,770.50 tx/s
40	198.65 tx/s	7,632.56 tx/s
50	195.56 tx/s	9,601.01 tx/s
60	196.36 tx/s	11,351.38 tx/s

Our experiments confirm that UTT can scale horizontally to tens of thousands of accountable privacy transactions per second.

7 Formalizing and Proving Security

It is nontrivial to formally capture the security properties that an ecash system should provide. Notions like Balance (e.g., preventing/detecting double-spending), and Privacy/Anonymity that were originally stated somewhat informally [1–5] have evolved into more formal game-based definitions (e.g., [7,16]). These definitions typically capture the different requirements (balance and privacy) via *separate* games, and, as discussed by Garman et al. [21], they tend to miss some critical desired properties.

A natural remedy is to define security by comparing the system to an *ideal functionality* and arguing *indistinguishability* via a suitable MPC framework. This approach allows one to define security without having to identify and specify concrete desired properties. (As a byproduct, this also leads to technically stronger notions of simulation-based security.) Continuing a recent line of research [21,25,79], we formalize the security of our system via an MPC-based definition and, within this framework, explore new tradeoffs between security and performance. Let us elaborate on some aspects of our definitions and proofs.

Defining the ideal functionality. There are many reasonable ways to model ecash as an ideal functionality. Since the exact choice may depend on the concrete application, we provide a minimal spec-

ification that mainly strives for simplicity. In a nutshell, our ideal functionality, denoted by $\mathcal{F}_{\mathsf{utt}}$, maintains for each "coin" a record that contains fields like value, owner-id, and coin identifier. The functionality supports 2 operations: A "Payment" that can be issued by any client, and a "Minting" that can be issued by a special *minting authority*. This allows us to leave the conditions under which Minting happens to an external mechanism. Whenever an operation takes place, the functionality updates its state, sends an announcement to the relevant parties and leaks to the Banks only the fact that an operation happened (or failed). Notably, our ideal functionality does not involve any cryptographic objects (e.g., commitments/signatures). Indeed, since cash is a non-cryptographic notion, we believe that its *modeling* should be cryptography-free.[3] The basic structure of $\mathcal{F}_{\mathsf{utt}}$ is easily extendable in various ways. Specifically, one can easily support anonymity budgets by slightly tweaking the functionality. This extension, as well as other variants of $\mathcal{F}_{\mathsf{utt}}$, are discussed in Appendix C. Jumping ahead, our protocol and its proof are also modular and can easily extend to support these variants.

Defining the MPC model. Since our ideal functionality is *reactive* (i.e., it maintains a state), 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.[4]

Garman et al. [21] suggest using a simplified MPC model in which the "strategies" of the honest parties are fixed in advance. As a result, the security definition does not take into account the ability of the adversary to inject online inputs to the honest parties based on its evolving view of the system. (Indeed, one can design a contrived system whose security completely breaks under such a attack; see Appendix C.2.) On the other extreme, security can be defined in the UC model [80] as done by [25,79], however, this leads to various complications (e.g., [81]) and seems to incur some inherent cost in performance due to the use of UC-secure building blocks. We suggest a new intermediate solution by presenting a limited version of the UC definition. Our definition can be viewed as a "standalone MPC security" for reactive functionalities, and so it fills an important gap in the MPC literature.[5]

In a nutshell, we assume a synchronous setting (like [83–85]), and assume that the protocol is invoked in "phases". In the beginning of each phase, the adversary and the honest parties receive inputs from the environment Env, then they participate in the protocol, and, at the end of the phase, they 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*, this framework allows, for example, to rewind the adversary to the beginning of the current phase and can be easily extended with ideal oracles if needed. See Appendix C.2 for more details.

Realizing the functionality via practical protocol. Our proof reduces the security of the protocol to the standalone security of the underlying cryptographic building blocks. This includes standard primitives like PRFs, anonymous identity-based encryption scheme, and non-interactive zero-knowledge proofs of knowledge, as well as a new notion of rerandomizable signatures over (homomorphic) commitments. Roughly speaking, this primitive allows us to take a signed commitment and randomize both the signature and the commitment in a way that maintains the validity of the signature. This notion may be of independent interest.

Our proof is modular in the sense that one can replace the instantiation of the underlying primitives without affecting security. The concrete instantia-

[3] This should be contrasted, for example, with [25] that models "token payments" via an ideal functionality that refers to commitments.

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

[5] For standard (non-reactive functionalities), the MPC literature consists of 2 main frameworks, the classical stand-alone model (as formalized in [82], and Canetti's UC model. While the latter provides stronger composability advantages, the former model is simpler to work with and, in many cases, provides sufficiently good security. Furthermore, even if one strives for full UC security, proving security in the standalone model provides a necessary intermediate goal, and in some cases, one can upgrade security to UC almost automatically (e.g., [83]). To the best of our knowledge, a standalone MPC definition that is tailored to the setting of reactive functionalities has not appeared in the MPC literature so far.

tions that we use sometimes employ idealized models (e.g., Random Oracle and AGM) and so our protocol inherits these assumptions from the concrete implementations. It seems likely that our proof yields a UC-secure protocol if all the underlying primitives are instantiated with UC-secure building blocks. Verification of this intuition is left for future research.

Remark 7.1. In Part I, we formalize the MPC framework, define the underlying primitives, represent the protocol in more abstract terms, and prove its security. Apart from space limitations, this separation into "cryptographic analysis" and "main text" reflects the different roles of the two parts (validating formal security vs. presenting the architecture in an easy-to-follow intuitive way) and allows us to tailor the presentation and terminology to the purpose of the text.

8 Discussion

Security against adversaries controlling more than f servers. A known disadvantage of the blind signature-based approach for anonymity is that an attacker who steals the bank's bsk can mint coins undetectably! While BFT mitigates against this, we could further strengthen security using *evolving-committee proactive secret sharing (ECPSS)* techniques [86] at a larger-scale. Another idea is to make coin issuance *transparent* using an accumulator [87] over all minted coins' serial numbers (not values or pid's). Then, each transaction input would additionally prove that its coin's sn is on this list. This would force attackers to add their maliciously-minted coins to the list too, which is now evident to an auditor. This requires a novel accumulator design, with logarithmic-sized zero-knowledge proofs which are updatable in time logarithmic in the number of changes to the accumulator. Importantly, this would *not* require a circuit-based ZK proof as in Zcash, which must prove knowledge of an opening of a coin commitment in their (Merkle) accumulator (among other things). We see this as an interesting design point that combines the advantages of ecash schemes [1] with the advantages of accumulator-based schemes [4, 12, 16, 51].

Key loss and theft. In case a user loses their IBE decryption secret key dsk, they can recover it by reengaging with the registration authority (RA) as per §4.2. The same can be done for a lost PRF key s, if it is secret-shared with the RA during registration. In contrast, key *theft* of dsk or of s requires keeping a revocation list of compromised pid's. To allow reusing pid's, we actually extend them with a version number and revoke by version. When paying Bob, Alice will update her view of this revocation list and use Bob's latest pid. Alice can retrieve this item from the revocation list using private information retrieval (PIR). If a revocation list is undesirable due to other privacy concerns, normal PK encryption can be used as a fallback. We hope to improve the privacy of this approach in future work.

Permissionless setting. Although our approach relies on a permissioned proof-of-membership BFT committee that maintains secrets in a threshold fashion, such a committee can also be implemented in a permissionless proof-of-stake setting by periodically refreshing the committee and handing over secrets, similar to DFINITY [88], Algorand [89] and the recent line of work by Benhamouda et al. [86].

On accountability. First, we can imagine other accountability rules that might create a better balance between anonymity and accountability. For example, anonymously purchasing prescription drugs from pharmacies could be allowed even without an anonymity budget. We could implement such a rule *efficiently* via a Σ-protocol that argues in ZK that the pid committed in the recipient's icm is in an *allowed list*. Another interesting example would be paying taxes in a *batched* manner without leaking each transaction's tax amount. Second, our anonymity budgets only limit outgoing payments. Limiting incoming payments could also be done using techniques from [31], although at the cost of adding interaction between Alice and Bob during a payment.

On transacting. First, our accountable transactions are *serial*. Specifically, since Alice has a single budget coin, she must wait for her current accountable transaction to be added to the ledger, before she can get back her budget coin change for her next transaction. However, this is not inherent in our design, which could be adapted to support multiple budget coins as input and give multiple budget coins as change, although at the cost of higher complexity.

Second, Bob cannot be easily informed by the bank when he is paid, since the bank is oblivious to this. This is similar to other DAPs [16, 64]. However, Alice could easily inform Bob she paid him by pointing him to the TXN on the ledger. Alternatively, Bob could manually inspect every TXN to check if it corresponds to a payment for him. Although inspecting

one TXN is fast (1.3 ms), having all users inspect all TXNs would be expensive. Ideally, Bob would delegate this task to an untrusted third party, but we leave this as future work.

Third, it would be interesting to design ZK proofs for Alice to convince a third party that she paid Bob, as well as for Bob to convince others that Alice did not pay him.

Black market for anonymity budgets. One might be worried about the possibility of a black market for anonymous transactions. For example, if Alice runs out of her privacy budget and still wishes to anonymously pay Bob, she might attempt to pay Carol in an accountable way and ask her to transfer this money to Bob anonymously (perhaps after Carol takes a cut for her service).

This issue is relevant to all systems which combine privacy with accountability. Our system has two properties which greatly limit the effect of such attacks: (1) Alice can only exceed her privacy budget by paying intermediaries in an *accountable* way, which is visible to auditors. And, (2) this type of attack cannot be amplified by Sybil attacks, since the registration process (Section 4.2) ensures that each user corresponds to a verified identity.

Payment disputes. There is currently no support for resolving "payment not received" disputes, but this can be done in principle using a Σ-protocol. For example, looking at our transaction format from Figure 2, the sender could easily prove in ZK that such a transaction on the ledger has (1) an input coin commitment whose PID is Alice, (2) an output identity commitment icm_B which opens to Bob's identity and (3) an output value commitment vcm_B which opens to v coins.

Faster validation. We see several avenues for speeding up our transaction validation. First, we can combine our many Σ-protocols into a single one which will save both time and space. Second, we can apply batch verification techniques to our range proofs, our signatures and our exponentiations, both within one TXN and across many. Third, we can investigate using Bulletproofs over value commitments that are (provably) computed in a faster prime-order group without pairings. Lastly, to speed up our pairing-and-exponentiation-based verification, we could use *generalized inner product arguments (GIPA)* [90] (and potentially reduce transaction size).

***k*-to-*m* TXNs** Our unaccountable transactions from §4.4 allow Alice to "split" two coins of value v_1 and v_2 amongst Bob and herself. But what if Alice has only one coin of denomination v to spend? In that case, she cannot create such a 2-to-2 transaction. One way to fix this is to allow 1-to-2 transactions that can split a single coin of value v into two coins. In general, we can allow arbitrary k-to-m transactions by simply adding more inputs and outputs and extending our cryptographic checks over all of them.

9 Conclusion

This paper makes progress on a novel design, security proof, and implementation of a decentralized ecash system with accountable privacy. Our design emulates physical cash and offers strong privacy as long as the total payments are below a privacy budget. We provide a security definition and proof for UTT in the framework of multi-party computation (MPC). Our experiments over a real-world distributed implementation confirm our hypothesis on scalability and applicability as a digital infrastructure for several tens of millions of active users (tens of thousands of accountable privacy transactions per second). We believe that significant additional performance can be obtained with further tuning. As detailed in the discussion section, our work is by no means a complete solution, and we expect follow-up work to explore additional challenges. In particular, scaling beyond hundreds of thousands of privacy-preserving transactions per second will most likely require a hierarchical approach.

10 Acknowledgements

We thank Michael Reiter, Dahlia Malkhi, and Anna Lysyanskaya for early discussions in 2018-2019. We thank Andrew Stone, Petar Ivanov, Yulia Sherman, Robert Muschner, and Evgeniy Dzhurov from the VMware Blockchain team for helping with concord-bft integration. We thank Naama Ben David, Sravya Yandamuri, Gadi Aharoni and Alim Karim for insightful discussions.

References

[1] D. Chaum, "Blind Signatures for Untraceable Payments," in *Advances in Cryptology*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Boston, MA: Springer US, 1983, pp. 199–203.

[2] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, 1988, pp. 319–327.

[3] S. A. Brands, "An Efficient Off-line Electronic Cash System Based On The Representation Problem." Amsterdam, The Netherlands, The Netherlands, Tech. Rep., 1993.

[4] T. Sander and A. Ta-Shma, "Auditable, Anonymous Electronic Cash," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 555–572.

[5] ——, "Flow Control: A New Approach for Anonymity Control in Electronic Cash Systems," in *Financial Cryptography*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 46–61.

[6] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials with optional anonymity revocation," in *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, 2001, pp. 93–118.

[7] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact E-Cash," Cryptology ePrint Archive, Report 2005/060, 2005, https://eprint.iacr.org/2005/060.

[8] ——, "Balancing Accountability and Privacy Using E-Cash (Extended Abstract)," in *Security and Cryptography for Networks*, R. De Prisco and M. Yung, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 141–155.

[9] S. Canard and A. Gouget, "Divisible E-Cash Systems Can Be Truly Anonymous," in *Advances in Cryptology - EUROCRYPT 2007*, M. Naor, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 482–497.

[10] M. H. Au, W. Susilo, and Y. Mu, "Practical Anonymous Divisible E-Cash from Bounded Accumulators," in *Financial Cryptography and Data Security*, G. Tsudik, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 287–301.

[11] S. Canard and A. Gouget, "Multiple Denominations in E-cash with Compact Transaction Data," in *Financial Cryptography and Data Security*, R. Sion, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 82–97.

[12] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous Distributed E-Cash from Bitcoin," in *IEEE Security and Privacy '13*, May 2013, pp. 397–411.

[13] F. Baldimtsi, M. Chase, G. Fuchsbauer, and M. Kohlweiss, "Anonymous Transferable E-Cash," in *PKC*. Springer, 2015, pp. 101–124. [Online]. Available: https://www.iacr.org/archive/pkc2015/90200211/90200211.pdf

[14] B. Bauer, G. Fuchsbauer, and C. Qian, "Transferable E-cash: A Cleaner Model and the First Practical Instantiation," Cryptology ePrint Archive, Report 2020/1400, 2020, https://eprint.iacr.org/2020/1400.

[15] Zcash, "Zcash," 2016, https://z.cash/.

[16] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized Anonymous Payments from Bitcoin," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 459–474.

[17] N. Oberheiden, "Bitcoin and money laundering," 2021. [Online]. Available: https://www.jdsupra.com/legalnews/bitcoin-and-money-laundering-2928902/

[18] G. Myre, "How bitcoin has fueled ransomware attacks," 2021. [Online]. Available: https://www.npr.org/2021/06/10/1004874311/how-bitcoin-has-fueled-ransomware-attacks

[19] F. Erazo, "Analyst who helped topple crypto child exploitation site honored," 2020. [Online]. Available: https://cointelegraph.com/news/analyst-who-helped-topple-crypto-child-exploitation-site-honored

[20] O. Kharif, "Privacy coins face existential threat amid regulatory pinch," 2019. [Online]. Available: https://www.bloomberg.com/news/articles/2019-09-19/privacy-coins-face-existential-threat-amid-regulatory-crackdown

[21] C. Garman, M. Green, and I. Miers, "Accountable Privacy for Decentralized Anonymous Payments," Cryptology ePrint Archive, Report 2016/061, 2016, https://eprint.iacr.org/2016/061.

[22] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi, "Solidus: Confidential Distributed Ledger Transactions via PVORM," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 701–717. [Online]. Available: https://doi.org/10.1145/3133956.3134010

[23] K. Wüst, K. Kostiainen, V. Capkun, and S. Capkun, "PRCash: Fast, Private and Regulated Transactions for Digital Currencies," in *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, ser. Lecture Notes in Computer Science, I. Goldberg and T. Moore, Eds., vol. 11598. Springer, 2019, pp. 158–178. [Online]. Available: https://doi.org/10.1007/978-3-030-32101-7_11

[24] N. Narula, W. Vasquez, and M. Virza, "zkLedger: Privacy-Preserving Auditing for Distributed Ledgers," Cryptology ePrint Archive, Report 2018/241, 2018, https://eprint.iacr.org/2018/241.

[25] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann, "Privacy-preserving auditable token payments in a permissioned blockchain system," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. ACM, oct 2020. [Online]. Available: https://doi.org/10.1145%2F3419614.3423259

[26] A. Barki and A. Gouget, "Achieving privacy and accountability in traceable digital currency," Cryptology ePrint Archive, Report 2020/1565, 2020, https://eprint.iacr.org/2020/1565.

[27] Y. Chen, X. Ma, C. Tang, and M. H. Au, "PGC: Decentralized Confidential Payment System with Auditability," in *Computer Security – ESORICS 2020*, L. Chen, N. Li, K. Liang, and S. Schneider, Eds. Cham: Springer International Publishing, 2020, pp. 591–610.

[28] I. Damgård, C. Ganesh, H. Khoshakhlagh, C. Orlandi, and L. Siniscalchi, "Balancing Privacy and Accountability in Blockchain Transactions," Cryptology ePrint Archive, Report 2020/1511, 2020, https://eprint.iacr.org/2020/1511.

[29] D. Bogatov, A. D. Caro, K. Elkhiyaoui, and B. Tackmann, "Anonymous Transactions with Revocation and Auditing in Hyperledger Fabric," S. Krenn, M. Conti, and M. Stevens, Eds. Springer International Publishing, 2021.

[30] P. Chatzigiannis and F. Baldimtsi, "MiniLedger: Compact-sized Anonymous and Auditable Distributed Payments," Cryptology ePrint Archive, Report 2021/869, 2021, https://eprint.iacr.org/2021/869.

[31] K. Wüst, K. Kostiainen, and S. Capkun, "Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy Preserving Regulation," Cryptology ePrint Archive, Report 2021/1443, 2021, https://ia.cr/2021/1443.

[32] M. McLeay, A. Radia, and R. Thomas, "Money in the modern economy: an introduction," *Bank of England Quarterly Bulletin*, vol. 54, no. 1, pp. 4–13, 2014. [Online]. Available: https://EconPapers.repec.org/RePEc:boe:qbullt:0127

[33] P. of Australia, "Currency (restrictions on the use of cash) bill," 2019, https://www.aph.gov.au/Parliamentary_Business/Bills_Legislation/bd/bd1920a/20bd089#:~:text=The%20cash%20payment%20limit%20is,of%20an%20Act%20of%20Parliament.

[34] D. Pointcheval and O. Sanders, "Short Randomizable Signatures," in *CT-RSA 2016*, K. Sako, Ed. Cham: Springer International Publishing, 2016, pp. 111–126.

[35] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," Cryptology ePrint Archive, Report 2000/067, 2000, https://eprint.iacr.org/2000/067.

[36] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, apr 1988. [Online]. Available: https://doi.org/10.1145%2F42282.42283

[37] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to Reliable and Secure Distributed Programming*, 2nd ed. Springer Publishing Company, Incorporated, 2011.

[38] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovič, and D.-A. Seredinschi, "The consensus number of a cryptocurrency," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, ser. PODC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 307–316. [Online]. Available: https://doi.org/10.1145/3293611.3331589

[39] M. Baudet, G. Danezis, and A. Sonnino, "Fastpay: High-performance byzantine fault tolerant settlement," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, ser. AFT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 163–177. [Online]. Available: https://doi.org/10.1145/3419614.3423249

[40] M. Mathys, R. Schmid, J. Sliwinski, and R. Wattenhofer, "A limitlessly scalable transaction system," 2021.

[41] M. Chase, S. Meiklejohn, and G. Zaverucha, "Algebraic MACs and Keyed-Verification Anonymous Credentials," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, nov 2014. [Online]. Available: https://doi.org/10.1145%2F2660267.2660328

[42] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure Applications of Pedersen's Distributed Key Generation Protocol," in *Topics in Cryptology — CT-RSA 2003*, M. Joye, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 373–390.

[43] S. Bowe, A. Gabizon, and I. Miers, "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model," 2017, https://eprint.iacr.org/2017/1050.

[44] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short Proofs for Confidential Transactions and More," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 315–334.

[45] S. Sinclair, "Israeli firm unveils tech allowing users to 'undo' erroneous ether transactions," 2020, https://www.coindesk.com/tech/2020/11/12/israeli-firm-unveils-tech-allowing-users-to-undo-erroneous-ether-transactior

[46] Keybase.io, "Keybase," http://keybase.io.

[47] E. Androulaki, J. Camenisch, A. D. Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann, "Privacy-preserving auditable token payments in a permissioned blockchain system," Cryptology ePrint Archive, Report 2019/1058, 2019, https://eprint.iacr.org/2019/1058.

[48] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," 2004. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA465464

[49] I. Abraham, B. Pinkas, and A. Yanai, "Blinder – Scalable, Robust Anonymous Committed Broadcast," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, oct 2020. [Online]. Available: https://doi.org/10.1145%2F3372297.3417261

[50] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and its Application to Anonymous Communication," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, nov 2019. [Online]. Available: https://doi.org/10.1145%2F3319535.3354238

[51] T. Sander, A. Ta-Shma, and M. Yung, "Blind, Auditable Membership Proofs," in *Financial Cryptography*, Y. Frankel, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 53–71.

[52] T. Ruffing, S. A. Thyagarajan, V. Ronge, and D. Schröder, "Burning Zerocoins for Fun and for Profit: A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol," Cryptology ePrint Archive, Report 2018/612, 2018, https://eprint.iacr.org/2018/612.

[53] A. Fiat and A. Shamir, "How To Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Advances in Cryptology — CRYPTO' 86*, A. M.

Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

[54] G. Maxwell, "Confidential transactions," 2015. [Online]. Available: https://elementsproject.org/features/confidential-transactions/investigation

[55] D. Pointcheval and O. Sanders, "Reassessing Security of Randomizable Signatures," Cryptology ePrint Archive, Report 2017/1197, 2017, https://eprint.iacr.org/2017/1197.

[56] J. Groth and M. Kohlweiss, "One-Out-of-Many Proofs: Or How to Leak a Secret and Spend a Coin," in *Advances in Cryptology - EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 253–280.

[57] G. Fuchsbauer, M. Orrù, and Y. Seurin, "Aggregate Cash Systems: A Cryptographic Investigation of Mimblewimble," Cryptology ePrint Archive, Report 2018/1039, 2018, https://eprint.iacr.org/2018/1039.

[58] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn, and G. Danezis, "Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers," in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/coconut-threshold-issuance-selective-disclosure-credentials-with-applications-to-distributed-ledgers/

[59] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A New Design for Anonymous Cryptocurrencies," in *Advances in Cryptology – ASIACRYPT 2019*, S. D. Galbraith and S. Moriai, Eds. Cham: Springer International Publishing, 2019, pp. 649–678.

[60] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards Privacy in a Smart Contract World," in *Financial Cryptography and Data Security*, J. Bonneau and N. Heninger, Eds. Cham: Springer International Publishing, 2020, pp. 423–443.

[61] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "ZEXE: Enabling Decentralized Private Computation," in *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020, pp. 947–964.

[62] M. Campanelli and M. Hall-Andersen, "Veksel: Simple, Efficient, Anonymous Payments with Large Anonymity Sets from Well-Studied Assumptions," Cryptology ePrint Archive, Report 2021/327, 2021, https://eprint.iacr.org/2021/327.

[63] N. van Saberhagen, "CryptoNote v2.0," 2013. [Online]. Available: https://web.archive.org/web/20201028121818/https://cryptonote.org/whitepaper.pdf

[64] S. Noether, "Ring Signature Confidential Transactions for Monero," Cryptology ePrint Archive, Report 2015/1098, 2015, https://ia.cr/2015/1098.

[65] T. P. Pedersen, "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing," in *Proceedings on Advances in Cryptology*, ser. CRYPTO '91, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140.

[66] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," *SIAM Journal on Computing*, vol. 32, no. 3, pp. 586–615, 2003. [Online]. Available: https://doi.org/10.1137/S0097539701398521

[67] S. Srinivasan, "New Security Notions for Identity Based Encryption," Ph.D. dissertation, 2010.

[68] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 526–545.

[69] D. Boneh, B. Fisch, A. Gabizon, and Z. Williamson, "A Simple Range Proof From Polynomial Commitments," 2020, https://hackmd.io/@dabo/B1U4kx8XI.

[70] M. Chase and A. Lysyanskaya, "On Signatures of Knowledge," Cryptology ePrint Archive, Report 2006/184, 2006, https://ia.cr/2006/184.

[71] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[72] Y. Dodis and A. Yampolskiy, "A Verifiable Random Function with Short Proofs and Keys," in *Public Key Cryptography - PKC 2005*, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 416–431.

[73] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A Scalable and Decentralized Trust Infrastructure," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2019, pp. 568–580.

[74] P. S. L. M. Barreto and M. Naehrig, "Pairing-Friendly Elliptic Curves of Prime Order," in *Selected Areas in Cryptography*, B. Preneel and S. Tavares, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 319–331.

[75] "github - scipr-lab/libff: C++ library for finite fields and elliptic curves 2021," 2021. [Online]. Available: https://github.com/scipr-lab/libff

[76] "vmware/concord-bft: Concord byzantine fault tolerant state machine replication library," 2021. [Online]. Available: https://github.com/vmware/concord-bft

[77] "vmware/concord-bft/bftengine/src/preprocessor/," 2020. [Online]. Available: https://github.com/vmware/concord-bft/pull/346

[78] "github - facebook/rocksdb: a library that provides an embeddable, persistent key-value store for fast storage." 2021. [Online]. Available: https://github.com/facebook/rocksdb/

[79] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, 2016, pp. 839–858.

[80] R. 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*. IEEE Computer Society, 2001, pp. 136–145. [Online]. Available: https://doi.org/10.1109/SFCS.2001.959888

[81] J. Camenisch, M. Drijvers, and B. Tackmann, "Multi-protocol UC and its use for building modular and efficient protocols," *IACR Cryptol. ePrint Arch.*, p. 65, 2019. [Online]. Available: https://eprint.iacr.org/2019/065

[82] G. Oded, *Foundations of Cryptography: Basic Tools*, 1st ed. USA: Cambridge University Press, 2003.

[83] E. Kushilevitz, Y. Lindell, and T. Rabin, "Information-theoretically secure protocols and security under composition," *SIAM J. Comput.*, vol. 39, no. 5, pp. 2090–2112, 2010. [Online]. Available: https://doi.org/10.1137/090755886

[84] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," *IACR Cryptol. ePrint Arch.*, p. 310, 2011. [Online]. Available: http://eprint.iacr.org/2011/310

[85] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure multi-party computation with identifiable abort," *IACR Cryptol. ePrint Arch.*, p. 325, 2015. [Online]. Available: http://eprint.iacr.org/2015/325

[86] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, "Can a Public Blockchain Keep a Secret?" in *Theory of Cryptography*, R. Pass and K. Pietrzak, Eds. Cham: Springer International Publishing, 2020, pp. 260–290.

[87] J. Benaloh and M. de Mare, "One-Way Accumulators: A Decentralized Alternative to Digital Signatures," in *EUROCRYPT '93*, T. Helleseth, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 274–285.

[88] T. Hanke, M. Movahedi, and D. Williams, "DFINITY Technology Overview Series Consensus System," https://dfinity.org/pdf-viewer/pdfs/viewer?file=../library/dfinity-consensus.pdf, 2018.

[89] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, oct 2017. [Online]. Available: https://doi.org/10.1145%2F3132747.3132757

[90] B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely, "Proofs for Inner Pairing Products and Applications," Cryptology ePrint Archive, Report 2019/1177, 2019, https://eprint.iacr.org/2019/1177.

[91] Electric Coin Company, "Zcash Feature UX Checklist," 2021, https://zcash.readthedocs.io/en/latest/rtd_pages/ux_wallet_checklist.html.

[92] A. Kate, G. M. Zaverucha, and I. Goldberg, "Polynomial commitments," Tech. Rep., 2010. [Online]. Available: https://pdfs.semanticscholar.org/31eb/add7a0109a584cfbf94b3afaa3c117c78c91.pdf

[93] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, Eds. ACM, 1993, pp. 62–73.

[94] G. Fuchsbauer, E. Kiltz, and J. Loss, "The algebraic group model and its applications," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10992. Springer, 2018, pp. 33–62.

[95] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discret. Appl. Math.*, vol. 156, no. 16, pp. 3113–3121, 2008.

[96] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed., vol. 576. Springer, 1991, pp. 129–140.

[97] P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi, "Get shorty via group signatures without encryption," in *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, ser. Lecture Notes in Computer Science, J. A. Garay and R. D. Prisco, Eds., vol. 6280. Springer, 2010, pp. 381–398. [Online]. Available: https://doi.org/10.1007/978-3-642-15317-4_24

[98] D. Boneh and X. Boyen, "Short signatures without random oracles and the SDH assumption in bilinear groups," *J. Cryptol.*, vol. 21, no. 2, pp. 149–177, 2008. [Online]. Available: https://doi.org/10.1007/s00145-007-9005-7

[99] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *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*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 2248. Springer, 2001, pp. 566–582. [Online]. Available: https://doi.org/10.1007/3-540-45682-1_33

[100] S. Srinivasan, "New security notions for identity based encryption," PHD, Royal Holloway, University of London, 2010. [Online]. Available: www.isg.rhul.ac.uk/~kp/theses/SSthesis.pdf

[101] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 537–554. [Online]. Available: https://doi.org/10.1007/3-540-48405-1_34

[102] ——, "How to enhance the security of public-key encryption at minimum cost," in *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*, ser. Lecture Notes in Computer Science, H. Imai and Y. Zheng, Eds., vol. 1560. Springer, 1999, pp. 53–68. [Online]. Available: https://doi.org/10.1007/3-540-49162-7_5

[103] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, 2003. [Online]. Available: https://doi.org/10.1137/S0097539701398521

[104] D. Bernhard, M. Fischlin, and B. Warinschi, "Adaptive proofs of knowledge in the random oracle model," *IET Inf. Secur.*, vol. 10, no. 6, pp. 319–331, 2016. [Online]. Available: https://doi.org/10.1049/iet-ifs.2015.0506

[105] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, ser. Lecture Notes in Computer Science, A. M. Odlyzko, Ed., vol. 263. Springer, 1986, pp. 186–194. [Online]. Available: https://doi.org/10.1007/3-540-47721-7_12

[106] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004. [Online]. Available: http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html

$\underline{\mathsf{CM.Setup}(1^\lambda, \ell; \mathbf{y})} \to \mathsf{ck}$ // $\mathbf{y}$ *is uniform in* $\mathbb{Z}_p^\ell$
$(g, \tilde{g}) \leftarrow\$ \mathbb{G}_1 \times \mathbb{G}_2 \quad \mathbf{g} \leftarrow g^{\mathbf{y}} \quad \tilde{\mathbf{g}} \leftarrow \tilde{g}^{\mathbf{y}} \quad \mathsf{ck} \leftarrow (g, \mathbf{g}, \tilde{g}, \tilde{\mathbf{g}})$

$\underline{\mathsf{CM.Com}_{\mathsf{ck}}(\mathbf{m} = (m_1, \ldots, m_\ell); r)} \to (\mathsf{cm}, \widetilde{\mathsf{cm}})$
Parse ck as $(g, \mathbf{g}, \tilde{g}, \tilde{\mathbf{g}}) \quad (\mathsf{cm}, \widetilde{\mathsf{cm}}) \leftarrow (\mathbf{g}^{\mathbf{m}} g^r, \tilde{\mathbf{g}}^{\mathbf{m}} \tilde{g}^r)$

$\underline{\mathsf{CM.Rerand}_{\mathsf{ck}}((\mathsf{cm}, \widetilde{\mathsf{cm}}); r)} \to (\mathsf{cm}', \widetilde{\mathsf{cm}}') \in \mathbb{G}_1 \times \mathbb{G}_2$
Parse ck as $(g, \cdot, \tilde{g}, \cdot)$ and $(\mathsf{cm}', \widetilde{\mathsf{cm}}') \leftarrow (\mathsf{cm} \cdot g^r, \widetilde{\mathsf{cm}} \cdot \tilde{g}^r)$

Figure 3: Algorithms for Pedersen commitments to ℓ messages, parameterized by the commitment key (CK) ck. We sometimes abuse notation and do not specify the CK; or only specify a $\mathbb{G}_i$ CK for non-dual commitments in $\mathbb{G}_i$.

$\underline{\mathsf{RS.KeyGen}(1^\lambda, \mathsf{ck})} \to (\mathsf{sk}, \mathsf{vk})$
Parse ck as $(g, \cdot, \tilde{g}, \cdot)$ and $x \leftarrow\$ \mathbb{Z}_p \quad (\mathsf{sk}, \mathsf{vk}) \leftarrow (g^x, \tilde{g}^x)$

$\underline{\mathsf{RS.Sign}_{\mathsf{ck}}(\mathsf{sk}, \mathsf{cm}; u)} \to \sigma$
// *Assuming previously-checked ZKPoK of opening on* cm *w.r.t.* ck
Parse ck as $(g, \cdot, \tilde{g}, \cdot)$ and $\sigma \leftarrow (g^u, (\mathsf{sk} \cdot \mathsf{cm})^u)$

$\underline{\mathsf{RS.Ver}_{\mathsf{ck}}(\mathsf{vk}, (\mathsf{cm}, \widetilde{\mathsf{cm}}), \sigma)} \to \{0, 1\}$
// *Implicitly verifies ZKPoK for* $\mathsf{cm} = g^r \prod_{k \in [\ell]} g_k^{m_k}$
Parse ck as $(g, \cdot, \tilde{g}, \cdot)$ and **assert** $e(\mathsf{cm}, \tilde{g}) = e(g, \widetilde{\mathsf{cm}})$
Parse σ as (h, σ_2) and **assert** $e(\sigma_2, \tilde{g}) = e(h, \mathsf{vk} \cdot \widetilde{\mathsf{cm}})$

$\underline{\mathsf{RS.Rerand}(\sigma; r_\Delta, u_\Delta)} \to \sigma'$
Parse σ as (h, σ_2) and $\sigma' \leftarrow (h^{u_\Delta}, (\sigma_2 \cdot h^{r_\Delta})^{u_\Delta})$

Figure 4: Algorithms for Pointcheval-Sanders (PS) signatures over Pedersen commitments from Fig. 3. (For the threshold variant, see Fig. 5.)

A Transaction fees and change

To support fees, we adjust our value preservation invariant to be $v_1 + v_2 = v_B + v_C + v_{\mathsf{fee}}$, where v_{fee} is the transaction fee. Note that the replicas can verify the fee was paid by multiplying the right-hand side of Eq. 17 by $g_3^{v_{\mathsf{fee}}}$:

$$\prod_{i \in \{1,2\}} \mathsf{vcm}_i = g_3^{v_{\mathsf{fee}}} \cdot \prod_{j \in \{A,B\}} \mathsf{vcm}_j \Leftrightarrow \tag{33}$$

$$v_1 + v_2 \equiv v_{\mathsf{fee}} + v_A + v_B \pmod{p} \tag{34}$$

Note that v_{fee} cannot depend on the transferred amounts, since that would break anonymity. To deal with this, we simply enforce the same fee for all anonymous transactions, similar to existing systems [91].

B Cryptography Background

Notation. $g^{\mathbf{x}} = [g^{x_1}, \ldots, g^{x_\ell}]$, where $g \in \mathbb{G}$, and $\mathbf{x} \in \mathbb{Z}_p^\ell$.

Re-randomizable signatures over commitments. We describe our dual Pedersen commitments in Fig. 3. We give our threshold variant of Pointcheval-Sanders in Fig. 5. For completeness, the single-signer version is in Fig. 4.

How to verify signature shares. To see why the $\mathsf{RS.ShareVer}$ share verification algorithm from Fig. 5 works, consider the left-hand side (LHS) of the verification equation:

$$\mathsf{lhs} = e([\sigma^*]_{i,2}, \tilde{g}) = \tag{35}$$

$$= e(h^{[x]_i + \sum_{k \in [\ell]} m_k [y_k]_i} \cdot g^{\sum_{k \in [\ell]} r_k [y_k]_i}, \tilde{g}) = \tag{36}$$

$$= e(h^{[x]_i}, \tilde{g}) e(h^{\sum_{k \in [\ell]} m_k [y_k]_i} \cdot g^{\sum_{k \in [\ell]} r_k [y_k]_i}, \tilde{g}) = \tag{37}$$

$$= e(h, \tilde{g}^{[x]_i}) e(\prod_{k \in [\ell]} h^{m_k [y_k]_i} \cdot \prod_{k \in [\ell]} g^{r_k [y_k]_i}, \tilde{g}) = \tag{38}$$

$$= e(h, \tilde{g}^{[x]_i}) e\left(\prod_{k \in [\ell]} \left(h^{m_k [y_k]_i} \cdot g^{r_k [y_k]_i}\right), \tilde{g}\right) = \tag{39}$$

$$= e(h, \tilde{g}^{[x]_i}) \prod_{k \in [\ell]} e\left(h^{m_k [y_k]_i} \cdot g^{r_k [y_k]_i}, \tilde{g}\right) = \tag{40}$$

$$= e(h, \tilde{g}^{[x]_i}) \prod_{k \in [\ell]} e\left((h^{m_k} g^{r_k})^{[y_k]_i}, \tilde{g}\right) = \tag{41}$$

$$= e(h, \tilde{g}^{[x]_i}) \prod_{k \in [\ell]} e\left(\mathsf{cm}_k, \tilde{g}^{[y_k]_i}\right) = \mathsf{rhs} \tag{42}$$

Identity-based encryption (IBE). We thresholdize an anonymous, CCA-secure variant of Boneh-Franklin IBE in Fig. 6.

Range proofs. In our protocol, we often need to prove that $\mathsf{cm} = g_1^v g^r$ is a commitment to a value $v \in [0, 2^N)$ for some positive integer N. For this, we use *zero-knowledge range proofs (ZKRPs)* [44] and, in particular, we use the protocol by Boneh et al [69] from Fig. 7.

B.1 Σ-protocols

As explained in §§4 and 5, UTT uses Σ-protocols to prove certain relations hold over the discrete log of group elements.

ZKPoK of Pedersen openings. We often use $\pi^{\mathsf{zkpok}} = \mathsf{ZK.Prove}_{\mathsf{PedOpen}}(\mathsf{ck}, \mathsf{cm}, [m_1, \ldots, m_\ell], r)$ to prove knowledge of an opening of $\mathsf{cm} = \prod_i g_i^m g^r$ under commitment key $\mathsf{ck} = (g, (g_1, \ldots, g_\ell))$. A verifier can check this proof via $\mathsf{ZK.Ver}_{\mathsf{PedOpen}}(\mathsf{ck}, \mathsf{cm}, \pi^{\mathsf{zkpok}})$. We often use this as a tool to prove that one (or more) of the m_i's are zero, by running the prover and verifier with an adjusted ck' that does not have the corresponding g_i's from ck. We refer to this as a *ZKPoK "for zeros"*

ZKPoK during registration and budget issuance. When first registering, Alice has to send $\mathsf{cm} = \mathsf{CM.Com}_{\mathsf{rck}}([0, s]; a)$ to the RA and prove knowledge of an opening $[\mathsf{pid}, s]$ of cm such that $\mathsf{pid} = 0$ (see §4.2). She can do so using the ZKPoK "for zeros" from above. Similarly, when asking for her budget Alice has to prove her rcm contains her pid, which she can do using a ZKPoK for "zeros" on $\mathsf{rcm}^* = \mathsf{rcm} \cdot \mathsf{CM.Com}([-\mathsf{pid}, 0]; 0)$. Then, the auditor can reconstruct rcm from rcm^* and pid and verify its registration signature rs.

ZKPoK for identity commitments. When creating a TXN, Alice has to compute π_j^{zkpok}'s proofs of knowledge of opening for the recipients' icm_j's from Eq. 32. She can do so easily using $\pi_j^{\mathsf{zkpok}} \leftarrow \mathsf{ZK.Prove}_{\mathsf{PedOpen}}(\mathsf{tck}, \mathsf{icm}_j, [\mathsf{pid}_j], t'_j)$, where $\mathsf{tck}_j = (h_j, g)$ is the commitment key pre-agreed upon by Alice and the bank for the current transaction. (Recall from §5, that Alice must use a different commitment key for the purpose of accommodating our threshold PS signatures.)

Pedersen equality proof. Recall from §5 that, when creating a TXN, Alice has to compute π_j^{pedeq}'s proofs that the recipients' $(\mathsf{vcm}_j, \mathsf{vcm}_j^*)$ from Eq. 32 both commit to the same value v_j. This can be done using a Σ-protocol for the relation $\mathcal{R}_{\mathsf{pedeq}}(\mathsf{ck}_1, \mathsf{cm}_1, \mathsf{ck}_2, \mathsf{cm}_2)$ holding when:

$$\left(\begin{array}{l} \mathsf{cm}_1 = \mathsf{CM.Com}_{\mathsf{ck}_1}(v; r_1) \\ \mathsf{cm}_2 = \mathsf{CM.Com}_{\mathsf{ck}_2}(v; r_2) \end{array} \right) \tag{43}$$

We will give details in the extended version of this paper.

ZKPoK on expiration date. Alice can easily prove her budget coin has expiration date zero using the ZKPoK "for zeros" from above. Note that Alice only proves this for $\mathsf{ccm} \in \mathbb{G}_1$ since the pairing-based check in $\mathsf{RS.Ver}$, which the bank performs when verifying the coin signature, implies this holds for $\widetilde{\mathsf{ccm}} \in \mathbb{G}_2$.

Splitproof. Recall that our split relation $\mathcal{R}_{\mathsf{split}}$ was decomposed into a randomized nullifier check (see Eq. 28) and an *inner* split relation $\mathcal{R}^*_{\mathsf{split}}$ (see Eq. 31). Since $\mathcal{R}^*_{\mathsf{split}}$ only reasons about the discrete logs of group elements, we can use a Σ-protocol to prove it. Importantly, recall from "Step 4" in §4.4, that we compute this proof as *ZK signature of knowledge (ZKSoK)*. For this, we additionally hash all transaction outputs when deriving the Σ-protocol challenge using the Fiat-Shamir transform [53]. For brevity, we will give details in the extended version of this paper.

Budget proof. The proof for the budget relation in Eq. 24 consists of two Pedersen equality proofs, which we gave in Eq. 43

Pedersen-KZG agreement proof. Our range proofs from Fig. 7 require proving, in ZK, that a KZG committed polynomial γ evaluated at a point i equals a Pedersen-committed value v with randomness r. Kate et al. give such a protocol in [92], which we reuse in our work. The proof is computed via $\pi^{\mathsf{pedkzg}} \leftarrow \mathcal{P}_{\mathsf{PedKZG}}(\mathsf{kpp}, \mathsf{ck}, v, r, \gamma, 1)$ and and verified as $\mathcal{V}_{\mathsf{PedKZG}}(\mathsf{kpp}, \mathsf{ck}, \mathsf{cm}, c_\gamma, 1, \pi_{\mathsf{pedkzg}})$, where kpp are the KZG public parameters and ck is the commitment key of the Pedersen commitment. We will give its details in the extended version of this paper.

B.2 Nullifier soundness and ZK

In this subsection, we argue our randomized nullifier verification from Eq. 28 is *sound* (i.e., proves the nullifier is correctly computed) and *zero-knowledge* (i.e., does not leak anything about s or sn).

Soundness. Recall that a nullifier on serial number

sn under PRF key s is computed as:

$$\mathsf{nullif} = h^{1/(s+\mathsf{sn})} \tag{44}$$

$$\mathsf{vk} = \tilde{h}^{s+\mathsf{sn}}\tilde{w}^t \tag{45}$$

$$y = e(\mathsf{nullif}, \tilde{w})^t \tag{46}$$

Here, t is secret randomness from $\mathbb{Z}_p$ and $(\tilde{h}, \tilde{w}) \in \mathbb{G}_2^2$ are part of the public parameters. The bank checks the nullifier as:

$$e(\mathsf{nullif}, \mathsf{vk}) = e(h, \tilde{h}) \cdot y \Leftrightarrow \tag{47}$$

$$e(\mathsf{nullif}, \tilde{h}^{s+\mathsf{sn}}\tilde{w}^t) = e(h, \tilde{h}) \cdot e(\mathsf{nullif}, \tilde{w}^t) \Leftrightarrow \tag{48}$$

$$e(\mathsf{nullif}, \tilde{h}^{s+\mathsf{sn}}) \cdot e(\mathsf{nullif}, \tilde{w}^t) = e(h, \tilde{h}) \cdot e(\mathsf{nullif}, \tilde{w}^t) \tag{49}$$

We can show that, when $s \neq \mathsf{sn}$, this implies $\mathsf{nullif} = h^{1/(s+\mathsf{sn})}$. Assume $\mathsf{nullif} = h^a$, for some $a \in \mathbb{Z}_p$. Then, since the check above holds, we have:

$$e(h^a, \tilde{h}^{s+\mathsf{sn}}) \cdot e(h^a, \tilde{w}^t) = e(h, \tilde{h}) \cdot e(h^a, \tilde{w}^t) \Leftrightarrow \tag{50}$$

$$e(h, \tilde{h})^{a(s+\mathsf{sn})} = e(h, \tilde{h}) \Leftrightarrow \tag{51}$$

$$a(s + \mathsf{sn}) = 1 \Leftrightarrow \tag{52}$$

$$a = 1/(s + \mathsf{sn}) \tag{53}$$

Thus, $\mathsf{nullif} = h^a = h^{1/(s+\mathsf{sn})}$ as desired, which means our randomized verification is sound.

Zero-knowledge. Regarding why this is zero knowledge, there exists a simulator that given any $\mathsf{nullif} \in \mathbb{G}_1$, can simulate a proof as:

- Pick random $\mathsf{vk} \in \mathbb{G}_2$,
- Let $y = e(\mathsf{nullif}, \mathsf{vk})/e(h, \tilde{h}) \in \mathbb{G}_T$,
- Simulate a Σ-protocol proof that argue correctness of vk and y.

// t is the corruption threshold
$\underline{\mathsf{RS.DistKeyGen}(1^\lambda, t+1, n, \ell) \to (\mathsf{ck}, \mathsf{vk}, (\mathsf{sk}_i, \mathsf{vk}_i)_{i\in[n]})}$
$\chi \leftarrow\$ \mathbb{Z}_p[X], \psi_k \leftarrow\$ \mathbb{Z}_p[X]$ be degree t polynomials, $\forall k \in [\ell]$
$x \leftarrow \chi(0) \quad y_k \leftarrow \psi_k(0), \quad \forall k \in [\ell]$
$[x]_i \leftarrow \chi(i) \quad [y_k]_i \leftarrow \psi_k(i), \quad \forall k \in [\ell], \forall i \in [n]$
// Note that $g_k = g^{y_k}$ and $\tilde{g}_k = \tilde{g}^{y_k}$
$\mathsf{vk} \leftarrow \tilde{g}^x \quad \mathsf{ck} \leftarrow \mathsf{CM.Setup}(1^\lambda, \ell; \mathbf{y})$ and parse as $(g, \mathbf{g}, \tilde{g}, \tilde{\mathbf{g}})$
$\mathsf{sk}_i \leftarrow ([x]_i, ([y_k]_i)_{k\in[\ell]}) \quad \mathsf{vk}_i \leftarrow \left(\tilde{g}^{[x]_i}, (\tilde{g}^{[y_k]_i})_{k\in[\ell]}\right), \forall i \in [n]$
return $\mathsf{ck}, \mathsf{vk}, (\mathsf{sk}_i, \mathsf{vk}_i)_{i\in[n]}$

$\underline{\mathsf{RS.ShareSign}_{\mathsf{ck}}(\mathsf{sk}_i, (\mathsf{cm}_k, \pi_k^{\mathsf{zkpok}})_{k\in[\ell]}; h) \to [\sigma^*]_i}$
// Assumes h uniform in $\mathbb{G}_1$; picked pseudo-randomly in practice
Parse ck as $(g, \cdot, \cdot, \cdot)$
// Check π_k^{zkpok} of opening $\mathsf{cm}_k = h^{m_k} g^{r_k}$
assert $\mathsf{ZK.Ver}_{\mathsf{PedOpen}}((h, g), \mathsf{cm}_k, \pi_k^{\mathsf{zkpok}})$ *// see App. B.1*
Parse sk_i as $[x]_i, ([y_k]_i)_{k\in[\ell]}$
$[\sigma^*]_i \leftarrow (h, h^{[x]_i} \prod_{k\in[\ell]} \mathsf{cm}_k^{[y_k]_i})$
*// i.e., **return** $(h, h^{[x]_i + \sum_{k\in[\ell]} m_k [y_k]_i} \cdot g^{\sum_{k\in[\ell]} r_k [y_k]_i})$*

$\underline{\mathsf{RS.ShareVer}_{\mathsf{ck}}(\mathsf{vk}_i, (\mathsf{cm}_k, \pi_k^{\mathsf{zkpok}})_{k\in[\ell]}, [\sigma^*]_i; h) \to \{0, 1\}}$
Parse $[\sigma^*]_i$ as $([\sigma^*]_{i,1}, [\sigma^*]_{i,2})$ and **assert** $h = [\sigma^*]_{i,1}$
Parse ck as $(g, \cdot, \tilde{g}, \cdot)$
// Check π_k^{zkpok} of opening $\mathsf{cm}_k = h^{m_k} g^{r_k}$
assert $\mathsf{ZK.Ver}_{\mathsf{PedOpen}}((h, g), \mathsf{cm}_k, \pi_k^{\mathsf{zkpok}})$ *// see App. B.1*
Parse vk_i as $\left(\tilde{g}^{[x]_i}, (\tilde{g}^{[y_k]_i})_{k\in[\ell]}\right)$
assert $e([\sigma^*]_{i,2}, \tilde{g}) = e(h, \tilde{g}^{[x]_i}) \cdot \prod_{k\in[\ell]} e(\mathsf{cm}_k, \tilde{g}^{[y_k]_i})$
// i.e., see Eq. 35 for why this works

$\underline{\mathsf{RS.Aggregate}_{\mathsf{ck}}((([\sigma^*]_i)_{i\in S}, (r_k)_{k\in[\ell]}) \to \sigma}$
Parse ck as $(\cdot, \mathbf{g}, \cdot, \cdot)$ and, $\forall i \in S$, parse $[\sigma^*]_i$ as $(h, [\sigma^*]_{i,2})$
$\mathcal{L}_i \leftarrow \prod_{j\in S, j\neq i} \frac{0-j}{i-j}, \forall i \in S$
// Lagrange interpolate the signature from the $|S| = t+1$ sigshares
$\sigma_2 \leftarrow \prod_{i\in S} ([\sigma^*]_{i,2})^{\mathcal{L}_i}$
// i.e., $(h, h^{x + \sum_{k\in[\ell]} m_k y_k} \cdot g^{\sum_{k\in[\ell]} r_k y_k})$, where $g_k = g^{y_k}$
$\sigma \leftarrow (h, \sigma_2 / \prod_{k\in[\ell]} g_k^{r_k})$ *// i.e., $(h, h^{x + \sum_{k\in[\ell]} m_k y_k})$*

Figure 5: Algorithms for Pointcheval-Sanders (PS) **threshold** signatures over Pedersen commitments from Fig. 3.

$\underline{\mathsf{IBE.Setup}(1^\lambda, t+1, n)} \to \mathsf{mpk}, (\mathsf{msk}_i, \mathsf{mpk}_i)_{i\in[n]}$
Let $\mathcal{H}_{\mathsf{IBE,id}} : \mathcal{P} \to \mathbb{G}_2$,
  $\mathcal{H}_{\mathsf{IBE,r}} : \mathbb{G}_2 \times \mathcal{P} \times \{0,1\}^{\mathsf{M}} \times \{0,1\}^{\mathsf{R}} \to \mathbb{Z}_p$, and
  $\mathcal{H}_{\mathsf{IBE,T}} : \mathbb{G}_T \to \{0,1\}^{\mathsf{M+R}}$ be CRHFs
$\phi \leftarrow\$\ \mathbb{Z}_p[X]$ be a degree t polynomial
$G_1 \leftarrow\$\ \mathbb{G}_1$
$\mathsf{mpk} \leftarrow G_1^{\phi(0)} \quad \forall i \in [n], \mathsf{msk}_i \leftarrow \phi(i) \quad \mathsf{mpk}_i \leftarrow G_1^{\mathsf{msk}_i}$

$\underline{\mathsf{IBE.ExtractShare}(\mathsf{msk}_i, \mathsf{pid})} \to \mathsf{sk}_i$
$\mathsf{sk}_i \leftarrow \mathcal{H}_{\mathsf{IBE,id}}(\mathsf{pid})^{\mathsf{msk}_i} \in \mathbb{G}_2$
// **Note**: IBE.Extract(msk, pid) = IBE.ExtractShare(msk, pid)

$\underline{\mathsf{IBE.VerShare}(\mathsf{mpk}_i, \mathsf{sk}_i, \mathsf{pid})} \to \{0,1\}$
assert $e(G_1, \mathsf{sk}_i) = e(\mathsf{mpk}_i, \mathcal{H}_{\mathsf{IBE,id}}(\mathsf{pid}))$

$\underline{\mathsf{IBE.Aggregate}((\mathsf{sk}_i)_{i\in S})} \to \mathsf{sk}$
// *Lagrange interpolate the* sk *from the* $|S| = t+1$ sk_i*'s*
$\mathcal{L}_i \leftarrow \prod_{j\in S, j\neq i} \frac{0-j}{i-j}, \forall i \in S$
$\mathsf{sk} \leftarrow \prod_{i\in S} (\mathsf{sk}_i)^{\mathcal{L}_i}$

$\underline{\mathsf{IBE.Enc}(\mathsf{mpk}, \mathsf{pid}, m)} \to \mathsf{ctxt}$
$\sigma \leftarrow\$\ \{0,1\}^{\mathsf{R}} \quad r \leftarrow \mathcal{H}_{\mathsf{IBE,r}}(\mathsf{mpk}, \mathsf{pid}, m, \sigma) \in \mathbb{Z}_p$
$T \leftarrow e(\mathsf{mpk}, \mathcal{H}_{\mathsf{IBE,id}}(\mathsf{pid}))^r \in \mathbb{G}_T$
$\mathsf{ctxt} \leftarrow (G_1^r, (m||\sigma) \oplus \mathcal{H}_{\mathsf{IBE,T}}(T)) \in \mathbb{G}_1 \times \{0,1\}^{\mathsf{M+R}}$

$\underline{\mathsf{IBE.Dec}(\mathsf{sk}, \mathsf{ctxt})} \to \{0,1\}^{\mathsf{M}} \cup \{\perp\}$
Parse ctxt as $(c_1, c_2) \in \mathbb{G}_1 \times \{0,1\}^{\mathsf{M+R}}$
$T \leftarrow e(\mathsf{sk}, c_1) \in \mathbb{G}_T$ // *i.e.,* $e(\mathsf{mpk}, \mathcal{H}_{\mathsf{IBE,id}}(\mathsf{pid}))^r$
Parse $c_2 \oplus \mathcal{H}_{\mathsf{IBE,T}}(T)$ as $(m, \sigma) \in \{0,1\}^{\mathsf{M}} \times \{0,1\}^{\mathsf{R}}$
return m **if** $c_1 = G_1^{\mathcal{H}_{\mathsf{IBE,r}}(\mathsf{mpk},\mathsf{pid},m,\sigma)}$
otherwise, **return** $\perp$

Figure 6: Algorithms for threshold-issuance, anonymous, CCA-secure IBE by Boneh-Franklin [66, 67] over PID space $\mathcal{P}$, message space $\{0,1\}^{\mathsf{M}}$ and randomness space $\{0,1\}^{\mathsf{R}}$. Here, $\oplus$ denotes the "bitwise" exclusive OR operation.

$\underline{\mathsf{ZKRP.Prove}(\mathsf{rpp}, v, r)} \to \pi$
Parse rpp as $(\mathsf{vck}, \mathsf{kpp}, N)$ and v as $\sum_{i\in[0,N)} v_i 2^i$
$\mathsf{cm} \leftarrow \mathsf{CM.Com}_{\mathsf{vck}}(v; r)$
$(\omega', \omega'') \leftarrow \mathbb{Z}_p^2 \setminus H$ and let $S = \mathbb{Z}_p \setminus (H \cup \{\omega', \omega''\})$
Interpolate $\gamma(X) \in \mathbb{Z}_p[X]$ of deg-$(N+1)$ such that:
  $\gamma(\omega^{N-1}) = v_{N-1}$
  $\gamma(\omega^i) = 2\gamma(\omega^{i+1}) + v_i, \forall i \in [0, N-1)$
  $\gamma(\omega') \leftarrow\$\ \mathbb{Z}_p \qquad \gamma(\omega'') \leftarrow\$\ \mathbb{Z}_p$ // *for ZK*
// *ZK proof that* $\gamma(1) = v$ *is committed in* $\mathsf{cm} = g_3^v g^r$
$\pi_{\mathsf{pedkzg}} \leftarrow \mathcal{P}_{\mathsf{PedKZG}}(\mathsf{kpp}, \mathsf{vck}, v, r, \gamma, 1)$ // *see App. B.1*
// $w_2(X) = 0$ *over* $H \Leftrightarrow v_{N-1} \in \{0,1\}$
$w_2(X) \leftarrow \gamma \cdot (1-\gamma) \cdot \frac{X^N - 1}{X - \omega^{N-1}}$
// $w_3(X) = 0$ *over* $H \Leftrightarrow v_i \in \{0,1\}, \forall i \in [0, N-2]$
$w_3(X) \leftarrow [\gamma(X) - 2\gamma(X\omega)] \cdot [1 - (\gamma(X) - 2\gamma(X\omega))] \cdot$
  $\cdot (X - \omega^{N-1})$
// *Randomness picked via Fiat-Shamir transform*
$\tau \leftarrow \mathcal{H}_{\mathsf{zkrp}}(\mathsf{cm}, 1) \qquad \rho \leftarrow \mathcal{H}_{\mathsf{zkrp}}(\mathsf{cm}, 2)$
Compute $q \in \mathbb{Z}_p[X]$ such that $w_2 + \tau w_3 = q \cdot (X^N - 1)$
$c_q \leftarrow \mathsf{KZG.Commit}(\mathsf{kpp}, q) \quad c_\gamma \leftarrow \mathsf{KZG.Commit}(\mathsf{kpp}, \gamma)$
$\pi_{q(\rho)} \leftarrow \mathsf{KZG.Prove}(\mathsf{kpp}, q, \rho) \quad \pi_{\gamma(\rho)} \leftarrow \mathsf{KZG.Prove}(\mathsf{kpp}, \gamma, \rho)$
$\pi_{\gamma(\rho\omega)} \leftarrow \mathsf{KZG.Prove}(\mathsf{kpp}, \gamma, \rho\omega)$
return $c_\gamma, c_q, \pi_{\mathsf{pedkzg}}, \gamma(\rho), \pi_{\gamma(\rho)}, \gamma(\rho\omega), \pi_{\gamma(\rho\omega)}, q(\rho), \pi_{q(\rho)}$

$\underline{\mathsf{ZKRP.Setup}(1^\lambda, 2^N)} \to \{\mathsf{rpp}\}$
$(\mathsf{vck}, \widetilde{\mathsf{vck}}) \leftarrow\$\ \mathsf{CM.Setup}(1^\lambda, 2)$
$\mathsf{kpp} \leftarrow \mathsf{KZG.Setup}(1^\lambda, 3N+1)$
$\mathsf{rpp} \leftarrow (\mathsf{vck}, \mathsf{kpp}, N)$

$\underline{\mathsf{ZKRP.Ver}(\mathsf{rpp}, \mathsf{cm}, \pi)} \to \{0,1\}$
Parse rpp as $(\mathsf{vck}, \mathsf{kpp}, N)$
Parse π as $\left(\begin{array}{c} c_\gamma, c_q, \pi_{\mathsf{pedkzg}}, \gamma(\rho), \pi_{\gamma(\rho)}, \\ \gamma(\rho\omega), \pi_{\gamma(\rho\omega)}, q(\rho), \pi_{q(\rho)} \end{array}\right)$
// *verify ZK proof that* $\gamma(1) = v$ *is committed in* $\mathsf{cm} = g_3^v g^r$
assert $\mathcal{V}_{\mathsf{PedKZG}}(\mathsf{kpp}, \mathsf{vck}, \mathsf{cm}, c_\gamma, 1, \pi_{\mathsf{pedkzg}})$ // *see App. B.1*
// *Randomness picked via Fiat-Shamir transform*
$\tau \leftarrow \mathcal{H}_{\mathsf{zkrp}}(\mathsf{cm}, 1) \qquad \rho \leftarrow \mathcal{H}_{\mathsf{zkrp}}(\mathsf{cm}, 2)$
assert $\mathsf{KZG.Ver}(\mathsf{kpp}, c_\gamma, \rho, \gamma(\rho), \pi_{\gamma(\rho)})$
assert $\mathsf{KZG.Ver}(\mathsf{kpp}, c_\gamma, \rho\omega, \gamma(\rho\omega), \pi_{\gamma(\rho\omega)})$
assert $\mathsf{KZG.Ver}(\mathsf{kpp}, c_q, \rho, q(\rho), \pi_{q(\rho)})$
$w_2(\rho) \leftarrow \gamma(\rho) \cdot (1 - \gamma(\rho)) \cdot \frac{\rho^N - 1}{\rho - \omega^{N-1}}$
$w_3(\rho) \leftarrow [\gamma(\rho) - 2\gamma(\rho\omega)]\,[1 - (\gamma(\rho) + 2\gamma(\rho\omega))]\,(\rho - \omega^{N-1})$
assert $w_2(\rho) + \tau w_3(\rho) = q(\rho)(\rho^N - 1)$

Figure 7: Boneh et al's [69] range proofs for Pedersen commitments $g_1^v g^r$ to v. Here, $H = \{\omega^0, \omega^1, \ldots, \omega^{N-1}\}$ are all the Nth roots of unity and $\mathcal{H}_{\mathsf{zkrp}}$ is a CRHF.

Part I

A Cryptographic Analysis of the UTT system

We analyze the UTT system 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 standalone MPC for reactive (aka stateful) functionalities that captures the ability of the adversary to adaptively select the inputs of the parties, without resorting to the full-fledged UC model of [80]. 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 C we present the security model and the ideal $\mathcal{F}_{\mathsf{utt}}$ functionality. Section D 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 D.2). Our protocol appears in Section E and some extensions are presented in Section F. In Sections G and H we describe the simulator and analyze it.

C 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. For simplicity, our basic formulation does not include anonymity budgets, which are added later as a simple extension (See Remark C.3 and Section F.2.)

C.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.

FUNCTIONALITY C.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 $(\mathsf{pay}, t_1, t_2, \mathsf{pid}_1, v_1, \mathsf{pid}_2, v_2)$ 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.

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.

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.[6] 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.

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.

[6] 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.

Remark C.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 as described below. In Section F.2 we briefly explain how to adopt the protocol to this more general setting.

Remark C.3 (Supporting anonymity budgets)**.** One can naturally extend $\mathcal{F}_{\mathsf{utt}}$ to support anonymity budget as follows. The coin dictionary is augmented with an additional expi field that can be set to either 0 (for "regular" coins) or to some positive integer to indicate that this is an "anonymity-budget" coin whose expiration date is expi. The Mint operation takes expi as an additional input and sets the expi field of the generated coin accordingly. Depending on the context, we may add a special party, "the Auditor", that is responsible for minting anonymous-budget coins, and disallow other parties to issue such a mint operation. (Alternatively, we can let the Minter also take the role of the Auditor.)

We move on to describe the modified Pay operation. (To be aligned with the main text, we assume that the first target coin is always delivered to the Sender, i.e., $\mathsf{pid}_1 = \mathsf{pid}$.) The Pay operation takes, as an additional input, an identifier t_3 of a budget coin, and checks the following conditions (in addition to the original ones): (1) t_3 is owned by Sender, i.e., $\mathsf{owner}(t_3) = \mathsf{pid}$; (2) The coins t_1, t_2 are regular and the coin t_3 is an anonymity-budget coin, i.e., $\mathsf{expi}(t_1) = \mathsf{expi}(t_2) = 0$ and $\mathsf{expi}(t_3) \neq 0$; (3) The budget coin has not expired $\mathsf{date} \leq \mathsf{expi}(t_3)$ where date is a "public clock" that is available to all the parties;[7] (3) (Enough Budget) The money that is being delivered to the receiver pid_2 does not exceed the current anonymity budget, i.e., $v_2 \leq \mathsf{val}(t_3)$. If any of these conditions fail, we broadcast a failure notification. Otherwise, we complete the operation as described in Figure C.1, except that we issue a new anonymous budget coin of value $v_C := \mathsf{val}(t_3) - v_2$ (a "change") with expiration date of expi and deliver it to the sender (i.e., set its owner id to pid). The counter is increased accordingly.

[7] Formally, this can be captured by an external ideal (reactive) functionality that takes no input and returns the "current time".

C.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.[8] 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.[9] Following the UC model of [80], 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 [83–85]), 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

[8] 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.

[9] 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.

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 C.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.[10] 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 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

$$\left\{\mathsf{Exec}_{\Pi,\mathsf{Adv},\mathsf{Env}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \tag{54}$$

is indistinguishable from the ensemble of random variables

$$\left\{\mathsf{Exec}_{\mathcal{F}_{\mathsf{utt}},\mathcal{S},\mathsf{Env}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}. \tag{55}$$

The use of idealized oracles. Our standalone model can be easily extended to work with ideal oracles such as Random Oracles (RO) [93] or the algebraic group model (AGM) [94], 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 F.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 E.)

D 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 algo-

[10] 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.

rithms are determined by the public parameters.[11] 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 [95], together with a random generator $h \leftarrow\$ \mathbb{G}_1$ (to be used later as a public bases for the Dodis-Yampolskiy PRF [72], see Section D.4).

D.1 Homomorphic Commitment Scheme

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

Definition D.1 (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. $\mathsf{CM.Com}(\mathsf{pp}, \mathsf{ck}, m; r) \to \mathsf{cm}$. Given public parameters pp, a commitment key ck, a message $m \in \mathcal{M}$ and randomness $r \leftarrow\$ \mathcal{R}$, 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 $(\mathsf{pp}, \mathsf{ck}, \mathsf{cm}_0 = \mathsf{CM.Com}(\mathsf{pp}, \mathsf{ck}, m_0; r))$ and $(\mathsf{pp}, \mathsf{ck}, \mathsf{cm}_1 = \mathsf{CM.Com}(\mathsf{pp}, \mathsf{ck}, m_1; r))$ 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 $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$ and the commitment key $\mathsf{ck} \leftarrow\$ \mathsf{CM.Setup}(\mathsf{pp})$, that $\mathsf{Adv}(\mathsf{pp}, \mathsf{ck})$ outputs a tuple (m_0, r_0, m_1, r_1) for which

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

and

$$(m_0, r_0) \neq (m_1, r_1)$$

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

$$\mathsf{cm}_0 = \mathsf{CM.Com}(\mathsf{pp}, \mathsf{ck}, m_0; r_0)$$

and

$$\mathsf{cm}_1 = \mathsf{CM.Com}(\mathsf{pp}, \mathsf{ck}, m_1; r_1)$$

then

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

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 D.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 $\mathsf{CM.Com}(\mathsf{ck}, m; r) = \mathsf{cm}$.

Remark D.2 (Rerandomizing 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

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

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

Remark D.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).

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

Instantiation. We use a dual "bilinear" version of the well-known Pedersen commitment [96]. The binding property holds under the SDL assumption of [97] which follows from 1-SDH of [98]. (See Section I for details.) As already mentioned, the hiding property hold information theoretically.

D.2 Committed Signature Scheme

Our protocol makes extensive use of signatures over commitments schemes. The signatures should be rerandomizable and, in addition, one should be able to rerandomize 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 D.2 (Rerandomizable Signatures over Commitments)**.** A rerandomizable signature-commitment scheme is defined with respect to a given Commitment scheme $(\mathsf{CM.Setup}, \mathsf{CM.Com})$ that is equipped with a rerandomization procedure $\mathsf{CM.Rerand}$ (see Definition D.1 and Remark D.2) and consists of the following additional algorithms:

1. $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \rightarrow (\mathsf{sk}, \mathsf{pk} = (\mathsf{pp}, \mathsf{vk}, \mathsf{ck}))$. 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. $\mathsf{RS.Sign}(\mathsf{sk}, \mathsf{cm}; u) \rightarrow \sigma$. 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. $\mathsf{RS.Rerand}(\mathsf{pk}, \sigma; r_\Delta, u_\Delta) \rightarrow \sigma'$. 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 $\sigma = \mathsf{RS.Sign}(\mathsf{sk}, \mathsf{CM.Com}(\mathsf{ck}, m; r); u)$ for some message m and randomizer r then $\sigma' = \mathsf{RS.Sign}(\mathsf{sk}, \mathsf{CM.Com}(\mathsf{ck}, m; r+r_\Delta); u+u_\Delta)$ 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. $\mathsf{RS.Ver}(\mathsf{pk}, \mathsf{cm}, \sigma) \rightarrow \{0, 1\}$. 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, $\mathsf{GAME}_{\mathsf{EU-CCA,Adv}}$, with more than negligible probability in λ:

1. The Challenger samples $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$, $\mathsf{CM.Setup}(\mathsf{pp}) \rightarrow \mathsf{ck}$ and $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \rightarrow (\mathsf{sk}, \mathsf{pk})$ 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 $\mathsf{cm} = \mathsf{CM.Com}(\mathsf{ck}, m; r)$. 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 $\mathsf{RS.Ver}(\mathsf{pk}, \mathsf{CM.Com}(\mathsf{ck}, m^*; r^*), \sigma^*) = 1$ and $m^* \notin M$.

Instantiation. We use a variant of the Pointcheval-Sanders signature scheme [34] over dual-Pedersen commitments. In Section J we describe the construction (Figure 9) and prove the following lemma.

Lemma D.4. Under Assumption 1 of [34], the scheme presented in Figure 9 is a rerandomizable signature-commitment scheme.

D.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 [99] 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 [100] defined below.

Definition D.3 (IND-RA-CCA IBE)**.** An IBE scheme consists of the following algorithms:

1. $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\$ \mathsf{IBE.Setup}(\mathsf{pp})$. 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 associated with the identifier sk.

3. $c \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{id}, m)$. 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 $\mathsf{sk}_{\mathsf{id}} = \mathsf{IBE.Extract}(\mathsf{msk}, \mathsf{id})$.[12]

An IBE scheme is IND-RA-CCA secure [100, 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 $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$ and a pair of master public/secret keys $(\mathsf{msk}, \mathsf{mpk}) \leftarrow\$ \mathsf{IBE.Setup}(\mathsf{pp})$ 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 $b \leftarrow\$ \{0, 1\}$ and returns a fresh challenge ciphertext $c^* \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{id}_b, m_b)$.

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 D.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 mpk and pp.

Instantiation. It is shown in [100, Chapter 4] that if one applies a variant of the Fujisaki-Okamoto transform [101, 102] to the basic (CPA-secure) variant of the Boneh-Franklin IBE [103] the resulting IBE is IND-RA-CCA under the BDH assumption in the Radnom-Oracle model. We use this instantiation in our protocols.

D.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 D.4. 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

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

is $\mathsf{negl}(\lambda)$, where $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda), k \leftarrow\$ \mathcal{K}_{\mathsf{pp}}$ 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 $\mathsf{PRF}(\mathsf{pp}, k, x)$.

[12] 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.

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[13] 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 [72] that takes retrieves from the public parameters a public element $h \leftarrow\$ \mathbb{G}_1$ 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.

D.5 Zero-Knowledge Proofs

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

Definition D.5 (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.

We require perfect completeness,[14] 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 $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$ and a secret challenge bit $b \leftarrow\$ \{0,1\}$. If $b = 0$ let $\mathsf{zkpp} \leftarrow\$ \mathsf{ZK.Setup}(\mathsf{pp})$, otherwise let $\mathsf{zkpp} \leftarrow\$ \mathsf{ZK.Sim}(\mathsf{pp})$.
  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 $\Pi \leftarrow\$ \mathsf{ZK.Prove}(\mathsf{pp}, \mathsf{zkpp}, \mathbb{x}, \mathbb{w})$ if $b = 0$, and $\Pi \leftarrow\$ \mathsf{ZK.Sim}(\mathbb{x})$ 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.[15]

Knowledge extraction. The soundness follows from the following (stronger) notion of *straight-line proof of knowledge* relative to an oracle which is adopted from [104, 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:

[13] 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.

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

[15] 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.

- Sample $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$, $\mathsf{zkpp} \leftarrow\$ \mathsf{ZK.Setup}(\mathsf{pp})$, 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.

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 [105]. 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).[16]

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 [69]) are Schnorr-like Sigma protocols that satisfy the special soundness property. (See Section K.) It seems likely that, at least for these proofs, one can completely get rid of knowledge assumptions/AGM and rely solely on random oracles.

[16] 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.

E 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.[17] 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 (as described in the main body of the paper) 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 F.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 body of the paper. 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,

[17] 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.

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 D.3.)

E.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 $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$. 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 $\mathsf{ck} \leftarrow\$ \mathsf{CM.Setup}(\mathsf{pp})$ where ck is a commitment key for a homomorphic commitment (as per Definition D.1) 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 $\mathsf{s}_0 \leftarrow\$ \mathcal{K}_{\mathsf{pp}}$, and computes a "temporary" registration commitment

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

where $a \leftarrow\$ \mathcal{R}$ 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 $\mathsf{rcm}_0 = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0); a)$. The Bank then verifies the proof.

- If verification passes, the Bank samples a random shift $\Delta \leftarrow\$ \mathcal{K}_{\mathsf{pp}}$, computes a registration commitment $\mathsf{rcm} = \mathsf{rcm}_0 + \mathsf{CM.Com}(\mathsf{ck}, (0, 0, 0, \Delta); 0)$ (supposedly $\mathsf{rcm} = (\mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0 + \Delta); a))$, 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 $\mathsf{s} \leftarrow\$ \mathcal{K}_{\mathsf{pp}}$, $a \leftarrow\$ \mathcal{R}$, and sends back the tuple

  $$(\mathsf{s}, \quad a, \quad \mathsf{rcm}, \quad \mathsf{rs}),$$

  where $\mathsf{rcm} = (\mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a)$, $\mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b)$ and b is chosen at random. The client records her "address secret key" as $\mathsf{ask} = (\mathsf{s}, \mathsf{sk}^{\mathsf{ibe}}, \mathsf{rcm}, \mathsf{rs}, a)$.

E.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

$$\mathsf{ccm}_i = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}_i, \mathsf{val}_i, 0); r) \quad \text{and} \quad \sigma_i = \mathsf{RS.Sign}(\mathsf{bsk}, \mathsf{ccm}_i; u_i)) \tag{56}$$

for some (unknown but rerandomized) 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.

E.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.[18]

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

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

and

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

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 rerandomizes the registration signature $(\mathsf{rcm}, \mathsf{rs})$ into $(\mathsf{rcm}', \mathsf{rs}')$. This is done, by letting

$$\mathsf{rcm}' = \mathsf{CM.Rerand}(\mathsf{ck}, \mathsf{rcm}; a_\Delta)$$

and

$$\mathsf{rs}' = \mathsf{RS.Rerand}(\mathsf{pk}, \sigma; a_\Delta, u_\Delta)$$

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

$$\mathbb{x} = (\mathsf{ccm}_i, \mathsf{vcm}_i, \mathsf{rcm}', \mathsf{nullif}_i), \quad (57)$$

$$\mathbb{w} = (\mathsf{s}_{\mathsf{Sender}}, \mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, r_i, z_i, a') \quad (58)$$

for which the following conditions hold

$$\begin{aligned} \mathsf{ccm}_i &= \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, 0); r_i) \\ \mathsf{vcm}_i &= \mathsf{CM.Com}(\mathsf{ck}, (0, 0, \mathsf{val}_i, 0); z_i) \quad (59) \\ \mathsf{rcm}' &= \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_{\mathsf{Sender}}, 0, 0, \mathsf{s}_{\mathsf{Sender}}); a') \\ \mathsf{nullif}_i &= \mathsf{PRF}_{\mathsf{s}_{\mathsf{Sender}}}(\mathsf{sn}_i). \end{aligned}$$

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

$$\begin{aligned} \mathsf{icm}_j &= \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_j, 0, 0, 0); t_j) \quad (60) \\ \text{and} \quad \mathsf{vcm}_j &= \mathsf{CM.Com}(\mathsf{ck}, (0, 0, v_j, 0); \rho_j), \end{aligned}$$

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 Π_{icm_j}, and ZKPOK Π_{Range_j} for the knowledge of (v_j, ρ_j) that satisfies the *range* relation

$$\begin{aligned} \mathcal{R}_{\mathsf{range}} = \{(\mathbb{x} = \mathsf{vcm}_j; \mathbb{w} = (v_j, \rho_j)) : \quad (61) \\ \mathsf{vcm}_j = \mathsf{CM.Com}(\mathsf{ck}, (0, 0, v_j, 0); \rho_j) \\ \wedge\, v_j \in [0, \mathsf{V}_{\mathsf{max}}]\}. \end{aligned}$$

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

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

for which the following conditions hold

$$\begin{aligned} \mathsf{vcm}_i &= \mathsf{CM.Com}(\mathsf{ck}, (0, 0, \mathsf{val}_i, 0); z_i), \ i \in \{1,2\} \\ \mathsf{vcm}_j &= \mathsf{CM.Com}(\mathsf{ck}, (0, 0, v_j, 0); \rho_j), \ j \in \{B,C\} \\ \mathsf{val}_1 + \mathsf{val}_2 &= v_B + v_C. \quad (62) \end{aligned}$$

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.)

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

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

$$((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'), \\ (\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}}, \ \Pi_{\mathsf{Sum}} \tag{63}$$

(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 28.). For $j \in \{B, C\}$, the Bank approves the jth coin request as follows:
   (a) The Bank Homomorphically computes the commitment
   $$\mathsf{ccm}_j = \mathsf{icm}_j \boxplus \mathsf{sncm}_j \boxplus \mathsf{vcm}_j,$$
   where
   $$\mathsf{sncm}_j = \mathsf{CM.Com}(\mathsf{ck}, (0, \mathsf{sn}_j, 0, 0); 0).$$
   (Supposedly, ccm_j equals to $\mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_j, \mathsf{sn}_j, v_j, 0); \rho_j + t_j)$.)
   (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
   $$(t, \mathsf{nullif}_1, \mathsf{nullif}_2), \ (\mathsf{ccm}_j, \sigma_j, \mathsf{ctxt}_j) \ j \in \{B, C\}, \tag{64}$$
   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 the 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
   $$\mathsf{ccm}_j = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}, v, 0); r)$$
   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', rerandomize the signed committed coin $(\mathsf{ccm}_j, \sigma_j)$ into (ccm', σ') where

$$\mathsf{ccm}' = \mathsf{CM.Rerand}(\mathsf{pp}, \mathsf{ck}, \mathsf{ccm}_j; r'),$$

and

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

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

E.4 The minting sub-protocol

The minting protocol can be 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:

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

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 $\mathsf{ctxt} \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}, (v, r))$ 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

$$t, \quad (\mathsf{sn}, \mathsf{ccm}, \sigma, \mathsf{ctxt}) \tag{65}$$

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 $\mathtt{minted}$.)

F Extending the Protocol

F.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 G and H), and so, by using proper MPC composition theorems (e.g., [80,106]), 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 D.2), and consider the initialization functionality that given pp samples a commitment key $\mathsf{CM.Setup}(\mathsf{pp}) \rightarrow \mathsf{ck}$ and 2 pairs of signature/verification keys $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \rightarrow (\mathsf{bsk}, \mathsf{bvk})$ and $\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \rightarrow (\mathsf{rsk}, \mathsf{rvk})$, 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 commit-

ment cm together with its opening m, r such that $\mathsf{cm} = \mathsf{CM.Com}(\mathsf{pp}, \mathsf{ck}, m; r)$, delivers to the client a valid signature σ over cm. In the main body of the paper (Section 3) we present information-theoretic protocols that realize these functionalities with respect to a variant of the PS signatures [34] over "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. [19]

F.2 Realizing the generalized $\mathcal{F}_{\mathsf{utt}}$ functionality

Recall that in Remark C.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.

Furthermore, the variant of $\mathcal{F}_{\mathsf{utt}}$ that supports anonymity-budget (as outlined in Remark C.3) can be realized by a simple extension of the protocol as described in Section 4. It can be verified that our security proof (Sections G and H) naturally extends to this generalized version of the protocol provided that the corresponding ZK building blocks are sound.

[19] Additionally, in our implementation 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 [103]).

G 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 cases. Following the discussion in Section F.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.

G.1 Initialization

The simulator proceeds as follows:

1. Samples public parameters $\mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda)$, 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 $\mathsf{ck} \leftarrow\$ \mathsf{CM.Setup}(\mathsf{pp})$, 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 $\mathsf{s}_0 \leftarrow\$ \mathcal{K}_{\mathsf{pp}}$, and computes a commitment

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

where $a \leftarrow\$ \mathcal{R}$ 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 $\mathsf{rcm}_0 = (\mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}_0); a)$. 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 $\Delta \leftarrow\$ \mathcal{K}_{\mathsf{pp}}$, $\mathsf{rcm} = \mathsf{rcm}_0 + \mathsf{CM.Com}(\mathsf{ck}, (0, 0, 0, \Delta); 0)$ 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.[20]

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.

   (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 $\Delta \leftarrow\$ \mathcal{K}_{\mathsf{pp}}$, $\mathsf{rcm} = \mathsf{rcm}_0 + \mathsf{CM.Com}(\mathsf{ck}, (0, 0, 0, \Delta); 0)$ 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

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

   and

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

   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.

[20] 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.

G.2 Minting operation

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

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

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

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 $\mathsf{ctxt} \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}_0, (v_0, r))$ for the receiver, and sends the tuple $(\mathsf{sn}, \mathsf{ccm}, \mathsf{ctxt})$ to the Virtual Bank. The Virtual Bank continues just as 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.

G.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" `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 also sends to the simulator 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 $[0, \mathsf{V}_{\mathsf{max}}]$.)

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

$$((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),$$
$$(\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}}, \quad (66)$$
$$\Pi_{\mathsf{Sum}}$$

  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 $\mathsf{s}_0 \leftarrow\$\ \mathcal{K}_{\mathsf{pp}}$ for this fake user and compute a fresh commitment $\mathsf{rcm} = (\mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a)$. The adversary locally generates a fresh signature $\mathsf{rs} = \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm}; b)$ where b is chosen at random.[21]
  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

$$\mathsf{ccm}_i = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_0, \mathsf{sn}_i, v_i, 0); r_i),$$

  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 21.)
  3. (Split incoming coins) For $i \in \{1, 2\}$, set

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

  and

$$\mathsf{nullif}_i = \mathsf{PRF}_{\mathsf{s}_0}(\mathsf{sn}_i),$$

  where z_i is a fresh randomizer.
  4. (Randomize registration signatures) Rerandomize the registration signature $(\mathsf{rcm}, \mathsf{rs})$ into $(\mathsf{rcm}', \mathsf{rs}')$.[22]
  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. (59). 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

$$\mathsf{icm}_j = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}'_j, 0, 0, 0); t_j) \quad (67)$$
$$\text{and} \quad \mathsf{vcm}_j = \mathsf{CM.Com}(\mathsf{ck}, (0, 0, v'_j, 0); \rho_j),$$

  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. (61). 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

[21] 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.

[22] 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.

the sum of values of the outgoing coins, i.e., that

$$\mathbb{x} = (\mathsf{vcm}_1, \mathsf{vcm}_2, \mathsf{vcm}_B, \mathsf{vcm}_C),$$

and

$$\mathbb{w} = (\mathsf{val}'_1, z_1, \mathsf{val}'_2, z_2, v'_B, \rho_B, v'_C, \rho_C)$$

satisfy the relation $\mathcal{R}_{\mathsf{sum}}$ as defined in Eq. 62. (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 $\mathsf{ctxt}_j \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}'_j, (v'_j, \rho_j + t_j))$ 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.

G.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

$$\begin{aligned}&((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),\\ &(\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}}, \qquad (68)\\ &\Pi_{\mathsf{Sum}}.\end{aligned}$$

The simulator passes (68) 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[23] 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

[23] 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.

sk associated with $\mathsf{C}_{\mathsf{pid}'_j}$. If decryption succeeds, parse the plaintext as (v, r) and check that $\mathsf{ccm}_j = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}'_j, \mathsf{sn}_j, v, 0); r)$ 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 (68) as in the real protocol and updates the ledger accordingly.

H 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 (54) and (55) are indistinguishable.

Fix a sequence $\{z_\lambda\}_{\lambda\in\mathbb{N}}$ of inputs for the environment. Let $\Delta(\lambda)$ denote the statistical distance between (54) and (55), 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.[24] 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.[25]

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

[24] 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.

[25] 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.

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 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 H.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 H.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 H.1), the event E_h in which the $(\ell+1)$th command is a payment of an honest party (Section H.2), and the event E_c in which the the $(\ell+1)$th command is a payment of a corrupted party (Section H.3).

H.1 Mint command

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

Observation H.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 H.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 $(\mathtt{mint}, v, \mathsf{pid})$. Our goal is to upper-bound Δ_{ℓ,E_m} by $O((\ell+1)\mu)$. By Observation H.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 $\mathtt{paid}$. 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

$$\begin{aligned} \mathsf{sn} &= H(\mathsf{T}+1, z), \\ \mathsf{ccm} &= \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_0, \mathsf{sn}, v_0, 0); r), \\ \mathsf{ctxt} &\leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}_0, (v_0, r, \mathsf{sn})), \end{aligned} \tag{69}$$

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

$$\begin{aligned} \mathsf{sn} &= H(\mathsf{T}'+1, z), \\ \mathsf{ccm} &= \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, \mathsf{sn}, v, 0); r), \\ \mathsf{ctxt} &\leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}, (v, r, \mathsf{sn})), \end{aligned} \tag{70}$$

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 (69) is indistinguishable from (70) 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 (69) but $\mathsf{ctxt} \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}, (v, r', \mathsf{sn}))$ 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 (69). 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 $\delta \le \mu_{\mathrm{IBE}}$.

Next, consider the sub-hybrid $\mathcal{H}'_{\ell+1}$ which is identical to (70) except that $\mathsf{ctxt} \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}, (v, r', \mathsf{sn}))$ 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

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

and

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

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

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

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

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

Proof of Claim H.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 H.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 H.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 H.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

$$\begin{aligned}&((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'),\\ &(\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}},\ \Pi_{\mathsf{Sum}}\end{aligned} \tag{71}$$

where in the real protocol it is distributed as in (63), and in the ideal protocol it is distributed as in (66). 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.,

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

where pid is the Sender's public identifier and r_i is a fresh randomizer.[26] 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.

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.,

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

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

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

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 $\mathcal{H}'_3$.

[26] 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 $\mathcal{H}'_5$ 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.,

$$\begin{aligned}\mathsf{icm}_j = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_j, 0, 0, 0); t_j) \quad \text{and} \\ \mathsf{vcm}_j = \mathsf{CM.Com}(\mathsf{ck}, (0, 0, v_j, 0); \rho_j), \quad (72)\end{aligned}$$

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.,

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

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 $\mathsf{ctxt}_j \leftarrow\$ \mathsf{IBE.Enc}(\mathsf{pid}'_j, (v'_j, \rho_j + t_j))$ 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 H.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 (63) and (66) remain indistinguishable even given the signature's key, the entire view of the adversary remains indistinguishable. This completes the proof of Lemma H.1 for the case where the $(\ell+1)$th command is an honest payment command. □

H.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

$$\begin{aligned}&((\mathsf{ccm}_i, \sigma_i, \mathsf{vcm}_i, \mathsf{nullif}_i, \Pi_{\mathsf{Split}_i})_{i\in\{1,2\}}, \mathsf{rcm}', \mathsf{rs}'), \\ &(\mathsf{icm}_j, \mathsf{vcm}_j, \Pi_{\mathsf{icm}\,j}, \Pi_{\mathsf{Range}_j}, \mathsf{ctxt}_j)_{j\in\{B,C\}},\ \Pi_{\mathsf{Sum}}. \quad (73)\end{aligned}$$

We show that Δ_{ℓ,E_c} is upper-bounded by $O(\ell\mu)$. As in Section H.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 H.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 (73), 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 H.1. For every iteration $k \leq \ell+1$ 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 $\mathsf{ccm}_{k,B} = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}_{k,B}, \mathsf{sn}_{k,B}, v_{k,B}, 0); r_{k,B})$. Observe that these values are well defined since we explicitly compute them during the experiment when we emulate the minter.[27]

[27] 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.

We prove the following key lemma in Section H.4.

Lemma H.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 H.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 H.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

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

$$\text{where } \mathsf{sn} = H(\mathsf{nullif}_1, \mathsf{nullif}_2, j).$$

By the construction of ccm_B, it also holds that

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

We conclude that

$$(\mathsf{pid}, \mathsf{sn}, v) = (\mathsf{pid}_{\ell+1,B}, \mathsf{sn}_{\ell+1,B}, v_{\ell+1,B}), \tag{74}$$

unless we can break the binding property of the commitment, which can happen with probability at most $\mu_{\mathrm{BCOM}} \leq \mu$. Conditioned on (74), 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 (74) 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 H.1 for the case where the $(\ell+1)$th command is a corrupted payment command. □

H.4 Proof of Lemma H.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 $(\ell+1)\mu_{\mathrm{KE}} \leq O((\ell+1)\mu)$

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 setup algorithm and the signature key-generation algorithm (as defined in Definition D.2), 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 $\mathsf{CM.Com}(\mathsf{ck}, m; r)$ 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 $\mathsf{rcm} = \mathsf{CM.Com}(\mathsf{ck}, (\mathsf{pid}, 0, 0, \mathsf{s}); a)$, queries the oracle with $m = (\mathsf{pid}, 0, 0, \mathsf{s})$ and a, and receives back the signature $\mathsf{rs} \leftarrow\$\ \mathsf{RS.Sign}(\mathsf{rsk}, \mathsf{rcm})$. 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 $(m^*, a'_{k,i}, \leftarrow\$'_k)$ 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 $\leftarrow\$'_k$ is a valid rsk-signature over $\mathsf{CM.Com}(\mathsf{ck}, m^*; a'_{k,i})$. 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

$$[m_1 = (\mathsf{pid}_{k,1}, 0, 0, \mathsf{s}_{k,1}), \quad a'_{k,1})] \neq [m_2 = (\mathsf{pid}_{k,2}, 0, 0, \mathsf{s}_{k,2}), \quad a'_{k,2}], \quad (75)$$

for which $\mathsf{CM.Com}(\mathsf{ck}, m_1; a'_{k,1}) = \mathsf{CM.Com}(\mathsf{ck}, m_2; a'_{k,2})$. 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 $\mu_{\text{SIG}} + \mu_{\text{BCOM}} \leq O(\mu)$, 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 D.2), 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 (rsk, 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 $\mathsf{CM.Com}(\mathsf{ck}, m; r)$ 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 H.1. If these entries satisfy condition (2), then the adversary $\mathcal{B}$ generates the Bank's response where the signature on

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

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 $\mu_{\text{SIG}} \leq \mu$.

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

$$q_i = k'(k, i) \qquad g_i = j(k, i) \quad (76)$$

and that

$$\mathsf{pid}_i = \mathsf{pid}_{k,i}. \quad (77)$$

If (76) 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 (77) conditioned on (76). Suppose that (77) 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.[28]

Item 4. If the kth command is an honest payment, the statement holds trivially (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 $\mu_{\text{BCOM}} \leq \mu$.

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 H.2 and for $k < \ell+1$ this is established as part of the induction hypothesis, i.e., Lemma H.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

$$\mathsf{coins}[\mathsf{T}_{k,1}] = (\mathsf{val}_{k,1}, \mathsf{pid}_{k,1}). \tag{78}$$

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 H.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. 78 holds. Next, assume towards a contradiction that Eq. 78 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. □

[28] 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.

I 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, $\mathsf{CM.Com}$ consists of

two instances of the Pedersen (vector) commitment algorithm, using each of the sub-keys. in (See Figure 8.)

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) [97, Assumption 2] that asserts that, given

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

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

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

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 $\varepsilon/2$.

J Security of PS-based rerandomizable signature-commitment scheme

The PS-based rerandomizable signature-commitment scheme is given in Figure 9.

Proof of Lemma D.4 We prove Lemma D.4. We begin by recalling Assumption 1 of [34] (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 [34] 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 $x, y \leftarrow\$ \mathbb{Z}_p$ 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 $u \leftarrow\$ \mathbb{Z}_p$. 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.

We prove Lemma D.4 by showing that the scheme in Figure 9 is a rerandomizable signature over the commitment scheme in Figure 8, 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

$\underline{\mathsf{CM.Setup}(1^\lambda) \to \mathsf{ck}}$
$(g, \tilde{g}) \leftarrow\$ \mathbb{G}_1 \times \mathbb{G}_2$
$(k_1, k_2) \leftarrow\$ \mathbb{Z}_p^2$
$g_1 \leftarrow g^{k_1} \quad g_2 \leftarrow g^{k_2}$
$\tilde{g}_1 \leftarrow \tilde{g}^{k_1} \quad \tilde{g}_2 \leftarrow \tilde{g}^{k_2}$
return $((g_1, g_2, g), (\tilde{g}_1, \tilde{g}_2, \tilde{g}))$

$\underline{\mathsf{CM.Com}(\mathsf{ck}, (m_1, m_2); r) \to \mathsf{cm}}$
Parse ck as $((g_1, g_2, g), (\tilde{g}_1, \tilde{g}_2, \tilde{g}))$
$\mathsf{cm}_{\mathbb{G}_1} \leftarrow g_1^{m_1} g_2^{m_2} g^r$
$\mathsf{cm}_{\mathbb{G}_2} \leftarrow \tilde{g}_1^{m_1} \tilde{g}_2^{m_2} \tilde{g}^r$
return $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2})$

$\underline{\mathsf{CM.Rerand}(\mathsf{ck}, \mathsf{cm}; r) \to \mathsf{cm}'}$
Parse ck as $(\cdot, \cdot, g), (\cdot, \cdot, \tilde{g})$
Parse cm as $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2})$
return $(\mathsf{cm}_{\mathbb{G}_1} \cdot g^r, \mathsf{cm}_{\mathbb{G}_2} \cdot \tilde{g}^r)$

Figure 8: The commitment scheme.

$\underline{\mathsf{RS.KeyGen}(\mathsf{pp}, \mathsf{ck}) \to (\mathsf{sk}, \mathsf{pk})}$
Parse ck as $((g_1, g_2, g), (\tilde{g}_1, \tilde{g}_2, \tilde{g}))$
$x \leftarrow\$ \mathbb{Z}_p$
$X \leftarrow g^x, \tilde{X} = \tilde{g}^x$
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 $(\mathsf{cm}_{\mathbb{G}_1}, \mathsf{cm}_{\mathbb{G}_2})$
$u \leftarrow\$ \mathbb{Z}_p$
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 9: The rerandomizable signatures over the commitment scheme in Figure 8.

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 $(\alpha_i, \beta_i) \leftarrow\$ \mathbb{Z}_p^2$, 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 $x \leftarrow\$ \mathbb{Z}_p$, 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 $u \leftarrow\$ \mathbb{Z}_p$, 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 has not been 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

  $$\begin{aligned}(U, B/U^{\beta_1 m_1^* + \beta_2 m_2^* + r^*}) &= (g^u, g^{b - u(\beta_1 m_1^* + \beta_2 m_2^* + r^*)}) \\ &= (g^u, X^u Y^{m^* u}).\end{aligned}$$

  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

  $$(g_1^{m_1} g_2^{m_2} g^r) = (g_1^{m_1^*} g_2^{m_2^*} g^r)$$

  and

  $$(\tilde{g}_1^{m_1} \tilde{g}_2^{m_2} \tilde{g}^r) = (\tilde{g}_1^{m_1^*} \tilde{g}_2^{m_2^*} \tilde{g}^r),$$

  in contradiction to the binding property of the commitment.

Lemma D.4 follows. □

K Zero-Knowledge Proofs

In this section, we describe some concrete instantiations 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 D.5. For compatibility with previous versions, we have that the commitment basis is denoted by (g_1, g_2, g_3, g_6, g).

K.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

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

for which the following conditions hold

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

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.

K.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:

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

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:

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

K.1.2 Verifier

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

$$e(\mathsf{nullif}_i, \mathsf{vk}_i) = e(h, \tilde{h}) \cdot y_i \tag{79}$$
$$\mathsf{ccm}_i^c \cdot X_1 = g_1^{\alpha_1} g_2^{\alpha_2} g_3^{\alpha_3} g^{\alpha_4} \tag{80}$$
$$\mathsf{vcm}_i^c \cdot X_2 = g_3^{\alpha_3} g^{\alpha_5} \tag{81}$$
$$\mathsf{rcm}'^c \cdot X_3 = g_1^{\alpha_1} g_6^{\alpha_7} g^{\alpha_6} \tag{82}$$
$$\mathsf{vk}_i^c \cdot X_4 = \tilde{h}^{\alpha_7} \tilde{h}^{\alpha_2} \tilde{w}^{\alpha_8} \tag{83}$$
$$y_i^c \cdot X_5 = q_i^{\alpha_8} \tag{84}$$

K.1.3 Completeness

For a honest prover, all statements in Eq. (79) hold:

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

And,

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

K.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 $\mathrm{w} = (\mathsf{s}_{\mathsf{Sender}}, \mathsf{pid}_{\mathsf{Sender}}, \mathsf{sn}_i, \mathsf{val}_i, r_i, z_i, a')$ 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:

$$\mathsf{ccm}_i^{c-c'} = g_1^{\alpha_1-\alpha_1'} g_2^{\alpha_2-\alpha_2'} g_3^{\alpha_3-\alpha_3'} g^{\alpha_4-\alpha_4'} \quad (85)$$

$$\mathsf{vcm}_i^{c-c'} = g_3^{\alpha_3-\alpha_3'} g_5^{\alpha_5-\alpha_5'} \quad (86)$$

$$\mathsf{rcm}'^{c-c'} = g_1^{\alpha_1-\alpha_1'} g_6^{\alpha_7-\alpha_7'} g^{\alpha_6-\alpha_6'} \quad (87)$$

$$y_1^{c-c'} = q_1^{\alpha_8-\alpha_8'} \quad (88)$$

$$\mathsf{vk}_i^{c-c'} = \tilde{h}^{\alpha_7-\alpha_7'} \tilde{h}^{\alpha_2-\alpha_2'} \tilde{w}^{\alpha_8-\alpha_8'} \quad (89)$$

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,

$$\begin{aligned}
(85,87) &\implies \mathsf{exp}(g_1, \mathsf{ccm}_i) = \mathsf{exp}(g_1, \mathsf{rcm}') \\
&= (\alpha_1 - \alpha_1')/(e - e') = \mathsf{pid} \\
(85,86) &\implies \mathsf{exp}(g_3, \mathsf{ccm}_i) = \mathsf{exp}(g_3, \mathsf{vcm}_i) \\
&= (\alpha_3 - \alpha_3')/(e - e') = \mathsf{val}_i \\
(85,87,89) &\implies \mathsf{exp}(g_2, \mathsf{ccm}_i) + \mathsf{exp}(g_6, \mathsf{rcm}') \\
&= \mathsf{exp}(\tilde{h}, \mathsf{vk}_i) \\
&= (\alpha_2 + \alpha_7 - \alpha_2' - \alpha_7')/(e - e') \\
&= \mathsf{s}_{\mathsf{Sender}} + \mathsf{sn}_i \\
(89,88) &\implies \mathsf{exp}(q_i, y_i) = \mathsf{exp}(\tilde{w}, \mathsf{vk}_i) = t
\end{aligned}$$

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.

K.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 $(\hat{\mathsf{init}}, \hat{\mathsf{challenge}}, \hat{\mathsf{response}})$:

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

■

K.2 KZG-Pedersen agreement

The prover wants to prove the following statement:

$$\mathbb{x} = (c, c', \mathsf{cm}, Y) \quad (90)$$

$$\mathbb{w} = [r, z, a] \quad (91)$$

$$\mathcal{R}(\mathbb{x}, \mathbb{w}) = \begin{pmatrix} c' = & c^r \wedge & \\ \mathsf{cm} = & g_3^a g^z \wedge & \\ Y = & q^{ar} & //q = e(G, \tilde{G}) \end{pmatrix} \quad (92)$$

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.

K.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 $\mathsf{response}$:

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

K.2.2 Verifier

$$c'^e \cdot X_1 = c^{\alpha_1} \quad (93)$$
$$\mathsf{cm}^e \cdot X_2 = g_3^{\alpha_2} g^{\alpha_3} \quad (94)$$
$$Y^e \cdot X_3 = q^{\alpha_4} \quad (95)$$
$$\mathsf{cm}'^e \cdot X_4 = g_3^{\alpha_4} g^{\alpha_5} \quad (96)$$
$$\mathsf{cm}'^e \cdot X_5 = \mathsf{cm}^{\alpha_1} \quad (97)$$

K.2.3 Completeness

$$\begin{aligned}(93): c'^e \cdot X_1 &= c^{e\cdot r}\cdot c^{x_1} = c^{x_1+e\cdot r} = c^{\alpha_1}\\ (94): \mathsf{cm}^e \cdot X_2 &= (g_3^a g^z)^e \cdot g_3^{x_2} g^{x_3} = g_3^{x_2+e\cdot a} g^{x_3+e\cdot z} = g_3^{\alpha_2} g^{\alpha_3}\\ (95): Y^e \cdot X_3 &= q^{e\cdot ar} q^{x_4} = q^{\alpha_4}\\ (96): \mathsf{cm}'^e \cdot X_4 &= g_3^{e\cdot ar} g^{e\cdot zr} \cdot g_3^{x_4} g^{x_5} = g_3^{x_4+e\cdot ar} g^{x_5+e\cdot zr} = g_3^{\alpha_4} g^{\alpha_5}\\ (97): \mathsf{cm}'^e \cdot X_5 &= g_3^{e\cdot ar} g^{e\cdot zr} \cdot \mathsf{cm}^{x_1}\\ &= g_3^{e\cdot ar} g^{e\cdot zr} \cdot g_3^{ax_1} g^{zx_1}\\ &= (g_3^a)^{x_1+er} (g^z)^{x_1+er}\\ &= (g_3^a g^z)^{x_1+er} = \mathsf{cm}^{\alpha_1}\end{aligned}$$

K.2.4 Soundness

Consider equations (98)-(102), which are implied by equations (93)-(97) above.

- From equation (98) we can extract $\log_c(c') = (\alpha_1 - \alpha_1')/(e-e')$.
- From equation (99) 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 (100) 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 (101) 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 (102) 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.

$$(93) \implies c'^{e-e'} = c^{\alpha_1-\alpha_1'} \quad (98)$$
$$(97) \implies \mathsf{cm}'^{e-e'} = \mathsf{cm}^{\alpha_1-\alpha_1'} \quad (99)$$
$$(94) \implies \mathsf{cm}^{e-e'} = g_3^{\alpha_2-\alpha_2'} g^{\alpha_3-\alpha_3'} \quad (100)$$
$$(95) \implies Y^{e-e'} = q^{\alpha_4-\alpha_4'} \quad (101)$$
$$(96) \implies \mathsf{cm}'^{e-e'} = g_3^{\alpha_4-\alpha_4'} g^{\alpha_5-\alpha_5'} \quad (102)$$

K.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:

$$(93): c'^e \cdot X_1 = c'^e \cdot c^{\alpha_1}\cdot c'^{-e} = c^{\alpha_1}$$
$$(94): \mathsf{cm}^e \cdot X_2 = \mathsf{cm}^e \cdot g_3^{\alpha_2} g^{\alpha_3}\cdot \mathsf{cm}^{-e} = g_3^{\alpha_2} g^{\alpha_3}$$
$$(95): Y^e \cdot X_3 = Y^e \cdot q^{\alpha_4}\cdot Y^{-e} = q^{\alpha_4}$$
$$(96): \mathsf{cm}'^e \cdot X_4 = \mathsf{cm}'^e \cdot g_3^{\alpha_4} g^{\alpha_5}\cdot \mathsf{cm}'^{-e} = g_3^{\alpha_4} g^{\alpha_5}$$
$$(96): \mathsf{cm}'^e \cdot X_5 = \mathsf{cm}'^e \cdot \mathsf{cm}^{\alpha_1}\cdot \mathsf{cm}'^{-e} = \mathsf{cm}^{\alpha_1}$$