Zero-Knowledge for Homomorphic Key-Value Commitments with Applications to Privacy-Preserving Ledgers

Matteo Campanelli[1], Felix Engelmann[2], and Claudio Orlandi[3]

[1] Protocol Labs, matteo@protocol.ai
[2] ITU, Denmark, fe-research@nlogn.org
[3] Aarhus University, Denmark, orlandi@cs.au.dk

Abstract. Commitments to key-value maps (or, authenticated dictionaries) are an important building block in cryptographic applications, including cryptocurrencies and distributed file systems.
In this work we study short commitments to key-value maps with two additional properties: double-hiding (both keys and values should be hidden) and homomorphism (we should be able to combine two commitments to obtain one that is the "sum" of their key-value openings). Furthermore, we require these commitments to be short and to support efficient transparent zero-knowledge arguments (i.e., without a trusted setup).
As our main contribution, we show how to construct commitments with the properties above as well as efficient zero-knowledge arguments over them. We additionally discuss a range of practical optimizations that can be carried out depending on the application domain.
Finally, we formally describe a specific application of commitments to key-value maps to scalable anonymous ledgers. We show how to extend QuisQuis (Fauzi et al. ASIACRYPT 2019). This results in an efficient, confidential multi-type system with a state whose size is independent of the number of transactions.

Keywords: Zero-knowledge · Key-Value map · Commitments.

1 Introduction

In this work we propose constructions for efficient commitments to key-value maps (with specific features) and for efficient zero-knowledge arguments that can prove properties on committed key-value maps.
Key-Value Maps. We can loosely consider a key-value map as the equivalent of a dictionary in some programming languages (e.g., Python): a way to map arbitrary keys—e.g., strings—to values—e.g., scalars. For example, the balance of a user in a wallet application could be represented by a key-value map as $\mathsf{kv} = \{(\mathsf{USD}, 100), (\mathsf{BTC}, 10)\}$, where each of the different asset types (the keys) are associated to an amount (the values). In this paper we will generally assume that values are in an algebraic group endowed with an addition operation $+$.
Our Focus: Short, Homomorphic, Doubly-Hiding Commitments. A commitment to a key-value map is roughly similar to an ordinary commitment: it cannot be opened to two different key-value maps (binding) and it should not leak anything about neither the keys nor the values in it. In the case of key-value maps, however, we are interested in some additional functional and efficiency-related requirements:

- *Large key universe:* our commitments should support a large universe of keys, potentially superpolynomial in the security parameter[4]. This implies that the algorithms of the commitment scheme should have a runtime independent of (or logarithmic in) the size of the key universe.
- *Short commitments:* our commitments should have size independent not only of the size of the key universe, but also of the *density* of the key-value map. The density is the number of elements whose value is not zero (e.g., the density of kv in the example above was 2).
- *Homomorphic commitments:* we require our commitments to support an homomorphic operation $\circ$. For example assume each commitment encodes a wallet and that we have two wallets c, c' with $c = \mathsf{Com}(\{(\mathsf{USD}, 100), (\mathsf{BTC}, 10)\})$ and $c' = \mathsf{Com}(\{(\mathsf{USD}, 20), (\mathsf{ETH}, 1)\})$. Then we can compute the commitment $c^* \leftarrow c \circ c' = \mathsf{Com}(\{(\mathsf{USD}, 120), (\mathsf{BTC}, 10), (\mathsf{ETH}, 1)\})$ without knowing the opening of any of the commitments. Requiring homomorphism rules out Merkle Trees as a solution. Homomorphic properties of commitments to "structured objects" have wide applications in cryptography (see, e.g., [KZG10] for homomorphic polynomial commitments). The homomorphic property is a natural one and allows many useful applications: as an example we describe applications to privacy-preserving cryptocurrencies in Section 7 and an additional class of application scenarios in Section 1.1.

[4] This is a way to describe our setting asymptotically. We stress, however, that is not necessary: an interesting setting for our constructions is just one where the universe of keys is *concretely* large.

– *Efficient and transparent*[5] *zero-knowledge proofs*: we should be able to prove (and verify) efficiently arbitrary properties over commitments of key-value maps. We are interested in zero-knowledge proofs—which allow to prove properties over a secret value without leaking it—and where both keys and values are part of the secret. For example, one can prove that two committed key-value maps hold the same value for some (hidden) key $\tilde{k}$. More formally, given as public input commitments c, c' and a public function f, one can prove knowledge of a key $\tilde{k}$ such that c, c' are commitments to key-value maps $\mathsf{kv}, \mathsf{kv}'$ respectively and $\mathsf{kv}[\tilde{k}] = f(\mathsf{kv}'[\tilde{k}])$.

While different subsets of these properties have been studied in literature, our contribution is to investigate constructions that require them *all*. Our goal is to provide concretely efficient tools useful in different application domains.

Key-Hiding Properties. Here we clarify what we mean by key-hiding properties and discuss how existing solutions fail to solve our problem. We have three *key sets* of interest: the set of all the keys in the universe (which we will assume to be $\{0,1\}^*$ or a field $\mathbb{F}$ from now on), the set K_{active} of *active keys*, defined as all the keys that are being used in the system, and the set K_{com} of *committed keys*, defined as the non-zero keys in any *given* commitment. As our commitment scheme always supports an exponentially large key space, the notion of active and committed keys is only relevant for commitments which require a NIZK about their opening. For example, in a wallet setting, K_{active} consists of all the keys (asset types) encoded in *some* wallet, while K_{com} would consist of those encoded in a *specific given* wallet. Depending on whether we want to hide the active or the committed keys or both we get four different settings, which we discuss below (see also Fig. 1).

Public Active keys. In the case where both the active keys and the committed keys are public, Pedersen commitments are already a solution to our problem. The system parameters will contain group elements $h, g_1, \ldots, g_n$ where there is a known association between k_i and g_i for all active keys k_i. E.g. A public coin setup process generates $g_i = \mathcal{H}(k_i)$ with a hash function $\mathcal{H}$. Thereby the association is known by all participants. We commit by computing $c = h^r \prod_i g_i^{v_i}$, and proving properties of values is trivial to do using existing sigma protocols since the verifier is allowed to learn the keys. In the case in which the active keys are public but the committed keys are private, Pedersen commitment can still be used but the (proving) complexity of the ZK proof would be linear in the number of active keys[6]. One of our contributions is to show how to bring this down to the size of the committed set.

Private Active keys. It does not make sense to consider the case where the set of active keys are private but the committed keys are public. The most interesting case is the one in which both of these sets are private. In this setting, it would be possible to commit using a non-homomorphic version of Pedersen commitment. We thus have $2n+1$ generators $(h, g_1, f_1, \ldots, g_n, f_n)$ and we commit computing $c = h^r \Pi g_i^{v_i} f_i^{k_i}$. Now it is possible to efficiently prove statements but the commitment is not homomorphic (and therefore not applicable in our settings of interest). Our main contribution is to provide a better solution for this case.

1.1 Applications of Our Work

Application: Multi-Type QuisQuis. The privacy-preserving transaction system QuisQuis [FMMO19] crucially relies commitments endowed with an homomorphic property. It builds upon accounts to which tokens are deposited in a transaction without interaction of the receiver. A crucial performance consideration is the storage needed for a client to participate in the system. This corresponds to the local state necessary to validate arising updates (i.e. transactions). Compared to other privacy-preserving transaction systems like Zcash [BCG+14] or Monero [NM+16], the design of QuisQuis achieves a state size linear in the number of participants instead of monotonically growing over time (i.e. requireing clients to store the full history). We extend this system through a notion of currency types such that different currencies share a common anonymity set. This allows for a dynamic creation of confidential tokens by any participant without setting up a full separate system. For this application, we also present a secret key based key-value map commitment[7]. In combination with efficient NIZKs, to show that transactions conserve all value, we achieve small transaction sizes. We formally describe this application in Section 7.

[5] In a *transparent* argument system the setup does not need to be produced by a trusted party. This property is interesting in the case of *non-interactive* argument systems, which are the focus of this work.

[6] This is true for the aforementioned approach with sigma-protocols as well as for other straightforward applications of NIZKs.

[7] I.e., a commitment which can be opened using a secret key in place of the randomness.

Application: Publicly Verifiable Evolving Database. Consider a database (representable as key-value store) which receives numerous updates and where we want the content of the database to remain private but we also are interested in the database publicly "evolving through time".

As an example of the above, consider a register of tax-related information where users are identified by their SSN. The set of valid identities grows dynamically, which results in a high overhead if the public parameter changes every time. Users provide their SSN to their employer who uses it to report the salary. At the end of the month, each employer creates a hiding key-value map commitment with one key per employee and their earned amount as value, e.g. $\delta C_{\text{Corp X,May}} = \mathsf{Com}(\{\text{SSN}_{\text{Alice}} : 3142, \text{SSN}_{\text{Bob}} : 2718\})$. The company may either prove that for the employee's identity the correct amount was committed (without revealing the identities of co-workers), or reveal the full opening to their employees. Every company publishes these commitments to a persistent log. At the end of the year, the tax authority homomorphically adds all published commitments and can then generate proofs on a single commitment instead of all commitments from all companies. The required value opening is provided by the tax payers and the randomness by the companies. Employees with multiple sources of income get the amounts homomorphically added. Different categories of income may be separated by namespaces in the key.

1.2 Technical Overview

Our Construction of Key-Value Map Commitments. In order to commit to a key-value map $\{v_k\}_{k\in K}$ we assume a group $\mathbb{G}$ where the discrete logarithm is hard and a hash function $\mathcal{H}$ modeled as a random oracle mapping keys to group elements. We then compute a commitment as $c = \prod_{k\in K} \mathcal{H}(k)^{v_k} h^r$ where h is a random generator of the group and r is a random scalar. This can be seen as a (vector) Pedersen commitment with random key-dependent generators and it has short homomorphic commitments. In the next paragraphs, we show how we can construct efficient zero-knowledge proofs for circuits over such commitments.

Modular Transparent Zero-Knowledge Arguments for Committed Key-Value Maps. Fix a (large) field $\mathbb{F}$ and consider a circuit C over key-value maps (we assume that $\mathbb{F}$ is also both the key and the value space of the key-value map). We assume the syntax of C to be of the type $C(\mathsf{kv}_1, \ldots, \mathsf{kv}_\ell, \omega)$, the kv_i-s as private key-value maps and ω as an additional private witness (ω is a vector of field elements). Given such a circuit we are interested in proving an augmented circuit that takes as public input commitments to the ℓ key-value maps and proves their opening in addition to the relation from circuit C. More specifically, we propose an argument system for:

$$\begin{aligned} &C^*(c_1, \ldots, c_\ell; (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega) \\ &:= C(\mathsf{kv}_1, \ldots, \mathsf{kv}_\ell, \omega) \wedge \bigwedge_{i\in[\ell]} c_i = \mathsf{Com}(\mathsf{kv}_i, \rho_i) \end{aligned} \tag{1}$$

where the part after the semicolon is considered the private witness. A more concrete intuition on the circuit above is: given committed key–value pairs we can prove properties of their values, their keys and any relation between these and other private values (contained in ω).

Our Template for Zero-Knowledge Arguments on Committed Key-Value Maps. We now describe how to prove properties on committed key-value maps. The following refers to the setting with private-active-keys/private-committed-keys (see lower-right quadrant in Figure 1). We denote the construction for the doubly-private case by Z🥝-dp (first part alternatively spelled "ZKeyWee", as a pun on "ZK for KV", and pronounced "*zee-kee-wee*"; "dp" stands for doubly-private).

Our construction follows a basic blueprint, which we now exemplify through a concrete case. Consider a committed key-value map $\{v_k\}_{k\in K}$ and the problem of proving in zero-knowledge that all its values v_k are in some range $\{0, \ldots 2^\mu - 1\}$. We proceed in two steps: we first let the prover send the verifier what we call *key-tags*, these are masked versions of the non-zero keys in the committed key-value map. The prover will also show that they are valid maskings of some set of keys. By providing key-tags, we can then break the rest of the relation in two parts: *a)* showing knowledge of values (and randomness) that combined with the key-tags produce the commitment c (part of the public input); *b)* showing that these values are in range. We now elaborate on each of these steps.

Given a key-value map $\{v_k\}_{k\in K}$ with density n (the number of non-zero keys[8]) we provide n key-tags by sending $b_k = \mathcal{H}(k)h^{r_k}$ for a random r_k. The prover should also prove that each of them is of the prescribed

[8] Here we consider the case where leaking the density of the key-value map is not a problem. We will also adapt our construction where this leakage does not occur if an upper bound on this density is known.

	Public K_{active}	Private K_{active}
Public K_{com}	$c = h^r \prod_i g_i^{v_i}$ & Σ-protocols	*(Uninteresting case)*
Private K_{com}	c as in Fig. 2 & Z🥝-set (Sec. 1.2+ Appx. A)	c as in Fig. 2 & Z🥝-dp (Sec. 4)

Fig. 1: How to construct commitments to key-value maps & NIZKs over them within our settings and with our requirements of interests. (K_{active}: "*active*" keys set, i.e., all the keys committed somewhere; K_{com} *committed* keys set, i.e. those that open the commitments we are using in a proof *right now*). The related constructions are specified in the second lines with our two contributions Z🥝-set and Z🥝-dp.

form. We stress that, in order to do this, *we use the heuristic technique of proving a random oracle in zero-knowledge by assuming there exists a circuit for it* (a non-standard but common technique; see, e.g., [Val08]). Next, the prover would show knowledge of values $v_1, \ldots, v_n$ and an appropriate r' such that $c = b_{k_1}^{v_1} \cdots b_{k_n}^{v_n} h^{r'}$ and $v_i \in \{0, \ldots 2^\mu - 1\}$ for all $i \in [n]$. The latter relation—comprising reconstructing the commitment from the key-tags and the range proof—can for example be performed through a system like the generalized Bulletproofs in LMR [LMR19] or compressed Σ-protocols [ACR20]. These provide interfaces to prove bilinear circuits, of which we only use the non-bilinear gates, with logarithmic proof sizes. We stress that our focus is on *transparent* solutions, i.e., without a trusted setup; all our constructions can be instantiated in a fully transparent manner. We discuss an experimental evaluation in Section 6. We estimate our system can open $n = 100$ values and prove they are in range in approximately one minute. A very loose estimate for the size of the corresponding proof is < 6KB using [LMR19] in the Ristretto curve (see also Table 1 and Section 4.3).

We remark on two properties of the template of our construction above. *First*, we can easily reduce its amortized cost by splitting it into an offline stage (independent of the commitments on which we are carrying out proofs) and an online stage. We further discuss these improvements in Section 5. Second, we adapt and optimize our construction to the scenario with *public* active keys and private committed keys (lower-left quadrant in Figure 1). We describe this next.

A 2ND CONSTRUCTION WITH REGISTRATION OF ACTIVE KEYS. In some settings, although the whole universe of keys can be extremely large, the set of active keys K_{active} at any given time can have a manageable size and be publicly known. Consider for example applications (e.g. multi-asset transaction system) where there is an exponentially large set of potential asset types (keys), but only a manageable subset of them are present in the system (active) at any given time. Moreover, before becoming active in the system they plausibly need to be registered (for example, through a first "genesis" transaction for that specific asset type). In such settings we can leverage the partial knowledge on existing keys to improve efficiency. We do this by introducing an operation that preprocesses the parameters of the system (or CRS) specializing them for a specific set of active keys. Our proposed construction for this setting—denoted by Z🥝-set (for "registered *set*" of keys) and discussed in more detail in Appendix A—assumes the specialized CRS for the set K_{active} to contain an accumulator[9] to the set of (unmasked) key-tags corresponding to K_{active}, i.e., to the set $B_{\text{active}} = \{\mathcal{H}(k) : k \in K_{\text{active}}\}$. Thus, in the online stage, we can produce a proof on a commitment $c = \mathsf{Com}(\{v_k\}_{k \in K})$ by: 1) producing masked key-tags $B' = \{\mathcal{H}(k) h^{r_k} : k \in K\}$ produced with some fresh randomness r_k; 2) showing that each $b'_k \in B'$ is of the form $b \cdot h^{r_k}$ for some b in the accumulator; 3) showing knowledge of v_k-s such that $c = h^r \prod_k b_k'^{v_k}$.

The main advantage of the construction above, Z🥝-set, is that it does not require the hashing $\mathcal{H}(k)$ for the key-tags to be proven in zero-knowledge (by exploiting the fact that the keys are public and "pre-registered"). This can result in savings in verification time of one order of magnitude compared to Z🥝-dp (we elaborate in Appendix A.4); such savings also apply to the multi-type transaction systems Multi-type QuisQuis (Section 7) and can be extended to other transaction systems (our techniques in Z🥝-set are compatible with other frameworks besides QuisQuis and they could be straightforwardly applied, e.g, to obtain a multi-type version of 𝕍eksel [CHA21]).

MULTI-TYPE QUISQUIS. In contrast to the original QuisQuis where an account stored a scalar value representing an amount, we generalize accounts to store tokens of different types in a key-value map. Each key corresponds

[9] An accumulator is a cryptographic data structure that allows to commit to a set in a binding manner and to prove membership of an element efficiently. NB: we can compute accumulators *deterministically* from a set, i.e., without a trusted authority.

to a type, also known as currency, and the value specifies how many tokens of the specific type are held by the account. An account acct holds a balance kv, represented as a key-value map. To transfer tokens from one account to another, a transaction includes both accounts as active inputs, denoted by the set $\mathcal{P}$. The transaction subtracts tokens from one account with a secret index s and adds them to the other one. The output of the transaction are two updated accounts belonging to the same users as the inputs; the input accounts are discarded. To achieve anonymity, the input consists of $\mathcal{P}$ together with a potentially large anonymity set of accounts $\mathcal{A}$ which keep their balance unchanged but provide set anonymity for the active accounts. The sender uses Z🥝-set with a circuit to prove that they have enough funds (knowledge of secret key and positive balance) and that the updates are consistent.

1.3 Related Work

Authenticated Data Structures Besides the aforementioned straw-man schemes based on Pedersen, a common approach to succinct key-value commitments uses Merkle trees. They are not homomorphic and opening them in zero-knowledge requires proving a number of hashes logarithmic in the number of committed elements, which can be expensive.

A related primitive is that of vector commitments [CF13]. A limitation of using vector commitments as key-value map commitments in general is that values need to be stored at positions that have already been agreed upon. That is, since we need to know for each key k what is the index i_k it refers to in the vector. This type of common agreement may be achieved in the setting in the bottom left quadrant in Fig. 1 (public active keys) but not in the bottom right one (private active keys). Also, while some constructions of vector commitments are homomorphic (e.g., [CF13]) they lose this property when hiding is added to them (which is usually achieved by storing hiding commitments to the values of interest). Other constructions do not have this limitation ([LY10, BG18]) but, like vector commitments in general, only support public active keys. We finally remark that, vector commitments focus on a different notion of *succinctness* than the one that is the focus in this paper. Our focus is on a proof size that is sublinear in the size of the circuit we apply on the opening, but not necessarily sublinear in the number of committed elements.

There is a large body of work on succinct commitments to key-value maps, e.g.[10], [AR20, CFG+20, BBF19, TXN20]. [11] Differently from our work, constructions in literature are not homomorphic and do not directly support hiding of keys/values. We observe, however, that if one could do without homomorphism the latter problem could be mitigated for some of these constructions by applying masking of keys/values and zero-knowledge. This is true for example for some of the works based on groups of unknown-order [CFG+20, BBF19] where we can use techniques to compose algebraic accumulators proofs and succinct zero-knowledge proof systems described in [BCFK19].

The work in [AK20] formalizes encryption on distributed key-value maps with consistency properties; it is not concerned with homomorphism or efficient zero-knowledge. The work in [GPR+21] studies "oblivious key-value stores". Their setting is, however, different from ours: these data-structures hide the keys only if the values are random (not applicable in our setting) and are not homomorphic. Other works on efficient Zero-Knowledge and key-value maps include Spice [SAGL18]. The authors use data-structures that hide the key but that are not homomorphic. Their constructions use a trusted setup.

Confidential Transaction Systems and Multiple Token Types Here we describe works related to our application, a multi-type version of QuisQuis.

Works on confidential transaction systems include Zcash [BCG+14], Monero [NM+16], Omniring [LRR+19], and 𝕍eksel [CHA21]. We now compare these works against the QuisQuis framework, which we extend in this work. The most critical aspect is sender anonymity. Zcash obtains the largest anonymity set among these works (as large as the UTXO set), but it does not have plausible deniability[12] and requires a trusted setup. Monero does not have these limitations; it is unclear how the anonymity in Monero fares against that in QuisQuis (see Discussion in [FMMO19]). Omniring improves the transaction size from being linear in the size of the anonymity

[10] We refer the reader to [TXN20] for a survey of this rapidly growing field.

[11] The constructions in [CFG+20] and [BBF19] focus on vector commitments but they can be both be compiled into commitments to key-value maps: both the construction in [BBF19] and the first construction in [CFG+20] can be using compiled the techniques described in Section 5.4 of [BBF19]; the second construction of [CFG+20] is very similar to that in [AR20] and can be turned into a key-value commitments with similar tricks.

[12] Plausible deniability: no one can tell if a user meant to be involved in a transaction.

set (Monero) to a logarithmic size. Both Zcash and Monero style systems, however, have transaction outputs that can (essentially) never be removed from the UTXO set. The payment system in 𝕍eksel, like QuisQuis, does not have this drawback. Differently from QuisQuis, 𝕍eksel achieves $O(1)$ transaction sizes, but at the price of weaker anonymity guarantees. In all these systems amounts are confidential, yet they lack a notion of type/currency.

The work in [PBF+19] introduces confidential types by using homomorphic commitments whose construction is the "single key" version of ours. Their design has also been used in SwapCT [EMP+21] and integrated in MimbleWimble [YYD+19]. Another construction of confidential types is that of Cloaked Assets developed by Stellar, which separates types and values in two different data structures, similar to our non-homomorphic example. Therefore a transactor requires the openings of all inputs to create a conservation proof, providing no sender anonymity.

2 Notation and Preliminaries

Preliminaries on (Sparse) Key-Value Maps We assume a universe of keys $\mathcal{K}$ and a universe of values $\mathcal{V}$ such that in a key-value map, keys are a subset of $\mathcal{K}$ and values are any element in $\mathcal{V}$; they may be of size superpolynomial in a security parameter λ. We assume $\mathcal{V}$ to be an additive group endowed with some operation $+$. A key-value map is defined as a function $\mathsf{kv} : \mathcal{K} \to \mathcal{V}$. We call its *density* the number of elements that are mapped to a non-zero value in $\mathcal{V}$. Our focus is on *sparse* key-value maps whose density grows asymptotically with $\mathsf{poly}(\lambda)$ (and in practice may be concretely small). We can represent a sparse key-value map as a set of pairs $\{(k, v_k)\}_{k\in K}$ where $K \subseteq \mathcal{K}$: this maps each element $k \in K$ to v_k and any other element to $0 \in \mathcal{V}$. We often use the more succinct notation $\{v_k\}_{k\in K}$ for a key-value map $\{(k, v_k)\}_{k\in K}$ over the set K (we assume that the set K is implicitly part of the description of $\{v_k\}_{k\in K}$). Hence the empty set $\emptyset$ represents the key-value map with all elements in the universe initialized to zero; we denote the latter empty key-value map $\emptyset_{kv}$ to be explicit. We denote by $-\mathsf{kv}$ the key-value map associating the value $-\mathsf{kv}(k)$ to each key k; we define a sum of key-value maps as follows: $\{v_k\}_{k\in K} + \{v'_{k'}\}_{k'\in K'} := (k, v_k + v'_{k'})_{k\in K\cup K'}$. A partition of a key-value map $\{v_k\}_{k\in K}$ is a pair of key-value maps $(\{v'_{k'}\}_{k'\in K'}, \{v''_{k''}\}_{k''\in K''})$ such that (K', K'') is a partition of K.

Cryptographic Assumptions For convenience we use a different, but equivalent, formulation of the discrete logarithm assumption. Below $\mathcal{G}$ denotes a group generator.

Assumption 1 (Generalized DLOG [BBB+18]) $\forall$ *PPT* $\mathcal{A}, m \geq 2$:

$$\Pr\left[\begin{array}{c} \mathbb{G} \leftarrow \mathcal{G}(1^\lambda); (g_1, \ldots, g_m) \leftarrow\!\$\ \mathbb{G} \\ (a_1, \ldots, a_m) \leftarrow \mathcal{A}(\mathbb{G}, g_1, \ldots, g_m) \end{array} : \begin{array}{c} \exists j^* \in [m]\ a_{j^*} \neq 0\ \wedge \\ \prod_{j\in[m]} g_j^{a_j} = 1_\mathbb{G} \end{array}\right] \leq \mathsf{negl}(\lambda)$$

NIZKs Here we describe the basic notion of non-interactive zero-knowledge. In Section 4 we provide explicit syntax for the specific setting of NIZKs over committed key-value maps.

Definition 1. *A NIZK for a relation family* $\mathcal{R} = \{R_\lambda\}_{\lambda\in\mathbb{N}}$ *is a tuple of algorithms* $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerProof})$ *with the following syntax:*

- $\mathsf{NIZK.Setup}(1^\lambda) \to \mathsf{crs}$ *outputs a common-reference string* crs*; if the argument system is transparent this can consist of uniform random elements.*
- $\mathsf{NIZK.Prove}(\mathsf{crs}, \mathbb{x}, \mathbb{w}) \to \pi$ *takes as input a string* crs*, an input description* $\mathbb{x}$ *(in which we embed the whole public input), a witness* $\mathbb{w}$ *such that* $R_\lambda(\mathbb{x}, \mathbb{w})$*; it returns a proof* π*.*
- $\mathsf{NIZK.VerProof}(\mathsf{crs}, \mathbb{x}, \pi) \to b \in \{0, 1\}$ *takes as input a string* crs*, a public input* $\mathbb{x}$*, a proof* π*; it accepts or rejects the proof.*

Whenever the relation family is obviously defined, we talk about a "NIZK for a relation R". We require a NIZK to be *complete*, that is, for any $\lambda \in \mathbb{N}$ and $(\mathbb{x}, \mathbb{w}) \in R_\lambda$ it holds with overwhelming probability that $\mathsf{VerProof}(\mathsf{crs}, \mathbb{x}, \pi)$ where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathbb{x}, \mathbb{w})$. Other properties we require are: *knowledge-soundness* and *zero-knowledge*. Informally, the former states we can efficiently "extract" a valid witness from a proof that passes verification; the latter states that the proof leaks nothing about the witness (this is modeled through a simulator

that can output a valid proof for an input in the language without knowing the witness). Notationally, we separate public and private inputs in relations and proving algorithm through a semicolon.

Knowledge-Soundness. For all $\lambda \in \mathbb{N}$ and for all (non-uniform) efficient adversaries $\mathcal{A}$, auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, there exists a (non-uniform) efficient extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{matrix}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda); (\mathbb{x}, \pi) \leftarrow \mathcal{A}(z, \mathsf{crs}) \\ \mathbb{w} \leftarrow \mathcal{E}(z, \mathsf{crs})\end{matrix} : \begin{matrix} R_\lambda(\mathbb{x}, \mathbb{w}) \neq 1 \wedge \\ \mathsf{Vfy}(\mathsf{crs}, \mathbb{x}, \pi) = 1\end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

Zero-Knowledge. There exists a PPT simulator $\mathcal{S}$ such that for any $\lambda \in \mathbb{N}$, PPT $\mathcal{A}$, auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and it holds $p_0 = p_1$ where

$$p_b := \Pr\left[\begin{matrix}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda); (\mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(z, \mathsf{crs}) \\ \pi \leftarrow X_b(\mathsf{crs}, \mathbb{x}, \mathbb{w}) \text{ if } R_\lambda(\mathbb{x}, \mathbb{w}) \text{ o.w. } \perp\end{matrix} : \mathcal{A}(z, \mathsf{crs}, \pi) = 1\right]$$

$X_0(\mathsf{crs}, \mathbb{x}, \mathbb{w}) := \mathcal{S}(z, \mathsf{crs}, \mathbb{x})$ and $X_1(\mathsf{crs}, \mathbb{x}, \mathbb{w}) := \mathsf{Prove}(\mathsf{crs}, \mathbb{x}, \mathbb{w})$.

On efficiency of NIZKs. The efficiency (proving/verification runtimes and proof size) of a NIZK often depends on the size of the description of a relation in *constraints* (these roughly correspond to the multiplication gates of its circuit representation). We will refer to this notion later in the text. See also [BBB+18].

3 Key-Value Commitments

Here we define homomorphic commitments to key-value maps where both keys and values are hidden. To the best of our knowledge this definition is new, but it straightforwardly extends homomorphic commitments with key-value maps as message space. In the appendix (Appendix D.2)we also present an extended construction, which we use to build Multi-Type Quis-Quis.

Definition 2 (Commitment to Key-Value Maps (kvC)). *The following is a syntax for our key-value maps*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ *generates public parameters.*
$\mathsf{Com}(\mathsf{pp}, \{v_k\}_{k\in K}; r) \to c$ *commits to the key-value map with randomness r. We keep the randomness implicit whenever it does not affect clarity and we assume it to be sampled from an additive group.*[13]

Definition 3 (Hiding). *A key-value map commitment is hiding if for all key-value maps $\{v'_{k'}\}_{k'\in K'}, \{v''_{k''}\}_{k''\in K''}$ (even of different size), for $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ the following two distributions are computationally indistinguishable:*

$$\{\mathsf{Com}(\mathsf{pp}, \{v'_{k'}\}_{k'\in K'})\} \approx \{\mathsf{Com}(\mathsf{pp}, \{v''_{k''}\}_{k''\in K''})\}$$

Definition 4 (Binding). *A key-value map commitment is (computationally) binding if for any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{matrix}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (c, \{v'_{k'}\}_{k'\in K'}, r', \\ \{v''_{k''}\}_{k''\in K''}, r'') \leftarrow \mathcal{A}(\mathsf{pp})\end{matrix} : \begin{matrix} c = \mathsf{Com}(\mathsf{pp}, \{v'_{k'}\}_{k'\in K'}, r') \wedge \\ c = \mathsf{Com}(\mathsf{pp}, \{v''_{k''}\}_{k''\in K''}, r'') \wedge \\ \{v'_{k'}\}_{k'\in K'} \neq \{v''_{k''}\}_{k''\in K''}\end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

Definition 5 (Homomorphism). *We say a commitment to a key-value map is homomorphic if there exists an operation $\circ$ such that* Setup *always produces* pp *such that for all maps $\{v_k\}_{k\in K}, \{v'_{k'}\}_{k'\in K'}$ and randomness r, r' it holds that*

$$\mathsf{Com}(\{v_k\}_{k\in K}; r) \circ \mathsf{Com}(\{v'_{k'}\}_{k'\in K'}; r') = \mathsf{Com}(\{v^*_{k^*}\}_{k^*\in K^*}; r + r')$$

where $K^* = K \cup K'$ and $(k^*, v^*_{k^*}) = \begin{cases}(k^*, v_{k^*}) \text{ if } k^* \in K \setminus K' \\ (k^*, v'_{k^*}) \text{ if } k^* \in K' \setminus K \\ (k^*, v_{k^*} + v'_{k^*}) \text{ if } k^* \in K \cap K'\end{cases}$

[13] We will use this when defining the homomorphic property of commitments.

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: samples group $\mathbb{G}$; $g, h \leftarrow\!\!\$\ \mathbb{G}$; return $\mathsf{pp} = (\mathbb{G}, g, h)$.
$\mathsf{Com}(\mathsf{pp}, \{v_k\}_{k\in K}; r) \to c$: return $\prod_{k\in K} \mathcal{H}(k)^{v_k} h^r$.

Fig. 2: Our construction for kvC.

3.1 Construction

We recap some of the properties we are interested in obtaining in our construction: (i) support large key universe; (ii) small commitments and small parameters; (iii) homomorphic (Definition 5); (iv) support efficient non-interactive zero-knowledge proof of knowledge of opening (in particular they should run in time linear in the density of the key-value map).

In Figure 2 we propose a construction based on random-oracle with the properties above. Our commitment construction is an extension to key-value maps of the one in [PBF+19]. Given a prime p we consider the universe of values $\mathcal{V} = \mathbb{Z}_p$, a group $\mathbb{G}$ isomorphic to it and for which the GDLOG assumption holds, a hash function $\mathcal{H}$ modeled as a random oracle and an arbitrary key universe $\mathcal{K}$ such that $\mathcal{H} : \mathcal{K} \to \mathbb{G}$. We prove Theorem 1 in Appendix B.

Theorem 1. *If $\mathcal{H}$ is a random oracle and under the GDLOG assumption the construction in Figure 2 is a kvC with value universe $\mathbb{Z}_p$.*

4 Arguments on Key-Value Commitments (Doubly-Private Setting)

Here we formalize and construct zero-knowledge arguments over key-value map commitments for the setting in which there is no information on the keys available in the system and those we are using in our proof.

Circuits over Key-Value Maps To support arbitrary computation on committed key-value maps, we provide an interface which supports any arithmetic circuit of the following form. The keys and values kv as well as an additional witness ω are field elements in $\mathbb{F}$. The circuit consists of multiplication gates of the form $\mathbb{F} \times \mathbb{F} \to \mathbb{F}$. They have an unbounded outbound degree and any linear relations are directly expressed between outputs and inputs.

We write any circuit using multiplication gates in the domain $\mathcal{KV}^\ell \times \mathbb{F}^{n_\omega}$ as $C^{\mathbb{F}}(\mathsf{kv}_1, \ldots, \mathsf{kv}_\ell, \omega)$. This circuit depends on the desired property the openings should have. Here ω is an additional private witness that may not depend on the opening to the key-value maps.

4.1 Arguments for Circuits over Committed Key-Value Maps

Here we present the overview of our argument system which works over committed key-value maps and takes an arbitrary inner circuit operating on the openings of the commitments. To be clear, we spell out its formalization explicitly but it is a special case of Definition 1. Given an inner circuit $C^{\mathbb{F}}$ as described above, our high level interface for proofs has the following form:

$\mathsf{kvNIZK.Setup}(1^\lambda) \to \mathsf{crs}$ takes a security parameter λ and outputs a crs.
$\mathsf{kvNIZK.Prove}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega) \to \pi$ takes the crs and a circuit $C^{\mathbb{F}}$ as well as ℓ commitments c_i and their openings (kv_i, ρ_i) and an auxiliary witness ω. It outputs a proof π
$\mathsf{kvNIZK.VerProof}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, \pi) \to b \in \{0, 1\}$ takes the crs, a circuit $C^{\mathbb{F}}$, and ℓ commitments c_i. The output bit b returns the validity of the proof π.

The relation we want to prove is defined by the circuit C^* in Equation (1). To clarify our notation we re-define correctness for arguments for committed key-value maps.

Definition 6 (kvNIZK Correctness). *A kvNIZK is correct if, for any $\lambda \in \mathbb{N}$ with $\mathsf{crs} \in \mathsf{kvNIZK.Setup}(1^\lambda)$, circuit $C^{\mathbb{F}}$, key-value maps $\vec{\mathsf{kv}} \in \mathcal{KV}^\ell$ and randomness $\vec{\rho} \in \mathbb{F}^\ell$ with $\forall i \in [\ell] : c_i = \mathsf{Com}(\mathsf{kv}_i, \rho_i)$ and any $\omega \in \mathbb{F}^{n_\omega}$ for which $C^*(c_1, \ldots, c_\ell; (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega) = 1$ it holds that*

$$\mathsf{kvNIZK.VerProof}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, \pi) = 1 \text{ where}$$
$$\pi \leftarrow \mathsf{kvNIZK.Prove}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega).$$

As for NIZKs, we require knowledge-soundness and zero-knowledge.

4.2 Construction with Intermediate Key-Tags

Our construction has two stages. First the prover creates some key-tags b_k and proves that they are well formed (i.e., they are obtained by hashing a key and masking with a random group element, both known to the prover). These key-tags are then used in a subsequent proof for the opening of the commitment and the actual relation. Intuitively, since the prover knows how the key-tags have been produced, the prover is able to compute openings of the input commitments under the new "base" $(h, b_1, \ldots, b_n)$—as opposed to the original base $(g, \mathcal{H}(k_1), \ldots, \mathcal{H}(k_n))$—by appropriately computing the randomizers as a function of the values in the commitment and the randomness used for producing the key-tags. This allows to avoid proving properties of the hash function in the second part of the proof. The full construction Z🥑-dp is presented in Figure 3. For sake of presentation and w.l.o.g., in the construction we assume that all our key-value maps include the same keys $k_1, \ldots, k_n$.

The protocol is described in the random oracle model where all parties have access to $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$.

$\mathsf{Setup}(1^\lambda)$**:** 1. compute $\mathsf{crs}_{\text{tags}} \leftarrow \mathsf{NIZK.Setup}_{\text{tags}}(1^\lambda)$, $\mathsf{crs}_{\text{C}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$.

2. generate the commitment key $h \xleftarrow{\$} \mathbb{G}$ (setup for kv commitments).
3. sample random generators $\vec{g} \xleftarrow{\$} \mathbb{G}^n$ where n is the maximum number of committed keys ($\leq$ number of active keys) in a commitment.
4. Output $\mathsf{crs} = (\mathsf{crs}_{\text{tags}}, \mathsf{crs}_{\text{C}}, h, \vec{g})$

$\mathsf{Prove}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega)$**:**

1. From all key-value maps kv_i, extract the set of non-zero keys K_i and define $(k_1, \ldots, k_n) := \bigcup_{i\in[\ell]}\{k \in K_i\}$. Parse $\mathsf{kv}_i = \{(k, v_{i,k}) : k = k_1, \ldots, k_n\}$ for each $i = 1, \ldots, \ell$
2. Sample random values $\vec{r} \xleftarrow{\$} \mathbb{F}^n$ and create the key-tags $b_{k_i} = \mathcal{H}(k_i)h^{r_i}$. Additionally, create a vector Pedersen commitment to all keys (called pre-image commitment) $c^* = h^s \prod_{i=1}^n g_i^{k_i}$ with randomness $s \xleftarrow{\$} \mathbb{F}$.
3. Generate a proof that the key-tags are well formed by proving knowledge of the pre-images k_i i.e., generate a proof
$$\pi_{\text{tags}} \leftarrow \mathsf{NIZK.Prove}_{\text{tags}}(\mathsf{crs}, b_{k_1}, \ldots, b_{k_n}, c^*; k_1, \ldots, k_n, \vec{r}, s)$$
for the relation
$$R_{\text{tags}}(b_{k_1}, \ldots, b_{k_n}, c^*; k_1, \ldots, k_n, \vec{r}, s) := (c^* = h^s \prod_{i=1}^n g_i^{k_i} \wedge \forall i \in [n] : b_{k_i} = \mathcal{H}(k_i)h^{r_i})$$
4. Generate a proof
$$\pi_{\text{C}} \leftarrow \mathsf{NIZK.Prove}_{\text{C}}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, b_{k_1}, \ldots, b_{k_n}, c^*; (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \vec{r}, s, \omega)$$
for the relation
$$R_{\text{C}}(C^{\mathbb{F}}, c_1, \ldots, c_\ell, b_{k_1}, \ldots, b_{k_n}, c^*; (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \vec{r}, s, \omega) := c^* = h^s \prod_{i=1}^n g_i^{k_i} \wedge \forall i \in [\ell] : c_i = h^{\rho_i - \sum_{j=1}^n r_j v_{i,k_j}} \prod_{j=1}^n b_{k_j}^{v_{i,k_j}} \wedge C^{\mathbb{F}}((\mathsf{kv}_i)_{i\in[\ell]}, \omega) = 1 \quad (2)$$
5. Return both proofs including the key-tags and the pre-image commitment $\pi := (b_{k_1}, \ldots, b_{k_n}, c^*, \pi_{\text{tags}}, \pi_{\text{C}})$

$\mathsf{VerProof}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, \pi)$**:**

1. Parse π as $(b_{k_1}, \ldots, b_{k_n}, c^*, \pi_{\text{tags}}, \pi_{\text{C}})$.
2. Verify $b_0 \leftarrow \mathsf{NIZK.VerProof}_{\text{tags}}(\mathsf{crs}, b_{k_1}, \ldots, b_{k_n}, c^*, \pi_{\text{tags}})$
3. Verify $b_1 \leftarrow \mathsf{NIZK.VerProof}_{\text{C}}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, b_{k_1}, \ldots, b_{k_n}, c^*, \pi_{\text{C}})$
4. return $b_0 \wedge b_1$

Fig. 3: Z🥑-dp, our construction for kvNIZK

Theorem 2. *Under the GDLOG assumption, if* NIZK_{tags} *and* NIZK_C *are secure (correct, zero-knowledge, knowledge-sound) NIZKs for their required relation families, then the construction* Z🥝-dp *is a secure kvNIZK for arbitrary circuits over the key-value map commitment in Fig. 2.*

Proof. Correctness: Correctness follows by inspection. In particular, note that when proving R_{C} the prover "opens" the commitments c_i under the base defined by the key-tags b_k's. Since the b_k's are generated with the same h as the original commitment and the prover knows the openings of the b_k's, the prover can find the right value to be used as exponent for h.

Knowledge-Soundness: To prove knowledge-soundness, assume the existence of extractors $\mathcal{E}_{\mathrm{tags}}, \mathcal{E}_{\mathrm{C}}$ for the two sub-relations. We build an extractor $\mathcal{E}^*$ that on input a statement $(C^{\mathbb{F}}, c_1, \ldots, c_\ell)$ and accepting proof $(b_1, \ldots, b_n, c^*, \pi_{\mathrm{tags}}, \pi_{\mathrm{C}})$, outputs $((\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega)$. Our $\mathcal{E}^*$ works as follows. It first extracts through $(k'_1, \ldots, k'_n, \vec{r}', s') \leftarrow \mathcal{E}_{\mathrm{tags}}(b_1, \ldots, b_n, c^*, \pi_{\mathrm{tags}})$ and then the rest through $((\mathsf{kv}_1, \rho'_1), \ldots, (\mathsf{kv}_\ell, \rho'_\ell), \vec{r}, s, \omega) \leftarrow \mathcal{E}_{\mathrm{C}}(C^{\mathbb{F}}, c_1, \ldots, c_\ell, b_1, \ldots, b_n, c^*, \pi_{\mathrm{C}})$ such that the two relations hold for the extracted witnesses. We first argue it holds that $(k'_1, \ldots, k'_\ell, s') = (k_1, \ldots, k_\ell, s)$, since otherwise we can construct an adversary that breaks the binding property of the Pedersen commitment c^*. Then, we show how to extract valid openings for the input commitments c_i. Remember that thanks to the knowledge-soundness of the second proof system (for relation R_{C}) we know that for all commitments c_i it holds $c = h^{\rho' - \sum_{j=1}^{n} r_j v_{k_j}} \prod_{j=1}^{n} b_{k_j}^{v_{k_j}}$ (we remove the index i here to improve readability) .

Thanks to the knowledge soundness of the first proof system (for relation R_{tags}) we know that $b_{k_j} = \mathcal{H}(k_j) h^{r'_j}$, thus we can rewrite c as

$$c = h^{\rho' - \sum_{j=1}^{n} r_j v_{k_j}} \prod_{j=1}^{n} \mathcal{H}(k_j)^{v_{k_j}} h^{r'_j \cdot v_{k_j}} \qquad = h^{\rho' - \sum_{j=1}^{n} (r_j - r'_j) v_{k_j}} \prod_{j=1}^{n} \mathcal{H}(k_j)^{v_{k_j}}$$

Thus our extractor can output $((\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega)$ with $\rho_i = \rho'_i - \sum_{j=1}^{n} (r_j - r'_j) v_{k_{i,j}}$ as the witness for the overall relation. Note that the proof does not guarantee that $\vec{r} = \vec{r'}$. However this is not a problem since we are still guaranteed that the extracted witness for the overall relation is a valid one. *Zero-Knowledge:* follows from the ZK property of the underlying arguments and the hiding property of the commitments. Details can be found in Appendix B.

4.3 How to Instantiate the Subprotocols in Z🥝-dp

To instantiate the well formedness of the tags, i.e. the relation R_{tags}, we propose to use a cryptographic hash function such as MiMC [AGR+16], Poseidon [GKR+21], GMiMC [AGP+19], or Marvellous [AAB+19] which are optimized for zero-knowledge proofs. They provide hashing to a field element ($\mathcal{H}^{\mathbb{F}} : \{0,1\}^* \to \mathbb{F}$) which can be leveraged to obtain random group elements through some of the techniques in [BCI+10, FFS+13, TK17]. A subsequent circuit then proves this hashing and the rerandomization of the group element as key-tags, i.e., $[b_{k_i}] = [\mathcal{H}(k_i)] + r_i \cdot [h]$, where brackets enclose group elements. The combined constraints can then proven by e.g. Bulletproofs [BBB+18] or [AC20].

The circuit for the relation R_{C} can be implemented through a generalization of Bulletproofs [LMR19][14]. They provide an interface supporting bilinear circuits with five gates to enable arbitrary computation of which we use a subset of gates for non-bilinear circuits only. Given a circuit C constructed from the available gates, they then provide an efficient protocol with communication complexity $6\lceil \log_2(|C|) \rceil + 28$ group elements.

5 Improvements in Practice: Offline/Online Stages

The proving algorithms of our constructions (both Z🥝-dp and Z🥝-set) follow a two-step template. In step *(a)* the prover provides key-tags b_k-s and proves they are valid. This can be done independently of the commitment. In a following step *(b)* we compute a proof about properties of the commitment opening (and that actually depends on the commitments). Crucially, in the latter step, the prover uses the key-tags as (rerandomized) "anchors" to the keys. We observe that we can exploit the fact that *(a)* does not depend on the commitments to the key-value maps (but only on the keys that they will contain) to preprocess this step.

Consider, for instance, the running example from the introduction where the commitments contain multi-currency wallets. Assume that the prover knows that *in the future* they will want to prove some properties about

[14] If malleability is a concern, Bulletproofs are proven to be non-malleable in the AGM [GOP+22].

some of their wallets (which are expected to keep changing between now and the proving time). Moreover, while a large set K_{active} of asset types might be circulating in the system, the prover knows that they will only hold a very specific and relatively small subset K_{pre} of these keys (e.g., maybe only ETH, USD and EUR). If that is the case they can preemptively perform an "offline proving stage" that would be valid for all of the *online* proofs they will have to carry out later. Specifically, in Z◉-dp the prover performs step *(a)* above offline as follows. On input set K_{pre}, the prover provides a set of $|K_{\text{pre}}|$ key-tags $B = \{b_k : k \in K_{\text{pre}}\}$, defined as usual as $b_k = \mathcal{H}(k)h^{r_k}$ together with a proof π_{tags} that they were constructed honestly. The output of this step is therefore $\pi_{\text{offl}} = (B, \pi_{\text{tags}})$. At a later time, when input commitments $c_1, \ldots, c_\ell$ are available, the prover uses the pre-computed set of key-tags B to produce a step (a) (the production of key-tags) for each of the commitments c_i. In order to preserve zero-knowledge, step (b) is modified to rerandomize the related b_k-s first. The rerandomization hides the mapping between online proofs, however the verifier learns that all commitments of the online phase contain the same set of keys. We can similarly adapt Z◉-set by performing a proof of membership and masking (step 1 in Fig. 4)before hand.

The advantage of this approach is to use a single offline stage for many proofs. The efficiency savings of this stage (both for proving and verification time) can be significant since it involves proving/verifying hashing in zero-knowledge. For example, we conservatively estimate approximately 5k constraints using Poseidon hashing on a Ristretto curve [GKR+21]. Each of these hashes can be proved in the order of hundreds of milliseconds (see, e.g., Table 5 in [GKR+21]). For $n \approx 100$ this involves for example saving half a million constraints[15] amounting to around half a minute of proving time. Savings for verification time are comparable.

Naturally the offline stage preprocesses an *upper bound* U on the total number of active keys, since each commitment may have openings to key-value maps of different density. This may incur a high overhead cost if U is far from the actual densities (because we still need to process U key-tags as input to the circuit). The gains from an offline stage can differ accordingly and should be weighed depending on the application.

Efficiency summary of our Constructions We summarize the (asymptotic) efficiency of our constructions in Table 1. We present it for the case with offline processing, but summing the offline and online columns corresponds to the setting without an offline stage.

6 Experimental Evaluation

Here we show the practical feasibility of our construction. Our focus is on Z◉-dp; we compare its efficiency to that of Z◉-set in Appendix A.4.

Recall that our construction-template uses two separate steps (see also beginning of Section 5 and Fig. 3): *(a)* validity of key-tags and *(b)* actual property on opening of commitment. We evaluate our construction on a representative application setting for cryptocurrencies, that is a 64-bit range proof as a circuit proven in step *(b)*.

Let n be the size of the opening of the commitment (also equal to the number of elements we are showing are in range). We estimate the following runtimes. For step *(a)*: $\approx n \cdot 700$ms for proving and $n \cdot 100$ms for verification; for step *(b)* $\approx n \cdot 235$ms for proving and $n \cdot 89$ms verification. We stress that proving times for step *(a)* are fully parallelizable (as we generate n independent proofs for key-tags).

These timings refer to those for a common laptop (i7-6820HQ CPU at 2.70 GHz) and aim at estimating an efficient instantiation through the zero-knowledge scheme LMR [LMR19] as $\mathsf{NIZK}_{\mathsf{tags}}$ and $\mathsf{NIZK}_{\mathrm{C}}$ using Ristretto Curve as an underlying group.

How we derive timings. A similar derivation for Bulletproof timings was previously used in [LRR+19]. For each timing we use the formula $T(n) \approx n \cdot \mathsf{num\_of\_constraints\_circuit}_{\mathrm{LMR}} \cdot \mathsf{cost\_per\_constraint}$. **Deriving step *(a)*:** for proving, $\mathsf{cost\_per\_constraint}$ is measured to be $\approx 8.97/64$ ms/constraint (our experimental finding) for the implementation in [dal]. For verification 1.22/64 ms/constraint. We estimate $\mathsf{num\_of\_constraints\_circuit}_{\mathrm{LMR}}(\mathsf{tag}) \approx 5k$: for a circuit for Poseidon hash [GKR+21] and fixed base exponentiation (for rerandomization) of curve points[16][17] requires $L = 2806$ multiplicative constraints. To use this in LMR [LMR19] we need to encode

[15] For some applications this can be huge—for comparison, the ZCash circuit [HBHW21] has approximately 100k constraints.

[16] This is a lower bound but we expect it be a reasonable estimate (up to approximately a factor 2) of hashing-to-group techniques close to those in Section 1.1 in [BCI+10].

[17] Since there is no public circuit implementation for Ristretto operations for this, we use arkworks [ark] BL12-381 implementation for this estimate. We expect this to be an upper bound on Ristretto points given their smaller field size—$\frac{255}{381}$x smaller, more precisely. We measure this number using the implementation in [ark]

Table 1: Efficiency of our constructions and comparison with non-homomorphic solutions (when they are applicable). Above we describe proof sizes, proving time and verification times during offline and online stage. We also describe the additional costs for proving the homomorphism in zero-knowledge with a non-homomorphic solution ($\mathsf{kv}_{\mathrm{add}}(n)$). All values are implicitly in big O notation and denote operations in a prime-order group unless underlined. Rows marked with $\star$ refer to this work. The construction of Z🥝-dp is in Fig. 3. "Z🥝-set (Acc)" refers to the instantiation of Z🥝-set with NIZKs over accumulators in unknown-order groups we describe in Appendix C (the more general construction is in Fig. 4 in Appendix A). "Pedersen (Non-Hom.)" refers to the non-homomorphic solution based on Pedersen described in the introduction. Typical values for our parameters could be $M \approx 1000$ and $n \approx 100$.

$K_{\mathbf{active}}$	$K_{\mathbf{com}}$	**Construction**	$\lvert\pi_{\mathbf{offl}}\rvert$	$\lvert\pi_{\mathbf{onl}}\rvert$	$\lvert\pi(\mathsf{kv}_{\mathbf{add}}(n))\rvert$
priv	priv	Z🥝-dp $\star$	$\log(n(\lvert\mathcal{H}\rvert + \lvert\mathbb{G}_{\mathrm{add}}\rvert))$	$\log(\lvert C\rvert)$	—
priv	priv	Pedersen (Non-Hom.)	—	$\log(\lvert C\rvert)$	$\log(n)$
publ	priv	Z🥝-set (Acc) $\star$	$n + \log(n\lvert\mathbb{G}_{\mathrm{add}}\rvert)$	$\log(\lvert C\rvert)$	—

$K_{\mathbf{active}}$	$K_{\mathbf{com}}$	**Construction**	$\mathcal{V}_{\mathbf{offl}}$	$\mathcal{V}_{\mathbf{onl}}$	$\mathcal{V}(\mathsf{kv}_{\mathbf{add}}(n))$
priv	priv	Z🥝-dp $\star$	$n(\lvert\mathcal{H}\rvert + \lvert\mathbb{G}_{\mathrm{add}}\rvert)$	$\lvert C\rvert$	—
priv	priv	Pedersen (Non-Hom.)	—	$\lvert C\rvert$	n
publ	priv	Z🥝-set (Acc) $\star$	$\underline{n\lvert\mathbb{G}_?\rvert} + n\lvert\mathbb{G}_{\mathrm{add}}\rvert$	$\lvert C\rvert$	—

$K_{\mathbf{active}}$	$K_{\mathbf{com}}$	**Construction**	$\mathcal{P}_{\mathbf{offl}}$	$\mathcal{P}_{\mathbf{onl}}$	$\mathcal{P}(\mathsf{kv}_{\mathbf{add}}(n))$
priv	priv	Z🥝-dp $\star$	$n(\lvert\mathcal{H}\rvert + \lvert\mathbb{G}_{\mathrm{add}}\rvert)$	$\lvert C\rvert$	—
priv	priv	Pedersen (Non-Hom.)	—	$\lvert C\rvert$	n
publ	priv	Z🥝-set (Acc) $\star$	$\underline{(M - n + n\log n)\lvert\mathbb{G}_?\rvert} + n\lvert\mathbb{G}_{\mathrm{add}}\rvert$	$\lvert C\rvert$	—

N :	Size of key universe $\mathcal{K}$
M :	Number of active / registered keys (K_{active})
n :	Number of keys in the opening of a key-value map commitment
	$N >> M \geq n$
$\lvert\mathcal{H}\rvert$:	Number of constraints for hashing to a group element
$\lvert\mathbb{G}_{\mathrm{add}}\rvert$:	Number of constraints for summing two group elements
$\lvert\mathbb{G}_?\rvert$:	Cost of exponentiation in unknown-order group
$\mathsf{kv}_{\mathrm{add}}(n)$:	Sum operation among key-value maps of size n
C :	Circuit computed on key-value map.

this as a witness vector (a very conservative upper bound is $2L$, which we approximate to $5k$). **Deriving step *(b)*:** for proving, $\mathsf{cost\_per\_constraint}$ is measured to be $\approx$ 232/64 ms/constraint. For verification 88/64 ms/constraint. We derive these estimations from BL12-381[18]; we know this to be a fair estimate for Ristretto [dal]. $\mathsf{num\_of\_constraints\_circuit}_{\mathrm{LMR}}(\mathsf{range}_{64}) \approx 65$ constraints.

7 Application: Multi-Type QuisQuis

QuisQuis [FMMO19] is a privacy-preserving transaction system which allows for pruning old transactions, keeping the state of each participant linear in the number of users. This is a major advantage over other privacy-preserving transaction systems, which require a state size linear in the number of transactions. A QuisQuis transaction is a redistribution of tokens among a set of accounts. An account belongs to an owner and stores their tokens. Instead of consuming accounts and creating new ones, QuisQuis updates the accounts. These update operations need to change the balance of a peer without knowing their total balance. This is achieved with homomorphic commitments.

In contrast to the original QuisQuis where an account stored a scalar value representing an amount, we generalize accounts to store tokens of different types in a key-value map. Each key corresponds to a type, also known as currency, and the value specifies how many tokens of the specific type are held by the account. An account acct then belongs to a secret key sk and holds a balance kv, represented as a key-value map. To transfer tokens from one account to another, a transaction includes both accounts as active inputs, denoted by the set $\mathcal{P}$. The transaction subtracts tokens from one account with a secret index s and adds them to the other one. The output of the transaction are two updated accounts belonging to the same users as the inputs; the input accounts are discarded. To achieve anonymity, the input consists of $\mathcal{P}$ together with a potentially large anonymity set of accounts $\mathcal{A}$ which keep their balance unchanged but provide set anonymity for the active accounts.

As the central building block of our multi-type QuisQuis system, we present an updatable account based on our key-value commitments.

7.1 Multi-Type QuisQuis: Syntax

The original QuisQuis transaction protocol consists of the three algorithms ($\mathsf{Setup}, \mathsf{Trans}, \mathsf{Verify}$). Their multi-type equivalent is as follows:

$\mathsf{Setup}(1^\lambda, \vec{\mathsf{kv}}) \to \mathsf{state}$**:** takes the security parameter λ and a vector of key-value balances $\vec{\mathsf{kv}}$ and outputs an initial state state. One part of the state is a set of unspent accounts where each key-value balance has an account.

$\mathsf{Trans}(\mathsf{sk}, \mathcal{P}, \mathcal{A}, \delta\vec{\mathsf{kv}}) \to \mathsf{tx}$**:** takes a secret key sk which corresponds to one account in the set of active accounts $\mathcal{P}$ and an anonymity set $\mathcal{A}$ with a vector of key-value maps $\delta\vec{\mathsf{kv}}$ to update tokens. Trans outputs a transaction tx.

$\mathsf{Verify}(\mathsf{state}, \mathsf{tx}) \to \bot/\mathsf{state}'$**:** takes a state and a transaction tx and outputs a new state' or fails with $\bot$.

To support dynamic registration of new types, we require an additional algorithm $\mathsf{Register}$, which is defined as:

$\mathsf{Register}(\mathsf{acct}, k, v_k) \to \mathsf{tx}$ takes an account acct and a new type k with amount v_k and outputs a transaction tx.

A registration transaction is accepted by Verify if the type k has not been registered before. We define the correctness of a transaction system more formally. Let for all $\lambda \in \mathbb{N}$ and $\vec{\mathsf{kv}}$ with $R^{\mathsf{kv}}_{\mathsf{rng}}(\mathsf{kv}_i) = 1$ be $\mathsf{state} \leftarrow \mathsf{Setup}(1^\lambda, \vec{\mathsf{kv}})$. For all accounts in $\mathcal{P}, \mathcal{A}$ with index sets $\mathcal{P}^* := \{i \in [|\mathsf{Sort}(\mathcal{P} \cup \mathcal{A})|] : \mathsf{acct}_i \in \mathcal{P}\}$ and $\mathcal{A}^*$ accordingly in a canonically ordered form with Sort. All accounts in $\mathcal{P} \cup \mathcal{A}$ are part of the UTXO set in state, all $\delta\vec{\mathsf{kv}}$ with $R^{\mathsf{kv}}_{\mathsf{rng}}(-\mathsf{kv}_s) = 1$ and $R^{\mathsf{kv}}_{\mathsf{rng}}(\mathsf{kv}_i) = 1$ for $i \in \mathcal{P}^*\{s\}$ and $\mathsf{kv}_i = \emptyset_{kv}$ for $i \in \mathcal{A}^*$ and sk corresponding to an account $\mathsf{acct}_s \in \mathcal{P}$ with enough tokens such that after the transaction there is no negative type $R^{\mathsf{kv}}_{\mathsf{rng}}(\mathsf{kv}_s + \delta\mathsf{kv}_s) = 1$, it holds that $\mathsf{Verify}(\mathsf{state}, \mathsf{Trans}(\mathsf{sk}, \mathcal{P}, \mathcal{A}, \delta\vec{\mathsf{kv}})) = \mathsf{state}'$ where $\mathsf{state}' \neq \bot$ and contains an updated UTXO set with all inputs $\mathcal{P} \cup \mathcal{A}$ removed and the transaction outputs added.

[18] We use a different implementation [zen] on BL12-381 points as the implementation in [dal] is not compatible with BL12-381

Multi-Type QuisQuis: Security The security of a QuisQuis-like transaction system consists of two main properties. The first property we need to achieve is *anonymity*. A transaction system is anonymous if an adversary cannot successfully distinguish two transactions. The transactions are created according to malicious instructions after the adversary has interacted with an oracle signing transactions on behalf of uncompromised participants.

The second property is *theft prevention*. This entails that *(i)* the adversary cannot steal tokens from uncompromised accounts; *(ii)* the adversary cannot create tokens out of thin air. Slightly more formally, we model these properties as follows. While interacting with the aforementioned signing oracle the balance of honest accounts (not controlled by the adversary) must not decrease. Additionally, the total amount of tokens must not increase from transaction to transaction. Notice, however, the number of tokens may increase as the result of mining or a token registration—the latter counts as a "genesis" transaction.

Our variant of QuisQuis with multiple token types shares many of the same properties as the non-type aware system. We refer the reader to the original QuisQuis paper [FMMO19] for details.

7.2 Construction

We construct the multi token QuisQuis scheme following the original QuisQuis but with two main adaptations: each account holds tokens in multiple types and making sure a transaction guarantees that the amounts of tokens are balanced for each of the token types.

The details of updatable accounts for key-value maps are presented in Appendix D. In a nutshell they have the same algorithms as the original QuisQuis construction but allow for multiple kinds of tokens.

Setup The setup algorithm generates a list of updatable accounts, one for each initial balance key-value map.

Trans Our transaction structure follows that in QuisQuis where a "transaction" denotes a redistribution of wealth among all accounts involved ($\mathcal{P} \cup \mathcal{A}$). The transaction takes a vector of key-value maps, one for each account. The account is then updated according to the key-value map. Key-value maps that contain only valid positive values ($R^{\mathsf{kv}}_{\mathsf{rng}}(\delta\mathsf{kv}_i) = 1$) are used to deposit tokens to receiving accounts. In order to ensure that the total number of tokens is preserved, we require that one key-value map holds negative values. This is to satisfy that the sum of all key-value maps is zero, or $\sum_{i=1}^{|\vec{\delta\mathsf{kv}}|} \delta\mathsf{kv}_i = \emptyset_{kv}$. For the account with the negative key-value map (indexed by s in the canonically ordered set $\mathcal{P} \cup \mathcal{A}$), the transaction signature ensures that the owner of the account acct_s authorizes the spending by proving knowledge of the matching secret key sk. The algorithm $\mathsf{Trans}((s, \mathsf{sk}_s, \mathsf{kv}_s), \mathcal{P}, \mathcal{A}, \vec{\mathsf{kv}})$ performs the following steps:

1. Parse all input accounts $\mathcal{P} \cup \mathcal{A} = \{\mathsf{acct}_1, \ldots, \mathsf{acct}_n\}$ and check the spending account is valid by checking $\mathsf{VerifyAcct}(\mathsf{acct}_s, (\mathsf{sk}_s, \mathsf{kv}_s)) = 1$. The transaction needs to be balanced: $\sum_{i=1}^{|\vec{\delta\mathsf{kv}}|} \delta\mathsf{kv}_i = \emptyset_{kv}$ and all key-value maps other than the spending account must be non-negative $\forall i \neq s : R^{\mathsf{kv}}_{\mathsf{rng}}(\delta\mathsf{kv}_i) = 1$. To support large anonymity sets $\mathcal{A}$, we choose to disclose the upper bound on active accounts by showing that $\delta\mathsf{kv}_i = \emptyset_{kv}$ for $i \in \mathcal{A}^*$ instead or a range proof. The spending account must be negative $R^{\mathsf{kv}}_{\mathsf{rng}}(-\delta\mathsf{kv}_s) = 1$ and the resulting account must be valid $R^{\mathsf{kv}}_{\mathsf{rng}}(\mathsf{kv}_s + \delta\mathsf{kv}_s) = 1$, to prevent overspending.
2. Let $\mathsf{outputs} = (\mathsf{acct}^{\mathcal{T}}_1, \ldots, \mathsf{acct}^{\mathcal{T}}_n)$ be a canonical order of the accounts generated by $\mathsf{UpdateAcct}(\mathcal{P} \cup \mathcal{A}, \vec{\delta\mathsf{kv}}; r)$.
3. Let $\psi : [n] \to [n]$ be the permutation that maps the canonically ordered inputs to the canonically ordered outputs, i.e. input i has the same secret key as output $\psi(i)$.
4. Create a zero knowledge proof π showing that the transaction is well formed, i.e. that it satisfies the following relation:

$$R_{\mathsf{tx\text{-}wf}} : \begin{cases} \mathbb{x} = (\mathsf{inputs}, \mathsf{outputs}), \mathbb{w} = (\mathsf{sk}, \mathsf{kv}_s, \vec{\delta\mathsf{kv}}, r = (r_1, r_2), \psi) \text{ s.t.} \\ \mathsf{VerifyUpdateAcct}(\mathsf{acct}^{\mathcal{S}}_i, \mathsf{acct}^{\mathcal{T}}_{\psi(i)}, \delta\mathsf{kv}_i) = 1 \forall i \in \mathcal{P}^* \\ \wedge R^{\mathsf{kv}}_{\mathsf{rng}}(\delta\mathsf{kv}_i) = 1 \forall i \in \mathcal{P}^*/\{s\} \wedge R^{\mathsf{kv}}_{\mathsf{rng}}(-\delta\mathsf{kv}_s) = 1 \\ \wedge \delta\mathsf{kv}_i = \emptyset_{kv} \forall i \in \mathcal{A}^* \wedge \sum_{i=1}^{n} \delta\mathsf{kv}_i = \emptyset_{kv} \\ \wedge \mathsf{VerifyAcct}(\mathsf{acct}^{\mathcal{T}}_{\psi(s)}, \mathsf{sk}, \mathsf{kv}_s + \delta\mathsf{kv}_s) = 1. \end{cases}$$

The relation ensures that the permuted output account is correctly updated by the transferred balance $\delta\mathsf{kv}_i$ for all active accounts. It then ensures that the updated key-value maps are valid, i.e. there is one spending account at index s and no value is taken from other accounts. The balances of the accounts in the anonymity set must not change. To ensure that the spender has enough tokens, the proof checks that the updated spender account has no negative balance. The transaction consists of the inputs, outputs and the proof π.

Verify A transaction is valid in respect to a state if all accounts in inputs have not been used in another transaction and the proof π is valid.

Security Analysis Our key-value commitments provide the same hiding and binding properties as the commitments to single scalars used in QuisQuis. The construction is a parallel version of the single type case and thereby the theft security holds also for all keys in parallel. Regarding anonymity, we achieve the same properties as QuisQuis, if we define an upper bound of the number of types involved in a transaction. For transactions with few different types, we achieve this through padding. With a constant size transaction proof, our new transactions are as indistinguishable as the original QuisQuis transactions.

Acknowledgements. Research supported by: the Concordium Blockhain Research Center, Aarhus University, Denmark; the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM); the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC). This work was partly produced while Matteo Campanelli and Felix Engelmann were affiliated with Aarhus University. We thank the anonymous reviewers for their valuable comments.

References

AAB+19. Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426, 2019. `https://eprint.iacr.org/2019/426`.

AC20. Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.

ACR20. Thomas Attema, Ronald Cramer, and Matthieu Rambaud. Compressed sigma-protocols for bilinear circuits and applications to logarithmic-sized transparent threshold signature schemes. Cryptology ePrint Archive, Report 2020/1447, 2020. `https://eprint.iacr.org/2020/1447`.

AGP+19. Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for MPC, and more. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pages 151–171. Springer, Heidelberg, September 2019.

AGR+16. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.

AK20. Archita Agarwal and Seny Kamara. Encrypted key-value stores. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 62–85. Springer, Heidelberg, December 2020.

AR20. Shashank Agrawal and Srinivasan Raghuraman. KVaC: Key-Value Commitments for blockchains and beyond. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 839–869. Springer, Heidelberg, December 2020.

ark. ark-works. `http://arkworks.rs`.

BBB+18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

BBF19. Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Heidelberg, August 2019.

BCFK19. Daniel Benarroch, Matteo Campanelli, Dario Fiore, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255, 2019. `https://eprint.iacr.org/2019/1255`.

BCG+14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

BCI+10. Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, Heidelberg, August 2010.

BG18. Jonathan Bootle and Jens Groth. Efficient batch zero-knowledge arguments for low degree polynomials. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 561–588. Springer, Heidelberg, March 2018.

BH11. Johannes Buchmann and Safuat Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2011.

CF13. Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Heidelberg, February / March 2013.

CFG+20. Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. Incrementally aggregatable vector commitments and applications to verifiable decentralized storage. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 3–35. Springer, Heidelberg, December 2020.

CHA21. Matteo Campanelli and Mathias Hall-Andersen. Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. *IACR Cryptol. ePrint Arch.*, 2021:327, 2021.

dal. Dalek bulletproofs implementation. `https://github.com/zkcrypto/bulletproofs.git`.

EMP+21. Felix Engelmann, Lukas Müller, Andreas Peter, Frank Kargl, and Christoph Bösch. SwapCT: Swap confidential transactions for privacy-preserving multi-token exchanges. *PoPETs*, 2021(4):270–290, October 2021.

FFS+13. Reza R Farashahi, Pierre-Alain Fouque, Igor Shparlinski, Mehdi Tibouchi, and J Voloch. Indifferentiable deterministic hashing to elliptic and hyperelliptic curves. *Mathematics of Computation*, 82(281):491–512, 2013.

FMMO19. Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. Quisquis: A new design for anonymous cryptocurrencies. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 649–678. Springer, Heidelberg, December 2019.

GKR+21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *30th USENIX Security Symposium*, 2021.

GOP+22. Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 397–426. Springer, Heidelberg, May / June 2022.

GPR+21. Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Annual International Cryptology Conference*, pages 395–425. Springer, 2021.

HBHW21. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification, version 2021.1.15, 2021.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

LMR19. Russell W. F. Lai, Giulio Malavolta, and Viktoria Ronge. Succinct arguments for bilinear group arithmetic: Practical structure-preserving cryptography. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2057–2074. ACM Press, November 2019.

LRR+19. Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. Omniring: Scaling private payments without trusted setup. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 31–48. ACM Press, November 2019.

LY10. Benoît Libert and Moti Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 499–517. Springer, Heidelberg, February 2010.

NM+16. Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.

PBF+19. Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *FC 2018 Workshops*, volume 10958 of *LNCS*, pages 43–63. Springer, Heidelberg, March 2019.

SAGL18. Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee. Proving the correct execution of concurrent services in zero-knowledge. In *13th USENIX Symposium on Operating Systems Design and Implementation OSDI 18)*, pages 339–356, 2018.

TK17. Mehdi Tibouchi and Taechan Kim. Improved elliptic curve hashing and point representation. *Designs, Codes and Cryptography*, 82(1):161–177, 2017.

TXN20. Alin Tomescu, Yu Xia, and Zachary Newman. Authenticated dictionaries with cross-incremental proof (dis)aggregation. Cryptology ePrint Archive, Report 2020/1239, 2020. `https://eprint.iacr.org/2020/1239`.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Heidelberg, March 2008.

YYD+19. Zheng Yi, Howard Ye, Patrick Dai, Sun Tongcheng, and Vladislav Gelfer. Confidential assets on MimbleWimble. Cryptology ePrint Archive, Report 2019/1435, 2019. `https://eprint.iacr.org/2019/1435`.

zen. Zengo-x bulletproofs implementation. `https://github.com/ZenGo-X/bulletproofs`.

Appendix

A Arguments on Key-Value Commitments with Public Active Keys

In this section we consider the case where the set of active keys is known by both the prover and the verifier. We can then add a feature by which in a setup phase it is possible to register a set of keys which are going to be used for later proofs. This can improve the amortized efficiency when several proofs share the same set of active keys. We dub our resulting construction Z🥝-set.

A.1 NIZKs with Key-Registration

We define NIZK for key-value maps which allow a key-registration phase. To this goal an algorithm $\mathsf{RegisterActiveSet}$ to the NIZK interface *deterministically* specializes a CRS to a set of active keys. Similar ideas have been used in previous work e.g., [BCFK19, CFG+20, CHA21] where a set-dependent specialized CRS is used.

Definition 7. *A NIZK for key-value maps with registration stage* kvNIZK^{reg} *consists of the following algorithms:*

$\mathsf{kvNIZK}^{\mathbf{reg}}.\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$ *on input* λ *it outputs a crs* crs.

$\mathsf{kvNIZK}^{\mathbf{reg}}.\mathsf{RegisterActiveSet}(\mathsf{crs}, K_{\mathbf{active}}) \to \mathsf{crs}^{\mathbf{reg}}$ *is a deterministic algorith that takes as input a* crs*, an active set* K_{active} *and outputs a specialized CRS* crs^{reg} [19].

$\mathsf{kvNIZK}^{\mathbf{reg}}.\mathsf{Prove}(\mathsf{crs}^{\mathbf{reg}}, K_{\mathbf{active}}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega) \to \pi$ *takes* crs^{reg}*, a circuit* $C^{\mathbb{F}}$*,* ℓ *commitments* c_i*, their openings* (kv_i, ρ_i) *and auxiliary witness* ω*. It outputs a proof* π*. Each of the* kv_i *should use keys that are subset of* K_{active}*. .*

$\mathsf{kvNIZK}^{\mathbf{reg}}.\mathsf{VerProof}(\mathsf{crs}^{\mathbf{reg}}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, \pi) \to b \in \{0,1\}$ *takes the* crs^{reg}*, a circuit* $C^{\mathbb{F}}$*, and* ℓ *commitments* c_i*.*

We require properties as in Definition 1: correctness (completeness), knowledge soundness, zero-knowledge. Definition 7 is *almost* a special case of the notion for kvNIZKs, but not quite. We could express it as a NIZK including K_{active} as input to verification and require whatever $\mathsf{RegisterActiveSet}$ does as part of the relation. However, as it's standard in NIZKs, that would require the verifier to run linearly in $|K_{\text{active}}|$. By taking it as input only a (potentially smaller) processing of it, $\mathsf{crs}^{\text{reg}}$, the verifier can even run sublinearly in the set of active keys (which is the case in our construction). This motivates the special interface we introduce in Definition 7. Otherwise, the required properties are straightforward variants of those for NIZKs.

The relation we want to prove is again that defined by the circuit in Equation (1). For sake of clarity, we spell out correctness.

Definition 8 (Correctness). *We say that a NIZK for key-value maps with registration stage* kvNIZK^{reg} *over key-value map commitment scheme* $\mathsf{C_{kv}}$ *is correct if for any* $\lambda \in \mathbb{N}$ *with* $\mathsf{crs} \in \mathsf{kvNIZK.Setup}(1^\lambda)$*, any* $\mathsf{crs} \in \mathsf{C_{kv}.Setup}(1^\lambda)$*, any set of keys* K_{active}*, any circuit* $C^{\mathbb{F}}$*, any key-value maps* $\vec{\mathsf{kv}} \in \mathcal{KV}^\ell$ *with keys all in* K_{active} *and randomness* $\vec{\rho} \in \mathbb{F}^\ell$ *with* $\forall i \in [\ell] : c_i = \mathsf{C_{kv}.Com}(\mathsf{crs}, \mathsf{kv}_i, \rho_i)$ *and any* $\omega \in \mathbb{F}^{n_\omega}$ *for which*

$$C(c_1, \ldots, c_\ell; (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega) = 1$$

it holds that $\mathsf{kvNIZK.VerProof}(\mathsf{crs}^{reg}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, \pi) = 1$ *where we compute* $\mathsf{crs}^{reg} \leftarrow \mathsf{RegisterActiveSet}(\mathsf{crs}, K_{active})$*,* $\pi \leftarrow \mathsf{kvNIZK.Prove}(\mathsf{crs}^{reg}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega)$*.*

A.2 Accumulator-based Construction

We require the two following building blocks (described below): *accumulators* and *NIZKs for membership over accumulators and rerandomization.*

The intuition behind our construction is to register the *already hashed* keys in an accumulator (without masking). This process can be performed publicly since producing the accumulator is deterministic. In order to output a proof, we produce a set of key-tags (masked version of the keys committed in the accumulator) and that these are appropriate rerandomized versions of the group elements in the accumulator.

[19] It is w.l.o.g. that we require the registration algorithm to register in batch the whole set of active keys. It would be possible to define a version where we incrementally register a new key to a previously preprocessed CRS. Several constructions, e.g. algebraic accumulators, would efficiently support this syntax.

A.2.1 Building Block: Accumulators

Definition 9 (Accumulator scheme). *An accumulator scheme* AccScm *over universe* $\mathcal{U}_\lambda(\mathsf{AccScm})$ *(where* λ *is a security parameter) consists of a quadruple of PPT algorithms* $\mathsf{AccScm} = (\mathsf{Setup}, \mathsf{Accum}, \mathsf{PrvMem}, \mathsf{VfyMem})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}_{\mathbf{acc}}$ *generates public parameters* pp_{acc}.
$\mathsf{Accum}(\mathsf{pp}_{\mathbf{acc}}, S) \to \mathsf{acc}$ *deterministically computes accumulator* acc *for set* $S \subseteq \mathcal{U}_\lambda(\mathsf{AccScm})$.
$\mathsf{PrvMem}(\mathsf{pp}_{\mathbf{acc}}, S, S') \to \pi_{acc}$ *computes witness* π_{acc} *that shows* $S' \subseteq S$.
$\mathsf{VfyMem}(\mathsf{pp}_{\mathbf{acc}}, \mathsf{acc}, S', \pi_{acc}) \to b \in \{0,1\}$ *verifies through witness whether subset* S' *is in the set accumulated in* acc. *We do not require parameter* S *to be in* $\mathcal{U}_\lambda(\mathsf{AccScm})$ *from the syntax.*

An accumulator scheme should satisfy correctness—the accumulator works as expected—and soundness—no efficient adversary can pick set S *and find a witness that checks on* $\mathsf{AccScm.Accum}(\mathsf{pp}_{acc}, S)$ *and* $S' \not\subseteq S$[20].

We assume the universe of the accumulator to be a group of prime order.

A.2.2 Building Block: NIZKs for membership and rerandomization We will use a NIZK for the following relation, which intuitively states that a set of group elements (fresh key-tags) are a rerandomization of elements accumulated in the set.

$$\begin{aligned} R_{\text{mem-rand}}((\mathsf{pp}_{\text{acc}}, h \in \mathbb{G}, \mathsf{acc}); (B = (b_1, \ldots, b_n) \in \mathbb{G}^n, \\ S = \{s_1, \ldots, s_n\} \in \mathbb{G}^n, \pi_{acc}, (r_1, \ldots, r_n) \in \mathbb{Z}_p^n)) \\ \text{s.t. } b_i = s_i \cdot h^{r_i} \text{ for } i \in [n] \text{ and } \mathsf{VfyMem}(\mathsf{pp}_{\text{acc}}, \mathsf{acc}, S) = 1 \end{aligned}$$

We refer to the NIZK for the relation above as $\mathsf{NIZK}_{\text{mem-rand}}$.

A.2.3 Construction We describe our construction in Fig. 4. In our description we assume the buildig blocks above and a general-purpose zero-knowledge scheme kvNIZK like the one assumed in Section 4.1.

The security of our construction can be argued analogously to the security proof of our construction in Fig. 3 given the security of $\mathsf{NIZK}_{\text{mem-rand}}$. The latter NIZK proves membership of the keys in the accumulator without revealing anything about them and simulation is straighforward from this observation. By the binding property of the accumulator and the knowledge-soundness of $\mathsf{NIZK}_{\text{mem-rand}}$ we are guaranteed that keys at proving time belong to the accumulated set.

A.3 Instantiation from Algebraic Accumulators in Groups of Unknown Order

It is possible to instantiate our construction above from accumulators in groups of unknown order [BBF19]. While we describe our instantiation in detail later in the appendix (see Appendix C), here we provide a high-level view of the proof system $\mathsf{NIZK}_{\text{mem-rand}}$. Our techniques are adaptations of those in [BCFK19] and their concrete improvements in [CHA21]. The proof system consists of two parts. One part proves in zero-knowledge the bulk of accumulator membership. This consists of a low communication-complexity protocol where the proof partly consists of elements in $\mathbb{G}_?$ (by which we denote the unknown-order group). The other component is a Bulletproof where we both perform some sanity check on the values in the first proof (for example, some hidden values claimed by the prover should be in a certain range) and we perform a rerandomization of the group elements. This construction can be instantiated in a completely transparent manner using class groups[BH11][21].

A breakdown of the communication complexity of the proof is as follows. Let n be the maximum number of keys in any of the key-value commitments input to the proof algorithm. The first component provides roughly δn elements in $\mathbb{G}_?$ and in the prime order group, for a small constant δ. The second component consists of roughly $\log(n|\mathbb{G}_{\text{add}}||C|)$ where $|C|$ is the size of the circuit representing the general relation on key-value maps and $|\mathbb{G}_{\text{add}}|$ is the cost of representing an addition in the prime-order group. The latter can be in the order of 1K constraints using some of the instantiations in [CHA21]. Concretely, for $n \approx 30$ we estimate the size of the proof to be lower than 180 KB plus the cost of a Bulletproof on the circuit C (of logarithmic size).

[20] These definitions are standard; see [BBF19] for a formal treatment.

[21] The more efficient instantiation based on RSA groups would require the existence of a trusted RSA modulus. Nonetheless, all the other components of the construction remain transparent.

$\mathsf{Setup}(1^\lambda)$: 1. compute accumulator parameters $\mathsf{pp}_{\text{acc}} \leftarrow \mathsf{AccScm.Setup}(1^\lambda)$
2. compute CRS for membership&randomize relation $\mathsf{crs}_{\text{mem-rand}} \leftarrow \mathsf{NIZK}_{\text{mem-rand}}.\mathsf{Setup}(1^\lambda)$
3. compute CRS for general-purpose NIZK $\mathsf{crs}_{\text{C}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
4. generate the commitment key $h \xleftarrow{\$} \mathbb{G}$.
5. sample random generators $\vec{g} \xleftarrow{\$} \mathbb{G}^n$ where n is the maximum number of keys in a commitment.
6. Output $\mathsf{crs} = (\mathsf{pp}_{\text{acc}}, \mathsf{crs}_{\text{mem-rand}}, \mathsf{crs}_{\text{C}}, h, \vec{g})$

$\mathsf{RegisterActiveSet}(\mathsf{crs}, K_{\mathbf{active}})$:
1. add hashes to an accumulator inside the new CRS by computing $\mathsf{acc} = \mathsf{AccScm.Accum}(\mathsf{pp}_{\text{acc}}, H)$ where $H := \{\mathcal{H}(k) : k \in K_{\text{active}}\}$
2. output $\mathsf{crs}^{\text{reg}} = (\mathsf{crs}, \mathsf{acc})$

$\mathsf{Prove}(\mathsf{crs}^{\mathbf{reg}}, K_{\mathbf{active}}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \omega)$: 1. Run steps 1,2 and 4 in $\mathsf{NIZK.Prove}$ as described in Figure 3, that is: 1) parse the input; 2) create the key-tags $b_{k_i} = \mathcal{H}_{\mathbb{G}}(k_i)h^{r_i}$ and commit to all keys $c^* = h^s \prod_{i=1}^n g_i^{k_i}$; and 4) Generate the proof π_{C} for the relation

$$\begin{aligned} R_{\text{C}}(C^{\mathbb{F}}, c_1, \ldots, c_\ell, b_{k_1}, \ldots, b_{k_n}, c^*; (\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), \vec{r}, s, \omega) := \\ c^* = h^s \prod_{i=1}^{n} g_i^{k_i} \wedge \forall i \in [\ell] : c_i = h^{\rho_i - \sum_{j=1}^n r_j v_{i,k_j}} \prod_{j=1}^{n} b_{k_j}^{v_{i,k_j}} \\ \wedge C^{\mathbb{F}}((\mathsf{kv}_i)_{i\in[\ell]}, \omega) = 1 \end{aligned} \tag{3}$$

2. Prove membership&randomize relation:

$$\pi_{\text{mem-rand}} \leftarrow \mathsf{NIZK}_{\text{mem-rand}}.\mathsf{Prove}(\mathsf{crs}_{\text{mem-rand}}, \mathbb{x}; \mathbb{w})$$

where

$$\mathbb{x} = (\mathsf{pp}_{\text{acc}}, h, \mathsf{acc}) \text{ and } \mathbb{w} = (B = (b_1, \ldots, b_n), K_{\text{active}}, \pi_{acc}, (r_1, \ldots, r_n)))$$

3. Return $\pi := (B, c^*, \pi_{\text{mem-rand}}, \pi_{\text{C}})$

$\mathsf{VerProof}(\mathsf{crs}^{\mathbf{reg}}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, \pi)$:
1. Parse π as $(B, c^*, \pi_{\text{mem-rand}}, \pi_{\text{C}})$
2. Verify $b_0 \leftarrow \mathsf{NIZK}_{\text{mem-rand}}.\mathsf{VerProof}(\mathsf{crs}_{\text{mem-rand}}, (\mathsf{pp}_{\text{acc}}, h, \mathsf{acc}))$
3. Verify $b_1 \leftarrow \mathsf{NIZK.VerProof}_{\text{C}}(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, B, c^*, \pi_{\text{C}})$
4. return $b_0 \wedge b_1$

Fig. 4: Z🥝-set, our accumulator-based construction for $\mathsf{kvNIZK}^{\text{reg}}$ (i.e., with public previously registered active keys)

A.4 Discussion on Practical Benefits of Z🥝-set

Intuitively, using an accumulator with a public set of active keys is beneficial whenever the "cost" of a proof of membership & randomization (step 1 in Fig. 4) is lower than that of a proof for hashing & randomization (steps 1–3 in Z🥝-dp, Fig. 3). Below we discuss when this can be the case; see also Table 1. *The bottomline is:* Z🥝-set *can improve verification time trading proof size and proving time.*

For proving time Fig. 4 is a worse choice because proving membership introduces a linear dependency in $|K_{\text{active}}|$, not present in our construction from Fig. 3. This is not just a matter of asymptotics: by our estimates, this would also be the case for common concrete parameters for accumulators in unknown-order groups. Similar observations hold for proof size.

Where we may see substantial gains from applying Z🥝-set is in verification time. Ignoring the cost of the circuit C and only considering the additive overhead we achieve the following verification times. Without registration (Z🥝-dp), we would have n times: one hashing subcircuit—each of approximately 5k constraints—plus rerandomization (additional 1k contraints). This gives a total of $6n$k constraints. With registration (Z🥝-set) we have: n group exponentiations in a group of unknown-order plus (proved with a special-purpose "accumulator-membership" NIZK; see also Appendix C), plus n additional rerandomizations and additional sanity check constraints (for a total around 1.5k constraints). To assess which solution is best one needs to compare the concrete costs of the n exponentiations in groups of unknown order vs the $n \times 4.5$k constraints of the hash function. We know the first to be cheaper and in particular to be an order of magnitude so (each exponentiation requiring tens of milliseconds, vs hundreds of milliseconds for the case of verifying a single hash function). This time difference in verification can be substantial. Our time estimates refer to RSA-groups and are derived from [GKR+21, CHA21, CFG+20].

B Additional Proofs

Proof of Theorem 1 As for standard Pedersen, the construction is perfectly hiding because of the masking factor h^r. As a consequence we have that for all $c \in \mathbb{G}$, for any two key-value maps $\{v_k\}_{k\in K}, \{v'_{k'}\}_{k'\in K'}$, the probability they yield commitment c is the same.

To show why the construction is binding we argue as follows. Assume an adversary that queries the random oracle on set K^* and that afterwards is able to provide two distinct openings for the same commitment c with non-negligible probability (each of these openings consisting of keys, respective values and randomnesses). We can write this commitment as $c = \prod_{k\in K} \mathcal{H}(k)^{v_k} h^r = \prod_{k'\in K'} \mathcal{H}(k')^{v'_{k'}} h^{r'}$ (where w.l.o.g. we assume $K^* = K \cup K'$) and observe that

$$\prod_{k\in K} \mathcal{H}(k)^{v_k} \prod_{k\in K'} \mathcal{H}(k')^{-v'_{k'}} h^{r-r'} = 1_{\mathbb{G}} \ .$$

Since the random oracle evaluations are distributed as random group generators we can use this adversary to break Assumption 1: we would pass it random generators $g_1, \ldots, g_{|K^*|}, h$ instead of the random oracle responses and return $(a_1, \ldots, a_{|K^*|}, r - r')$ where for each i we set $a_i = v_{k_i} - v'_{k_i}$ if $k_i \in K \cup K'$, else $a_i = v_{k_i}$ if $k_i \in K \setminus K'$ (and $a_i = v'_{k_i}$ if vice versa). The probability that there would exist a single non-zero element in the vector is non-negligible by assumption on the adversary.

Homomorphism follows easily by inspection.

Zero-Knowledge Proof in Theorem 2 For zero-knowledge, we build our simulator $\mathcal{S}^*$ using the simulators for the two subprotocols, namely $\mathcal{S}_{\text{tags}}, \mathcal{S}_{\text{R}}$. On input a valid statement $(\mathsf{crs}, C^{\mathbb{F}}, c_1, \ldots, c_\ell)$, $\mathcal{S}^*$ outputs $\pi = (b_1, \ldots, b_n, c^*, \pi_{\text{tags}}, \pi_{\text{C}})$ where: $b_1, \ldots, b_n, c^*$ are random group elements, and we build simulated proofs $\pi_{\text{tags}} \leftarrow \mathcal{S}_{\text{tags}}(\mathsf{crs}_{\text{tags}}, b_1, \ldots, b_n, c^*)$ and $\pi_{\text{C}} \leftarrow \mathcal{S}_{\text{C}}(\mathsf{crs}_{\text{C}}, C^{\mathbb{F}}, c_1, \ldots, c_\ell, b_1, \ldots, b_n, c^*)$. We can prove indistinguishability of the simulated and the real transcript through the following hybrids:

- *Hybrid 0.* This is the view in the real protocol.
- *Hybrid 1.* This is the same as the previous hybrid, but we replace π_{tags} with a simulated proof. Indistinguishability follows from the ZK property of the subprotocol for R_{tags}.
- *Hybrid 2.* This is the same as the previous hybrid, but we replace π_{C} with a simulated proof. Indistinguishability follows from the ZK property of the subprotocol for R_{C}.
- *Hybrid 3.* This is the same as the previous hybrid, but we replace $b_1, \ldots, b_n, c^*$ with random group elements. Indistinguishability follows since these group elements are distributed identically in both hybrid 2 and 3 due to the choice of the randomizers $s, \vec{r}$ in the real protocol/Hybrid 2. Note moreover that when replacing these

commitments with random group elements we do not change the valdity of the statements to be proven by the two simulators $\mathcal{S}_{\text{tags}}$ and $\mathcal{S}_{\text{C}}$. That is, $(b_1, \ldots, b_n, c^*)$ and $(C^{\mathbb{F}}, c_1, \ldots, c_\ell, b_1, \ldots, b_n, c^*)$ are true statements for the two sub-relations also when the group elements are randomly sampled. This is due to the fact that both relations are trivial: since Pedersen commitments are perfectly hiding, for any $\mathsf{kv}_1, \ldots, \mathsf{kv}_\ell$ there exist $s, \vec{r}$ such that the relation holds.

Since the distribution in Hybrid 3 is identical to a simulated execution, this concludes the proof of zero-knowledge.

C Zero-Knowledge over Algebraic Accumulators

The numbers in our Table 1 for the registration construction refer to instantiations described in this section.

Construction of accumulators for group elements Below we describe an instantiation of an accumulator for group elements pt on an elliptic curve. The original construction is that of accumulators in unknown-order groups from [BBF19], while the explicit description for elliptic curve points is from [CHA21]. The adaptations we describe in this section are important to obtain concretely efficient instantiations.

We define a permissible set $\mathcal{P}$ of commitments which allows to avoid collisions. It is parametrized by an integer μ and consists of elliptic curve points where the x-coordinate is a μ-bit prime and the y-coordinate is the "canonically chosen" square root so that the point can be described by its x-coordinate alone: $\mathcal{P} := \{(\mathsf{x}, \mathsf{y}) \in (\mathbb{F}, \mathbb{F}) \mid \mathsf{x} \in [2^{\mu-1}, 2^\mu) \wedge \mathsf{y} \equiv 0 \mod 2\}$

One can use $\mu = 251$ bits for concrete instantiations in the Ristretto curve (which we can use with [LMR19]).

Below we restrict elements to prime numbers. This is standard and necessary for the soundness of the accumulator scheme. We observe that, when the accumulator is computed publicly (which is the case in our applications), it is not necessary to prove in zero-knowledge that its elements are primes. This can be done at registration time ensuring that, when adding a key k, its hash $\mathcal{H}(k)$ is prime. We can naturally add a padding to the key before registration time and, through rejection sampling, ensure we obtain a prime.

$\underline{\mathsf{Setup}(1^\lambda) \to \mathsf{pp}}$

$(\mathbb{G}_?, g_?) \leftarrow \mathcal{G}_?(1^\lambda)$

return $\mathsf{pp} = (\mathbb{G}_?, g_?)$

$\underline{\mathsf{Add}(\mathsf{pp}, \mathsf{pt}, \mathsf{acc}) \to \mathsf{acc}'}$

Parse pt as $\mathsf{pt} := (\mathsf{x}, \mathsf{y})$

if $\mathsf{pt} \notin \mathcal{P} \vee \mathsf{x}$ not a prime **then**

return $\perp$; **else return** $\mathsf{acc}^{\mathsf{x}}$

$\underline{\mathsf{VfyMem}(\mathsf{pp}, \mathsf{acc}, \mathsf{pt}, \pi_{acc})}$

Parse pt as $\mathsf{pt} := (\mathsf{x}, \mathsf{y})$

Accept iff $\pi_{acc}^{\mathsf{x}} = \mathsf{acc}$

$\underline{\mathsf{PrvMem}(\mathsf{pp}, S, \mathsf{pt}) \to \pi_{acc}}$

$S' := \{\mathsf{x}' : (\mathsf{x}', \mathsf{y}') \in S \setminus \{\mathsf{pt}\}\}$

$\mathsf{prd} \leftarrow \prod_{\mathsf{x}' \in S'} \mathsf{x}'$

return $g_?^{\mathsf{prd}}$

Fig. 5: Accumulator Instantiation for AccScm.

For efficiency, we observe that PrvMem can executed in time growing with $(M - n) + n \log n$ for a subset of size n and accumulated set of size M (the quasilinear factor derives from $\mathsf{RootFactor}$ algorithm in [BBF19]).

Instantiating $\mathsf{NIZK}_{\text{mem-rand}}$ In Appendix A, we need to efficiently prove in zero-knowledge that a key-tag (roughly, the hash of a key) is inside an accumulator ($\mathsf{NIZK}_{\text{mem-rand}}$). For our instantiations in unknown-order groups, we can adopt a variant of the solution in [CHA21] (see their Section 5.2 and instantiations in 5.4.1 and 5.4.2). Our relation $R_{\text{mem-rand}}$ has only one difference from their "one-out-of-many" relation in Section 5.2: we need to prove the same relation for multiple key-tags (which correspond to what they call an "outer commitment"). In other words, we need a "many-out-of-many" relation.

The subsequent change to their instantiation is almost straightforward. Their one-out-of-many relation is composed of two parts: one described in 5.4.1 (roughly consisting of a Bulletproof execution on a gadget of

approximately 1.5K constraints) the other in 5.4.2 (a NIZK whose proof consists of a constant number of unknown-order and prime-order group elements). To turn them into a many-out-of-many variant we do the following. The constraint system for 5.4.1 can simply be adapted to take as input n inputs, instead of just one. The batching properties of Bulletproof allow the proof size to be affected only minimally. For the second component, the one from 5.4.2, we can simply provide n runs of it. This part of the argument system grows linearly in n.

D Updatable Accounts

D.1 Preliminaries: Updatable Public Keys [FMMO19]

We need updatable public keys. Intuitively, such a key can be publicly updated and the update is indistinguishable from a newly created key. An updatable public key scheme consists of the following algorithms:

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$ outputs the public parameters pp.
$\mathsf{Gen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$ generates a new key pair
$\mathsf{Update}(\mathsf{pk}) \to (\mathsf{pk}')$ updates the key pk to pk'
$\mathsf{VerifyKP}(\mathsf{pk}, \mathsf{sk}) \to b \in \{0,1\}$ verifies that a key pair is consistent.
$\mathsf{VerifyUpdate}(\mathsf{pk}, \mathsf{pk}', r) \to b \in \{0,1\}$ verifies that the public key pk was correctly updated to pk' with some randomness r.

The correctness follows directly from QuisQuis such that given a key pair, Update generates again a valid key pair. The update process is verifiable by $\mathsf{VerifyUpdate}$ and the updated key must keep the original secret key. The security requires (a) that an updated key is indistinguishable from a new, randomly generated one and (b) that the update process cannot change the secret key if the update is verified.

D.2 A secret-key version with updatable keys

We now describe an extended commitment to key-value map, which is the one we will use in our constructions for multi-type QuisQuis. In this extension we allow a party with a secret key to open the commitment even without knowing the randomness used to commit to it. This will be useful to allow to verify that an account has a certain balance. In order to do this we let the setup return a pair of public–secret keys $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$ and we use the notation $\mathsf{VerifyCommSk}(\mathsf{pk}, c, \mathsf{sk}, \{v_k\}_{k\in K})$. We also require the commitment to be binding also with respect to this function, that is no efficient adversary can output $(\mathsf{pk}, c, \mathsf{sk}, \{v_k\}_{k\in K}, \mathsf{sk}', \{v'_{k'}\}_{k'\in K'})$ with $\{v_k\}_{k\in K} \neq \{v'_{k'}\}_{k'\in K'}$ but such that $\mathsf{VerifyCommSk}(\mathsf{pk}, c, \mathsf{sk}, \{v_k\}_{k\in K}) = \mathsf{VerifyCommSk}(\mathsf{pk}, c, \mathsf{sk}', \{v'_{k'}\}_{k'\in K'}) = 1$. We provide a construction reminiscent of ElGamal commitments in Figure 6.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$**:** samples a group $\mathbb{G}$ and a generator g, $z \leftarrow\!\!\$\ [|\mathbb{G}|]$ and return $(\mathsf{pk} = (\mathbb{G}, g, h := g^z), \mathsf{sk} = z)$.
$\mathsf{Com}(\mathsf{pk}, \{v_k\}_{k\in K}; r) \to c$**:** return $(g^r, \prod_{k\in K} \mathcal{H}(k)^{v_k} h^r)$.
$\mathsf{VerifyCommSk}(\mathsf{pk}, c = (u, v), \mathsf{sk} = z, \{v_k\}_{k\in K})$**:** return 1 iff $\prod_{k\in K} \mathcal{H}(k)^{v_k} = v/u^z$.

Fig. 6: Secret-key variant of construction for kvC.

D.3 Formalization

With a key-value commitment scheme and updatable keys, we build updatable accounts which belong to a secret key and hold a key-value map of tokens. The important feature of an updatable account is, that anyone can rerandomize the account and create a proof of correct rerandomization. An updated account is not linkable to the previous version and indistinguishable from a newly created account. In combination with a shuffling proof, a set of accounts is permuted without changing ownership and without knowledge of openings. We provide formal details in the full version.

Definition 10. *Updatable Accounts consist of the following algorithms:*

$\mathsf{GenAcct}(\mathsf{pp}, \mathsf{kv}) \to (\mathsf{acct}, \mathsf{sk})$ *takes an amount map* $\{v_k\}_{k\in K}$ *and outputs an account* acct *and a secret key* sk

$\mathsf{VerifyAcct}(\mathsf{acct}, (\mathsf{sk}, \mathsf{kv})) \to b \in \{0,1\}^*$ *checks the account* acct *for consistency with the private key* sk *and the key-value map* $\{v_k\}_{k\in K}$ *such that* $R^{\mathsf{kv}}_{\mathsf{rng}}(\mathsf{kv}) = 1$.

These accounts express the same updatability as the original QuisQuis accounts, such that they remain valid for updates to the public key and homomorphic operations on the commitment. This leads to the following additional operations:

$\mathsf{UpdateAcct}(\{(\mathsf{acct}_i, \delta\mathsf{kv}_i)\}_{i=1}^n; r_1, r_2) \to \{\mathsf{acct}'_i\}_{i=1}^n$ *takes as input set of accounts* $\mathsf{acct}_i = (\mathsf{pk}_i, c_i)$ *and key-value maps* $\delta\mathsf{kv}_i$*, representing the change of amount in different types, such that* $R^{\mathsf{kv}}_{\mathsf{rng}}(\delta\mathsf{kv}_i) = 1$ *or* $R^{\mathsf{kv}}_{\mathsf{rng}}(-\delta\mathsf{kv}_i) = 1$*; it outputs a new set of accounts* $\{\mathsf{acct}'_i\}_{i=1}^n$ *with* $\mathsf{acct}'_i = (\mathsf{Update}(\mathsf{pk}_i; r_1)), c \circ \mathsf{Com}(\delta\mathsf{kv}_i, \mathsf{pk}_i; r_2)$

$\mathsf{VerifyUpdateAcct}(\{(\mathsf{acct}_i, \mathsf{acct}'_i, \delta\mathsf{kv}_i)\}_{i=1}^n; r_1, r_2) \to \{0,1\}$ *outputs* 1 *if* $\{\mathsf{acct}'_i\}_{i=1}^n = \mathsf{UpdateAcct}(\{(\mathsf{acct}_i, \delta\mathsf{kv}_i)\}_{i=1}^n, \mathsf{kv}; r_1, r_2)$ *and* $R^{\mathsf{kv}}_{\mathsf{rng}}(\delta\mathsf{kv}_i) = 1$ *or* $R^{\mathsf{kv}}_{\mathsf{rng}}(-\delta\mathsf{kv}_i) = 1$, 0 *otherwise.*

E Multi-Type QuisQuis Transaction Relation

With $(c_1, \ldots, c_\ell) := (\delta\mathsf{acct}_2, \ldots, \delta\mathsf{acct}_m, \mathsf{acct}'^{\mathcal{T}}_1, \delta\mathsf{acct}_1)$, we construct a circuit which checks that the first update account $\delta\mathsf{acct}_1$ is all negative and the remaining ones are all positive. Last, it checks that for each type, the update values sum to zero.

$$C^{\mathbb{F}}((k_1, (v_{1,k_1}, \ldots, v_{\ell,k_1})), \ldots, (k_n, (v_{1,k_n}, \ldots, v_{\ell,k_n})), \omega) :=$$
$$\forall_{k\in K} \begin{cases} R_{\mathsf{rng}}(-v_{\ell,k}) = 1 \\ \wedge \forall_{i\in[\ell-1]} R_{\mathsf{rng}}(v_{i,k}) = 1 \\ \wedge \sum_{i\in[\ell-2]\cup\{\ell\}} v_{i,k} = 0 \end{cases}$$

Our kvNIZK construction builds upon Pedersen style key-value commitments, but for the multi-type QuisQuis setting we need public key based commitments as shown in Appendix D.2. Therefore we change the relation R_{C} in Equation (3) to prove that the input to the circuit is still equivalent to the commitment openings.

$$R_{\mathrm{C}}(C^{\mathbb{F}}, ((g_1, h_1), c_1), \ldots, ((g_\ell, h_\ell), c_\ell), b_{k_1}, \ldots, b_{k_n}, c^*;$$
$$(\mathsf{kv}_1, \rho_1), \ldots, (\mathsf{kv}_\ell, \rho_\ell), r, \omega) :=$$
$$c^* = h^r \prod_{i=1}^n g_i^{k_i} \wedge \forall i \in [\ell] : c_i = (g_i^{\rho_i}, h_i^{\rho_i - \sum_{j=1}^n r_j v_{i,k_j}} \prod_{j=1}^n b_{k_j}^{v_{i,k_j}})$$
$$\wedge C^{\mathbb{F}}((k_1, (v_{1,k_1}, \ldots, v_{\ell,k_1}), \ldots, (k_n, (v_{1,k_n}, \ldots, v_{\ell,k_n})), \omega) = 1$$