Vector Commitments and their Applications

Dario Catalano[1] and Dario Fiore[2⋆]

[1] Dipartimento di Matematica e Informatica, Università di Catania, Italy.
catalano@dmi.unict.it
[2] Max Planck Institute for Software Systems (MPI-SWS)
fiore@mpi-sws.org

Abstract. We put forward the study of a new primitive that we call *Vector Commitment* (VC, for short). Informally, VCs allow to commit to an ordered sequence of q values $(m_1, \ldots, m_q)$ in such a way that one can later open the commitment at specific positions (e.g., prove that m_i is the i-th committed message). For security, Vector Commitments are required to satisfy a notion that we call *position binding* which states that an adversary should not be able to open a commitment to two different values at the same position. Moreover, what makes our primitive interesting is that we require VCs to be *concise*, i.e. the size of the commitment string and of its openings has to be independent of the vector length.

We show two realizations of VCs based on standard and well established assumptions, such as RSA, and Computational Diffie-Hellman (in bilinear groups). Next, we turn our attention to applications and we show that Vector Commitments are useful in a variety of contexts, as they allow for compact and efficient solutions which significantly improve previous works either in terms of efficiency of the resulting solutions, or in terms of "quality" of the underlying assumption, or both. These applications include: Verifiable Databases with Efficient Updates, Updatable Zero-Knowledge Databases, and Universal Dynamic Accumulators.

Keywords. Vector Commitments, Commitments, Accumulators, Zero-Knowledge Databases, Verifiable Databases.

1 Introduction

Commitment schemes are one of the most important primitives in cryptography. Informally, they can be seen as the digital equivalent of a sealed envelope: whenever a party S wants to commit to a message m, she puts m in the envelope. At a later moment, S opens the envelope to publicly reveal the message she committed to. In their most basic form commitment schemes are expected to meet two requirements. A commitment should be *hiding*, meaning with this that it should not reveal information about the committed message, and *binding* which means that the committing mechanism should not allow S to change her mind about m. More precisely, this means that the commitment comes with an opening procedure that can be efficiently verified, i.e. one should be able to efficiently check that the opened message is the one S originally committed to. Thus, a commitment scheme typically involves two phases: a *committing* one, where a sender S creates a commitment C on some messages m, using some appropriate algorithm and a *decommitting* stage, where S reveals m and should "convince" a receiver R that C contains m. A commitment scheme is said to be non-interactive if each phase requires only one messages from S to R.

Commitment schemes turned out to be extremely useful in cryptography and have been used as a building block to realize highly non-trivial protocols and primitives. Because of this, the basic properties discussed above have often turned out to be insufficient for realizing the desired functionalities. This led researchers to investigate more complex notions realizing additional properties

⋆ Work done while at NYU supported by NSF grant CNS-1017471.

and features. Here we discuss a couple of these extensions, those more closely related to the results presented in this paper.

Trapdoor commitment schemes (also known as *chameleon* commitments) come with a public key and a (matching) secret key (also known as the trapdoor). Knowledge of the trapdoor allows to completely destroy the binding property. On the other hand, the scheme remains binding for those who know only the public key. A special case of trapdoor commitments are (trapdoor) Mercurial commitments, a notion formalized by Chase *et al.* in [12]. Here the binding property is further relaxed to allow for two different decommitting procedures: a *hard* and a *soft* one. In the committing phase one can decide as whether to create a hard commitment or a soft one. A hard commitment is like a standard one: it is created to a specific message m, and it can be opened only to m. Instead, a soft commitment is initially created to "no message", and it can later be soft-opened (or *teased*) to any m, but it cannot be hard-opened.

1.1 Our Contributions

In this paper we introduce a new and simple, yet powerful notion of commitment, that we call *Vector Commitment* (VC, for short). Informally, VCs allow to commit to an ordered sequence of q values (i.e. a vector), rather than to single messages. This is done in a way such that it is later possible to open the commitment w.r.t. specific positions (e.g., to prove that m_i is the i-th committed message). More precisely, vector commitments are required to satisfy what we call *position binding*. Position binding states that an adversary should not be able to open a commitment to two different values at the same position. While this property, by itself, would be trivial to realize using standard commitment schemes, what makes our design interesting is that we require VCs to be *concise*, i.e., the size of the commitment string as well as the size of each opening have to be independent of the vector length.

Vector commitments can also be required to be *hiding*, in the sense that one should not be able to distinguish whether a commitment was created to a vector $(m_1, \ldots, m_q)$ or to $(m'_1, \ldots, m'_q)$, even after seeing some openings. We, however, notice that hiding is not a crucial property in the realization of vector commitments. Therefore, in our constructions we will not focus on it. While this might be surprising at first, we motivate it as follows. First, all the applications of VCs described in this paper do not require such a property. Second, hiding VCs can be easily obtained by composing a non-hiding VC with a standard commitment scheme (see Section 3 for more details).

Additionally, Vector Commitments need to be *updatable*. Very roughly, this means that they come equipped with two algorithms to update the commitment and the corresponding openings. The first algorithm allows the committer, who created a commitment $\texttt{Com}$ and wants to update it by changing the i-th message from m_i to m'_i, to obtain a (modified) $\texttt{Com}'$ containing the updated message. The second algorithm allows holders of an opening for a message at position j w.r.t. $\texttt{Com}$ to update their proof so as to become valid w.r.t. the new $\texttt{Com}'$.

Next, we turn our attention to the problem of realizing vector commitments. Our technical contributions are two realizations of VCs from standard and well established assumptions, namely RSA and Computational Diffie-Hellman (over bilinear groups)[3].

Finally, we confirm the power of this new primitive by showing several applications (see below) in which our notion of Vector Commitment allows for compact and efficient solutions, which signif-

[3] Precisely, our construction relies on the Square Computational Diffie-Hellman assumption (see Assumption 3 in Section 2.1) which however has been shown equivalent to the standard CDH [23, 1].

icantly improve previous works either in terms of efficiency of the resulting solutions, or in terms of "quality" of the underlying assumption, or both.

Verifiable Databases with Efficient Updates. Very recently, Benabbas, Gennaro and Vahlis [4] formalized the notion of Verifiable Databases with Efficient Updates (VDB, for short). This primitive turns out to be extremely useful to solve the following problem in the context of verifiable outsourcing of storage. Assume that a client with limited resources wants to store a large database on a server so that it can later retrieve a database record, and update a record by assigning a new value to it. For efficiency, it is crucial that the computational resources invested by the client to perform such operations must not depend on the size of the database (except for an initial pre-processing phase). On the other hand, for security, the server should not be able to tamper with any record of the database without being detected by the client.

For the static case (i.e., the client does not perform any update) simple solutions can be achieved by using message authentication or signature schemes. For example, the client first signs each database record before sending it to the server, and then the server is requested to output the record together with its valid signature. However, this idea does not work well if the client performs updates on the database. The problem is that the client should have a mechanism to revoke the signatures given to the server for the previous values. To solve this issue, the client could keep track of every change locally, but this is in contrast with the main goal, i.e., using less resources than those needed to store the database locally.

Solutions to this problem have been addressed by works on accumulators [27, 7, 8], authenticated data structures [26, 22, 28, 32], and the recent work on verifiable computation [4]. Also, other recent works have addressed a slightly different and more practical problem of realizing authenticated remote file systems [31]. However, as pointed out in [4], previous solutions based on accumulators and authenticated data structures either rely on non-constant size assumptions (such as q-Strong Diffie-Hellman), or they require expensive operations such as generation of prime numbers, and re-shuffling procedures. Benabbas *et al.* propose a nice solution with efficient query and update time [4]. Their scheme relies on a constant size assumption in bilinear groups of composite order, but does not support public verifiability (i.e., only the client owner of the database can verify the correctness of the proofs provided by the server).

In this work, we show that Vector Commitments can be used to build Verifiable Databases with efficient updates that allow for public verifiability. More importantly, if we instantiate this construction with our VC based on CDH, then we obtain an implementation of Verifiable Databases that relies on a standard constant-size assumption, and whose efficiency improves over the scheme of Benabbas *et al.* as we can use bilinear groups of *prime order*.

Updatable Zero Knowledge Elementary Databases. Zero Knowledge Sets allow a party P, called the *prover*, to commit to a secret set S in a way such that he can later produce proofs for statements of the form $x \in S$ or $x \notin S$. The required properties are the following. First, any user V (the *verifier*) should be able to check the validity of the received proofs without learning any information on S (not even its size) beyond the mere membership (or non-membership) of the queried elements. Second, the produced proofs should be reliable in the sense that no dishonest prover should be able to convince V of the validity of a false statement. Zero Knowledge Sets (ZKS) were introduced and constructed by Micali, Rabin and Kilian [24][4]. Micali *et al.*'s construction was

[4] More precisely, Micali *et al.* addressed the problem for the more general case of elementary databases (EDB), where each key x has associated a value $D(x)$ in the committed database. In the rest of this paper we will slightly abuse the notation and use the two acronyms ZKS and ZK-EDB interchangeably to indicate the same primitive.

abstracted away by Chase *et al.* [12], and by Catalano, Dodis and Visconti [9]. The former showed that ZKS can be built from trapdoor mercurial commitments and collision resistant hash functions, and also that ZKS imply collision-resistant hash functions. The latter showed generic constructions of (trapdoor) mercurial commitments from the sole assumptions that one-way functions exist. These results taken together [12, 9], thus, show that collision-resistant hash functions are necessary and sufficient to build ZKS in the CRS model. From a practical perspective, however, none of the above solutions can be considered efficient enough to be used in practice. A reason is that all of them allow to commit to a set $S \subset \{0,1\}^k$ by constructing a Merkle tree of depth k, where each internal node is filled with a mercurial commitment (rather than the hash) of its two children. A proof that $x \in \{0,1\}^k$ is in the committed set consists of the openings of all the commitments in the path from the root to the leaf labeled by x (more details about this construction can be found in [24, 12]). This implies that proofs have size linear in the height k of the tree. Now, since 2^k is an upper bound for $|S|$, to guarantee that no information about $|S|$ is revealed, k has to be chosen so that 2^k is much larger than any reasonable set size.

Catalano, Fiore and Messina addressed in [10] the problem of building ZKS with shorter proofs. Their proposed idea was a construction that uses q-ary trees, instead of binary ones, and suggested an extension of mercurial commitment (that they called q-Trapdoor Mercurial Commitment) which allows to implement it. The drawback of the specific realization of qTMC in [10] is that it is not as efficient as one might want. In particular, while the size of soft openings is independent of q, hard openings grow linearly in q. This results in an "unbalanced" ZK-EDB construction where proofs of membership are much longer than proofs of non membership.

In a follow-up work, Libert and Yung [20] proposed a very elegant solution to this problem. Specifically, they managed to construct a q-mercurial commitment (that they called *concise*) achieving constant-size (soft and hard) openings. This resulted in ZK-EDB with very short proofs, as by increasing q one can get an arbitrarily "flat" tree[5]. Similarly to [10], the scheme of Libert and Yung [20] also relies on a non-constant size assumption in bilinear groups: the q-Diffie-Hellman Exponent [6].

Our main application of VCs to ZKS is the proof of the following theorem:

Theorem 1 (informal) *A (concise) trapdoor q-mercurial commitment can be obtained from a vector commitment and a trapdoor mercurial commitment.*

The power of this theorem comes from the fact that, by applying the generic transform of Catalano *et al.* [10], we can immediately conclude that Compact ZKS (i.e. ZKS with short membership and non-membership proofs) can be built from mercurial commitments and vector commitments.

Therefore, when combining our realizations of Vector Commitments with well known (trapdoor) mercurial ones (such as that of Gennaro and Micali [14] for the RSA case, or that from [24], for the CDH construction) we get concise qTMCs from RSA and CDH. Moreover, when instantiating the ZK-EDB construction of Catalano *et al.* [10] with such schemes, one gets the first compact ZK-EDB realizations which are provably secure under standard assumptions.

Our CDH realization induces proofs whose length is comparable to that induced by Libert and Yung's commitment [20], while relying on more standard and better established assumptions.

Additionally, and more importantly, we show the first construction of *updatable* ZK-EDB with short proofs. The notion of Updatable Zero Knowledge EDB was introduced by Liskov [21] to extend ZK-EDB to the (very natural) case of "dynamic" databases. In an updatable ZK-EDB the prover is

[5] The only limitation is that the resulting CRS grows linearly in q.

allowed to change the value of some element x in the database and then output a new commitment C' and some update information U. Users holding a proof π_y for a $y \neq x$ valid w.r.t. C, should be able to use U to produce an updated proof π'_y that is valid w.r.t. C'. In [21] is given a definition of *Updatable Zero Knowledge (Elementary) Databases* together with a construction based on mercurial commitments and Verifiable Random Functions [25] in the random oracle model. More precisely, Liskov introduced the notion of *updatable mercurial commitment* and proposed a construction, based on discrete logarithm, which is a variant of the mercurial commitment of Micali *et al.* [24].

Using Vector Commitments, we realize the *first* constructions of "compact" Updatable ZK-EDB whose proofs and updates are much shorter than those of Liskov [21]. In particular, we show how to use VCs to build Updatable ZK-EDB from updatable qTMCs (which we also define and construct) and Verifiable Random Functions in the random oracle model. We stress that our solutions, in addition to solving the open problem of realizing Updatable ZK-EDB with short proofs, further improve on previous work as they allow for much shorter updates as well[6].

In [14] Gennaro and Micali addressed the problem of extending the security of zero knowledge databases to resist against adversaries that may correlate existing database commitments and proofs to create fake commitments or proofs. They introduced the notion of *Independent Zero Knowledge Databases* and proposed several realizations under different assumptions. As an additional contribution, in Appendix 5.4 we show that, using the same approach, Vector Commitments can be used also for realizing *compact* constructions of Independent Zero-Knowledge Databases.

Fully Dynamic Universal Accumulators. Cryptographic Accumulators were first introduced by Benaloh and De Mare [5], and allow to produce a compact representation of a set of values, the *accumulator*, in such a way that one can later give evidence that a given element is contained in the accumulator. The security notion for this primitive requires that users different from the one who generated the accumulator should not be able to produce false proofs. The notion has been later extended to capture the *dynamic* case in which the accumulated set can evolve in time, and it is possible to update both the accumulator and previously issued proofs [8]. In particular, performing such updates should be more efficient than recomputing these values from scratch. Another extension of cryptographic accumulators are Universal Accumulators [18]. In addition to proving that a given value is in the accumulator, these allow to issue proofs of non-membership, i.e., giving evidence that a given element is not in the accumulator.

It is worth noting that all known constructions of cryptographic accumulators (dynamic and universal) secure without random oracles either work over composite order groups and use expensive operations for mapping set elements to prime numbers [2, 8], or they work in prime order groups but rely on relatively young non-constant size assumptions, such as q-Strong Diffie Hellman or q-Diffie Hellman Exponent [18, 27, 7]. Moreover, a drawback of the schemes in prime order groups is that they support only sets of bounded polynomial size. However, as argued in [7], this restriction can be fairly reasonable in practice. More importantly, this drawback is justified by a great efficiency as these schemes only need to do group multiplications for performing the most critical operations (e.g., creating the accumulator). [7]

[6] This is because, in all known constructions, the size of the update information linearly depends on the height of the tree.

[7] There are also constructions of accumulators based on Merkle tree, which rely only on collision-resistant hash functions. However, in such schemes the size of the proofs logarithmically depends on the size of the set, and do not allow for efficient updates. In this paper we are interested in schemes allowing for constant-size proofs and efficient updates.

Additional Applications of (Hiding) Vector Commitments. If one is willing to consider Vector Commitments which also hide the committed vector, additional applications are possible. Here we briefly describe the case of *pseudonymous credentials*, an application already identified by Kate *et al.* [17] when introducing polynomial commitments. Assume a user owns a set of credentials $(m_1, \ldots, m_q)$, each corresponding to a position i, and he wants to prove that his i-th credential is m_i (without revealing m_j, $\forall j \neq i$). The credentials are certified by a trusted entity who signs them. However, if the user makes a commitment C to $(m_1, \ldots, m_q)$ and the trusted entity signs C, then the user's proof can be just the signature on C and the opening of C to m_i at position i. If the latter proof has constant size, then the communication cost of the verification protocol is constant too. Kate *et al.* show how to solve this problem using polynomial commitment schemes. However, it is easy to see that our notion of concise VC fits this application as well. Moreover, our concrete schemes allow for two new solutions of this problem relying, respectively, on RSA and CDH (rather than on Strong DH in bilinear groups as in [17]).

1.2 Road map

The paper is organized as follows. Section 2 contains some basic notations and definitions. In Section 3 we introduce the notion of Vector Commitments and we describe our concrete schemes. In Sections 4, 5 and 6 we describe applications of Vector Commitments, namely the ones of Verifiable Databases, Updatable Zero-Knowledge Databases, and Universal Dynamic Accumulators respectively. Finally, we postpone to Appendix A the formal definitions of ZK-EDBs and the generic construction of Catalano et al. [11] while in Appendix B we discuss a result on stronger security properties of updatable mercurial commitments which is of independent interest.

2 Preliminaries

In what follows we will denote with $k \in \mathbb{N}$ the security parameter, and by $poly(k)$ any function which bounded by a polynomial in k. An algorithm $\mathcal{A}$ is said to be PPT if it is modeled as a probabilistic Turing machine that runs in time polynomial in k. Informally, we say that a function is *negligible* if it vanishes faster than the inverse of any polynomial. If S is a set, then $x \xleftarrow{\$} S$ indicates the process of selecting x uniformly at random over S (which in particular assumes that S can be sampled efficiently). If n is an integer, we denote with $[n]$, the set containing the integers $1, 2, \ldots, n$.

2.1 Computational assumptions

An integer N is called *RSA modulus* if it is the product of two distinct prime numbers p, q. Given a public RSA modulus N, a public exponent e, such that $\gcd(e, \phi(N)) = 1$, and a random value $z \in \mathbb{Z}_N$, the RSA problem asks to compute the unique $y \in \mathbb{Z}_N$ such that $z = y^e \bmod N$. The public exponent can be chosen according to various distributions, in particular different distributions give rise to different variant of the problem. In this paper we consider the RSA problem where e is chosen as a random $(\ell + 1)$-bit prime (for some parameter ℓ). More formally, the corresponding RSA assumption states the following.

Assumption 2 [RSA] Let $k \in \mathbb{N}$ be the security parameter, N a random RSA modulus of length k, z be a random element in $\mathbb{Z}_N$ and e be an $(\ell + 1)$-bit prime (for a parameter ℓ).Then we say

that the RSA assumption holds if for any PPT adversary $\mathcal{A}$ the probability

$$Pr[y \leftarrow \mathcal{A}(N, e, z) : y^e = z \bmod N]$$

is negligible in k. ∎

Assumption 3 [Square-CDH] Let $k \in \mathbb{N}$ be the security parameter. Let $\mathbb{G}$ be a group of prime order p, $g \in \mathbb{G}$ be a generator and $a \xleftarrow{\$} \mathbb{Z}_p$. We say that the Square Computational Diffie-Hellman Assumption holds in $\mathbb{G}$ if for every PPT algorithm $\mathcal{A}$, the probability $\Pr[\mathcal{A}(g, g^a) = g^{a^2}]$ is at most a negligible function in k. ∎

In [23, 1] is shown that the Square-CDH assumption is equivalent to the classical Computational Diffie-Hellman (CDH) assumption.

3 Vector Commitments

In this section we introduce the notion of *Vector Commitment*. Informally speaking, a vector commitment allows to commit to an ordered sequence of values in such a way that it is later possible to open the commitment only w.r.t. a specific position. We define Vector Commitments as a non-interactive primitive, that can be formally described via the following algorithms:

$\mathsf{VC.KeyGen}(1^k, q)$ Given the security parameter k and the size q of the committed vector (with $q = poly(k)$), the key generation outputs some public parameters pp (which implicitly define the message space $\mathcal{M}$).

$\mathsf{VC.Com}_{\mathsf{pp}}(m_1, \ldots, m_q)$ On input a sequence of q messages $m_1, \ldots, m_q \in \mathcal{M}$ and the public parameters pp, the committing algorithm outputs a commitment string C and an auxiliary information $\mathtt{aux}$.

$\mathsf{VC.Open}_{\mathsf{pp}}(m, i, \mathtt{aux})$ This algorithm is run by the committer to produce a proof Λ_i that m is the i-th committed message. In particular, notice that in the case when some updates have occurred the auxiliary information $\mathtt{aux}$ can include the update information produced by these updates.

$\mathsf{VC.Ver}_{\mathsf{pp}}(C, m, i, \Lambda_i)$ The verification algorithm accepts (i.e., it outputs 1) only if Λ_i is a valid proof that C was created to a sequence $m_1, \ldots, m_q$ such that $m = m_i$.

$\mathsf{VC.Update}_{\mathsf{pp}}(C, m, m', i)$ This algorithm is run by the committer who produced C and wants to update it by changing the i-th message to m'. The algorithm takes as input the old message m, the new message m' and the position i. It outputs a new commitment C' together with an update information U.

$\mathsf{VC.ProofUpdate}_{\mathsf{pp}}(C, \Lambda_j, m', i, U)$ This algorithm can be run by any user who holds a proof Λ_j for some message at position j w.r.t. C, and it allows the user to compute an updated proof Λ'_j (and the updated commitment C') such that Λ'_j will be valid w.r.t. C' which contains m' as the new message at position i. Basically, the value U contains the update information which is needed to compute such values.

For correctness, we require that $\forall k \in \mathbb{N}, q = poly(k)$, for all honestly generated parameters $\mathsf{pp} \xleftarrow{\$} \mathsf{VC.KeyGen}(1^k, q)$, if C is a commitment on a vector $(m_1, \ldots, m_q) \in \mathcal{M}^q$ (obtained by running $\mathsf{VC.Com}_{\mathsf{pp}}$ possibly followed by a sequence of updates), Λ_i is a proof for position i generated by $\mathsf{VC.Open}_{\mathsf{pp}}$ or $\mathsf{VC.ProofUpdate}_{\mathsf{pp}}$ ($\forall i = 1, \ldots, q$), then $\mathsf{VC.Ver}_{\mathsf{pp}}(C, m_i, i, \mathsf{VC.Open}_{\mathsf{pp}}(m_i, i, \mathtt{aux}))$ outputs 1 with overwhelming probability.

The attractive feature of vector commitments is that they are required to meet a very simple security requirement, that we call *position binding*. Informally, this says that it should be infeasible, for any polynomially bounded adversary having knowledge of pp, to come up with a commitment C and two different valid openings for the same position i. More formally:

Definition 4 [Position Binding] A vector commitment satisfies position binding if $\forall i = 1, \ldots, q$ and for every PPT adversary $\mathcal{A}$ the following probability (which is taken over all honestly generated parameters) is at most negligible in k:

$$Pr\left[\begin{array}{l}\mathsf{VC.Ver}_{\mathsf{pp}}(C, m, i, \Lambda) = 1 \wedge \\ \mathsf{VC.Ver}_{\mathsf{pp}}(C, m', i, \Lambda') = 1 \wedge m \neq m'\end{array} \mid (C, m, m', i, \Lambda, \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})\right]$$

Moreover, we require a vector commitment to be *concise* in the sense that the size of the commitment string C and the outputs of $\mathsf{VC.Open}$ are both independent of q.

Vector commitments can also be required to be *hiding*. Informally, a vector commitment is hiding if an adversary cannot distinguish whether a commitment was created to a sequence $(m_1, \ldots, m_q)$ or to $(m'_1, \ldots, m'_q)$, even after seeing some openings (at positions i where the two sequences agree). We observe, however, that hiding is not a critical property in the realization of vector commitments. Indeed, any construction of vector commitments which does not satisfy hiding, can be easily fixed by composing it with a standard commitment scheme, i.e., first commit to each message separately using a standard commitment scheme, and then apply the VC to the obtained sequence of commitments. Moreover, neither the applications considered in this paper nor that considered in [19] require the underlying VC to be hiding. For these reasons, in our constructions we will only focus on the realization of the position binding property. We leave a formal definition of hiding for the full version of the paper.

3.1 A Vector Commitment based on CDH

Here we propose an implementation of concise vector commitments based on the CDH assumption in bilinear groups. Precisely, the security of the scheme reduces to the Square Computational Diffie-Hellman assumption(see Definition 3 in Section 2.1), which has been shown equivalent to the standard CDH assumption [23, 1]. Our construction is reminiscent of the incremental hash function by Bellare and Micciancio [3], even if we develop new techniques for creating our proofs that open the commitment at a specific position.

$\mathsf{VC.KeyGen}(1^k, q)$ Let $\mathbb{G}, \mathbb{G}_T$ be two bilinear groups of prime order p equipped with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Let $g \in \mathbb{G}$ be a random generator. Randomly choose $z_1, \ldots, z_q \xleftarrow{\$} \mathbb{Z}_p$. For all $i = 1, \ldots, q$ set: $h_i = g^{z_i}$. For all $i, j = 1, \ldots, q$, $i \neq j$ set $h_{i,j} = g^{z_i z_j}$.

Set $\mathsf{pp} = (g, \{h_i\}_{i \in [q]}, \{h_{i,j}\}_{i,j \in [q], i \neq j})$. The message space is $\mathcal{M} = \mathbb{Z}_p$. [8]

$\mathsf{VC.Com}_{\mathsf{pp}}(m_1, \ldots, m_q)$ Compute

$$C = h_1^{m_1} h_2^{m_2} \cdots h_q^{m_q}$$

and output C and the auxiliary information $\mathtt{aux} = (m_1, \ldots, m_q)$.

[8] The scheme can be easily extended to support arbitrary messages in $\{0,1\}^*$ by using a collision-resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_p$.

$\mathsf{VC.Open}_{\mathsf{pp}}(m_i, i, \mathtt{aux})$ Compute

$$\Lambda_i = \prod_{j=1, j\neq i}^{q} h_{i,j}^{m_j} = \left(\prod_{j=1, j\neq i}^{q} h_j^{m_j} \right)^{z_i}$$

$\mathsf{VC.Ver}_{\mathsf{pp}}(C, m_i, i, \Lambda_i)$ If the following equation holds

$$e(C/h_i^{m_i}, h_i) = e(\Lambda_i, g)$$

then output 1. Otherwise output 0.

$\mathsf{VC.Update}_{\mathsf{pp}}(C, m, m', i)$ Compute the updated commitment $C' = C \cdot h_i^{m'-m}$. Finally output C' and $U = (m, m', i)$.

$\mathsf{VC.ProofUpdate}_{\mathsf{pp}}(C, \Lambda_j, m', U)$ A client who owns a proof Λ_j, that is valid w.r.t. to C for some message at position j, can use the update information $U = (m, m', i)$ to compute the updated commitment C' and produce a new proof Λ'_j which will be valid w.r.t. C'. We distinguish two cases:

1. $i \neq j$. Compute the updated commitment $C' = C \cdot h_i^{m'-m}$ while the updated proof is $\Lambda'_j = \Lambda_j \cdot (h_i^{m'-m})^{z_j} = \Lambda_j \cdot h_{j,i}^{m'-m}$.
2. $i = j$. Compute the updated commitment as $C' = C \cdot h_i^{m'-m}$ while the updated proof remains the same Λ_i.

The correctness of the scheme can be easily verified by inspection. We prove its security via the following theorem

Theorem 5 *If the CDH assumption holds, then the scheme defined above is a concise vector commitment.*

Proof. We prove the theorem by showing that the scheme satisfies the position binding property. For sake of contradiction assume that there exists an efficient adversary $\mathcal{A}$ who produces two valid openings to two different messages at the same position, then we show how to build an efficient algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the Square Computational Diffie-Hellman assumption.

$\mathcal{B}$ takes as input a tuple (g, g^a) and its goal is to compute g^{a^2}.

First, $\mathcal{B}$ selects a random $i \xleftarrow{\$} [q]$ as a guess for the index i on which $\mathcal{A}$ will break the position binding. Next, $\mathcal{B}$ chooses $z_j \xleftarrow{\$} \mathbb{Z}_p$, $\forall j \in [q], j \neq i$, and it computes:

$$\forall j \in [q] \setminus \{i\} : h_j = (g^{z_j}), h_{i,j} = (g^a)^{z_j}, \quad h_i = g^a$$
$$\forall k, j \in [q] \setminus \{i\} : k \neq j : h_{k,j} = g^{z_k z_j},$$

$\mathcal{B}$ sets $\mathsf{pp} = (g, \{h_i\}_{i\in[q]}, \{h_{i,j}\}_{i,j\in[q], i\neq j})$ and runs $\mathcal{A}(\mathsf{pp})$. Notice that the public parameters are perfectly distributed as the real ones. The adversary is supposed to output a tuple $(C, m, m', j, \Lambda_j, \Lambda'_j)$ such that: $m \neq m'$ and both Λ_j and Λ'_j correctly verify at position j.

If $i \neq j$, then $\mathcal{B}$ aborts the simulation. Otherwise it computes

$$g^{a^2} = (\Lambda_i / \Lambda'_i)^{(m_i - m'_i)^{-1}}.$$

To see that the output is correct, observe that since the two openings verify correctly, then it holds:

$$e(C, h_i) = e(h_i^m, h_i) e(\Lambda_i, g) = e(h_i^{m'}, h_i) e(\Lambda'_i, g)$$

which means that

$$e(h_i, h_i)^{m-m'} = e(\Lambda'_i/\Lambda_i, g)$$

Since $h_i = g^a$, one can easily sees that this justifies the correctness of $\mathcal{B}$'s output.

Notice that if $\mathcal{A}$ succeeds with probability ϵ, then $\mathcal{B}$ has probability ϵ/q of breaking the Square-CDH assumption.

Efficiency and Optimizations. A drawback of our scheme is that the size of the public parameters pp is $O(q^2)$. This can be significant in those applications where the vector commitment is used with large datasets (e.g., the applications proposed in sections 4 and 6). However, we first notice that the verifier does not need the elements $h_{i,j}$. Furthermore, our construction can be easily optimized in such a way that the verifier does not need to store all the elements h_i of pp. The optimization works as follows. Who runs the setup signs each pair (i, h_i), includes the resulting signatures σ_i in the public parameters given to the committer pp, and publishes the signature's verification key. Next, the committer includes σ_i, h_i in the proof of an element at position i. This way the verifier can store only g and the verification key of the signature scheme. Later, each time it runs the verification of the vector commitment it has to check the validity of h_i by checking that σ_i is a valid signature on (i, h_i).

3.2 A Vector Commitment based on RSA

Here we propose a realization of vector commitments from the RSA assumption (whose definition is given in section 2.1).

$\mathsf{VC.KeyGen}(1^k, \ell, q)$ Randomly choose two $k/2$-bit primes p_1, p_2, set $N = p_1 p_2$, and then choose q $(\ell+1)$-bit primes $e_1, \ldots, e_q$ that do not divide $\phi(N)$. For $i = 1$ to q set

$$S_i = a^{\prod_{j=1, j\neq i}^{q} e_j}.$$

The public parameters pp are $(N, a, S_1, \ldots, S_q, e_1, \ldots, e_q)$. The message space is $\mathcal{M} = \{0,1\}^\ell$. [9]

$\mathsf{VC.Com}_{\mathsf{pp}}(m_1, \ldots, m_q)$ Compute

$$C \leftarrow S_1^{m_1} \cdots S_q^{m_q}$$

and output C and the auxiliary information $\mathsf{aux} = (m_1, \ldots, m_q)$.

$\mathsf{VC.Open}_{\mathsf{pp}}(m, i, \mathsf{aux})$, Compute

$$\Lambda_i \leftarrow \sqrt[e_i]{\prod_{j=1, j\neq i}^{q} S_j^{m_j}} \bmod N$$

Notice that knowledge of pp allows to compute Λ_i efficiently *without* the factorization of N.

$\mathsf{VC.Ver}_{\mathsf{pp}}(C, m, i, \Lambda_i)$ The verification algorithm returns 1 if $m \in \mathcal{M}$ and

$$C = S_i^m \Lambda_i^{e_i} \bmod N$$

Otherwise it returns 0.

$\mathsf{VC.Update}_{\mathsf{pp}}(C, m, m', i)$ Compute the updated commitment $C' = C \cdot S_i^{m'-m}$. Finally output C' and $U = (m, m', i)$.

[9] As in the CDH case, also this scheme can be extended to support arbitrary messages by using a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^\ell$.

$\mathsf{VC.ProofUpdate}_{\mathsf{pp}}(C, \Lambda_j, m', i, U)$ A client who owns a proof Λ_j, that is valid w.r.t. to C for some message at position j, can use the update information U to compute the updated commitment C' and to produce a new proof Λ'_j which will be valid w.r.t. C'. We distinguish two cases:

1. $i \neq j$. Compute the updated commitment as $C' = CS_i^{m'-m}$ while the updated proof is $\Lambda'_j = \Lambda_j \sqrt[e_j]{S_i^{m'-m}}$ (notice that such e_j-th root can be efficiently computed using the elements in the public key).
2. $i = j$. Compute the updated commitment $C' = C \cdot S_i^{m'-m}$ while the updated proof remains the same Λ_i.

In order for the verification process to be correct, notice that one should also verify (only once) the validity of the public key by checking that the S_i's are correctly generated with respect to a and the exponents $e_1, \ldots, e_q$.

The correctness of the scheme can be easily verified by inspection. We prove its security via the following theorem.

Theorem 6 *If the RSA assumption holds, then the scheme defined above is a concise vector commitment.*

Proof. We prove the theorem by showing that the scheme satisfies the position binding property. More precisely, assume for sake of contradiction that there exists an efficient adversary that produces two valid openings to two different messages at the same position, then we show how to build an algorithm $\mathcal{B}$ that breaks the RSA assumption.

$\mathcal{B}$ is run on input (N, z, e), where e is an $(\ell+1)$-bit prime, and it uses $\mathcal{A}$ to compute a value y such that $z = y^e \bmod N$. $\mathcal{B}$ proceeds as follows. First, it randomly chooses an index $j \in \{1, \ldots, q\}$ and sets $e_j = e$ and $a = z$. The rest of the public key is computed as required by the $\mathsf{VC.KeyGen}$ algorithm.

$\mathcal{B}$ runs $\mathcal{A}(\mathsf{pp})$ and gets back $(S, m, m', i, \Lambda_i, \Lambda'_i)$ where $m \neq m'$ and both Λ_i and Λ'_i are correctly verified. If $i \neq j$, then $\mathcal{B}$ aborts the simulation. Otherwise it proceeds as follows. From the equations $S = S_i^m \Lambda_i^{e_i}$, $S = S_i^{m'} \Lambda'^{e_i}_i$ we get

$$S_i^{m-m'} = (\Lambda'_i/\Lambda_i)^{e_i}$$

Let $\delta = m - m'$ and $\Lambda = \Lambda'_i/\Lambda_i$, the equation above can be rewritten as

$$a^{\delta \prod_{j \neq i} e_j} = (\Lambda)^{e_i}$$

If $\Lambda = 1$ then we can factor with non-negligible probability (this is because, a being a random element in $\mathbb{Z}_N$ it is of high order with very high probability, see [15] for details). Thus, assuming $\Lambda \neq 1$ we can apply the Shamir's trick [30] to get an e_i-th root of a. In particular, since $gcd(m \prod_{j \neq i} e_j, e_i) = 1$, by the extended Euclidean Algorithm we can compute two integers λ, μ such that $\lambda m \prod_{j \neq i} e_j + \mu e_i = 1$. This leads to the equation

$$a = \Lambda^{\lambda e_i} a^{\mu e_i}$$

thus $\Lambda^\lambda a^\mu$ is the required root.

Efficiency and Optimizations. This scheme suffers the same drawback as the one based on CDH given in the previous section. Namely, the size of the public parameters is linear in q. Also

this construction can be easily optimized in such a way that the verifier stores only a constant number of elements, instead of the entire public parameters. The ideas is basically the same as that given for the scheme based on CDH. The only difference is that here the signature is computed on (i, S_i, e_i).

3.3 A Vector Commitment with constant-size public parameters

A limitation of the constructions of Vector Commitments described before is that the size of the public parameters depends on q. However, this might be a limitation in several applications, such as the ones we showed. For instance, it affects the size of the common reference string in ZK EDB constructions, or it bounds the size of the sets or the databases that can be supported by our accumulators and verifiable database schemes.

In this section we show that our Vector Commitment scheme based on RSA given in section 3.2 can be modified so as to obtain a scheme where the public parameters have constant size. Unfortunately, the new scheme is computationally less efficient compared to the other constructions. Though, we give it as a proof of concept to show that realizing vector commitments with public parameters of size constant and independent of q is possible.

In order to obtain such variant we apply a technique used by Waters and Hohenberger in [16] to obtain a deterministic function that outputs primes.

We describe below the modified key generation algorithm.

$\mathsf{VC.KeyGen}(k, \ell, q)$ Randomly choose two $k/2$-bit primes p_1, p_2 and set $N = p_1 p_2$. Then randomly select a seed s for a pseudorandom function $f_s : \{0,1\}^* \to \{0,1\}^{\ell+1}$ and a binary string $c \xleftarrow{\$} \{0,1\}^{\ell+1}$ and define the function $F_{s,c}(j) = f_s(i,j) \oplus c$ where $i \geq 1$ is the smallest index such that $f_s(i,j) \oplus c$ is odd and prime. Let a be a random element in $\mathbb{Z}_N$. For $i = 1$ to q we implicitly define

$$S_i = a^{\prod_{j=1, j\neq i}^{q} e_j}$$

where $e_j = F_{s,c}(j)\ \forall j = 1, \ldots, q$. The public key vpk is set as (N, a, s, c) and the message space is $\mathcal{M} = \{0,1\}^{\ell}$.

The rest of the scheme is basically the same where the elements S_i and e_i are reconstructed using the function $F_{s,c}(\cdot)$.

The scheme above can be proved secure using the same proof of Theorem 6 extended with the techniques presented in [16] (Theorem 4.1) to show that the function $f_s(\cdot)$ is "good enough" to have a correct simulation. The reader can refer to [16] for details omitted here. This allows us to state the following theorem:

Theorem 7 *If the RSA assumption holds, and f is a pseudorandom function, then the scheme given above is a Vector Commitment.*

4 Verifiable Databases with Efficient Updates from Vector Commitments

In this section we show that vector commitments allow to build a verifiable database scheme. This notion has been formalized very recently by Benabbas, Gennaro and Vahlis [4]. Intuitively, a verifiable database allows a weak client to outsource the storage of a large database D to a server in such a way that the client can later retrieve the database records from the server and be convinced

that the records have not been tampered with. In particular, since the main application is in the context of cloud computing services for storage outsourcing, it is crucial that the resources invested by the client after transmitting the database (e.g., to retrieve and update the records) must be independent of the database's size. While a solution for the static case in which the database is not updated can be obtained using standard techniques (e.g., digital signatures), the setting in which the client can update the values of the database records need different ideas.

Here we describe a solution based on our notion of Vector Commitments. Our construction, when instantiated with our CDH-based VC, allows for an efficient scheme, yet it is based on a standard constant-size assumption such as Computation Diffie-Hellman in bilinear groups. Furthermore, our scheme allows for public verifiability, that was not supported by the scheme in [4].

We begin by recalling the definition of Verifiable Databases. Our definition closely follows that in [4] except for some changes that we introduce because we consider public verifiability. We denote a database D as a set of tuples (x, v_x) in some appropriate domain, where x is the key, and v_x is the corresponding value. We denote this by writing $D(x) = v_x$. In our case we consider keys that are integers in the interval $\{1, \ldots, q\}$, where $q = poly(k)$, whereas the DB values can be arbitrary strings $v \in \{0,1\}^*$.

A Verifiable Database scheme VDB is defined by the following algorithms:

$\mathsf{VDB.Setup}(1^k, D)$ On input the security parameter k and a database D, the setup algorithm is run by the client to generate a secret key SK that is kept private by the client, a database encoding S that is given to the server, and a public key PK that is distributed to all users (including the client itself) who wish to verify the proofs.

$\mathsf{VDB.Query}(\mathsf{PK}, S, x)$ On input a database key x, the query processing algorithm is run by the server, and returns a pair $\tau = (v, \pi)$.

$\mathsf{VDB.Verify}(\mathsf{PK}, x, \tau)$ The public verification algorithm outputs a value v if τ verifies correctly w.r.t. x (i.e., $D(x) = v$), and an error $\perp$ otherwise.

$\mathsf{VDB.ClientUpdate}(\mathsf{SK}, x, v')$ The client update algorithm is used by the client to change the value of the database record with key x, and it outputs a value t'_x and an updated public key PK'.

$\mathsf{VDB.ServerUpdate}(\mathsf{PK}, S, x, t'_x)$ The server update algorithm is run by the server to update the database according to the value t'_x produced by the client.

Before defining the notion of security we remark that a crucial requirement is that the size of the information stored by the client as well as the time needed to compute verifications and updates must be independent of the size $|D|$ of the database.

Roughly speaking, a Verifiable Database is secure if the server cannot convince users about the validity of false statements, i.e., that $D(x) = v$ where v is not the value v_x that the client wrote in the record with key x.

More formally, this notion is defined by the following experiment:

Experiment $\mathbf{Exp}^{\mathsf{VDB}}_{\mathcal{A}}(k)[D]$

$(\mathsf{PK}, \mathsf{SK}, S) \xleftarrow{\$} \mathsf{VDB.Setup}(1^k, D)$

For $i = 1$ to $T = poly(k)$

$(x_i, v^{(i)}_{x_i}) \xleftarrow{\$} \mathcal{A}(\mathsf{PK}, S, t'_1, \ldots, t'_{i-1})$

$(\mathsf{PK}_i, t'_i) \xleftarrow{\$} \mathsf{VDB.ClientUpdate}(\mathsf{SK}, x_i, v^{(i)}_{x_i})$

$(x^*, \tau^*) \xleftarrow{\$} \mathcal{A}(\mathsf{PK}, S, t'_1, \ldots, t'_T)$

$v^* \leftarrow \mathsf{VDB.Verify}(\mathsf{PK}, x^*, \tau^*)$

If $v^* \neq \perp$ and $v^* \neq v^{(T)}_{x^*}$ return 1, else return 0.

In the experiment, after every update query, we implicitly assign $\mathsf{PK} \leftarrow \mathsf{PK}_i$, and $v_x^{(i)} \leftarrow v_x^{(i-1)}$ for all the $x \neq x_i$ not involved in the update.

For any initial database D, we define the advantage of an adversary $\mathcal{A}$ in the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{VDB}}(k)[D]$ as follows:

$$\mathbf{Adv}_{\mathcal{A},D}^{\mathsf{VDB}}(k) = \Pr\left[\mathbf{Exp}_{\mathcal{A}}^{\mathsf{VDB}}(k)[D] = 1\right]$$

Definition 8 [Security] A Verifiable Database scheme VDB is secure if for any database $D \in [q] \times \{0,1\}^*$, where $q = poly(k)$, and for any PPT adversary $\mathcal{A}$, it holds:

$$\mathbf{Adv}_{\mathcal{A},D}^{\mathsf{VDB}}(k) \leq \epsilon(k)$$

where ϵ is a negligible function in k.

4.1 A Verifiable Database scheme from Vector Commitments

Now we show how to build a verifiable database scheme VDB from a vector commitment VC. The construction follows.

$\mathsf{VDB.Setup}(1^k, D)$ Let $D = \{(i, v_i)\}_{i=1}^q$. Run $\mathsf{pp} \xleftarrow{\$} \mathsf{VC.KeyGen}(1^k, q)$. Compute $(C, \mathsf{aux}) \leftarrow \mathsf{VC.Com}_{\mathsf{pp}}(v_1, \ldots, v_q)$ and set $\mathsf{PK} = (\mathsf{pp}, C)$, $S = (\mathsf{pp}, \mathsf{aux}, D)$, $\mathsf{SK} = \perp$.

$\mathsf{VDB.Query}(\mathsf{PK}, S, x)$ Let $v_x = D(x)$. Compute $\Lambda_x \leftarrow \mathsf{VC.Open}_{\mathsf{pp}}(v_x, x, \mathsf{aux})$ and return $\tau = (v_x, \Lambda_x)$.

$\mathsf{VDB.Verify}(\mathsf{PK}, x, \tau)$ Parse τ as (v_x, Λ_x). If $\mathsf{VC.Ver}_{\mathsf{pp}}(C, x, v_x, \Lambda_x) = 1$, then return v_x. Otherwise return $\perp$.

$\mathsf{VDB.ClientUpdate}(\mathsf{SK}, x, v'_x)$ To update the record with key x, the client first retrieves the record x from the server (i.e., it asks the server for $\tau \leftarrow \mathsf{VDB.Query}(\mathsf{PK}, S, x)$ and checks that $\mathsf{VDB.Verify}(\mathsf{PK}, x, \tau) = v_x \neq \perp$). Then, it computes $(C', U) \xleftarrow{\$} \mathsf{VC.Update}_{\mathsf{pp}}(C, v_x, v'_x, x)$ and outputs $\mathsf{PK}' = (\mathsf{pp}, C')$ and $t'_x = (\mathsf{PK}', v'_x, U)$.

$\mathsf{VDB.ServerUpdate}(\mathsf{pk}, S, x, t'_x)$ Let $t'_x = (\mathsf{PK}', v'_x, U)$. The server writes v'_x in the database record with key x and adds the update information U to aux in S.

Theorem 9 *If* VC *is a vector commitment, then the Verifiable Database scheme described above is secure.*

Proof. As usual, we proceed by contradiction. Assume there exists an adversary $\mathcal{A}$ that has non-negligible advantage ϵ in the experiment $\mathbf{Exp}_{\mathcal{A}}^{\mathsf{VDB}}(k)[D]$ for some initial database D. Then we show that such adversary can be used to build an efficient algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the position binding of the vector commitment.

$\mathcal{B}$ takes as input the public parameters of the vector commitment pp. Given the database D, $\mathcal{B}$ computes a commitment $(C, \mathsf{aux}) \leftarrow \mathsf{VC.Com}_{\mathsf{pp}}(\{v_x\}_{x \in D})$ on the database. Next, it sets $\mathsf{PK} = (\mathsf{pp}, C)$ and $S = (\mathsf{pp}, \mathsf{aux}, D)$ and runs $\mathcal{A}(\mathsf{PK}, S)$. It is easy to see that PK and S are distributed exactly as in the real construction. To answer the update queries of $\mathcal{A}$, $\mathcal{B}$ simply runs the real $\mathsf{VDB.ClientUpdate}$ algorithm. Notice that this not require any secret knowledge.

Let (x^*, τ^*) be the tuple returned by the adversary at the end of the experiment where $\tau^* = (v^*, \Lambda^*)$. If the adversary wins in breaking the security of the verifiable database scheme, then recall that it must hold $\mathsf{VDB.Verify}(\mathsf{PK}, x^*, \tau^*) = v^*$, $v^* \neq \perp$ and $v^* \neq v_{x^*}^{(T)}$, where $v_{x^*}^{(T)}$ is the database value with key x^* correctly computed after the T update queries. $\mathcal{B}$ computes the honest proof

$\Lambda_{x^*} \leftarrow \mathsf{VC.Open}(v_{x^*}^{(T)}, x^*, \mathsf{aux})$ (which correctly verifies by the completeness of the scheme), and it outputs $(C, x^*, v_{x^*}^{(T)}, v^*, \Lambda_{x^*}, \Lambda^*)$. Since $\mathcal{A}$ breaks the security of the verifiable database, Λ^* must verify correctly for v^* (i.e., $\mathsf{VC.Ver}_{\mathsf{pp}}(C, x^*, v^*, \Lambda^*) = 1$). Therefore, one can see that the tuple $(C, x^*, v_{x^*}^{(T)}, v^*, \Lambda_{x^*}, \Lambda^*)$ breaks the position binding of the vector commitment. This completes the proof.

Supporting Private Verifiability. Our construction allows any user holding the public key PK to verify the proofs produced by the server. Although this is an additional property with respect to some previous works (i.e., [4]), in some applications one may need to enable only the database owner, i.e., the client, to verify the proofs produced by the server. Here we notice that our scheme can be easily modified in order to support *private* verifiability. The idea is to use a pseudorandom function $f_s(\cdot)$ and to compute the vector commitment on the values $y_x = f_s(v_x)$ instead of v_x. The client keeps the seed s as part of its secret key and it uses s to recompute $f_s(\cdot)$ in the (private) verification algorithm. It is not hard to see that such modified scheme is secure with private verification.

Furthermore, our scheme can also be modified to work in a hybrid case when the client would like to decide from time to time whether to allow public verification of specific records. The idea is to use a verifiable random function [25] instead of a PRF. Similarly to the private case, the vector commitment is computed on the outputs of the VRF (computed on each database value v_x). Next, when the client wants to convince a third party about the validity of a proof for a record x, it can release the VRF proof for v_x. If the VRF proofs are not revealed, then the scheme preserves private verification. This follows immediately from the security of the VRF. Indeed, as long as the proofs are not revealed, the outputs of the function look random, i.e., the function behaves like a PRF.

A note on the size of the public key. If one looks at the concrete Verifiable Database scheme resulting by instantiating the vector commitment with one of our constructions in sections 3.1 and 3.2 a problem arises. In VDBs the public key must have size independent of the DB size, but this happens not to be the case in our CDH and RSA constructions where the public parameters pp depend on q. To solve this issue we thus require this transform to use vector commitments with constant size parameters. Concretely, we can use the variant of our RSA construction that has this property, or, for a better efficiency, our CDH/RSA constructions in Section 3 with the respective optimizations that enable the verifier to store only a constant number of elements of pp. A detailed description of this optimization was given in the previous section.

5 (Updatable) Zero-Knowledge Elementary Databases from Vector Commitments

In this section we show that Vector Commitments can be used to build Zero-Knowledge Elementary Databases (ZK-EDBs). In particular, following the approach of Catalano, Fiore and Messina [10], we can solve the open problem of building compact ZK-EDBs based on standard constant-size assumptions. Furthermore, in the next section we will show that the same approach can be extended to build *Updatable* ZK-EDBs, thus allowing for the first compact construction of this primitive. Since the updatable case is more interesting in practice, we believe that this can be a significant improvement.

Zero-Knowledge Elementary Databases. We first recall the notion of Zero-Knowledge Elementary Databases. Let D be a database and $[D]$ be the set of all the keys in D. We assume that

$[D]$ is a proper subset of $\{0,1\}^*$. If $x \in [D]$, we denote with $y = D(x)$ its associated value in the database D. If $x \notin [D]$ we let $D(x) = \perp$. A Zero Knowledge (Elementary) Database system is formally defined by a tuple of algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{V})$ that work as follows:

- $\mathsf{Setup}(1^k)$ takes as input the security parameter k and generates a common reference string CRS.
- $\mathsf{Commit}(CRS, D)$, the *committer* algorithm, takes as input a database D and the common reference string CRS and outputs a public key ZPK and a secret key ZSK.
- $\mathsf{Prove}(CRS, ZSK, x)$ On input the common reference string CRS, the secret key ZSK and an element x, the *prover* algorithm produces a proof π_x of either $D(x) = y$ or $D(x) = \perp$.
- $\mathsf{V}(CRS, ZPK, x, \pi_x)$ The *verifier* algorithm outputs y if $D(x) = y$, *out* if $D(x) = \perp$, and $\perp$ if the proof π_x is not valid.

We say that such a scheme is a Zero-Knowledge Elementary Database if it satisfies *completeness*, *soundness* and *zero-knowledge*. The exact meaning of such requirements is given in appendix A. Here we only explain them informally. In a nutshell, *completeness* requires that proofs generated by honest provers are correctly verified; *soundness* imposes that a dishonest prover cannot prove false statements about elements of the database; *zero-knowledge* guarantees that proofs do not reveal any information on the database (beyond their validity).

Towards Building Zero-Knowledge Elementary Databases. Chase *et al.* showed a general construction of ZK-EDB from a new primitive, that they called trapdoor mercurial commitment, and collision-resistant hash functions [12]. At a very high level, the idea of the construction is to build a Merkle tree in which each node is the mercurial commitment (instead of a hash) of its two children. This construction has been later generalized by Catalano *et al.* so as to work with q-ary trees instead of binary ones [10, 11] in order to obtain more efficient schemes. This required the introduction of a new primitive called trapdoor q-mercurial commitments (qTMC), and it basically shows that the task of building ZK-EDBs can be reduced to that of building qTMCs. Therefore, in what follows we simply show how to build qTMCs using vector commitments. Then one can apply the generic methodology of Catalano *et al.* (that is recalled in Appendix A.1) to obtain compact ZK-EDBs. We stress that in the construction of Catalano *et al.* the value q is the branching factor of the tree and is *not* related to the size of the database. Thus, even if vector commitments reveal q in the clear, this does not compromise the security of ZK-EDBs.

5.1 Useful definitions of mercurial commitments

Trapdoor mercurial commitments A *trapdoor mercurial commitment scheme* is defined by the following set of algorithms (the definitions given in this section are taken from [10]):

$\mathsf{KeyGen}(1^k)$ is a probabilistic algorithm that takes in input a security parameter k and outputs a pair of public/private keys (pk, tk).

$\mathsf{HCom}_{pk}(m)$ Given a message m, this algorithm computes a hard commitment C to m using the public key pk and returns some auxiliary information aux.

$\mathsf{HOpen}_{pk}(m, \mathsf{aux})$ The hard opening algorithm produces a hard decommitment π to the message m correlated to $(C, \mathsf{aux}) = \mathsf{HCom}_{pk}(m)$.

$\mathsf{HVer}_{pk}(m, C, \pi)$ The hard verification algorithm $\mathsf{HVer}_{pk}(m, C, \pi)$ accepts (outputs 1) only if π proves that C is a hard commitment to m.

$\mathsf{SCom}_{pk}()$ produces a soft commitment C and an auxiliary information $\mathtt{aux}$. We observe that a soft commitment string C is created to no message in particular.

$\mathsf{SOpen}_{pk}(m, \mathsf{flag}, \mathtt{aux})$ produces a soft decommitment τ (also known as *"tease"*) to a message m. The parameter $\mathsf{flag} \in \{\mathbb{H}, \mathbb{S}\}$ points out if τ corresponds to a hard commitment $(C, \mathtt{aux}) = \mathsf{HCom}_{pk}(m)$ or to a soft commitment $(C, \mathtt{aux}) = \mathsf{SCom}_{pk}()$. A soft decommitment τ to m says that "if the commitment C produced together with $\mathtt{aux}$ can be opened at all, then it would open to m".

$\mathsf{SVer}_{pk}(m, C, \tau)$ checks if τ is a valid decommitment for C to m. If it outputs 1 and τ corresponds to a hard commitment C to m, then C could be hard-opened to m.

$\mathsf{Fake}_{pk,tk}()$ produces a "fake" commitment C which at the beginning is not bound to any message. It also returns an auxiliary information $\mathtt{aux}$.

$\mathsf{HEquiv}_{pk,tk}(m, \mathtt{aux})$ The hard equivocation algorithm generates a hard decommitment π for $(C, \mathtt{aux}) = \mathsf{Fake}_{pk,tk}()$ to the message m. A fake commitment is quite similar to a soft commitment with the additional property that it can be hard-opened.

$\mathsf{SEquiv}_{pk,tk}(m, \mathtt{aux})$ generates a soft decommitment τ to m using the auxiliary information produced by the Fake algorithm.

To satisfy the correctness property we require that $\forall m \in \mathcal{M}$ the following statements are false only with negligible probability:

1. if $(C, \mathtt{aux}) = \mathsf{HCom}_{pk}(m)$:

$$\mathsf{HVer}_{pk}(m, C, \mathsf{HOpen}_{pk}(m, \mathtt{aux})) = 1$$

$$\mathsf{SVer}_{pk}(m, C, \mathsf{SOpen}_{pk}(m, \mathbb{H}, \mathtt{aux})) = 1$$

2. If $(C, \mathtt{aux}) = \mathsf{SCom}_{pk}()$:

$$\mathsf{SVer}_{pk}(m, C, \mathsf{qSOpen}_{pk}(m, \mathbb{S}, \mathtt{aux})) = 1.$$

3. If $(C, \mathtt{aux}) = \mathsf{Fake}_{pk,tk}()$:

$$\mathsf{HVer}_{pk}(m, C, \mathsf{HEquiv}_{pk,tk}(m, \mathtt{aux})) = 1$$

$$\mathsf{SVer}_{pk}(m, C, \mathsf{SEquiv}_{pk,tk}(m, \mathtt{aux})) = 1$$

Security properties. We require that a trapdoor mercurial commitment scheme satisfies the following security properties:

- **Mercurial binding**. Having knowledge of pk, it is computationally infeasible for an algorithm $\mathcal{A}$ to come up with C, m, π, m', π' such that either one of the following cases holds:
  - π is a valid hard decommitment for C to m and π' is a valid hard decommitment for C to m', with $m \neq m'$. We call such case a "hard collision".
  - π is a valid hard decommitment for C to m and π' is a valid soft decommitment for C to m', with $m \neq m'$. We call such case a "soft collision".
- **Mercurial hiding**. There exists no PPT adversary $\mathcal{A}$ that, knowing pk, can find a message $m \in \mathcal{M}$ for which it can distinguish $(C,\ \mathsf{SOpen}_{pk}(m, \mathbb{H}, \mathtt{aux}))$ from $(C', \mathsf{SOpen}_{pk}(m, \mathbb{S}, \mathtt{aux}'))$, where $(C, \mathtt{aux}) = \mathsf{HCom}_{pk}(m)$ and $(C', \mathtt{aux}') = \mathsf{SCom}_{pk}()$.

– **Equivocations**. There exists no PPT adversary $\mathcal{A}$ that, having knowledge of the public key pk and the trapdoor key tk, can win in the following games with non-negligible probability. In such games $\mathcal{A}$ must tell apart the "real" world from its corresponding "ideal" world. The challenger flips a binary coin $b \in \{0,1\}$. If $b = 0$ it gives to $\mathcal{A}$ a real commitment/decommitment tuple; if $b = 1$ it gives to $\mathcal{A}$ an ideal tuple produced using the fake algorithms.
  - **HHEquivocation.** $\mathcal{A}$ chooses $m \in \mathcal{M}$ and gives it to the challenger. If $b = 0$ the challenger gives to $\mathcal{A}$ a pair (C, π) such that: $(C, \mathsf{aux}) = \mathsf{HCom}_{pk}(m)$ and $\pi = \mathsf{HOpen}_{pk}(m, \mathsf{aux})$. Otherwise it returns (C, π) such that: $(C, \mathsf{aux}) = \mathsf{Fake}_{pk,tk}()$ and $\pi = \mathsf{HEquiv}_{pk,tk}(m, \mathsf{aux})$.
  - **HSEquivocation.** $\mathcal{A}$ chooses $m \in \mathcal{M}$ and gives it to the challenger. If $b = 0$ the challenger gives to $\mathcal{A}$ a real commitment-decommitment tuple (C, τ) such that: $(C, \mathsf{aux}) = \mathsf{HCom}_{pk}(m)$ and $\tau = \mathsf{SOpen}_{pk}(m, \mathbb{H}, \mathsf{aux})$. Otherwise the challenger produces an ideal tuple with: $(C, \mathsf{aux}) = \mathsf{Fake}_{pk,tk}()$ and $\tau = \mathsf{SEquiv}_{pk,tk}(m, \mathsf{aux})$.
  - **SSEquivocation.** If $b = 0$ the challenger generates $(C, \mathsf{aux}) = \mathsf{SCom}_{pk}()$ and gives C to $\mathcal{A}$. Then $\mathcal{A}$ chooses $m \in \mathcal{M}$, gives such m to the challenger and receives $\mathsf{SOpen}_{pk}(m, \mathbb{S}, \mathsf{aux})$. Otherwise if $b = 1$ $\mathcal{A}$ first gets $\mathsf{Fake}_{pk,tk}()$, then it chooses $m \in \mathcal{M}$, gives it to the challenger and finally receives $\mathsf{SEquiv}_{pk,tk}(m, \mathsf{aux})$.

  At some point $\mathcal{A}$ outputs b' as its guess for b and wins if $b' = b$.
  As claimed in [9] it is easy to see that the mercurial hiding is implied by the HHEquivocation and HSEquivocation.

Trapdoor q-mercurial commitments The notion of trapdoor q-mercurial commitment (qTMC) extends the notion of mercurial commitments in the sense that it allows to (mercurially) commit to (ordered) sequences of messages, rather than to single ones. Formally qTMCs are defined in terms of the following algorithms (the definitions given in this section are taken from [10]):

$\mathsf{qKeyGen}(1^k, q)$ is a probabilistic algorithm that receives as input a security parameter k, the parameter q indicating the length of valid sequences and outputs a pair of public/private keys (pk, tk).

$\mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$ Given an ordered tuple of q messages, qHCom computes a hard commitment C to $(m_1, \cdots, m_q)$ using the public key pk and returns some auxiliary information aux.

$\mathsf{qHOpen}_{pk}(m, i, \mathsf{aux})$ Let $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$, if $m = m_i$ the hard opening algorithm produces a hard decommitment π. The algorithm returns an error message otherwise.

$\mathsf{qHVer}_{pk}(m, i, C, \pi)$ The hard verification algorithm $\mathsf{qHVer}_{pk}(m, i, C, \pi)$ accepts (outputs 1) only if π proves that C is created to a sequence $(m_1, \cdots, m_q)$ such that $m_i = m$.

$\mathsf{qSCom}_{pk}()$ produces a soft commitment C and an auxiliary information aux. A soft commitment string C is created to no specific sequence of messages.

$\mathsf{qSOpen}_{pk}(m, i, \mathsf{flag}, \mathsf{aux})$ produces a soft decommitment τ (also known as *"tease"*) to a message m at position i. The parameter $\mathsf{flag} \in \{\mathbb{H}, \mathbb{S}\}$ indicates if τ corresponds to either a hard commitment $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$ or to a soft commitment $(C, \mathsf{aux}) = \mathsf{qSCom}_{pk}()$. The algorithm returns an error message if C is a hard commitment and $m \neq m_i$.

$\mathsf{qSVer}_{pk}(m, i, C, \tau)$ checks if τ is a valid decommitment for C to m of index i. If it outputs 1 and τ corresponds to a hard commitment C to $(m_1, \cdots, m_q)$, then C could be hard-opened to (m, i), or rather $m = m_i$.

$\mathsf{qFake}_{pk,tk}()$ takes as input the trapdoor tk and produces a q-fake commitment C. C is not bound to any sequence $(m_1, \cdots, m_q)$. It also returns an auxiliary information aux.

$\mathsf{qHEquiv}_{pk,tk}(m_1, \cdots, m_q, i, \mathsf{aux})$ The non-adaptive hard equivocation algorithm generates a hard decommitment π for $(C, \mathsf{aux}) = \mathsf{qFake}_{pk,tk}()$ to the i-th message of $(m_1, \cdots, m_q)$. The algorithm is non adaptive in the sense that, for a given C, the sequence $(m_1, \cdots, m_q)$ has to be determined once and for all, before $\mathsf{qHEquiv}$ is executed. A q-fake commitment is very similar to a soft commitment with the additional property that it can be hard-opened.

$\mathsf{qSEquiv}_{pk,tk}(m, i, \mathsf{aux})$ generates a soft decommitment τ to m of position i using the auxiliary information produced by the qFake algorithm.

The correctness requirements for trapdoor q-mercurial commitments are essentially the same as those for "traditional" commitment schemes. In particular we require that $\forall (m_1, \cdots, m_q) \in \mathcal{M}^q$, the following statements are false only with negligible probability.

1. if $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$: $\mathsf{qHVer}_{pk}(m_i, i, C, \mathsf{qHOpen}_{pk}(m_i, i, \mathsf{aux})) = 1 \ \ \forall i = 1 \ldots q$
2. If $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$ $\mathsf{qSVer}_{pk}(m_i, i, C, \mathsf{qSOpen}_{pk}(m_i, i, \mathbb{H}, \mathsf{aux})) = 1 \ \ \forall i = 1 \ldots q$
3. If $(C, \mathsf{aux}) = \mathsf{qSCom}_{pk}()$ $\mathsf{qSVer}_{pk}(m_i, i, C, \mathsf{qSOpen}_{pk}(m_i, i, \mathbb{S}, \mathsf{aux})) = 1 \ \ \forall i = 1 \ldots q$
4. If $(C, \mathsf{aux}) = \mathsf{qFake}_{pk,tk}()$ $\mathsf{qHVer}_{pk}(m_i, i, C, \mathsf{qHEquiv}_{pk,tk}(m_1, \cdots, m_q, i, \mathsf{aux})) = 1$ and $\mathsf{qSVer}_{pk}(m_i, i, C, \mathsf{qSEquiv}_{pk,tk}(m_i, i, \mathsf{aux})) = 1 \quad \forall i = 1 \ldots q$

Security. The security properties for a trapdoor q-mercurial commitment scheme are as follows:

- **q-Mercurial binding**. Having knowledge of pk it is computationally infeasible for an algorithm $\mathcal{A}$ to come up with C, m, i, π, m', π' such that either one of the following cases holds:
  - π is a valid hard decommitment for C to (m, i) and π' is a valid hard decommitment for C to (m', i), with $m \neq m'$. We call such case a "hard collision".
  - π is a valid hard decommitment for C to (m, i) and π' is a valid soft decommitment for C to (m', i), with $m \neq m'$. We call such case a "soft collision".
- **q-Mercurial hiding**. There exists no PPT adversary $\mathcal{A}$ that, knowing pk, can find a tuple $(m_1, \cdots, m_q) \in \mathcal{M}^q$ and an index i for which it can distinguish $(C, \mathsf{qSOpen}_{pk}(m_i, i, \mathbb{H}, \mathsf{aux}))$ from $(C', \mathsf{qSOpen}_{pk}(m_i, i, \mathbb{S}, \mathsf{aux}'))$, where $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$ and $(C', \mathsf{aux}') = \mathsf{qSCom}_{pk}()$.
- **Equivocations**. There exists no PPT adversary $\mathcal{A}$ that, knowing pk and the trapdoor tk, can win any of the following games with non-negligible probability. In such games $\mathcal{A}$ should be able to tell apart the "real" world from the corresponding "ideal" one. The games are formalized in terms of a challenger that flips a binary coin $b \in \{0, 1\}$. If $b = 0$ it gives to $\mathcal{A}$ a real commitment/decommitment tuple; if $b = 1$ it gives to $\mathcal{A}$ an ideal tuple produced using the fake algorithms.

  In the q-HHEquivocation and the q-HSEquivocation games below, $\mathcal{A}$ chooses $(m_1, \cdots, m_q) \in \mathcal{M}^q$ and receives a commitment string C. Then $\mathcal{A}$ gives an index $i \in \{1, \cdots, q\}$ to the challenger and finally it receives a hard decommitment π.
  - **q-HHEquivocation.** If $b = 0$ the challenger hands to $\mathcal{A}$ the value $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$. $\mathcal{A}$ gives i to the challenger and gets back $\pi = \mathsf{qHOpen}_{pk}(m_i, i, \mathsf{aux})$. Otherwise the challenger computes $(C, \mathsf{aux}) = \mathsf{qFake}_{pk,tk}()$, $\pi = \mathsf{qHEquiv}_{pk,tk}(m_1, \cdots, m_q, i, \mathsf{aux})$.
  - **q-HSEquivocation.** The challenger computes $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q)$, $\pi = \mathsf{qSOpen}_{pk}(m_i, i, \mathbb{H}, \mathsf{aux})$ in the case $b = 0$ or $(C, \mathsf{aux}) = \mathsf{qFake}_{pk,tk}(), \pi = \mathsf{qSEquiv}_{pk,tk}(m_i, i, \mathsf{aux})$ if $b = 1$.

- **q-SSEquivocation.** If $b = 0$ the challenger generates $(C, \mathsf{aux}) = \mathsf{qSCom}_{pk}()$ and gives C to $\mathcal{A}$. Next, $\mathcal{A}$ chooses $m \in \mathcal{M}$ and an index $i \in \{1, \cdots, q\}$, it gives (m, i) to the challenger and receives back $\mathsf{qSOpen}_{pk}(m, i, \mathbb{S}, \mathsf{aux})$. If $b = 1$, $\mathcal{A}$ first gets $\mathsf{qFake}_{pk,tk}()$, then it chooses $m \in \mathcal{M}, i \in \{1, \cdots, q\}$, gives (m, i) to the challenger and gets back $\mathsf{qSEquiv}_{pk,tk}(m, i, \mathsf{aux})$.

At some point $\mathcal{A}$ outputs b' as its guess for b and wins if $b' = b$.

As for the case of trapdoor mercurial commitments (see [9]) it is easy to see that the q-mercurial hiding is implied by the q-HSEquivocation and q-SSEquivocation.

5.2 Trapdoor q-Mercurial Commitments from Vector Commitments and Mercurial Commitments

Here we show how to combine (concise) vector commitments and standard trapdoor commitment to obtain (concise) trapdoor qTMC. For lack of space, we defer the interested reader to [11] for the definitions of (trapdoor) mercurial commitments and trapdoor q-mercurial commitments.

Let $\mathsf{TMC} = (\mathsf{KeyGen}, \mathsf{HCom}, \mathsf{HOpen}, \mathsf{HVer}, \mathsf{SCom}, \mathsf{SOpen}, \mathsf{SVer}, \mathsf{Fake}, \mathsf{HEquiv}, \mathsf{SEquiv})$ be a trapdoor mercurial commitment and $\mathsf{VC} = (\mathsf{VC.KeyGen}, \mathsf{VC.Com}, \mathsf{VC.Open}, \mathsf{VC.Ver})$ be a vector commitment. We construct a trapdoor qTMC as follows:

$\mathsf{qKeyGen}(1^k)$ Run $\mathsf{pp} \xleftarrow{\$} \mathsf{VC.KeyGen}(1^k, q)$ and $(PK_{\mathsf{TMC}}, TK_{\mathsf{TMC}}) \xleftarrow{\$} \mathsf{KeyGen}(1^k)$ and set $pk = (\mathsf{pp}, PK_{\mathsf{TMC}})$ and $tk = TK_{\mathsf{TMC}}$.

$\mathsf{qHCom}_{pk}(m_1, \ldots, m_q)$ For $i = 1$ to q compute $(C_i, \mathsf{aux}^i_{\mathsf{TMC}}) \xleftarrow{\$} \mathsf{HCom}_{PK_{\mathsf{TMC}}}(m_i)$. Next, compute $(C, \mathsf{aux}_{\mathsf{VC}}) \leftarrow \mathsf{VC.Com}_{\mathsf{pp}}(C_1, \ldots C_q)$. The output is C and the auxiliary information is $\mathsf{aux} = (\mathsf{aux}_{\mathsf{VC}}, m_1, C_1, \mathsf{aux}^1_{\mathsf{TMC}}, \ldots, m_q, C_q, \mathsf{aux}^q_{\mathsf{TMC}})$.

$\mathsf{qHOpen}_{pk}(m_i, i, \mathsf{aux})$ Extract $(m_i, C_i, \mathsf{aux}^i_{\mathsf{TMC}})$ from aux and set $\Lambda_i \leftarrow \mathsf{VC.Open}_{\mathsf{pp}}(C_i, i, \mathsf{aux}_{\mathsf{VC}})$. The opening information is $\tau_i = (\Lambda_i, C_i, \pi_i)$ where π_i is the output of $\mathsf{HOpen}_{PK_{\mathsf{TMC}}}(m_i, \mathsf{aux}^i_{\mathsf{TMC}})$.

$\mathsf{qHVer}_{pk}(C, m_i, i, \tau_i)$ Parse τ_i as (Λ_i, C_i, π_i). The hard verification algorithm returns 1 if and only if both $\mathsf{HVer}_{PK_{\mathsf{TMC}}}(C_i, m_i, \pi_i)$ and $\mathsf{VC.Ver}_{\mathsf{pp}}(C, C_i, i, \Lambda_i)$ return 1.

$\mathsf{qSCom}_{pk}()$ For $i = 1$ to q compute $(C_i, \mathsf{aux}^i_{\mathsf{TMC}}) \leftarrow \mathsf{SCom}_{PK_{\mathsf{TMC}}}()$. Next, compute $(C, \mathsf{aux}_{\mathsf{VC}}) \leftarrow \mathsf{VC.Com}_{\mathsf{pp}}(C_1, \ldots, C_q)$. The output is C and the auxiliary information is $\mathsf{aux} = (\mathsf{aux}_{\mathsf{VC}}, m_1, C_1, \mathsf{aux}^1_{\mathsf{TMC}}, \ldots, m_q,$

$\mathsf{qSOpen}_{pk}(m_i, i, \mathsf{flag}, \mathsf{aux})$ Extract $(m_i, C_i, \mathsf{aux}^i_{\mathsf{TMC}})$ from aux and set $\Lambda_i \leftarrow \mathsf{VC.Open}_{\mathsf{pp}}(C_i, i, \mathsf{aux}_{\mathsf{VC}})$. The opening information is $\tau_i = (\Lambda_i, C_i, \pi_i)$ where π_i is the output of $\mathsf{SOpen}_{PK_{\mathsf{TMC}}}(m_i, \mathsf{aux}^i_{\mathsf{TMC}})$.

$\mathsf{qSVer}_{pk}(C, m, i, \tau_i)$ Parse τ_i as (Λ_i, C_i, π_i). The soft verification algorithm returns 1 if and only if both $\mathsf{SVer}_{PK_{\mathsf{TMC}}}(C_i, m_i, \pi_i)$ and $\mathsf{VC.Ver}_{\mathsf{pp}}(C, C_i, i, \Lambda_i)$ return 1.

$\mathsf{qFake}_{pk,tk}()$ This is the same as the qSCom algorithm.

$\mathsf{qHEquiv}_{pk,tk}(m, i, \mathsf{aux})$ Extract $(C_i, \mathsf{aux}^i_{\mathsf{TMC}})$ (for all $i = 1$ to q) and set $\Lambda_i \leftarrow \mathsf{VC.Open}_{\mathsf{pp}}(C_i, i, \mathsf{aux}_{\mathsf{VC}})$. The hard equivocation is $\tau_i = (\Lambda_i, C_i, \pi_i)$ where π_i is the output of $\mathsf{HEquiv}_{PK_{\mathsf{TMC}}, tk_{\mathsf{TMC}}}(m, \mathsf{aux}^i_{\mathsf{TMC}})$

$\mathsf{qSEquiv}_{pk,tk}(m, i, \mathsf{aux})$ Extract $(C_i, \mathsf{aux}^i_{\mathsf{TMC}})$ (for all $i = 1$ to q) and set $\Lambda_i \leftarrow \mathsf{VC.Open}_{\mathsf{pp}}(C_i, i, \mathsf{aux}_{\mathsf{VC}})$. The soft equivocation is $\tau_i = (\Lambda_i, C_i, \pi_i)$ where π_i is the output of $\mathsf{SEquiv}_{PK_{\mathsf{TMC}}, tk_{\mathsf{TMC}}}(m, \mathsf{aux}^i_{\mathsf{TMC}})$

The correctness of the scheme easily follows from the correctness of the underlying building blocks. Its security follows from the following theorem

Theorem 10 *Assuming that* TMC *is a trapdoor mercurial commitment and* VC *is a vector commitment, then the scheme defined above is a trapdoor q-mercurial commitment.*

Proof. We prove the theorem by showing that all the properties of trapdoor q-mercurial commitments are satisfied.

q-Mercurial Binding. Here we show that neither kind of collisions (i.e. hard or soft ones) are possible. We start by discussing the case of hard collisions. The case of soft collisions is basically the same.

Hard Collisions. Assume that there is an adversary $\mathcal{A}$ that manages to succeed in the following experiment with non negligible probability. $\mathcal{A}$ receives in input the public key of the qTMC he intends to attack and produces a commitment C together with two valid (but different!) openings (τ_i, τ_i'). In particular assume that, wlog, both τ_i and τ_i' are valid decommitments for messages in position i. For simplicity assume that $\tau_i = (\Lambda_i, C_i, \pi_i)$ opens to m_i at position i, while $\tau_i' = (\Lambda_i', C_i', \pi_i')$ opens to m_i' at position i (and of course $m_i \neq m_i'$). We distinguish two (mutually exclusive) types of openings the adversary might produce:

Type I $C_i = C_i'$
Type II $C_i \neq C_i'$

Type I. If the adversary produces two different openings of type I it is easy to see that one can break the mercurial binding of TMC. Indeed both τ_i and τ_i' are correctly verified (i.e. $\mathsf{HVer}_{PK_{\mathsf{TMC}}}(C_i, m_i, \pi_i) = 1$ and $\mathsf{HVer}_{PK_{\mathsf{TMC}}}(C_i, m_i', \pi_i') = 1$) and thus the tuple $(C_i, m_i, \pi_i, m_i', \pi_i')$ is a hard collision for the mercurial binding of TMC.

Type II. If the adversary produces two openings of type II then it is easy to see that this case breaks the position binding property of the vector commitment VC. Indeed, by definition, both Λ_i and Λ_i' are correctly verified openings (i.e. $\mathsf{VC.Ver}_{\mathsf{pp}}(C, C_i, i, \Lambda_i) = 1$ and $\mathsf{VC.Ver}_{\mathsf{pp}}(C, C_i', i, \Lambda_i') = 1$). Thus one can output $(C, C_i, C_i', i, \Lambda_i, \Lambda_i')$ to break the position binding.

Soft Collisions. This proof is almost identical to that given for hard collision and is omitted. The only difference is that if the adversary outputs two openings of type I, here we break the mercurial binding of TMC by finding a soft collision for it.

Equivocation and Hiding. In all the cases we show that it is infeasible for an adversary, on input both the public key and the trapdoor, to distinguish between a real commitment/decommitment tuple and a fake/equivocation one.

In general $\mathcal{A}$ receives in input the pair (pk, tk), gives an index i and a tuple $(m_1, \ldots, m_q)$ and receives back a tuple (S_b, τ_b) that is generated using different algorithms according to a secret bit b. If we consider $\tau_b = (\Lambda_i, C_i, \pi_i)$ and observe the pair (S_b, Λ_i), it is easy to see that in all the cases these elements have always the same distribution no matter of which algorithm has generated them. Thus the only elements that might have different distributions according to b are C_i and π_i, but these are indistinguishable provided that TMC satisfies hiding and equivocations.

On the efficiency of the CDH instantiation. By instantiating the above scheme with our vector commitment based on CDH (and with the discrete log based TMC from [24]), one gets a qTMC based on CDH whose efficiency is roughly the same as that of the scheme in [20] based on q-DHE. For the sake of a fair comparison we notice that in our construction the reduction to CDH is not tight (due to the non-tight reduction from Square-DH to CDH [23]), and our scheme suffers from public parameters of size $O(q^2)$. In contrast, the scheme by Libert and Yung has a

tight reduction to the q-DHE problem and achieves public parameters of size $O(q)$. However, we think that the CDH and the q-DHE assumptions are not easily comparable, especially given that the latter is defined with instances of size $O(q)$, where q is known to degrade the quality of the assumption (see [13, 29] for some attacks). Furthermore, a more careful look shows that in our scheme the verifier *does not* need to store this many elements. This is because the $h_{i,j}$'s are not required for verification. Thus, from the verifier side, the space required is actually only $O(q)$. In the application of ZK-EDBs such an optimization reflects on the size of the common reference string. More precisely, while the server still needs to store a CRS of size $O(q^2)$, the client is required to keep in memory only a portion of the CRS of size of $O(q)$ (thus allowing for comparable client-side requirements with respect to [20]).

5.3 Updatable ZK-EDBs with Short Proofs and Updates

In most practical applications databases are frequently updated. The constructions of ZK-EDBs described so far do not deal with this and the only way of updating a ZK-EDB is to actually recompute the entire commitment from scratch every time the database changes. This is highly undesirable as previously issued proofs can no longer be valid.

Liskov addressed this problem in [21] by introducing the notion of *Updatable Zero-Knowledge Databases* and proposing a construction for it. Updatable ZK-EDB solve the issue by providing two update procedures. On the one hand, the prover is allowed to change the value $D(x)$ of elements in the database and then output a new commitment Com' together with some update information U. On the other hand, a user who holds a proof π_y for a key $y \neq x$ can use U to produce π'_y which will be valid w.r.t. Com' and the key y.

Liskov distinguished between opaque and transparent updates, but addressed only the problem of constructing ZK EDBs with transparent updates[10]. Informally, the difference between the two notions is that an opaquely updated database commitment is indistinguishable from a new one. Instead, transparent updates explicitly reveals that an update has occurred an thus provide a way to update existing proofs.

In [21] it is showed how to build Updatable ZK EDB by appropriately modifying the basic approach of combining Merkle trees and mercurial commitments [21]. In particular, rather than using standard mercurial commitments, Liskov employed a new primitive called *updatable mercurial commitment*. Very informally, updatable mercurial commitments are like standard ones with the additional feature that they allow for two update procedures. The committer can change the message inside the commitment and produce: a new commitment and an update information. These can later be used by verifiers to update their commitments and the associated proofs (that will be valid w.r.t. the new commitment). Therefore whenever the prover changes some value $D(x)$ in the database, first he has to update the commitment in the leaf labeled by x and then he updates all the commitments in the path from x to the root. The new database commitment is the updated commitment in the root node, while the database update information contains the update informations for all the nodes involved in the update.

A natural question raised by the methodology above is whether the zero-knowledge property remains preserved after an update occurs, as the latter reveals information about the updated key. To solve this issue Liskov proposed to "mask" the label of each key x (i.e. the paths in the tree) using a pseudorandom pseudonym $N(x)$ and he relaxed the zero-knowledge property to hold w.r.t. $N()$. Further details can be found in [21].

[10] More precisely the proposed constructions of opaquely updatable ZK EDBs are either trivial or highly inefficient.

In [21] two constructions of updatable mercurial commitments are given. One is generic and uses both standard and mercurial commitments. The other one is direct and builds from the DL-based mercurial commitment of [24].

Our Result. We introduce the notion of *updatable* q-mercurial commitments, and then we show that these can be built from vector commitments and updatable mercurial commitments. Next, by applying the methodology of Liskov sketched above, adapted with the compact construction of Catalano *et al.* [10], we can build the first *compact* Updatable ZK-EDB. It is interesting to observe that by using the compact construction the resulting ZK-EDB improves over the scheme in [21] both in terms of proofs' size and length of the update information, as these both grow linearly in the height of the tree $log_q 2^k$ (which is strictly less than k for $q > 2$).

Updatable mercurial commitments First, we provide a formal definition of updatable mercurial commitments since in [21] it is only given as a sketch.

An *updatable (trapdoor) mercurial commitment* (uTMC for brevity) is defined like a mercurial commitment with the following additional algorithms:

$\mathsf{Update}_{pk}(C, \mathsf{aux}, m')$ This algorithm is run by the committer who produced C (and holds aux). It takes as input a message m' and outputs a new commitment C' and an update information U. Regardless of the type of C, the updated commitment C' is always a hard commitment.

$\mathsf{ProofUpdate}_{pk}(C, \pi, m', U)$ This algorithm is run by any user who holds a proof π for a commitment C. Given the update information U and the new message m', ProofUpdate outputs the updated commitment C' and an updated proof π' which will be valid w.r.t. C' and m'. The updated proof π' will be of the same type of π.

The mercurial binding is defined as usual, namely for any PPT adversary it is computationally infeasible to open a commitment (even an updated one) to two different messages. Then we require updatable mercurial commitments to satisfy the usual mercurial hiding and equivocations properties of TMCs, but we also require such properties to hold even for updated commitments, namely when the adversary can see the update information[11].

Remark 11 [On the security of updatable mercurial commitments] Given the hiding and equivocations properties depicted above, one may wonder if they are the strongest possible properties one can have in a scenario where updates are issued. In fact we notice they are not. For example, there is no guarantee that an updated commitment is hiding w.r.t. the original message contained before the update. However, as already noticed by Liskov in [21], such weakness is not a problem for the means of building updatable ZK EDBs as in this case the committed messages are commitments themselves.

However since updatable mercurial commitments might have applications outside the context of updatable ZK EDBs, it would be interesting to consider more general security properties. In Appendix B we address this problem and we provide augmented security properties for updatable mercurial commitments that we call *Updatable Hiding*, *Updatable mercurial hiding* and *Updatable Equivocations*. The schemes given by Liskov in [21] (both the generic one and the one based on Discrete Log) and the scheme we propose in the following section *do not* satisfy the new security definition. Therefore we will show in Appendix B how to realize an updatable mercurial commitment from RSA that is secure under updatable hiding and equivocations.

[11] Notice that since an updated commitment is always a hard commitment, in this case we are interested only to Hiding and HHEquivocation.

An updatable mercurial commitment from RSA Here we extend the scheme of Gennaro and Micali [14] to make it updatable via the following additional algorithms:

$\mathsf{Update}_{pk}(C, s, \mathtt{aux}_{\mathsf{TMC}}, m')$ If C is a hard commitment, parse $\mathtt{aux}$ as (m, R, r) and proceed as follows. Pick a random $r' \xleftarrow{\$} \mathbb{Z}_N$ and compute: $s' = r'^e T \bmod N$ and $C' = s'^{m'} R^e \bmod N$. Otherwise, if C is a soft commitment (i.e. $\mathtt{aux} = (R, r)$) that has been teased to a message m then proceed as follows. Pick a random $r' \xleftarrow{\$} \mathbb{Z}_N$, set $R' = r^{-m} R$ and compute $s' = r'^e T \bmod N$ and $C' = s'^{m'} R'^e \bmod N$. Finally output the updated commitment (C', s') and the update information $U = r'$.

$\mathsf{ProofUpdate}_{pk}(C, \pi, m', U)$ Who holds a proof π that was valid for C, can use the update information U to compute the updated commitment C' to m' and produce a new proof π' which will be valid w.r.t. C' and m'. The updated commitment is $(C' = s'^{m'} R^e \bmod N, s' = r'^e T \bmod N)$. If π is a soft opening (i.e. $\pi = R$) the updated proof π' remains the same. Otherwise, if π is a hard opening (i.e. $\pi = (R, r)$) the updated proof is $\pi' = (R, r')$.

Theorem 12 *If the RSA assumption holds, then the scheme given above is an updatable trapdoor mercurial commitment.*

It is easy to see that the same security proof for the scheme of Gennaro and Micali [14] still holds in this case.

Updatable q-mercurial commitments. An updatable q-(trapdoor) mercurial commitment is defined like a qTMC with the following two additional algorithms:

$\mathsf{qUpdate}_{pk}(C, \mathtt{aux}, m', i)$ This algorithm is run by the committer who produced C (and holds the corresponding $\mathtt{aux}$) and wants to change the i-th committed message with m'. The algorithm takes as input m' and the position i and outputs a new commitment C' and an update information U.

$\mathsf{qProofUpdate}_{pk}(C, \tau_j, m', i, U)$ This algorithm can be run by any user who holds a proof τ_j for some message at position j in C and allows the user to produce a new proof τ'_j (and the updated commitment C') which will be valid w.r.t. C' that contains m' as the new message at position i. The value U contains the update information which is needed to compute such values.

The q-mercurial binding property is defined as usual, namely for any PPT adversary it is computationally infeasible to open a commitment (even an updated one) to two different messages at the same position.

Hiding and equivocations for updatable qTMCs easily follow from those of updatable mercurial commitments given in the previous section by extending them to the case of sequences of q messages.

Updatable q-mercurial commitments from Vector Commitments. In this section we extend the result of Theorem 10 to the updatable case. Namely we show that an updatable qTMC can be built using an updatable (trapdoor) mercurial commitment uTMC and a vector commitment VC. The construction is essentially the same as that given in Section 5.2 augmented with the following update algorithms:

$\mathsf{qUpdate}_{pk}(C, \mathtt{aux}, m', i)$ Parse $\mathtt{aux}$ as $(m_1, C_1, \mathtt{aux}^1_{\mathsf{uTMC}}, \ldots, m_q, C_q, \mathtt{aux}^q_{\mathsf{uTMC}}, \mathtt{aux}_{\mathsf{VC}})$, extract $(C_i, m_i, \mathtt{aux}^i_{\mathsf{uTMC}})$ from it and run $(C'_i, U_{\mathsf{uTMC}}) \leftarrow \mathsf{Update}_{pk_{\mathsf{uTMC}}}(C_i, \mathtt{aux}^i_{\mathsf{uTMC}}, m')$. Then update the vector commitment $(C', U') \leftarrow \mathsf{VC.Update}_{\mathsf{pp}}(C, C_i, C'_i, i)$ and output C' and $U = (U', C_i, C'_i, i, U^i_{\mathsf{uTMC}})$.

$\mathsf{qProofUpdate}_{pk}(C, \tau_j, m', i, U)$ The client who holds a proof $\tau_j = (\Lambda_j, C_j, \pi_j)$ that is valid w.r.t. to C for some message at position j, can use the update information U to compute the updated commitment C' and produce a new proof τ'_j which will be valid w.r.t. the new C'. We distinguish two cases:

1. $i \neq j$. Compute $(C', \Lambda'_j) \leftarrow \mathsf{VC.ProofUpdate}_{\mathsf{pp}}(C, \Lambda_j, C'_i, i, U')$ and output the updated commitment C' and the updated proof $\tau_j = (\Lambda'_j, C_j, \pi_j)$.
2. $i = j$. Let $(C'_i, \pi'_i) \leftarrow \mathsf{ProofUpdate}_{pk_{\mathsf{uTMC}}}(C_i, m', \pi_i, U^i_{\mathsf{uTMC}})$. Compute the updated commitment as $(C', \Lambda'_i) \leftarrow \mathsf{VC.ProofUpdate}_{\mathsf{pp}}(C, \Lambda_i, C'_i, i, U')$ and the updated proof is $\tau'_i = (\Lambda'_i, C'_i, \pi'_i)$.

Theorem 13 *If* uTMC *is an updatable trapdoor mercurial commitment and* VC *is an updatable vector commitment, then the scheme given above is an updatable concise trapdoor q-mercurial commitment.*

Proof (Sketch). The proof of the q-mercurial binding remains the same as that of Theorem 10 and the same holds for hiding and equivocations of "standard" commitments. To see that hiding and equivocations hold even for updated commitments, consider the update information $U = (U', C_i, C'_i, i, U^i_{\mathsf{uTMC}})$ where C'_i is an updated commitment to m'. Then it is easy to see that U does not reveal information about m' as long as the scheme uTMC is hiding.

5.4 Strongly-Independent Compact Zero-Knowledge Databases

Gennaro and Micali introduced in [14] the notion of *Independent Zero-Knowledge Databases*. Roughly speaking the *independence* property prevents an adversary from correlating its database and proofs to those of an honest prover. In particular Gennaro and Micali show that previous constructions of ZKDB do not satisfy this stronger notion of security and propose new ones achieving it.

In this section, we show that vector commitments can be used also to significantly improve the constructions of Independent ZKDB. More specifically, vector commitments can be employed to build multi-trapdoor q-mercurial commitments that are a basic building block for the construction of Compact Independent ZKDB.

The section is organized as follows. First, we recall below the notion of Independent Zero-Knowledge Databases given in [14]. Then, in Appendix 5.4 we show a construction based on multi-trapdoor q-mercurial commitments. And finally, in Appendix 5.4, we give a construction of multi-trapdoor q-mercurial commitments from Vector Commitments.

Independent Zero-Knowledge Databases. The definition of strong-independence considers a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. $\mathcal{A}_1$ sees ℓ database commitments $Com_1, \ldots, Com_\ell$ generated by honest provers and is allowed to make queries on the underlying databases $Db_1, \ldots, Db_\ell$. Finally $\mathcal{A}_1$ must output a new database commitment Com. Then two copies of the second-stage adversary $\mathcal{A}_2$ are run: the first is given oracle access to provers that output proofs of elements in Db_i; the second copy of $\mathcal{A}_2$ has access to an oracle that makes proofs for databases Db'_i such that, for all $i = 1$ to ℓ, Db_i and Db'_i agree on the elements queried by $\mathcal{A}_1$.

More formally, we say that a ZK EDB system is *strongly-independent* if for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and databases $Db_1, \ldots, Db_\ell, Db'_1, \ldots, Db'_\ell$ there exists a simulator $SIM =$ (SimSetup, SimCom, SimProve) such that the following probability

$$Pr\left[\begin{array}{l}(CRS,t)\leftarrow \mathsf{SimSetup}(1^k);\\ (Com_i,st_i)\leftarrow \mathsf{SimCom}(CRS,t)\forall i=1,\ldots,\ell;\\ (Com,st)\leftarrow \mathcal{A}_1^{\mathsf{SimProve}^{Db_i}(st_i,Com_i)}(CRS,st);\\ (x,\pi_x)\leftarrow \mathcal{A}_2^{\mathsf{SimProve}^{Db_i}(st_i,Com_i)}(CRS,st);\\ (x,\pi'_x)\leftarrow \mathcal{A}_2^{\mathsf{SimProve}^{Db'_i\vdash_{Q_i}Db_i}(st_i,Com_i)}(CRS,st):\\ \mathsf{V}(CRS,Com,x,\pi_x)\ \neq\ \mathsf{V}(CRS,ZPK,x,\pi'_x)\ \neq\\ \perp\wedge((Com\neq Com_i\forall i)\vee(\exists i: Com=Com_i\wedge x\notin\\ Q'_i))\end{array}\right]$$

is negligible in k.

In the definition above, Q_i is the list of queries made by $\mathcal{A}_1$ to the oracle $\mathsf{SimProve}^{Db_i}(st_i, Com_i)$ and we denote with $Db'_i \vdash_{Q_i} Db_i$ that the database Db'_i is restricted to agree with Db_i on the elements of Q_i.

Gennaro and Micali [14] give also two weaker notions of independence. The first one (*weak independence*) considers an adversary $\mathcal{A}_1$ that outputs the commitment without making any query. The second one (simply *independence*) is the same as strong-independence except that $\mathcal{A}_2$ is not allowed to copy one of the honest provers' commitments.

Strongly-Independent Zero-Knowledge Databases from (multi-trapdoor) q-mercurial commitments The construction of strongly-independent ZKDBs of Gennaro and Micali [14] uses as building block a *multi-trapdoor mercurial commitment* (which we define below) that is combined with a signature scheme and a collision resistant hash function. Libert and Yung [20] later adapted this construction to work in the q-ary tree setting of Catalano *et al.* [10] by using, as building block, a *multi-trapdoor q-mercurial commitment*.

Before describing this construction, we first give the definition of *multi-trapdoor q-mercurial commitments*. Such a scheme is a collection of the following algorithms:

$\mathsf{qKeyGen}(1^k, q)$ is a probabilistic algorithm that takes in input a security parameter k, the parameter q indicating the length of valid sequences and outputs a pair of public/private keys (pk, tk). The public key also contains the description of a tag space $\mathcal{T}$.

$\mathsf{qHCom}_{pk}(m_1, \cdots, m_q, tag)$ Given an ordered tuple of q messages and $tag \in \mathcal{T}$, qHCom computes a hard commitment C to $(m_1, \cdots, m_q)$ under (pk, tag) and returns an auxiliary information aux.

$\mathsf{qHOpen}_{pk}(m, i, tag, \mathsf{aux})$ Let $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q, tag')$, if $m = m_i$ and $tag = tag'$ the hard opening algorithm produces a hard decommitment π. Otherwise it returns an error message.

$\mathsf{qHVer}_{pk}(m, i, tag, C, \pi)$ The hard verification algorithm $\mathsf{qHVer}_{pk}(m, i, tag, C, \pi)$ accepts (outputs 1) only if π proves that C is created to a sequence $(m_1, \cdots, m_q)$ and a tag tag' such that $m_i = m$ and $tag' = tag$.

$\mathsf{qSCom}_{pk}(tag)$ produces a soft commitment C and an auxiliary information aux under (pk, tag). A soft commitment string C is created to no specific sequence of messages.

$\mathsf{qSOpen}_{pk}(m, i, tag, \mathsf{flag}, \mathsf{aux})$ produces a soft decommitment τ (also known as *"tease"*) to a message m at position i. The parameter $\mathsf{flag} \in \{\mathbb{H}, \mathbb{S}\}$ indicates if τ corresponds to either a hard commitment $(C, \mathsf{aux}) = \mathsf{qHCom}_{pk}(m_1, \cdots, m_q, tag')$ or to a soft commitment $(C, \mathsf{aux}) = \mathsf{qSCom}_{pk}(tag')$. The algorithm returns an error message if C is a hard commitment and $m \neq m_i$ or $tag \neq tag'$.

$\mathsf{qSVer}_{pk}(m, i, tag, C, \tau)$ checks if τ is a valid decommitment for C to m of index i under (pk, tag). If it outputs 1 and τ corresponds to a hard commitment C to $(m_1, \cdots, m_q)$, then C could be hard-opened to (m, i), or rather $m = m_i$.

$\mathsf{qTrapGen}_{pk,tk}(tag)$ given $tag \in \mathcal{T}$, it outputs a tag-specific trapdoor key tk_{tag}.

$\mathsf{qFake}_{pk,tk_{tag}}(tag)$ takes as input a tag-specific trapdoor tk_{tag} and produces a q-fake commitment C. C is not bound to any sequence $(m_1, \cdots, m_q)$ but is created to a specific tag. It also returns an auxiliary information aux.

$\mathsf{qHEquiv}_{pk,tk}(m, i, tag, \mathsf{aux})$ The hard equivocation algorithm generates a hard decommitment π to the message of m at position i for $(C, \mathsf{aux}) = \mathsf{qFake}_{pk,tk_{tag}}(tag)$. A q-fake commitment is very similar to a soft commitment with the additional property that it can be hard-opened.

$\mathsf{qSEquiv}_{pk,tk_{tag}}(m, i, tag, \mathsf{aux})$ generates a soft decommitment τ to m of position i using the auxiliary information produced by the qFake algorithm.

The correctness requirements are essentially the same as those for trapdoor q-mercurial commitment schemes with the addition of the tag.

The security properties are an extension of the q-mercurial binding, hiding and equivocations that are given in section 5.1.

q-Mercurial binding The adversary is allowed to choose a set of strings $tag_1, \ldots, tag_n \in \mathcal{T}$ that are given to a Challenger who replays back with pk and $tk_{tag_1}, \ldots, tk_{tag_n}$ (that are generated using $\mathsf{qTrapGen}_{pk,tk}(tag_i)$). Then the q-mercurial binding holds if it is computationally infeasible for an algorithm $\mathcal{A}$ to come up with $tag^* \in \mathcal{T} \setminus \{tag_1, \ldots, tag_n\}$, a commitment C and two different openings m, i, π, m', π' such that $\mathsf{qHVer}_{pk}(C, m, i, tag^*, \pi) = 1$ and $\mathsf{qHVer}_{pk}(C, m', i, tag^*, \pi') = 1$, or $\mathsf{qHVer}_{pk}(C, m, i, tag^*, \pi) = 1$ and $\mathsf{qSVer}_{pk}(C, m', i, tag^*, \pi') = 1$.

q-Mercurial hiding and Equivocations These properties are basically the same of standard q-mercurial commitments. The only difference is that here the adversary is allowed to specify in each game a tag under which the challenge commitment is created. This means that the tag is not necessarily hidden in the commitment, but this suffices for the means of our application.

We notice that our definition is slightly different from that given in [20] in that we require the qSCom and qFake algorithms to take in input the tag. Though this definition is weaker, it is sufficient for the purposes of our application, namely constructing Compact Independent ZKDBs.

Given a multi-trapdoor q-mercurial commitment, a signature scheme $\Sigma = (\mathsf{kg}, \mathsf{sig}, \mathsf{ver})$ and a collision resistant hash function $H : \{0,1\}^* \to \mathcal{T}$ (where $\mathcal{T}$ is the tag space), strongly-independent ZKDBs can be realized as follows. To commit to a database one first generates a signing key pair $(vk, sk) \xleftarrow{\$} \mathsf{kg}()$ and then builds a ZKDB using the construction of [10] (which is recalled in Appendix A.1) where all the (multi-trapdoor) commitments are generated with the same tag $H(vk)$. If Com is the database commitment obtained from the procedure above, then the committer outputs (Com, vk). To generate a proof for an element x, the prover produces π_x, as usual, and outputs (π_x, σ_x) where $\sigma_x = \mathsf{sig}_{sk}(Com, x)$. Finally, the verification procedure checks the validity of both π_x (w.r.t. the tag $H(vk)$) and the signature σ_x.

The strong-independence property of this construction is implied by the q-mercurial binding of the multi-trapdoor qTMC, the collision resistance of H and the security (unforgeability under chosen-message attack) of Σ. The proof is the same as that given in [20]. As already noticed above, our definition of multi-trapdoor q-mercurial commitments is slightly different, but we observe that

this does not invalidate the existing proof in [20] as the tag $H(vk)$ is known by the simulator when it computes fake commitments.

Building multi-trapdoor q-mercurial commitments The construction of multi-trapdoor q-mercurial commitment is basically the same as that of standard trapdoor q-mercurial commitment in section 5.2 except that the standard mercurial commitment is replaced with a multi-trapdoor one. This means that the tag is employed only to compute the underlying mercurial commitments. Multi-trapdoor mercurial commitments were introduced by Gennaro and Micali in [14], though they did not give an explicit definition for them. The primitive is defined essentially in the same way as multi-trapdoor q-mercurial commitments when restricted to the case $q = 1$.

Since Gennaro and Micali give in their paper multi-trapdoor mercurial commitment schemes based on either the Discrete Log or Strong RSA assumptions [14], we can plug these schemes in the above construction so as to finally obtain a version of Compact Strongly-Independent ZKDBs. In particular, by using our vector commitments, we obtain a compact scheme based on standard constant-size assumptions.

6 Universal Dynamic Accumulators from Vector Commitments

As an additional application of vector commitments, we show that these essentially generalize the notion of Universal Dynamic Accumulators (UDA). Informally, UDAs are dynamic accumulators which also support proofs of non membership. More precisely, we show a generic construction of UDA from vector commitments. Interestingly, by combining this result with the vector commitment given in Section 3.1, we are able to provide the first accumulator that works in prime order groups based on a standard constant-size assumption (i.e., CDH). Indeed, all previous constructions [18, 27, 7] in prime order groups rely on relatively young assumptions whose instances' size is polynomial in the security parameter.

We start by giving a definition for Universal Dynamic Accumulators. Our definition is inspired by that given in [7] for the case of basic (i.e. non universal) Dynamic Accumulators[12]. The definition uses an accumulator state state, which (also) contains bookkeeping information about the accumulated set V. A Universal Dynamic Accumulator is defined by the following algorithms:

$\mathsf{Acc.Setup}(1^k, n)$ takes as input the security parameter k and an integer n such that $[n]$ is a superset to all the sets that can be accumulated. This algorithm is run by the accumulator authority and outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Acc.Create}(\mathsf{pk}, \mathsf{sk}, V, n)$. If $V \not\subseteq [n]$, output some error message $\perp$. Otherwise, create an accumulating value acc_V for V. Output acc_V and the updated state information state.

$\mathsf{Acc.Add}(\mathsf{pk}, \mathsf{sk}, i, \mathsf{acc}_V, \mathsf{state})$. This algorithm allows to add the element i into the currently accumulated set V. It uses state to retrieve V from acc_V. Next, it produces an updated accumulator $\mathsf{acc}_{V'}$ for $V' = V \cup \{i\}$, it updates state accordingly, and produces an update information U. The update information basically keeps track of the changes in the set.

[12] We point out, that there are several definitions of UDA in the literature, none of which seems to emerge as the most popular one (basically almost each construction comes with its own definition). To add our two cents to this state of affairs, we decided to provide yet another definition, that we believe to be simple to use and general enough to capture the relevant properties a UDA should have.

$\mathsf{Acc.Remove}(\mathsf{pk}, \mathsf{sk}, i, \mathsf{acc}_V, \mathsf{state})$. This algorithm allows to remove the element i from the currently accumulated set V. It uses state to retrieve V from acc_V. Next, it produces the updated accumulator $\mathsf{acc}_{V'}$ for $V' = V - \{i\}$, it updates state accordingly, and produces an update information U.

$\mathsf{Acc.CreateWit}(\mathsf{pk}, i, \mathsf{acc}_V, \mathsf{state})$. Again, let V be the set accumulated in acc_V. This algorithm uses state to create a witness Wit_i to prove that $i \in V$ (or $i \notin V$).

$\mathsf{Acc.WitUpdate}(\mathsf{pk}, \mathsf{acc}_V, \mathsf{Wit}_j, U)$**.** Let V be the set (accumulated in acc_V) for which Wit_j is a valid witness for the element j. Moreover, denote with V' the set obtained from V with the changes in U. This algorithm allows the owner of Wit_j to produce a new Wit'_j, valid with respect to the accumulated value $\mathsf{acc}_{V'}$.

$\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}_i, \mathsf{acc}_V, b)$ allows to verify whether $i \in V$ or not. In particular, the algorithm outputs 1 if (1) $b = 1$ and Wit_i is a valid witness w.r.t. pk, i and acc_V that $i \in V$, or (2) $b = 0$ and Wit_i is a valid witness w.r.t. pk, i and acc_V that $i \notin V$. In all the remaining cases the algorithm does not accept and outputs 0.

We require a universal dynamic accumulator to satisfy the correctness and security properties. Roughly speaking, an UDA is *correct* if $\mathsf{Acc.Verify}$ always outputs 1 when invoked on honestly generated inputs. More formally, let $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{Acc.Setup}(1^k, n)$ be honestly generated. If acc_V is the accumulator for the set V (generated by an execution of $\mathsf{Acc.Create}$, possibly followed by executions of $\mathsf{Acc.Add}$ and/or $\mathsf{Acc.Remove}$) and Wit_i is a witness for $i \in V$ (resp. $i \notin V$), obtained either from $\mathsf{Acc.CreateWit}$ or from $\mathsf{Acc.WitUpdate}$, then it holds that $\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}_i, \mathsf{acc}_V, 1) = 1$ (resp. $\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}_i, \mathsf{acc}_V, 0) = 1$)

For security, we require that a dynamic accumulator scheme satisfies a property that we call *set binding*. Informally, this property says that an adversary should not be able to prove that an element i is (resp. is not) in the accumulator represented by acc_V while $i \notin V$ (resp. while $i \in V$). More formally, for any polynomially bounded algorithm $\mathcal{A}$ we define:

$$\begin{aligned}\mathbf{Adv}^{sb}_{\mathcal{A}}(k) = \Pr[\ &(\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}_i, \mathsf{acc}_{V_{\mathcal{O}}}, 1) \wedge i \notin V_{\mathcal{O}}) \vee (\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}_i, \mathsf{acc}_{V_{\mathcal{O}}}, 0) \wedge i \in V_{\mathcal{O}}) \mid \\ &(i, \mathsf{Wit}_i) \leftarrow \mathcal{A}^{Add(\cdot), Remove(\cdot)}(\mathsf{pk}, \mathsf{acc}_{\emptyset}, \mathsf{state}); \\ &(\mathsf{acc}_{\emptyset}, \mathsf{state}) \leftarrow \mathsf{Acc.Create}(\mathsf{pk}, \mathsf{sk}, \emptyset, n), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Acc.Setup}(1^k, n)]\end{aligned}$$

$\mathcal{A}$ is given access to the oracles $Add(\cdot), Remove(\cdot)$ that allow the adversary to, respectively, add and remove an element into/from the accumulator. Moreover, we denote with $V_{\mathcal{O}}$ the set obtained after the updates asked by $\mathcal{A}$.

Definition 14 [Set binding] A universal dynamic accumulator satisfies *set binding* if for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{sb}_{\mathcal{A}}(k) \leq \epsilon(k)$ where $\epsilon(k)$ is a function at most negligible in the security parameter k.

6.1 A scheme based on Vector Commitments

Here we describe how to construct UDA based on the simple notion of Vector Commitments.

$\mathsf{Acc.Setup}(1^k, n)$ Run $\mathsf{pp} \leftarrow \mathsf{VC.KeyGen}(1^k, n)$. Output $(\mathsf{pk} = \mathsf{pp}, \mathsf{sk} = \epsilon)$, where ϵ denotes the empty string here.

$\mathsf{Acc.Create}(\mathsf{pk}, \mathsf{sk}, V, n)$ If $V \not\subseteq [n]$, output some error message $\perp$. Otherwise, let $(m_1, \ldots, m_n)$ be the vector where $m_i = 1$ if $i \in V$ and $m_i = 0$ otherwise. Run $\mathsf{VC.Com}_{\mathsf{pk}}(m_1, \ldots, m_n)$ to get (aux, C). Output $\mathsf{acc}_V \leftarrow C$ and $\mathsf{state} \leftarrow (V, \mathsf{aux})$.

$\mathsf{Acc.Add}(\mathsf{pk}, \mathsf{sk}, i, \mathsf{acc}_V, \mathsf{state})$. Parse state as (V, aux) and run $\mathsf{VC.Update}_{\mathsf{pk}}(\mathsf{acc}_V, 0, 1, i)$ to get the updated commitment $\mathsf{acc}_{V'}$ and the update information U. Set $V' \leftarrow V \cup i$ and update state to (V', aux). The output is $(\mathsf{aux}_{V'}, \mathsf{state}, U)$.

$\mathsf{Acc.Remove}(\mathsf{pk}, \mathsf{sk}, i, \mathsf{acc}_V, \mathsf{state})$. Parse state as (V, aux) and run $\mathsf{VC.Update}_{\mathsf{pk}}(\mathsf{acc}_V, 1, 0, i)$ to get the updated commitment $\mathsf{acc}_{V'}$ and the update information U. Set $V' \leftarrow V \cup i$ and update state to (V', aux). The output is $(\mathsf{aux}_{V'}, \mathsf{state}, U)$.

$\mathsf{Acc.CreateWit}(\mathsf{pk}, i, \mathsf{acc}_V, \mathsf{state})$. Parse state as (V, aux) and run $\mathsf{VC.Open}_{\mathsf{pk}}(m_i, i, \mathsf{aux})$ to get the witness (i.e. the opening) Wit_i. The output is Wit_i.

$\mathsf{Acc.WitUpdate}(\mathsf{pk}, \mathsf{acc}_V, \mathsf{Wit}_j, U)$**.** Extract from U the updated value m' and its position. Run $\mathsf{VC.ProofUpdate}_{\mathsf{pk}}(\mathsf{acc}_V, \mathsf{Wit}_j, m', i, U)$ to get the new accumulated valued $\mathsf{acc}_{V'}$ (i.e. the one obtained when applying to acc_V the changes in U) and the updated witness Wit'_j. Output Wit'_j.

$\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}_i, \mathsf{acc}_V, b)$ Output $\mathsf{VC.Ver}_{\mathsf{pk}}(\mathsf{acc}_V, b, i, \mathsf{Wit}_i)$

The correctness of the scheme follows from the correctness of the underlying vector commitment. Set binding can be proved via the following theorem

Theorem 15 *If* VC *is a vector commitment, the dynamic universal accumulator constructed above is set binding.*

Proof. Let $\mathcal{B}$ be an adversary against the set binding property of the proposed scheme, we show how to build an equally efficient adversary $\mathcal{A}$ against the position binding property of the vector commitment. $\mathcal{A}$ receives on input the parameters pp, and sets $\mathsf{pk} = \mathsf{pp}$ and $\mathsf{sk} = \epsilon$. Next, $\mathcal{A}$ runs $\mathsf{Acc.Create}(pk, sk, \emptyset, n)$ and executes $\mathcal{B}$ on the so obtained values $(\mathsf{pk}, \mathsf{acc}_\emptyset, \mathsf{state})$. Notice that $\mathcal{A}$ can easily answer all *Add* and *Remove* queries, by running $\mathsf{Acc.Add}$ and $\mathsf{Acc.Remove}$, respectively. At some point, $\mathcal{B}$ halts and hands to $\mathcal{A}$ a triplet $(i, \mathsf{Wit}'_i, \mathsf{acc}_{V_\mathcal{O}})$ such that either $\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}'_i, \mathsf{acc}_{V_\mathcal{O}}, 1) \wedge i \notin V_\mathcal{O}$ or $\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}'_i, \mathsf{acc}_{V_\mathcal{O}}, 0) \wedge i \in V_\mathcal{O}$. Assume, w.l.o.g., that the former case applies. Notice that, being $\mathsf{acc}_\mathcal{O}$ a correctly computed accumulator, for correctness, there must exists a witness Wit_i such that $\mathsf{Acc.Verify}(\mathsf{pk}, i, \mathsf{Wit}_i, \mathsf{acc}_{V_\mathcal{O}}, 0)$. In other words, Wit_i is a valid witness of the fact that $i \notin V_\mathcal{O}$. Moreover, since $\mathsf{acc}_{V_\mathcal{O}}$ has been created by $\mathcal{A}$, it knows such a Wit_i. This means that the tuple $(\mathsf{acc}_{V_\mathcal{O}}, 1, 0, i, \mathsf{Wit}'_i, \mathsf{Wit}_i)$ will contradict the position binding of the underlying vector commitment.

Remark 16 In the construction above we restrict our scheme to accumulate integers $1, \ldots, n$. In some applications, however, one might want to be able to accumulate any set of size n. A simple way to do so, would be to publish (signed) triples (i, v_i, T) where v_i is the value one wants to associate to position i, ad T is some auxiliary information that might be required by the application. More precisely, the issuer of the accumulator uses a secure digital signature scheme, to sign i together with v_i and T. To verify a witness, one would be then required to verify the signature as well. To update an (accumulated) value at position i, a new signature to (i, v'_i, T') is (or might be) required (and this is why in our description above, $\mathsf{Acc.Add}$ and $\mathsf{Acc.Remove}$ both are allowed to require knowledge of a secret key)

Remark 17 We observe that the accumulator presented in this section is not expected to conceal the actual value of the accumulated elements. While it is a common practice to not require any

hiding feature from accumulators, there might be cases where such a feature could be useful. We remark that, at the cost of some inefficiency overhead, our scheme can be easily converted into an hiding accumulator as follows. To accumulate a set $V = \{v_1, \ldots, v_n\}$ one first creates a commitment c_i to each v_i and then accumulates the resulting c_i's.

References

1. Feng Bao, Robert Deng, and HuaFei Zhu. Variations of diffie-hellman problem. In Sihan Qing, Dieter Gollmann, and Jianying Zhou, editors, *Information and Communications Security*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer Berlin / Heidelberg, 2003.
2. Niko Bari and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 480–494. Springer, May 1997.
3. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 163–192. Springer, May 1997.
4. Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 111–131. Springer, August 2011.
5. Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, May 1993.
6. Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 258–275. Springer, August 2005.
7. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 481–500. Springer, March 2009.
8. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, August 2002.
9. Dario Catalano, Yevgeniy Dodis, and Ivan Visconti. Mercurial commitments: Minimal assumptions and efficient constructions. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 120–144. Springer, March 2006.
10. Dario Catalano, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 433–450. Springer, April 2008.
11. Dario Catalano, Mario Di Raimondo, Dario Fiore, and Mariagrazia Messina. Zero-knowledge sets with short proofs. *Information Theory, IEEE Transactions on*, 57(4):2488 –2502, april 2011.
12. Melissa Chase, Alexander Healy, Anna Lysyanskaya, Tal Malkin, and Leonid Reyzin. Mercurial commitments with applications to zero-knowledge sets. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 422–439. Springer, May 2005.
13. Jung Hee Cheon. Security analysis of the strong Diffie-Hellman problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 1–11. Springer, May / June 2006.
14. Rosario Gennaro and Silvio Micali. Independent zero-knowledge sets. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 34–45. Springer, July 2006.
15. J. Hastad, A. Schrift, and A. Shamir. The discrete logarithm modulo a composite hides o(n) bits. In *Journal of Computer and System Science*, volume 47 (3), pages 376–404, 1993.
16. Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 654–670. Springer, August 2009.
17. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010*, volume 6477, page 178, 2010.
18. Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 253–269. Springer, June 2007.
19. Benoît Libert, Thomas Peters, and Moti Yung. Group signatures with almost-for-free revocation. In *CRYPTO 2012*, LNCS, pages 571–589. Springer, August 2012.

20. 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, February 2010.
21. Moses Liskov. Updatable zero-knowledge databases. In Bimal K. Roy, editor, *ASIACRYPT 2005*, volume 3788 of *LNCS*, pages 174–198. Springer, December 2005.
22. Charles U. Martel, Glen Nuckolls, Premkumar T. Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.
23. Ueli M. Maurer and Stefan Wolf. Diffie-Hellman oracles. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 268–282. Springer, August 1996.
24. Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-knowledge sets. In *44th FOCS*, pages 80–91. IEEE Computer Society Press, October 2003.
25. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.
26. Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *Proceedings of the 7th conference on USENIX Security Symposium*, volume 7, pages 17–17, 1998.
27. Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer, February 2005.
28. Charalampos Papamanthou and Roberto Tamassia. Time and space efficient algorithms for two-party authenticated data structures. In Sihan Qing, Hideki Imai, and Guilin Wang, editors, *ICICS 07*, volume 4861 of *LNCS*, pages 1–15. Springer, December 2007.
29. Yumi Sakemi, Goichiro Hanaoka, Tetsuya Izu, Masahiko Takenaka, and Masaya Yasuda. Solving a discrete logarithm problem with auxiliary input on a 160-bit elliptic curve. In *PKC 2012*, LNCS, pages 595–608. Springer, 2012.
30. Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Trans. Comput. Syst.*, 1(1):38–44, 1983.
31. Emil Stefanov, Marten van Dijk, Alina Oprea, and Ari Juels. Iris: A scalable cloud file system with efficient integrity checks. Cryptology ePrint Archive, Report 2011/585, 2011. `http://eprint.iacr.org/`.
32. Roberto Tamassia and Nikos Triandopoulos. Certification and authentication of data structures. In *Alberto Mendelzon Workshop on Foundations of Data Management*, 2010.

A Zero-Knowledge Databases

Here we formally describe the properties of a zero-knowledge EDB. Let $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{V})$ be an EDB system as defined in section 5. We say that it is a *zero-knowledge EDB* if:

1. *Completeness.* $\forall$ databases D and $\forall x \in [D]$

$$Pr\begin{bmatrix} CRS \xleftarrow{\$} \mathsf{Setup}(1^k); (ZPK, ZSK) \xleftarrow{\$} \\ \mathsf{Commit}(CRS, D); \\ \pi_x \leftarrow \mathsf{Prove}(CRS, ZSK, x) \ : \ \mathsf{V}(CRS, ZPK, x, \pi_x) \ = \\ D(x) \end{bmatrix} = 1 - \epsilon(k)$$

   where $\epsilon(k)$ is negligible in k.
2. *Soundness.* $\forall x \in \{0,1\}^*$ and $\forall$ PPT algorithms P'

$$Pr\begin{bmatrix} CRS \xleftarrow{\$} \mathsf{Setup}(1^k); (ZPK', \pi_x, \pi'_x) \xleftarrow{\$} \\ P'(CRS, D) : \\ \mathsf{V}(CRS, ZPK', x, \pi_x), \mathsf{V}(CRS, ZPK', x, \pi'_x) \ \neq \\ \bot \wedge \\ \mathsf{V}(CRS, ZPK', x, \pi_x) \neq \mathsf{V}(CRS, ZPK', x, \pi'_x) \end{bmatrix}$$

   is negligible.

3. *Zero-Knowledge.* Let us assume that exists an oracle that, queried on a key x, returns the correct value associated to x in a database D if $x \in [D]$, otherwise it returns $\perp$. Then there exists a simulator $SIM = (\mathsf{SimSetup}, \mathsf{SimCom}, \mathsf{SimProve})$, where $\mathsf{SimProve}$ has oracle access to D, such that $\forall$ adversaries $\mathcal{A}$, $\forall k \in \mathbb{N}$ and $\forall$ databases D, $\mathcal{A}$'s view $View(k)$ in the real game is indistinguishable from its view $View'(k)$ in the game played by SIM.

$$View(k) = \left\{ \begin{array}{c} CRS \xleftarrow{\$} \mathsf{Setup}(1^k); (ZPK, ZSK) \xleftarrow{\$} \mathsf{Commit}(CRS, D); \\ (x_1, s_1) \xleftarrow{\$} \mathcal{A}(CRS, ZPK); \pi_{x_1} \leftarrow \mathsf{Prove}(ZSK, x_1) : \\ (x_2, s_2) \xleftarrow{\$} \mathcal{A}(CRS, ZPK, s_1, \pi_{x_1}); \\ \pi_{x_2} \leftarrow \mathsf{Prove}(x_2, ZSK); \\ \vdots \\ : ZPK, x_1, \pi_{x_1}, x_2, \pi_{x_2} ... \end{array} \right\}$$

$$View'(k) = \left\{ \begin{array}{c} (CRS', t) \xleftarrow{\$} \mathsf{SimSetup}(1^k); (ZPK', ZSK') \xleftarrow{\$} \mathsf{SimCom}(); \\ (x_1, s_1) \xleftarrow{\$} \mathcal{A}(CRS, ZPK'); \pi'_{x_1} \leftarrow \mathsf{SimProve}^D(x_1, ZSK') : \\ (x_2, s_2) \xleftarrow{\$} \mathcal{A}(CRS', ZPK', s_1, \pi'_{x_1}); \\ \pi'_{x_2} \leftarrow \mathsf{SimProve}^D(x_2, ZSK'); \\ \vdots \\ : ZPK', x_1, \pi'_{x_1}, x_2, \pi'_{x_2} \end{array} \right\}$$

A.1 Compact Zero-Knowledge Databases from trapdoor q-mercurial commitments

In this section we recall the construction given by Catalano *et al.* in [10, 11] that builds ZK EDBs from trapdoor q-mercurial commitments, trapdoor mercurial commitments and collision resistant hash functions.

Let T_h be the complete q-ary tree of height h, with q^h leaves. Let $\mathcal{U}_k$ be a universe of size 2^k and T_h be the complete q-ary tree of height h. Each element $x \in \mathcal{U}$ can be associated to a leaf of T_h and all the nodes in the path from the root to x can be labeled with the q-ary encoding of x. Thus ϵ (the empty string) is the label for the root. If v is a non-leaf node, then $v1, \cdots, vq$ are the labels for its q sons. Let $S \subseteq \mathcal{U}_k$ and consider the following two subsets of T_h: $TREE(S)$ and $FRONTIER(S)$. $TREE(S)$ is the subtree of T_h containing all the nodes in the paths from the leaves in S to the root. $FRONTIER(S) = \{v : v \notin TREE(S) \wedge parent(v) \in TREE(S)\}$.

The construction is given by describing the following algorithms:

$\mathsf{Setup}(1^k)$ Output the common reference string $CRS = (PK, PKM, H)$ where PK is the public key of a trapdoor q-mercurial commitment scheme $\mathcal{QC}$, PKM the public key of a trapdoor mercurial commitment scheme $\mathcal{C}$ and H is a collision resistant hash function.

$\mathsf{Commit}(CRS, D)$

1. Let $S = \{H(x) : x \in D, D(x) \neq \perp\}$. Construct the tree $T = TREE(S) \bigcup FRONTIER(S)$.
2. $\forall$ leaf-nodes $H(x)$ set:

$$n_{H(x)} = \begin{cases} H(y) & \text{if } D(x) = y, \\ 0 & \text{if } D(x) = \perp. \end{cases}$$

compute a hard commitment $(C_{H(x)}, \mathsf{aux}_{H(x)}) = \mathsf{HCom}_{PKM}(n_{H(x)})$ and set $m_{H(x)} = H(C_{H(x)})$.

3. $\forall$ internal nodes u such that $u \in FRONTIER(S) : (C_u, \mathtt{aux}_u) = \mathsf{qSCom}_{PK}()$.
4. $\forall$ internal nodes $u \in TREE(S)$: $(C_u, \mathtt{aux}_u) = \mathsf{qHCom}_{PK}(m_{u1}, \cdots, m_{uq})$, store $\mathtt{aux}_u$ in u and set $m_u = H(C_u)$ if $u \neq \epsilon$.
5. Output the commitment of the root: $ZPK = (C_\epsilon)$. The secret key ZSK contains all the stored elements.

$\mathsf{Prove}(CRS, ZSK, x)$ The prover algorithm produces a proof π_x of the database value of x. We distinguish between two cases:

1. $\mathbf{D}(\mathbf{x}) = \mathbf{y}$. The proof π_x contains the commitments and the hard openings in the path from the leaf-node $H(x)$ toward the root: $\pi_x = \Big\{ y, C_{H(x)}, HO_{H(x)} = \mathsf{HOpen}_{PKM}(H(y), \mathtt{aux}_{H(x)}),$ $\{C_u, qHO_u = \mathsf{qHOpen}_{PK}(m_v, i, \mathtt{aux}_u)\}_{v=H(x),\cdots,\epsilon-1} \Big\}$ with $u = parent(v), i = index_u(v)$.
2. $\mathbf{D}(\mathbf{x}) = \perp$. The prover first checks if the leaf node $H(x)$ is in the tree T. If $H(x) \notin T$, let $w \in FRONTIER(S)$ be the root of the missing subtree of T_k containing $H(x)$. The prover builds the subtree rooted in w using the same algorithm as in Commit. The proof π_x contains the commitments and the soft openings in the path from $H(x)$ toward the root ϵ: $\Big\{ C_{H(x)}, SO_{H(x)} = \mathsf{SOpen}_{PKM}(0, \mathbb{S}, \mathtt{aux}_{H(x)}),$ $\{C_u, qSO_u = \mathsf{qSOpen}_{PK}(m_v, i, \mathbb{H}/\mathbb{S}, \mathtt{aux}_u)\}_{v=H(x),\cdots,\epsilon-1} \Big\}$ with $u = parent(v), i = index_u(v)$.

$\mathsf{V}(CRS, ZPK, x, \pi_x)$ We consider two cases according to the type of π_x:

1. $D(x) = y$.
   (a) Check if $\mathsf{HVer}_{PKM}(H(y), C_{H(x)}, HO_{H(x)}) = 1$.
   (b) Compute $m_{H(x)} = H(C_{H(x)})$.
   (c) Let $v = parent(H(x))$ and i such that $H(x) = vi$. Check if $\mathsf{qHVer}_{PK}(m_{H(x)}, i, C_v, qHO_v) = 1$.
   (d) compute $m_v = H(C_v)$ and iterate as above for $u = parent(v), \cdots, \epsilon - 1$.
   (e) Check if $\mathsf{qHVer}_{PK}(m_u, i, ZPK, qHO_\epsilon) = 1$, where $u = i$ is the first node in the path from the root toward $H(x)$.
   (f) If none of the tests above fails, output y, otherwise output $\perp$.
2. $D(x) = \perp$.
   (a) Check if $\mathsf{SVer}_{PKM}(0, C_{H(x)}, SO_{H(x)}) = 1$
   (b) Compute $m_{H(x)} = H(C_{H(x)})$.
   (c) Let $v = parent(H(x))$ and i such that $H(x) = vi$. Check if $\mathsf{qSVer}_{PK}(m_{H(x)}, i, C_v, qSO_v) = 1$.
   (d) Compute $m_v = H(C_v)$ and and iterate as above for $u = parent(v), \cdots, \epsilon - 1$.
   (e) Check if $\mathsf{qSVer}_{PK}(m_u, i, ZPK, qSO_\epsilon) = 1$, where $u = i$ is the first node in the path from the root toward $H(x)$.
   (f) If none of the tests above fails output *out*, else output $\perp$.

B Updatable mercurial commitments with updatable hiding

As already noticed in section 5.3, it is interesting to consider security properties for updatable mercurial commitments that are reasonable in general, not only for the application of building ZK EDBs. In this section we address this problem and provide augmented security properties for hiding

and equivocations of updatable mercurial commitments. Finally we will also propose a scheme that realizes our stronger security definition.

Roughly speaking we would like that the usual hiding and equivocations properties should be preserved even when updates are issued. For this reason, we define the following augmented properties:

Updatable Hiding We say that an uTMC has *updatable hiding* if any PPT adversary $\mathcal{A}$ has at most negligible probability of winning the following game.

The adversary is allowed to choose two messages m_0, m_1 and then receives C which is a commitment to m_b for a randomly chosen $b \xleftarrow{\$} \{0,1\}$. Then $\mathcal{A}$ is allowed to choose other two messages m'_0, m'_1 and gets (C', U) that are an updated commitment and the update information for m'_d where $d \xleftarrow{\$} \{0,1\}$ is chosen at random. Finally $\mathcal{A}$ outputs two bits b', d' and we say that it *wins* if $b' = b$ and $d' = d$.

Informally this property says that given C, C' and U it is still infeasible to extract information about the committed messages. The definition above is intended for *computational* updatable hiding. The corresponding perfect and statistical definitions easily follow by making no assumptions on the power of the adversary.

Updatable mercurial hiding and equivocations We slightly modify the games of mercurial hiding and all equivocations (HH, HS, SS) and we say that an uTMC satisfies these properties if any PPT adversary can win in such games with at most negligible probability. Each game is modified in a way similar to updatable hiding. Namely, after having seen a commitment/decommitment tuple, we add a final phase where the adversary is allowed to see an updated commitment C' and the related update information U for a message m' of its choice. Then we require that the type of C remains hidden even when the adversary is given C, U. Since an updated commitment is always a hard commitment (by its definition) we are not interested into concealing its type.

It is easy to see that if a scheme satisfies these augmented properties, then it also satisfies the standard mercurial hiding and equivocations. Moreover if the mercurial commitment is proper, then it suffices to check only HH and SS equivocations as these imply all the other properties.

Remark 18 While in standard mercurial commitments the mercurial hiding implies the standard one, this no longer holds here. Indeed, observe that the games of updatable hiding and updatable mercurial hiding differ in that in the latter one does not concern about hiding the type of C'. This means that updatable hiding has to be independently checked.

B.1 Updatable mercurial commitments with updatable hiding from RSA

In this section we show an updatable mercurial commitment scheme that satisfies our security definitions given above under the RSA assumption. The scheme is obtained by extending the one given in section 5.3.

$\mathsf{KeyGen}(k, \ell)$ First, randomly choose two $k/2$-bit primes p_1, p_2 and a prime $e > N$.Next, let T a random element in $\mathbb{Z}_N$ and ck be the public key of a standard commitment $\mathcal{C} = (\mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$. The public key $\mathrm{PK}_{\mathsf{TMC}}$ is set as (N, T, e, ck) while the trapdoor tk is the e-th root of T.

$\mathsf{HCom}_{PK_{\mathsf{TMC}}}(m)$ Choose random elements $R, r, \alpha \in \mathbb{Z}_N$. Then compute $s = Tr^e \bmod N$, $C = s^{(m+\alpha)}R^e$, $(\mu, \mathsf{aux}_\mu) = \mathsf{Com}_{ck}(\alpha)$. Finally output the commitment (C, s, μ) and the auxiliary information $\mathsf{aux}_{\mathsf{TMC}} = (m, R, r, \alpha, \mathsf{aux}_\mu)$.

$\mathsf{HOpen}_{PK_{\mathsf{TMC}}}(m, \mathtt{aux}_{\mathsf{TMC}})$ Parse $\mathtt{aux}_{\mathsf{TMC}}$ as $(m', R, r, \alpha, \mathtt{aux}_\mu)$. Output $\perp$ if $m \neq m'$ and (R, r, α, π_μ) otherwise (where $\pi_\mu \leftarrow \mathsf{Open}_{ck}(\alpha, \mathtt{aux}_\mu)$).

$\mathsf{HVer}_{PK_{\mathsf{TMC}}}(C, s, \mu, m, R, r, \alpha, \pi_\mu)$ The hard verification algorithm returns 1 if and only if $s = Tr^e \bmod N$, $C = s^{(m+\alpha)}R^e$ and $\mathsf{Ver}_{ck}(\mu, \alpha, \pi_\mu) = 1$.

$\mathsf{SCom}_{PK_{\mathsf{TMC}}}()$ Choose random elements $R, r, \alpha \in \mathbb{Z}_N$. Then compute $s = r^e \bmod N$, $C = R^e$ and $(\mu, \mathtt{aux}_\mu) = \mathsf{Com}_{ck}(\alpha)$. Output the soft commitment (C, s, μ) and the auxiliary information $\mathtt{aux}_{\mathsf{TMC}} = (R, r, \alpha, \mathtt{aux}_\mu)$.

$\mathsf{SOpen}_{PK_{\mathsf{TMC}}}(m, \mathsf{flag}, \mathtt{aux}_{\mathsf{TMC}})$. If $\mathsf{flag} = \mathbb{H}$ proceed as follows. Parse $\mathtt{aux}_{\mathsf{TMC}}$ as $(m', R, r, \alpha, \mathtt{aux}_\mu)$, if $m' \neq m$ return $\perp$, otherwise output R, α, π_μ where $\pi_\mu \leftarrow \mathsf{Open}_{ck}(\alpha, \mathtt{aux}_\mu)$.
If $\mathsf{flag} = \mathbb{S}$, then output (R', α, π_μ) such that $R' = r^{-(m+\alpha)}R \bmod N$, and $\pi_\mu \leftarrow \mathsf{Open}_{ck}(\alpha, \mathtt{aux}_\mu)$.

$\mathsf{SVer}_{PK_{\mathsf{TMC}}}(C, s, \mu, R, \alpha, \pi_\mu)$ The soft verification algorithm returns 1 if and only if $C = s^{(m+\alpha)}R^e$ and $\mathsf{Ver}_{ck}(\mu, \alpha, \pi_\mu) = 1$.

$\mathsf{Fake}_{PK_{\mathsf{TMC}}, tk}()$ This is the same as the SCom algorithm.

$\mathsf{HEquiv}_{PK_{\mathsf{TMC}}, tk}(m, \mathtt{aux}_{\mathsf{TMC}})$ Parse $\mathtt{aux}_{\mathsf{TMC}}$ as $(R, r, \alpha, \mathtt{aux}_\mu)$ and output $(R', r', \alpha, \pi_\mu)$ where $r' = r \cdot (tk)^{-1} \bmod N$, $R' = r^{-(m+\alpha)}R \bmod N$ and $\pi_\mu \leftarrow \mathsf{Open}_{ck}(\alpha, \mathtt{aux}_\mu)$.

$\mathsf{SEquiv}_{PK_{\mathsf{TMC}}, tk}(m, \mathtt{aux}_{\mathsf{TMC}})$ Parse $\mathtt{aux}_{\mathsf{TMC}}$ as $(R, r, \alpha, \mathtt{aux}_\mu)$ and output (R', α, π_μ) where $R' = r^{-(m+\alpha)}R \bmod N$ and $\pi_\mu \leftarrow \mathsf{Open}_{ck}(\alpha, \mathtt{aux}_\mu)$.

$\mathsf{Update}_{pk}(C, s, \mu, \mathtt{aux}_{\mathsf{TMC}}, m')$ If C is a hard commitment, parse $\mathtt{aux}_{\mathsf{TMC}}$ as $(m, R, r, \alpha, \mathtt{aux}_\mu)$ and proceed as follows. Pick a random $r' \xleftarrow{\$} \mathbb{Z}_N$ and compute: $s' = r'^e T \bmod N$ and $C' = s'^{(m'+\alpha)}R^e \bmod N$. Otherwise, if C is a soft commitment (i.e. $\mathtt{aux}_{\mathsf{TMC}} = (R, r, \alpha, \mathtt{aux}_\mu)$) that has been teased to a message m then proceed as follows. Pick a random $r' \xleftarrow{\$} \mathbb{Z}_N$, set $R' = r^{-(m+\alpha)}R$ and compute $s' = r'^e T \bmod N$ and $C' = s'^{(m'+\alpha)}R'^e \bmod N$. Finally output the updated commitment (C', s', μ) and the update information $U = r'$.

$\mathsf{ProofUpdate}_{pk}(C, m', \pi, U)$ Who holds a proof π that was valid for C, can use the update information U to compute the updated commitment C' to m' and produce a new proof π' which will be valid w.r.t. C' and m'. The updated commitment is $(C' = s'^{(m'+\alpha)}R^e \bmod N, s' = r'^e T \bmod N, \mu)$. If π is a soft opening (i.e. $\pi = (R, \alpha, \pi_\mu)$) the updated proof π' remains the same. Otherwise, if π is a hard opening (i.e. $\pi = (R, r, \alpha, \pi_\mu)$) the updated proof is $\pi' = (R, r', \alpha, \pi_\mu)$.

Theorem 19 *If the RSA assumption holds and $\mathcal{C}$ is a commitment, then the scheme given above is an updatable trapdoor mercurial commitment.*

Proof. We prove the theorem by showing that each property is satisfied. The proof of the mercurial binding is basically the same as that of the scheme of Gennaro and Micali [14]. Let (C, s, μ) be a commitment and let (R, r, α, π_μ) and $(R', r', \alpha', \pi'_\mu)$ be two hard openings to m and m' respectively (such that $m \neq m'$). Here we can have two different cases. If $\alpha \neq \alpha'$ then we can break the binding of $\mathcal{C}$. Otherwise, if $\alpha = \alpha'$ we can break RSA as in the proof of [14] (with the additional case that $m = m' \bmod \phi(N)$ in which one can factor N).

The scheme clearly satisfies the usual hiding and equivocations properties for the same argument of [14]. Finally we show below that it satisfies our stronger properties of updatable hiding and equivocations.

Updatable Hiding and equivocations. In order to prove that the scheme satisfies updatable hiding we recall below the view of the adversary in this game:

$$\mathcal{D}_{b,d} = \{C = s^{(m_b+\alpha)}R^e, s = r^e T, \mu = \mathsf{Com}_{ck}(\alpha), C' = s'^{(m'_d+\alpha)}R^e, s' = r'^e T, U = r'\}.$$

One can observe that for any choice of the messages (m_b, m'_d) there exist α, R that satisfy the equations above. This means that, without μ, the distributions $\mathcal{D}_{b,d}$ for $b, d \in \{0,1\}$ are perfectly indistinguishable for any messages m_0, m_1, m'_0, m'_1. Therefore, if the commitment μ is hiding, updatable hiding also holds.

A similar argument applies in the case of the equivocations properties.

In the case of updatable HHEquivocation the adversary sees either one of the following tuples:

$$\{C = s^{(m+\alpha)}R^e, s = r^eT, \mu = \mathsf{Com}(\alpha), R, r, \pi_\mu, T^{1/e}, C' = s'^{(m'+\alpha)}R^e, s' = r'^eT, U = r'\}$$

$$\{C = R^e, s = r^e, \mu = \mathsf{Com}(\alpha), (r^{-(m+\alpha)}R), (rT^{1/e}), \pi_\mu, T^{1/e}, C' = s'^{(m'+\alpha)}(r^{-(m+\alpha)}R)^e, s' = r'^eT, U = r'\}$$

where it chooses m and m'. The two views are clearly perfectly indistinguishable.

Finally, in the case of updatable SSEquivocation it is trivial to see that the two views are exactly the same:

$$\{C = R^e, s = r^e, \mu = \mathsf{Com}(\alpha), (r^{-(m+\alpha)}, R), \pi_\mu, T^{1/e}, C' = s'^{(m'+\alpha)}(r^{-(m+\alpha)}R)^e, s' = r'^eT, U = r'\}$$

$$\{C = R^e, s = r^e, \mu = \mathsf{Com}(\alpha), (r^{-(m+\alpha)}, R), \pi_\mu, T^{1/e}, C' = s'^{(m'+\alpha)}(r^{-(m+\alpha)}R)^e, s' = r'^eT, U = r'\}$$