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.

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.

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

K. Kurosawa and G. Hanaoka (Eds.): PKC 2013, LNCS 7778, pp. 55–72, 2013.

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 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) have 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.

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 2 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)[1].

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 significantly 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 [3] 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 [26,6,7], authenticated data structures [25,21,27,30], and the recent work on verifiable computation [3]. Also, other recent works have addressed a slightly different and more practical problem of realizing authenticated remote file systems [29]. However, as pointed out in [3], previous solutions based on accumulators and authenticated data structures either rely on non-constant size assumptions

[1] Precisely, our construction relies on the Square Computational Diffie-Hellman assumption which however has been shown equivalent to the standard CDH [22,1].

(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 [3]. 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 [23][2]. Micali *et al.*'s construction was abstracted away by Chase *et al.* [12], and by Catalano, Dodis and Visconti [8]. 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 [12,8], 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 [23,12]). 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.

[2] 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.

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 [19] 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[3]. Similarly to [10], the scheme of Libert and Yung [19] also relies on a non-constant size assumption in bilinear groups: the q-Diffie-Hellman Exponent [5].

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 [23], 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 [19], 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 [20] to extend ZK-EDB to the (very natural) case of "dynamic" databases. In an updatable ZK-EDB the prover is 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 [20] is given a definition of *Updatable Zero Knowledge (Elementary) Databases* together with a construction based on mercurial commitments and Verifiable Random Functions [24] 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.* [23].

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

Using Vector Commitments, we realize the *first* constructions of "compact" Updatable ZK-EDB whose proofs and updates are much shorter than those of Liskov [20]. 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[4].

Additional Applications of Vector Commitments. We leave to the full version of this paper [9] a description of additional applications of Vector Commitments to compact *Independent Zero-Knowledge Databases* [14], *Fully Dynamic Universal Accumulators* [4,7,17], and *pseudonymous credentials* [16]. Very recently, Libert, Peters and Yung also used our Vector Commitments to improve the efficiency of group signatures with revocation [18].

Preliminaries and Definitions. 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 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 aux.

$\mathsf{VC.Open}_{\mathsf{pp}}(m, i, \mathsf{aux})$ This algorithm is run by the committer to produce a proof Λ_i that m is the i-th committed message.

$\mathsf{VC.Ver}_{\mathsf{pp}}(C, m, i, \Lambda_i)$ 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 $m = m_i$.

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

$\mathsf{VC.Update}_{\mathsf{pp}}(C, m, m', i)$ 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 $\mathsf{VC.Ver}_{\mathsf{pp}}(C, m_i, i, \mathsf{VC.Open}_{\mathsf{pp}}(m_i, i, \mathsf{aux}))$ 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 2. [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 [18] 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.

2.1 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. Roughly speaking, the Square-CDH assumption says that it is computationally infeasible to compute the value g^{a^2}, given $g, g^a \in \mathbb{G}$. This has been shown equivalent to the standard CDH assumption [22,1]. Our construction is reminiscent of the incremental hash function by Bellare and Micciancio [2], 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 $\mathcal{M} = \mathbb{Z}_p$. [5]

$\mathsf{VC.Com_{pp}}(m_1, \ldots, m_q)$ Compute $C = h_1^{m_1} h_2^{m_2} \cdots h_q^{m_q}$ and output C and the auxiliary information $\mathsf{aux} = (m_1, \ldots, m_q)$.

$\mathsf{VC.Open_{pp}}(m_i, i, \mathsf{aux})$ Compute

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

$\mathsf{VC.Ver_{pp}}(C, m_i, i, \Lambda_i)$ If $e(C/h_i^{m_i}, h_i) = e(\Lambda_i, g)$ then output 1. Otherwise output 0.

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

$\mathsf{VC.ProofUpdate_{pp}}(C, \Lambda_j, m', U)$ 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 whose proof appears in the full version.

Theorem 3. *If the CDH assumption holds, then the scheme defined above is a concise vector commitment.*

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

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., verifiable databases and accumulators). 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).

2.2 A Vector Commitment Based on RSA

Here we propose a realization of vector commitments from the RSA assumption.

$\mathsf{VC.KeyGen}(1^k, \ell, q)$ 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 $\mathcal{M} = \{0,1\}^\ell$. [6]

$\mathsf{VC.Com_{pp}}(m_1, \ldots, m_q)$ Compute $C \leftarrow S_1^{m_1} \cdots S_q^{m_q}$ and output C and the auxiliary information $\mathsf{aux} = (m_1, \ldots, m_q)$.

$\mathsf{VC.Open_{pp}}(m, i, \mathsf{aux})$, 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.

$\mathsf{VC.Ver_{pp}}(C, m, i, \Lambda_i)$ 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.

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

$\mathsf{VC.ProofUpdate_{pp}}(C, \Lambda_j, m', i, U)$ 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 $C' = C S_i^{m'-m}$ 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).

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

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

An optimization similar to the one suggested for CDH construction in the previous section applies to this scheme as well, and thus allows the verifier to store only a constant number of elements of the public parameters.

Achieving Constant-Size Public Parameters. In the full version of this work we show a variant of this RSA scheme that achieves *constant-size* public parameters. Very roughly, to do this we borrow some techniques from [15] that introduce a "special" pseudorandom function f that generates prime numbers, and we use such f to compute each prime e_i as $f(i)$. This new scheme is computationally less efficient compared to the other RSA and CDH constructions. Though, it shows that vector commitments with constant-size public parameters exist.

3 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 [3]. 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 [3].

We begin by recalling the definition of Verifiable Databases. Our definition closely follows that in [3] 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. We defer the interested reader to [3] and our full version for a more precise definition.

3.1 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 $\mathtt{aux}$ in S.

The security of our scheme follows from the following theorem whose proof appears in the full version.

Theorem 5. *If* VC *is a vector commitment, then the Verifiable Database scheme described above is secure.*

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 2.1 and 2.2 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 2 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.

4 (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*. A precise description of such requirements can be found in [11]. 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.* 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.

4.1 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 $\mathsf{aux} = (\mathsf{aux}_{\mathsf{VC}}, m_1, C_1, \mathsf{aux}^1_{\mathsf{TMC}}, \ldots, m_q, C_q, \mathsf{aux}^q_{\mathsf{TMC}})$.

$\mathsf{qSOpen}_{pk}(m_i, i, \mathsf{flag}, \mathsf{aux})$**.** 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 (whose proof appears in the full version).

Theorem 6. *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.*

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 [23]), one gets a qTMC based on CDH whose efficiency is roughly the same as that of the scheme in [19] 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 [22]), 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 [13,28] 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 [19]).

4.2 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.

This problem was studied by Liskov in [20] where he showed how to build Updatable ZK EDB by appropriately modifying the basic approach of combining Merkle trees and mercurial commitments. 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 [20].

In [20] 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 [23].

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 [20] 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 $q > 2$). For lack of space, we defer the interested reader to [20] for formal definitions of (basic) updatable mercurial commitments.

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

$\mathsf{qUpdate}_{pk}(C, \mathsf{aux}, m', i)$**.** This algorithm is run by the committer who produced C (and holds the corresponding 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 by extending them to the case of sequences of q messages.

Updatable q-mercurial Commitments from Vector Commitments. In this section 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 4.1 augmented with the following update algorithms:

$\mathsf{qUpdate}_{pk}(C, \mathsf{aux}, m', i)$**.** Parse aux as $(m_1, C_1, \mathsf{aux}^1_{\mathsf{uTMC}}, \ldots, m_q, C_q, \mathsf{aux}^q_{\mathsf{uTMC}}, \mathsf{aux}_{\mathsf{VC}})$, extract $(C_i, m_i, \mathsf{aux}^i_{\mathsf{uTMC}})$ from it and run $(C'_i, U_{\mathsf{uTMC}}) \leftarrow \mathsf{Update}_{pk_{\mathsf{uTMC}}}(C_i, \mathsf{aux}^i_{\mathsf{uTMC}}, m')$. 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 7. *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.*

The proof is very similar to that of Theorem 6 and is omitted here.

References

1. Bao, F., Deng, R.H., Zhu, H.: Variations of Diffie-Hellman Problem. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 301–312. Springer, Heidelberg (2003)
2. Bellare, M., Micciancio, D.: A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 163–192. Springer, Heidelberg (1997)
3. Benabbas, S., Gennaro, R., Vahlis, Y.: Verifiable Delegation of Computation over Large Datasets. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 111–131. Springer, Heidelberg (2011)
4. Benaloh, J.C., de Mare, M.: One-Way Accumulators: A Decentralized Alternative to Digital Signatures (Extended Abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 274–285. Springer, Heidelberg (1994)
5. Boneh, D., Gentry, C., Waters, B.: Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 258–275. Springer, Heidelberg (2005)
6. Camenisch, J., Kohlweiss, M., Soriente, C.: An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 481–500. Springer, Heidelberg (2009)
7. Camenisch, J., Lysyanskaya, A.: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
8. Catalano, D., Dodis, Y., Visconti, I.: Mercurial Commitments: Minimal Assumptions and Efficient Constructions. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 120–144. Springer, Heidelberg (2006)
9. Catalano, D., Fiore, D.: Vector commitments and their applications. Cryptology ePrint Archive (2011), `http://eprint.iacr.org/2011/495`
10. Catalano, D., Fiore, D., Messina, M.: Zero-Knowledge Sets with Short Proofs. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 433–450. Springer, Heidelberg (2008)
11. Catalano, D., Di Raimondo, M., Fiore, D., Messina, M.: Zero-knowledge sets with short proofs. IEEE Transactions on Information Theory 57(4), 2488–2502 (2011)
12. Chase, M., Healy, A., Lysyanskaya, A., Malkin, T., Reyzin, L.: Mercurial Commitments with Applications to Zero-Knowledge Sets. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 422–439. Springer, Heidelberg (2005)
13. Cheon, J.H.: Security Analysis of the Strong Diffie-Hellman Problem. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 1–11. Springer, Heidelberg (2006)
14. Gennaro, R., Micali, S.: Independent Zero-Knowledge Sets. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 34–45. Springer, Heidelberg (2006)
15. Hohenberger, S., Waters, B.: Short and Stateless Signatures from the RSA Assumption. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 654–670. Springer, Heidelberg (2009)
16. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010)
17. Li, J., Li, N., Xue, R.: Universal Accumulators with Efficient Nonmembership Proofs. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 253–269. Springer, Heidelberg (2007)

18. Libert, B., Peters, T., Yung, M.: Group Signatures with Almost-for-Free Revocation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 571–589. Springer, Heidelberg (2012)
19. Libert, B., Yung, M.: Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 499–517. Springer, Heidelberg (2010)
20. Liskov, M.: Updatable Zero-Knowledge Databases. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 174–198. Springer, Heidelberg (2005)
21. Martel, C.U., Nuckolls, G., Devanbu, P.T., Gertz, M., Kwong, A., Stubblebine, S.G.: A general model for authenticated data structures. Algorithmica 39(1), 21–41 (2004)
22. Maurer, U.M., Wolf, S.: Diffie-Hellman Oracles. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 268–282. Springer, Heidelberg (1996)
23. Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: 44th FOCS, pp. 80–91. IEEE Computer Society Press (October 2003)
24. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS, pp. 120–130. IEEE Computer Society Press (October 1999)
25. Naor, M., Nissim, K.: Certificate revocation and certificate update. In: Proceedings of the 7th Conference on USENIX Security Symposium, vol. 7, p. 17 (1998)
26. Nguyen, L.: Accumulators from Bilinear Pairings and Applications. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 275–292. Springer, Heidelberg (2005)
27. Papamanthou, C., Tamassia, R.: Time and Space Efficient Algorithms for Two-Party Authenticated Data Structures. In: Qing, S., Imai, H., Wang, G. (eds.) ICICS 2007. LNCS, vol. 4861, pp. 1–15. Springer, Heidelberg (2007)
28. Sakemi, Y., Hanaoka, G., Izu, T., Takenaka, M., Yasuda, M.: Solving a Discrete Logarithm Problem with Auxiliary Input on a 160-Bit Elliptic Curve. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 595–608. Springer, Heidelberg (2012)
29. Stefanov, E., van Dijk, M., Oprea, A., Juels, A.: Iris: A scalable cloud file system with efficient integrity checks. Cryptology ePrint Archive, Report 2011/585 (2011), `http://eprint.iacr.org/`
30. Tamassia, R., Triandopoulos, N.: Certification and authentication of data structures. In: Alberto Mendelzon Workshop on Foundations of Data Management (2010)