Nearly Optimal Verifiable Data Streaming

Johannes Krupp[1], Dominique Schröder[1], Mark Simkin[1], Dario Fiore[2], Giuseppe Ateniese[3,4], and Stefan Nuernberger[1]

[1] Saarland University, CISPA, Germany
{krupp,ds,simkin}@ca.cs.uni-saarland.de
nuernberger@cs.uni-saarland.de
[2] IMDEA Software Institute, Madrid, Spain
dario.fiore@imdea.org
[3] Sapienza - University of Rome, Italy
[4] Johns Hopkins University, USA
ateniese@cs.jhu.edu

Abstract The problem of verifiable data streaming (VDS) considers a client with limited computational and storage capacities that streams an a-priori unknown number of elements to an untrusted server. The client may retrieve and update any outsourced element. Other parties may verify each outsourced element's integrity using the client's public-key. All previous VDS constructions incur a bandwidth and computational overhead on both client and server side, which is at least logarithmic in the number of transmitted elements. We propose two novel, fundamentally different approaches to constructing VDS. The first scheme is based on a new cryptographic primitive called Chameleon Vector Commitment (CVC). A CVC is a trapdoor commitment scheme to a vector of messages where both commitments and openings have constant size. Using CVCs we construct a tree-based VDS protocol that has constant computational and bandwidth overhead on the client side. The second scheme shifts the workload to the server side by combining signature schemes with cryptographic accumulators. Here, all computations are constant, except for queries, where the computational cost of the server is linear in the total number of updates.

1 Introduction

In this work we study the problem of verifiable data streaming (VDS) [23,24], where a resource-constrained device outsources large amounts of data to a possibly untrusted cloud storage provider in such a manner, that the following three properties are maintained. First, the data is streamed from the client to the cloud in a unidirectional fashion; namely, adding an element to the remote storage consists of a single message including the data element and an authentication information (we call this property *unidirectional upload*). The current state of the remote storage is succinctly represented by a verification key pk maintained by the client. Second, the client must be able to update any element in the remote storage efficiently. In this case, any update results in a new verification key pk', which invalidates the old data element so as to prevent rollback (aka replay) attacks. In contrast, whenever a new element is streamed, the public key remains unchanged (*Updatability* and *Freshness*). Third, the data owner, or any other client trusting the data owner, should be able to retrieve arbitrary subsets of the outsourced data along with the corresponding proof and verify their integrity using the data owner's public key (*Public Verifiability*). Notably, such a verification must guarantee that elements have not been altered and are maintained in the correct position in the stream (*Integrity* and *Order Enforcement*).

Various applications for VDS protocols have been discussed in [23,24]. One further interesting application for VDS protocols are wireless sensor networks, where a number of wireless sensors constantly monitor their surrounding environment and stream the resulting data to a central server. These sensors usually only have very limited computational capabilities and are powered by batteries. For such applications it is of key importance to reduce the computational and bandwidth overhead, incurred by the cryptographic primitives on the client side, to a bare minimum. In this work we propose novel solutions to the problem described above that are asymptotically and practically faster than all known previous constructions.

	Un-bounded	Stand. Model	Assump.	Stream time/space	Retrieve time/space	Verification time/space	Update time/space	Size π	Size pk
[23]	✗	✓	Dlog	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(\log_2 M)$	$\mathcal{O}(1)$
[24]	✓	✗	Dlog	$\mathcal{O}(\log_2 N)$	$\mathcal{O}(\log_2 N)$	$\mathcal{O}(\log_2 N)$	$\mathcal{O}(\log_2 N)$	$\mathcal{O}(\log_2 N)$	$\mathcal{O}(1)$
CVC	✓	✓	CDH	$\mathcal{O}(1)$	$\mathcal{O}(\log_q N)$	$\mathcal{O}(\log_q N)$	$\mathcal{O}(\log_q N)$	$\mathcal{O}(\log_q N)$	q^2
ACC	✓	✓	q-strong DH	$\mathcal{O}(1)$	$\mathcal{O}(U)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Table 1. Comparison of the different constructions. Unbounded indicates whether the construction can authenticate an unbounded amount of elements. M is an upper bound on the number of streamed values, N denotes the number of elements streamed so far, and U is the number of updates. The base of all logarithms is stated explicitly to highlight the hidden factors.

1.1 Our Contribution

We propose two novel and fundamentally different approaches to VDS that result in the first practical verifiable data streaming protocols in the standard model in which the length of the stream does not have to be bounded a-priori (see Table 1 for a comparison). Somewhat surprisingly, our schemes are asymptotically more efficient even compared with solutions in the random oracle model [24].

1. Our first scheme is based on a new cryptographic primitive called *Chameleon Vector Commitments* (CVCs) which extends Vector Commitments (VCs) [6]. Informally, a VC allows anyone to commit to a vector of messages and to prove that a certain message m is at a given position i. VCs satisfy two main properties: *conciseness* guarantees that the size of the commitment as well as the size of the proofs is constant, no matter of how large the vector is; *position binding* guarantees that it is not possible to create two valid proofs for two different messages $m \neq m'$ for one position i. Our new notion of CVCs extends VCs with the additional property that the owner of a trapdoor can open the commitment, at any vector position, to an arbitrary message.
   We use CVCs in combination with a novel approach of building tree-based VDS protocols to provide the first unbounded VDS construction in the standard model that is *asymptotically faster* than all previously known constructions.
   We believe that CVCs may also be of independent interest in other contexts and we will provide a comprehensive formal treatment of this primitive in section 3.
2. Our second VDS scheme shifts the workload to the server side by combining signature schemes with cryptographic accumulators [15]. The basic idea of our scheme is to let the client sign all elements in the stream. Whenever the client wishes to update an element, it "revokes" the signature on the old value by adding it to the accumulator that is part of the public key. Whenever the client queries an outsourced element, the server has to perform computation that is linear in the number of updates to compute a proof of correctness.[5] This scheme is *asymptotically optimal* w.r.t. to all operations apart from querying of elements.
   We have instantiated this construction with the accumulator based on the q-strong Diffie-Hellman assumption in bilinear groups [15] in a nonblack-box way to obtain a public key of size $\mathcal{O}(1)$. That is, in the original construction of [15], the public key consists of U elements, where U is an upper bound on the number of accumulated values. In our case, we split the key between the client and the server to obtain a scheme that supports an unbounded number of updates but still maintain constant-size public key.
3. We evaluated our schemes based on a full implementation in Java running on Amazon EC2. Our experiments show that the two constructions achieve practical performances. For instance, using our accumulator-based VDS scheme a client can stream a data block of 256KB by spending 2.3ms in authentication and by sending 236 more bytes. Retrieving and verifying the authenticity of about 8GB of data can be performed in about 10 minutes.

1.2 Related Work

Verifiable data streaming is a generalization of verifiable databases (VDB) [2] and was first introduced by Schröder and Schröder [23], who also presented a construction for an *a-priori fixed* number of elements

[5] This cost can even be made sublinear using techniques from Papamanthou, Tamassia, and Triandopoulos [19].

M. The computational and bandwidth overhead of all their operations was logarithmic in M. Recently, Schröder and Simkin suggested the first VDS protocol that supports streams of unbounded length [24], but is only provably secure in the random oracle model. Their overhead is logarithmic in the number of already outsourced elements. In Table 1, we compare previous constructions with ours.

VDBs were first introduced by Benabbas, Gennaro, and Vahlis [2] with the main difference, compared to VDS, being that the size of the database is already defined during the setup phase, while in a streaming protocol it is unknown. Furthermore, in a VDS protocol, data can be added to the database *non-interactively* by sending a single message to the server. Notably this message does not affect the verification key of the database. VDBs have been extensively investigated in the context of accumulators [16,4,5] and authenticated data structures [14,12,18,26]. More recent works, such as [2] or [6], usually only support a polynomial number of values instead of exponentially many, and the scheme of [2] is not publicly verifiable.

Other works like streaming authenticated data structures by Papamanthou et al. [17] or the Iris cloud file system [25] consider untrusted cloud storage provider, but require key updates after each streamed element.

"Pure" streaming protocols between a sender and possibly multiple clients, such as TESLA and their variants, e.g., [22,21], require the sender and receiver to be loosely synchronized and these protocols do not offer public verifiability. The signature based solution of [22] does not support efficient updates.

1.3 Straw Man Approaches

One idea to construct a VDS protocol may be to use a Merkle tree with a signed root value. That is, for each outsourced element we recompute the Merkle tree and sign the new root. Whenever we update some outsourced element, we recompute the root node's value, pick a new key pair for the signature scheme, and sign the new root node value under the new key pair. Unfortunately, this approach requires one to recompute the root node of the complete existing tree at each insertion. This is not a feasible approach for resource-constrained devices and is asymptotically slower than the constructions we present. Another idea might be to use digital signatures to sign elements in an interleaved fashion during outsourcing. While this approach allows unidirectional upload, integrity, and order enforcement, it is not clear how to update an already outsourced element efficiently. Simply signing the new element is not sufficient, since the old element is not invalidated, i.e., upon a query for the updated index, the server can simply return the old element instead of the new one. Forward-secure signature schemes [11] are not applicable here either. Either the root of this tree should be part of the public-key, which would violate the unidirectional uploading property, or if we do not update the public key after updating a data element, rollback attacks would be possible, i.e., it would violate the freshness property.

2 Preliminaries

In this section we define our notation and describe some cryptographic primitives and assumptions used in this work. The security parameter is denoted by λ. $a||b$ refers to an encoding of that allows to uniquely recover the strings a and b. If $A(x;r)$ is an efficient (possibly randomized) algorithm, then $y \leftarrow A(x;r)$ refers to running A on input x using randomness r and assigning the output to y.

2.1 Bilinear Maps

Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic multiplicative groups of prime order p, generated by g_1 and g_2, respectively. Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ be a bilinear pairing with the following properties:

1. Bilinearity $e(P^a, Q^b) = e(P, Q)^{ab}$ for all $P \in \mathbb{G}_1$ and all $Q \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$;
2. Non-degeneracy $e(g_1, g_2) \neq 1$;
3. Computability: There exists an efficient algorithm to compute $e(P, Q)$ for all $P \in \mathbb{G}_1$ and all $Q \in \mathbb{G}_2$.

If $\mathbb{G}_1 = \mathbb{G}_2$ then the bilinear map is called *symmetric*. A *bilinear instance generator* is an efficient algorithm that takes as input the security parameter 1^λ and outputs a random tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$.

2.2 Computational Assumptions

Now we briefly recall the definitions of the Computational Diffie-Hellman (CDH) assumption, the Square-Computational Diffie-Hellman (Square-CDH), and the q-strong Diffie-Hellman assumptions.

Assumption 1 (CDH). *Let $\mathbb{G}$ be a group of prime order p, let $g \in \mathbb{G}$ be a generator, and let a, b be two random elements from $\mathbb{Z}_p$. The Computational Diffie-Hellman assumption holds in $\mathbb{G}$ if for every PPT adversary $\mathcal{A}$ the probability* $\text{Prob}\left[\mathcal{A}(g, g^a, g^b) = g^{ab}\right]$ *is negligible in λ.*

Assumption 2 (Square-CDH assumption). *Let $\mathbb{G}$ be a group of prime order p, let $g \in \mathbb{G}$ be a generator, and let a be a random element from $\mathbb{Z}_p$. The Square Computational Diffie-Hellman assumption holds in $\mathbb{G}$ if for every PPT adversary $\mathcal{A}$ the probability* $\text{Prob}\left[\mathcal{A}(g, g^a) = g^{a^2}\right]$ *is negligible in λ.*

It is worth noting that the Square-CDH assumption has been shown equivalent to the standard CDH assumption [1,13].

Assumption 3 (q-strong DH assumption). *Let $(p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a tuple generated by a bilinear instance generator, and let s be randomly chosen in $\mathbb{Z}_p^*$. We say that the q-Strong DH (q-SDH) assumption holds if every PPT algorithm $\mathcal{A}(p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, g^{s^2}, \ldots, g^{s^q})$ has negligible probability (in λ) of returning a pair $(c, g^{1/(s+c)}) \in \mathbb{Z}_p^* \times \mathbb{G}$.*

2.3 Chameleon Hash Functions

A chameleon hash function is a randomized collision-resistant hash function that provides a trapdoor to find collisions. This means that given the trapdoor *csk*, a message m, some randomness r and another message m', it is possible to find a randomness r' s.t. $\mathsf{Ch}(m; r) = \mathsf{Ch}(m'; r')$.

Definition 1 (Chameleon Hash Function). *A* chameleon hash function *is a tuple of PPT algorithms $\mathcal{CH} = (\mathsf{CHGen}, \mathsf{Ch}, \mathsf{Col})$:*

$\mathsf{CHGen}(1^\lambda)$**:** *The key generation algorithm returns a keypair (csk, cpk) and sets $\mathsf{Ch}(\cdot) := \mathsf{Ch}(cpk, \cdot)$.*

$\mathsf{Ch}(m; r)$**:** *The input of the hashing algorithm is a message m and some randomness $r \in \{0,1\}^\lambda$ and it outputs a hash value.*

$\mathsf{Col}(csk, m, r, m')$**:** *Upon input of the trapdoor csk, a message m, some randomness r and another message m', the collision finding algorithm returns some randomness r' s.t. $\mathsf{Ch}(m; r) = \mathsf{Ch}(m'; r')$.*

Uniform Distribution: *The output of Ch is uniformly distributed, thus the output of $\mathsf{Ch}(m; r)$ is independent of m. Furthermore, the output of $\mathsf{Col}(csk, m, r, m')$ also has the same distribution as r itself.*

A chameleon hash is required to be collision-resistant, i.e., no PPT adversary should be able to find (m, r) and (m', r') s.t. $(m, r) \neq (m', r')$ and $\mathsf{Ch}(m; r) = \mathsf{Ch}(m'; r')$.

Definition 2 (Collision Resistance). *A chameleon hash $\mathcal{CH}$ is* collision-resistant *if the success probability for any PPT adversary $\mathcal{A}$ in the following game is negligible in λ:*

Experiment $\mathsf{HashCol}_{\mathcal{A}}^{\mathcal{CH}}(\lambda)$
$(csk, cpk) \leftarrow \mathsf{CHGen}(1^\lambda)$
$(m, m', r, r') \leftarrow \mathcal{A}(\mathsf{Ch})$
if $\mathsf{Ch}(m; r) = \mathsf{Ch}(m'; r')$ *and* $(m, r) \neq (m', r')$ *output* 1
else output 0

2.4 Vector Commitments

A vector commitment is a commitment to an ordered sequence of messages, which can be opened at each position individually. Furthermore, a vector commitment provides an interface to update single messages and their openings.

Definition 3 (Vector Commitment). *A* vector commitment *is a tuple of PPT algorithms* $\mathcal{VC} = (\mathsf{VGen}, \mathsf{VCom}, \mathsf{VOpen}, \mathsf{VVer}, \mathsf{VUpdate}, \mathsf{VProofUpdate})$*:*

$\mathsf{VGen}(1^\lambda, q)$**:** *The key generation algorithm gets as input the security parameter* λ *and the size of the vector* q *and outputs some public parameters* pp*.*

$\mathsf{VCom}_{\mathsf{pp}}(m_1, \ldots, m_q)$**:** *Given* q *messages, the commitment algorithm returns a commitment* C *and some auxiliary information* aux*, which will be used for proofs and updates.*

$\mathsf{VOpen}_{\mathsf{pp}}(m, i, \mathsf{aux})$**:** *The opening algorithm takes as input a message* m*, a position* i *and some auxiliary information* aux *and returns a proof* Λ_i *that* m *is the message at position* i*.*

$\mathsf{VVer}_{\mathsf{pp}}(C, m, i, \Lambda_i)$**:** *The verification algorithm outputs 1 only if* Λ_i *is a valid proof that* C *was created to a sequence with* m *at position* i*.*

$\mathsf{VUpdate}_{\mathsf{pp}}(C, m, m', i)$**:** *The update algorithm allows to change the* i*-th message in* C *from* m *to* m'*. It outputs an updated commitment* C' *and some update-information* U*.*

$\mathsf{VProofUpdate}_{\mathsf{pp}}(C, \Lambda_j, m', i, U)$**:** *The proof-update algorithm may be run by anyone holding a proof* Λ_j *that is valid w.r.t.* C *to obtain the updated commitment* C' *and an updated proof* Λ'_j *that is valid w.r.t.* C'*.*

The security definition of vector commitments requires a vector commitment to be position-binding, i.e., no PPT adversary $\mathcal{A}$ given pp can open a commitment to two different messages at the same position. Formally:

Definition 4 (Position-Binding). *A vector commitment is* position-binding *if the success probability for any PPT adversary* $\mathcal{A}$ *in the following game is negligible in* λ*:*

Experiment $\mathsf{PosBdg}^{\mathcal{VC}}_{\mathcal{A}}(\lambda)$
$\mathsf{pp} \leftarrow \mathsf{VGen}(1^\lambda, q)$
$(C, m, m', i, \Lambda, \Lambda') \leftarrow \mathcal{A}(\mathsf{pp})$
if $m \neq m' \wedge \mathsf{VVer}_{\mathsf{pp}}(C, m, i, \Lambda) \wedge \mathsf{VVer}_{\mathsf{pp}}(C, m', i, \Lambda')$ *output* 1
else output 0*.*

2.5 The Bilinear-Map Accumulator

We briefly recall the accumulator based on bilinear maps first introduced by Nguyen [15]. For simplicity, we describe the accumulator using symmetric bilinear maps. A version of the accumulator using asymmetric pairings can be obtained in a straightforward way, and our implementation does indeed work on asymmetric MNT curves.

For some prime p, the scheme accumulates a set $\mathcal{E} = \{e_1, \ldots, e_n\}$ of elements from $\mathbb{Z}_p^*$ into an element $f'(\mathcal{E})$ in $\mathbb{G}$. Damgård and Triandopoulos [7] extended the construction with an algorithm for issuing constant size proofs of non-membership, i.e., proofs that an element $e \notin \mathcal{E}$. The public key of the accumulator is a tuple of elements $\{g^{s^i} | 0 \leq i \leq q\}$, where q is an upper bound on $|\mathcal{E}| = n$ that grows polynomially with the security parameter $\lambda = \mathcal{O}(\log p)$. The corresponding secret key is s. More precisely, the accumulated value $f'(\mathcal{E})$ is defined as

$$f'(\mathcal{E}) = g^{(e_1+s)(e_2+s)\ldots(e_n+s)}.$$

The *proof of membership* is a witness A_{e_i} which shows that an element e_i belongs to the set $\mathcal{E}$ and it is computed as

$$A_{e_i} = g^{\prod_{e_j \in \mathcal{E}: e_j \neq e_i} (e_j+s)}.$$

Given the witness A_{e_i}, the element e_i, and the accumulated values $f'(\mathcal{E})$, the verifier can check that

$$e(A_{e_i}, g^{e_i} \cdot g^s) = e(f'(\mathcal{E}), g).$$

The proof of *non-membership*, which shows that $e_i \notin \mathcal{E}$, consists of a pair of witnesses $\hat{w} = (w, u) \in \mathbb{G} \times \mathbb{Z}_p^*$ with the requirement that: $(i)\, u \neq 0$, and $(ii)\, (e_i + s) | [\prod_{e \in \mathcal{E}} (e + s) + u]$. The verification algorithm in this case checks that

$$e(w, g^{e_i} \cdot g^s) = e(f'(\mathcal{E}) \cdot g^u, g).$$

The authors also show that the proof of non-membership can be computed efficiently without knowing the trapdoor s. The security of this construction relies on the q-strong Diffie-Hellman assumption. In particular, Nguyen showed that the accumulator is collision-resistant under the q-SDH assumption:

Lemma 1 ([15]). *Let λ be the security parameter and $t = (p, \mathbb{G}, \mathbb{G}_T, e, g)$ be a tuple of bilinear pairings parameter. Under the q-SDH assumption, given a set of elements $\mathcal{E}$, the probability that, for some s chosen at random in $\mathbb{Z}_p^*$, any efficient adversary $\mathcal{A}$, knowing only $t, g, g^s, g^{s^2}, \ldots, g^{s^q}$ $(q \geq |\mathcal{E}|)$, can find a set $\mathcal{E}' \neq \mathcal{E}$ $(q \geq |\mathcal{E}'|)$ such that $f'(\mathcal{E}') = f'(\mathcal{E})$ is negligible.*

We recall the security claim for the non-membership test:

Lemma 2 ([7]). *Under the q-SDH assumption, for any set $\mathcal{E}$ there exists a unique non-membership witness with respect to the accumulated value $f'(\mathcal{E})$ and a corresponding efficient and secure proof of non-membership verification test.*

2.6 Verifiable Data Streaming

A VDS protocol [23] allows a client, who possesses a private key, to store a large amount of ordered data $d_1, d_2, \ldots$ on a server in a verifiable manner, i.e., the server can neither modify the stored data nor append additional data. Furthermore, the client may ask the server about data at a position i, who then has to return the requested data d_i along with a publicly verifiable proof $\tilde{\pi}_i$, which proves that d_i was actually stored at position i. Formally, a VDS protocol is defined as follows:

Definition 5 (Verifiable Data Streaming). *A* verifiable data streaming *protocol $\mathcal{VDS} = (\mathsf{Setup}, \mathsf{Append}, \mathsf{Query}, \mathsf{Verify}, \mathsf{Update})$ is a protocol between a client $\mathcal{C}$ and a server $\mathcal{S}$, which are both PPT algorithms. The server holds a database DB.*

$\mathsf{Setup}(1^\lambda)$**:** *The setup algorithm takes as input the security parameter and generates a keypair (pk, sk), gives the public verification key pk to the server $\mathcal{S}$ and the secret key sk to the client $\mathcal{C}$.*

$\mathsf{Append}(sk, d)$**:** *The append protocol takes as input the secret key and some data d. During the protocol, the client $\mathcal{C}$ sends a single message to the server $\mathcal{S}$, who will then store the new item d in DB. This protocol may output a new secret key sk' to the client, but the public key does not change.*

$\mathsf{Query}(pk, DB, i)$**:** *The query protocol runs between $\mathcal{S}(pk, DB)$ and $\mathcal{C}(i)$. At the end the client will output the i-th entry of the database DB along with a proof $\tilde{\pi}_i$.*

$\mathsf{Verify}(pk, i, d, \tilde{\pi}_i)$**:** *The verification algorithm outputs d, iff d is the i-th element in the database according to $\tilde{\pi}_i$. Otherwise it outputs $\perp$.*

$\mathsf{Update}(pk, DB, sk, i, d')$**:** *The update protocol runs between $\mathcal{S}(pk, DB)$ and $\mathcal{C}(i, d')$. At the end, the server will update the i-th entry of its database DB to d' and both parties will update their public key to pk'. The client may also update his secret key to sk'.*

Intuitively the security of a VDS protocol demands that an attacker should not be able to modify stored elements nor should he be able to add further elements to the database. In addition the elements should be fresh, meaning that outdated elements do not verify. This can be formalized in the following game VDSsec:

Setup: First, the challenger generates a keypair $(sk, pk) \leftarrow \mathsf{Setup}(1^\lambda)$. It sets up an empty database DB and gives the public key vp to the adversary $\mathcal{A}$.

Streaming: In this adaptive phase, the adversary $\mathcal{A}$ can add new data by giving some data d to the challenger, which will then run $(sk', i, \tilde{\pi}_i) \leftarrow \mathsf{Append}(sk, d)$ to append d to its database. The challenger then returns $(i, \tilde{\pi}_i)$ to the adversary. $\mathcal{A}$ may also update existing data by giving a tuple (d', i) to the challenger, who will then run the update protocol $\mathsf{Update}(pk, DB, sk, i, d')$ with the adversary $\mathcal{A}$. The challenger will always keep the latest public key pk^* and a ordered sequence of the database $Q = \{(d_1, 1), \ldots, (d_{q(\lambda)}, q(\lambda))\}$.

Output: To end the game, the adversary $\mathcal{A}$ can output a tuple $(d^*, i^*, \hat{\pi})$. Let $\hat{d} \leftarrow \mathsf{Verify}(pk^*, i^*, d^*, \hat{\pi})$. The adversary wins iff $\hat{d} \neq \perp$ and $(\hat{d}, i^*) \notin Q$.

Definition 6 (Secure VDS). *A VDS protocol is* secure*, if the success probability of any PPT adversary in the above game* VDSsec *is at most negligible in* λ.

3 Chameleon Vector Commitments

In this section we introduce chameleon vector commitments (CVCs). CVCs extend the notion of vector commitments [9,6] in the sense that a CVC (like a VC) allows one to commit to an ordered sequence of messages in such a way that: it is possible to open each position individually, and the commitment value as well as the openings are concise, i.e., of size independent of the length of the message vector. In addition, we require CVCs to satisfy a novel chameleon property saying that a CVC can come with a trapdoor with which one can replace messages at individual positions *without* changing the commitment value.

3.1 Defining CVCs

We define CVCs as a tuple of seven efficient algorithms: a key generation algorithm CGen to compute a set of public parameters and a trapdoor, a commitment algorithm CCom to commit to a vector of messages, an opening algorithm COpen to open a position of a commitment, a collision finding algorithm CCol that uses the trapdoor to output the necessary information for opening a commitment to a different value, an updating algorithm CUpdate to update the values in a commitment without recomputing the entire commitment, a proof update algorithm CProofUpdate to update proofs accordingly, and a verification algorithm CVer to verify the correctness of an opening w.r.t. a commitment.

Definition 7 (Chameleon Vector Commitment). *A* chameleon vector commitment *is a tuple of PPT algorithms* $\mathcal{CVC} = (\mathsf{CGen}, \mathsf{CCom}, \mathsf{COpen}, \mathsf{CVer}, \mathsf{CCol}, \mathsf{CUpdate}, \mathsf{CProofUpdate})$ *working as follows:*

Key Generation $\mathsf{CGen}(1^\lambda, q)$**:** *The key generation algorithm takes as inputs the security parameter* λ *and the vector size* q*. It outputs some public parameters* pp *and a trapdoor* td*.*

Committing $\mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_q)$**:** *On input of a list of* q *ordered messages, the committing algorithm returns a commitment* C *and some auxiliary information* aux*.*

Opening $\mathsf{COpen}_{\mathsf{pp}}(i, m, \mathsf{aux})$**:** *The opening algorithm returns a proof* π *that* m *is the* i*-th committed message in a commitment corresponding to* aux*.*

Verification $\mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi)$**:** *The verification algorithm returns 1 iff* π *is a valid proof that* C *was created on a sequence of messages with* m *at position* i*.*

Collision finding $\mathsf{CCol}_{\mathsf{pp}}(C, i, m, m', \mathsf{td}, \mathsf{aux})$**:** *The collision finding algorithm returns a new auxiliary information* aux' *such that the pair* (C, aux') *is indistinguishable from the output of* $\mathsf{CCom}_{\mathsf{pp}}$ *on a vector of* q *messages with* m' *instead of* m *at position* i*.*

Updating $\mathsf{CUpdate}_{\mathsf{pp}}(C, i, m, m')$**:** *The update-algorithm allows to update the* i*-th message from* m *to* m' *in the commitment* C*. It outputs a new commitment* C' *and an update information* U*, which can be used to update both* aux *and previously generated proofs.*

Updating Proofs $\mathsf{CProofUpdate}_{\mathsf{pp}}(C, \pi_j, i, U)$**:** *The proof-update-algorithm allows to update a proof* π_j *that is valid for position* j *w.r.t.* C *to a new proof* π'_j *that is valid w.r.t.* C' *using the update information* U*.*

A tuple $\mathcal{CVC}$ of algorithms as defined above is a chameleon vector commitment if it is *correct*, *concise* and *secure*. Conciseness is a property about the communication efficiency of CVCs which is defined as follows:

Definition 8 (Concise). *A $\mathcal{CVC}$ is* concise*, if both the size of the commitment C and the size of the proofs π_i are independent of the vector size q.*

Informally, correctness guarantees that a CVC works as expected when its algorithms are honestly executed. A formal definition follows:

Definition 9 (Correctness). *A $\mathcal{CVC}$ is* correct *if for all $q = poly(\lambda)$, all honestly generated parameters $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{CGen}(1^\lambda, q)$ and all messages $(m_1, \ldots, m_q)$, if $(C, \mathsf{aux}) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_q)$ and $\pi \leftarrow \mathsf{COpen}_{\mathsf{pp}}(i, m, \mathsf{aux})$, then the verification algorithm $\mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi)$ outputs 1 with overwhelming probability. Furthermore, correctness must hold even after some updates occur. Namely, considering the previous setting, any message m' and any index i, if $(C', U) \leftarrow \mathsf{CUpdate}_{\mathsf{pp}}(C, i, m_i, m')$ and $\pi' \leftarrow \mathsf{CProofUpdate}_{\mathsf{pp}}(C, \pi, i, U)$, then the verification algorithm $\mathsf{CVer}_{\mathsf{pp}}(C', i, m', \pi')$ must output 1 with overwhelming probability*

Security of CVCs Finally, we discuss the security of CVCs which is defined by three properties: *indistinguishable collisions*, *position binding* and *hiding*. Informally speaking, a CVC has indistinguishable collisions if one is not able to find out if the collision finding algorithm had been used or not. Secondly, a scheme satisfies position binding if, without knowing the trapdoor, it is not possible to open a position in the commitment in two different ways. Thirdly, hiding guarantees that the commitment does not leak any information about the messages that the commitment was made to. In what follows we provide formal definitions of these properties.

Indistinguishable Collisions. This is the main novel property of CVCs: one can use the trapdoor to change a message in the commitment without changing the commitment itself. Intuitively, however, when seeing proofs, one should not be able to tell whether the trapdoor has been used or not. We call this notion indistinguishable collisions, and we formalize it in the following game. Observe that we require the indistinguishability to hold even when having knowledge of the trapdoor.

Definition 10 (Indistinguishable Collisions). *A $\mathcal{CVC}$ has* indistinguishable collisions *if the success probability of any stateful PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ in the game $\mathsf{CollInd}$ is only negligibly bigger than 1/2 in λ.*

Experiment $\mathsf{CollInd}^{\mathcal{CVC}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{CGen}(1^\lambda, q)$
$b \leftarrow \{0, 1\}$
$((m_1, \ldots, m_q), (i, m'_i)) \leftarrow \mathcal{A}_0(\mathsf{pp}, \mathsf{td})$
$(C_0, \mathsf{aux}^*) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_i, \ldots, m_q)$
$\mathsf{aux}_0 \leftarrow \mathsf{CCol}_{\mathsf{pp}}(C_0, i, m_i, m'_i, \mathsf{td}, \mathsf{aux}^*)$
$(C_1, \mathsf{aux}_1) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m'_i, \ldots, m_q)$
$b' \leftarrow \mathcal{A}_1(C_b, \mathsf{aux}_b)$
if $b = b'$output 1
else output 0

Position-binding This property aims to capture that, without knowing the trapdoor, one should not be able to open the same position of a chameleon vector commitment to two different messages. In particular, we consider a strong definition of this notion in which the adversary is allowed to use an oracle CCol for computing collisions. Namely, even when seeing collisions in some of the positions, an adversary must not find two different openings for other positions. This notion, called position-binding, is formalized as follows:

Definition 11 (Position-Binding). *A $\mathcal{CVC}$ satisfies* position-binding *if no PPT adversary $\mathcal{A}$ can output two valid proofs for different messages (m, m') at the same position i with non-negligible probability. Formally, the success probability af any PPT adversary $\mathcal{A}$ in the following game PosBdg should be negligible in λ. Whenever the adversary queries the collision oracle with a commitment, a position, two messages and some auxiliary information $(C, i, m, m', \mathsf{aux})$, the game runs the collision finding algorithm $\mathsf{aux}' \leftarrow \mathsf{CCol}_{\mathsf{pp}}(C, i, m, m', \mathsf{td}, \mathsf{aux})$ and returns aux' to the adversary.*

Experiment $\mathsf{PosBdg}^{\mathcal{CVC}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pp}, \mathtt{td}) \leftarrow \mathsf{CGen}(1^\lambda, q)$
$(C, i, m, m', \pi, \pi') \leftarrow \mathcal{A}^{\mathsf{CCol}(\cdot,\cdot,\cdot,\cdot,\mathtt{td},\cdot)}(\mathsf{pp})$
store (C, i) queried to CCol in Q
if $m \neq m' \wedge (C, i) \notin Q$
$\wedge \mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi)$
$\wedge \mathsf{CVer}_{\mathsf{pp}}(C, i, m', \pi')$
output 1
else output 0

Hiding A chameleon vector commitment is required to be hiding. Informally, this means that a PPT adversary knowing the public parameters should not be able to tell the difference between two commitments to two vectors of messages of his choice even after seeing openings for all positions where they agree.

Definition 12 (Hiding). *A $\mathcal{CVC}$ is* hiding *if no stateful PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ is able to infer information about messages at positions, at which the commitment has not been opened yet. Formally, the success probability of $\mathcal{A}$ in the following game* Hiding *should be negligible in λ:*

Experiment $\mathsf{Hiding}^{\mathcal{CVC}}_{\mathcal{A}}(\lambda)$
$(\mathsf{pp}, \mathtt{td}) \leftarrow \mathsf{CGen}(1^\lambda, q)$
$(M_0, M_1) = ((m_1^0, \ldots, m_q^0), (m_1^1, \ldots, m_q^1)) \leftarrow \mathcal{A}_0(\mathsf{pp})$
$b \leftarrow \{0, 1\}$
$(C, \mathsf{aux}) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(M_b)$
for $i = 1, \ldots, q$:
if $m_i^0 = m_i^1$
$\pi_i \leftarrow \mathsf{COpen}_{\mathsf{pp}}(i, m_i^b, \mathsf{aux})$
else
$\pi_i \leftarrow \perp$
$\Pi \leftarrow (\pi_1, \ldots, \pi_q)$
$b' \leftarrow \mathcal{A}_1(C, \Pi)$
if $b = b'$ output 1
else output 0

Note that we provided the definition of the hiding property for completeness, even though it is not needed in our constructions.

3.2 Our Generic Scheme

The idea of our generic construction is to put a chameleon hash into each position of a vector commitment. This allows to find collisions by finding a new randomness for the chameleon hash.

Construction 1. *Let $\mathcal{VC} = (\mathsf{VGen}, \mathsf{VCom}, \mathsf{VOpen}, \mathsf{VVer}, \mathsf{VUpdate}, \mathsf{VProofUpdate})$ be a vector commitment scheme and $\mathcal{CH} = (\mathsf{CHGen}, \mathsf{Ch}, \mathsf{Col})$ a chameleon hash function.*

$\underline{\mathsf{CGen}(1^\lambda, q)}$
$\mathsf{pp} \leftarrow \mathsf{VGen}(1^\lambda, q)$
for $i = 1, \ldots, q$
$(csk_i, cpk_i) \leftarrow \mathsf{CHGen}(1^\lambda)$
set $\mathsf{Ch}_i(\cdot) := \mathsf{Ch}(cpk_i, \cdot)$
$\mathtt{td} \leftarrow (csk_1, \ldots, csk_q)$
Output $(\mathsf{pp}, \mathtt{td})$

$\underline{\mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_q)}$
$(r_1, \ldots, r_q) \leftarrow (\{0,1\}^\lambda, \ldots, \{0,1\}^\lambda)$
$(c_1, \ldots, c_q) \leftarrow (\mathsf{Ch}_1(m_1; r_1), \ldots, \mathsf{Ch}_q(m_q; r_q))$
$(C, \mathsf{aux}') \leftarrow \mathsf{VCom}_{\mathsf{pp}}(c_1, \ldots, c_q)$
$\mathsf{aux} \leftarrow (\mathsf{aux}', (r_1, \ldots, r_q))$
Output (C, aux)

$\underline{\mathsf{COpen}_{\mathsf{pp}}(i, m, \mathtt{aux})}$
parse $\mathtt{aux}$ *as* $(\mathtt{aux}', (r_1, \ldots, r_q))$
$c_i \leftarrow \mathsf{Ch}_i(m; r_i)$
$\Lambda \leftarrow \mathsf{VOpen}_{\mathsf{pp}}(c_i, i, \mathtt{aux}')$
$\pi \leftarrow (\Lambda, r_i)$
Output π

$\underline{\mathsf{CUpdate}_{\mathsf{pp}}(C, i, m, m', \pi)}$
parse π *as* (Λ, r)
$r' \leftarrow \{0,1\}^\lambda$
$c \leftarrow \mathsf{Ch}_i(m; r)$
$c' \leftarrow \mathsf{Ch}_i(m'; r')$
$(C', U) \leftarrow \mathsf{VUpdate}_{\mathsf{pp}}(C, i, c, c')$
Output $(C', (U, c', r'))$

$\underline{\mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi)}$
parse π *as* (Λ, r_i)
$c \leftarrow \mathsf{Ch}_i(m; r_i)$
Output $\mathsf{VVer}_{\mathsf{pp}}(C, i, c, \Lambda)$

$\underline{\mathsf{CCol}_{\mathsf{pp}}(C, i, m, m', \mathtt{td}, \mathtt{aux})}$
parse $\mathtt{aux}$ *as* $(\mathtt{aux}', (r_1, \ldots, r_q))$
parse $\mathtt{td}$ *as* $(csk_1, \ldots, csk_q)$
$r_i' \leftarrow \mathsf{Col}(csk_i, m, r_i, m')$
Output $(\mathtt{aux}', (r_1, \ldots, r_i', \ldots, r_q))$

$\underline{\mathsf{CProofUpdate}_{\mathsf{pp}}(C, \pi_j, i, U)}$
parse π_j *as* (Λ_j, r)
parse U *as* (U', c', r')
$\Lambda_j' \leftarrow \mathsf{VProofUpdate}_{\mathsf{pp}}(C, \Lambda_j, c', i, U')$
If $i = j$ *output* (Λ_j', r')
else output (Λ_j', r)

The following theorem shows the security of this construction.

Theorem 1. *If $\mathcal{VC}$ is a concise position-binding vector commitment and $\mathcal{CH}$ is a collision-resistant chameleon hash, then the above construction is a concise, position-binding, and hiding chameleon vector commitment with indistinguishable collisions.*

We prove this theorem via the following lemmata.

Lemma 3. *If $\mathcal{VC}$ is a vector commitment and $\mathcal{CH}$ is a collision-resistant chameleon hash, then the above construction has indistinguishable collisions.*

The idea of the proof is that any PPT adversary that wins against the game ColInd must be able to distinguish between a truly random value and the output of Col. As this is impossible, such an adversary cannot exist. The fact that for given m, m' the output of $\mathsf{Col}(csk, m, r, m')$ has the same distribution as r immediately implies that no stateful PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ can distinguish the output of $\mathsf{Col}(csk, m, r, m')$ from a truly random r, i.e., has a success probability of $1/2$ in the following game:

Experiment $\mathsf{UniformDist}_{\mathcal{A}}^{\mathsf{Ch}}(\lambda)$
$(csk, cpk) \leftarrow \mathsf{CHGen}(1^\lambda)$
$(m, m') \leftarrow \mathcal{A}_0(cpk)$
$r_0 \leftarrow \{0,1\}^\lambda$
$r_1 \leftarrow \mathsf{Col}(csk, m, r_0, m')$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}_1(r_b)$
if $b = b'$ output 1
else output 0

Proof. Assume there exists a PPT adversary $\mathcal{A}$ that wins the game ColInd with probability non-negligibly bigger than $1/2$. From this we construct another adversary $\mathcal{B}$ against the game $\mathsf{UniformDist}$ as defined above. To achieve this, $\mathcal{B}$ computes $(\mathtt{pp}, \mathtt{td}) \leftarrow \mathsf{CGen}(1^\lambda, q)$ and invokes $\mathcal{A}(\mathtt{pp}, \mathtt{td})$ in a black-box way, who outputs $((m_1, \ldots, m_q), (i, m_i'))$. Our reduction $\mathcal{B}$ outputs (i, m_i, m_i') and receives r_i, which is either a truly random value r or the output of $\mathsf{Col}(\mathtt{td}, m_i, r, m_i')$. $\mathcal{B}$ then chooses $r_j \leftarrow \{0,1\}^\lambda, \forall j \neq i$, computes $c_i \leftarrow \mathsf{Ch}(m_i'; r_i)$ and $c_j \leftarrow \mathsf{Ch}(m_j; r_j), \forall j \neq i$, computes the commitment $(C, \mathtt{aux}') \leftarrow \mathsf{VCom}_{\mathsf{pp}}(c_1, \ldots, c_q)$ and sets $\mathtt{aux} \leftarrow (\mathtt{aux}', (r_1, \ldots, r_q))$. It then invokes $\mathcal{A}(C, \mathtt{aux})$. Eventually $\mathcal{A}$ outputs a bit b' and the reduction

then also outputs b'. Observe that the reduction is efficient and perfectly simulates the case $b = 1$ of the game $\mathsf{CollInd}$ for $\mathcal{A}$ when given a truly random r and the case $b = 0$ when given the output of Col. This gives that $\text{Prob}\left[\mathsf{UniformDist}_{\mathcal{B}}^{\mathsf{Ch}}(\lambda) = 1\right] = \text{Prob}\left[\mathsf{CollInd}_{\mathcal{A}}^{\mathcal{CVC}}(\lambda, q) = 1\right]$. Since the first probability equals $1/2$, but the second one was assumed to be non-negligibly bigger than $1/2$, this is a contradiction. Therefore such an adversary $\mathcal{A}$ cannot exist.

The following lemma shows that our generic construction also satisfies the position-binding property.

Lemma 4. *If $\mathcal{VC}$ is a position-binding vector commitment and $\mathcal{CH}$ is a collision-resistant chameleon hash, then the above construction is position-binding.*

The main idea of the proof is that an adversary may win against the game PosBdg in two ways, either by finding a collision in the chameleon hash, or by breaking the position-binding property of the underlying vector-commitment. We then show that both cases only happen with negligible probability.

Proof. Let $\mathcal{A}$ be a PPT adversary against position-binding as defined in game PosBdg. Denote by coll the event that $\mathcal{A}$ outputs (C, i, m, m', π, π') with $\pi = (\Lambda_1, r_1)$ and $\pi' = (\Lambda_2, r_2)$ such that $\mathsf{Ch}(m, r_1) = \mathsf{Ch}(m', r_2)$ and $m \neq m'$. Obviously

$$\begin{aligned}
&\text{Prob}\left[\mathsf{PosBdg}_{\mathcal{A},\mathcal{CVC}}(\lambda) = 1\right] \\
=&\,\text{Prob}\left[\mathsf{PosBdg}_{\mathcal{A},\mathcal{CVC}}(\lambda) = 1 \wedge \mathsf{coll}\right] + \text{Prob}\left[\mathsf{PosBdg}_{\mathcal{A},\mathcal{CVC}}(\lambda) = 1 \wedge \overline{\mathsf{coll}}\right] \\
\leq&\,\text{Prob}\left[\mathsf{coll}\right] + \text{Prob}\left[\mathsf{PosBdg}_{\mathcal{A},\mathcal{CVC}}(\lambda) = 1 \wedge \overline{\mathsf{coll}}\right].
\end{aligned}$$

We now show that both parts of this sum are negligible. To see that $\text{Prob}[\,\mathsf{coll}]$ is negligible, consider the following reduction $\mathcal{B}_{\mathcal{CH}}$ against the collision-resistance of $\mathcal{CH}$: On input Ch, the reduction $\mathcal{B}_{\mathcal{CH}}$ chooses the index i, for which the adversary $\mathcal{A}$ will output his collision, at random and computes $(\mathsf{pp}, \mathtt{td}) \leftarrow \mathsf{VGen}(1^\lambda, q)$. It then computes $(csk_j, cpk_j) \leftarrow \mathsf{CHGen}(1^\lambda)$ and sets $\mathsf{Ch}_j(\cdot) := \mathsf{Ch}(cpk_j, \cdot)$ for $j \neq i$, sets $\mathsf{Ch}_i(\cdot) := \mathsf{Ch}$, and runs a black-box simulation of $\mathcal{A}^{\mathsf{CCol}(\cdot,\cdot,\cdot,\cdot,\mathtt{td},\cdot)}(\mathsf{pp})$. Whenever the adversary asks for a collision at position i the reduction aborts, otherwise it can compute the collision since it knows all the other trapdoors. The algorithm $\mathcal{A}$ outputs (C, i, m, m', π, π') and $\mathcal{B}_{\mathcal{CH}}$ parses $\pi = (\Lambda_1, r_1)$ and $\pi' = (\Lambda_2, r_2)$, and stops, outputting (m, m', r_1, r_2).

For the analysis observe that $\mathcal{B}_{\mathcal{CH}}$ is efficient and that

$$\text{Prob}\left[\mathsf{Hashcoll}_{\mathcal{B}_{\mathcal{CH}},\mathcal{CH}}(\lambda) = 1\right] = \frac{1}{q}\,\text{Prob}\left[\mathsf{coll}\right].$$

This follows easily because $\mathcal{B}_{\mathcal{CH}}$ perfectly simulates the view of $\mathcal{A}$ as in the original game whenever it chooses the correct index i. As $\mathcal{CH}$ is collision-resistant, this probability is negligible.

Now, to see that $\text{Prob}\left[\mathsf{PosBdg}_{\mathcal{A},\mathcal{CVC}}(\lambda) = 1 \wedge \overline{\mathsf{coll}}\right]$ is also negligible, consider the following reduction $\mathcal{B}_{\mathcal{VC}}$ against the position-binding of $\mathcal{VC}$: On input pp, $\mathcal{B}_{\mathcal{VC}}$ computes q key-pairs, $(csk_i, cpk_i) \leftarrow \mathsf{CHGen}(1^\lambda)$, sets $\mathsf{Ch}_i(\cdot) := \mathsf{Ch}(cpk_i, \cdot)$ and runs $\mathcal{A}^{\mathsf{CCol}(\cdot,\cdot,\cdot,\cdot,\mathtt{td},\cdot)}(\mathsf{pp})$ in a black-box way. Observe that the reduction $\mathcal{B}_{\mathcal{VC}}$ can perfectly simulate the collision oracle using the chameleon-hash trapdoors. I.e. whenever $\mathcal{A}$ submits a query $(C, i, m, m', \mathtt{aux})$ the reduction parses $\mathtt{aux} = (\mathtt{aux}', (r_1, \ldots, r_q))$ and computes $r_i' \leftarrow \mathsf{Col}(csk_i, m, r_i, m')$. It then outputs $(\mathtt{aux}', (r_1, \ldots, r_i', \ldots, r_q))$ to the adversary $\mathcal{A}$. At the end $\mathcal{A}$ outputs (C, i, m, m', π, π') and $\mathcal{B}_{\mathcal{VC}}$ parses $\pi = (\Lambda_1, r_1)$ and $\pi' = (\Lambda_2, r_2)$, computes $c_1 \leftarrow \mathsf{Ch}_i(m; r_1)$, $c_2 \leftarrow \mathsf{Ch}_i(m; r_2)$, and outputs $(C, i, c_1, c_2, \Lambda, \Lambda')$.

For the analysis observe that $\mathcal{B}_{\mathcal{VC}}$ is efficient. Furthermore, it holds that

$$\text{Prob}\left[\mathsf{PosBdg}_{\mathcal{B}_{\mathcal{VC}},\mathcal{VC}}(\lambda) = 1\right] = \text{Prob}\left[\mathsf{PosBdg}_{\mathcal{A},\mathcal{CVC}}(\lambda) = 1 \wedge \overline{\mathsf{coll}}\right],$$

because pp is perfectly distributed as in PosBdg and $\mathcal{B}_{\mathcal{VC}}$ wins only if $c_1 \neq c_2$. As $\mathcal{VC}$ is position-binding, this probability is also negligible.

As both probabilities are negligible, any adversary $\mathcal{A}$ wins only with negligible probability, therefore the construction above is position-binding.

Lemma 5. *If $\mathcal{VC}$ is a vector commitment and $\mathcal{CH}$ is a collision-resistant chameleon hash, then Construction 1 is hiding.*

Proof. As the distribution of a chameleon hash is uniform and independent of the message, the above construction is essentially a vector commitment to q independent random values. Therefore the vector commitment is independent of the messages and no adversary $\mathcal{A}$ can win Hiding with probability non-negligibly bigger than $1/2$.

Lemma 6. *If $\mathcal{VC}$ is a concise vector commitment, then Construction 1 is concise.*

Proof. Since C is the output of VCom, and proofs consist of a proof from COpen plus some randomness, it is easy to see that Construction 1 is concise exactly when the underlying $\mathcal{VC}$ is.

3.3 Construction of CVCs based on CDH

In this section we show a direct construction of CVCs based on the Square-CDH assumption in bilinear groups, which – we note – has been shown equivalent to the standard CDH assumption [1,13]. Our direct construction can be seen as an aggregated variant of the Krawczyk-Rabin chameleon hash function [8], or as a generalization of the VC scheme due to Catalano and Fiore [6]. For simplicity we describe the scheme in symmetric pairings, but we stress that this scheme can be expressed using asymmetric pairings and our implementation does use asymmetric pairings.

Construction 2. *Let $\mathbb{G}, \mathbb{G}_T$ be two groups of prime order p with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$.*

$\mathsf{CGen}(1^\lambda, q)$**:** *Let $g \in \mathbb{G}$ be a random generator. Choose $z_1, \ldots, z_q \leftarrow \mathbb{Z}_p$ at random, set $h_i = g^{z_i}$ for $i = 1, \ldots, q$ and $h_{i,j} = g^{z_i z_j}$ for $i, j = 1, \ldots, q, i \neq j$. Finally, set $\mathsf{pp} = (g, \{h_i\}_{i=1,\ldots,q}, \{h_{i,j}\}_{i,j=1,\ldots,q,i\neq j})$ and $\mathtt{td} = \{z_i\}_{i=1,\ldots,q}$.*

$\mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_q)$**:** *Choose $r \leftarrow \mathbb{Z}_p$ at random. Set $C = h_1^{m_1} \cdots h_q^{m_q} g^r$ and $\mathtt{aux} = (m_1, \ldots, m_q, r)$.*

$\mathsf{COpen}_{\mathsf{pp}}(i, m, \mathtt{aux})$**:** *Compute $\pi = h_i^r \cdot \prod_{j=1, j\neq i}^{q} h_{i,j}^{m_j}$.*

$\mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi)$**:** *If $e(C/h_i^m, h_i) = e(\pi, g)$ output 1, else output 0.*

$\mathsf{CCol}_{\mathsf{pp}}(C, i, m, m', \mathtt{td}, \mathtt{aux})$**:** *Parse $\mathtt{aux} = (m_1, \ldots, m_q, r)$. Compute $r' = r + z_i(m - m')$ and set $\mathtt{aux}' = (m_1, \ldots, m', \ldots, m_q, r')$.*

$\mathsf{CUpdate}_{\mathsf{pp}}(C, i, m, m')$**:** *Compute $C' = C \cdot h_i^{m'-m}$, set $U = (i, u) = (i, m' - m)$.*

$\mathsf{CProofUpdate}_{\mathsf{pp}}(C, \pi_j, j, U)$**:** *Parse $U = (i, u)$. Compute $C' = C \cdot h_i^u$. If $i \neq j$ compute $\pi'_j = \pi_j \cdot h_{j,i}^u$, else $\pi'_j = \pi_j$.*

We also show two additional algorithms that allow to "accumulate" several updates at different positions, and then to apply these updates to a proof in constant time.

$\mathsf{accumulateUpdate}_{\mathsf{pp}}(AU, U)$**:** Parse $AU = (j, au)$ and $U = (i, u)$. If $j \neq i$, compute $au = au \cdot h_{i,j}^u$.

$\mathsf{CProofUpdate}'_{\mathsf{pp}}(\pi_j, AU)$**:** Parse $AU = (j, au)$. Compute $\pi'_j = \pi_j \cdot au$.

Theorem 2. *If the CDH assumption holds, Construction 2 is a chameleon vector commitment.*

First, we show the correctness of this construction. To this end, observe that

$$C = h_1^{m_1} \cdots h_q^{m_q} g^r = g^{r + \sum_{j=1}^{q} z_j m_j}$$

and

$$\pi_i = h_i^r \cdot \prod_{j=1, j\neq i}^{q} h_{i,j}^{m_j} = (g^r \cdot \prod_{j=1, j\neq i}^{q} h_j^{m_j})^{z_i} = g^{z_i \cdot (r + \sum_{j=1, j\neq i}^{q} z_j m_j)}$$

Thereby

$$e(C/h_i^{m_i}, h_i) = e(g^{r+\sum_{j=1,j\neq i}^{q} z_j m_j}, g^{z_i})$$
$$= e\left(g^{z_i \cdot \left(r+\sum_{j=1,j\neq i}^{q} z_j m_j\right)}, g\right) = e(\pi_i, g)$$

Correctness after updates is almost the same as above: simply observe that by construction it holds $C' = C \cdot h_i^{m'-m} = C \cdot h_i^U$ and (for $j \neq i$) $\pi'_j = \pi_j \cdot h_{j,i}^U$.

Second, it is easy to see that the construction is concise.

Lemma 7. *Construction 2 is concise.*

Proof. The proof is straightforward and follows from observing that both the commitment C and all proofs π_i are single elements of the group $\mathbb{G}$, which is chosen independently of the vector size q.

Next, we proceed to show that our construction has indistinguishable collisions. Note that we actually show that it has perfectly indistinguishable collisions, which means that even an unbounded adversary cannot win the game CollInd with probability non-negligibly greater than 1/2.

Lemma 8. *Construction 2 has perfectly indistinguishable collisions.*

The main idea of the proof is that both cases in the game CollInd essentially give a random group element to the adversary, who thereby cannot distinguish them.

Proof. In the game CollInd it holds that

$$C_0 = h_1^{m_1} \cdots h_i^{m_i} \cdots h_q^{m_q} g^{r_0} = h_1^{m_1} \cdots h_i^{m'_i} \cdots h_q^{m_q} g^{r'_0}$$
$$= g^{r_0+\sum_{j=1}^{q} z_j m_j} = g^{r'_0+z_i m'_i+\sum_{j=1,j\neq i}^{q} z_j m_j}$$

and

$$C_1 = h_1^{m_1} \cdots h_i^{m'_i} \cdots h_q^{m_q} g^{r_1} = g^{r_1+z_i m'_i+\sum_{j=1,j\neq i}^{q} z_j m_j}$$

The auxiliary information aux consists of all messages and r'_0 resp. r_1. Since r_0 is a uniformly random element from $\mathbb{Z}_p$, the element $r'_0 = r_0 + z_i(m_i - m'_i)$ is also a uniformly random element. Therefore, both r'_0 and r_1 are uniformly distributed in $\mathbb{Z}_p$, thus any adversary against CollInd has a success probability of exactly 1/2.

We continue by showing that our construction is position-binding.

Lemma 9. *If the CDH assumption holds, Construction 2 is position-binding.*

We prove this claim by giving a reduction against Square-CDH. As mentioned in Section 2.2, the Square-CDH assumption has been shown equivalent to the standard CDH assumption. The main idea of the reduction is to guess the index at which the adversary will break the position binding and to embed the challenge from the Square-CDH problem.

Proof. Assume there exists an efficient adversary $\mathcal{A}$ that wins the game PosBdg by returning two valid proofs with non-negligible probability ϵ. Given black-box access to $\mathcal{A}$, we build an adversary $\mathcal{B}$ that breaks the Square-CDH assumption with non-negligible probability $\geq \epsilon/q$. $\mathcal{B}$ is given (g, g^a) as input. It then tries to guess the index for which $\mathcal{A}$ will output the two proofs by choosing a random $i \leftarrow \{1, \ldots, q\}$. For

$j = 1, \ldots, q, j \neq i$ the reduction $\mathcal{B}$ randomly chooses $z_j \leftarrow \mathbb{Z}_p$, sets $h_j = g^{z_j}$ and $h_{i,j} = (g^a)^{z_j}$. It then sets $h_i = g^a$, computes $h_{j,k} = g^{z_k z_j}$ for $j, k = 1, \ldots, q, j, k \neq i, j \neq k$, sets $\mathsf{pp} = (g, \{h_i\}_{i=1,\ldots,q}, \{h_{i,j}\}_{i,j=1,\ldots,q,i\neq j})$, and runs $\mathcal{A}^{\mathsf{CCol}(\cdot,\cdot,\cdot,\cdot,\mathtt{td},\cdot)}(\mathsf{pp})$. Whenever the adversary $\mathcal{A}$ asks for a collision at position i $\mathcal{B}$ aborts. Otherwise $\mathcal{B}$ can perfectly simulate the collision oracle since it knows all the z_j values, for $j \neq i$. The adversary $\mathcal{A}$ is supposed to output (C, m, m', j, π, π'). If $j \neq i$, i.e., if $\mathcal{B}$ did not guess the index correctly, $\mathcal{B}$ aborts. Otherwise it outputs

$$(\pi'/\pi)^{(m-m')^{-1}} = g^{a^2}.$$

Observe that pp is perfectly distributed as the output of $\mathsf{CGen}(1^\lambda)$, and thus the random index i chosen by $\mathcal{B}$ is uniformly distributed in $\mathcal{A}$'s view. Hence, the probability that $\mathcal{A}$ outputs $j = i$ (and thus $\mathcal{B}$ does not abort) is $1/q$. Furthermore, since

$$\begin{aligned} e(C, h_i) &= e(h_i^m, h_i) \cdot e(\pi, g) = e(h_i^{m'}, h_i) \cdot e(\pi', g) \\ &= e(h_i, h_i)^m \cdot e(\pi, g) = e(h_i, h_i)^{m'} \cdot e(\pi', g) \end{aligned}$$

it holds that $e(h_i, h_i)^{m-m'} = e(\pi'/\pi, g)$. As $e(h_i, h_i) = e(g^a, g^a) = e(g^{a^2}, g)$ it is easy to see that $g^{a^2} = (\pi'/\pi)^{(m-m')^{-1}}$. Therefore, if $\mathcal{B}$ does not abort and $\mathcal{A}$ breaks position binding $\mathcal{B}$ successfully breaks the Square-CDH assumption.

We go on by showing that our construction is also hiding. Again, note that we even prove our construction to be perfectly hiding, i.e., even an unbounded adversary cannot win the game Hiding.

Lemma 10. *Construction 2 is perfectly hiding.*

The main idea of this proof is that for two fixed vectors of messages M_0 and M_1, we can find two random values r_0 and r_1 such that (M_0, r_0) and (M_1, r_1) map to the same commitment value C. We then show that the distribution of r_0 and r_1 are identical and therefore any adversary $\mathcal{A}$ has a success probability of exactly $1/2$ to win the game Hiding with probability $> 1/2$.

Proof. Let M_0 and M_1 be the output of an adversary $\mathcal{A}$ against the hiding property as defined in the game Hiding. Now set $(C_0, aux_0) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(M_0)$, set $(C_1, \mathtt{aux}^0) = (C_0, \mathtt{aux}_0)$, and compute $\mathtt{aux}^i \leftarrow \mathsf{CCol}(C_1, i, M_0[i], M_1[i], \mathtt{td}, \mathtt{aux}^{i-1})$ for $i = 1, \ldots, q$. Finally set $\mathtt{aux}_1 = \mathtt{aux}^q$. Note that $(C_0, \mathtt{aux}_0)$ is a chameleon vector commitment to M_0 and $(C_1, \mathtt{aux}_1)$ is a chameleon vector commitment to M_1. Furthermore, since this construction has indistinguishable collisions, the distribution of $(C_0, \mathtt{aux}_0)$ and $(C_1, \mathtt{aux}_1)$ is identical to the case where $\mathtt{aux}_1$ is the output of CCom and $\mathtt{aux}_0$ is computed using the collision finding algorithm.

It now holds that $\mathtt{aux}_0 = (m_1^0, \ldots, m_q^0, r^0)$ and $\mathtt{aux}_1 = (m_1^1, \ldots, m_q^1, r^1)$. Since $C_0 = C_1$ we also have

$$r^0 + \sum_{j=1}^{q} z_j m_j^0 = r^1 + \sum_{j=1}^{q} z_j m_j^1.$$

For every position i with $m_i^0 = m_i^1$ proofs for the same position have the same value, i.e.,

$$\pi_i^0 = g^{z_i \cdot \left(r^0 + \sum\limits_{j=1, j\neq i}^{q} z_j m_j^0\right)} = g^{z_i \cdot \left(r^1 + \sum\limits_{j=1, j\neq i}^{q} z_j m_j^1\right)} = \pi_i^1.$$

This means, that the view of the adversary $\mathcal{A}$ is completely identical in the case $b = 0 \wedge r = r^0$ and in the case $b = 1 \wedge r = r^1$. Overall, for fixed (M_0^*, M_1^*), this gives

$$\begin{aligned} &\mathrm{Prob}\left[\mathcal{A}(\Pi) = b' \;\middle|\; b = 0 \wedge r = r^0 \wedge (M_0, M_1) = (M_0^*, M_1^*)\right] \\ &= \mathrm{Prob}\left[\mathcal{A}(\Pi) = b' \;\middle|\; b = 1 \wedge r = r^1 \wedge (M_0, M_1) = (M_0^*, M_1^*)\right] \end{aligned}$$

and

$$\begin{aligned}&\mathrm{Prob}\left[\mathcal{A}(\Pi)=0 \;\middle|\; b=0 \wedge r=r^0 \wedge (M_0,M_1)=(M_0^*,M_1^*)\right]\\&+\mathrm{Prob}\left[\mathcal{A}(\Pi)=1 \;\middle|\; b=1 \wedge r=r^1 \wedge (M_0,M_1)=(M_0^*,M_1^*)\right]\\&=1\end{aligned}$$

Since this gives a bijective mapping from r^0 to r^1, i.e., $r^1 = f_{M_0,M_1}(r^0)$, and r^0 and r^1 are uniformly distributed, we have that $\mathrm{Prob}\left[\mathsf{Hiding}_{\mathcal{A}}^{\mathcal{CVC}}=1 \;\middle|\; (M_0,M_1)=(M_0^*,M_1^*)\right]$ is

$$\begin{aligned}&=\frac{1}{2p}\left(\sum_{r^0\in\mathbb{Z}_p}\Pr[\mathcal{A}(\Pi)=0 \mid b=0 \wedge r=r^0 \wedge (M_0,M_1)=(M_0^*,M_1^*)]\right.\\&\left.+\sum_{r^1\in\mathbb{Z}_p}\mathrm{Prob}\left[\mathcal{A}(\Pi)=1 \;\middle|\; b=1 \wedge r=r^1 \wedge (M_0,M_1)=(M_0^*,M_1^*)\right]\right)\\&=\frac{1}{2p}\sum_{r^0\in\mathbb{Z}_p}\left(\mathrm{Prob}\left[\mathcal{A}(\Pi)=0 \;\middle|\; b=0 \wedge r=r^0 \wedge (M_0,M_1)=(M_0^*,M_1^*)\right]\right.\\&\left.+\mathrm{Prob}\left[\mathcal{A}(\Pi)=1 \;\middle|\; b=1 \wedge r=f_{M_0,M_1}(r^0) \wedge (M_0,M_1)=(M_0^*,M_1^*)\right]\right)\\&=\frac{1}{2p}\sum_{r^0\in\mathbb{Z}_p}1=\frac{1}{2}\end{aligned}$$

Thus the overall success-probability of $\mathcal{A}$ is

$$\begin{aligned}&\mathrm{Prob}\left[\mathsf{Hiding}_{\mathcal{A}}^{\mathcal{CVC}}=1\right]\\&=\sum_{(M_0^*,M_1^*)\leftarrow\mathcal{A}(\mathsf{pp})}\mathrm{Prob}\left[\mathsf{Hiding}_{\mathcal{A}}^{\mathcal{CVC}}=1 \;\middle|\; (M_0,M_1)=(M_0^*,M_1^*)\right]\\&\cdot\mathrm{Prob}[(M_0,M_1)=(M_0^*,M_1^*)]\\&=\sum_{(M_0^*,M_1^*)\leftarrow\mathcal{A}(\mathsf{pp})}\frac{1}{2}\cdot\mathrm{Prob}[(M_0,M_1)=(M_0^*,M_1^*)]=\frac{1}{2}\end{aligned}$$

Therefore, no adversary can win Hiding with probability bigger than $1/2$ and Construction 2 is hiding.

Remark 1. We stress that our security definitions only guarantee hiding and position-binding if the attacker has not seen collisions for a position. For instance, in our construction if the attacker has seen a collision s.t. both (π_i, m) and (π_i', m') are valid w.r.t. C, he can extract $g^{z_i^2} = (\pi'/\pi)^{(m-m')^{-1}}$. However, we stress that this is not an issue for our application.

4 VDS From CVCs

In this section we present our VDS protocol from CVCs. The section is structured as follows: in Section 4.1 we explain the main idea of the construction; the formal description is given in Section 4.2..

4.1 Intuition

We explain the intuition of our construction with the help of Figure 1. The main idea is to build a q-ary tree where each node of the tree is a CVC of size $q+1$ that "stores" some data in the first component and q pointers to its q children in the q remaining components (or 0 as a dummy value for children that do not yet exist). The root of the tree is treated as the public verification key, while the CVC trapdoor acts as the private key which enables the client to add further elements to the tree.

The tree is constructed in such a way that it grows dynamically from top to bottom and works as follows. Initially, the tree is completely empty. Elements are inserted into the tree from left to right and new children are "inserted" into the parent by finding a collision in the appropriate position. Consider for example the tree shown in Figure 1 where $q = 2$. The first element d_1 is stored in the first node together with two pointers p_{11} and p_{12}. When the client wishes to add the next element d_2, it generates a new CVC to the vector of three elements consisting of d_2 and two pointers p_{21} and p_{22}. Next, the client finds a collision in p_{02} such that the new CVC belongs to the root CVC.

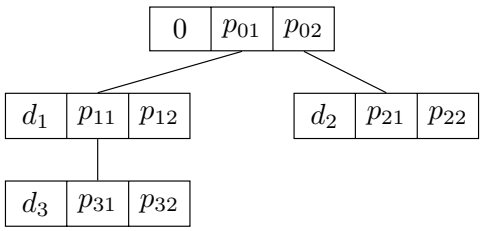

Figure 1. The authenticated data structure for $q = 2$ that is used to construct VDS from CVCs, where d refers to a data item and p is a pointer to the next node.

The proof that a data element d is stored at a certain position in the tree consists of a list of CVC-proofs and intermediate nodes of the tree, i.e. $\tilde{\pi}_i \leftarrow (\pi_l, n_l, \ldots, n_1, \pi_0)$, where π_l is a proof, that the element d is "stored" in the first component of node n_l, π_{l-1} is a proof that the node n_l is a child of node n_{l-1}, and eventually π_0 is a proof that node n_1 is a child of the root node. The whole proof $\tilde{\pi}_i$ is called an *authentication path*.

To reduce the amount of data the client has to store, we exploit the key-features of our CVCs. Adding a new node to the tree requires knowledge of the parent of the new node, i.e., the value and the corresponding auxiliary information, but storing the entire tree at the client would defeat the purpose of a VDS protocol. To circumvent this problem, we make nodes recomputable. Namely, to create a new node n_i with value d_i, we first derive the randomness r_i of the commitment deterministically using a pseudorandom function, and then we compute the node $(n_i, \mathtt{aux}_i^*) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(0, 0, \ldots, 0; r_i)$ and find a collision $\mathtt{aux}_i \leftarrow \mathsf{CCol}_{\mathsf{pp}}(n_i, 1, 0, d, \mathtt{td}, \mathtt{aux}_i^*)$.

This idea allows us to split the tree between the client and the server in the following way: The client only stores the secret key $\mathsf{sp} = (k, \mathtt{td}, \mathtt{cnt})$, whereas the server simply stores all data $d_1, d_2, \ldots$ and all nodes $n_1, n_2, \ldots$ along with their insertion paths. Now, whenever the client wants to add a new data element d, it determines the next free index $i = \mathtt{cnt} + 1$ and the level l of the new node, as well as the index p of its parent and the position j this node will have in its parent. The client then computes the new node as $(n_i, \mathtt{aux}_i^*) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(0, 0, \ldots, 0; r_i)$, where the randomness r_i is computed using the PRF. It adds the new data element by finding a collision for the first component $\mathtt{aux}_i \leftarrow \mathsf{CCol}_{\mathsf{pp}}(n_i, 1, 0, d, \mathtt{td}, \mathtt{aux}_i^*)$. With this, the client then can compute a proof, that d is indeed stored in n_i in the first component: $\pi_l \leftarrow \mathsf{COpen}_{\mathsf{pp}}(1, d, \mathtt{aux}_i)$. To insert this new node n_i in the tree, the client recomputes the parent node as $(n_p, \mathtt{aux}_p^*) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(0, 0, \ldots, 0; r_p)$ and finds a collision in the j-th position: $\mathtt{aux}_p \leftarrow \mathsf{CCol}_{\mathsf{pp}}(n_p, j, 0, n_i, \mathtt{td}, \mathtt{aux}_p^*)$. From there, the client can then compute a proof for the new node: $\pi_{l-1} \leftarrow \mathsf{COpen}_{\mathsf{pp}}(j, n_i, \mathtt{aux}_p)$. We call (π_l, n_i, π_{l-1}) an insertion path (since it is a partial authentication path).

The client then streams the new data element and the insertion path $(d, (\pi_l, n_i, \pi_{l-1}))$ to the server, that stores this tuple in its database. To answer a query on index i, the server computes a full authentication path for i by concatenating the insertion paths of all nodes in the path from i until the root. Note that each insertion path is of constant size due to the conciseness requirement for CVCs (see Definition 8), and thus the size of a full authentication path is only in $\mathcal{O}(\log_q N)$, where N is the number of elements outsourced so far.

At first glance, the idea described so far seems to work. However, there is one more issue that needs to be overcome. That is, the client may perform updates (i.e., change a data value from d to d') and continue streaming new elements afterwards. Although update queries do not affect the pointers values in

the commitments, they do change the value of a commitment and its related proofs. Basically, the issue is that the client may not be able to recompute the commitment and the updated proofs of the parent node n_p anymore (without storing all update information locally). We solve this issue by letting the server store "an aggregate" of all update information. This way, the client can compute an "outdated" proof (i.e., a proof as before any update) and then this proof can be updated by the server in constant time. Precisely, in the general case the server should store a list of all updates and apply them one by one on incoming proofs. However, in the case of our CDH-based CVCs, we exploit their homomorphic property to "accumulate" update-information by only storing its sum, thus achieving *constant* insertion time.

4.2 Formal Description of our Construction

In this section, we present the formal description of our construction.

Construction 3. *Let* $\mathcal{CVC} = (\mathsf{CGen}, \mathsf{CCom}, \mathsf{COpen}, \mathsf{CVer}, \mathsf{CCol}, \mathsf{CUpdate}, \mathsf{CProofUpdate})$ *be a CVC. Define* $\mathcal{VDS} = (\mathsf{Setup}, \mathsf{Append}, \mathsf{Query}, \mathsf{Verify}, \mathsf{Update})$ *as follows:*

$\mathsf{Setup}(1^\lambda, \mathsf{q})$ *This algorithm picks a random PRF key* $k \leftarrow \{0,1\}^\lambda$*, computes a key-pair for the chameleon vector commitment* $(\mathsf{pp}, \mathtt{td}) \leftarrow \mathsf{CGen}(1^\lambda, q+1)$*, and sets the counter* $\mathtt{cnt} := 0$*. It computes* $r_0 \leftarrow \mathsf{f}(k, 0)$*, sets the root as* $(\rho, \mathtt{aux}_\rho) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(0, \ldots, 0; r_0)$*, the secret key* $sk := (k, \mathtt{td}, \mathtt{cnt})$*, and the public key* $pk := (\mathsf{pp}, \rho)$*. Finally, the secret key* sk *is kept by the client while the public key* pk *is given to the server. Before proceeding with the remaining algorithms, we summarize the information stored by the server. The server maintains a database* DB *consisting of tuples* $(i, d_i, n_i, \pi_i, \pi_{p,j}, AU_i, \{AU_{i,j}\}_{j=1}^{q+1})$ *where:* $i \geq 0$ *is an integer representing the index of every* DB *element,* d_i *is* DB *value at index* i*,* n_i *is a CVC commitment,* π_i *is a CVC proof that* d_i *is the first committed message in* n_i*,* $\pi_{p,j}$ *is a CVC proof that* n_i *is the the message at position* $j+1$ *committed in* n_p *(which is the CVC of* n_i*'s parent node),* AU_i *is the accumulated update information that can be used to update the proof* π_i*, and* $AU_{i,j}$ *are the accumulated update informations that can be used to update the children's proofs* $\pi_{i,j}$*.*

$\mathsf{Append}(sk, d)$ *The algorithm parses* $sk = (k, \mathtt{td}, \mathtt{cnt})$ *and determines the index* $i = \mathtt{cnt} + 1$ *of the new element, the index* $p = \lfloor \frac{i-1}{q} \rfloor$ *of its parent node, the position* $j = ((i-1) \mod q) + 2$ *that this element will have in its parent, and then it increases the counter* $\mathtt{cnt}' = \mathtt{cnt} + 1$*. Next, it computes the new node as* $(n_i, \mathtt{aux}_i^*) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(0, 0, \ldots, 0; r_i)$ *where* $r_i \leftarrow \mathsf{f}(k, i)$ *and inserts the data* d *by finding a collision* $\mathtt{aux}_i \leftarrow \mathsf{CCol}_{\mathsf{pp}}(n_i, 1, 0, d, \mathtt{td}, \mathtt{aux}_i^*)$*. To insert the node* n_i *in the tree, the algorithm recomputes the parent node as* $(n_p, \mathtt{aux}_p^*) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(0, 0, \ldots, 0; r_p)$ *and inserts* n_i *as the* j*-th child of* n_p *by finding a collision in the parent node at position* j*, i.e., the client runs* $\mathtt{aux}_p \leftarrow \mathsf{CCol}_{\mathsf{pp}}(n_p, j, 0, n_i, \mathtt{td}, \mathtt{aux}_p^*)$*. It then computes* $\pi_{i,1} \leftarrow \mathsf{COpen}_{\mathsf{pp}}(1, d, \mathtt{aux}_i)$ *and* $\pi_{p,j} \leftarrow \mathsf{COpen}_{\mathsf{pp}}(j, n_i, \mathtt{aux}_p)$*, and sets the insertion path* $(\pi_{i,1}, n_i, \pi_{p,j})$*. The client* $\mathcal{C}$ *sends the above insertion path and the new element* d *to the server* $\mathcal{S}$*.* $\mathcal{S}$ *then applies the accumulated update* $AU_{p,j}$ *to* $\pi_{p,j}$ *(i.e., compute* $\pi'_{p,j} \leftarrow \mathsf{CProofUpdate}'_{\mathsf{pp}}(\pi_{p,j}, AU_{p,j})$*) and stores these items in its database* DB*.*

$\mathsf{Query}(pk, DB, i)$ *In the query protocol, the client sends* i *to the server, who determines the level* $l \leftarrow \lceil \log_q((q-1)(i+1)+1) - 1 \rceil$ *and constructs an authentication path:*

$\tilde{\pi}_i \leftarrow (\pi_{i,1})$
$a \leftarrow i$
$b \leftarrow \lfloor \frac{a-1}{q} \rfloor$
for $h = l-1, \ldots, 0$
  $c \leftarrow ((a-1) \mod q) + 2$
  $\tilde{\pi}_i \leftarrow \tilde{\pi}_i :: (n_a, \pi_{b,c})$
  $a \leftarrow b$
  $b \leftarrow \lfloor \frac{b-1}{q} \rfloor$

Finally, the server returns $\tilde{\pi}_i$ *to the client.*

$\mathsf{Verify}(pk, i, d, \tilde{\pi}_i)$ *This algorithm parses* $pk = (\mathsf{pp}, \rho)$ *and* $\tilde{\pi}_i = (\pi_l, n_l, \ldots, n_1, \pi_0)$*. It then proceeds by verifying all proofs in the authentication path:*

$v \leftarrow \mathsf{CVer}_{\mathsf{pp}}(n_i, 1, d, \pi_l) \wedge n_i \neq 0$
$a \leftarrow i$
$b \leftarrow \lfloor \frac{a-1}{q} \rfloor$
for $h = l-1, \ldots, 0$
  $c \leftarrow ((a-1) \mod q) + 2$
  $v \leftarrow v \wedge \mathsf{CVer}_{\mathsf{pp}}(n_b, c, n_a, \pi_h) \wedge n_b \neq 0$
  $a \leftarrow b$
  $b \leftarrow \lfloor \frac{b-1}{q} \rfloor$

If $v = 1$ *then output* d*. Otherwise output* $\perp$*.*

$\mathsf{Update}(pk, DB, sk, i, d')$ *In the update protocol, the client, on input the secure key* sk*, sends an index* i *and a value* d' *to the server. The server answers by sending the value* d *currently stored at position* i *and the corresponding authentication path* $\tilde{\pi}_i = (\pi_l, n_l, \ldots, n_1, \pi_0)$ *(this is generated as in the* Query *algorithm). The client then checks the correctness of* $\tilde{\pi}_i$ *by running* $\mathsf{Verify}(pk, i, d, \tilde{\pi}_i)$*. If the verification fails, the client stops running. Otherwise, it continues as follows. First, it parses* sk *as* $(k, \mathtt{td}, \mathtt{cnt})$*, it determines the level of the updated node* $l \leftarrow \lceil \log_q((q-1)(i+1)+1) - 1 \rceil$*, and computes the new root* $\rho' = n'_0$ *as follows:*

$(n'_l, U_l) \leftarrow \mathsf{CUpdate}_{\mathsf{pp}}(n_i, 1, d, d', \pi_l)$
$a \leftarrow i$
$b \leftarrow \lfloor \frac{a-1}{q} \rfloor$
for $h = l-1, \ldots, 0$
  $c \leftarrow ((a-1) \mod q) + 2$
  $(n'_h, U_h) \leftarrow \mathsf{CUpdate}_{\mathsf{pp}}(n_h, c, n_{h+1}, n'_{h+1}, \pi_h)$
  $a \leftarrow b$
  $b \leftarrow \lfloor \frac{b-1}{q} \rfloor$

On the other side, after receiving (i, d')*, the server runs a similar algorithm to update all stored elements and proofs along the path of the new node. It also accumulates the new update information for every node in this path:*

$(n'_i, U_i) \leftarrow \mathsf{CUpdate}_{\mathsf{pp}}(n_i, 1, d, d')$
for $j = 1, \ldots, q+1$
  $AU_{i,j} \leftarrow \mathsf{accumulateUpdate}(AU_{i,j}, U_i)$
  $\pi'_{i,j} \leftarrow \mathsf{CProofUpdate}'_{\mathsf{pp}}(\pi_{i,j}, AU_{i,j})$
$(\cdot, \pi'_i) \leftarrow \mathsf{CProofUpdate}_{\mathsf{pp}}(n_i, \pi_i, i, U_i)$
$a \leftarrow i$
$b \leftarrow \lfloor \frac{a-1}{q} \rfloor$
for $h = l-1, \ldots, 0$
  $c \leftarrow ((a-1) \mod q) + 2$
  $(n'_b, U_b) \leftarrow \mathsf{CUpdate}_{\mathsf{pp}}(n_b, c, n_a, n'_a)$
  for $j = 1, \ldots, q+1$
    $AU_{b,j} \leftarrow \mathsf{accumulateUpdate}(AU_{b,j}, U_b)$
    $\pi'_{b,j} \leftarrow \mathsf{CProofUpdate}'_{\mathsf{pp}}(\pi_{b,j}, U_b)$
  $a \leftarrow b$
  $b \leftarrow \lfloor \frac{b-1}{q} \rfloor$

In the above algorithms, a *is the index of the changed node,* b *the index of its parent node, and* c *the position of node* a *in* b*. Basically, the algorithms change the value of node* i*, and then this change propagates up to the root* ($\rho' = n'_0$)*.*
Finally, both the client and the server compute the new public key as $pk = (\mathtt{pp}, \rho')$*.*

Theorem 3. *If* f *is a pseudorandom function and* $\mathcal{CVC}$ *is a secure CVC, then Construction 3 is a secure VDS.*

Proof. We prove the theorem by first defining a hybrid game in which we replace the PRF with a random function. Such hybrid is computationally indistinguishable from the real VDSsec security game by assuming that f is pseudorandom. Then, we proceed to show that any efficient adversary cannot win with non-negligible probability in the hybrid experiment by assuming that the CVC scheme is secure. In what follows we use $Gm_{i,\mathcal{A}}(\lambda)$ to denote the experiment defined by Game i run with adversary $\mathcal{A}$.

Game 0 : this identical to the experiment VDSsec.

Game 1 : this is the same as Game 0 except that the PRF f is replaced with a random function (via lazy sampling). It is straightforward to see that this game is negligibly close to Game 0 under the pseudo randomness of f, i.e., $\mathrm{Prob}[\,Gm_{0,\mathcal{A}}(\lambda) = 1] - \mathrm{Prob}[\,Gm_{1,\mathcal{A}}(\lambda) = 1] = \mathsf{negl}(\lambda)$.
Now, consider Game 1. Let $(i^*, d^*, \hat{\pi})$ be the tuple returned by the adversary at the end of the game, d be the value currently stored in the database at index i^*. Recall that Game 1 outputs 1 if $\mathsf{Verify}(pk, i^*, d^*, \hat{\pi}) = 1$ and $d \neq d^*$. Consider a honestly computed authentication path $\tilde{\pi}$ for (i^*, d) (this is the path which can be computed by the challenger), and observe that by construction the sequence in $\hat{\pi}$ ends up at the public root. Intuitively, this means that $\hat{\pi}$ and $\tilde{\pi}$ must deviate at some point in the path from i^* up to the root. We define dcol as the event that the two authentication paths deviate exactly in i^*, i.e., that $n_i = n_i^*$. Dually, if dcol does not occur, it intuitively means that the adversary managed to return a valid authentication path that deviates from a honestly computed one in some internal node. Clearly, we have:

$$\begin{aligned}\mathrm{Prob}[\,Gm_{1,\mathcal{A}}(\lambda) = 1] = \mathrm{Prob}[\,Gm_{1,\mathcal{A}}(\lambda) = 1 \wedge \mathsf{dcol}] \\ + \mathrm{Prob}\left[\,Gm_{1,\mathcal{A}}(\lambda) = 1 \wedge \overline{\mathsf{dcol}}\right]\end{aligned}$$

Our proof proceeds by showing that both $\mathrm{Prob}[\,Gm_{1,\mathcal{A}}(\lambda) = 1 \wedge \mathsf{dcol}]$ and $\mathrm{Prob}\left[\,Gm_{1,\mathcal{A}}(\lambda) = 1 \wedge \overline{\mathsf{dcol}}\right]$ are negligible under the assumption that the CVC is position-binding.

Case dcol. In this case we build a reduction $\mathcal{B}$ against the position-binding property of the underlying CVC.

On input pp, the reduction $\mathcal{B}$ computes the root node as described in the catGen algorithm, sets the counter $\mathtt{cnt} := 0$, and sets $pk \leftarrow (\mathsf{pp}, \rho)$. It then runs $\mathcal{A}(\mathsf{vp})$ by simulating the VDSsec game.

Whenever the adversary $\mathcal{A}$ streams some data element d, the reduction proceeds as described in the catAdd algorithm (except that pseudorandom values are now sampled randomly, as per Game 1). However, $\mathcal{B}$ does not know the full secret key sk – it does not know the CVC trapdoor – but it can use its collision-oracle to compute the necessary collisions in the CVC in order to add new nodes to the tree. So, the reduction $\mathcal{B}$ returns $(i, \tilde{\pi}_i)$ to the adversary and stores the tuple $(d, i, \tilde{\pi}_i)$ in a list L.

Whenever the adversary $\mathcal{A}$ wants to update the element at position i to the new value d', the reduction proceeds as described in the Update algorithm. Note that the CVC trapdoor is not needed in this phase. $\mathcal{B}$ then returns the updated proof $\tilde{\pi}'_i$ to $\mathcal{A}$ and updates the tuple $(d', i, \tilde{\pi}'_i)$ in L.

Eventually the adversary outputs $(d^*, i^*, \hat{\pi}^*)$. The reduction then finds the actual data d and the corresponding proof $\tilde{\pi}_i$ for position i by searching for the tuple $(d, i^*, \tilde{\pi}_i)$ in L, parses $\hat{\pi}^* = (\pi^*, n_i^*, \dots)$ and $\tilde{\pi}_i = (\pi, n_i, \dots)$ and outputs $(n_i, 1, d, d^*, \pi, \pi^*)$.

For the analysis now observe that $\mathcal{B}$ is efficient as so is $\mathcal{A}$, and searching for a tuple in an ordered list can be done in polynomial time. It is easy to see that $\mathcal{B}$ perfectly simulates the view for $\mathcal{A}$ as in the game VDSsec. Now, whenever dcol happens, we know that $n_i^* = n_i$. Hence both (π, d) and (π^*, d^*) must verify correctly whenever $\mathcal{A}$ wins. Furthermore, observe that $\mathcal{B}$ never uses its collision-oracle on position 1, since the Append algorithm always uses CCol on index $j > 1$. This means essentially means that

$$\begin{aligned}\mathrm{Prob}[\,Gm_{1,\mathcal{A}}(\lambda) = 1 \wedge \mathsf{dcol}] &\leq \mathrm{Prob}[\,\mathsf{PosBdg}_{\mathcal{B}}(\lambda, q) = 1] \\ &= \mathsf{negl}(\lambda)\end{aligned}$$

Case $\overline{\mathsf{dcol}}$ Recall that in this case Game 1 outputs 1 only if the adversary wins by returning an authentication path which deviates from the correct one at some internal node in the path from i^* up to the root. In this case too, we build a reduction $\mathcal{B}$ against the position-binding property of the underlying CVC.

On input $\mathtt{pp}$, the reduction $\mathcal{B}$. It then tries to set an upper limit on the number of elements the adversary will authenticate in the tree by choosing its depth $l = \lambda$. The reduction then builds a tree of CVCs of size l from bottom to top, where in each CVC every position which does not point to a child (especially the first position) is set to 0. Denote the root of this tree by ρ. Finally, the reduction $\mathcal{B}$ sets $\mathtt{cnt} := 0$, sets $pk \leftarrow (\mathtt{pp}, \rho)$ and runs $\mathcal{A}(pk)$ by simulating Game 1 to it.

Whenever the adversary $\mathcal{A}$ streams some data element d to the reduction, the reduction determines the index $i = \mathtt{cnt} + 1$ for the new data element, increases $\mathtt{cnt}$ by one and inserts the new element into the tree by finding a collision in the first component of node n_i using its collision-oracle. It then computes an authentication path $\tilde{\pi}_i$ for d as described in the Append algorithm. The reduction returns $(i, \tilde{\pi}_i)$ to the adversary and stores $(d, i, \tilde{\pi}_i)$ in some list L. If the adversary exceeds the number of elements l, the reduction stops the adversary $\mathcal{A}$, increases $l \leftarrow l \cdot \lambda$, and starts again.

Whenever the adversary $\mathcal{A}$ wants to update the element at position i to some new value d' the reduction proceeds as described in the Update algorithm. It then returns the updated proof $\tilde{\pi}'_i$ to the adversary and updates the tuple $(d', i, \tilde{\pi}'_i)$ in L.

At the end of the game the adversary outputs $(d^*, i^*, \hat{\pi}^*)$. The reduction parses $\hat{\pi}^* = (\pi_0^*, n_0^*, \pi_1^*, \dots)$ and finds the largest j for which $n_i^* = n_j$, i.e., for which the authentication path $\hat{\pi}^*$ still agrees with the actual tree. Also $\mathcal{B}$ finds the authentication path $\tilde{\pi}_{i^*} = (\pi_0^{i^*}, n_0^{i^*}, \pi_1^{i^*}, \dots)$ up to i^*, if i^* was streamed by the adversary, or otherwise up to the deepest ancestor of i^*. Clearly, π_i^* then must be a proof that n_{i-1}^* is "stored" in n_i^* at some position h.

If n_j is the deepest node in the path towards i^* that is stored by the challenger then the h-th message committed in n_j by the challenger is 0. In this case $\mathcal{B}$ can produce a honest proof $\pi_{h,0}$ that 0 is the h-th message committed in n_j, and then outputs $(n_j, h, 0, n_{i-1}^*, \pi_{h,0}, \pi_i^*)$.

Otherwise, if n_j is not the deepest node, the honest path $\tilde{\pi}_{i^*}$ must contain a node $n_k^{i^*} = n_j$ as well as a proof $\pi_k^{i^*}$ that the node $n_{k-1}^{i^*}$ is the h-th child of n_j. In this case $\mathcal{B}$ outputs $(n_j, h, n_{k-1}^{i^*}, n_{i-1}^*, \pi_k^{i^*}, \pi_i^*)$. As one can check, this case also captures the on in which $h = 1$ and $n_{k-1}^{i^*} = d \neq d^* = n_{i-1}^*$.

For the analysis see that $\mathcal{B}$ is efficient because $\mathcal{A}$ is and because the limit l will be large enough after a polynomial number of times. Now observe that $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in Game 1 (otherwise the underlying CVC would not have indistinguishable collisions). It is easy to see that whenever $\mathcal{A}$ wins the pair (π_i^*, n_{i-1}^*) verifies w.r.t. n_j. As $\mathcal{B}$ honestly computed $n_{k-1}^{i^*}$ and $\pi_k^{i^*}$, this pair will also verify w.r.t. n_j. Note that $\mathcal{B}$ only asks for collisions at position 1, but $h > 1$. Therefore $\mathcal{B}$ wins whenever $\mathcal{A}$ does. Since the underlying CVC is position-binding, this probability is at most negligible.

$$\begin{aligned}\mathrm{Prob}\left[\, Gm_{1,\mathcal{A}}(\lambda) = 1 \wedge \overline{\mathsf{dcol}} \right] &\leq \mathrm{Prob}\left[\, \mathsf{PosBdg}_{\mathcal{B}}(\lambda, q) = 1 \right] \\ &= \mathsf{negl}(\lambda)\end{aligned}$$

Since both parts are at most negligible in λ, the overall success probability of any efficient adversary can only be negligible in λ, hence Construction 3 is secure according to Definition 6.

5 VDS from Accumulators

Our second construction is conceptually very different from all previous VDS constructions, since it does not rely on any tree structure. The basic idea is to let the client sign each element of the stream with a regular signature scheme, and to use a cryptographic accumulator to "invalidate" (aka revoke) the signatures of all elements that have been updated. That is, the verification key consists of the verification key of the signature scheme and the accumulator. Whenever a client retrieves an element from the server, then the server attaches a proof of non-membership which shows that the signature is not part of the accumulator, i.e., the signature is still valid.

However, this idea does not work immediately. One reason is that some accumulators only support the accumulation of an a-priori fixed number of elements such as [15]. Another issue is that the size of the public-key is typically linear in the number of accumulated values, and thus the client might not be able to store it. In our construction we solve this issue by exploiting the specific algebraic properties of the bilinear-map

accumulator. Somewhat interestingly, this allows us to reduce the size of the public-key to $\mathcal{O}(1)$, and to support an unbounded number of updates.

5.1 Our Scheme

In this section we show how to use the bilinear-map accumulator described in Section 2.5 to build a VDS protocol.

Construction 4. *Let* $\mathsf{Sig} = (\mathsf{SKg}, \mathsf{Sign}, \mathsf{Vrfy})$ *be a signature scheme and* $H : \{0,1\}^* \mapsto \mathbb{Z}_p^*$ *be a hash function. The VDS protocol* $\mathcal{VDS} = (\mathsf{Setup}, \mathsf{Append}, \mathsf{Query}, \mathsf{Verify}, \mathsf{Update})$ *is defined as follows:*

$\mathsf{Setup}(1^\lambda)$ *The setup algorithm first generates a tuple of bilinear the parameters* $(p, \mathbb{G}, \mathbb{G}_T, e, g)$*. Second, it chooses* s *at random from* $\mathbb{Z}_p^*$ *and computes* g^s*. Next, it generates a key-pair* $(ssk, vk) \leftarrow \mathsf{SKg}(1^\lambda)$*, initializes two counters* $\mathtt{cnt} := 0$ *and* $\mathsf{upd} := 0$*, and sets* $S_{last} = g$*. It also generates an initially empty accumulator* $f'(\mathcal{E}) := g$*. Note that during the lifetime of the scheme, the server will increasingly store* $g, g^s, \ldots, g^{s^{\mathsf{upd}}}$ *while the client stores* $S_{last} = g^{s^{\mathsf{upd}}}$*. Finally, the algorithm outputs the secret key* $sk = (g, p, s, ssk, S_{last}, \mathtt{cnt}, \mathsf{upd})$ *which is kept by the client, and the public key* $pk = (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, vk, f'(\mathcal{E}))$ *which is given to the server.*

$\mathsf{Append}(sk, d)$ *The append algorithm increases the counter* $\mathtt{cnt} := \mathtt{cnt} + 1$*, chooses a random tag* $\mathsf{tag} \leftarrow \{0,1\}^\lambda$*, and signs the value* $m := d \| \mathsf{tag} \| \mathtt{cnt}$ *by computing* $\sigma \leftarrow \mathsf{Sign}(ssk, m)$*. The pair* $((d, \mathsf{tag}, \mathtt{cnt}), \sigma)$ *is finally sent to* $\mathcal{S}$*, who stores it at position* $\mathtt{cnt}$ *in* DB*.*

$\mathsf{Query}(pk, DB, i)$ *To retrieve the* i*-th element from* DB*, the client sends* i *to* $\mathcal{S}$*, who computes the response as follows:* $\mathcal{S}$ *retrieves the pair* (m, σ) *from* DB *and computes a proof of non-membership* (w, u) *for the element* $e_i \leftarrow H(\sigma)$ *as described in Section 2.5. Finally,* $\mathcal{S}$ *returns* $(d, \pi) = ((d', \mathsf{tag}, i), (\sigma, w, u))$*.*

$\mathsf{Verify}(pk, i, d, \pi)$ *This algorithm parses* $pk = (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, vk, f'(\mathcal{E}))$*,* $d = (d', \mathsf{tag}, i)$*, and* $\pi = (\sigma, w, u)$*. It sets* $e_i \leftarrow H(\sigma)$*, and outputs* d' *iff* $\mathsf{Vrfy}(vk, d' \| \mathsf{tag} \| i, \sigma) = 1$ *and* $e(w, g^{e_i} \cdot g^s) = e(f'(\mathcal{E}) \cdot g^u, g)$*.*

$\mathsf{Update}(sk, DB, sk, i, d')$ *The clients runs* $(d, \pi) \leftarrow \mathsf{Query}(pk, DB, i)$ *to retrieve the* i*th entry from* DB*. Afterwards,* $\mathcal{C}$ *verifies the correctness of the entry running the verification algorithm* $\mathsf{Verify}(pk, i, d, \pi)$*. If* Verify *outputs* 1*, then* $\mathcal{C}$ *increases the counter* $\mathsf{upd} := \mathsf{upd} + 1$*, signs the new element* $\sigma' \leftarrow \mathsf{Sign}(ssk, d' \| \mathsf{tag} \| i)$ *using a random tag* $\mathsf{tag} \leftarrow \{0,1\}^\lambda$*, and adds the old signature to the accumulator as follows. The client computes* $g^{s^{\mathsf{upd}}} = S_{last} := (S_{last})^s$*,* $e_i \leftarrow H(\sigma)$*, and* $f''(\mathcal{E}) := f'(\mathcal{E})^{e_i + s}$*. The client sends* $(g^{s^{\mathsf{upd}}}, f''(\mathcal{E}), \sigma')$ *to* $\mathcal{S}$*. The server stores* (σ', d') *at position* i *in* DB*, and* $g^{s^{\mathsf{upd}}}$ *in its parameters. The public key is then updated to* $pk' := (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, vk, f''(\mathcal{E}))$*.*

Theorem 4. *If* Sig *is a strongly unforgeable signature scheme,* H *a collision-resistant hash function, and* ACC *the collision-resistant accumulator as defined in Section 2.5, then Construction 4 is a secure VDS protocol.*

Proof. Let $\mathcal{A}$ be an efficient adversary against the security of Construction 4 as defined in game VDSsec, denote by $pk^* := (p, \mathbb{G}, \mathbb{G}_T, e_i, g, g^s, vk, f^*(\mathcal{E}))$ the public-key hold by the challenger at the end of the game, and let $Q := ((d_1 \| \mathsf{tag}_1 \| 1, \sigma_1), (d_2 \| \mathsf{tag}_2 \| 2, \sigma_2), \ldots, (d_n \| \mathsf{tag}_n \| n, \sigma_n))$ be the state of the database DB at the end of the game. By $(w_1, u_1), \ldots, (w_n, u_n)$ we denote the witnesses of the non-membership proofs that be can be computed by the challenger using public values only, as discussed in Section 2.5, and denote by $Q' := ((d'_1 \| \mathsf{tag}'_1 \| i'_1, \sigma'_1), (d'_2 \| \mathsf{tag}'_2 \| i'_2, \sigma'_2), \ldots, (d'_n \| \mathsf{tag}'_n \| i_n, \sigma'_n))$ all queries sent by $\mathcal{A}$. Clearly, $Q \subseteq Q'$ and let $O := Q' \backslash Q$ be the set of queries that have been sent by the adversary and which are not in the current database. Let $\mathcal{A}$ be an efficient adversary that outputs $((d^*, \mathsf{tag}^*, i^*), (\sigma^*, w^*, u^*))$ such that $(\hat{d}, i^*) \notin Q$, $\mathsf{Vrfy}(vk, d^* \| \mathsf{tag}^* \| i^*, \sigma) = 1$, and $e(w^*, g^{e_i^*} \cdot g^s) = e(f'(\mathcal{E}) \cdot g^{u^*}, g)$, with $e_i^* := H(\sigma^*)$. Then, we define the following events:

- hcol is the event that there exists an index $1 \leq i \leq n$ such that $H(\sigma^*) = H(\sigma'_i)$ and $\sigma^* \neq \sigma'_i$.
- fake is the event that $e(w^*, g^{e_i^*} \cdot g^s) = e(f'(\mathcal{E}) \cdot g^{u^*}, g)$ and $e_i \in \mathcal{E}$.

Observe that the case where the adversary finds a collision in the accumulator and the one where he finds a fake witness for a membership of a non-member of $\mathcal{E}$, do not help to break the security of the VDS scheme. Given these events, we can bound $\mathcal{A}$'s success probability as follows:

$$\begin{aligned}\text{Prob}\left[\mathsf{VDSsec}_{\mathcal{A}}^{\mathcal{VDS}}(\lambda)=1\right] \leq \text{Prob}[\mathsf{hcol}] + \text{Prob}[\mathsf{fake}] + \\ \text{Prob}\left[\mathsf{VDSsec}_{\mathcal{A}}^{\mathcal{VDS}}(\lambda)=1 \ \wedge \ \overline{\mathsf{hcol}} \ \wedge \ \overline{\mathsf{fake}}\right]\end{aligned}$$

In the following we show that the parts of the sum are negligible. The fact that Prob[hcol] is negligible, follows trivially by the collision-resistance of the hash function. Furthermore, it is also easy to see that Prob[fake] is negligible as well due to proof of non-membership properties of the accumulator.

Claim. $\text{Prob}\left[\mathsf{VDSsec}_{\mathcal{A}}^{\mathcal{VDS}}(\lambda)=1 \wedge \overline{\mathsf{hcol}} \wedge \overline{\mathsf{fake}}\right] \approx 0.$

This claim follows from the strong unforgeability of the underlying signature scheme. The intuition is that the correct proof non-membership guarantees that the signature is not stored in the accumulator. Since it is not stored in the accumulator, the adversary did not receive this signature from the signing oracle and thus, is a valid forger w.r.t. strong unforgeabiility.

More formally, let $\mathcal{A}$ be an efficient adversary against the security of the VDS protocol. Then we construct an algorithm $\mathcal{B}$ against the strong unforgeability as follows. The input of $\mathcal{B}$ is a public-key vk and it has access to a signing oracle. It generates random elements $(p, \mathbb{G}, \mathbb{G}_T, e, g, g^s)$ for some s chosen at random from $\mathbb{Z}_p^*$, counters $\texttt{cnt} := 0$ and $\texttt{upd} := 0$, and $\mathcal{B}$ also generates an initially empty accumulator $f'(\mathcal{E}) := g$. In runs $\mathcal{A}$ on $pk = (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, vk, f'(\mathcal{E}))$ in a black-box way. Whenever $\mathcal{A}$ wishes to append an element d, then $\mathcal{B}$ increments the counter $\texttt{cnt} := \texttt{cnt} + 1$, chooses a fresh tag $\texttt{tag}$, sends $m' := d||\texttt{tag}||\texttt{cnt}$ to its signing oracle and forwards the response σ together with the corresponding proof of non-membership to $\mathcal{A}$. It is understood that $\mathcal{B}$ records alls queries and answers. If $\mathcal{A}$ wants to update the ith element, then $\mathcal{B}$ adds the corresponding signature σ_i to the accumulator, increases the counter $\texttt{upd} := \texttt{upd} + 1$, picks a fresh tag $\texttt{tag}'$ at random, sends $m' := d'||\texttt{tag}'||\texttt{cnt}$ to its singing oracle and forwards the response together with a proof of non-membership and $g^{s^{\texttt{upd}}}$ to $\mathcal{A}$. Eventually, $\mathcal{A}$ stops, outputting $((d^*, \texttt{tag}^*, i^*), (\sigma^*, w^*, u^*))$. The algorithm $\mathcal{B}$ stops, outputting $((d^*||\texttt{tag}^*||i^*), \sigma^*)$.

For the analysis, it's easy to see that $\mathcal{B}$ is efficient and performs a perfect simulation from $\mathcal{A}$'s point of view. In the following, let's assume that $\mathcal{B}$ succeeds with non-negligible probability, i.e., $((d^*, \texttt{tag}^*, i^*), (\sigma^*, w^*, u^*))$ satisfies the following: $(\hat{d}, i^*) \notin Q$, $\mathsf{Vrfy}(vk, d^*||\texttt{tag}^*||i^*, \sigma) = 1$, and $e(w^*, g^{e_i^*} \cdot g^s) = e(f'(\mathcal{E}) \cdot g^{u^*}, g)$, with $e_i^* := H(\sigma^*)$. We now argue, that $\mathcal{B}$ succeeds whenever $\mathcal{A}$ does. To see this observe that $(\hat{d}, i^*) \notin Q$ and that the proof of non-membership verifies. Thus, conditioning on $\overline{\mathsf{fake}}$ and on $\overline{\mathsf{hcol}}$ the claim follows.

6 Experimental Results

We implemented the construction based on chameleon vector commitments (section 3) and on accumulators (section 5) in Java 1.7.

In this section, we provide comprehensive benchmarks of all proposed schemes to evaluate their practicality. We investigate the computational and bandwidth overhead induced by our protocols. We use the PBC library [10] in combination with a java wrapper for pairing-based cryptographic primitives (using a type D-201 MNT curve), and the Bouncy Castle Cryptographic API 1.50 [3] for all other primitives. For the construction based on accumulators, we used RSA-PSS with 1024 bit long keys as our underlying signature scheme, and SHA-1 as our hash function. Our experiments were performed on an Amazon EC2 `r3.large` instance equipped with 2 vCPUs Intel Xeon Ivy Bridge processors, 15 GiB of RAM and 32 GB of SSD storage running Ubuntu Server 14.04 LTS (Image ID `ami-0307d674`).

We stress that this is an unoptimized, prototype implementation, and that better performance results may be achieved by further optimizations.

DATASET: We evaluated our schemes by outsourcing and then retrieving 8GB, using chunk-sizes of 256kB, 1MB, and 4MB. We measured the insertion and the verification time on the client side, as well as the sizes of all transmitted proofs.

BRANCHING FACTOR: The CVC-based VDS was instantiated with branching factors of $q = 32, 64, 128$, and 256, and the public-key consists of q^2 elements that amount to 2.7MB (for $q = 256$). This is in contrast to the public key size in the accumulator-based VDS which is about 350bytes.

NUMBER OF UPDATES: To quantify the impact of updates on the accumulator-based construction, we performed separate experimental runs. Therefore, we performed 0, 10, 50 and 100 updates prior to retrieving and verifying the outsourced data set.

BLOCK SIZES: The block size is a parameter which depends heavily on the application. Larger block sizes reduce the number of authenticated blocks, and thus yield a smaller tree with more efficient bandwidth and computation performance in the CVC-based VDS. However, larger blocks decrease the granularity of on-the-fly verification, since one has to retrieve an entire block prior to verification.

We took measurements for block sizes of 256kB, 1MB and 4MB, which we believe is a sensible range of block sizes for various applications. For example, consider HD-video streaming with a bandwidth of 8MB/s. Using a block size of 4MB means that one can perform a verification every 0.5 seconds. However, in other applications such as the verifiable stock market, one may want to only retrieve a small fraction of the outsourced data set, for which a smaller block size such as 256kB might be more desirable.

6.1 Streaming Data

Both proposed constructions have constant client-side insertion times as well as constant bandwidth overheads. The insertion proofs in the accumulator-based and the CVC-based construction are respectively 236 and 1070 bytes large. Rather than processing each block directly, we first compute its hash value and pass it to our VDS protocol. Since the insertion time is dominated by this hash function, we give the average insertion times for our different block sizes in Figure 2. As one can see, the accumulator-based construction performs slightly better than the CVC-based one, and both give rise to quite practical timings for the insertion phase.

6.2 Verifying Retrieved Elements

The time needed for verifying the correctness of 8GB data retrieved from the server is depicted in Figure 3. In this figure, various insights about the performance behavior of our protocols are visible.

Firstly, the block size plays a crucial role for the overall performance, since the number of verifications needed to authenticate the 8GB dataset depends linearly on the inverse of the block size.

Secondly, in the CVC-based construction higher tree arity results in shorter proofs, and therefore faster verification.

Thirdly, we observe the impact of updates on our accumulator-based construction. While without updates the accumulator-based construction is faster than the CVC-based one, adding just 10 updates decreases the performance of the accumulator dramatically. Still, for applications where only a few updates are expected, our accumulator-based construction is preferable.

These effects can also be observed in Figure 4. Here the time required per block for retrieving a constant number of blocks is depicted (in contrast to a constant amount of data, as in Figure 3). From top to bottom, the influence of the block size is visible, whereas from left to right the performance of our different constructions is shown in a comparable manner. To remove outliers from our measurements, we used a Hampel-filter [20] with a window-size of 5.

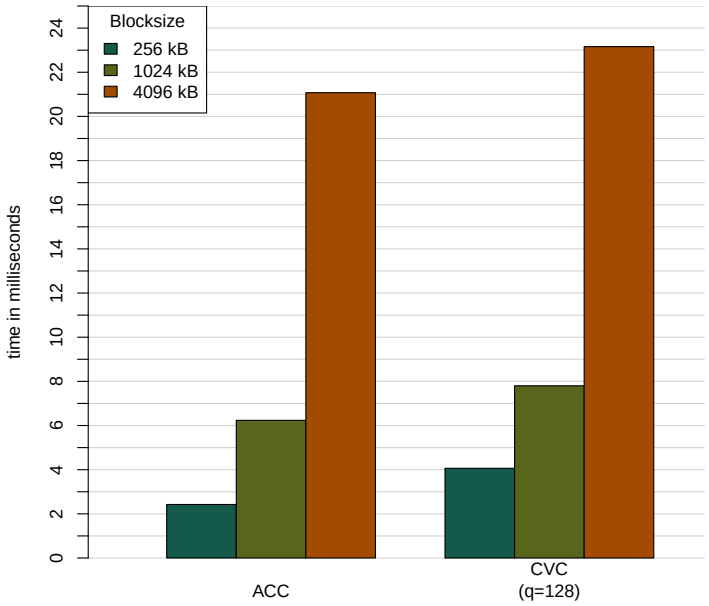

Figure 2. Average insertion time for different block sizes

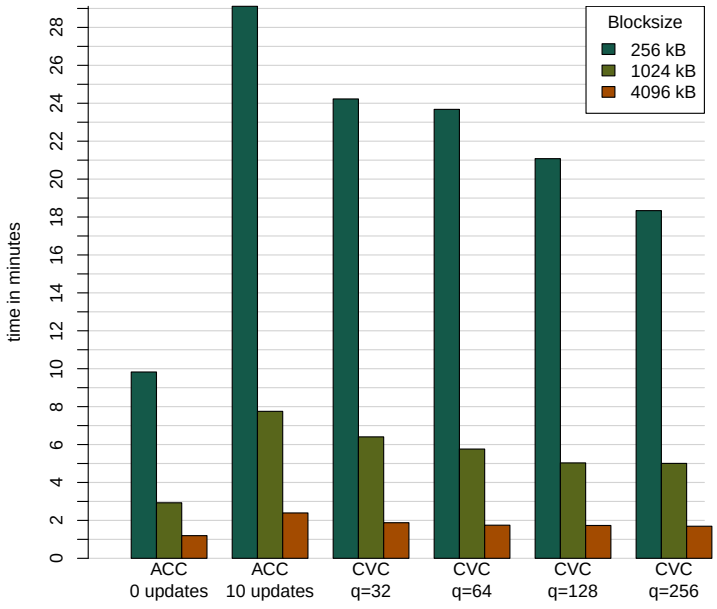

Figure 3. Accumulated verification time for retrieving 8GB

6.3 Bandwidth Overhead

The bandwidth overhead incurred by retrieving 8GB of data is shown in Figure 5. As in Figure 3 the impact of the block size is visible, as is the impact of the branching factor of the CVC-based construction. This figure also shows that the accumulator-based construction, although slower when handling updates, achieves a much smaller bandwidth overhead compared to the CVC-based construction. However, in this scenario for a block-size of 4MB, both constructions introduce a total bandwidth overhead of less than 4MB, or 0.05%.

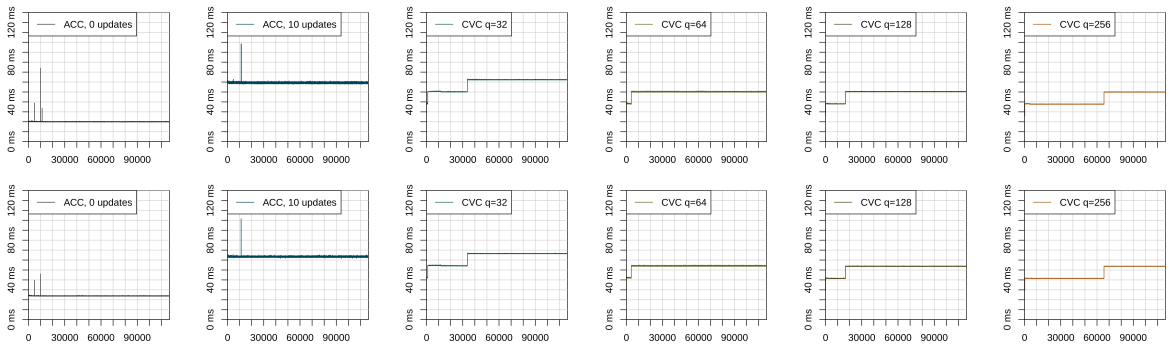

Figure 4. Influence of block size and branching factor/number of updates on the verification time. From top to bottom, the results for a block size of 256kB and 1MB are shown.

References

1. F. Bao, R. Deng, and H. Zhu. Variations of diffie-hellman problem. In S. Qing, D. Gollmann, and J. Zhou, editors, *Information and Communications Security*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer Berlin Heidelberg, 2003. 2.2, 3.3
2. S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. pages 111–131, 2011. 1.2
3. Bouncy Castle. The Legion of the Bouncy Castle. online at `https://www.bouncycastle.org`. 6
4. J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. pages 481–500, 2009. 1.2
5. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. pages 61–76, 2002. 1.2
6. D. Catalano and D. Fiore. Vector commitments and their applications. In K. Kurosawa and G. Hanaoka, editors, *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2013. 1, 1.2, 3, 3.3
7. I. Damgård and N. Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. `http://eprint.iacr.org/`. 2.5, 2
8. H. Krawczyk and T. Rabin. Chameleon signatures. 2000. 3.3
9. B. Libert and M. Yung. Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. pages 499–517, 2010. 3
10. B. Lynn. PBC - C Library for Pairing Based Cryptography. online at `http://crypto.stanford.edu/pbc/`. 6
11. T. Malkin, D. Micciancio, and S. K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. pages 400–417, 2002. 1.3
12. C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39:2004, 2001. 1.2
13. U. M. Maurer and S. Wolf. Diffie-Hellman oracles. pages 268–282, 1996. 2.2, 3.3
14. M. Naor and K. Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000. 1.2
15. L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005. 2, 2.5, 1, 5
16. L. Nguyen. Accumulators from bilinear pairings and applications. pages 275–292, 2005. 1.2
17. C. Papamanthou, E. Shi, R. Tamassia, and K. Yi. Streaming authenticated data structures. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2013. 1.2
18. C. Papamanthou and R. Tamassia. Time and space efficient algorithms for two-party authenticated data structures. In *Proceedings of the 9th international conference on Information and communications security*, ICICS'07, pages 1–15, Berlin, Heidelberg, 2007. Springer-Verlag. 1.2
19. C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. pages 437–448, 2008. 5
20. R. K. Pearson. Outliers in process modeling and identification. *Control Systems Technology, IEEE Transactions on*, 10(1):55–63, 2002. 6.2

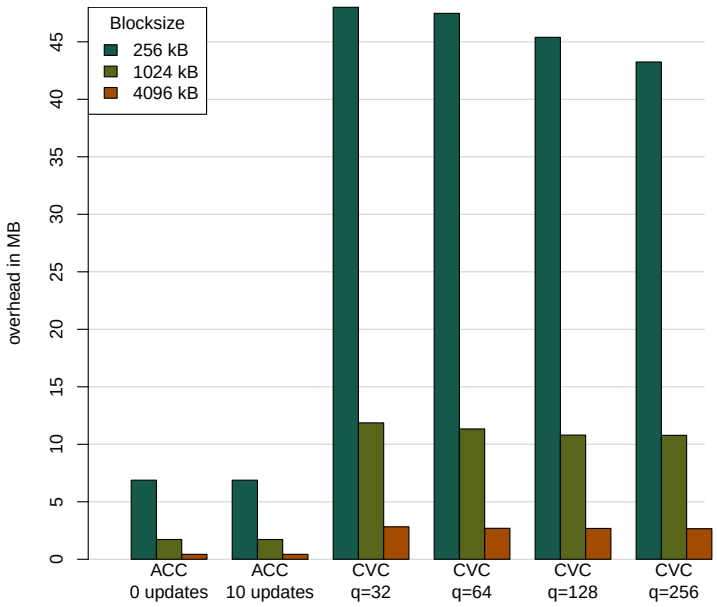

Figure 5. Accumulated bandwidth overhead for retrieving 8GB

21. A. Perrig, R. Canetti, D. X. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. pages 35–46, 2001. 1.2
22. A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. pages 56–73, 2000. 1.2
23. D. Schröder and H. Schröder. Verifiable data streaming. In *ACM Conference on Computer and Communications Security*, pages 953–964. ACM, 2012. 1, 1.1, 1.2, 2.6
24. D. Schröder and M. Simkin. Veristream - a framework for unbounded verifiable data streaming. In *Financial Cryptography*. Springer, 2015. 1, 1.1, 1.1, 1.2
25. E. Stefanov, M. van Dijk, A. Juels, and A. Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 229–238, New York, NY, USA, 2012. ACM. 1.2
26. R. Tamassia and N. Triandopoulos. Certification and authentication of data structures. In *AMW*, 2010. 1.2

A Multi-collision indistinguishability

Definition 10 implies indistinguishability in the case where $\mathcal{A}$ is allowed to output $p = poly(\lambda)$ collision pairs (i_j, m'_{i_j}), $j \in \{1, \ldots, p\}$ instead of only one pair. Denote by M_j the vector of messages after the j-th collision query (i.e. $M_0 = (m_0, \ldots, m_q)$, M_p is the vector where all replacements have been made). Denote by $M_j[i]$ the i-th message in such a vector. Then construct the following hybrids:

$(C_j, \mathtt{aux}_j) \leftarrow \mathsf{CCom}_{\mathtt{pp}}(M_j)$
for $k = (j+1) \ldots p$:
  $\mathtt{aux}_j \leftarrow \mathsf{CCol}_{\mathtt{pp}}(C_j, i_k, M_{k-1}[i_k], m'_{i_k}, \mathtt{td}, \mathtt{aux}_j)$

Now, if any PPT adversary $\mathcal{A}$ is able to distinguish $(C_0, \mathtt{aux}_0)$ from $(C_p, \mathtt{aux}_p)$ with non-negligible probability then there must exist an index j such that $\mathcal{A}$ can distinguish between $(C_j, \mathtt{aux}_j)$ and $(C_{j+1}, \mathtt{aux}_{j+1})$ with non-negligible probability, which contradicts the security definition of $\mathsf{CollInd}$. Therefore, such an $\mathcal{A}$ cannot exist and the claim holds.

B Separating our Security Definitions

In the following let $\mathcal{CVC} = (\mathsf{CGen}, \mathsf{CCom}, \mathsf{COpen}, \mathsf{CVer}, \mathsf{CCol}, \mathsf{CUpdate}, \mathsf{CProofUpdate})$ be a concise, hiding, and position-binding chameleon vector commitment with indistinguishable collisions.

B.1 ColInd $\wedge$ PosBdg $\not\Longrightarrow$ Hiding

Consider $\mathcal{CVC}' = (\mathsf{CGen}', \mathsf{CCom}', \mathsf{COpen}', \mathsf{CVer}', \mathsf{CCol}', \mathsf{CUpdate}', \mathsf{CProofUpdate}')$:

Construction 5. $\mathsf{CGen}'(1^\lambda, q)$**:** *Output* $\mathsf{CGen}(1^\lambda, q)$.
$\mathsf{CCom}'_{\mathsf{pp}}(m_1, \ldots, m_q)$**:** $(C, \mathtt{aux}^*) \leftarrow (\mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_q)$*, set* $\mathtt{aux} \leftarrow (\mathtt{aux}^*, \bigoplus_{j=1}^{q} m_j)$*, output* $(C, \mathtt{aux})$.
$\mathsf{COpen}'_{\mathsf{pp}}(i, m, \mathtt{aux})$**:** *Parse* $\mathtt{aux} = (\mathtt{aux}^*, x)$*, compute* $\pi_i^* \leftarrow \mathsf{COpen}_{\mathsf{pp}}(i, m, \mathtt{aux}^*)$*, output* (π_i^*, x).
$\mathsf{CVer}'_{\mathsf{pp}}(C, i, m, \pi)$**:** *Parse* $\pi = (\pi^*, x)$*, output* $\mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi^*)$.
$\mathsf{CCol}'_{\mathsf{pp}}(C, i, m, m', \mathtt{td}, \mathtt{aux})$**:** *Parse* $\mathtt{aux} = (\mathtt{aux}^*, x)$*, compute* $\mathtt{aux}^{*\prime} \leftarrow \mathsf{CCol}_{\mathsf{pp}}(C, i, m, m', \mathtt{td}, \mathtt{aux}^*)$*, set* $x' \leftarrow x \oplus m \oplus m'$*, output* $(\mathtt{aux}^{*\prime}, x')$.
$\mathsf{CUpdate}'_{\mathsf{pp}}(C, i, m, m', \pi)$**:** *Parse* $\pi = (\pi^*, x)$*, compute* $(C', U^*) \leftarrow \mathsf{CUpdate}_{\mathsf{pp}}(C, i, m, m', \pi^*)$*, set* $U \leftarrow (U^*, m \oplus m')$*, output* (C', U).
$\mathsf{CProofUpdate}'_{\mathsf{pp}}(C, \pi_j, i, U)$**:** *Parse* $\pi_j = (\pi_j^*, x)$ *and* $U = (U^*, y)$*, compute* $x' \leftarrow x \oplus y$ *and* $(C', \pi_j^{*\prime}) \leftarrow \mathsf{CProofUpdate}_{\mathsf{pp}}(C, \pi_j^*, i, U^*)$*, set* $\pi_j' \leftarrow (\pi_j^{*\prime}, x')$*. Output* (C', π_j').

Lemma 11. *Construction 5 has indistinguishable collisions.*

Proof. Assume there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against $\mathsf{ColInd}^{\mathcal{CVC}'}$ that has a success probability which is non-negligibly bigger than $1/2$. We can then construct the following reduction $\mathcal{B}$ against $\mathsf{ColInd}^{\mathcal{CVC}}$.

On input $(\mathtt{pp}, \mathtt{td})$ the reduction invokes $\mathcal{A}_0(\mathtt{pp}, \mathtt{td})$. The adversary is expected to output $((m_1, \ldots, m_q), (i, m_i'))$. The reduction $\mathcal{B}$ then computes $x = m_i' \oplus \bigoplus_{j=1, j\neq i}^{q} m_j$ and outputs $((m_1, \ldots, m_q), (i, m_i'))$. The game then returns $(C_b, \mathtt{aux}_b^*)$. The reduction sets $\mathtt{aux} \leftarrow (\mathtt{aux}_b^*, x)$ and invokes $\mathcal{A}_1(C_b, \mathtt{aux})$. Eventually $\mathcal{A}_1$ outputs a bit b and the reduction then also outputs this bit.

Observe that $\mathcal{B}$ is efficient and simulates the view of $\mathcal{A}$ perfectly as in $\mathsf{ColInd}^{\mathcal{CVC}'}$. Therefore it holds that

$$\mathrm{Prob}\left[\mathsf{ColInd}_{\mathcal{B}}^{\mathcal{CVC}}(\lambda) = 1\right] = \mathrm{Prob}\left[\mathsf{ColInd}_{\mathcal{A}}^{\mathcal{CVC}'}(\lambda) = 1\right].$$

Since $\mathcal{CVC}$ has indistinguishable collisions, we have

$$\mathrm{Prob}\left[\mathsf{ColInd}_{\mathcal{B}}^{\mathcal{CVC}}(\lambda) = 1\right] = \mathrm{Prob}\left[\mathsf{ColInd}_{\mathcal{A}}^{\mathcal{CVC}'}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

This contradicts the assumption that $\mathcal{A}$ wins with non-negligible probability bigger than $1/2$, therefore such an adversary cannot exist.

Lemma 12. *Construction 5 is position-binding.*

Proof. For the sake of contradiction assume there exists a PPT adversary against $\mathsf{PosBdg}^{\mathcal{CVC}'}$ that wins with non-negligible probability. We can then construct a reduction $\mathcal{B}$ against the position-binding property of $\mathcal{CVC}$.

On input $\mathtt{pp}$, the reduction $\mathcal{B}$ invokes $\mathcal{A}^{\mathsf{CCol}_{\mathsf{pp}}(\cdot,\cdot,\cdot,\cdot,\mathtt{td},\cdot)}(\mathtt{pp})$. To simulate the collision-oracle, the reduction $\mathcal{B}$ parses $\mathtt{aux} = (\mathtt{aux}^*, x)$, computes $\mathtt{aux}^{*\prime}$ using its own collision-oracle, and sets $x' = x \oplus m \oplus m'$. It then

returns $(\mathsf{aux}^{*\prime}, x')$ to the adversary. The adversary $\mathcal{A}$ is expected to output (C, i, m, m', π, π'). The reduction then parses $\pi = (\pi^*, x)$ and $\pi' = (\pi^{*\prime}, x')$ and outputs $(C, i, m, m', \pi^*, \pi^{*\prime})$.

It is easy to see that $\mathcal{B}$ is efficient and perfectly simulates the view of $\mathcal{A}$ as in $\mathsf{PosBdg}^{\mathcal{CVC}'}$. It also holds for $\pi = (\pi^*, x)$ that $\mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi^*)$ verifies whenever $\mathsf{CVer}'_{\mathsf{pp}}(C, i, m, \pi)$ does. Therefore $\mathcal{B}$ always wins exactly when $\mathcal{A}$ does, i.e.

$$\mathrm{Prob}\left[\mathsf{PosBdg}^{\mathcal{CVC}}_{\mathcal{B}}(\lambda) = 1\right] = \mathrm{Prob}\left[\mathsf{PosBdg}^{\mathcal{CVC}'}_{\mathcal{A}}(\lambda) = 1\right].$$

Again, since $\mathcal{CVC}$ is position-binding, we have that

$$\mathrm{Prob}\left[\mathsf{PosBdg}^{\mathcal{CVC}}_{\mathcal{B}}(\lambda) = 1\right] = \mathrm{Prob}\left[\mathsf{PosBdg}^{\mathcal{CVC}'}_{\mathcal{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

Therefore, such an adversary $\mathcal{A}$ cannot exist, hence $\mathcal{CVC}'$ is position-binding as well.

Lemma 13. *Construction 5 is not hiding.*

Proof. Assume $q = 2$ and consider the following adversary $\mathcal{A}$: First, $\mathcal{A}$ outputs $((0,0),(0,1))$. It then receives back $\Pi = (\pi_1, \bot)$, since both vectors only agree in the first position. The adversary $\mathcal{A}$ then simply parses $\pi = (\pi', x)$ and outputs x. Clearly, if the first vector was chosen, we have $x = 0 \oplus 0 = 0$, therefore $\mathcal{A}$ is right. Also, if the second vector was chosen, we have $x = 0 \oplus 1 = 1$, so $\mathcal{A}$ is right as well. This means that $\mathcal{A}$ wins with probability 1 which is non-negligibly bigger than $1/2$ in λ.

B.2 CollInd $\wedge$ Hiding $\not\Longrightarrow$ PosBdg

Consider $\mathcal{CVC}' = (\mathsf{CGen}', \mathsf{CCom}', \mathsf{COpen}', \mathsf{CVer}', \mathsf{CCol}', \mathsf{CUpdate}', \mathsf{CProofUpdate}')$:

Construction 6. $\mathsf{CGen}'(1^\lambda, q)$**:** *Output* $\mathsf{CGen}(1^\lambda, q)$.
$\mathsf{CCom}'_{\mathsf{pp}}(m_1, \ldots, m_q)$**:** *Output* $\mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_q)$.
$\mathsf{COpen}'_{\mathsf{pp}}(i, m, \mathsf{aux})$**:** *compute* $\pi_i^* \leftarrow \mathsf{COpen}_{\mathsf{pp}}(i, m, \mathsf{aux})$*, output* $(\pi_i^*, 0)$.
$\mathsf{CVer}'_{\mathsf{pp}}(C, i, m, \pi)$**:** *Parse* $\pi = (\pi^*, x)$*, output* $\mathsf{CVer}_{\mathsf{pp}}(C, i, m \oplus x, \pi^*)$.
$\mathsf{CCol}'_{\mathsf{pp}}(C, i, m, m', \mathsf{td}, \mathsf{aux})$**:** *Output* $\mathsf{CCol}_{\mathsf{pp}}(C, i, m, m', \mathsf{td}, \mathsf{aux})$.
$\mathsf{CUpdate}'_{\mathsf{pp}}(C, i, m, m', \pi)$**:** *Parse* $\pi = (\pi^*, x)$*, output* $\mathsf{CUpdate}_{\mathsf{pp}}(C, i, m, m', \pi^*)$.
$\mathsf{CProofUpdate}'_{\mathsf{pp}}(C, \pi_j, i, U)$**:** *Parse* $\pi_j = (\pi_j^*, x)$*, compute* $(C', \pi_j^{*\prime}) \leftarrow \mathsf{CProofUpdate}_{\mathsf{pp}}(C, \pi_j^*, i, U)$*, set* $\pi'_j \leftarrow (\pi_j^{*\prime}, x)$*. Output* (C', π'_j).

Lemma 14. *Construction 6 has indistinguishable collisions.*

Proof. It is easy to see that each adversary $\mathcal{A}$ against $\mathsf{CollInd}^{\mathcal{CVC}'}$ is also an adversary against $\mathsf{CollInd}^{\mathcal{CVC}}$ that wins with the same probability. Since $\mathcal{CVC}$ is assumed to have indistinguishable collisions, $\mathcal{CVC}'$ has indistinguishable collisions as well.

Lemma 15. *Construction 6 is hiding.*

Proof. Assume there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ against $\mathsf{Hiding}^{\mathcal{CVC}'}$ that has a success probability which is non-negligibly bigger than $1/2$. We can then construct the following reduction $\mathcal{B}$ against $\mathsf{Hiding}^{\mathcal{CVC}}$.

On input pp, the reduction invokes $\mathcal{A}_0(\mathsf{pp})$, which is expected to output two vectors of messages $(M_0, M_1) = ((m_1^0, \ldots, m_q^0), (m_1^1, \ldots, m_q^1))$. The reduction $\mathcal{B}$ then outputs M_0, M_1. The game returns a commitment C and vector of proofs Π^* which the reduction parses as $\Pi^* = (\pi_1^*, \ldots, \pi_q^*)$. The reduction then sets $\pi_i \leftarrow (\pi_i^*, 0)$,

if $\pi_i^* \neq \perp$, and $\pi_i \leftarrow \perp$ otherwise for $i = 1, \ldots, q$. It finally sets $\Pi \leftarrow (\pi_1, \ldots, \pi_q)$ and invokes $\mathcal{A}_1(C, \Pi)$. Eventually, $\mathcal{A}_1$ outputs a bit b and the reduction $\mathcal{B}$ then outputs this bit as well.

Observe that $\mathcal{B}$ is efficient and perfectly simulates the view of $\mathcal{A}$ as in $\mathsf{Hiding}^{\mathcal{CVC}'}$. Furthermore, $\mathcal{B}$ wins exactly when $\mathcal{A}$ does, i.e.

$$\mathrm{Prob}\left[\mathsf{Hiding}_{\mathcal{B}}^{\mathcal{CVC}}(\lambda) = 1\right] = \mathrm{Prob}\left[\mathsf{Hiding}_{\mathcal{A}}^{\mathcal{CVC}'}(\lambda) = 1\right].$$

Since $\mathcal{CVC}$ is hiding, we have that

$$\mathrm{Prob}\left[\mathsf{Hiding}_{\mathcal{B}}^{\mathcal{CVC}}(\lambda) = 1\right] = \mathrm{Prob}\left[\mathsf{Hiding}_{\mathcal{A}}^{\mathcal{CVC}'}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Therefore, such an adversary $\mathcal{A}$ cannot exists, hence Construction 6 is hiding.

Lemma 16. *Construction 6 is not position-binding.*

Proof. Consider the following adversary $\mathcal{A}$ against the position-binding property of $\mathcal{CVC}'$:

Upon input pp, the adversary $\mathcal{A}$ chooses q random messages $m_1, \ldots, m_q$, computes $(C, \mathsf{aux}) \leftarrow \mathsf{CCom}'_{\mathsf{pp}}(m_1, \ldots, m_q)$, and also computes the opening $\pi_1 \leftarrow \mathsf{COpen}'_{\mathsf{pp}}(1, m_1, \mathsf{aux})$. It then parses $\pi_1 = (\pi_1^*, 0)$ and sets $m_1' \leftarrow m_1 \oplus 1$ and $\pi_1' \leftarrow (\pi_1^*, 1)$. Finally, the adversary $\mathcal{A}$ outputs $(C, 1, m_1, m_1', \pi_1, \pi_1')$.

Observe that $\mathcal{A}$ is obviously efficient. Since $m_1 \oplus 0 = m_1' \oplus 1$, both (m_1, π_1) and (m_1', π_1') verify. Because $m_1 \neq m_1'$, the adversary $\mathcal{A}$ wins with probability 1, which is non-negligibly bigger than $1/2$. Hence $\mathcal{CVC}'$ is not position-binding.

B.3 PosBdg $\wedge$ Hiding $\not\Rightarrow$ CollInd

Consider $\mathcal{CVC}' = (\mathsf{CGen}', \mathsf{CCom}', \mathsf{COpen}', \mathsf{CVer}', \mathsf{CCol}', \mathsf{CUpdate}', \mathsf{CProofUpdate}')$:

Construction 7. $\mathsf{CGen}'(1^\lambda, q)$**:** *Output* $\mathsf{CGen}(1^\lambda, q)$.
$\mathsf{CCom}'_{\mathsf{pp}}(m_1, \ldots, m_q)$**:** *Compute* $(C, \mathtt{aux}^*) \leftarrow \mathsf{CCom}_{\mathsf{pp}}(m_1, \ldots, m_q)$*, set* $\mathtt{aux} \leftarrow (\mathtt{aux}^*, 1)$*, output* $(C, \mathtt{aux})$.
$\mathsf{COpen}'_{\mathsf{pp}}(i, m, \mathtt{aux})$**:** *Parse* $\mathtt{aux} = (\mathtt{aux}^*, x)$*, output* $\mathsf{COpen}_{\mathsf{pp}}(i, m, \mathtt{aux}^*)$.
$\mathsf{CVer}'_{\mathsf{pp}}(C, i, m, \pi)$**:** *output* $\mathsf{CVer}_{\mathsf{pp}}(C, i, m, \pi)$.
$\mathsf{CCol}'_{\mathsf{pp}}(C, i, m, m', \mathtt{td}, \mathtt{aux})$**:** *Parse* $\mathtt{aux} = (\mathtt{aux}^*, x)$*, compute* $\mathtt{aux}^{*\prime} \leftarrow \mathsf{CCol}_{\mathsf{pp}}(C, i, m, m', \mathtt{td}, \mathtt{aux}^*)$*, output* $(\mathtt{aux}^{*\prime}, 0)$.
$\mathsf{CUpdate}'_{\mathsf{pp}}(C, i, m, m', \pi)$**:** *Output* $\mathsf{CUpdate}_{\mathsf{pp}}(C, i, m, m', \pi)$.
$\mathsf{CProofUpdate}'_{\mathsf{pp}}(C, \pi_j, i, U)$**:** *Output* $\mathsf{CProofUpdate}_{\mathsf{pp}}(C, \pi_j, i, U)$.

Lemma 17. *Construction 7 is position-binding.*

Proof. Since $\mathcal{CVC}$ and $\mathcal{CVC}'$ only differ in the fact that $\mathcal{CVC}'$ adds a bit to the auxiliary information, a reduction against the position-binding property of $\mathcal{CVC}$ can be constructed trivially (i.e. whenever the adversary asks his oracle for a collision, the reduction computes the collision using its own oracle and appends a 0). Hence no efficient adversary which breaks the position-binding of $\mathcal{CVC}'$ can exist.

Lemma 18. *Construction 7 is hiding.*

Proof. Since $\mathcal{CVC}$ and $\mathcal{CVC}'$ only differ in the auxiliary information, any adversary against the hiding property of $\mathcal{CVC}'$ is also an adversary against the hiding property of $\mathcal{CVC}$ that wins with the same probability. Hence no efficient adversary which breaks hiding of $\mathcal{CVC}'$ can exist.

Lemma 19. *Construction 7 does not have indistinguishable collisions.*

Proof. Consider the following adversary $\mathcal{A}$ against $\mathsf{CollInd}$ (assume $q = 2$): First, $\mathcal{A}$ outputs $((m_1, m_2), (1, m_1'))$ where m_1, m_1', m_2 are chosen at random. He then receives back $(C_b, \mathtt{aux}_b)$, parses $\mathtt{aux} = (\mathtt{aux}^*, x)$ and outputs x.

If he was given $(C_0, \mathtt{aux}_0)$ then $x = 1$, since $\mathtt{aux}_0$ is found using the collision finding algorithm. Hence $\mathcal{A}$ wins in this case. If he was given $(C_1, \mathtt{aux}_1)$ then $x = 1$, since $\mathtt{aux}_1$ is the output of CCom'. Therefore $\mathcal{A}$ also wins in this case. This means that $\mathcal{A}$ wins with probability 1, which is non-negligibly bigger than $1/2$ in λ.