PointProofs, Revisited

Benoît Libert[1,2], Alain Passelègue[2,3], and Mahshid Riahinia[2]

[1] CNRS, Laboratoire LIP, France
[2] ENS de Lyon, Laboratoire LIP (U. Lyon, CNRS, ENSL, Inria, UCBL), France
mahshid.riahinia@ens-lyon.fr
[3] Inria, France
alain.passelegue@inria.fr

Abstract. Vector commitments allow a user to commit to a vector of length n using a constant-size commitment while being able to locally open the commitment to individual vector coordinates. Importantly, the size of position-wise openings should be independent of the dimension n. Gorbunov, Reyzin, Wee, and Zhang recently proposed PointProofs (CCS 2020), a vector commitment scheme that supports non-interactive aggregation of proofs across multiple commitments, allowing to drastically reduce the cost of block propagation in blockchain smart contracts. Gorbunov *et al.* provide a security analysis combining the algebraic group model and the random oracle model, under the weak n-bilinear Diffie-Hellman Exponent assumption (n-wBDHE) assumption. In this work, we propose a novel analysis that does not rely on the algebraic group model. We prove the security in the random oracle model under the n-Diffie-Hellman Exponent (n-DHE) assumption, which is implied by the n-wBDHE assumption considered by Gorbunov *et al.* We further note that we do not modify their scheme (and thus preserve its efficiency) nor introduce any additional assumption. Instead, we prove the security of the scheme as it is via a strictly improved analysis.

Keywords. Vector commitments, aggregation, provable security.

1 Introduction

As introduced in [12, 22], vector commitments (VCs) allow a user to commit to a vector of messages by generating a short commitment string. Later, the committer should be able to concisely reveal individual coordinates of the message vector. Here, "concisely" means that the partial opening information (called "proof" hereafter) should have constant size – no matter how large the committed vector is – and still convince the verifier that the opened coordinate is correct. As in standard commitments, a vector commitment scheme should satisfy two security properties: (1) A *binding* property which asserts that no efficient adversary should be able to generate a commitment of a vector that can be opened to two different values at the same position, and (2) a *hiding* property which guarantees that revealing a subset of components should not reveal any information about messages at non-revealed positions. Vector commitments enable significant savings in terms of storage, by storing only a constant-size commitment

to a vector instead of commitments to individual coordinates, and bandwidth, thanks to the ability to provably and succinctly open individual positions.

In 2020, Gorbunov, Reyzin, Wee, and Zhang [16] introduced a vector commitment scheme, called PointProofs, which additionally supports non-interactive aggregation of proofs across multiple commitments. Two types of aggregation are supported:

- *Same-commitment aggregation* allows anyone to publicly aggregate single-position proofs for the same vector commitment into a single proof;
- *Cross-commitment aggregation* allows anyone to further aggregate same-commitment-aggregated proofs for distinct commitments (and possibly distinct subsets of positions) and fold them into a single constant-size proof.

Supporting proof aggregations is particularly useful for optimizing distributed applications, such as blockchain propagation. In this context, a third party (the validator) validates blocks by performing operations that depend on data owned by several distinct users. Vector commitments that support proof aggregation make it possible to drastically reduce storage: Instead of storing all users' data, each user can commit to their data individually so that a validator stores only their respective (concise) commitments. When needed, a user can compute proofs for opening positions relevant to the block validation, and aggregate these proofs into a single proof using same-commitment aggregation. Cross-commitment aggregation further allows a validator to aggregate all proofs from distinct users into a single proof that can be included in a block, letting other validators verify the block using a single proof. We also note that PointProofs supports updates of commitments, allowing a user who has already committed to a vector to update some components of this vector without having to compute a new vector commitment from scratch.

In [16], Gorbunov *et al.* consider the use case of blockchain smart contracts. They show that using their scheme instead of former state-of-the-art vector commitments allows a 60%-reduction of bandwidth overheads for propagating a block of transactions. In this work, we focus on improving the security analysis of the scheme without modifying it. We thus refer to [16] for further applications as well as for a detailed efficiency analysis of PointProofs in terms of space and time.

The security requirements of vector commitments with aggregation are easily defined by extending the standard hiding and binding requirements. Specifically, the hiding property requires that (possibly aggregated) proofs for opened positions do not reveal any information about unopened messages. The *binding* property is extended as follows: For same-commitment (resp. cross-commitment) aggregation, binding requires that no efficient adversary be able to come up with a vector commitment C (resp. a set of vector commitments $C_1, \ldots, C_\ell$) together with two conflicting aggregated proofs, which open a position of an output vector commitment to two distinct values. While the PointProofs commitment is perfectly hiding (like its underlying vector commitment scheme [22]), its computational binding property is argued in the algebraic group model, as well as in the random oracle model (ROM), under the n-wBDHE assumption in bilinear

groups.

Recall that the algebraic group model (AGM) is an intermediate idealized model, introduced in [15], that stands between the generic group model and the standard model. As a reminder, in the generic group model, adversaries do not have access to the bit representation of group elements: From the adversary's standpoint, each group element is represented by a unique uniformly random bitstring, and group operations are performed by querying an oracle that returns the representation of the resulting group element (to ensure uniqueness, the oracle keeps track of all group elements known to the adversary). In the generic group model, computational problems and their decisional variants are equivalent, and the Discrete Logarithm problem is provably intractable, as shown by Shoup in [30]. This illustrates why proofs in the generic group model are more often considered as sanity checks rather than proofs of security.

The algebraic group model is a security model weaker than the generic group model, in which one only considers *algebraic adversaries*. Unlike the generic group model, no restriction is made regarding access to the group elements in the AGM: Algebraic adversaries have the same access to group elements as in the standard model. Yet, adversaries are restricted to only handle group elements that are computed by applying group operations to known group elements, similarly as in the generic group model. That is, given elements $g_1, \ldots, g_\ell$ from a multiplicative group $\mathbb{G}$, an algebraic adversary can only access elements of the form $\prod_{i=1}^{\ell} g_i^{\lambda_i}$ for coefficients λ_i's of its choice. Hence, the main difference between the generic group model and the algebraic group model is that the latter allows using coefficients λ_i that depend on the actual bit representation of elements g_i's, while the former forbids it. Despite this minor relaxation, the algebraic group model is still considered as being very idealistic and to be avoided when it is possible.

Our Contribution. In this work, we provide a different security analysis of PointProofs, which relies on the *Generalized Forking Lemma* [1] and the *Local Forking Lemma* [2]. Using these tools, we prove the scheme to be binding in the random oracle model, under the n-Diffie-Hellman Exponent (n-DHE) assumption in groups equipped with a bilinear map. As opposed to the original proof of Gorbunov *et al.* [16], we circumvent the use of algebraic group model, and rely on a weaker assumption; the n-DHE assumption being implied by the aforementioned n-wBDHE assumption. In the ROM, we thus prove the binding property under the same assumption as the one used in the underlying vector commitment scheme due to Libert and Yung [22].

We believe this result to be important in the context of vector commitments as it proves the security of PointProofs as a vector commitment scheme supporting cross-commitment aggregation with constant-size openings under a falsifiable assumption [26] without restricting oneself to algebraic adversaries. Moreover, PointProofs is extremely efficient [32] and, among known candidates supporting cross-commitment aggregation [5, 16, 31], it is the only one that simultaneously provides optimal proof length and linear-size public parameters. Even if we only consider same-commitment aggregation, it implies one of the most efficient

schemes with sub-vector openings among those [13, 16, 17, 32] that simultaneously feature linear-size public parameters and optimal-size proofs (recall that elements of a pairing-friendly group usually have a shorter representation than those of hidden-order groups).

We insist that we do not introduce any additional assumptions in PointProofs, neither do we alter the efficiency of the scheme in the process. Our approach thus provides a strict improvement over the prior analysis. Before outlining our security proof, we first briefly recall the PointProofs construction.

Construction. PointProofs builds on the vector commitment of [22] and can also be seen as an application of the inner product functional commitment scheme of [21], which are both inspired by the broadcast encryption scheme of Boneh, Gentry and Waters [8]. Let n denote the dimension of committed vectors, and consider cyclic groups $\mathbb{G} = \langle g \rangle$ and $\hat{\mathbb{G}} = \langle \hat{g} \rangle$ of prime order p equipped with an asymmetric bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. Let $g_T = e(g, \hat{g})$ be the generator of $\mathbb{G}_T$. The scheme uses public parameters $\left(g, \hat{g}, \{g_i\}_{i \in [2n] \setminus \{n+1\}}, \{\hat{g}_i\}_{i \in [n]}, H\right)$, with $g_i = g^{(\alpha)^i}$ and $\hat{g}_i = \hat{g}^{(\alpha)^i}$, where α is chosen uniformly at random from $\mathbb{Z}_p$, and $H : \{0,1\}^* \to \mathbb{Z}_p$ is a hash function modeled as a random oracle.

To commit to a vector $\mathbf{m} = (m_1, \ldots, m_n) \in \mathbb{Z}_p^n$, one chooses $\gamma \leftarrow U(\mathbb{Z}_p)$ uniformly and computes a multi-base Pedersen commitment [27] of the form

$$C = g^{\gamma} \cdot \prod_{j=1}^{n} g_j^{m_j} = g^{\gamma + \sum_{j=1}^{n} m_i \cdot \alpha^j} \quad ,$$

which can be seen as raising g to the evaluation of a polynomial defined by the coefficients contained in $\mathbf{m}$. To open a position $i \in [n]$ of $\mathbf{m}$ to m_i, the committer reveals a proof

$$\pi_i = g_{n+1-i}^{\gamma} \cdot \prod_{j=1, j \neq i}^{n} g_{n+1-i+j}^{m_j} = \left(C / g^{m_i \cdot \alpha^i}\right)^{\alpha^{n+1-i}} \quad ,$$

which is an element of $\mathbb{G}$ whose discrete logarithm is the same polynomial evaluation as in C, except that the coefficient m_i is lacking, and the polynomial is multiplied by α^{n-i+1}, so that π_i does not depend on the monomial α^{n+1}. This proof can be easily verified by checking that

$$e(C, \hat{g}^{\alpha^{n+1-i}}) = e(\pi_i, \hat{g}) \cdot g_T^{m_i \cdot \alpha^{n+1}} \quad ,$$

where $g_T^{\alpha^{n+1}} = e(g_1, \hat{g}_n)$ is computable from the public parameters.

In order to aggregate multiple proofs $(\pi_i)_{i \in S}$ involving the same commitment C, where $S \subseteq [n]$, anyone can derive randomness for each proof π_i from the random oracle as $t_i \leftarrow H(i, C, S, \mathbf{m}[S])$, with $\mathbf{m}[S]$ being the sub-vector $(m_i)_{i \in S}$, and define the aggregated proof as $\pi_S = \prod_{i \in S} \pi_i^{t_i}$. Verification is achieved in a similar way to the single position case, by additionally verifying the linear

combination for coefficients provided by the random oracle evaluations. That is, the verifier checks that

$$e(C, \hat{g}^{\sum_{i\in S}\alpha^{n+1-i}\cdot t_i}) = e(\pi_S, \hat{g}) \cdot g_T^{\alpha^{n+1}\sum_{i\in S} m_i\cdot t_i} \quad .$$

Finally, the cross-commitment aggregation of proofs $(\pi_{S^{(j)}})_{j\in[\ell]}$ (which might result from the same-commitment-aggregation process) proceeds in a similar way. Again, some randomness $t'_j \leftarrow H'(j, \{C^{(j)}, S^{(j)}, \mathbf{m}^{(j)}[S^{(j)}]\}_{j\in[d]})$ is first derived from random oracle evaluations (of a second random oracle H'), and the cross-commitment aggregated proof is defined to be $\pi = \prod_{j=1}^{\ell}(\pi_{S^{(j)}})^{t'_j}$. Verification is performed in a similar way to the same-commitment-aggregated case: The verifier first derives all random coefficients (the $t_i^{(j)}$ for $i \in S^{(j)}$ for each underlying same-commitment-aggregated proof, as well as the t'_j) before verifying that

$$\prod_{j\in[\ell]} \left(e(C^{(j)}, \hat{g}^{\sum_{i\in S^{(j)}}\alpha^{n+1-i}\cdot t_i^{(j)}})\right)^{t'_j} = e(\pi, \hat{g}) \cdot \prod_{j\in[\ell]} \left(g_T^{\alpha^{n+1}\cdot\sum_{i\in S^{(j)}} m_i^{(j)}\cdot t_i^{(j)}}\right)^{t'_j} \quad . \tag{1}$$

Technical Overview. As already mentioned, our proof strategy relies on the Local [2] and Generalized [1] Forking Lemmas. We first briefly remind the intuition behind these lemmas. The standard Forking Lemma [29] considers the setting in which a probabilistic polynomial time adversary $\mathcal{A}$, given access to a random oracle H, succeeds with non-negligible probability in some experiment which consists of outputting a pair (y, aux), where y lies in the domain of H and aux is some auxiliary information, such that the triplet $(y, H(y), \mathsf{aux})$ satisfies a target condition. Let us denote by $x_1, \ldots, x_q$ the q queries made by $\mathcal{A}$ to the random oracle, and let us assume that y is the j-th query for some $j \in [q]$[4].

The Forking Lemma states that running $\mathcal{A}$ again with the same coins but replacing H with another random oracle H' which satisfies $H'(x_i) = H(x_i)$ for $i < j$, results in $\mathcal{A}$ succeeding again with some non-negligible probability with output $(y, H'(y), \mathsf{aux}')$. The important bit here is that $\mathcal{A}$'s output involves the same y, but now $H'(y)$ differs (with overwhelming probability) from $H(y)$. This new triplet $(y, H'(y), \mathsf{aux}')$ is called a fork. This lemma has notably been used in the context of signature schemes based on applying the Fiat-Shamir paradigm to 3-round identification protocols [14] with special-soundness, where the triplet $(y, H(y), \mathsf{aux})$ is a transcript and the target condition is for it to be valid.

On one hand, the Local Forking Lemma [2] is a refinement of the standard Forking Lemma, which states that one is able to create a fork by replacing H by a random oracle H' that only differs from H on the specific input y.

On the other hand, the Generalized Forking Lemma [1] is an extension to the setting where multiple y's are generated by the adversary. That is, the output of $\mathcal{A}$ is of the form $(y_1, H(y_1), \ldots, y_\ell, H(y_\ell), \mathsf{aux})$ and allows to create forks

[4] Typically, the target condition is sparse and the range of H is exponentially large, therefore $\mathcal{A}$ must query $H(x)$ to succeed with non-negligible probability.

for different y_i's, e.g., a first fork $(y_1, H'(y_1), \ldots, y'_\ell, H(y'_{\ell'}), \mathsf{aux}')$ and a second fork $(y_1, H(y_1), \ldots, y_\ell, H''(y_\ell), \mathsf{aux}'')$. Each fork is obtained by using a different random oracle that outputs the same values as H for all the queries preceding the forking point, and whose values are sampled uniformly at random and independently of H as soon as the forking point is hit. In the previous example, y_1 is the forking point of the first tuple and y_ℓ is that of the second one.

We start by explaining how we reduce binding in the case of same-commitment aggregation to the hardness of the n-DHE problem. First, we recall that the n-DHE problem asks to compute g_{n+1} given $\left(g, \hat{g}, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \{\hat{g}_i\}_{i\in[n]}\right)$ (borrowing the notation from the construction described in the previous part). Consider an adversary $\mathcal{A}$ that manages to break the binding property with non-negligible probability. Our n-DHE solver sets the public parameters to be the n-DHE instance and uses $\mathcal{A}$ as follows.

$\mathcal{A}$ is able to generate a tuple $(C, S_0, S_1, \mathbf{m}_0[S_0], \mathbf{m}_1[S_1], \pi_0, \pi_1)$ containing a commitment C as well as two sets $S_0, S_1 \subseteq [n]$ such that $S_0 \cap S_1 \neq \emptyset$ and valid proofs π_0, π_1 with respect to sub-vectors $\mathbf{m}_0[S_0] \in \mathbb{Z}_p^{|S_0|}$, $\mathbf{m}_1[S_1] \in \mathbb{Z}_p^{|S_1|}$ such that $\mathbf{m}_0[i^*] \neq \mathbf{m}_1[i^*]$ for some $i^* \in S_0 \cap S_1$. The following holds for this output:

$$e(C, \hat{g}^{\alpha^{n+1-i}})^{\sum_{i\in S_b} t_i^{(b)}} = e(\pi_b, \hat{g}) \cdot g_T^{\alpha^{n+1} \sum_{i\in S_b} m_i \cdot t_i^{(b)}} , \tag{2}$$

where $t_i^{(b)} = H(i, C, S_b, \mathbf{m}_b)$, for both $b = 0$ and $b = 1$.

Random oracle queries made by $\mathcal{A}$ are of the form $(i, C, S, \mathbf{m}[S])$, and both $(i^*, C, S_0, \mathbf{m}_0[S_0])$ and $(i^*, C, S_1, \mathbf{m}_1[S_1])$ must have been queried for Equation 2 to hold. One can then apply the Generalized Forking Lemma in order to create forks. In our case, the auxiliary information aux is the pair of proofs. We aim to obtain two forks: a first one of the form $(C, S_0, S'_1, \mathbf{m}_0[S_0], \mathbf{m}'_1[S'_1], \pi'_0, \pi'_1)$, related to a random oracle H' such that $H(i^*, C, S_0, \mathbf{m}_0[S_0]) \neq H'(i^*, C, S_0, \mathbf{m}_0[S_0])$, and a second one of the form $(C, S''_0, S_1, \mathbf{m}''_0[S''_0], \mathbf{m}_1[S_1], \pi''_0, \pi''_1)$, related to H'' and such that $H(i^*, C, S_1, \mathbf{m}_1[S_1]) \neq H''(i^*, C, S_1, \mathbf{m}_1[S_1])$. In addition, we need that for all $i \in S_0 \setminus \{i^*\}$, $H(i, C, S_0, \mathbf{m}_0[S_0]) = H'(i, C, S_0, \mathbf{m}_0[S_0])$ and that for all $i \in S_1 \setminus \{i^*\}$, $H(i, C, S_1, \mathbf{m}_1[S_1]) = H''(i, C, S_1, \mathbf{m}_1[S_1])$.

Such forks are obtained by applying the Generalized Forking Lemma as follows: to ensure that the mentioned conditions about the values of H, H', H'' are satisfied, we design the reduction algorithm to simulate the random oracle such that all hash values for inputs $(i, C, S_0, \mathbf{m}_0[S_0])$ where $i \in S_0 \setminus \{i^*\}$ (resp. $(i^*, C, S_1, \mathbf{m}_1[S_1])$ where $i \in S_1 \setminus \{i^*\}$) are set before setting the hash values for $(i^*, C, S_0, \mathbf{m}_0[S_0])$ (resp. $(i^*, C, S_1, \mathbf{m}_1[S_1])$). More precisely, our reduction first makes a random guess about the value of i^*, which is a correct guess with probability $1/n$. Then, on receiving a query $(i, C, S, \mathbf{m}[S])$, it checks whether $i^* \in S$. If so, it first defines the hash values for inputs $(i, C, S, \mathbf{m}[S])$ for all $i \neq i^*$, and finally sets the value of $H(i^*, C, S, \mathbf{m}[S])$ at the end. Doing so, the conditions on the values of H, H', H'' are satisfied, and the Generalized Forking Lemma guarantees that the two desired forks can be obtained.

To conclude the proof, one simply re-writes Equation (2) with each fork using $b = 0$ and $b = 1$, respectively. The first fork leads to equation:

$$e(C, \hat{g}^{\alpha^{n+1-i}})^{\sum_{i \in S_0} t'_i} = e(\pi'_0, \hat{g}) \cdot g_T^{\alpha^{n+1} \sum_{i \in S_0} m_i \cdot t'_i} \quad ,$$

where $t'_i = H'(i, C, S_0, \mathbf{m}_0)$. A similar equation is obtained for the second fork. Thanks to the conditions satisfied by H, H', H'' values, it follows that for all $i \neq i^*$, we have $t_i^{(0)} = t'_i$ and $t_i^{(1)} = t''_i$, where $t''_i = H''(i, C, S_1, \mathbf{m}_1)$.

Finally, combining these equations allows to recover an equation that depends only on terms $e(g_{n+1}, \hat{g})$, $e(\pi_1, \hat{g})$, $e(\pi''_1, \hat{g})$, $e(\pi_0, \hat{g})$, $e(\pi'_0, \hat{g})$. Setting aside the term $e(g_{n+1}, \hat{g})$ and focusing on the $\mathbb{G}$-component, the reduction manages to compute g_{n+1} as a combination of $\pi_1, \pi''_1, \pi_0, \pi'_0$, which is the solution to the n-DHE problem.

In the supplementary material, we also propose a different proof which relies on the Local Forking Lemma, and compare this approach to the above one. These two proofs provide different bounds for the advantage and run-time of the reduction, and we believe that, in the context of PointProofs, the proof based on the Generalized Forking Lemma provides a tighter reduction.

The case of cross-commitment aggregation follows a similar strategy, but this time we reduce the binding property of the cross-commitment aggregations to that of the same-commitment aggregations. This proof is also in the random oracle model. Given an adversary $\mathcal{A}$ against the binding property of the cross-commitment scheme, we construct an adversary $\mathcal{B}$ against the binding property of the same-commitment scheme as follows: $\mathcal{B}$ splits the random oracles queries made by $\mathcal{A}$ into two categories: those corresponding to same commitment evaluations (the $t_i^{(j)}$'s in the above description), which are redirected by $\mathcal{B}$ to the random oracle to which it has access, and those corresponding to cross-commitment evaluations (the t'_j's). $\mathcal{B}$ simulates the response to the queries of the second category. It first runs $\mathcal{A}$ which outputs two tuples of the form

$$\left(\pi_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j \in [d_0]}\right), \qquad \left(\pi_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j \in [d_1]}\right) \quad ,$$

that breaks the binding property. In other words, there exist $j_0 \in [d_0]$, $j_1 \in [d_1]$ such that $C_0^{(j_0)} = C_1^{(j_1)}$ and $\mathbf{m}_0^{(j_0)}[i^*] \neq \mathbf{m}_1^{(j_1)}[i^*]$ for some $i^* \in S_0^{(j_0)} \cap S_1^{(j_1)}$.

$\mathcal{B}$ then uses the Local Forking Lemma twice. In the first fork, it redefines the hash value of the query $(j_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}\}_{j \in [d_0]})$. The output of $\mathcal{A}$ in this fork involves the same first collection of commitments $\{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j \in [d_0]}$ as in the initial execution, together with a proof π'_0. The equation obtained by running the verification algorithm (Equation 1) on this collection of commitments is such that the value of all t'_j's are the same as $\mathcal{A}$'s first run except if $j = j_0$. By forking a second time on query $(j_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}\}_{j \in [d_1]})$ and repeating the same arguments, $\mathcal{B}$ obtains two pairs of equations, that can be combined using a similar gymnastic as in the same-commitment proof to recover a valid attack against the binding property of the same-commitment aggregations.

We emphasize that using the Generalized Forking Lemma instead of the Local Forking Lemma does not seem to be an option in this case. Indeed, focusing on the first fork, our proof relies on the capacity to create a fork for the query $(j_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}\}_{j\in[d_0]})$ without changing every other hash values $H(j, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}\}_{j\in[d_0]})$, for $j \in [d_0]\setminus\{j_0\}$. Using the Generalized Forking Lemma would require to set all the latter hash values before the value for $(j_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}\}_{j\in[d_0]})$, but contrary to the previous case, one cannot simply guess j_0 as it lies in an arbitrary range. For this reason, we rely on the Local Forking Lemma for proving the binding property in this case.

Related Work. Historically, vector commitments with logarithmic-size openings have been known for 3 decades, with the folklore construction based on Merkle trees [23]. In 2008, Catalano *et al.* [11] called for constructions with constant-size openings with the motivation of compressing proofs in zero-knowledge databases [24]. Vector commitments with $O(1)$-size openings appeared later on [12, 22], with a first realization based on a q-type assumption put forth by Libert and Yung [22]. Catalano and Fiore [12] obtained constructions from the standard RSA assumption and the Computational Diffie-Hellman assumption in pairing-friendly groups. Peikert *et al.* [28] recently came up with the first candidate under standard lattice assumptions. Meanwhile, applications of vector commitments were considered in the context of zero-knowledge databases [22], verifiable data streaming [18], authenticated dictionaries [20, 34], de-centralized storage [10], succinct arguments [3, 19], cryptocurrencies [13, 32] and blockchain transactions [3, 16], or certificates of collective knowledge [25].

Back in 2010, Kate, Zaverucha and Goldberg [17] introduced the related notion of polynomial commitments, which allows committing to a polynomial in such a way that the committer can later prove that the committed polynomial evaluates to specific values on certain inputs. They showed that their scheme enables batch openings, where a constant-size proof convinces the verifier about multiple polynomial evaluations at once. Libert, Ramanna and Yung [21] suggested inner product functional commitments, which imply both vector commitments and polynomial commitments.

Lai and Malavolta [19] and Boneh *et al.* [3] independently generalized VCs by introducing the notion of sub-vector commitments, where the sender can generate a short proof π_S that opens a sub-vector $\mathbf{m}[S]$ of $\mathbf{m}$, for some subset $S \subseteq [n]$. Lai and Malavolta [19] provided instantiations in hidden order groups and also showed that a variant of the Catalano-Fiore commitment [12] allows sub-vector openings under a constant-size assumption (namely, the cube-CDH assumption) in pairing-friendly groups. We remark that sub-vector commitments can also be realized from polynomial commitments with batch openings, as shown by Camenisch *et al.* [9, Section 3.1]. However, their construction intentionally prevents proof aggregation.

The property of proof aggregation was considered in [3, 10, 16, 31, 34]. Boneh, Bünz and Fisch [3] and Tomescu *et al.* [32] independently considered same-commitment aggregation in hidden-order groups and under q-type assumptions

in pairing-friendly groups, respectively. Campanelli *et al.* [10] defined incrementally aggregatable VCs, where different sub-vector openings can be merged into a constant-size opening for the union of their sub-vectors. Moreover, aggregated proofs support further aggregation. They showed how to realize incrementally aggregatable VCs in hidden-order groups.

Gorbunov *et al.* [16] proposed PointProofs as the first construction enabling cross-commitment aggregation. They also showed [16, Appendix A] that a variant of the Lai-Malavolta commitment [19] supports same-commitment aggregation, but still at the cost of quadratic-size public parameters. As underlined in [16], same-commitment aggregation implies sub-vector openings by having the committer aggregate same-commitment proofs.

In [16, Appendix B], Gorbunov *et al.* also showed that the restriction to algebraic adversaries is unnecessary if one just aims at a relaxed binding property – which may be sufficient in certain blockchain applications – which assumes honestly generated commitments. Here, we remove the restriction to algebraic adversaries even when commitments are adversarially generated.

The recent Hyperproofs construction of Srinivasan *et al.* [31] also allows cross-commitment aggregation and makes it possible to update all proofs in sub-linear time when the vector changes. On the downside, it loses the conciseness of PointProofs as its proofs have size $O(\log n)$. Besides [16, 31], we are only aware of one alternative VC scheme supporting cross-commitment aggregation, which was proposed by Boneh *et al.* [5]. However, it is only known to be secure in the combined AGM+ROM setting.

2 Preliminaries

We use λ to denote the security parameter. For a natural integer $n \in \mathbb{N}$, the set $\{1, 2, \cdots, n\}$ is denoted by $[n]$. We denote by $\mathsf{negl}(\lambda)$ a negligible function in λ, and PPT stands for probabilistic polynomial-time. For a finite set S, we write $x \xleftarrow{R} S$ to denote that x is sampled uniformly at random from S. We denote vectors by bold characters e.g. $\mathbf{m}$. For a vector $\mathbf{m} = (m_1, \ldots, m_n)$ and a subset of indices $S \subseteq [n]$, we denote by $\mathbf{m}[S]$ the sub-vector $(m_{i_1}, \ldots, m_{i_{|S|}})$, where $S = \{i_1, \ldots, i_{|S|}\}$ with $i_j < i_{j+1}$ for each $j \in \{1, \ldots, |S| - 1\}$. For a single index $i \in [n]$, we sometimes denote m_i by $\mathbf{m}[i]$.

2.1 Bilinear Maps and Complexity Assumptions

Let $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ be cyclic groups of prime order p that are equipped with a bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$. We rely on a parameterized assumption which was introduced by Boneh, Gentry and Waters [8]. While this assumption was originally defined using symmetric pairings [4, 8], we consider a natural extension to asymmetric pairings, which were used in PointProofs.

Definition 1. *Let $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ be asymmetric bilinear groups of prime order p. The n-**Diffie-Hellman Exponent** (n-DHE) problem is, given*

$$(g, g^{\alpha}, g^{(\alpha^2)}, \ldots, g^{(\alpha^n)}, g^{(\alpha^{n+2})}, \ldots, g^{(\alpha^{2n})}, \hat{g}, \hat{g}^{\alpha}, \hat{g}^{(\alpha^2)}, \ldots, \hat{g}^{(\alpha^n)})$$

where $\alpha \xleftarrow{R} \mathbb{Z}_p$, $g \xleftarrow{R} \mathbb{G}$, $\hat{g} \xleftarrow{R} \hat{\mathbb{G}}$, *to compute* $g^{(\alpha^{n+1})}$.

We note that this assumption is the same as the one underlying the binding property of the vector commitment scheme of Libert and Yung [22]. As an artifact of the algebraic group model, Gorbunov *et al.* [16] considered a stronger version of the above assumption where $g^{(\alpha^{2n+1})}, \ldots, g^{(\alpha^{3n})}$ are also given.

2.2 Generalized Forking Lemma

Here, we recall the Generalized Forking Lemma as stated in [1] which is later used in one of our proofs.

Game $G_{\mathcal{A},\pi}$

$f = (\rho, h_1, \ldots, h_Q) \xleftarrow{R} \mathcal{F}$
$(L, \{out_\ell\}_{\ell \in L}) \leftarrow \mathcal{A}(\pi, f)$

If $L = \emptyset$ output 0.

Else, let $L = (\ell_1, \ldots, \ell_m)$ such that $\ell_1 \leq \cdots \leq \ell_m$

For $i = 1, \ldots, m$ do

  Set $succ_i \leftarrow 0$, $k_i \leftarrow 0$, $k_{max} \leftarrow 8mQ/\epsilon \cdot \ln(8m/\epsilon)$

  While $succ_i = 0$ and $k_i < k_{max}$ do

    $f' \xleftarrow{R} \mathcal{F}$ such that $f'_{|\ell_i} = f_{|\ell_i}$

    Let $f' = (\rho, h_1, \ldots, h_{\ell_i - 1}, h'_{\ell_i}, \ldots, h'_Q)$

    $(L', \{out'_\ell\}_{\ell \in L'}) \leftarrow \mathcal{A}(\pi, f')$

    If $h'_{\ell_i} \neq h_{\ell_i}$ and $L' \neq \emptyset$ and $\ell_i \in L'$ then
      keep out'_{ℓ_i} and set $succ_i \leftarrow 1$
    Else
      set $k_i \leftarrow k_i + 1$

If $succ_i = 1$ for all $i \in [m]$, then
  output $(L, \{out_\ell\}_{\ell \in L}, \{out'_\ell\}_{\ell \in L})$
Else output 0

Fig. 1. Game $G_{\pi,\mathcal{A}}$ where an algorithm is forked in the Generalized Forking Lemma.

Let $\mathsf{SAMP}()$ be a probabilistic algorithm that returns a value π which we think of as parameters. Also, consider an algorithm $\mathcal{A}$ that takes as input the parameters π and uses some randomness $f = (\rho, h_1, \ldots, h_Q)$, where ρ is value of the random tape of $\mathcal{A}$ and $h_1, \ldots, h_Q$ are responses received by querying a random oracle $H : \{0,1\}^* \to \mathbb{Z}_p$, and Q is the maximal number of the hash queries. We denote by $\mathcal{F}$ the space of all such randomness. Also, for any $f \in \mathcal{F}$ and $i \in [Q]$, we denote by $f_{|i}$ the sub-vector $(\rho, h_1, \ldots, h_{i-1})$. Algorithm $\mathcal{A}$

outputs a pair $(L, \{out_\ell\}_{\ell \in L})$, where L is a subset of $[Q]$, and each out_ℓ is a string, for $\ell \in L$. We consider $L = \emptyset$ as failure, and $L \neq \emptyset$ as success. Let ϵ be the probability that the output of $\mathcal{A}(\pi, f)$ is successful. We define the game $G_{\pi,\mathcal{A}}$ as in Figure 1 parameterized by π and $\mathcal{A}$, where $\pi \xleftarrow{R} \mathsf{SAMP}()$. We now state the lemma as proven by Bagherzandi *et al.* [1] and used in [7].

Lemma 1 (Generalized Forking Lemma [1]). *Let* SAMP*,* $\mathcal{A}$*, and* $H : \{0,1\}^* \to \mathbb{Z}_p$ *be as described, where* $\mathcal{A}$ *runs in time* τ *and succeeds with probability* ϵ*. If* $p > 8mQ/\epsilon$*, then the game* $G_{\pi,\mathcal{A}}$ *runs in time at most* $\tau \cdot 8m^2Q/\epsilon \cdot \ln(8m/\epsilon)$*, and is successful with probability at least* $\epsilon/8$*.*

2.3 Local Forking Lemma

We also recall the Local Forking Lemma of Bellare *et al.* [2], which will be used in our proof for the cross-commitment case. The difference with the classical Forking Lemma is that the random oracle is only reprogrammed on the forking point, instead of all points from the forking point onwards.

Let $\mathsf{SAMP}()$ be a probabilistic algorithm that returns a value π which we think of as parameters. We also consider a deterministic algorithm $\mathcal{A}$, that given $\pi \xleftarrow{R} \mathsf{SAMP}()$, and having access to a random oracle $H \in \mathcal{H}$, outputs an integer $\alpha \geq 0$, and a string x. We consider $\alpha = 0$ as failure, and $\alpha \geq 1$ as success. If $\alpha \geq 1$, we require x to be the α-th query that $\mathcal{A}$ has issued to the oracle H. We consider the two following games parameterized by π and $\mathcal{A}$, where $\pi \xleftarrow{R} \mathsf{SAMP}()$:

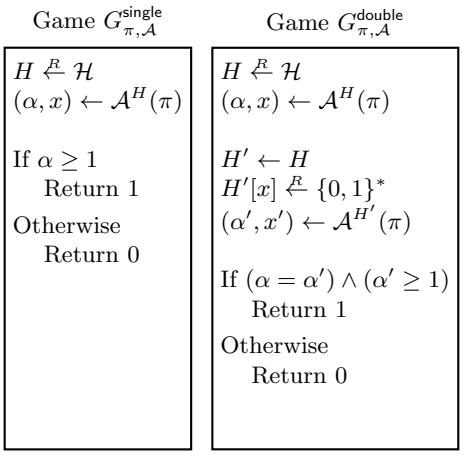

Fig. 2. Game $G^{\mathsf{single}}_{\pi,\mathcal{A}}$, and Game $G^{\mathsf{double}}_{\pi,\mathcal{A}}$, where the local forking happens in the latter.

We now recall the statement of the lemma.

Lemma 2 (Local Forking Lemma, [2]). *Let* SAMP *and* $\mathcal{A}$ *be as described, and let* q *be the number of* H*-queries issued by* $\mathcal{A}$*. It holds that:*

$$\Pr[G^{\mathsf{double}}_{\pi,\mathcal{A}} = 1] \geq 1/q \cdot \Pr[G^{\mathsf{single}}_{\pi,\mathcal{A}} = 1]^2.$$

2.4 Vector Commitments with Aggregation

We recall the formal definition of vector commitments with aggregation introduced in [16]. As in [16], we divide the definition into two parts; the case of same-commitment aggregation, and the case of cross-commitment aggregation.

2.4.1 Same-Commitment Aggregation

Definition 2 (Vector Commitment with Same-Commitment Aggregation [16]). A vector commitment with same-commitment aggregation for message space $\mathcal{M}$ consists of six algorithms $\mathsf{Setup}, \mathsf{Commit}, \mathsf{UpdateCommit}, \mathsf{Prove}, \mathsf{Aggregate}, \mathsf{Verify}$ as follows:

$\mathbf{Setup}(1^\lambda, 1^n) \to \mathsf{pp}$: On input the security parameter λ and a number n which is the length of underlying vector of a commitment in the scheme, it outputs public parameters pp that is used by all other algorithms.

$\mathbf{Commit}_{\mathsf{pp}}(\mathbf{m}; \mathsf{aux}) \to C$: On input a message vector $\mathbf{m} = (m_1, \ldots, m_n)$ of length n, uses some auxiliary information (i.e. randomness) aux to output a commitment C.

$\mathbf{UpdateCommit}_{\mathsf{pp}}(C, S, \mathbf{m}[S], \mathbf{m}'[S], \mathsf{aux}) \to C'$: Takes as input a commitment C, a subset $S \subseteq [n]$, and $\mathbf{m}[S] = (m_i)_{i \in S}$ as the underlying message vector of C, and uses some auxiliary information aux to change $\mathbf{m}[S]$ to $\mathbf{m}'[S] = (m'_i)_{i \in S}$ and outputs a new commitment C' for this new vector of messages.

$\mathbf{Prove}_{\mathsf{pp}}(i, m_i, \mathsf{aux}) \to \pi_i$: On input an index $i \in [n]$ and a message bit $m_i \in \mathcal{M}$, uses the auxiliary information aux that was used in the algorithm Commit to output a proof π_i for this message bit.

$\mathbf{Aggregate}_{\mathsf{pp}}(C, S, \mathbf{m}[S], \{\pi_i\}_{i \in S}) \to \pi_S$: Takes as input a commitment C, a subset $S \subseteq [n]$, a subset of message bits $\mathbf{m}[S] = (m_i)_{i \in S}$, and a set of proofs $\{\pi_i\}_{i \in S}$ where each π_i for $i \in S$ is the proof generated for m_i using the algorithm Prove. It outputs an aggregated proof π_S.

$\mathbf{Verify}_{\mathsf{pp}}(C, S, \mathbf{m}[S], \pi_S) \to b$: On input a commitment C, a subset $S \subseteq [n]$, a sub-vector of messages $\mathbf{m}[S]$, and an aggregated proof π_S, and outputs a bit $b \in \{0, 1\}$.

We require a vector commitment scheme with same-commitment aggregation to satisfy the following properties:

Correctness of Opening. For all λ, n, $\mathbf{m} = (m_1, \ldots, m_n) \in \mathcal{M}^n$, and $S \subseteq [n]$,

$$\Pr\left[\mathsf{Verify}(C, S, \mathbf{m}[S], \pi_S) = 1 : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ C \leftarrow \mathsf{Commit}_{\mathsf{pp}}(\mathbf{m}; \mathsf{aux}) \\ \{\pi_i \leftarrow \mathsf{Prove}_{\mathsf{pp}}(i, m_i, \mathsf{aux})\}_{i \in S} \\ \pi_S \leftarrow \mathsf{Aggregate}(C, S, \mathbf{m}[S], \{\pi_i\}_{i \in S}) \end{array}\right] = 1.$$

Correctness of Updates. For all parameters λ, n, message vectors $\mathbf{m} = (m_1, \ldots, m_n), \mathbf{m}' = (m'_1, \ldots, m'_n) \in \mathcal{M}^n$, subset $S \subseteq [n]$, and aux such that $m_i = m'_i$ for all $i \in [n] \setminus S$, we have

$$\mathsf{UpdateCommit}_{\mathsf{pp}}(C, S, \mathbf{m}[S], \mathbf{m}'[S], \mathsf{aux}) = \mathsf{Commit}_{\mathsf{pp}}(\mathbf{m}'; \mathsf{aux}),$$

where $C \leftarrow \mathsf{Commit}_{\mathsf{pp}}(\mathbf{m}; \mathsf{aux})$, and $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$.

Binding. For all λ, n, and any PPT adversary $\mathcal{A}$, the probability of finding a tuple $(C, S_0, S_1, \mathbf{m}_0[S_0], \mathbf{m}_1[S_1], \pi_0, \pi_1)$, such that

$$\mathsf{Verify}(C, S_0, \mathbf{m}_0[S_0], \pi_0) = \mathsf{Verify}(C, S_1, \mathbf{m}_1[S_1], \pi_1) = 1$$

and $\mathbf{m}_0[i] \neq \mathbf{m}_1[i]$ for some $i \in S_0 \cap S_1$, is negligible in λ.

2.4.2 Cross-Commitment Aggregation

Definition 3 (Vector Commitment with Cross-Commitment Aggregation [16]). A vector commitment with cross-commitment aggregation for message space $\mathcal{M}$ consists of the six algorithms Setup, Commit, $\mathsf{UpdateCommit}$, Prove, $\mathsf{Aggregate}$, Verify as in the same-commitment case, and two additional algorithms $\mathsf{AggregateAcross}$, and $\mathsf{VerifyAcross}$ that are as follows:

- $\mathsf{AggregateAcross}_{\mathsf{pp}}(\{C_j, S_j, \mathbf{m}_j[S_j], \pi_j\}_{j \in [d]}) \to \pi$: Takes as input a collection of commitments C_j together with each of their aggregated proofs π_j with respect to subset S_j and message sub-vector $\mathbf{m}_j[S_j]$, and outputs a cross-aggregated proof π.
- $\mathsf{VerifyAcross}_{\mathsf{pp}}(\pi, \{C_j, S_j, \mathbf{m}_j[S_j], \pi_j\}_{j \in [d]}) \to b$: Given a cross-aggregated proof π, and a collection of underlying commitments, subsets, and message sub-vectors $\{C_j, S_j, \mathbf{m}_j[S_j], \pi_j\}_{j \in [d]}$, this algorithm outputs a bit $b \in \{0, 1\}$.

We require a vector commitment scheme with cross-commitment aggregation to satisfy the correctness of opening as in Definition 2 extended to cross-commitment aggregations. Also, it should satisfy an extension of binding property as follows:

Binding (for cross-commitments). For all λ, n, and any PPT adversary $\mathcal{A}$, the probability of finding the two following tuples with the following described properties is negligible in λ:

$$(\pi_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[\ell_0]}), \text{ and } (\pi_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[\ell_1]}),$$

such that

$$\begin{aligned}&\mathsf{VerifyAcross}(\pi_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[\ell_0]})\\ &\qquad = \mathsf{VerifyAcross}(\pi_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[\ell_1]}) = 1\end{aligned}$$

and there exist $j_0 \in [\ell_0]$, and $j_1 \in [\ell_1]$, for which it holds that $C_0^{(j_0)} = C_1^{(j_1)}$, and $\mathbf{m}_0^{(j_0)}[i] \neq \mathbf{m}_1^{(j_1)}[i]$, for some $i \in S_0^{(j_0)} \cap S_1^{(j_1)}$.

2.4.3 Statistical Hiding

As stated in [16], one can optionally require a vector commitment scheme to generate commitments that are hiding. Intuitively, this property requires that any commitment C that is generated in the scheme must reveal no information about its underlying message vector $\mathbf{m}$. Also, any proof π_i generated with respect to the i-th element m_i of a message vector $\mathbf{m} = (m_1, \ldots, m_n)$ should not reveal any information about any other element m_j of the message, where $j \neq i$.

Since PointProofs were already proven to be *statistically hiding* in [16], we rather focus only on the binding property and do not further detail hiding. We refer the reader to [16] for a details regarding the hiding property.

3 The Case of Same-Commitment Aggregation

We first consider the simpler variant of PointProofs [16] which only allows aggregating proofs for sub-vectors contained in the same commitment. This construction implicitly uses the functional commitment scheme of [21] to aggregate proofs using randomizers derived from a random oracle. Its description is as follows.

Setup$(1^\lambda, 1^n)$: To generate public parameters, do the following:

1. Choose bilinear groups $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T)$ of prime order $p > 2^\lambda$ and $g \xleftarrow{R} \mathbb{G}$, $\hat{g} \xleftarrow{R} \hat{\mathbb{G}}$.
2. Pick a random $\alpha \xleftarrow{R} \mathbb{Z}_p^*$ and compute $g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n} \in \mathbb{G}$ as well as $\hat{g}_1, \ldots, \hat{g}_n \in \hat{\mathbb{G}}$, where $g_i = g^{(\alpha^i)}$ for each $i \in [2n] \setminus \{n+1\}$ and $\hat{g}_i = \hat{g}^{(\alpha^i)}$ for each $i \in [n]$.
3. Choose a hash function $H : \{0,1\}^* \to \mathbb{Z}_p$ that will be modeled as a random oracle in the analysis.

The public parameters are defined to be

$$\mathsf{pp} = \left(g, \hat{g}, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \{\hat{g}_i\}_{i\in[n]}, H\right)$$

and the trapdoor is $tk = g_{n+1} = g^{(\alpha^{n+1})}$.[5]

Commit$_{\mathsf{pp}}(m_1, \ldots, m_n; \mathsf{aux})$: To commit to a vector $(m_1, \ldots, m_n) \in \mathbb{Z}_p^n$, choose $\gamma \xleftarrow{R} \mathbb{Z}_p$ and compute

$$C = g^{\gamma} \cdot \prod_{j=1}^{n} g_j^{m_j}.$$

The output is C and the auxiliary information is $\mathsf{aux} = (m_1, \ldots, m_n, \gamma)$.

UpdateCommit$_{\mathsf{pp}}(C, S, \mathbf{m}[S], \mathbf{m}'[S], \mathsf{aux})$: Given $C \in \mathbb{G}$ and the state information $\mathsf{aux} = (m_1, \ldots, m_n, \gamma)$, choose $\gamma' \xleftarrow{R} \mathbb{Z}_p$ and compute

$$C' = g^{\gamma'} \cdot C \cdot \prod_{j\in S} g_j^{\mathbf{m}'[j]-\mathbf{m}[j]}$$

together with $\mathsf{aux}' = (\bar{m}_1, \ldots, \bar{m}_n, \gamma + \gamma')$, where $\bar{m}_i = m'_i$ if $i \in S$ and $\bar{m}_i = m_i$ if $i \notin S$.

Prove$_{\mathsf{pp}}(m_i, i, \mathsf{aux})$: Parse aux as $(m_1, \ldots, m_n, \gamma)$ and compute

$$\pi_i = g_{n+1-i}^{\gamma} \cdot \prod_{j=1, j\neq i}^{n} g_{n+1-i+j}^{m_j}. \tag{3}$$

The opening of C at position i consists of $\pi_i \in \mathbb{G}$.

Aggregate$_{\mathsf{pp}}(C, S, \mathbf{m}[S], \{\pi_i\}_{i\in S})$: Given a commitment $C \in \mathbb{G}$, a sub-vector $\mathbf{m}[S]$ with $S \subseteq [q]$, and the corresponding proofs $\{\pi_i\}_{i\in S}$, compute

$$\pi_S = \prod_{i\in S} \pi_i^{t_i}$$

where $t_i = H(i, C, S, \mathbf{m}[S])$.

Verify$_{\mathsf{pp}}(C, S, \mathbf{m}[S], \pi_S)$: Given $\pi_S \in \mathbb{G}$ return 1 if $C \in \mathbb{G}$ and

$$e(C, \prod_{i\in S} \hat{g}_{n+1-i}^{t_i}) = e(\pi_S, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S} m[i]\cdot t_i} \tag{4}$$

where $t_i = H(i, C, S, \mathbf{m}[S])$ for each $i \in S$. Otherwise, it returns 0.

In [16], the scheme was proven binding in the algebraic group model and in the random oracle model. We now show that, using the Forking Lemma [29] (more precisely, its generalization used in [1,7]), we can prove its security in the random oracle model (i.e., without using the algebraic group model) under the

[5] The trapdoor is only used to prove the hiding property.

n-DHE assumption, which already underlies the binding property of the vector commitment scheme in [22].

In Supplementary Material A, we provide an alternative proof using the Local Forking Lemma [2] and give a comparison between the advantage/running-time ratios of the two reductions.

We note that the security analysis of [16] highlighted the necessity of including S and $\mathbf{m}[S]$ among the inputs of the hash function when the coefficient $\{t_i\}_{i\in S}$ are computed in the aggregation algorithm. Consistently with this observation, the proof of Theorem 1 crucially relies on the fact that $(S, \mathbf{m}[S])$ are hashed along with (i, C).

In order to rely on the General Forking Lemma, we need to answer random oracle queries in a careful way, by adapting a technique used by Boneh *et al.* [7] in the context of multi-signatures supporting key aggregation.

Theorem 1. *The above commitment is binding in the random oracle model if the n-DHE assumption holds.*

Proof. Suppose that the adversary $\mathcal{A}$ is able to generate a commitment C as well as two sets $S_0, S_1 \subset [n]$ such that $S_0 \cap S_1 \neq \emptyset$ and convincing proofs π_0, π_1 respectively for sub-vectors $\mathbf{m}_0[S_0] \in \mathbb{Z}_p^{|S_0|}$, $\mathbf{m}_1[S_1] \in \mathbb{Z}_p^{|S_1|}$ such that $\mathbf{m}_0[i] \neq \mathbf{m}_1[i]$ for some $i \in S_0 \cap S_1$. Let $\Pr[\neg\mathsf{Bind}]$ be the probability of $\mathcal{A}$ generating such tuple. In the random oracle model, we build an algorithm $\mathcal{C}$ that uses $\mathcal{A}$ to solve the n-DHE problem. Let q_H be the maximum number of queries that $\mathcal{A}$ can issue to the random oracle H.

Consider an algorithm SAMP that generates an n-DHE instance $\pi = \big(g, \hat{g}, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \{\hat{g}_i\}_{i\in[n]}\big)$, and an algorithm $\mathcal{B}$ that on input π and randomness $f = (\rho, h_1, \ldots, h_Q)$, where $Q = n \cdot q_H$, does as follows.

$\mathcal{B}$ begins by drawing a random index $i^\dagger \xleftarrow{R} U([n])$. It runs $\mathcal{A}$ on input $\big(g, \hat{g}, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \{\hat{g}_i\}_{i\in[n]}\big)$ and randomness ρ. First, $\mathcal{B}$ initializes a counter $\ell = 0$, which it increments every time it sets a new hash value. When $\mathcal{A}$ makes a H-query $(i, C, S, \mathbf{m}[S])$, $\mathcal{B}$ uses the values in $(h_1, \ldots, h_Q)$ to respond as follows:

- If the input $(i, C, S, \mathbf{m}[S])$ is such that the hash value $H(i, C, S, \mathbf{m}[S])$ was previously defined, $\mathcal{B}$ returns the previously-defined value.
- If $(i, C, S, \mathbf{m}[S])$ is such that $i^\dagger \in S$, then $\mathcal{B}$ does the following: For each index $i' \in S \setminus \{i^\dagger\}$, it increments ℓ and sets $H(i', C, S, \mathbf{m}[S]) \leftarrow h_\ell$. Finally, it increments ℓ and sets $H(i^\dagger, C, S, \mathbf{m}[S]) \leftarrow h_\ell$. Note that $\mathcal{B}$ programs H on $|S|$ inputs at once for such hash queries and that $H(i^\dagger, C, S, \mathbf{m}[S])$ is set after all other inputs for indices $i' \in S \setminus \{i^\dagger\}$. Then, $\mathcal{B}$ returns the corresponding value of $H(i, C, S, \mathbf{m}[S])$ to $\mathcal{A}$.
- Else, $\mathcal{B}$ increments ℓ and returns h_ℓ as the value for $H(i, C, S, \mathbf{m}[S])$.

Since $\mathcal{A}$ makes at most q_H and $\mathcal{B}$ sets at most n hash values at each query, at most $Q = n \cdot q_H$ values are set in the process.

With probability $\epsilon := \Pr[\neg\mathsf{Bind}]$, $\mathcal{A}$ outputs

$$(C, S_0, S_1, \mathbf{m}_0[S_0], \mathbf{m}_1[S_1], \pi_0, \pi_1)$$

such that

$$e(C, \prod_{i\in S_0} \hat{g}_{n+1-i}^{t_i^{(0)}}) = e(\pi_0, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_0} \mathbf{m}_0[i]\cdot t_i^{(0)}} \tag{5}$$

$$e(C, \prod_{i\in S_1} \hat{g}_{n+1-i}^{t_i^{(1)}}) = e(\pi_1, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_1} \mathbf{m}_1[i]\cdot t_i^{(1)}} \tag{6}$$

where $t_i^{(b)} = H(i, C, S_b, \mathbf{m}_b[S_b])$ for each $i \in S_b$ and $b \in \{0,1\}$. Then, $\mathcal{B}$ determines the smallest $i^\star \in S_0 \cap S_1$ such that $\mathbf{m}_0[i^\star] \neq \mathbf{m}_1[i^\star]$. If $i^\star \neq i^\dagger$, it aborts and outputs $(\emptyset, \emptyset)$. Otherwise, if $i^\star = i^\dagger$ (which is the case with probability $1/n$ since $i^\dagger$ is drawn uniformly and independently of $\mathcal{A}$'s view), let $\ell_0 \in [Q]$ be the index of the random oracle query $H(i^\star, C, S_0, \mathbf{m}_0[S_0])$ and let $\ell_1 \in [Q]$ be the index of the random oracle query $H(i^\star, C, S_1, \mathbf{m}_1[S_1])$. Let $h_{\ell_0}, h_{\ell_1} \in \mathbb{Z}_p$ be the corresponding responses. Note that, due to the way $\mathcal{B}$ sets the responses to random oracle queries, any value $H(i, C, S_b, \mathbf{m}_b[S_b])$ for an index $i \in S_b \setminus \{i^\star\}$ is set before $H(i^\star, C, S_b, \mathbf{m}_b[S_b]) = h_{\ell_b}$, for $b \in \{0,1\}$. Finally, $\mathcal{B}$ outputs $(L = \{\ell_0, \ell_1\}, \{out_{\ell_0}, out_{\ell_1}\})$, where

$$out_{\ell_0} = out_{\ell_1} = (C, S_0, S_1, \mathbf{m}_0[S_0], \mathbf{m}_1[S_1], \pi_0, \pi_1, h_{\ell_0}, h_{\ell_1}) \tag{7}$$

Note that $\mathcal{B}$ outputs a-non empty subset $L = \{\ell_0, \ell_1\}$ successfully with probability at least ϵ/n.

Now, we describe a reduction $\mathcal{C}$ that solves an n-DHE instance $\big(g, \hat{g}, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \{\hat{g}_i\}_{i\in[n]}\big)$ using $\mathcal{A}$. Algorithm $\mathcal{C}$ runs the game $G_{\mathcal{B},\pi}$ defined in Figure 1, where π is the given n-DHE instance, and $\mathcal{B}$ is the algorithm that uses the adversary $\mathcal{A}$ as described above. If the game outputs 0, then $\mathcal{C}$ aborts. By Lemma 1, with probability at least $(\epsilon/n)/8$, $G_{\mathcal{B},\pi}$ outputs $(L = \{\ell_0, \ell_1\}, \{out_{\ell_0}, out_{\ell_1}\}, \{out'_{\ell_0}, out'_{\ell_1}\})$ after having forked twice. Then, $\mathcal{C}$ parses out_{ℓ_0} and out_{ℓ_1} as in (7) and obtains C and $(S_b, \mathbf{m}_b[S_b], \pi_b)$ for $b \in \{0,1\}$ satisfying equations (5)-(6). It also parses out'_{ℓ_0} and out'_{ℓ_1} as described hereunder and performs the following computations:

- For the first forking, it parses

  $$out'_{\ell_0} = (C, S_0, S'_1, \mathbf{m}_0[S_0], \mathbf{m}'_1[S'_1], \pi'_0, \pi'_1, h'_{\ell_0}, h'_{\ell_1}),$$

  where Verify outputs 1 on both $(C, S_0, \mathbf{m}_0[S_0], \pi'_0)$, and $(C, S'_1, \mathbf{m}'_1[S'_1], \pi'_1)$. Note that $(S'_1, \mathbf{m}'_1[S'_1])$ may differ from their counterparts $(S_1, \mathbf{m}_1[S_1])$ of the first execution but it does not matter. What matters is that the second run involves the same $(S_0, \mathbf{m}_0[S_0])$ as in the first execution and the hash query

$H(i^\star, C, S_0, \mathbf{m}_0[S_0])$ is also the ℓ_0-th hash query in the second execution, where it obtains a different response h'_{ℓ_0}. This holds since the forking occurs at the ℓ_0-th oracle query and the executions are identical up to that point. Hence, the ℓ_0-th query is indeed issued on the input $(i^\star, C, S_0, \mathbf{m}_0[S_0])$ during the forking. Now, since the index $i^\star \in S_0 \cap S_1$ determined in the first run belongs to S_0, we know that

$$t^{(0)}_{i^\star} = H(i^\star, C, S_0, \mathbf{m}_0[S_0]) = h'_{\ell_0} \neq h_{\ell_0},$$

but other hash values of the form $H(i, C, S_0, \mathbf{m}_0[S_0])$ for $i \neq i^\star$ are the same as in the initial execution because $\mathcal{B}$ assigned the values of $H(i, C, S_0, \mathbf{m}_0[S_0])$ for $i \neq i^\star$ before the forking point. Let us now consider the proofs π_0, π'_0 obtained in the first run of $\mathcal{B}$ and the first forking, respectively. By dividing out the equations (5) of both runs, we have

$$e(C, \hat{g}_{n+1-i^\star}^{\Delta t^{(0)}_{i^\star}}) = e(\pi_0/\pi'_0, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_0[i^\star] \cdot \Delta t^{(0)}_{i^\star}}, \tag{8}$$

where $\Delta t^{(0)}_{i^\star} \triangleq h_{\ell_0} - h'_{\ell_0} \neq 0$.

- For the second forking, the reduction $\mathcal{C}$ parses out'_{ℓ_1} as

$$out'_{\ell_1} = (C, S''_0, S_1, \mathbf{m}''_0[S''_0], \mathbf{m}_1[S_1], \pi''_0, \pi''_1, h''_{\ell_0}, h''_{\ell_1}).$$

Here, $(S''_0, \mathbf{m}''_0[S''_0])$ may differ from the pair $(S_0, \mathbf{m}_0[S_0])$ extracted from out_{ℓ_1} at the very first execution, but it does not matter. The forking point being the ℓ_1-th hash query, we know that the first and third executions are identical up to that point. Consequently, the ℓ_1-th query is issued on the input $(i^\star, C, S_1, \mathbf{m}_1[S_1])$ in this forking, where it obtains a different response h''_{ℓ_1} than in the very first run. With non-negligible probability, Lemma 1 ensures that $\mathcal{B}$'s output in this forking involves the same $(S_1, \mathbf{m}_1[S_1])$ as in the first run and the hash query $H(i^\star, C, S_1, \mathbf{m}_1[S_1])$ is also the ℓ_1-th hash query. Since $i^\star \in S_1$, we can repeat the same arguments as in the first fork and, by dividing out the verification equations (6) of the first and third runs, $\mathcal{B}$ obtains

$$e(C, \hat{g}_{n+1-i^\star}^{\Delta t^{(1)}_{i^\star}}) = e(\pi_1/\pi''_1, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_1[i^\star] \cdot \Delta t^{(1)}_{i^\star}}, \tag{9}$$

where $\Delta t^{(1)}_{i^\star} \triangleq h_{\ell_1} - h''_{\ell_1} \neq 0$.

Then, raising both members of (9) to the power $\omega \triangleq \Delta t^{(0)}_{i^\star}/\Delta t^{(1)}_{i^\star}$ yields

$$e(C, \hat{g}_{n+1-i^\star}^{\Delta t^{(0)}_{i^\star}}) = e((\pi_1/\pi''_1)^\omega, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_1[i^\star] \cdot \Delta t^{(0)}_{i^\star}}. \tag{10}$$

If we now use the hypothesis that $\mathbf{m}_0[i^\star] \neq \mathbf{m}_1[i^\star]$, the combination of (10) and (8) implies

$$e(\pi_0/\pi'_0, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_0[i^\star] \cdot \Delta t^{(0)}_{i^\star}} = e((\pi_1/\pi''_1)^\omega, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_1[i^\star] \cdot \Delta t^{(0)}_{i^\star}}.$$

Now, since $e(g_1, \hat{g}_n) = e(g_{n+1}, \hat{g})$, we have:

$$e(g_{n+1}, \hat{g})^{\Delta t_{i^\star}^{(0)} \cdot (\mathbf{m}_0[i^\star] - \mathbf{m}_1[i^\star])} = e((\pi_1/\pi_1'')^\omega, \hat{g})/e(\pi_0/\pi_0', \hat{g}),$$

which then allows $\mathcal{B}$ to compute the sought-after n-DHE solution, by looking only at the $\mathbb{G}$-components, as

$$g_{n+1} \triangleq \left(\frac{(\pi_1/\pi_1'')^\omega}{\pi_0/\pi_0'}\right)^{1/\left(\Delta t_{i^\star}^{(0)} \cdot (\mathbf{m}_0[i^\star] - \mathbf{m}_1[i^\star])\right)}. \tag{11}$$

By Lemma 1, with probability at least $(\epsilon/n)/8$, the reduction $\mathcal{C}$ succeeds in solving the n-DHE problem. □

4 The Case of Cross-Commitment Aggregation

Let $H' : \{0,1\}^* \to \mathbb{Z}_p$ be a hash function modeled as a random oracle. Algorithms AggregateAcross and VerifyAcross are as follows:

AggregateAcross$_{\mathsf{pp}}(\{C^{(j)}, S^{(j)}, \mathbf{m}^{(j)}[S^{(j)}], \pi_j\}_{j\in[d]})$: Given a collection of $(\{C^{(j)}, S^{(j)}, \mathbf{m}^{(j)}[S^{(j)}], \pi_j\}_{j\in[d]})$, where each π_j is the same-commitment-aggregated proof of $C^{(j)}$ with respect to the sub-vector of message $\mathbf{m}^{(j)}$ limited to indices in $S^{(j)}$, compute and output

$$\pi = \prod_{j=1}^{d} (\pi_j)^{t_j'},$$

where $t_j' = H'(j, \{C^{(j)}, S^{(j)}, \mathbf{m}^{(j)}[S^{(j)}]\}_{j\in[d]})$.

VerifyAcross$(\pi, \{C^{(j)}, S^{(j)}, \mathbf{m}^{(j)}[S^{(j)}], \pi_j\}_{j\in[d]})$: Given $\pi \in \mathbb{G}$, return 1 if $C^{(j)} \in \mathbb{G}$ for all $j \in [d]$, and

$$\prod_{j=1}^{d} e\left(C^{(j)}, \prod_{i\in S^{(j)}} \hat{g}_{n+1-i}^{t_{j,i}}\right)^{t_j'} = e(\pi, \hat{g}) \cdot e\left(g_1, \hat{g}_n\right)^{\sum_{j\in[d], i\in S^{(j)}} \mathbf{m}^{(j)}[i] \cdot t_{j,i} \cdot t_j'}$$

where

$$t_{j,i} = H(i, C^{(j)}, S^{(j)}, \mathbf{m}^{(j)}[S^{(j)}]), \qquad t_j' = H'(j, \{C^{(j)}, S^{(j)}, \mathbf{m}^{(j)}[S^{(j)}]\}_{j\in[d]}).$$

We now prove the cross-commitment binding property under the n-DHE assumption in the ROM, without restricting ourselves to algebraic adversaries.

Here, we rely on the Local Forking Lemma instead of the Generalized Forking Lemma. The reason is that the proof of Theorem 2 proceeds with a reduction from the same-commitment case. In the process, it has to fork on the hash function H'. For this purpose, if we were to use the Generalized Forking Lemma as in the proof of Theorem 1, we would have no way to guess which hash query should be defined after other hash queries involving related inputs. Therefore we need the Local Forking Lemma to force *all but one* of the cross-commitment aggregation coefficients $\{t_j'\}_{j\in[d]}$ to be identical in two adversarial runs.

Theorem 2. *The above cross-commitment scheme is binding in the random oracle model assuming the hardness of the n-DHE problem.*

Proof. For the sake of contradiction, let us assume that an adversary $\mathcal{A}$ has non-negligible probability ϵ of contradicting the binding property of the cross-commitment aggregation in PointProofs. Namely, with probability ϵ, $\mathcal{A}$ can generate two tuples

$$\left(\pi_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[d_0]}\right), \qquad \left(\pi_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[d_1]}\right),$$

such that $\mathsf{VerifyAcross}$ accepts $(\pi_b, \{C_b^{(j)}, S_b^{(j)}, \mathbf{m}_b^{(j)}[S_b^{(j)}]\}_{j\in[d_b]})$ for each value of $b \in \{0,1\}$, and there exist indices $j_0 \in [d_0]$ and $j_1 \in [d_1]$ for which $C_0^{(j_0)} = C_1^{(j_1)}$ and $\mathbf{m}_0^{(j_0)}[i] \neq \mathbf{m}_1^{(j_1)}[i]$, for some $i \in S_0^{(j_0)} \cap S_1^{(j_1)}$. In the random oracle model, we give a reduction $\mathcal{B}$ that uses $\mathcal{A}$ to break the binding property of the *same-commitment* aggregation of PointProofs.

$\mathcal{B}$ receives public parameters $\mathsf{pp} = \left(g, \hat{g}, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \{\hat{g}_i\}_{i\in[n]}\right)$ from its own challenger in the same-commitment aggregation game and runs $\mathcal{A}$ on pp. Note that $\mathcal{B}$, which is attacking the binding property of the same-commitment aggregation of PointProofs, has oracle access to H, but $\mathcal{A}$ has also oracle access to H'. Algorithm $\mathcal{B}$ responds to $\mathcal{A}$'s oracle queries to H by redirecting the query to H (so, it does not simulate H itself for $\mathcal{A}$). It responds to $\mathcal{A}$'s queries to the second random oracle H' in the following way. In a first execution, it answers H'-queries with values $h_1, h_2, \ldots, h_Q \in \mathbb{Z}_p$, where Q denotes the total number of queries made by $\mathcal{A}$ to H'. We assume w.l.o.g. that all these queries are distinct. With probability ϵ, $\mathcal{A}$ outputs

$$\left(\pi_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[d_0]}\right), \qquad \left(\pi_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[d_1]}\right),$$

such that the aggregated verification algorithm returns 1 running on both tuples. Namely,

$$\prod_{j=1}^{d_0} e\left(C_0^{(j)}, \prod_{i\in S_0^{(j)}} \hat{g}_{n+1-i}^{t_{j,i}^{(0)}}\right)^{t_j'^{(0)}} = e(\pi_0, \hat{g}) \cdot e\left(g_1, \hat{g}_n\right)^{\sum_{j\in[d_0], i\in S_0^{(j)}} \mathbf{m}_0^{(j)}[i]\cdot t_{j,i}^{(0)}\cdot t_j'^{(0)}} \tag{12}$$

$$\prod_{j=1}^{d_1} e\left(C_1^{(j)}, \prod_{i\in S_1^{(j)}} \hat{g}_{n+1-i}^{t_{j,i}^{(1)}}\right)^{t_j'^{(1)}} = e(\pi_1, \hat{g}) \cdot e\left(g_1, \hat{g}_n\right)^{\sum_{j\in[d_1], i\in S_1^{(j)}} \mathbf{m}_1^{(j)}[i]\cdot t_{j,i}^{(1)}\cdot t_j'^{(1)}} \tag{13}$$

where $t_{j,i}^{(b)} = H(i, C_b^{(j)}, S_b^{(j)}, \mathbf{m}_b^{(j)}[S_b^{(j)}])$, for each $j \in [d_b]$, $i \in S_b^{(j)}$, and $b \in \{0,1\}$, and $t_j'^{(b)} = H'(j, \{C_b^{(j)}, S_b^{(j)}, \mathbf{m}_b^{(j)}[S_b^{(j)}]\}_{j\in[d_b]})$, for each $j \in [d_b]$ and $b \in \{0,1\}$.

Then, $\mathcal{B}$ determines the two indices $j_0 \in [d_0]$, $j_1 \in [d_1]$, for which the binding property is contradicted, i.e., $C_0^{(j_0)} = C_1^{(j_1)}$, and $\mathbf{m}_0^{(j_0)}[i] \neq \mathbf{m}_1^{(j_1)}[i]$, for some $i \in S_0^{(j_0)} \cap S_1^{(j_1)}$. Let $\ell_b \in [Q]$ be the index of the query

$$H'(j_b, \{C_b^{(j)}, S_b^{(j)}, \mathbf{m}_b^{(j)}[S_b^{(j)}]\}_{j\in[d_b]}),$$

for $b \in \{0,1\}$. Let $h_{\ell_0}, h_{\ell_1} \in \mathbb{Z}_p$ be the corresponding responses.

The reduction $\mathcal{B}$ then locally forks the adversary twice. It first runs $\mathcal{A}$ a second time with the same random tape and answers all random oracle queries using the outputs $h_1, \ldots, h_{\ell_0-1}, h'_{\ell_0}, h_{\ell_0+1}, \ldots, h_Q \in \mathbb{Z}_p$, where $h'_{\ell_0} \xleftarrow{R} \mathbb{Z}_p$ is chosen afresh and all other outputs h_ℓ for $\ell \neq \ell_0$ are identical to those of the first execution. The Local Forking Lemma (Lemma 2) ensures that with probability at least equal to $1/Q \cdot \epsilon^2$, $\mathcal{A}$'s second run outputs

$$\left(\pi'_0,\ \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[d_0]},\ \pi'_1,\ \{C_1'^{(j)}, S_1'^{(j)}, \mathbf{m}_1'^{(j)}[S_1'^{(j)}]\}_{j\in[d'_1]}\right).$$

Note that the second collection $\{C_1'^{(j)}, S_1'^{(j)}, \mathbf{m}_1'^{(j)}[S_1'^{(j)}]\}_{j\in[d'_1]}$ might be different from its counterpart $\{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[d_1]}$ of the first run, but what matters is that this run involves the same collection $\{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[d_0]}$ as the first execution of $\mathcal{A}$ and the hash query $H'(j_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[d_0]})$ is also the ℓ_0-th query issued by $\mathcal{A}$ where it receives a different response h'_{ℓ_0}. Since $j_0 \in d_0$, we know that

$$t_{j_0}^{\prime(0)} = H'(j_0, \{C_0^{(j)}, S_0^{(j)}, \mathbf{m}_0^{(j)}[S_0^{(j)}]\}_{j\in[d_0]}) = h_{\ell_0} \neq h'_{\ell_0},$$

but other hash values $t_j^{\prime(0)}$ for $j \in [d_0] \setminus \{j_0\}$ are the same as in the initial execution because of the *local* forking. If we now consider the two proofs π_0, π'_0 obtained in the two runs, by dividing out the equations (12) of both runs, we have

$$e\left(C_0^{(j_0)}, \prod_{i\in S_0^{(j_0)}} \hat{g}_{n+1-i^\star}^{t_{j_0,i}^{(0)}}\right)^{\Delta t_{j_0}^{\prime(0)}} = e(\pi_0/\pi'_0, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_0^{(j_0)}} \mathbf{m}_0^{(j_0)}[i]\cdot t_{j_0,i}^{(0)}\cdot \Delta t_{j_0}^{\prime(0)}} \tag{14}$$

where $\Delta t_{j_0}^{\prime(0)} = h_{\ell_0} - h'_{\ell_0} \neq 0$. Raising both sides of equation (14) to the power $\omega_0 \triangleq 1/(\Delta t_{j_0}^{\prime(0)})$ yields

$$e\left(C_0^{(j_0)}, \prod_{i\in S_0^{(j_0)}} \hat{g}_{n+1-i^\star}^{t_{j_0,i}^{(0)}}\right) = e((\pi_0/\pi'_0)^{\omega_0}, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_0^{(j_0)}} \mathbf{m}_0^{(j_0)}[i]\cdot t_{j_0,i}^{(0)}} \tag{15}$$

Then, $\mathcal{B}$ locally forks $\mathcal{A}$ a second time on the hash query $H'(j_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[d_1]})$, which was the ℓ_1-th H'-query in the first execution. Namely, it runs $\mathcal{A}$ a second time with the same random tape as in the first run and now answers $\mathcal{A}$'s random oracle queries to H' using the outputs $h_1, \ldots, h_{\ell_1-1}, h''_{\ell_1}, h_{\ell_1+1}, \ldots, h_Q \in \mathbb{Z}_p$, where $h''_{\ell_1} \xleftarrow{R} \mathbb{Z}_p$ is freshly sampled and all other outputs h_ℓ for $\ell \neq \ell_1$ are the same as in the first execution. By the Local Forking Lemma, with probability at least equal to $1/Q \cdot \epsilon^2$, $\mathcal{A}$'s third run outputs

$$\left(\pi''_0,\ \{C_0''^{(j)}, S_0''^{(j)}, \mathbf{m}_0''^{(j)}[S_0''^{(j)}]\}_{j\in[d''_0]},\ \pi''_1,\ \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[d_1]}\right).$$

Again, the collection $\{C_0''^{(j)}, S_0''^{(j)}, \mathbf{m}_0''^{(j)}[S_0''^{(j)}]\}_{j\in[d''_0]}$ might differ from its counterpart $\{C_0{}^{(j)}, S_0{}^{(j)}, \mathbf{m}_0{}^{(j)}[S_0{}^{(j)}]\}_{j\in[d_0]}$ of the first execution of $\mathcal{A}$, yet, this run involves the same collection $\{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[d_1]}$ as in the first run and the hash query $H'(j_1, \{C_1^{(j)}, S_1^{(j)}, \mathbf{m}_1^{(j)}[S_1^{(j)}]\}_{j\in[d_1]})$ is also the ℓ_1-th query to H' where $\mathcal{A}$ receives a different response h''_{ℓ_1}. Since $j_1 \in d_1$, we can repeat the same arguments as in the first fork and, by dividing out the verification equations (13), $\mathcal{B}$ obtains

$$e\left(C_1^{(j_1)}, \prod_{i\in S_1^{(j_1)}} \hat{g}_{n+1-i^\star}^{t^{(1)}_{j_1,i}}\right)^{\Delta t'^{(1)}_{j_1}} = e(\pi_1/\pi''_1, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_1^{(j_1)}} \mathbf{m}_1^{(j_1)}[i]\cdot t^{(0)}_{j_1,i}\cdot \Delta t'^{(1)}_{j_1}} \tag{16}$$

where $\Delta t'^{(1)}_{j_1} = h_{\ell_1} - h''_{\ell_1} \neq 0$. Again, by raising both members of equation (14) to the power $\omega_1 \triangleq 1/(\Delta t'^{(1)}_{j_1})$ we have

$$e\left(C_1^{(j_1)}, \prod_{i\in S_1^{(j_1)}} \hat{g}_{n+1-i^\star}^{t^{(1)}_{j_1,i}}\right) = e((\pi_1/\pi''_1)^{\omega_1}, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_1^{(j_1)}} \mathbf{m}_1^{(j_1)}[i]\cdot t^{(0)}_{j_1,i}} \tag{17}$$

Finally, $\mathcal{B}$ outputs the tuple

$$\left(C, S_0^{(j_0)}, S_1^{(j_1)}, \mathbf{m}_0^{(j_0)}[S_0^{(j_0)}], \mathbf{m}_1^{(j_1)}[S_1^{(j_1)}], (\pi_0/\pi'_0)^{\omega_0}, (\pi_1/\pi''_1)^{\omega_1}\right),$$

where $C = C_0^{(j_0)} = C_1^{(j_1)}$. Regarding equations (15) and (17), Verify accepts both

$$\left((\pi_0/\pi'_0)^{\omega_0}, C, S_0^{(j_0)}, \mathbf{m}_0^{(j_0)}[S_0^{(j_0)}]\right), \qquad \left((\pi_1/\pi''_1)^{\omega_1}, C, S_1^{(j_1)}, \mathbf{m}_1^{(j_1)}[S_1^{(j_1)}]\right),$$

and there exists an index $i \in S_0^{(j_0)} \cap S_1^{(j_1)}$ for which $\mathbf{m}_0^{(j_0)}[i] \neq \mathbf{m}_1^{(j_1)}[i]$. With non-negligible probability $(1/Q \cdot \epsilon^2)^2$, $\mathcal{B}$ thus breaks the binding property of the

same-commitment aggregation construction from Section 3, which contradicts the statement of Theorem 1. □

Acknowledgements. The second and third authors were supported by the French ANR RAGE project (ANR-20-CE48-0011) and the PEPR Cyber France 2030 programme (ANR-22-PECY-0003).

References

1. A. Bagherzandi, J.-H. Cheon, S. Jarecki. Multisignatures Secure under the Discrete Logarithm Assumption and a Generalized Forking Lemma. In *ACM-CCS*, 2008.
2. M. Bellare, W. Dai, L. Li. The Local Forking Lemma and its Application to Deterministic Encryption. In *Asiacrypt 2019*.
3. D. Boneh, B. Bünz, B. Fisch. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In *Crypto*, 2019.
4. D. Boneh, X. Boyen, E.-J. Goh. Hierarchical Identity-Based encryption with Constant Size Ciphertext. In *Eurocrypt'05*, *LNCS* 3494, pp. 440–456, 2005.
5. D. Boneh, J. Drake, B. Fisch, A. Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive Report 2020/81.
6. D. Boneh, J. Drake, B. Fisch, A. Gabizon. Halo Infinite: Proof-Carrying Data from Additive Polynomial Commitments. In *CRYPTO'21*, *LNCS* 12825, pp. 649–680, 2021.
7. D. Boneh, M. Drijvers, G. Neven. Compact Multi-Signatures for Smaller Blockchains. In *Asiacrypt*, 2018.
8. D. Boneh, C. Gentry, B. Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *Crypto'05*, *LNCS* 3621, pp. 258–275, 2005.
9. J. Camenisch, M. Dubovitskaya, K. Haralambiev, M. Kohlweiss. Composable & Modular Anonymous Credentials: Definitions and Practical Constructions. In *Asiacrypt*, 2015.
10. M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, L. Nizzardo. Incrementally Aggregatable Vector Commitments and Applications to Verifiable Decentralized Storage. In *Asiacrypt*, 2020.
11. D. Catalano, D. Fiore, M. Messina. Zero-Knowledge Sets with Short Proofs. In *Eurocrypt*, 2008.
12. D. Catalano, D. Fiore. Vector commitments and their applications. In *PKC*, 2013.
13. A. Chepurnoy, C. Papamanthou, S. Srinivasan, Y. Zhang. Edrax: A Cryptocurrency with Stateless Transaction Validation. Cryptology ePrint Archive Report 2018/968.
14. A. Fiat, A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto*, 1086.
15. G. Fuchsbauer, E. Kiltz, J. Loss. The Algebraic Group Model and its Applications. In *Crypto*, 2018.
16. S. Gorbunov, L. Reyzin, H. Wee, Z. Zhang. PointProofs: Aggregating Proofs for Multiple Vector Commitments In *ACM-CCS* 2020.
17. A. Kate, G. Zaverucha, I. Goldberg. Constant-Size Commitments to Polynomials and Applications. In *Asiacrypt*, 2010.
18. J. Krupp, D. Schröder, M. Simkin, D. Fiore, G. Ateniese, S. Nürnberger. Nearly Optimal Verifiable Data Streaming. In *PKC*, 2016.

19. R.-F. Lai, G. Malavolta. Subvector Commitments with Application to Succinct Arguments. In *Crypto* 2019.
20. D. Leung, Y. Gilad, S. Gorbunov, L. Reyzin, N. Zeldovich. Aardvark: A Concurrent Authenticated Dictionary with Short Proofs. Cryptology ePrint Archive Report 2020/975, 2020.
21. B. Libert, S. Ramanna, M. Yung. Functional Commitment Schemes: From Polynomial Commitments to Pairing-Based Accumulators from Simple Assumptions. . In *ICALP* 2016.
22. B. Libert, M. Yung. Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In *TCC* 2010.
23. R. Merkle. A Certified Digital Signature. In *Crypto'89*, 1989.
24. S. Micali, M.-O. Rabin, J. Kilian. Zero-Knowledge Sets. In *FOCS*, 2003.
25. S. Micali, L. Reyzin, G. Vlachos, R. Wahby, N. Zeldovich. Compact Certificates of Collective Knowledge. In *IEEE S & P*, 2021.
26. M. Naor. On Cryptographic Assumptions and Challenges. In *Crypto*, 2003.
27. T. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Crypto*, 1991.
28. C. Peikert, Z. Pepin, C. Sharp. Vector and Functional Commitments from Lattices. In *TCC*, 2021.
29. D. Pointcheval, J. Stern. Security Arguments for Signature Schemes. In *J. of Cryptology*, 2000.
30. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In *Eurocrypt*, 1997.
31. S. Srinivasan, A. Chepurnoy, C. Papamanthou, A. Tomescu, Y. Zhang. Hyperproofs: Aggregating and Maintaining Proofs in Vector Commitments. In *USENIX Security*, 2022.
32. A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, D. Khovratovich. Aggregatable Subvector Commitments for Stateless Cryptocurrencies. In *SCN*, 2020.
33. A. Tomescu. How to compute all Pointproofs. Cryptology ePrint Archive Report 2020/1516.
34. A. Tomescu, Y. Xia, Z. Newman. Authenticated Dictionaries with Cross-Incremental Proof (Dis)aggregation. Cryptology ePrint Archive Report 2020/1239.

Supplementary Material

A Proof via Local Forking Lemma

In this section, in order to compare with the reduction obtained from the Generalized Forking Lemma, we prove the binding property of same-commitment aggregation of PointProofs [16] in the random oracle model via Local Forking Lemma [2].

A.1 The Case of Same-Commitment Aggregation

Theorem 3. *The vector commitment scheme described in Description 3 is binding in the random oracle model if the n-DHE assumption holds.*

Proof. Suppose there exists an adversary $\mathcal{A}$, that with a non-negligible probability ϵ generates a commitment C as well as two sets $S_0, S_1 \subset [n]$, such that $S_0 \cap S_1 \neq \emptyset$, together with convincing proofs π_0, π_1 for sub-vectors $\mathbf{m}_0[S_0] \in \mathbb{Z}_p^{|S_0|}$, $\mathbf{m}_1[S_1] \in \mathbb{Z}_p^{|S_1|}$ such that $\mathbf{m}_0[i] \neq \mathbf{m}_1[i]$ for some $i \in S_0 \cap S_1$. We build an algorithm $\mathcal{B}$ in the random oracle model that uses $\mathcal{A}$ to solve the n-DHE problem.

$\mathcal{B}$ first runs $\mathcal{A}$ on input of an n-DHE instance $\big(g, \hat{g}, \{g_i\}_{i\in[2n]\setminus\{n+1\}}, \{\hat{g}_i\}_{i\in[n]}\big)$ set as the public parameters pp of the vector commitment scheme. In a first execution, it answers random oracle queries with values $h_1, \ldots, h_Q \in \mathbb{Z}_p$, where Q denotes the total number of H-queries made by $\mathcal{A}$. We assume w.l.o.g. that all random oracle queries are distinct. With probability ϵ, $\mathcal{A}$ outputs a tuple

$$(C, S_0, S_1, \mathbf{m}_0[S_0], \mathbf{m}_1[S_1], \pi_0, \pi_1)$$

such that

$$e(C, \prod_{i\in S_0} \hat{g}_{n+1-i}^{t_i^{(0)}}) = e(\pi_0, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_0} \mathbf{m}_0[i]\cdot t_i^{(0)}} \tag{18}$$

$$e(C, \prod_{i\in S_1} \hat{g}_{n+1-i}^{t_i^{(1)}}) = e(\pi_1, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\sum_{i\in S_1} \mathbf{m}_1[i]\cdot t_i^{(1)}} \tag{19}$$

where $t_i^{(b)} = H(i, C, S_b, \mathbf{m}_b[S_b])$ for each $i \in S_b$ and $b \in \{0, 1\}$. Then, algorithm $\mathcal{B}$ determines the smallest $i^\star \in S_0 \cap S_1$ such that $\mathbf{m}_0[i^\star] \neq \mathbf{m}_1[i^\star]$. Let $\ell_0 \in [Q]$ be the index of the random oracle query $H(i^\star, C, S_0, \mathbf{m}_0[S_0])$ and let $\ell_1 \in [Q]$ be the index of the random oracle query $H(i^\star, C, S_1, \mathbf{m}_1[S_1])$. Let $h_{\ell_0}, h_{\ell_1} \in \mathbb{Z}_p$ be the corresponding responses.

The reduction then locally forks the adversary twice. It first runs $\mathcal{A}$ a second time with the same random tape and answers all random oracle queries using the outputs $h_1, \ldots, h_{\ell_0-1}, h'_{\ell_0}, h_{\ell_0+1}, \ldots, h_Q \in \mathbb{Z}_p$, where $h'_{\ell_0} \xleftarrow{R} \mathbb{Z}_p$ is chosen afresh and all other outputs h_ℓ for $\ell \neq \ell_0$ are identical to those of the first execution. The

Local Forking Lemma (Lemma 2) ensures that with probability at least equal to $1/Q \cdot \epsilon^2$, $\mathcal{A}$'s second run outputs $(C, S_0, S_1', \mathbf{m}_0[S_0], \mathbf{m}_1'[S_1'], \pi_0', \pi_1')$. Note that $(S_1', \mathbf{m}_1'[S_1'])$ may differ from their counterparts $(S_1, \mathbf{m}_1[S_1])$ of the first execution but this run involves the same $(S_0, \mathbf{m}_0[S_0])$ as in the first execution and the hash query $H(i^\star, C, S_0, \mathbf{m}_0[S_0])$ is also the ℓ_0-th hash query in this execution, where $\mathcal{A}$ receives a different response h_{ℓ_0}'. Since $i^\star \in S_0$, we know that

$$t_{i^\star}^{(0)} = H(i^\star, C, S_0, \mathbf{m}_0[S_0]) = h_{\ell_0} \neq h_{\ell_0}'$$

with overwhelming probability $1-1/p$, but other hash values $t_i^{(0)}$ for $i \in S_0 \setminus \{i^\star\}$ are the same as in the initial execution because of the *local* forking. If we now consider the two proofs π_0, π_0' obtained in the two runs, by dividing out the equations (18) of both runs, we have

$$e(C, \hat{g}_{n+1-i^\star}^{\Delta t_{i^\star}^{(0)}}) = e(\pi_0/\pi_0', \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_0[i^\star] \cdot \Delta t_{i^\star}^{(0)}}, \tag{20}$$

where $\Delta t_{i^\star}^{(0)} \triangleq h_{\ell_0} - h_{\ell_0}' \neq 0$ except with probability $1/p$.

Then, $\mathcal{B}$ locally forks $\mathcal{A}$ a second time on the hash query $H(i^\star, C, S_1, \mathbf{m}_1[S_1])$, which was the ℓ_1-th hash query in the first execution. Namely, it runs $\mathcal{A}$ a second time with the same random tape as in the first run and now answers all random oracle queries using the outputs $h_1, \ldots, h_{\ell_1-1}, h_{\ell_1}'', h_{\ell_1+1}, \ldots, h_Q \in \mathbb{Z}_p$, where $h_{\ell_1}'' \xleftarrow{R} \mathbb{Z}_p$ is freshly sampled and all other outputs h_ℓ for $\ell \neq \ell_1$ are the same as in the first execution. Here again, regarding the Local Forking Lemma, with probability at least equal to $1/Q \cdot \epsilon^2$, $\mathcal{A}$'s third run outputs $(C, S_0'', S_1, \mathbf{m}_0''[S_0''], \mathbf{m}_1[S_1], \pi_0'', \pi_1'')$. Note that $(S_0'', \mathbf{m}_0''[S_0''])$ may differ from the pair $(S_0, \mathbf{m}_0[S_0])$ of the first execution but this run involves the same $(S_1, \mathbf{m}_1[S_1])$ as in the first execution of $\mathcal{A}$ and the hash query $H(i^\star, C, S_1, \mathbf{m}_1[S_1])$ is also the ℓ_1-th query in the this run, where $\mathcal{A}$ receives a different response h_{ℓ_1}''. Since $i^\star \in S_1$, we can repeat the same arguments as in the first fork and, by dividing out the verification equations (19) of the first and third runs, $\mathcal{B}$ obtains

$$e(C, \hat{g}_{n+1-i^\star}^{\Delta t_{i^\star}^{(1)}}) = e(\pi_1/\pi_1'', \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_1[i^\star] \cdot \Delta t_{i^\star}^{(1)}}, \tag{21}$$

where $\Delta t_{i^\star}^{(1)} \triangleq h_{\ell_1} - h_{\ell_1}'' \neq 0$ except with probability $1/p$. Then, raising both sides of (21) to the power $\omega \triangleq \Delta t_{i^\star}^{(0)}/\Delta t_{i^\star}^{(1)}$ yields

$$e(C, \hat{g}_{n+1-i^\star}^{\Delta t_{i^\star}^{(0)}}) = e((\pi_1/\pi_1'')^\omega, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_1[i^\star] \cdot \Delta t_{i^\star}^{(0)}}. \tag{22}$$

If we now use the hypothesis that $m_{i^\star} \neq m_{i^\star}'$, the combination of (22) and (8) implies

$$e(\pi_0/\pi_0', \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_0[i^\star] \cdot \Delta t_{i^\star}^{(0)}} = e((\pi_1/\pi_1'')^\omega, \hat{g}) \cdot e(g_1, \hat{g}_n)^{\mathbf{m}_1[i^\star] \cdot \Delta t_{i^\star}^{(0)}},$$

which allows $\mathcal{B}$ to compute and output

$$g_{n+1} \triangleq \left(\frac{(\pi_1/\pi_1'')^\omega}{\pi_0/\pi_0'}\right)^{1/\left(\Delta t_{i^\star}^{(0)} \cdot (\mathbf{m}_0[i^\star] - \mathbf{m}_1[i^\star])\right)}. \tag{23}$$

Thus, with probability at least equal to $\left((1/Q)\cdot\epsilon^2\right)^2$, the reduction $\mathcal{B}$ succeeds in solving the n-DHE problem. □

A.2 Local vs Generalized Forking Lemma

We provided two proofs for the binding property of same-commitment aggregation of the vector commitment scheme proposed in [16] as PointProofs. In one technique, we used the Generalized Forking Lemma (Lemma 1) to prove the binding property (Theorem 1), and in the other technique, we used the Local Forking Lemma (Lemma 2) to prove the same (Theorem 3). Here, we compare the two techniques in terms of the advantage/run-time ratio of the reduction in each case.

Let ϵ, t be respectively the winning probability and run-time of $\mathcal{A}$ which is the adversary of PointProofs' binding property. Furthermore, let q be the number of oracle queries that $\mathcal{A}$ issues to H. We denote by m the number of forkings (i.e. the number of times an n-DHE adversary re-runs $\mathcal{A}$) which is equal to 2 in both of our proofs. The advantage/run-time ratio in each case is as follows:

- Using Local Forking Lemma

$$\frac{\mathsf{Adv}}{t} = \frac{(1/q\cdot\epsilon^2)^m}{t} = \frac{\epsilon^4}{t\cdot q^2}$$

- Using Generalized Forking Lemma

$$\frac{\mathsf{Adv}}{t} = \frac{\epsilon/8}{t\cdot 8\cdot m^2\cdot q\cdot 1/\epsilon\cdot\ln(8m/\epsilon)} = \frac{\epsilon^2}{t\cdot 256\cdot q\cdot\ln(16/\epsilon)},$$

  Since we can consider $\epsilon \geq 1/2$, we have $\ln(16/\epsilon) = \ln(16) - \ln(\epsilon) \leq \ln(16) - \ln(1/2) \approx 3.4$. So, for the Generalized Forking Lemma case, we have

$$\frac{\mathsf{Adv}}{t} \geq \frac{\epsilon^2}{t\cdot 256\cdot q\cdot 3.4} \geq \frac{\epsilon^2}{870\cdot t\cdot q}$$

  Since q, the number of oracle queries, can be potentially very large, using Generalized Forking Lemma seems to give a tighter reduction in the case of PointProofs.