Jackpot: Non-interactive Aggregatable Lotteries

Nils Fleischhacker[1(✉)], Mathias Hall-Andersen[2], Mark Simkin[3], and Benedikt Wagner[4]

[1] Ruhr University Bochum, Bochum, Germany
mail@nilsfleischhacker.de
[2] ZkSecurity, New York, USA
mathias@zksecurity.xyz
[3] Berlin, Germany
msimkin@gmx.de
[4] Ethereum Foundation, Berlin, Germany
benedikt.wagner@ethereum.org

Abstract. In proof-of-stake blockchains, liveness is ensured by repeatedly selecting random groups of parties as leaders, who are then in charge of proposing new blocks and driving consensus forward. The lotteries that elect those leaders need to ensure that adversarial parties are not elected disproportionately often and that an adversary can not tell who was elected before those parties decide to speak, as this would potentially allow for denial-of-service attacks. Whenever an elected party speaks, it needs to provide a winning lottery ticket, which proves that the party did indeed win the lottery. Current solutions require all published winning tickets to be stored individually on-chain, which introduces undesirable storage overheads.

In this work, we introduce *non-interactive aggregatable lotteries* and show how these can be constructed efficiently. Our lotteries provide the same security guarantees as previous lottery constructions, but additionally allow any third party to take a set of published winning tickets and aggregate them into one short digest. We provide a formal model of our new primitive in the universal composability framework.

As one of our technical contributions, which may be of independent interest, we introduce aggregatable vector commitments with simulation-extractability and present a concretely efficient construction thereof in the algebraic group model in the presence of a random oracle. We show how these commitments can be used to construct non-interactive aggregatable lotteries. We have implemented our construction, called *Jackpot*, and provide benchmarks that underline its concrete efficiency.

M. Hall-Andersen—This work was done while this author was at Aarhus University. Funded by the Concordium Foundation.
B. Wagner—This work was done while the author was at CISPA Helmholtz Center for Information Security. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 507237585.
M. Simkin—Independent Researcher.

K.-M. Chung and Y. Sasaki (Eds.): ASIACRYPT 2024, LNCS 15489, pp. 365–397, 2025.
https://doi.org/10.1007/978-981-96-0938-3_12

Keywords: Lotteries · Aggregation · Vector Commitments · Simulation-Extractability · KZG Commitments

1 Introduction

Blockchains rely on lottery mechanisms for repeatedly electing one or multiple leaders at random from the pool of all participants. These leaders are then in charge of proposing new blocks and driving the protocol's consensus forward, thereby ensuring liveness of the blockchain. In proof-of-stake blockchains, the participants' probabilities of being elected are tied to their stake, i.e., to the amount of money they have put into the system. In Ethereum, each participant deposits a fixed amount of money to participate in the lotteries and thus everybody has the same probability of being elected. In Algorand [26], on the other hand, participants may have deposited different amounts of money and therefore have different probabilities of being elected.

In the context of proof-of-stake blockchains a lottery mechanism needs to satisfy several properties. From a security perspective, lotteries should not allow corrupt parties to be elected disproportionately often. Lotteries should hide who the elected leaders are, as an adversary could otherwise prevent the chain from growing by taking the leaders off the network right after they have been elected, but before they have had a chance to speak. Leaders should privately learn whether they won the lottery and obtain a publicly verifiable winning ticket. When a leader is ready to speak, they can attach the winning ticket to their message, so that everybody can verify that they are indeed one of the leaders.

From an efficiency perspective, lotteries should aim to minimize both the network bandwidth and storage overheads that they incur, since new leaders may need to be elected frequently among a large number of participants. In terms of bandwidth overhead we would like to minimize the amount of communication needed to run each lottery. In terms of storage overhead we would like to minimize the amount of memory needed to store all published winning tickets. Ideally, we would like the storage overhead to grow sublinearly in the number of published winning tickets.

Various constructions of lotteries schemes have already been proposed in the literature, but all of them either do not keep the lottery output secret [1,4,11], require a trusted party [16,31], or have storage overheads that are linear in the number of published winning tickets [17,26] per election.

1.1 Our Contribution

In this work, we introduce *non-interactive aggregatable lotteries*. In this setting we have a set of parties where each party is identified by a short verification key and holds a corresponding secret key. We assume the existence of a randomness beacon functionality which broadcasts uniformly random values to all parties in regular intervals. We will associate the randomness beacon output at time t with the t-th lottery execution.

Whenever the randomness beacon outputs a lottery seed, every party can, without interacting with the other parties, check whether they have won the current lottery. Each party will win each lottery independently with probability $1/k$ for some fixed parameter[1] k. Maliciously generated keys do not allow the adversary to increase their winning probabilities or to coordinate which corrupt parties win which lotteries at which times. The adversary is not able to determine which honest parties are winning which elections with probability noticeably better than guessing. Each winning party can locally compute a publicly verifiable proof, the winning ticket, that allows them to convince other parties that they won a lottery. Finally, and most importantly, the lotteries are aggregatable. By this we mean that all published winning tickets belonging to the same lottery execution can be compressed into one short ticket by any (possibly untrusted) third party. Given the public keys of all winning parties and the compressed lottery ticket anybody can still be convinced of the fact that each individual party won the lottery. We formally model these lotteries in the universal composability (UC) framework of Canetti [13].

Lotteries from Simulation-Extractable Vector Commitments. We introduce the notion of aggregatable vector commitments with a strong simulation-extractability property and show that these commitments can be used to instantiate our non-interactive aggregate lotteries. On an intuitive level, a vector commitment is said to be aggregatable if openings belonging to different commitments can be compressed into one short opening. A vector commitment is said to be simulation-extractable if it satisfies the following two properties: in security proofs, knowing a trapdoor, we can issue dummy commitments and later open those to arbitrary messages at arbitrary positions. Additionally we can extract the committed messages from any valid but adversarially chosen commitment. While our notion effectively requires the commitments to be "non-malleable", the openings of such a commitment scheme can still have homomorphic properties, which is of crucial importance for being able to aggregate them.

Simulation-Extractable Vector Commitments from KZG. We present a construction of such an aggregatable vector commitment with simulation-extractability proven secure in the algebraic group model (AGM) [24]. Our construction is a modification of the polynomial commitment scheme of Kate, Zaverucha, and Goldberg (KZG) [30] and uses the exact same trusted setup. While KZG itself is malleable and can therefore not be simulation-extractable, we show that our construction is simulation-extractable. At the same time it preserves the homomorphic properties of KZG needed for aggregation. The proof turns out to be rather involved and we present it in a modular way. We believe that our construction, our notion of simulation-extractability, and our modular proof may be of independent interest beyond their applications in this work.

Implementation and Benchmarks. To show the practicality of our construction, called *Jackpot*, we have implemented it and provide benchmarks for various parameter settings. For instance, Jackpot allows for aggregating 2048 winning tickets in less than 15 milliseconds and verifying the aggregated ticket takes less than 17 milliseconds on a regular Macbook Pro. Storing the 2048 winning tickets

[1] We also show how to generalize our notion of lotteries and our constructions to the setting where parties have different winning probabilities.

in aggregated form is 1228.8 times more efficient than storing a list of all tickets of a state-of-the-art lottery based on VRFs explicitly. The main bottleneck of our construction is the time it takes to generate the public keys. For generating a public key that is good for 2^{20} lotteries, i.e., for one lottery every 5 minutes for 10 years nonstop, our protocol takes around 8 seconds. The corresponding public key is 160 bytes large.

1.2 Related Work

Lotteries have appeared throughout cryptographic research in various shapes and forms. In the following we discuss a few of those research works and highlight how they differ from ours.

Lotteries without Secrecy. The problem of allowing a group of parties to select a random set of leaders among them has already been addressed by Broder and Dolev [11] over 40 years ago. Their work, however, requires a large amount of interaction during each election and does not hide who is elected. The works of Bentov and Kumaresan [4] and of Bartoletti and Zunino [1] allow parties to run financial lotteries that enjoy certain fairness properties on top of cryptocurrencies like Bitcoin or Ethereum. Here each party can deposit a coin and a random parties is elected to be the winner that obtains all deposited coins. Neither of those protocols provides any privacy guarantees and their techniques do not seem applicable to our setting.

Lotteries without Aggregation. A lottery that satisfies all of our desired properties apart from aggregation was proposed by Gilad et al. [26]. In their construction each party is identified via a public key for a verifiable random functions (VRF) [37]. The public key of party i can be viewed as a commitment to a secret random function f_i and, using their corresponding secret key, party i is able to output pairs (x, y) and prove that $y = f_i(x)$. Whenever a randomness beacon provides lseed, party i can check whether they won the corresponding lottery by computing whether $f_i(\mathsf{lseed}) < k$ for some parameter k. Since the function is random, nobody can predict whether party i wins a lottery. At the same time the verifiability property of the random function allows party i to claim the win. In subsequent work David et al. [17] properly formalized this approach and showed that the VRF actually needs to satisfy an additional property ensuring that high entropy inputs produce high entropy outputs even if the VRF keys were chosen by a malicious party.

Both works [17,26] show different ways of how their lotteries can then be used to select committees that then drive consensus forward in their respective blockchain designs. Both works would benefit from being able to aggregate lottery tickets as it would allow them to reduce their storage complexities.

Single Secret Leader Elections. A recent work by Boneh et al. [7] introduces the problem of secret leader elections and shows how it can be solved using cryptographic tools like indistinguishability obfuscation [25], threshold fully homomorphic encryption [8], or proofs of correct shuffles [2]. Whereas our work focuses on electing a certain number of leaders *in expectation*, they focus on computing an ordered list of an *exact* number of leaders. As their problem is significantly harder to solve, their protocols are significantly more expensive computationally and require large amounts of interaction for each lottery.

Aggregatable Vector Commitments. We mentioned above that our main technical tool is an aggregatable vector commitment that satisfies a strong form of simulation-extractability. Various aggregatable or linearly homomorphic vector commitments [14,22,23,27,32–34,38,41] have previously been proposed but all of these works fail to achieve simulation-extractability which is of crucial importance for our application.

On a technical level a recent result by Faonio et al. [18] uses some observations similar to ours. They construct simulation-extractable succinct non-interactive arguments of knowledge (SNARKs) [28,39]. To this end they show that the KZG polynomial commitment scheme satisfies a weak notion of simulation-extractability in the AGM. Indeed, there is no hope of proving full simulation-extractability for KZG commitments as both commitments and openings are homomorphic. Conceptually, both our work and theirs show that opening KZG at a random point chosen after the commitment is fixed makes the commitment simulation-extractable. However, we highlight three important differences: firstly, the notion that they show for the original KZG construction is tailored to their specific use-case in SNARKs. Contrary to that, we define a new scheme and show a full simulation-extractability notion that is more self-contained. Secondly, their notion states that we can extract a preimage from a KZG commitment (under certain restrictions), whereas our notion additionally guarantees that any future (aggregated) opening provided by the adversary is consistent with the extracted preimage. We can view this as a new form of binding for aggregated openings that composes with extraction. Interestingly, Faonio et al. also need a binding property, but only implicitly show it during the compilation to a SNARK. Finally, our analysis is more modular: we manage to generically separate simulation, extractability, and binding aspects.

In a concurrent and independent work, Libert [35] also constructs a simulation-extractable version of KZG commitments. However, the goals of our work and Libert's work are orthogonal: Libert's construction allows to commit to multivariate polynomials and can be used in HyperPlonk [15]. At the same time, openings can not be aggregated, which is an essential feature of our construction. Indeed, openings in Libert's construction contain a non-interactive Schnorr-style proof [40]. While such proofs can be batched while they are created, it is not possible to aggregate given proofs publicly.

1.3 Technical Overview

One way of instantiating VRFs for lotteries that rely on them, e.g. [17,26], is to use the unique signature scheme of Boneh, Lynn, and Shacham (BLS) [10] as a verifiable unpredictable function and then apply a random oracle to the signature to make the output pseudorandom. More concretely, whenever the randomness beacon outputs the unpredictable lottery seed lseed, each participant j signs

lseed (as well as potentially additional context such as their own identity) using their BLS signing key sk_j resulting in a unique signature σ_j. Participant j wins the lottery iff $\mathsf{H}(\sigma_j) < t$, where H is a random oracle and t is an appropriate threshold to achieve the desired winning probability. To prove that they won, the party presents σ_j as their winning ticket. Anyone can verify that they won by verifying the signature using the BLS public key pk_j and checking that indeed $\mathsf{H}(\sigma_j) < t$.

When considering the possibility of aggregating winning tickets, the use of BLS might seem promising at first glance. After all, BLS signatures are known to be aggregatable [9] even in the presence of rogue keys [6] by computing a random linear combination of the signatures. One might thus be tempted to store this short aggregated signature σ instead of a long list of all individual signatures. Alas, this does not work. Although we could still verify that all aggregated σ_j were valid, the exact values of the individual signatures would be lost. We therefore could not recompute their individual hash values to check that all aggregated tickets were winning tickets.

The first idea to solve this dilemma is to try to avoid using the random oracle and directly look for a VRF with nice linearity properties. Specifically, let $(\mathsf{pk}_j, \mathsf{sk}_j)$ be key pairs of a VRF and let $y_j = \mathsf{VRF}(\mathsf{sk}_j, x)$. Further, let τ_j be proofs of the former equality. Then, we want that the following holds for arbitrary weights ξ_j:

$$\mathsf{VRF.Ver}\Big(\sum_{j=1}^{n} \xi_j \mathsf{pk}_j, x, \sum_{j=1}^{n} \xi_j y_j, \sum_{j=1}^{n} \xi_j \tau_j\Big) = 1 \tag{1}$$

The ith round of the lottery could now proceed as follows: given lseed, derive per party challenges x_j. Party j wins the lottery iff $\mathsf{VRF}(\mathsf{sk}_j, (i, \mathsf{lseed})) = x_j$. The corresponding winning ticket is the proof τ_j. Using the linearity of the VRF, we could aggregate the proofs by computing a random linear combination of the winning tickets and weights $(\xi_1, \ldots, \xi_n)$, which are obtained by hashing the set of public keys. The aggregated ticket $\tau = \sum_{i=1}^{n} \xi_j \tau_j$ allows full verification of all proofs via Eq. (1) simultaneously.

For this construction to be sensible we would, however, require a linearly homomorphic VRF with small codomain. Specifically, to achieve a winning probability of $1/k$, the VRF needs a codomain of size exactly k. There are currently no known constructions of such VRFs for usefully small values of k. Fortunately, we can still make the above approach work, if we are willing to make some concessions, namely that a public key will only be valid for a limited number T of successive lotteries. Since T can be chosen sufficiently large for practical purposes and because we can simply generate fresh keys after T lotteries, the concession we make is rather small.

Naive Homomorphic VRFs via Vector Commitments. If we use a vector commitment to commit to a uniformly random vector $\mathbf{v} \in [k]^T$, it can in many ways be viewed as a VRF with domain $[T]$ and codomain $[k]$. The public key is now the commitment and the secret key is the vector $\mathbf{v}$ as well as the randomness used to commit. To participate in T lotteries each party j initially commits to a random vector $\mathbf{v}^{(j)} \in [k]^T$. In the ith lottery round we again derive per party challenges x_j from lseed and party j wins iff $\mathbf{v}_i^{(j)} = x_j$. Each party can *prove* that they won by revealing an opening for position i of their commitment. If the vector commitment has the required homomorphic properties of Eq. (1), we can verify all openings using only the aggregated opening. Luckily for us, such linearly homomorphic vector commitments do exist, with KZG [30] being the most prominent among them.

The Woes of Universal Composability. For our lottery scheme to be useful as part of more complex protocols, it is necessary that it composes securely with itself and other protocols. To this end, we define the security of a lottery scheme in the universal composability (UC) framework [13]. This, however, causes issues with the proof of the construction sketched above. Namely, in the security proof the simulator would need to both equivocate commitments for honest participants and extract from commitments of corrupted participants. This implies that the vector commitment requires some kind of simulation-extractability, i.e., a guarantee that it is possible to extract preimages from any valid commitment produced by an adversary, even if the adversary was previously given equivocal commitments (from which extraction would not be possible).

Unfortunately, not only does KZG not have this property, the required simulation-extractability and the linear homomorphism described above in fact contradict each other. Let com be a valid *simulated* commitment and let τ be an opening proving that com contained x at position i. Then by the linear homomorphism $\mathsf{com}' = \mathsf{com} + \mathsf{com}$ is also a valid commitment and $\tau' = \tau + \tau$ could be used to prove that com' contained $x + x$ at position i. However, it would not be possible to extract a preimage from com'. We thus need to depart from using a regular linear homomorphism for aggregation.

Making KZG Simulation-Extractable. To get around this problem, we make the commitments non-malleable, while maintaining the linear homomorphism on the openings (and a part of the commitments). An expensive black-box way of achieving this might be to add a simulation-extractable proof of knowledge of the secret vector to the commitment. Instead, we can leverage the fact that KZG is not just a vector commitment, but a polynomial commitment. When KZG is used to commit to a vector $\mathbf{v} \in [k]^T$, we are actually committing to the polynomial f of degree $T-1$ over a large field $\mathbb{F}$ that is uniquely defined by the points $(j, \mathbf{v}_j)$. While we have only explicitly defined f on $[T] \subset \mathbb{F}$, we can still open the commitment at any position in $\mathbb{F}$. Now, the idea is to force anyone presenting a fresh commitment to also open their commitment at a random position. If the commitment is derived from simulated commitments, then providing such an opening should not be possible. Since this is an additional opening we need to increase the degree of the polynomial to T and they will turn out that a technicality in the proof actually requires the degree to be $T+1$. The actual construction of our simulation-extractable vector commitment will work as follows: to commit to a vector $\mathbf{v} \in \mathbb{F}^T$ we uniformly choose a polynomial f of degree $T+1$ conditioned on $f(j) = \mathbf{v}_j$ for $j \in [T]$ and commit to it using a regular

KZG commitment $\mathsf{com}_{\mathsf{KZG}}$. The full commitment then consists of $\mathsf{com}_{\mathsf{KZG}}$ as well as an opening of the commitment at position $\mathsf{H}(\mathsf{com}_{\mathsf{KZG}})$ where H is a random oracle mapping to $\mathbb{F}$. The idea is that whenever an adversary would derive a commitment from existing commitments, they would need to open their commitment at a new random position, which the hiding property of KZG should prevent them from doing. At the same time, aggregation of openings can still be done using a random linear combination, just as with regular KZG. Aggregated openings can be verified given the list of commitments by verifying that each individual commitment is indeed valid and then using the linear combination of the KZG part of the commitments to verify the aggregated opening. Finally, we note that while our commitment is conceptually simple, the proof that it provides simulation-extractability is far from it.

On the Necessity of Randomness Beacons. Throughout our paper, we assume that all parties have access to a randomness beacon. It is sensible to ask how necessary this assumption is. Intuitively, we would like our lotteries to ensure that no party can predict when they will win a lottery. For this to be feasible, there needs to be a source of entropy associated with each lottery execution, which is exactly what a randomness beacon provides. From a practical perspective, assuming the existence of a randomness beacon is also not too problematic, as they are deployed and running already. In the context of Ethereum, for example, the randomness beacon is known as Randao[2].

On Simulation-Based Security. In this work, we have chosen to define aggregatable lotteries through ideal functionalities in the UC framework. An alternative approach could have been to give game-based definitions. We believe that ideal functionalities are the right approach here for two reasons. Firstly, it is not at all clear what equivalent "clean" game-based notions would look like. Designing game-based notions that, for example, ensure that the adversary does not win disproportionally often or that the winning probabilities in each lottery are independent is non-trivial and would result in complex definitions. This would then make using our primitive in other contexts more cumbersome. Secondly, we would like to guarantee that our lotteries remain secure, even if composed arbitrarily with other protocols. Ideal functionalities in the UC model provide us with this guarantee, whereas game-based notions do not in general.

Parallels with Multi-signatures. On a conceptual level, our contribution in this work has some strong parallels to multi-signatures [29,36]. These allow for aggregating many individual signatures for the same message into one short aggregate signature. Using multi-signatures one can significantly reduce the on-chain storage in blockchains like Ethereum, as each block only needs to store a small value, which simultaneously vouches for many different signers having approved the block's contents. Similarly, our aggregate lotteries allow for storing a short digest, which simultaneously vouches for all elected committee members within one election. Apart from our technical realization of such lotteries, we

[2] https://eth2book.info/capella/part2/building_blocks/randomness/.

view our conceptual idea of compressing this lotteries as one of our important contributions.

1.4 Paper Organization

The main part of the paper is organized as follows. In Sect. 3, we introduce syntax and game-based security notions for aggregatable vector commitments. We also present our construction of this primitive. Then, in Sect. 4 we define aggregatable lotteries in the UC framework, present our construction from any aggregatable vector commitment. We show UC security of our lottery assuming the vector commitment meets the game-based notions we have defined. Finally, in Sect. 5, we discuss practical aspects and the efficiency of our construction.

2 Preliminaries

In this section, we fix notation and recall relevant cryptographic preliminaries.

Notation. For a finite set S, writing $s \leftarrow_{\$} S$ means that s is sampled uniformly at random from S. For a probabilistic algorithm $\mathcal{A}$, we write $s := \mathcal{A}(x;\rho)$ to state that $\mathcal{A}$ is run on input x with random coins ρ, and the result is assigned to the variable s. If the coins ρ are sampled uniformly at random, we write $s \leftarrow \mathcal{A}(x)$. If we write $s \in \mathcal{A}(x)$, we mean that there are random coins such that when $\mathcal{A}$ is run on input x with these random coins, it outputs s. The security parameter λ is given implicitly to all algorithms (in unary). We denote the running time of an algorithm $\mathcal{A}$ by $\mathbf{T}(\mathcal{A})$. We use standard cryptographic notions, e.g., PPT, negligible. We define $[L] := \{1,\ldots,L\} \subseteq \mathbb{N}$. We let $\mathcal{B}(p)$ denote a Bernoulli distribution with $\Pr[b=1] = p$ for b sampled from $\mathcal{B}(p)$ (written as $b \leftarrow \mathcal{B}(p)$).

Pairings and Assumptions. We rely on the ℓ-DLOG assumption and the ℓ-SDH assumption [5,30]. For this and the remainder of this paper, let PGGen be an algorithm that on input 1^λ outputs the description of prime order groups $\mathbb{G}_1$, $\mathbb{G}_2, \mathbb{G}_T$ of order p, generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and the description of a pairing, i.e., a non-degenerate bilinear map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ for which $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$. That is, PGGen outputs $\mathsf{par} = (\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, p, e)$. Then, informally, the ℓ-DLOG assumption states that it is hard to output α given $(g_1^{\alpha^i})_{i=1}^{\ell}, g_2^{\alpha}$ for a random $\alpha \leftarrow_{\$} \mathbb{Z}_p$, and the ℓ-SDH assumption states that it is hard to output $(c, g_1^{1/(\alpha+c)})$ for some c on the same input. Clearly, ℓ-DLOG is implied by ℓ-SDH.

Definition 1 (ℓ-DLOGAssumption). *We say that the ℓ-DLOG assumption holds relative to PGGen, if for any PPT algorithm $\mathcal{A}$, the following advantage is negligible:*

$$\mathsf{Adv}^{\ell\text{-}\mathsf{DLOG}}_{\mathcal{A},\mathsf{PGGen}}(\lambda) := \Pr\left[\mathcal{A}(\mathsf{par},\mathsf{In}) = \alpha \;\middle|\; \begin{array}{l} \mathsf{par} \leftarrow \mathsf{PGGen}(1^\lambda), \\ \alpha \leftarrow_{\$} \mathbb{Z}_p,\ \mathsf{In} := ((g_1^{\alpha^i})_{i=1}^{\ell}, g_2^{\alpha}) \end{array}\right].$$

Definition 2 (ℓ-SDHAssumption). *We say that the ℓ-SDH assumption holds relative to* PGGen*, if for any PPT algorithm $\mathcal{A}$, the following advantage is negligible:*

$$\mathsf{Adv}^{\ell\text{-SDH}}_{\mathcal{A},\mathsf{PGGen}}(\lambda) := \Pr\left[\begin{array}{l} \exists c \in \mathbb{Z}_q \setminus \{-\alpha\} : \\ \mathcal{A}(\mathsf{par}, \mathsf{In}) = \left(c, g_1^{1/(\alpha+c)}\right) \end{array} \middle| \begin{array}{l} \mathsf{par} \leftarrow \mathsf{PGGen}(1^\lambda), \\ \alpha \leftarrow_\$ \mathbb{Z}_p, \\ \mathsf{In} := ((g_1^{\alpha^i})_{i=1}^{\ell}, g_2^{\alpha}) \end{array}\right].$$

Universal Composability. We define an ideal functionality for aggregatable lotteries and prove security of our construction in the universal composability (UC) framework [13] in the presence of static corruptions. Our construction relies on synchronous broadcast and a synchronous randomness beacon. We include definitions of the UC functionalities, protocols, and the security proof in our full version [21]. When specifying functionalities and simulators, we write $\mathsf{msg} \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{port}$ to denote the event of receiving the message msg on the (possibly emulated) port port. Correspondingly, we use $\mathsf{port} \overset{\mathsf{send}}{\hookleftarrow} \mathsf{msg}$ to denote the sending of the message msg on the (possibly emulated) port port.

Random Oracle Model. For some of our proofs, we use the (programmable) random oracle model (ROM) [3]. To recall, in the ROM, hash functions are modeled by oracles implementing perfectly random functions via lazy sampling. For our UC proof, we use the standard ROM, which is sometimes known as the local ROM as opposed to the global ROM [12].

Algebraic Group Model. For some of our proofs and extractors, we leverage the algebraic group model (AGM) [24]. In this model, we only consider so called algebraic algorithms. This means that whenever such an algorithm outputs a group element Y in some cyclic group $\mathbb{G}$ of prime order p, it also outputs a so called algebraic representation, which is a vector $(c_1, \ldots, c_k) \in \mathbb{Z}_p^k$ such that $Y = \prod_{i=1}^{k} X_i^{c_i}$. Here, $X_1, \ldots, X_k$ are all group elements that the algorithm received so far. We emphasize that we analyze the game-based security of some of our building blocks in the AGM, and then use this security in a black-box manner for our UC proof.

3 Aggregatable Vector Commitments

In this section, we define and instantiate a special class of vector commitments that we will use to construct aggregatable lotteries.

3.1 Syntax of Our Vector Commitments

A vector commitment allows a party to commit to a vector $\mathbf{m} \in \mathcal{M}^\ell$ over some alphabet $\mathcal{M}$, resulting in a commitment com. Later, the committer can open com at any position $i \in [\ell]$ by revealing $\mathbf{m}_i$ and a corresponding opening (proof) τ. One can then publicly verify the pair $(\mathbf{m}_i, \tau)$ with respect to com and i.

Our definition of vector commitments is special in two ways. First, it should be possible to publicly aggregate several openings for different commitments with respect to the same position. Precisely, we require the existence of an algorithm $\mathsf{Aggregate}$ that takes a list of L openings $\tau_j, j \in [L]$ (all for the same position $i \in [\ell]$) and outputs an aggregated opening τ. One can then verify τ with respect to a list of L commitments. For non-triviality, the aggregated τ should ideally be as large as one single τ_j. Note that a similar aggregation feature for openings of different commitments has been defined in [27]. The second non-standard part of our definition is that we explicitly model an algorithm VerCom that verifies whether commitments (not openings) are well-formed. For our security notions, this will be convenient.

Definition 3 (Vector Commitment Scheme). *A vector commitment scheme (VC) is a tuple* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$ *of PPT algorithms with the following syntax:*

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to \mathsf{ck}$ *takes as input the security parameter and a message length* ℓ*, and outputs a commitment key* ck*. We assume that* ck *specifies a message alphabet* $\mathcal{M}$*, opening space* $\mathcal{T}$*, and commitment space* $\mathcal{C}$*.*
- $\mathsf{Com}(\mathsf{ck}, \mathbf{m}) \to (\mathsf{com}, St)$ *takes as input a commitment key* ck *and a vector* $\mathbf{m} \in \mathcal{M}^\ell$*, and outputs a commitment* $\mathsf{com} \in \mathcal{C}$ *and a state* St*.*
- $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) \to b$ *is deterministic, takes as input a commitment key* ck *and a commitment* com*, and outputs a bit* $b \in \{0, 1\}$*.*
- $\mathsf{Open}(\mathsf{ck}, St, i) \to \tau$ *takes as input a commitment key* ck*, a state* St*, and an index* $i \in [\ell]$*, and outputs an opening* $\tau \in \mathcal{T}$*.*
- $\mathsf{Aggregate}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, (\tau_j)_{j=1}^L) \to \tau$ *is deterministic, takes as input a commitment key* ck*, an index* $i \in [\ell]$*, a list of commitments* $\mathsf{com}_j \in \mathcal{C}$*, a list of symbols* $m_j \in \mathcal{M}$*, and a list of openings* $\tau_j \in \mathcal{T}$*, and outputs an opening* $\tau \in \mathcal{T}$*.*
- $\mathsf{Ver}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau) \to b$ *is deterministic, takes as input a commitment key* ck*, an index* $i \in [\ell]$*, a list of commitments* $\mathsf{com}_j \in \mathcal{C}$*, a list of symbols* $m_j \in \mathcal{M}$*, and an opening* $\tau \in \mathcal{T}$*, and outputs a bit* $b \in \{0, 1\}$*.*
  Further, we require that the following properties holds:
  1. ***Commitment Completeness.*** *For any* $\ell \in \mathbb{N}$*, any* $\mathsf{ck} \in \mathsf{Setup}(1^\lambda, 1^\ell)$*, and any* $\mathbf{m} \in \mathcal{M}^\ell$*, we have*
  $$\Pr\left[\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 1 \mid (\mathsf{com}, St) \leftarrow \mathsf{Com}(\mathsf{ck}, \mathbf{m})\right] = 1.$$
  2. ***Opening Completeness.*** *For any* $\ell \in \mathbb{N}$*, any* $\mathsf{ck} \in \mathsf{Setup}(1^\lambda, 1^\ell)$*, any* $\mathbf{m} \in \mathcal{M}^\ell$*, and any* $i \in [\ell]$*, we have*
  $$\Pr\left[\mathsf{Ver}(\mathsf{ck}, i, \mathsf{com}, \mathbf{m}_i, \tau) = 1 \;\middle|\; \begin{array}{l}(\mathsf{com}, St) \leftarrow \mathsf{Com}(\mathsf{ck}, \mathbf{m}),\\ \tau \leftarrow \mathsf{Open}(\mathsf{ck}, St, j)\end{array}\right] = 1.$$
  3. ***Aggregation Completeness.*** *For any* $\ell \in \mathbb{N}$*, any* $\mathsf{ck} \in \mathsf{Setup}(1^\lambda, 1^\ell)$*, any* $L \in \mathbb{N}$*, any index* $i \in [\ell]$*, any list* $(m_j)_{j=1}^L \in \mathcal{M}^L$*, any list* $(\mathsf{com}_j)_{j=1}^L \in$

$\mathcal{C}^L$, any list $(\tau_j)_{j=1}^L \in \mathcal{T}^L$, we have

$$\begin{aligned} &\forall j \in [L] : \mathsf{Ver}(\mathsf{ck}, i, \mathsf{com}_j, m_j, \tau_j) = 1 \\ &\wedge\ \tau = \mathsf{Aggregate}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, (\tau_j)_{j=1}^L) \\ \Longrightarrow \quad &\mathsf{Ver}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau) = 1. \end{aligned}$$

3.2 Simulation-Extractability

We define a strong simulation-extractability property for vector commitments. This property captures all properties that we will need for our UC proof, including both hiding and binding properties. Beyond that, it may be interesting in itself. The notion states that no adversary can distinguish between two games in which it is running, where one game models the real world, and the other game models an ideal world. The first property that our notion models is a strong form of *hiding*. Namely, we require that there is a way to set up the commitment key with a trapdoor, and this trapdoor allows a simulator to compute commitments without knowing the message, and later open these commitments at arbitrary positions to arbitrary symbols. This is modeled in our notion as follows. In the real world game, the adversary gets an honest commitment key. It also gets access to an oracle GetCom that outputs honestly computed commitments to messages of the adversary's choice. Another oracle GetOp provides openings for these commitments when the adversary asks for them. In the ideal world game, the commitment key is set up with a trapdoor and both commitments and openings are simulated. In addition to this hiding property, our notion models a strong form of *binding*. Namely, the adversary gets access to oracles SubCom and SubOp that allow it to submit commitments and openings for them. While the commitments and openings are simply verified in the real world game, there are additional checks in the ideal world game. Concretely, when the adversary submits a commitment com that is not output by GetCom, the game not only verifies it, but also tries to extract a preimage $(\mathbf{m}, \varphi)$ from it, such that $\mathbf{m}$ with randomness φ commits to com. If this extraction fails but com verifies, SubCom outputs 0 in the ideal world game, whereas it would output 1 in the real world game. In other words, indistinguishability of the games ensures that we can always extract preimages of commitments. In addition, our notion ensures that openings are consistent: (1) whatever we extracted in SubCom is consistent with any valid opening that the adversary submits later, and (2) if the adversary opens a commitment output by GetCom($\mathbf{m}$) at position i, then (2a) it opens to the respective $\mathbf{m}_i$, and (2b) it queried GetOp for this commitment at position i before. Our notion ensures this because in the ideal game, SubOp outputs 0 if one of the inconsistencies (1, 2a, 2b) occurs, whereas in the real game the output of SubOp only depends on whether the opening verifies.

Definition 4 (Simulation-Extractability of VC). *Consider a vector commitment scheme* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$. *For any algorithm $\mathcal{A}$, any $\ell \in \mathbb{N}$, any algorithm* Ext, *and any triple of algorithms*

$\mathsf{Sim} = (\mathsf{TSetup}, \mathsf{TCom}, \mathsf{TOpen})$, *consider the game* ℓ-$\mathbf{SIM}$-$\mathbf{EXT}^{\mathcal{A}}_{\mathsf{VC},0}(\lambda)$ *and the game* ℓ-$\mathbf{SIM}$-$\mathbf{EXT}^{\mathcal{A}}_{\mathsf{VC},\mathsf{Sim},\mathsf{Ext},1}(\lambda)$ *defined in Fig. 1. We say that* VC *is simulation-extractable, if there are PPT algorithms* Ext *and* $\mathsf{Sim} = (\mathsf{TSetup}, \mathsf{TCom}, \mathsf{TOpen})$ *such that for any polynomial* $\ell \in \mathbb{N}$ *and any PPT algorithm* $\mathcal{A}$*, the following advantage is negligible:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{sim\text{-}ext}}_{\mathcal{A},\mathsf{VC},\mathsf{Sim},\mathsf{Ext},\ell}(\lambda) := \Big|&\Pr\left[\ell\text{-}\mathbf{SIM}\text{-}\mathbf{EXT}^{\mathcal{A}}_{\mathsf{VC},0}(\lambda) \Rightarrow 1\right]\\ &-\Pr\left[\ell\text{-}\mathbf{SIM}\text{-}\mathbf{EXT}^{\mathcal{A}}_{\mathsf{VC},\mathsf{Sim},\mathsf{Ext},1}(\lambda) \Rightarrow 1\right]\Big|.\end{aligned}$$

Then, we say VC *is simulation-extractable with extractor* Ext *and simulator* Sim*.*

Our simulation-extractability notion is well-suited for our UC proof. However, it models several distinct properties of the vector commitment simultaneously, which renders a direct proof of simulation-extractability complicated. Thus, we define three less complex security notions and show that in combination they imply simulation-extractability. The first notion, *equivocality*, is the hiding part of our simulation-extractability notion.

Definition 5 (Equivocal VC). *Consider a vector commitment scheme* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$*. For any algorithm* $\mathcal{A}$*, any* $\ell \in \mathbb{N}$*, and any triple of algorithms* $\mathsf{Sim} = (\mathsf{TSetup}, \mathsf{TCom}, \mathsf{TOpen})$ *consider the games* ℓ-$\mathbf{EQUIV}^{\mathcal{A}}_{\mathsf{VC},\mathsf{Sim},b}(\lambda)$ *for* $b \in \{0,1\}$ *defined in Fig. 2. We say that* VC *is equivocal, if there are PPT algorithms* $\mathsf{Sim} = (\mathsf{TSetup}, \mathsf{TCom}, \mathsf{TOpen})$ *such that for any polynomial* $\ell \in \mathbb{N}$ *and any PPT algorithm* $\mathcal{A}$*, the following advantage is negligible:*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{equiv}}_{\mathcal{A},\mathsf{VC},\mathsf{Sim},\ell}(\lambda) := \Big|&\Pr\left[\ell\text{-}\mathbf{EQUIV}^{\mathcal{A}}_{\mathsf{VC},0}(\lambda) \Rightarrow 1\right]\\ &-\Pr\left[\ell\text{-}\mathbf{EQUIV}^{\mathcal{A}}_{\mathsf{VC},\mathsf{Sim},1}(\lambda) \Rightarrow 1\right]\Big|.\end{aligned}$$

In this case, we say that VC *is equivocal with simulator* Sim*.*

The second and third notion focus on binding. Namely, the notion of *augmented extractability* states that we can extract preimages of commitments from any opening that the adversary outputs, even if it sees some honest commitments and openings. Notably, we do not allow the extractor to inspect the internal state of the oracles that output these honest commitments and openings, which is crucial for making this notion compose with equivocality.

Definition 6 (Augmented Extractability of VC). *Let* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$ *denote a vector commitment scheme. For any algorithm* $\mathcal{A}$*, any algorithm* Ext*, any* $\ell \in \mathbb{N}$*, consider the game* ℓ-$\mathbf{AUG}$-$\mathbf{EXT}^{\mathcal{A}}_{\mathsf{VC},\mathsf{Ext}}(\lambda)$ *defined in Fig. 3. We say that* VC *satisfies augmented*

Game ℓ-**SIM-EXT**$^{\mathcal{A}}_{\mathsf{VC},0}(\lambda)$

$c := 0,\ \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$
$\mathrm{O}_G := (\textsc{GetCom}_0, \textsc{GetOp}_0)$
$\mathrm{O}_S := (\textsc{SubCom}_0, \textsc{SubOp}_0)$
return $\mathcal{A}^{\mathrm{O}_G, \mathrm{O}_S}(\mathsf{ck})$

Oracle $\textsc{GetCom}_0(\mathbf{m})$

$c := c + 1,\quad \mathsf{Msgs}[c] := \mathbf{m}$
$(\mathsf{com}, \mathit{St}) \leftarrow \mathsf{Com}(\mathsf{ck}, \mathbf{m})$
$\mathsf{Coms}[c] := \mathsf{com},\quad \mathsf{St}[c] := \mathit{St}$
$\mathsf{Ops}[c] := \emptyset$
return com

Oracle $\textsc{GetOp}_0(k, i)$

if $\mathsf{Coms}[k] = \bot$: **return** $\bot$
if $i \in \mathsf{Ops}[k]$: **return** $\bot$
$\mathsf{Ops}[k] := \mathsf{Ops}[k] \cup \{i\}$
$\tau \leftarrow \mathsf{Open}(\mathsf{ck}, \mathsf{St}[k], i)$
return τ

Oracle $\textsc{SubCom}_0(\mathsf{com})$

if $\exists k$ s.t. $\mathsf{Coms}[k] = \mathsf{com}$:
  return 0
if $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 0$:
  return 0
$\mathsf{Sub} := \mathsf{Sub} \cup \{\mathsf{com}\}$
return 1

Game ℓ-**SIM-EXT**$^{\mathcal{A}}_{\mathsf{VC},\mathsf{Sim},\mathsf{Ext},1}(\lambda)$

$c := 0,\ (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{TSetup}(1^\lambda, 1^\ell)$
$\mathrm{O}_G := (\textsc{GetCom}_1, \textsc{GetOp}_1)$
$\mathrm{O}_S := (\textsc{SubCom}_1, \textsc{SubOp}_1)$
return $\mathcal{A}^{\mathrm{O}_G, \mathrm{O}_S}(\mathsf{ck})$

Oracle $\textsc{GetCom}_1(\mathbf{m})$

$c := c + 1,\quad \mathsf{Msgs}[c] := \mathbf{m}$
$(\mathsf{com}, \mathit{St}) \leftarrow \mathsf{TCom}(\mathsf{ck})$
$\mathsf{Coms}[c] := \mathsf{com},\quad \mathsf{St}[c] := \mathit{St}$
$\mathsf{Ops}[c] := \emptyset$
return com

Oracle $\textsc{GetOp}_1(k, i)$

if $\mathsf{Coms}[k] = \bot$: **return** $\bot$
if $i \in \mathsf{Ops}[k]$: **return** $\bot$
$\mathsf{Ops}[k] := \mathsf{Ops}[k] \cup \{i\}$
$\tau \leftarrow \mathsf{TOpen}(\mathsf{td}, \mathsf{St}[k], i, \mathsf{Msgs}[k]_i)$
return τ

Oracle $\textsc{SubCom}_1(\mathsf{com})$

if $\exists k$ s.t. $\mathsf{Coms}[k] = \mathsf{com}$:
  return 0
if $\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 0$:
  return 0
$(\mathbf{m}, \varphi) \leftarrow \mathsf{Ext}(\mathsf{td}, \mathsf{com})$
$(\mathsf{com}', \mathit{St}) := \mathsf{Com}(\mathsf{ck}, \mathbf{m}; \varphi)$
if $\mathsf{com}' \neq \mathsf{com}$: **return** 0
$\mathsf{MsgsExt}[\mathsf{com}] := \mathbf{m}$
$\mathsf{Sub} := \mathsf{Sub} \cup \{\mathsf{com}\}$
return 1

Oracle $\textsc{SubOp}_b(i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau)$

if $b = 1$: **for** $j \in [L]$:
  if $\mathsf{com}_j \in \mathsf{Sub} \wedge m_j \neq \mathsf{MsgsExt}[\mathsf{com}]_i$: **return** 0
  if $\exists k$ s.t. $\mathsf{com}_j = \mathsf{Coms}[k] \wedge i \in \mathsf{Ops}[k] \wedge m_j \neq \mathsf{Msgs}[k]_i$: **return** 0
  if $\exists k$ s.t. $\mathsf{com}_j = \mathsf{Coms}[k] \wedge i \notin \mathsf{Ops}[k]$: **return** 0
return $\mathsf{Ver}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau)$

Fig. 1. The simulation-extractability games ℓ-**SIM-EXT** for a vector commitment $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$, an adversary $\mathcal{A}$, an extractor Ext, and a simulator $\mathsf{Sim} = (\mathsf{TSetup}, \mathsf{TCom}, \mathsf{TOpen})$. In the random oracle model, Ext gets as additional input the list of random oracle queries of $\mathcal{A}$. In the algebraic group model, Ext gets as additional input the algebraic representation of all group elements contained in the commitment com submitted by $\mathcal{A}$.

Game ℓ-$\mathbf{EQUIV}^{\mathcal{A}}_{\mathsf{VC},0}(\lambda)$	**Game** ℓ-$\mathbf{EQUIV}^{\mathcal{A}}_{\mathsf{VC},\mathsf{Sim},1}(\lambda)$
$c := 0$	$c := 0$
$\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$	$(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{TSetup}(1^\lambda, 1^\ell)$
return $\mathcal{A}^{\textsc{GetCom}_0, \textsc{GetOp}_0}(\mathsf{ck})$	**return** $\mathcal{A}^{\textsc{GetCom}_1, \textsc{GetOp}_1}(\mathsf{ck})$

Fig. 2. The equivocality games ℓ-**EQUIV** for a vector commitment $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$, an adversary $\mathcal{A}$, and a simulator $\mathsf{Sim} = (\mathsf{TSetup}, \mathsf{TCom}, \mathsf{TOpen})$. Oracles $\textsc{GetCom}_b$ and $\textsc{GetOp}_b$ are as in Fig. 1.

extractability, if there is a PPT algorithm Ext *such that for any polynomial* $\ell \in \mathbb{N}$ *and any PPT algorithm* $\mathcal{A}$*, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{aug\text{-}ext}}_{\mathcal{A},\mathsf{VC},\mathsf{Ext},\ell}(\lambda) := \Pr\left[\ell\text{-}\mathbf{AUG}\text{-}\mathbf{EXT}^{\mathcal{A}}_{\mathsf{VC},\mathsf{Ext}}(\lambda) \Rightarrow 1\right].$$

In this case, we say that VC *satisfies augmented extractability with extractor* Ext.

Augmented extractability states that we can extract some preimage of adversarially submitted commitments. It does not state that what we extract is consistent with whatever the adversary opens later. For that, we define *aggregation position-binding*. Intuitively, we want that any two lists of commitments and openings that an adversary outputs are consistent, i.e., if they share a commitment, then the opened symbols for that commitment are the same. It turns out that we can further simplify this by assuming that one of the lists contains exactly one honestly computed commitment (with potentially biased randomness).

Definition 7 (Aggregation Position-Binding of VC). *Let* $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$ *be a vector commitment scheme. For any algorithm* $\mathcal{A}$ *and any* $\ell \in \mathbb{N}$*, consider the game* ℓ-$\mathbf{A}$-$\mathbf{POS}$-$\mathbf{BIND}^{\mathcal{A}}_{\mathsf{VC}}(\lambda)$ *defined in Fig. 4. We say that* VC *is aggregation position-binding, if for any polynomial* $\ell \in \mathbb{N}$ *and any PPT algorithm* $\mathcal{A}$*, the following advantage is negligible:*

$$\mathsf{Adv}^{\mathsf{a\text{-}pos\text{-}bind}}_{\mathcal{A},\mathsf{VC},\ell}(\lambda) := \Pr\left[\ell\text{-}\mathbf{A}\text{-}\mathbf{POS}\text{-}\mathbf{BIND}^{\mathcal{A}}_{\mathsf{VC}}(\lambda) \Rightarrow 1\right].$$

Next, we show that our notions imply simulation-extractability.

Lemma 1 (Informal). *If a vector commitment is equivocal, aggregation position-binding, and satisfies augmented extractability, then it is simulation-extractable.*

We give the formal statement and proof in our full version [21]. Here, we sketch a proof: we need to show that the real game and the ideal game of simulation-extractability are indistinguishable. For that, we start with the real game. In a first step, we change the game by extracting from all commitments that the adversary submits via $\textsc{SubCom}$, and let the oracle return 0 if extraction does not yield a valid preimage. The games are indistinguishable by augmented extractability. Note that now oracle $\textsc{SubCom}$ is as in the ideal game. In the

Game ℓ-AUG-EXT$^{\mathcal{A}}_{\mathsf{VC},\mathsf{Ext}}(\lambda)$

$\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$
$\mathsf{com} \leftarrow \mathcal{A}^{\textsc{GetCom}_0, \textsc{GetOp}_0}(\mathsf{ck})$
$(\mathbf{m}, \varphi) \leftarrow \mathsf{Ext}(\mathsf{ck}, \mathsf{com}), \quad (\mathsf{com}', St) := \mathsf{Com}(\mathsf{ck}, \mathbf{m}; \varphi)$
if $\nexists k$ s.t. $\mathsf{Coms}[k] = \mathsf{com} \wedge \mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 1 \wedge \mathsf{com} \neq \mathsf{com}'$: **return** 1
return 0

Fig. 3. The augmented extractability game ℓ-**AUG-EXT** for a vector commitment $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$, an extractor Ext, and an adversary $\mathcal{A}$. Oracles $\textsc{GetCom}_0$ and $\textsc{GetOp}_0$ are as in Fig. 1. In the random oracle model, Ext gets as additional input the list of random oracle queries of $\mathcal{A}$. In the algebraic group model, Ext gets as additional input the algebraic representation of all group elements contained in the commitment com submitted by $\mathcal{A}$. Notably, Ext does not share any internal state with the rest of the game.

Game ℓ-A-POS-BIND$^{\mathcal{A}}_{\mathsf{VC}}(\lambda)$

$\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$
$(\mathbf{m}, \varphi, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau) \leftarrow \mathcal{A}(\mathsf{ck})$
$(\mathsf{com}, St) := \mathsf{Com}(\mathsf{ck}, \mathbf{m}; \varphi)$
if $\mathsf{Ver}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau) = 0$: **return** 0
if $\exists j^* \in [L]$ s.t. $\mathsf{com}_{j^*} = \mathsf{com} \wedge m_{j^*} \neq \mathbf{m}_i$: **return** 1
return 0

Fig. 4. The aggregation position-binding game ℓ-**A-POS-BIND** for a vector commitment $\mathsf{VC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerCom}, \mathsf{Open}, \mathsf{Aggregate}, \mathsf{Ver})$ and an adversary $\mathcal{A}$.

second step, we change oracle SubOp to be as in the ideal world game as well. The adversary can only distinguish this, if one of the three conditions on which the implementations of oracle SubOp in the real game and the ideal game differ occurs. It turns out that we can bound this probability using aggregation position-binding, see the full proof for more details. Now, it remains to change the implementation of oracles GetCom and GetOp to be as in the ideal game. This change can be done using equivocality of the commitment. Here, it is essential that we defined our extractor in an appropriate way, see Fig. 3: the extractor does not rely on any internals of the oracles GetCom and GetOp and just sees their input and output behavior. Otherwise, a reduction for this final change would not be able to run the extractor correctly.

3.3 Simulation-Extractable Vector Commitments from KZG

We present an instantiation of vector commitments with suitable properties based on the KZG commitment scheme [30]. We first recall the KZG commitment scheme [30]. Then, we modify it to get our vector commitment scheme with suitable properties.

- $\mathsf{KZG.Setup}(1^\lambda, 1^d) \rightarrow \mathsf{ck}$:
  1. Run $\mathsf{par} := (\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, p, e) \leftarrow \mathsf{PGGen}(1^\lambda)$.
  2. Sample $\alpha \leftarrow_\$ \mathbb{Z}_p$ and $\beta \leftarrow_\$ \mathbb{Z}_p^*$, and set $h_1 := g_1^\beta \in \mathbb{G}_1$.
  3. Set $u_i := g_1^{\alpha^i}$ and $\hat{u}_i := h_1^{\alpha^i}$ for each $i \in \{0, \ldots, d\}$. Set $R := g_2^\alpha$
  4. Return $\mathsf{ck} := (\mathsf{par}, h_1, R, (u_i)_{i=0}^d, (\hat{u}_i)_{i=0}^d)$.
- $\mathsf{KZG.Com}(\mathsf{ck}, f \in \mathbb{Z}_p[X]) \rightarrow (\mathsf{com}, St)$
  1. If the degree of f is larger than d, abort.
  2. Sample a polynomial $\hat{f} \in \mathbb{Z}_p[X]$ of degree d uniformly at random.
  3. Compute $\mathsf{com} = g_1^{f(\alpha)} \cdot h_1^{\hat{f}(\alpha)}$. Note that com can be computed efficiently.
  4. Return com and $St := (f, \hat{f})$.
- $\mathsf{KZG.Open}(\mathsf{ck}, St = (f, \hat{f}), z) \rightarrow \tau$
  1. Let $\psi := (f - f(z))/(X - z) \in \mathbb{Z}_p[X]$ and $\hat{\psi} := (\hat{f} - \hat{f}(z))/(X - z) \in \mathbb{Z}_p[X]$.
  2. Set $v := g_1^{\psi(\alpha)} \cdot h_1^{\hat{\psi}(\alpha)}$. Note that v can be computed efficiently.
  3. Return $\tau := (\hat{f}(z), v)$.
- $\mathsf{KZG.Ver}(\mathsf{ck}, \mathsf{com}, z, y, \tau = (\hat{y}, v)) \rightarrow b$
  1. If $e\left(\mathsf{com} \cdot g_1^{-y} \cdot h_1^{-\hat{y}}, g_2\right) = e\left(v, R \cdot g_2^{-z}\right)$, return $b = 1$. Else, return $b = 0$.

Let $\mathsf{H}\colon \{0,1\}^* \rightarrow \mathbb{Z}_p$ and $\mathsf{H}'\colon \{0,1\}^* \rightarrow \mathbb{Z}_p$ be random oracles. We now define the vector commitment scheme $\mathsf{VC}_{\mathsf{KZG}} = (\mathsf{VC}_{\mathsf{KZG}}.\mathsf{Setup}, \mathsf{VC}_{\mathsf{KZG}}.\mathsf{Com}, \mathsf{VC}_{\mathsf{KZG}}.\mathsf{Open}, \mathsf{VC}_{\mathsf{KZG}}.\mathsf{Ver})$. Roughly, we use the linear properties of KZG to make aggregation work. To add non-malleability at the same time, we include an opening at a random position z_0 in the commitments. Typically, to commit to a vector of ℓ elements, one would work with polynomials of degree $d = \ell - 1$. As we give out one additional point $f(z_0)$ for non-malleability and still need hiding, it is natural to increase d by one to $d = \ell$. It turns out that for a technical reason in our extractability proof (see paragraph "Proof Strategy" in the proof of Lemma 4), we have to choose $d = \ell + 1$.

- $\mathsf{VC}_{\mathsf{KZG}}.\mathsf{Setup}(1^\lambda, 1^\ell) \rightarrow \mathsf{ck}$:
  1. Run $\mathsf{ck}_{\mathsf{KZG}} \leftarrow \mathsf{KZG.Setup}(1^\lambda, 1^d)$, where $d := \ell + 1$. The parameters specify message alphabet $\mathcal{M} := \mathbb{Z}_p$, opening space $\mathcal{T} := \mathbb{Z}_p \times \mathbb{G}_1$, and commitment space $\mathcal{C} := \mathbb{G}_1 \times \mathbb{Z}_p \times \mathcal{T}$.
  2. Let $\iota\colon [\ell] \rightarrow \mathbb{Z}_p$ be a fixed injection and $z_{\mathsf{out}} \in \mathbb{Z}_p$ such that 0 and z_{out} are not in the image of ι.
  3. Return $\mathsf{ck} := (\mathsf{ck}_{\mathsf{KZG}}, z_{\mathsf{out}}, \iota)$.
- $\mathsf{VC}_{\mathsf{KZG}}.\mathsf{Com}(\mathsf{ck}, \mathbf{m}) \rightarrow (\mathsf{com}, St)$
  1. Sample $\delta_0, \delta_1 \leftarrow_\$ \mathbb{Z}_p$ and let $f \in \mathbb{Z}_p[X]$ be the unique polynomial of degree $d := \ell + 1$ such that $f(0) = \delta_0$, $f(z_{\mathsf{out}}) = \delta_1$ and $f(\iota(i)) = \mathbf{m}_i$ for all $i \in [\ell]$.
  2. Run $(\mathsf{com}_{\mathsf{KZG}}, St) \leftarrow \mathsf{KZG.Com}(\mathsf{ck}, f \in \mathbb{Z}_p[X])$.
  3. Compute $z_0 := \mathsf{H}(\mathsf{com}_{\mathsf{KZG}})$ and set $y_0 := f(z_0)$.
  4. Run $\tau_0 \leftarrow \mathsf{KZG.Open}(\mathsf{ck}_{\mathsf{KZG}}, St, z_0)$.
  5. Return $\mathsf{com} := (\mathsf{com}_{\mathsf{KZG}}, y_0, \tau_0)$ and St.
- $\mathsf{VC}_{\mathsf{KZG}}.\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) \rightarrow b$
  1. Parse $\mathsf{com} = (\mathsf{com}_{\mathsf{KZG}}, y_0, \tau_0)$ and set $z_0 := \mathsf{H}(\mathsf{com}_{\mathsf{KZG}})$.
  2. Return $\mathsf{KZG.Ver}(\mathsf{ck}_{\mathsf{KZG}}, \mathsf{com}_{\mathsf{KZG}}, z_0, y_0, \tau_0)$.

- $\mathsf{VC_{KZG}.Open}(\mathsf{ck}, St, i) \to \tau$
  1. Return $\mathsf{KZG.Open}(\mathsf{ck_{KZG}}, St, \iota(i))$.
- $\mathsf{Aggregate}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, (\tau_j)_{j=1}^L) \to \tau$
  1. For each $j \in [L]$, parse $\tau_j = (\hat{y}_j, v_j) \in \mathbb{Z}_p \times \mathbb{G}_1$.
  2. Set $\xi := \mathsf{H}'(i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L)$.
  3. Set $\hat{y} := \sum_{j=1}^L \xi^{j-1}\hat{y}_j$ and $v := \prod_{j=1}^L v_j^{\xi^{j-1}}$.
  4. Return $\tau = (\hat{y}, v)$.
- $\mathsf{VC_{KZG}.Ver}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau) \to b$
  1. For each $j \in [L]$, parse $\mathsf{com}_j = (\mathsf{com}_{\mathsf{KZG},j}, y_{0,j}, \tau_{0,j}) \in \mathbb{G}_1 \times \mathbb{Z}_p \times \mathcal{T}$.
  2. Set $\xi := \mathsf{H}'(i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L)$.
  3. Compute $m := \sum_{j=1}^L \xi^{j-1}m_j$ and $\mathsf{com} := \prod_{j=1}^L \mathsf{com}_{\mathsf{KZG},j}^{\xi^{j-1}}$.
  4. Return $\mathsf{KZG.Ver}(\mathsf{ck_{KZG}}, \mathsf{com}, \iota(i), m, \tau)$.

In the following, we show that $\mathsf{VC_{KZG}}$ satisfies equivocality, aggregation position-binding, and augmented extractability. Simulation-extractability then follows from Lemma 1.

Lemma 2 (Informal). *Let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$ be a random oracle. Then, $\mathsf{VC_{KZG}}$ is equivocal.*

We provide the formal statement and proof in our full version [21]. Intuitively, the simulator sets $\mathsf{com_{KZG}}$ to be a random group element, samples the polynomials f and $\hat{f}$ in a lazy way, and computes openings on the fly using knowledge of the trapdoor α and the equation $v = \left(\mathsf{com_{KZG}} \cdot g_1^{-y} h_1^{-\hat{y}}\right)^{\frac{1}{\alpha - z}}$. To make the formal argument work, we need to carry out the changes in the correct order and pay attention to the degrees of the polynomials.

Lemma 3. *If the d-DLOG assumption holds relative to PGGen and $\mathsf{H}'\colon \{0,1\}^* \to \mathbb{Z}_p$ is modeled as a random oracle, then $\mathsf{VC_{KZG}}$ is aggregation position-binding in the algebraic group model. Concretely, for any polynomial $\ell \in \mathbb{N}$ and any algebraic PPT algorithm $\mathcal{A}$ that makes at most $Q_{\mathsf{H}'}$ queries to random oracle H', there are PPT algorithms $\mathcal{B}_1, \mathcal{B}_2$ with $\mathbf{T}(\mathcal{B}_1) \approx \mathbf{T}(\mathcal{B}_2) \approx \mathbf{T}(\mathcal{A})$ and*

$$\mathsf{Adv}^{\mathsf{a\text{-}pos\text{-}bind}}_{\mathcal{A},\mathsf{VC_{KZG}},\ell}(\lambda) \le 2 \cdot \mathsf{Adv}^{1\text{-}\mathsf{DLOG}}_{\mathcal{B}_1,\mathsf{PGGen}}(\lambda) + 2 \cdot \mathsf{Adv}^{(\ell+1)\text{-}\mathsf{DLOG}}_{\mathcal{B}_2,\mathsf{PGGen}}(\lambda) + \frac{Q_{\mathsf{H}'} L_{max}}{p},$$

Proof. We first recall the aggregation position-binding game for an algebraic adversary $\mathcal{A}$ and a dimension ℓ to fix some notation. Set $d := \ell + 1$ as in the scheme. First, a commitment key $\mathsf{ck} = (\mathsf{ck_{KZG}}, z_{\mathsf{out}}, \iota)$ is generated, where ι is a mapping from $[\ell]$ to $\mathbb{Z}_p$ and $\mathsf{ck_{KZG}} = (\mathsf{par}, h_1, R, (u_i)_{i=0}^d, (\hat{u}_i)_{i=0}^d)$ is a commitment key for the KZG polynomial commitment, with group parameters $\mathsf{par} = (\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, p, e)$. That is, $h_1 = g_1^\beta$ for some $\beta \in \mathbb{Z}_p$, and there is some $\alpha \in \mathbb{Z}_p$ such that $u_i = g_1^{\alpha^i}$ and $\hat{u}_i = h_1^{\alpha^i}$ for each $i \in \{0, \dots, d\}$. Further, $R = g_2^\alpha$. Then, when $\mathcal{A}$ terminates, it outputs the following:

– A message $\mathbf{m} \in \mathbb{Z}_p^\ell$ and randomness φ. Here φ has the form $\varphi = (\delta_0, \delta_1, \hat{f}') \in \mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}_p[X]$, where $\hat{f}'$ is of degree ℓ. Based on this output, the aggregation position-binding game honestly computes a commitment com. More concretely, let $f' \in \mathbb{Z}_p[X]$ be the polynomial of degree $d = \ell + 1$ with $f'(0) = \delta_0$, $f'(z_{\mathsf{out}}) = \delta_1$, and $f'(\iota(i)) = \mathbf{m}_i$ for every $i \in [\ell]$. Then, the commitment com has the form $\mathsf{com} = (\mathsf{com}_{\mathsf{KZG}}, y_0, \tau_0)$, where $\mathsf{com}_{\mathsf{KZG}} = g_1^{f'(\alpha)} h_1^{\hat{f}'(\alpha)}$.
– An index $i^* \in [\ell]$. We will denote $z := \iota(i^*)$. Further, we will denote by ψ', $\hat{\psi}' \in \mathbb{Z}_p[X]$ the polynomials

$$\psi' := \frac{f' - f'(z)}{X - z}, \quad \hat{\psi}' := \frac{\hat{f}' - \hat{f}'(z)}{X - z}$$

as in an honest KZG opening for f' at position z. The game can efficiently compute these polynomials.
– Lists $(\mathsf{com}_j)_{j=1}^L$ and $(m_j)_{j=1}^L$, and an opening $\tau = (\hat{y}, v) \in \mathbb{Z}_p \times \mathbb{G}_1$. Concretely, each com_j has the form $\mathsf{com}_j = (\mathsf{com}_{\mathsf{KZG},j}, y_{0,j}, \tau_{0,j})$, where $\mathsf{com}_{\mathsf{KZG},j} \in \mathbb{G}_1$. As $\mathcal{A}$ is algebraic, it also outputs an algebraic representation for each $\mathsf{com}_{\mathsf{KZG},j}$ and for τ. Due to the structure of the commitment key, this is equivalent to saying that $\mathcal{A}$ outputs polynomials $f_j, \hat{f}_j \in \mathbb{Z}_p[X]$ and $\psi, \hat{\psi} \in \mathbb{Z}_p[X]$ all of degree at most $d = \ell + 1$ such that

$$\tau = g_1^{\psi(\alpha)} \cdot h_1^{\hat{\psi}(\alpha)} \wedge \forall j \in [L] : \mathsf{com}_{\mathsf{KZG},j} = g_1^{f_j(\alpha)} \cdot h_1^{\hat{f}_j(\alpha)}.$$

We denote the event that $\mathcal{A}$ breaks aggregation position-binding by Win. Assuming that Win occurs, we know the following: There is an index $j^* \in [L]$ such that $\mathsf{com}_{j^*} = \mathsf{com}$ and $m_{j^*} \neq \mathbf{m}_{i^*}$. In particular, this means that $\mathsf{com}_{\mathsf{KZG}} = \mathsf{com}_{\mathsf{KZG},j^*}$ and $m_{j^*} \neq f'(z)$. We have $\mathsf{VC}_{\mathsf{KZG}}.\mathsf{Ver}(\mathsf{ck}, i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L, \tau) = 1$. In particular, by reading the verification equation in the exponent, we have

$$\sum_{j=1}^{L} \xi^{j-1}(f_j(\alpha) + \beta \hat{f}_j(\alpha) - m_j) - \beta \hat{y} = (\psi(\alpha) + \beta \hat{\psi}(\alpha))(\alpha - z)$$

for $\xi := \mathsf{H}'(i, (\mathsf{com}_j)_{j=1}^L, (m_j)_{j=1}^L)$. Defining polynomials $f := \sum_{j=1}^L f_j \xi^{j-1} \in \mathbb{Z}_p[X]$ and $\hat{f} := \sum_{j=1}^L \hat{f}_j \xi^{j-1} \in \mathbb{Z}_p[X]$, and the element $m := \sum_{j=1}^L m_j \xi^{j-1}$ simplifies this equation to

$$f(\alpha) + \beta \hat{f}(\alpha) - m - \beta \hat{y} = (\psi(\alpha) + \beta \hat{\psi}(\alpha))(\alpha - z).$$

By construction of $f', \hat{f}'$ and $\psi', \hat{\psi}'$, we also have

$$f'(\alpha) + \beta \hat{f}'(\alpha) - f'(z) - \beta \hat{f}'(z) = (\psi'(\alpha) + \beta \hat{\psi}'(\alpha))(\alpha - z).$$

Subtracting the two equations, we get our *core equation*, namely,

$$\begin{aligned} & f(\alpha) - f'(\alpha) + \beta(\hat{f}(\alpha) - \hat{f}'(\alpha)) - (m - f'(z)) - \beta(\hat{y} - \hat{f}'(z)) \\ = \; & (\psi(\alpha) - \psi'(\alpha) + \beta(\hat{\psi}(\alpha) - \hat{\psi}'(\alpha)))(\alpha - z). \end{aligned}$$

Our goal will be to simplify the structure of this core equation. To do so, our first step is to eliminate the terms containing β. We define the following event:

– Event $\mathsf{Complex}$: This event occurs, if $\eta := \hat{f}(\alpha) - \hat{f}'(\alpha) - (\hat{y} - \hat{f}'(z)) - (\hat{\psi}(\alpha) - \hat{\psi}'(\alpha))(\alpha - z) \neq 0$.

Claim. There is a PPT algorithm $\mathcal{B}$ with $\Pr[\mathsf{Win} \wedge \mathsf{Complex}] \leq \mathsf{Adv}^{\mathsf{1\text{-}DLOG}}_{\mathcal{B},\mathsf{PGGen}}(\lambda)$.

Proof of Claim. Reduction $\mathcal{B}$ is as follows. It gets as input the group parameters, the generator g_1 and the element $h_1 = g_1^{\beta}$. We show that it can simulate the game for $\mathcal{A}$ and compute β if $\mathsf{Win} \wedge \mathsf{Complex}$ occurs. For that, $\mathcal{B}$ first samples $\alpha \leftarrow_{\$} \mathbb{Z}_p$ and computes the commitment key ck as in algorithm Setup. Observe that for that, β is not needed. Now, if Win occurs, then we know that the core equation holds. For convenience, we group together the β-terms in our core equation, getting

$$\beta \cdot \eta = f'(\alpha) - f(\alpha) + (m - f'(z)) + (\psi(\alpha) - \psi'(\alpha))(\alpha - z).$$

Clearly, if $\mathsf{Complex}$, then reduction $\mathcal{B}$ can compute β by multiplying the right hand-side with η^{-1}.

From now on, we condition on $\neg\mathsf{Complex}$. In other words, we assume that $\eta = 0$, which implies that the *simplified core equation*

$$f(\alpha) - m - (f'(\alpha) - f'(z)) = (\psi(\alpha) - \psi'(\alpha))(\alpha - z)$$

holds. Our goal will be to show that if this equation holds, we can build a reduction breaking the d-DLOG assumption. For that, we define the following events:

– Event NoColl : This event occurs, if $f'(\alpha) \neq f_{j^*}(\alpha)$.
– Event Ambig : This event occurs, if $f' \neq f_{j^*}$ over $\mathbb{Z}_p[X]$.
– Event $\mathsf{AggFail}$: This event occurs, if $m = f(z)$.

In the next claims, we bound the probability that one of these event occurs. Informally, if NoColl occurs, then one can use $\mathsf{com}_{\mathsf{KZG}} = \mathsf{com}_{\mathsf{KZG},j^*}$ solve for β to break DLOG. If Ambig occurs but NoColl does not, then we can find α efficiently as a root of the non-zero polynomial $f' - f_{j^*}$. We will bound the case that $\mathsf{AggFail}$ occurs using a statistical argument using the fact that ξ is chosen after the f_j and m_j are fixed.

Claim. There is a PPT algorithm $\mathcal{B}$ with $\Pr[\mathsf{Win} \wedge \mathsf{NoColl}] \leq \mathsf{Adv}^{\mathsf{1\text{-}DLOG}}_{\mathcal{B},\mathsf{PGGen}}(\lambda)$.

Proof of Claim. Note that if Win occurs, then we have $f'(\alpha) + \beta\hat{f}'(\alpha) = f_{j^*}(\alpha) + \beta\hat{f}_{j^*}(\alpha)$, because $\mathsf{com}_{\mathsf{KZG}} = \mathsf{com}_{\mathsf{KZG},j^*}$. If NoColl occurs at the same time, then we know that $\hat{f}'(\alpha) - \hat{f}_{j^*}(\alpha) \neq 0$ and one can efficiently solve for β. It is not hard to turn that into a formal reduction that determines β given $h_1 = g_1^{\beta}$.

Claim. There is a PPT algorithm $\mathcal{B}$ with $\Pr[\mathsf{Ambig} \wedge \neg\mathsf{NoColl}] \leq \mathsf{Adv}^{d\text{-}\mathsf{DLOG}}_{\mathcal{B},\mathsf{PGGen}}(\lambda)$.

Proof of Claim. Note that if $\neg\mathsf{NoColl}$ occurs, then we have $f'(\alpha) \neq f_{j^*}(\alpha)$. If Ambig occurs at the same time, we know that α is a root of the non-zero polynomial $f' - f_{j^*}$, which has degree at most $d = \ell + 1$. Hence, α can be found in polynomial time by a reduction. We leave details to the reader.

Claim. We have $\Pr[\mathsf{Win} \wedge \mathsf{AggFail} \wedge \neg\mathsf{Ambig}] \leq Q_{\mathsf{H}'} L_{max}/p$.

Proof of Claim. By definition of m and f, event $\mathsf{AggFail}$ is equivalent to the equation

$$\sum_{j=1}^{L} m_j \xi^{j-1} = \sum_{j=1}^{L} f_j(z)\xi^{j-1}.$$

In other words, if $\mathsf{AggFail}$ occurs, then the polynomial

$$\zeta = \sum_{j=1}^{L} (f_j(z) - m_j) X^{j-1} \in \mathbb{Z}_p[X]$$

has a root at ξ. Observe that ζ has degree $L \leq L_{max}$ and is fixed before[3] ξ is chosen at random by the random oracle H'. Thus, for any fixed random oracle query where $\zeta \neq 0$, this event occurs with probability at most L_{max}/p. It remains to argue that $\zeta \neq 0$ if Win occurs and Ambig does not. This can be seen by observing that the j^*th coefficient of ζ is non-zero, i.e., $m_{j^*} \neq f_{j^*}(z)$. Namely, we know that $f' = f_{j^*}$ due to $\neg\mathsf{Ambig}$. Thus, if we had $m_{j^*} = f_{j^*}(z)$, then we had $m_{j^*} = f'(z)$, contradicting Win.

Concluding the Proof. To conclude the proof, we can now assume that Win occurs, but neither of the events defined above occurs. We come back to our simplified core equation. The equation tells us that α is a root of the polynomial Ψ of degree at most $d = \ell + 1$, which is given as

$$\Psi = f - m - (f' - f'(z)) - (\psi - \psi')(X - z) \in \mathbb{Z}_p[X].$$

Now, if we can argue that Ψ is non-zero, then one can efficiently find α based on $\mathcal{A}$'s output, leading to our final reduction. To argue that Ψ is non-zero, note that $\Psi = 0$ implies that

$$f - m = (f' - f'(z)) + (\psi - \psi')(X - z).$$

While the right hand-side is a multiple of $X - z$, the left hand-side is not, as we assume $\neg\mathsf{AggFail}$. Therefore, this equality can not hold, which means that Ψ is non-zero. In combination, setting $\mathsf{Bad} := \mathsf{NoColl} \vee \mathsf{Ambig} \vee \mathsf{AggFail} \vee \mathsf{Complex}$ we get a reduction $\mathcal{B}$ with

$$\Pr[\mathsf{Win} \wedge \neg\mathsf{Bad}] \leq \mathsf{Adv}^{d\text{-}\mathsf{DLOG}}_{\mathcal{B},\mathsf{PGGen}}(\lambda).$$

Lemma 4 (Informal). *If d-DLOG holds and $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$ is a random oracle, then $\mathsf{VC}_{\mathsf{KZG}}$ satisfies augmented extractability in the algebraic group model.*

[3] It could be that the adversary submitted a different algebraic representation to the random oracle. In this case, we just define the f_j's to be this representation.

We provide the formal statement and proof in our full version [21]. Here, we first recall the augmented extractability game to fix notation and define our extractor Ext. Then, we provide a proof intuition.

Game, Extractor and Notation. In the augmented extractability game, the adversary $\mathcal{A}$ gets as input a commitment key ck and access to a commitment oracle GetCom and an opening oracle GetOp. These are as follows:

- The commitment key ck has the form $\mathsf{ck} = (\mathsf{ck}_{\mathsf{KZG}}, z_{\mathsf{out}}, \iota)$ where $\iota \colon [\ell] \to \mathbb{Z}_p$ is injective and $\mathsf{ck}_{\mathsf{KZG}} = (\mathsf{par}, h_1, R, (u_i)_{i=0}^d, (\hat{u}_i)_{i=0}^d)$ is a KZG commitment key with group parameters $\mathsf{par} = (\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, p, e)$. We have $h_1 = g_1^{\beta}$ for some random $\beta \in \mathbb{Z}_p$, and $u_i = g_1^{\alpha^i}$ and $\hat{u}_i = h_1^{\alpha^i}$ for each $i \in \{0, \ldots, d\}$ for some random $\alpha \in \mathbb{Z}_p$. We also have $R = g_2^{\alpha}$.
- On input $\mathbf{m} \in \mathbb{Z}_p^{\ell}$, the commitment oracle returns a commitment com for $\mathbf{m}$. We use the subscript k to refer to the kth commitment returned by the oracle. That is, $\mathsf{com}_k = (\mathsf{com}_{\mathsf{KZG},k}, f_k(z_{k,0}), \tau_{k,0})$ is the kth commitment returned by the oracle, where $\mathsf{com}_{\mathsf{KZG},k} = g_1^{f_k(\alpha)} h_1^{\hat{f}_k(\alpha)}$ is a KZG commitment to a polynomial f_k of degree d with randomness $\hat{f}_k$, and $\tau_{k,0} = (\hat{f}_k(z_{k,0}), v_{k,0})$ is a KZG opening for f_k at position $z_{k,0} = \mathsf{H}(\mathsf{com}_{\mathsf{KZG},k})$ to value $f_k(z_{k,0})$. We denote the number of queries to GetCom by Q_C and assume without loss of generality that $Q_C \geq 1$.
- On input (k, i) such that com_k is defined, the opening oracle GetOp opens com_k at position i. To recall, such an opening is a KZG openings for commitment $\mathsf{com}_{\mathsf{KZG},k}$ at position $z_{k,i} := \iota(i)$. We denote the opening by $\tau_{k,i} = (\hat{f}_k(z_{k,i}), v_{k,i})$ for $v_{k,i} = g_1^{\psi_{k,i}(\alpha)} h_1^{\hat{\psi}_{k,i}(\alpha)}$, where $\psi_{k,i} = (f_k - f_k(z_{k,i}))/(X - z_{k,i}) \in \mathbb{Z}_p[X]$ and $\hat{\psi}_{k,i} := (\hat{f}_k - \hat{f}_k(z_{k,i}))/(X - z_{k,i})$. Without loss of generality, we can assume that for every commitment com_k returned by the commitment oracle, $\mathcal{A}$ queries the opening oracle for every $i \in [\ell]$, and thus it obtained all of these openings $\tau_{k,i}$ for $i \in \{0, \ldots, \ell\}$.

When $\mathcal{A}$ terminates, it outputs a commitment outputs $\mathsf{com} = (\mathsf{com}_{\mathsf{KZG}}, y_0, \tau_0)$ with $\tau_0 = (\hat{y}_0, v_0)$. As $\mathcal{A}$ is algebraic, it also outputs the algebraic representation of all group elements in com. Due to the structure of ck and the group elements that $\mathcal{A}$ obtained from the commitment and opening oracles, we can assume that this representation is given by polynomials $f, \hat{f}, \psi, \hat{\psi} \in \mathbb{Z}_p[X]$ of degree $d = \ell + 1$ and lists of exponents $(w_k)_k, (r_k)_k$ and $(t_{k,i})_{k,i}, (s_{k,i})_{k,i}$ over $\mathbb{Z}_p$ such that

$$\mathsf{com}_{\mathsf{KZG}} = g_1^{f(\alpha)} \cdot h_1^{\hat{f}(\alpha)} \cdot \underbrace{\prod_{k=1}^{Q_C} \mathsf{com}_{\mathsf{KZG},k}^{w_k} \cdot \prod_{k=1}^{Q_C} \prod_{i=0}^{\ell} v_{k,i}^{t_{k,i}}}_{=:L},$$

$$v_0 = g_1^{\psi(\alpha)} \cdot h_1^{\hat{\psi}(\alpha)} \cdot \prod_{k=1}^{Q_C} \mathsf{com}_{\mathsf{KZG},k}^{r_k} \cdot \prod_{k=1}^{Q_C} \prod_{i=0}^{\ell} v_{k,i}^{s_{k,i}}.$$

Without loss of generality, we assume that $\mathcal{A}$ queried $\mathsf{H}(\mathsf{com}_{\mathsf{KZG}})$, and it did so with the same algebraic representation for $\mathsf{com}_{\mathsf{KZG}}$ as the one that it outputs in the end. Given the output of the adversary, the extractor returns the message $\mathbf{m} \in \mathbb{Z}_p^\ell$ defined by $\mathbf{m}_i := f(\iota(i))$ for all $i \in [\ell]$ and the randomness $\varphi := (\delta_0, \delta_1, \hat{f})$, where $\delta_0 := f(0)$ and $\delta_1 := f(z_{\mathsf{out}})$. Now, $\mathcal{A}$ wins the game, if the following three properties hold:

- The commitment com is fresh, i.e., it was not output by the commitment oracle GetCom.
- Committing to $\mathbf{m}$ with randomness φ does not yield com. It is easy to see that this can only happen if $L \neq g_1^0$.
- The commitment com verifies, i.e., $\mathsf{VC}_{\mathsf{KZG}}.\mathsf{VerCom}(\mathsf{ck}, \mathsf{com}) = 1$. This is equivalent to saying that τ_0 is a valid KZG opening for $\mathsf{com}_{\mathsf{KZG}}$ at position $z_0 = \mathsf{H}(\mathsf{com}_{\mathsf{KZG}})$ to value y_0, i.e., that

$$e\left(\mathsf{com}_{\mathsf{KZG}} \cdot g_1^{-y_0} \cdot h_1^{-\hat{y_0}}, g_2\right) = e\left(v_0, g_2^\alpha \cdot g_2^{-z_0}\right).$$

Proof Strategy. In a preparatory phase (see $\mathbf{G}_0$ to $\mathbf{G}_3$), we rule out some simple bad events and simplify some equations. Namely, we rule out that there are collisions among the z's, e.g., that $z_0 = z_{k,i}$ for some i and k. We also rule out that α is equal to one of those z's. Further, we ensure that not only com is fresh, but instead $\mathsf{com}_{\mathsf{KZG}}$ is fresh. For that, we need to rule out that the adversary reuses a $\mathsf{com}_{\mathsf{KZG},k}$ with a different opening. We also expand the verification equation using the algebraic representation, and simplify it by ensuring that the exponent of h_1 is zero. Indeed, if this were not the case, one could compute the discrete logarithm β of $h_1 = g_1^\beta$. After this preparatory phase, our main argument follows (see $\mathbf{G}_4$ to $\mathbf{G}_7$). Namely, we show that from the adversary's output, a reduction can efficiently compute $f_{k^*}(z_0)$ for some k^*, while it only provided the $\ell + 1 = d$ evaluations $f_{k^*}(z_{k^*,i})$ for $i \in \{0, \ldots, \ell\}$ to the adversary. With this additional evaluation point $f_{k^*}(z_0)$, the reduction can compute f_{k^*} entirely. Roughly, this observation can be used as follows: The reduction guesses k^*, interprets a DLOG instance $X^* = g_1^{x^*}$ as $g_1^{f_{k^*}(\alpha)}$, and embeds it into the commitment $\mathsf{com}_{\mathsf{KZG},k^*}$. With the output of the adversary, it can efficiently recover f_{k^*} and therefore the discrete logarithm $x^* = f_{k^*}(\alpha)$. The details of the proof can be found in our full version [21].

4 Aggregatable Lotteries

In this section, we discuss how our lotteries are defined and how they can be constructed from our notion of vector commitments. Due to space constraints, we defer the formal description of our protocol as well as all corresponding security proofs to our full version [21].

4.1 Definition of Aggregatable Lotteries

We formally present our ideal functionality $\mathcal{F}_{\text{lottery}}(p, T)$ for *non-interactive aggregatable lotteries* in Figs. 5 and 6. It is parameterized by an upper bound on the winning probability p and the number of lotteries T. The probability p specifies how likely it is that a party wins in a lottery round. As described previously, our lotteries should intuitively prevent an adversary from winning the lotteries a disproportionate amount of times and the adversary should also not be able to tell which honest parties win the lotteries when. We do, however, allow the adversary to reduce its winning probability, i.e., the adversary can misbehave in a way that makes them win the lottery less often. We model this by allowing the adversary to specify their own winning probabilities for each lottery, capped at p, upon registration.

Our ideal functionality also relies on a party called the $\mathsf{Croupier}$, which we did not mention so far. It is in charge of initiating lottery rounds and registering participants. Note that this model allows the adversary to corrupt $\mathsf{Croupier}$, meaning that security is guaranteed even when the adversary can arbitrarily control the lottery, i.e., by registering players or initiating new lotteries.

Parties can be registered by $\mathsf{Croupier}$ via the Register interface. A lottery execution is run by $\mathsf{Croupier}$ via the NextLottery interface. Upon calling this interface, the functionality flips a biased coin for every currently registered party and stores whether they won the currently executed lottery. Parties can call the Participate interface to see whether they won a specific lottery. If they did, then they obtain a lottery ticket label, otherwise they receive nothing. Parties can call the Aggregate interface with a set of winning ticket labels and party identifiers to obtain an aggregate ticket label. Lastly, the Verify interface takes an aggregate ticket label and the corresponding winning parties' identifiers as input and checks whether the ticket is valid.

4.2 Our Construction

Let us now proceed with our construction of aggregatable lotteries from vector commitments. For that, let $\mathsf{VC} = (\mathsf{VC.Setup}, \mathsf{VC.Com}, \mathsf{VC.VerCom}, \mathsf{VC.Open}, \mathsf{VC.Aggregate}, \mathsf{VC.Ver})$ be a vector commitment scheme according to Definition 3. Let $T, k \in \mathbb{N}$ be arbitrary parameters. We construct a T-time aggregatable lottery with winning probability $p = 1/k$ using a random oracle $\mathsf{H}\colon \{0,1\}^* \to [k]$. The main idea is that each player commits to a vector $\mathbf{v} \in [k]^T$ upfront, and wins the ith lottery if and only if its *personal challenge* x is equal to $\mathbf{v}_i$. Crucially, the challenge x has to be independent for different players and over different lotteries, and should be unpredictable before the lottery seed lseed is known. Thus, we define $x := \mathsf{H}(\mathsf{pk}, \mathsf{pid}, i, \mathsf{lseed})$, where pid is the identifier of the player and pk is its public key, i.e., its commitment to $\mathbf{v}$.

Algorithms. To define our lottery protocol, we first define algorithms Setup, $\mathsf{Gen}, \mathsf{VerKey}, \mathsf{Participate}, \mathsf{Aggregate}, \mathsf{Ver}$ that will be used in our protocol:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{par}$:
  1. Run $\mathsf{ck} \leftarrow \mathsf{VC.Setup}(1^\lambda, 1^T)$. Recall that ck defines message alphabet $\mathcal{M}$, opening space $\mathcal{T}$, and commitment space $\mathcal{C}$. We assume that $[k] \subseteq \mathcal{M}$.
  2. Set and return $\mathsf{par} := \mathsf{ck}$.
- $\mathsf{Gen}(\mathsf{par}) \to (\mathsf{pk}, \mathsf{sk})$:

1. Sample $\mathbf{v} \leftarrow\!\!\$\, [k]^T$ and run $(\mathsf{com}, St) \leftarrow \mathsf{VC.Com}(\mathsf{ck}, \mathbf{v})$.
2. Set and return $\mathsf{pk} := \mathsf{com}$ and $\mathsf{sk} := (\mathbf{v}, St)$.

– $\mathsf{VerKey}(\mathsf{pk}) \to b$
  1. Return $b := \mathsf{VC.VerCom}(\mathsf{ck}, \mathsf{pk})$.

– $\mathsf{Participate}(t, \mathsf{lseed}, \mathsf{pid}, \mathsf{sk}) \to \mathsf{ticket}/\perp$:
  1. Set $x := \mathsf{H}(\mathsf{pk}, \mathsf{pid}, t, \mathsf{lseed})$. If $\mathbf{v}_i \neq x$, return $\perp$.
  2. Otherwise, set $\tau \leftarrow \mathsf{VC.Open}(\mathsf{ck}, St, i)$ and return $\mathsf{ticket} := \tau$.

– $\mathsf{Aggregate}(t, \mathsf{lseed}, (\mathsf{pid}_j, \mathsf{pk}_j)_{j=1}^L, (\mathsf{ticket}_j)_{j=1}^L) \to \mathsf{agg}$:
  1. For each $j \in [L]$, write $\mathsf{pk}_j = \mathsf{com}_j$ and $\mathsf{ticket}_j = \tau_j$.
  2. For each $j \in [L]$, set $x_j := \mathsf{H}(\mathsf{pk}_j, \mathsf{pid}_j, i, \mathsf{lseed})$.
  3. Return $\mathsf{agg} := \mathsf{VC.Aggregate}(\mathsf{ck}, t, (\mathsf{com}_j)_{j=1}^L, (x_j)_{j=1}^L, (\tau_j)_{j=1}^L)$.

– $\mathsf{Ver}(t, \mathsf{lseed}, (\mathsf{pid}_j, \mathsf{pk}_j)_{j=1}^L, \mathsf{agg} = \tau) \to b$:
  1. For each $j \in [L]$, write $\mathsf{pk}_j = \mathsf{com}_j$ and set $x_j := \mathsf{H}(\mathsf{pk}_j, \mathsf{pid}_j, t, \mathsf{lseed})$.
  2. Return $b := \mathsf{VC.Ver}(\mathsf{ck}, t, (\mathsf{com}_j)_{j=1}^L, (x_j)_{j=1}^L, \tau)$.

Protocol. We informally explain how to turn these algorithms into a protocol Π_{lottery} for aggregatable lotteries using a randomness beacon $\mathcal{F}_{\text{random}}$ (see full version [21]) and a broadcast channel $\mathcal{F}_{\text{broadcast}}$ (see full version [21]). Parties register for the lottery by running $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{par})$ and broadcasting their public key pk to other parties who verify it using VerKey. To commence the next lottery, the random beacon samples $\mathsf{lseed} \leftarrow\!\!\$\, \{0,1\}^\lambda$ and distributes it to all the parties, each party then locally computes a ticket ticket as $\mathsf{ticket} \leftarrow \mathsf{Participate}(t, \mathsf{lseed}, \mathsf{pid}, \mathsf{sk})$ where t is the index of the current lottery, to observe whether they obtained a winning ticket for the current lottery. These tickets can be verified and aggregated by any party. Given a set of tickets $(\mathsf{ticket}_j)_{j=1}^L$ a party can locally use $\mathsf{Aggregate}$ to compute an aggregated ticket agg, similarly it can locally use Ver to verify an aggregated ticket agg. The full formal description and UC security proof can be found in our full version [21].

5 Discussion and Efficiency

In the final section of our paper, we present some practical thoughts about our construction and evaluate its concrete efficiency.

5.1 Practical Considerations

In practice, one can make some natural adjustments to our lottery, which we discuss here.

Weighted Lotteries. One may assign a weight p_j to participant j (e.g., based on its stake) such that j wins independently with probability p_j. We can adjust our lottery to support this: we simply let a participant with weight $p_j = 1/k_j$ commit to vectors over the range $[k_i]$ instead of $[k]$, and let the hash function defining j's challenge $x_j = \mathsf{H}(\mathsf{pk}_j, \mathsf{pid}_j, i, \mathsf{lseed})$ map to the range $[k_i]$.

Late Registration. Consider the case of ℓ lotteries and assume that a user registers late, say after the ith and before the $(i+1)$st lottery. In the extreme case, we have $i = \ell - 1$. As written, the user would have to sample a random vector of length ℓ and commit to it, while only the coordinates from $i+1$ to ℓ would be relevant. After the ℓth lottery, the entire system restarts and the user would have to generate a new key and register again. This is wasteful. Fortunately, there are ways to deal with this: (1) the user could implicitly set the first i coordinates to 0, which makes committing much more efficient (in evaluation form, see below). A similar solution applies when the user only wants to take part in any small subset of lotteries; (2) the user could keep its key for the next lifecycle of the lottery system until after the ith lottery, where for the i'th lottery ($i' \leq i$) in the next lifecycle, it would use the i'th coordinate.

High Entropy. Our proof only relies on the fact that the lottery seed output by the randomness beacon has high entropy. It is actually not necessary that it is uniformly random.

Evaluation Form. When KZG [30] is used as a vector commitment (as in our case), we should avoid explicitly computing the interpolating polynomial. Instead, we can compute the KZG commitments and openings more efficiently in the Lagrange basis. For that, we need to assume that the KZG setup contains elements $g_1^{\lambda_i(\alpha)}$, where λ_i is the ith Lagrange polynomial with respect to our evaluation domain. Note that this can be publicly pre-computed from a standard KZG setup. Optimal for efficiency is the case where we choose our evaluation domain to be the group of roots of unity and the polynomials we work with have degree d such that $d+1$ is a power of two, meaning that $\ell + 2$ has to be a power of two. In this case, we can benefit from several tricks to compute commitments and openings efficiently [19]. We emphasize that this changes the way we compute commitments and openings, but not their final value, meaning that this has no negative impact on security.

Pre-computing Openings. Note that for participants of a lottery, the most time-critical part is not key generation, but rather the time it takes to participate and compute winning tickets (algorithm Participate). In our scheme, a winning ticket is just a KZG opening proof within our evaluation domain. While computing a single proof takes linear time in ℓ (the number of lotteries), we can instead pre-compute all KZG opening proofs right after key generation and before lotteries take place. This can be done efficiently [20].

5.2 Efficiency Evaluation

With these considerations in mind, we evaluate the efficiency of our aggregatable lottery scheme Jackpot. We focus on communication/storage and computation costs.

Lottery Schemes. We have implemented the following lottery schemes in Rust using the arkworks[4] framework. VRF-BLS: The VRF-based lottery [17,26]

[4] http://arkworks.rs.

instantiated with BLS signatures [10] over curve BLS12-381 and SHA-256; more precisely, a party with secret key sk wins in lottery i with seed lseed if $\mathsf{H}(\sigma) < t$ for appropriate $t = t(k)$, where σ is a BLS signature of i and lseed and H is a hash function; σ is the winning ticket; To verify L tickets, we use BLS batch verification and L individual hash operations; Note that batch verification is only possible if all parties sign the same message, which is why can not include the party's identifier in the signed message. This also means that two parties with the same public key do not win independently, which has to be taken care of by other means. Jackpot: Our lottery scheme, using curve BLS12-381 to implement KZG; we follow the practical considerations discussed above; we have also implemented the technique from [20] to optionally pre-compute all openings. For all of our benchmarks, we assume $k = 512$ and $\mathsf{lseed} \in \{0,1\}^{256}$. Our code is available at

https://github.com/b-wagn/jackpot.

Bandwidth and Memory Consumption. Public keys for Jackpot have size $2 \cdot 48 + 2 \cdot 32 = 160$ Bytes, whereas they have size 48 Bytes for VRF-BLS. Now, assume that L winning tickets have to be stored or communicated. For VRF-BLS, this requires a storage of (ignoring public keys and identifiers of winning parties) $48 \cdot L$ Bytes, whereas for Jackpot each ticket has size $48 + 32 = 80$ Bytes, but the L tickets can be aggregated into one. This means that for L tickets, the relative improvement of Jackpot in comparison to VRF-BLS is $48 \cdot L/80 = 0.6 \cdot L$, which is a significant improvement even for small L. Table 1 shows some example numbers.

Table 1. Comparison of the memory consumption for storing or communicating L winning tickets for Jackpot with VRF-BLS. Memory is given in Bytes, and the column "Ratio" describes the ratio between the two schemes. We do not count player identifiers and their public keys, as they have to be stored on registration independent of winning tickets.

Tickets L	VRF-BLS [B]	Jackpot [B]	Ratio VRF-BLS/Jackpot
1	48	80	0.6
16	768	80	9.6
256	12288	80	153.6
1024	49152	80	614.4
2048	98304	80	1228.8

System Setup. For the following benchmarks, we have used a Macbook Pro (2020) with an Apple M1 processor, 16 GB of RAM, and MacOS Ventura 13.4. We benchmark our Rust implementation using the Criterion benchmark crate[5].

[5] https://github.com/bheisler/criterion.rs.

Aggregation and Verification. As our first benchmark in terms of running time, we evaluate the running times of aggregation (for Jackpot) and verification (for Jackpot and VRF-BLS) for different numbers L of tickets in Table 2. The running time is independent of the total number of lotteries. For VRF-BLS, we use BLS batch verification to verify multiple tickets with only one pairing equation. In this way, both schemes require L many hash evaluations and one pairing equation. Remarkably, our results demonstrate a increase in efficiency by a factor of 2 for Jackpot in comparison to VRF-BLS. This advantage arises from the fact that BLS batch verification necessitates operations over $\mathbb{G}_2$, whereas Jackpot's verification exclusively relies on operations over $\mathbb{G}_1$.

Key Generation and Pre-computation. In Table 3, we evaluate the parts of our scheme Jackpot for which the running time and memory depend on the total number of lotteries ℓ. For VRF-BLS, the running time is independent of ℓ and there is no preprocessing, and thus VRF-BLS is not listed in this table. We focus on three parameter settings $\ell \approx 2^z$ for $z \in \{10, 15, 20\}$. The table also shows the lifetime of keys for such settings, assuming one lottery every 5 minutes. In this case, 2^{20} lotteries are sufficient for 10 years. For these three parameter sets, we evaluate the running time of key generation (algorithm Gen) and the pre-computation of all KZG openings (algorithm Precompute). The table also shows the memory consumption for storing the result of the pre-computation.

Table 2. Benchmarked running times for aggregation and verification of L tickets for Jackpot and VRF-BLS. All times are given in milliseconds.

		$L = 1$	$L = 16$	$L = 256$	$L = 1024$	$L = 2048$
Jackpot	Aggregate [ms]	0.038	0.390	2.377	6.899	14.242
Jackpot	Ver [ms]	1.413	1.959	3.948	8.875	15.422
VRF-BLS	Ver [ms]	1.663	2.990	7.959	19.010	33.838

Table 3. Benchmark results of running times for our lottery scheme Jackpot for key generation (Gen), pre-computing all openings (Precompute), and participating (Participate and ComputeTicket) in a lottery for different numbers of lotteries $\ell = 2^z - 2$, $z \in \{10, 15, 20\}$. Lifetimes are estimated by assuming one lottery every 5 minutes.

Number of Lotteries	$\ell \approx 2^{10}$	$\ell \approx 2^{15}$	$\ell \approx 2^{20}$
Lifetime	3.5 days	4 months	10 years
Time Gen	14.83 ms	317.82 ms	8.27 s
Time Precompute	2.20 s	65.45 s	45 min
Memory Precompute	147 KB	5 MB	151 MB
Time Participate	1.26 μs	1.27 μs	1.31 μs
Time ComputeTicket	9.45 ms	215.80 ms	6.36 s

Functionality $\mathcal{F}_{\text{lottery}}(p, T)$

The functionality for T lotteries with winning probability p interacts with parties Player_j and a dedicated party $\mathsf{Player}_{\mathsf{Croupier}}$. It also interacts with the ideal world adversary Sim.

Interface Initialize:

$\mathsf{Tickets} := \emptyset, \quad \mathsf{AggTickets} := \emptyset$
$\mathsf{Win} := \emptyset, \quad \mathsf{Registered} := \emptyset, \quad \mathsf{LotCnt} := 1$

Interface Register: On $(\text{Register}, j, \mathsf{pid}) \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Player}_{\mathsf{Croupier}}$

// *Check if* pid *already registered*
if $\exists(\_, \mathsf{pid}, \_) \in \mathsf{Registered}$: **return**
$\mathsf{Sim} \overset{\mathsf{send}}{\hookleftarrow} (\text{Register}, j, \mathsf{pid})$

// *Corrupted parties can win with smaller probability*
if Player_j **is corrupted:**
  $\mathbf{p} \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Player}_j$
  for $t \in [T]$: $\mathbf{p}_t := \min\{\mathbf{p}_t, p\}$
else :
  $\mathbf{p} := (p, \ldots, p) \in \{p\}^T$

// *Add to registrations*
$\mathsf{Registered} := \mathsf{Registered} \cup \{(j, \mathsf{pid}, \mathbf{p})\}$
$\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} (\text{Register}, j, \mathsf{pid})$

Interface NextLottery: On $(\text{NextLottery}) \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Player}_{\mathsf{Croupier}}$

// *Obtain a label for this lottery*
if $\mathsf{LotCnt} > T$: **return**
$l \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Sim}$

// *Sample lottery outputs for all registered parties*
for $(j, \mathsf{pid}, \mathbf{p}) \in \mathsf{Registered}$:
  $w \leftarrow \mathcal{B}(\mathbf{p}_{\mathsf{LotCnt}})$ // Sample from Bernoulli distribution
  if $w = 1$: $\mathsf{Win} := \mathsf{Win} \cup \{(l, j, \mathsf{pid})\}$
  $\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} (\text{NextLottery}, \mathsf{pid}, l, w)$

// *Advance lottery counter*
$\mathsf{LotCnt} := \mathsf{LotCnt} + 1$

Fig. 5. Ideal functionality $\mathcal{F}_{\text{lottery}}(p, T)$ for an T-time aggregatable lottery with winning probability p. The remaining interfaces are given in Fig. 6.

Even for 2^{20} lotteries, running times and memory consumption remain within practical bounds. Especially, the pre-computation's duration of approximately 45 minutes is acceptable, given that it can be run in the background at the user's convenience, and it is a one-time task for the entire lifespan of a key.

Functionality $\mathcal{F}_{\text{lottery}}(p, T)$ (continued)

Interface Participate: On $(\text{Participate}, l, \mathsf{pid}) \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Player}_j$

// *Obtain fresh ticket label from* Sim

$\mathsf{Sim} \overset{\mathsf{send}}{\hookleftarrow} (\text{Participate}, l, \mathsf{pid})$

$\mathsf{ticket} \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Sim}$

// *If player won add ticket to table*

if $(l, \mathsf{pid}, j) \in \mathsf{Win}$:

  $\mathsf{Tickets} := \mathsf{Tickets} \cup \{(l, \mathsf{pid}, \mathsf{ticket})\}$

  $\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} \mathsf{ticket}$

else $\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} \bot$

Interface Aggregate:

On $(\text{Aggregate}, l, I = (\mathsf{pid}_{j'})_{j' \in [L]}, T = (\mathsf{ticket}_{j'})_{j' \in [L]}\}) \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Player}_j$

// *Check that the tickets are winning*

for $(\mathsf{pid}, \mathsf{ticket}) \in T$:

  if $(l, \mathsf{pid}, \mathsf{ticket}) \notin \mathsf{Tickets}$:

    $\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} \bot$

    return

// *Obtain aggregated ticket label from* Sim

$\mathsf{Sim} \overset{\mathsf{send}}{\hookleftarrow} (\text{Aggregate}, l, I, T)$

$\mathsf{agg} \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Sim}$

// *Store aggregated ticket label*

$\mathsf{AggTickets} := \mathsf{AggTickets} \cup \{(l, \{\mathsf{pid} : \mathsf{pid} \in I\}, \mathsf{agg})\}$

$\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} \mathsf{agg}$

Interface Verify: $(\text{Verify}, l, \mathsf{agg}, I) \overset{\mathsf{recv}}{\hookleftarrow} \mathsf{Player}_j$

// *Only tickets generated by the functionality are valid*

$\mathsf{Sim} \overset{\mathsf{send}}{\hookleftarrow} (\text{Verify}, l, \mathsf{agg}, I)$

if $(l, I, \mathsf{agg}) \in \mathsf{AggTickets}$:

  $\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} \top$

else

  $\mathsf{Player}_j \overset{\mathsf{send}}{\hookleftarrow} \bot$

Fig. 6. Remaining interfaces of ideal functionality $\mathcal{F}_{\text{lottery}}(p, T)$ for T aggregatable lotteries with winning probability p. The other interfaces are given in Fig. 5.

Participation. To participate in a lottery round, a party determines whether it won, and it computes a winning ticket in case it did. While we combined these two tasks in our modeling (algorithm Participate), we separate them for our benchmarks (algorithms Participate and ComputeTicket, respectively). We show the results in Table 3. For all parameter sets, the running time of Participate (i.e., checking if the party won) is negligibly small, namely, within microseconds. The running time to compute a ticket scales linearly with ℓ, but even for 2^{20} lotteries it is within a practical range: waiting 7 seconds for a ticket is fine, as one can compute a ticket in our scheme even before the lottery round started. Also, assuming we did a pre-computation as discussed above, this cost is completely avoided.

References

1. Bartoletti, M., Zunino, R.: Constant-deposit multiparty lotteries on bitcoin. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y.A., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) FC 2017 Workshops. LNCS, vol. 10323, pp. 231–247. Springer, Heidelberg (Apr 2017)
2. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4_17
3. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168596
4. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1_24
5. Boneh, D., Boyen, X.: Short signatures without random oracles and the SDH assumption in bilinear groups. Journal of Cryptology **21**(2), 149–177 (Apr 2008). https://doi.org/10.1007/s00145-007-9005-7
6. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 435–464. Springer, Heidelberg (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_15
7. Boneh, D., Eskandarian, S., Hanzlik, L., Greco, N.: Single secret leader election. In: AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020. pp. 12–24. ACM (2020), https://doi.org/10.1145/3419614.3423258
8. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 565–596. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_19
9. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_26
10. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_30
11. Broder, A.Z., Dolev, D.: Flipping coins in many pockets (byzantine agreement on uniformly random values). In: 25th FOCS. pp. 157–170. IEEE Computer Society Press (Oct 1984). https://doi.org/10.1109/SFCS.1984.715912

12. Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 280–312. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78381-9_11
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
14. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 55–72. Springer, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_5
15. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 499–530. Springer, Heidelberg (Apr 2023). https://doi.org/10.1007/978-3-031-30617-4_17
16. Chow, S.S., Hui, L.C., Yiu, S.M., Chow, K.: An e-lottery scheme using verifiable random function. In: International Conference on Computational Science and its Applications. pp. 651–660. Springer (2005)
17. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 66–98. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_3
18. Faonio, A., Fiore, D., Kohlweiss, M., Russo, L., Zajac, M.: From polynomial IOP and commitments to non-malleable zksnarks. IACR Cryptol. ePrint Arch. p. 569 (2023), https://eprint.iacr.org/2023/569
19. Feist, D.: PCS multiproofs using random evaluation. https://dankradfeist.de/ethereum/2021/06/18/pcs-multiproofs.html (2021), accessed: 2023-09-28
20. Feist, D., Khovratovich, D.: Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033 (2023), https://eprint.iacr.org/2023/033
21. Fleischhacker, N., Hall-Andersen, M., Simkin, M., Wagner, B.: Jackpot: Non-interactive aggregatable lotteries. Cryptology ePrint Archive, Paper 2023/1570 (2023), https://eprint.iacr.org/2023/1570
22. Fleischhacker, N., Herold, G., Simkin, M., Zhang, Z.: Chipmunk: Better synchronized multi-signatures from lattices. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022. pp. 1109–1123. ACM (2022), https://doi.org/10.1145/3548606.3560655
23. Fleischhacker, N., Simkin, M., Zhang, Z.: Squirrel: Efficient synchronized multi-signatures from lattices. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 1109–1123. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560655
24. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_2
25. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th FOCS. pp. 40–49. IEEE Computer Society Press (Oct 2013). https://doi.org/10.1109/FOCS.2013.13
26. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium

on Operating Systems Principles, Shanghai, China, October 28-31, 2017. pp. 51–68. ACM (2017), https://doi.org/10.1145/3132747.3132757
27. Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 2007–2023. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417244
28. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 581–612. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_20
29. Itakura, K.: A public-key cryptosystem suitable for digital multisignature. NEC research and development **71**, 1–8 (1983)
30. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_11
31. Liang, B., Banegas, G., Mitrokotsa, A.: Statically aggregate verifiable random functions and application to e-lottery. Cryptography **4**(4), 37 (2020)
32. Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 1–31. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_1
33. Libert, B., Passelègue, A., Riahinia, M.: PointProofs, revisited. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part IV. LNCS, vol. 13794, pp. 220–246. Springer, Heidelberg (Dec 2022). https://doi.org/10.1007/978-3-031-22972-5_8
34. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 499–517. Springer, Heidelberg (Feb 2010). https://doi.org/10.1007/978-3-642-11799-2_30
35. Libert, B.: Simulation-extractable KZG polynomial commitments and applications to HyperPlonk. In: PKC 2024. LNCS, Springer, Heidelberg (May 2024), to appear
36. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: Extended abstract. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001. pp. 245–254. ACM Press (Nov 2001). https://doi.org/10.1145/501983.502017
37. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th FOCS. pp. 120–130. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814584
38. Papamanthou, C., Shi, E., Tamassia, R., Yi, K.: Streaming authenticated data structures. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 353–370. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_22
39. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS. pp. 543–553. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814628
40. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (Jan 1991). https://doi.org/10.1007/BF00196725
41. Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., Khovratovich, D.: Aggregatable subvector commitments for stateless cryptocurrencies. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 45–64. Springer, Heidelberg (Sep 2020). https://doi.org/10.1007/978-3-030-57990-6_3