Insta-Pok3r: Real-time Poker on Blockchain

Sanjam Garg[*] Aniket Kate[†] Pratyay Mukherjee[‡] Rohit Sinha[§] Sriram Sridhar[¶]

Abstract

We develop a distributed service for generating correlated randomness (e.g. permutations) for multiple parties, where each party's output is private but publicly verifiable. This service provides users with a low-cost way to play online poker in real-time, without a trusted party.

Our service is backed by a committee of compute providers, who run a multi-party computation (MPC) protocol to produce an (identity-based) encrypted permutation of a deck of cards, in an offline phase well ahead of when the players' identities are known. When the players join, what we call the online phase, they decrypt their designated cards immediately after deriving the identity-based decryption keys, a much simpler computation. In addition, the MPC protocol also generates a publicly-verifiable proof that the output is a permutation.

In our construction, we introduce a new notion of succinctly verifiable multi-identity based encryption (SVME), which extends the existing notion of verifiable encryption to a multi-identity-based setting, but with a constant sized proof – this may be of independent interest. We instantiate this for a permutation relation (defined over a small set) along with identity-based encryption, polynomial commitments and succinct proofs – our choices are made to enable a distributed computation when the card deck is always secret shared. Moreover, we design a new protocol to efficiently generate a secret-sharing of random permutation of a small set, which is run prior to distributed SVME.

Running these protocols offline simplifies the online phase substantially, as parties only derive their identity-specific keys privately via secure channels with the MPC committee, and then decrypt locally to obtain their decks. We provide a rigorous UC-based formalization in a highly modularized fashion.

Finally, we demonstrate practicality with an implementation that shows that for 8 MPC parties, generating a secret publicly-verifiable permutation of 64 cards takes under 3 seconds, while accessing cards for a player takes under 0.3 seconds.

1 Introduction

Can a group of players play poker over the internet, without trusting other players or any other third party? This question was first asked in the groundbreaking *Mental Poker* paper [36, 48] in 1979. Mainstream gaming platforms today wield substantial and often unchecked authority over the operation of their gaming services; unsurprisingly, reports have surfaced about the misuse of "God view" by insiders to gain a complete advantage over their customers [32, 6].

In this paper, we build a decentralized system for *privately* and *verifiably* shuffling a deck of cards — with *practical* computational, communication, and storage costs — that can be accessed by the participants almost *instantaneously*, thereby enabling online poker in real-time. We call our system Insta-Pok3r.[1]

Status Quo. Recent proposals [3, 5] have each player take turns shuffling an encrypted deck of cards, and attach a zero-knowledge proof to prove the correctness of their shuffles. While modern SNARK proof-systems, such as Plonk [8], make these operations undoubtedly far more efficient than once imagined (e.g., back in 1982 [36]), practical concerns linger. For example, each step of the shuffle incurs a separate proof, or alternatively, the proof must be made recursive in the style of incrementally verifiable computation (IVC),

[*]University of California, Berkeley sanjamg@berkeley.edu
[†]Purdue University & Supra Research aniket@purdue.edu
[‡]Supra Research pratyay85@gmail.com
[§]Swirlds Labs & zkBricks sinharo@gmail.com
[¶]University of California, Berkeley srirams@berkeley.edu
[1]This name was inspired by Game3r [2], a Web3 gaming forum.

which has prohibitive cost for real-time settings. More importantly, what is our recourse when a player has low connectivity or has simply dropped out since joining? Do other players reach an agreement on whether to skip their turn and continue with the shuffle?[2] These issues incur substantial, and often unpredictable delays between the players' arrival and the game start.

Our Approach. We observe that shuffling a card deck is an input-less functionality, and can therefore be performed ahead of time, even before (the identities of) the players are determined. Using this to our advantage, we consider a new design that shifts work away from the players onto a (distributed) service, or a committee of servers. We want the committee to produce (many) encrypted decks ahead of time, such that when the players come online (or join a table) at some later point, they can immediately access their own cards (with respect to that table or session) via decryption. Crucially, the encryptions must be publicly verifiable, as the players would like to check the validity of the deck (whether it is indeed a permutation), for which we use zero-knowledge proofs. Moreover, since the player's identities are not known ahead of time, the encryption must be identity-based, where each identity is unique to a card and a session. When players come online, they simply request a derivation of their identity-specific keys and decrypt their cards. The key idea behind Insta-Pok3r is that the relatively-expensive multiparty protocol of generating a verifiably (identity-based) encrypted permutation is executed in advance, and only the inexpensive operations (i.e., key-derivation and decryption) are executed in real-time.

Our Framework: Insta-Pok3r. Our framework consists of three sets of participants: (i) a committee of MPC servers; (ii) a committee of key servers, called keypers, each of which holds a Shamir share of the IBE master key; (iii) a set of players. The parties within a committee are connected via point-to-point authenticated channels, and also perform a light setup phase amongst themselves. Insta-Pok3r consists of distinct offline and online phases, outlined below. The offline phase is run continuously, preparing encrypted decks ahead of time, to supply the demand for poker games.

Offline Sampling. In the offline phase (before players' identities are known), the MPC servers collaborate to:

1. sample a pseudorandom permutation in a distributed manner, where each server has an additive secret share of each of the 52 cards;
2. encrypt the cards under an identity-based encryption (IBE) scheme – an identity is a tuple of the form $(\mathsf{table}, \mathsf{card})$ (e.g., card #2 on table #8);
3. generate a PLONK-style proof for the claim that the IBE ciphertext encodes a valid permutation.

The ciphertext and proof are then submitted on-chain.[3]

Online Retrieval. Later, when each player joins the table (i.e., comes online), they put their request on-chain asking for their IBE decryption key. The keypers fetch these requests, and then they: 1) compute the (partial) keys using their share of the IBE master secret; 2) collaboratively aggregate them; and, 3) submit the (aggregated) decryption key on-chain, which can be publicly verified using commitments to the IBE master secret. The players then fetch the respective decryption keys along with the ciphertext and locally decrypt their cards. Note that, to ensure that every decryption key is only available to the entitled player, a private key-derivation akin to [19, 42] is used. Should the associated proof fail to verify, anyone can submit a dispute transaction, triggering the on-chain contract to run the verification algorithm and potentially penalize the cheating party.[4]

Fig. 1 has the flow-diagram. By executing all MPC steps ahead of time in the offline phase – in a sense, our entire MPC protocol can be construed as pre-processing – we have a constantly running factory-like shuffling pipeline that can absorb surges in player demand.

[2]Consensus algorithms require honest majority assumptions by definition, contrary to our requirement of not placing trust in *any* other player. Use of bulletin boards, such as public blockchains, for this agreement is far too expensive.

[3]Note that the ciphertext size is large, and is virtually impossible to reduce significantly. In practice, this can mitigated by just posting a hash on-chain, and storing the ciphertexts in a layer-2 data-availability service such as [1].

[4]For simplicity we assume a smart bulletin board abstraction which runs the verification procedure on each submitted value. This obviates dealing with an explicit dispute mechanism in our exposition.

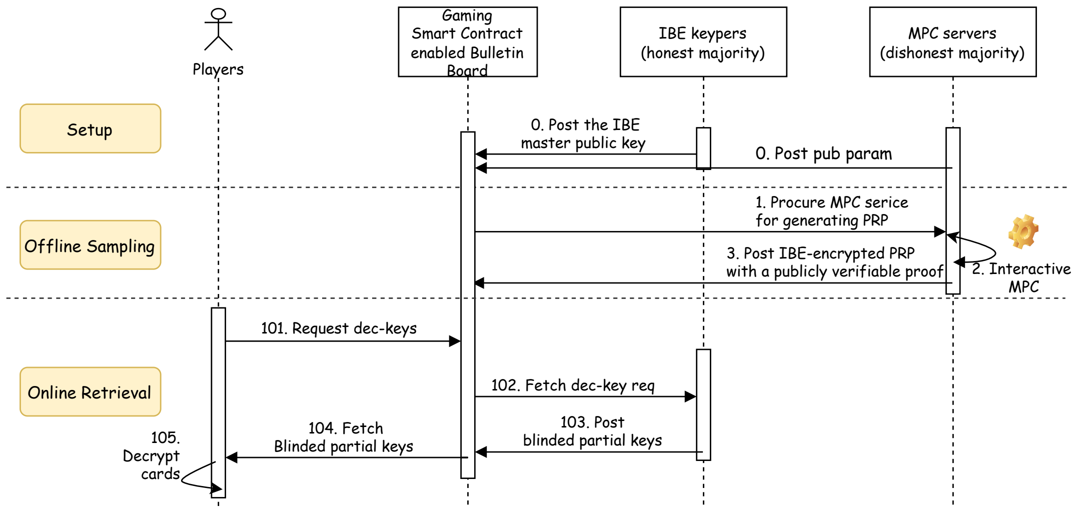
Players
Gaming Smart Contract enabled Bulletin Board
IBE keypers (honest majority)
MPC servers (dishonest majority)
Setup
Offline Sampling
Online Retrieval
0. Post the IBE master public key
0. Post pub param
1. Procure MPC serice for generating PRP
2. Interactive MPC
3. Post IBE-encrypted PRP with a publicly verifiable proof
101. Request dec-keys
102. Fetch dec-key req
103. Post blinded partial keys
104. Fetch Blinded partial keys
105. Decrypt cards

Figure 1: Flow Diagram for Insta-Pok3r protocol. To clearly distinguish between the offline publicly verifiable shuffling process and the online gaming activities, we start the offline shuffling at step 0, and the online process at step 100.

1.1 Contributions

The Insta-Pok3r Framework. We introduce a new framework defining offline verifiably-encrypted permutation generation, combined with a fast online retrieval. We formalize our framework via an ideal functionality $\mathcal{F}_{\mathsf{VES}}$ in the universal composability [17] framework.

The SVME Primitive. Along the way, we develop a new primitive called succinctly verifiable multi-identity based encryption (SVME). This is an enhancement of existing verifiable encryptions [16, 52, 28]. For any tuple of plaintexts satisfying a relation, SVME generates a multi-receiver ciphertext, along with a (constant-size) proof allowing for public verification of that relation while ensuring privacy. We construct a SVME scheme, and further show a specific instantiation of that scheme for relations encoding a permutation argument (within a small range). Our construction includes a novel combination of (i) a variant of Boneh-Franklin's IBE [13]; (ii) KZG polynomial commitments [42]; (iii) Plonk proof techniques [8]; (iv) and a *new (succinct) sigma-proof system* "linking" the commitment and the encryption. This can be of independent interest.

Distributed Protocol for Insta-Pok3r. To realize our framework, we design an MPC protocol to be executed in the offline phase to generate verifiably encrypted random permutation. The protocol combines several new and existing ideas:

• We construct a *new protocol* to generate an additive sharing of a set of small values (such as $\{1, \omega, \omega^2, \ldots, \omega^k\}$, where ω is a k-th root of unity), where the sharing is over a large prime field. This builds on ideas from [26] for generating shares of random bits over a large field.

• Combining that with a (input and key homomorphic) distributed PRF by Dodis and Yampolskiy [29] we obtain a new MPC protocol for generating pseudorandom permutations of a small set – here, each party obtains a secret share of the permutation.

• Then, we use the generated permutation as the plaintext input tuple within our SVME scheme, although now in a distributed setting, in that parties only possess secret shares of the plaintexts. In particular, we design an MPC protocol to realize SVME. This is made possible by converting each of the building blocks to a distributed setting. While some of them, such as Boneh-Franklin's IBE and KZG commitment, were simple to distribute by exploiting their respective homomorphic structures, others such as PLONK needed more effort, requiring recent techniques from collaborative SNARKs [47]. Importantly, the distributed version does not compromise privacy or succinctness.

Finally, the online phase is instantiated simply by using the threshold compatibility of the IBE decryption scheme. In particular, to execute over on-chain, we use a blinded key-derivation technique, similar to [19, 41].

We provide UC-based formal definitions for each component, describe our protocols in highly modularized hybrid models, and prove them individually via the real / ideal paradigm; the analysis of the final protocol Π_{VES} follows essentially from UC. This abstraction / modularization of individual components enables possible improvements in the future without redoing the full analysis. Remarkably, for our functionality (which is input-less) we are able to achieve resilience against malicious behavior by using (public) verifiability of the SVME scheme without explicitly resorting to cheating-prevention techniques in MPC literature (such as authenticated sharing [27]).[5]

Implementation. We implement the protocol as a Rust crate, and perform performance testing. For example, for 8 MPC parties (each with a single CPU), generating a secret publicly verifiable permutation of 52 cards takes about 3 secs; with 20 parties, the MPC time grows to about 11 secs – this is the computation latency for the offline sampling phase. It convincingly manifests that factory-like pipelining (along with parallelization) is quite feasible. In the online retrieval phase, accessing cards for a player, which includes proof verification and IBE decryption, takes under 0.3 secs.

2 Technical Overview

We briefly describe the setting and requirements for realizing $\mathcal{F}_{\mathsf{VES}}$, before delving into techniques.

Setting. There are three group of parties: (i) n MPC servers $S_1, \ldots, S_n$, among whom $\leq n-1$ can be corrupt. (ii) $\tilde{n}$ keypers, with a corruption threshold $\tilde{t} < n/2$, and execution threshold $\tilde{t}+1$ (these many parties suffice for an execution); (iii) m players. We consider an **adversarial model**, in that subset of these parties may be corrupt (and collude), maliciously and statically, as long as the number of corrupt MPC servers does not exceed $n-1$, and the number of corrupt keypers is bounded by $\tilde{t}$ – we call this admissible corruption, this guarantees full security. If all n servers are corrupt, the permutation is not private anymore, but public verifiability can still be guaranteed. We remark that our choices are made keeping the requirements in mind, as explained later.

Looking ahead we use a variant of SPDZ [27], secure against $n-1$ *semi-honest* corruption for the MPC. This suffices to provide full malicious security (against admissible corruption) because of the public verifiability of the output. Also, this helps immensely in keeping the overall cost low. Honest majority in the keyper committee is necessary to ensure guaranteed output delivery in the online phase. As long as the corruption is admissible, we wish the construction to satisfy the following **security requirements**:

– First, cards dealt to an honest player must remain **private** throughout, until they are opened by a player during a game (e.g. flop, turn, river cards in Texas Hold'em poker, or players opening their cards).
– Second, the encrypted deck, produced at the end of the offline phase must be **publicly verifiable**. This simplifies the game mechanics and also allows for fee renumeration on smart contracts.

We also stress some key **performance requirements**:

– First, **guaranteed output delivery** is needed in the online phase to ensure the liveness of the game. The offline phase, however, does not require this, as that can be dealt with other system measures, such as restarting, penalty etc.[6]
– Second, we require **low and predictable latency** in the online phase, from the time the players arrive to play a hand to the time cards are dealt. This enables (almost) instantaneous delivery to each player.

Before diving deeper, we take a detour and outline our core primitive: succinctly verifiable multi-identity encryption (SVME) (full details are in Section 5).

[5]This may, in fact, extend to any input-less functionality – a formal analysis is out of scope for this work.

[6]While our choice of the MPC protocol executed by the servers may not provide guaranteed output delivery (incorporating one might blow up the cost prohibitively), we can still achieve a relaxed form of liveness by performing the MPC sufficiently ahead-of-time, and in case of abort, identify and then kick off the cheater, and restart – this is possible here because the shuffling functionality is input-less, which means identifiable abort is achieved asking everyone to open up randomnesses in case of an abort.

2.1 Overview of SVME

Verifiable encryption [16, 52, 28] guarantees public verification of a ciphertext to attest a property of the encrypted plaintext, with respect to an efficiently verifiable relation. This can be immediately extended to support multi-receiver IBE, in that a tuple of plaintexts $\mathbf{m} = (m_1, \ldots, m_N)$ is encrypted with respect to a tuple of identities $\mathbf{id} = (id_1, \ldots, id_N)$. Similar to IBE, we would need to support id-key derivation and subsequent decryption. Taking this concept a step further, we require a succinct proof π, that neither grows with the size of the verification circuit nor N – this succintness property is needed in addition to the usual properties of correctness, soundness, and zero-knowledge.

Constructing SVME for Permutations. We construct an SVME scheme where $\mathbf{m}$ is a permutation of $(1, \omega, \ldots, \omega^{63})$ for a fixed public 64^{th} root ω (this choice will be made clear in Sec 2.2.1). Here, it is important to note that, while many constructions are possible (for example generically using an IBE with ZK proofs), we are restricted to choosing certain schemes as building blocks, that support distributed computation when the plaintext tuple is additively shared. Towards that, we choose a variant of pairing based (assume a pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, where g_i generates $\mathbb{G}_i$) Boneh-Franklin IBE scheme which supports the homomorphic computation of both message and plaintext. In this scheme, the ciphertext $\mathbf{c} = (c_1, \ldots, c_N)$, is such that each component is: $(g_1^\rho, e(\mathsf{H}(id)^\rho, \mathsf{mpk}) \cdot g_1^m)$ for message m and randomness ρ. Clearly, putting the message in the exponent makes it additively homomorphic like exponentiated ElGamal. However, in general, for arbitrary message space, this does not work because decryption would require solving discrete logs. Fortunately, in our case we need to decrypt messages that are within a small range – this makes the decryption highly efficient, for example, just using a look-up table.

Choosing Appropriate Succinct Proof Systems. We need to choose a proof system, which would translate easily into the collaborative setting when the witnesses are secret-shared. We notice that using a proof directly with encryption may not be that efficient, especially in the distributed setting. Therefore, we use a deterministic polynomial commitment, namely KZG [42], to commit to the permutation tuple – let us denote the commitment of m as M. Our proof π consists of two parts: (i) a proof, π_{perm} showing that M and V are commitments of two vectors that are permutation of each other, where V is a fixed commitment of $(1, \omega, \ldots, \omega^{63})$; (ii) another proof π_{lec} which links the commitment M with ciphertext $\mathbf{c}$, attesting that they have the same $\mathbf{m}$ committed / encrypted. We observe that this way we delegate the complex permutation checking task to generating π_{perm}, for which there is already an existing collaborative construction [47], that can be plugged in directly. To generate π_{lec}, we construct a succinct variant of sigma protocol, which uses the homomorphic property of the ciphertext to evaluate the polynomial corresponding to $\mathbf{m}$ in the exponent, and then uses a standard discrete log equality proof [23] between this and KZG commitment M. For soundness, we need an evaluation on a random point (output of a random oracle), for which we use a KZG opening. It is a well-known fact that Sigma protocols support collaborative proof generation with more interaction – we deploy this in the distributed setting. Finally, it is important to note that the KZG polynomial commitments are highly structured, and due to that support homomorphic computation, which suffice to compute them in a distributed setting.

2.2 Designing Insta-Pok3r Protocol

Armed with our main tool (SVME), we now focus on the construction of Π_{VES} that realizes the $\mathcal{F}_{\mathsf{VES}}$ functionality. As discussed earlier, there is a setup, followed by (i) an offline sampling phase; and (ii) an online retrieving phase. While we provide extensive details, we remark that during setup, public parameters pp for SVME need to be established and posted on-chain. From the above description, pp would consist of a master public key mpk for the IBE, with each keyper K_i holding their $\tilde{t}$-out-of-$\tilde{n}$ Shamir secret share msk_i – this can be established, for example, via a distributed key generation [35, 39, 38] (each individual public key $\mathsf{mpk}_i = g_1^{\mathsf{msk}_i}$ is also made part of pp using the same DKG). Additionally, pp would contain the KZG public parameters, establishing which is more involved. For that, either a decentralized protocol [54] or a setup ceremony can be performed.[7]

[7]Alternatively, we can rely on one of several existing SRS; our implementation uses Ethereum's ceremony [50].

2.2.1 Constructing Offline Sampling Protocol

Now we provide an overview of the offline sampling protocol that is executed between the MPC servers to produce a tuple $(\mathbf{c}, \pi)$, which is an SVME encryption (and proof) of a permutation $\mathbf{m}$ of $\{1, \ldots, \omega^{63}\}$. This phase can be split into two parts: (i) protocol $\Pi_{\textsf{R-Perm-Gen}}$ which implements an input-less computation that produces a pseudorandom permutation, additively shared between the participants; and (ii) the distributed SVME encryption protocol $\Pi_{\textsf{Enc}}$ which takes shares of the permutation, and collaboratively produces $(\mathbf{c}, \pi)$ – discussed in Section 2.1. We only focus on $\Pi_{\textsf{R-Perm-Gen}}$.

Step-1 of $\Pi_{\textsf{R-Perm-Gen}}$: Secret-sharing many random roots of unity. In this step the goal is to generate secret shares of many (say B-many, where $B \gg 64$) random values within the range $\{1, \ldots, 64\}$. However, generating secret shares of small (random) values are not easy – parties must ensure that the shared value is in this small range. Using ideas from [26] we construct a *new protocol* to additively secret share a random 64-th root of unity in a field. By fixing a 64-th root of unity $\omega \in \mathbb{Z}_p$, $\Omega = \{1, \omega, \omega^2, \ldots, \omega^{63}\}$ contains all the 64-th roots of unity. There is also a bijection between Ω with $\{1, \ldots, 64\}$ assuming $p = 1 \mod 64$. Then, we observe that, one can simply generate a random element in Ω by sampling a uniform random x in $\mathbb{Z}_p$ and then computing $x \cdot (\sqrt[64]{x^{64}})^{-1}$. Based on this principle, our protocol works as follows: we start with a secret-sharing of a random field element $[x]$, where $[x]$ denotes an additive sharing of x; this is simply done by each party locally sampling a random field element x_i, where $x = \sum_i x_i$. Next, by making *six* sequential invocations of SPDZ's multiplication parties compute shares of $[x^{64}]$. After that, they collaboratively reconstruct x^{64} in the clear. Finally, each party computes $(\sqrt[64]{x^{64}})^{-1} \cdot [x]$, which, by the linearity of additive sharing, is a share of a random 64^{th} root of unity, and thus a random element in Ω. At the end of this step MPC servers end up with (shares of) a large deck, where all cards come from Ω, but with a large number of repetitions. In the next step we eliminate the repetition.

Step-2 of $\Pi_{\textsf{R-Perm-Gen}}$: Checking for collisions. Each server must discard any share that, if reconstructed, will equal some other card in the shuffled deck. Equality checking is trivial if the cards are reconstructed in the clear, but we need to retain them in the secret-shared form. To that end, we compute a pseudo-random function (PRF) of the cards, which would allow for equality checking in the clear – the PRF hides the secret input. The challenge then is to compute the PRF without letting any single party learn the reconstructed card; i.e., the distributed PRF protocol must operate with shares of the input $[x]$ (along with shares of the PRF key $[sk]$, which is again additively shared). Fortunately, the Dodis-Yampolski PRF [30] fits this requirement ($\text{PRF}_{sk}(x) := g^{\frac{1}{x+sk}}$), and has an efficient distributed protocol that has each party locally compute and reveal $g^{([x]+[sk])^{-1}}$, which is reconstructed via group additions to get the PRF output $g^{([x]+[sk])^{-1}}$ – the PRF output is a group element for which the element g is a generator. With this distributed PRF functionality, we can do rejection sampling, or better yet, generate several samples in a batch, and search for 64 unique cards amongst them (with a small probability of failure, which is handled by generating more samples – we use 400 for 64 cards). At the end of the shuffling protocol, we end up with shares of all 64 cards $\{[x_j]\}_{j \in [64]}$ without repetition, in other words a permutation.
In most card games, we require decks of cards to be permuted, which is not a power of two. In this case, the servers first compute a permutation of size 64. To obtain a permutation of size smaller than 64, (say 52), the servers explicitly compute *and open* PRFs for the cards that should not be present in the final permutation $(53 \ldots 64$ in the above example). Then they discard the (unopened) cards in the permutation that have PRFs corresponding to the above PRFs. The remaining cards form a permutation of size 52. We note that this can be avoided by constructing a permutation of size 52 directly, which requires a 52^{nd} root of unity to exist in the field, equivalently $p = 1 \mod 52$ – this does not hold for most fields (but does for bn254).

Next, we use $\Pi_{\textsf{Enc}}$, which realizes the SVME encryption on the shared deck to produce ciphertext-proof pair $(\mathbf{c}, \pi)$, which is posted onto the blockchain.

2.2.2 Online Retrieval

In the online phase, when players come, they put a request on-chain for their decks. The requests are fetched by the keypers; each keyper K_i then returns a partial id-key $\mathsf{H}(id)^{\mathsf{msk}_i}$ – a player who owns the identity combines $\tilde{t} + 1$ such responses to obtain $\mathsf{H}(id)^{\mathsf{msk}}$, which it uses to IBD-decrypt the card associated with id. However, the main issue here is that the communication is happening on-chain, so everyone can compute all secret keys.

This could have been prevented if we assumed the existence of direct secure channels between the players and the keypers, but that would not fit into our offline-online setting. Instead, we can borrow the verifiable private key-derivation technique recently put forward in [19, 41]. The idea is: a player P_j sends a blinded request $\mathsf{H}(id)^{\mu_j}$ with a blind μ chosen by P_j. This is verified publicly on-chain whether it is indeed corresponding to a legitimate identity via a simple proof of exponent. The rest of the computation stays pretty much the same, except now the secret keys available on the chain are blinded as $\mathsf{H}(id)^{\mu_j \mathsf{msk}_i}$, and μ_j is only known to P_j. This ensures that P_j can obtain the corresponding key exclusively. However, in this setting, we assume malicious corruption of up to $\tilde{t}$ keypers. Therefore, each response by the keyper must be publicly verifiable – this is enabled by another proof of exponent with respect to $\mathsf{mpk}_i = g_1^{\mathsf{msk}_i}$, which is available as part of pp.

Finally, it is within the smart contract's capability to ensure that a player only gets keys corresponding to legitimate identities. For example, in a four-player game, each card of the deck can be sequentially indexed as $s_{i,j}$ for $i \in \{1, \ldots, 4\}$ and $j \in \{1, \ldots, 16\}$. An id can simply be $id_{i,j} = (\mathsf{gid}, i, j)$ where gid is the unique game/table identity. Each player P_i owns all identities $id_{i,j}$ – this can be easily checked by a contract.

3 Related work

Starting with Mental poker [48, 36] in 1981, there is a long series of works considered designing card games securely. The aim is to demonstrate that it is possible to play a classic card game without physical cards, without trusting the participants, and without a trusted third party. While it became apparent that the problem can be modeled and solved as an MPC problem, most recent efforts particularly focus on making solutions practically relevant and efficient. In this section, we compare some prominent approaches with our solution.

Randomness Service. Verifiable (pseudo-) randomness services [20, 4, 31], abundant on blockchains today, are commonly used for coin-tossing applications such as lotteries, where unpredictability and verifiability are desired properties. However, for poker, the shuffled card deck must remain private to all entities (except the owner) for the duration of the poker hand, and possibly well beyond, requiring the randomness to be private – specifically, we want each player to learn only her cards, and no other player or third party must learn those cards. This leads us to recent developments such as FlexiRand [41], which implements a VRF with output privacy. This is insufficient, as shuffling needs correlated randomness, where the private outputs of multiple players satisfy a relation, specifically that the cards represent a valid permutation of the deck.

Secure Card Shuffling. Most existing approaches [3, 5] in the Web3 space have each player take turns shuffling an encrypted deck of cards, and attach a zero-knowledge proof to prove the correctness of their shuffles. Previously, Barnett and Smart [9], Stamer [51], and Golle [37] proposed variants of mental poker protocols where the players run an interactive protocol for shuffling. As mentioned before, we seek a different model of "instant" poker where the deck is shuffled a priori, independent of the players (who may trigger delays by dropping out during the shuffling phase).

Nevertheless, the general shuffle-then-re-encrypt approach adopted by these protocols is of interest. Specifically, each player takes as input an encrypted list of cards, applies her own random permutation, and re-encrypts the output for the next player along with a ZK proof of correct shuffling. In the final phase, the players engage in a threshold decryption protocol to reveal the designated cards to players. Similar to our protocol, assuming a known fixed bound on message delays (i.e., bounded synchrony), it is possible to have identifiable aborts. However, the key challenge with these approaches is the intermediate ciphertexts and associated ZK proofs that make the on-chain (i.e., smart-contract-based) public verifiability aspect of the shuffling process expensive. While, in principle it is possible to use recursive ZK proofs to compress the argument (i.e., the intermediate ciphertexts and ZK proofs); this makes the off-chain proof generation process more expensive. Moreover, we find the on-chain verifiability remains significantly expensive as compared to our protocol.

Bultel and Lafourcade [15], and Bella et al. [11] consider the problem of secure Trick-Taking Game. While the considered games are also card-playing games such as Spades, they focused on preventing real-world cheating where a dishonest user can play a wrong card when having a correct card available. This work focuses on generating the (pseudo)randomly permuted decks and leaves detection/prevention of in-game trickeries to the smart contract. We expect smart contracts can deal with these issues in a crypto-economic sense; however, further details are out of scope.

Publicly Auditable MPC. More broadly, our system is providing public verifiability for an MPC-based computation, for which new constructions have been proposed over the years [40, 10]. Not only is our MPC protocol tailored for shuffling, we also develop a tailored, efficient proof system based on a PLONK-style permutation argument and sigma protocols, much more efficient than generic proofs for arithmetic circuits.

Blockchain-based Solutions. Kumaresan et al. [44] and Bentovet al. [12] considered the card problems over the Bitcoin-like ledgers. However, similar to the above-discussed works, these approaches require the players to be present and live to perform MPC. As we discussed earlier, we focus on the settings where the players need not even know when we shuffle cards and build the game as MPC as a Web3 service. We also observe most of these games allow security-with-abort. A single player can disturb the game proceeding, which may not be acceptable in the real world. Finally, these approaches also focus on the monetary aspects of card games like Poker such as how to perform fair exchanges with the restricted programming capabilities of Bitcoin. These issues can be addressed on a Turing-complete smart contract platform, and hence, we leave post-shuffling issues to the game's smart contract.

Anonymity Communication Networks. As anonymous communication networks (ACN) [22, 21, 53] also shuffle/mix users' inputs to make individual users' input messages unlinkable to a set of outputs, several ACN solutions become immediately applicable to the card shuffling problem. However, most of these ACN solutions (such as [7, 46, 45, 24, 33]) are focused on efficiency for a larger number of inputs than what is required for card games. Moreover, they do not offer publicly verifiable ZK proofs of correct shuffling and thus cannot be applied to our setting in a simple way.

4 Notations and Preliminaries

Notations. We use $\mathbb{Z}$ for integers, and $\mathbb{N}$ for $\{1, 2 \ldots\}$. $x \xleftarrow{\$} \mathcal{D}$ indicates that x is randomly sampled from the domain $\mathcal{D}$ and $h := y$ indicates that the h is assigned the value y. Also, for any (possibly randomized) algorithm A, $y \leftarrow A(x)$ indicates that A on input x yields the output y. Unless explicitly mentioned, all algorithms (including adversaries) in this paper are PPT. $y := A(x; r)$ is used to determinize A, with input x and fixed randomness r. The security parameter is denoted by λ. Tuples $(x_1, x_2 \ldots)$ and vector $\mathbf{x}$, and used interchangeably. We denote $\approx_c$ to denote computational indistinguishability of two distributions. A negligible function is defined as one that vanishes quickly than $1/poly(\lambda)$ for a security parameter λ. For n-party distributed systems, where we denote the set of corrupt parties by $\mathcal{C} \subseteq [n]$ and honest parties by $\mathcal{H} := [n] \setminus \mathcal{C}$.

Throughout the paper, we use the following notations for polynomials over finite fields. Let $\Omega \subset \mathbb{Z}_p$ be a multiplicative subgroup of order $|\Omega|$ over the finite field $\mathbb{Z}_p$. We publicly fix an ω, a $|\Omega|$-th root of unity in $\mathbb{F}$, that generates $\Omega = \{1, \omega, \ldots, \omega^{|\Omega|-1}\}$. Each polynomial $p(X)$ of degree $(|\Omega| - 1)$ over $\mathbb{Z}_p$ can be expressed in the lagrange basis and evaluated via interpolation $p(x) = \sum_{i=1}^{|\Omega|} \ell_i(x)p(\omega^{i-1})$ where: $\ell_i(x) := \omega^{i-1}(x^{|\Omega|} - 1)/|\Omega|(x - \omega^{i-1})$. Also, we define the degree-$|\Omega|$ vanishing polynomial over Ω as $z_\Omega(x) = \prod_{i=1}^{|\Omega|}(x - \omega^{i-1})$.

For any element $x \in \mathbb{Z}_p$ we use $[x]$ to denote an additive secret-share of x. For a cyclic group $\langle g \rangle = \mathbb{G}$ of order p we use the same notation $[X] \in \mathbb{G}$ which implies $[X] = g^{[x]}$ for any $X = g^x$. Furthermore, the notation extends to support a set of shares $[x]_S$ denoting all secret shares owned by parties in set S. An adequate number of parties, all holding $[x]$, can collaborate to reconstruct x. We denote $[x] \leftarrow_{\$} \mathcal{D}$, to imply each participant sampling local share such that x is being implicitly sampled as the secret randomness.

Shamir's secret sharing [49]. Shamir's scheme shares a field element $s \in \mathbb{Z}_p$ in a t out of n access structure by choosing a uniform random t degree polynomial $p(X)$ over $\mathbb{Z}_p$ subject to $p(0) = s$ and then defining the i-th share as $p(i)$. Given any t evaluation points, s is unconditionally hidden, and any $t + 1$ evaluation points can uniquely recover s through Lagrange interpolation.

5 SVME: Definition & Construction

We introduce the notion of succinctly verifiable multi-identity-based encryption (SVME). This is related to verifiable encryption [16, 52, 28] extended to a setting where τ-many messages $m_1, \ldots, m_\tau$ are encrypted under possibly different identities $id_1, \ldots, id_\tau$. In addition to a multi-ciphertext, the encryption procedure produces a proof π to allow anyone to publicly verify that the messages satisfy an efficiently verifiable τ-ary relation $\mathfrak{R}$. $\mathcal{C}_{\mathfrak{R}}$ denotes the (polynomial size) arithmetic circuit used to verify $(m_1 \ldots, m_\tau) \in \mathfrak{R}$. The proof π is *succinct* (i.e. constant-sized).

Definition 1 (SVME)**.** An SVME scheme for an efficiently verifiable τ-ary relation $\mathfrak{R}$ consists of algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{KeyExt}, \mathsf{Ver})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (\mathsf{msk}, \mathsf{pp}, \mathsf{td}_{\mathsf{svme}})$. The key-generation algorithm outputs a master secret key msk and public parameters pp. It may also optionally output a trapdoor $\mathsf{td}_{\mathsf{svme}}$ (deleted for running the protocol, but may be used in simulations).
- $\mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id}) \rightarrow (\mathbf{c}, \pi)$. The encryption algorithm takes as input $\mathbf{m} = (m_1 \ldots, m_\tau) \in \mathfrak{R}$ and identities $\mathbf{id} = (id_1, \ldots, id_\tau)$, then outputs a multi-ciphertext as $\tau + 1$ tuple $\mathbf{c} = (c_{\mathsf{aux}}, c_1, \ldots, c_\tau)$ and a proof π.
- $\mathsf{KeyExt}(\mathsf{pp}, \mathsf{msk}, id) \rightarrow \mathsf{sk}_{id}$. Similar to IBE the key-extraction works using the master secret and any identity to compute a secret key.
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_{id}, (c_{\mathsf{aux}}, c_{id})) \rightarrow m$. Similar to IBE the decryption algorithm takes a single ciphertext pair to compute message m using an appropriate derived key.
- $\mathsf{Ver}(\mathsf{pp}, \mathbf{c}, \mathbf{id}, \pi) \rightarrow 1/0$. The public verification algorithm takes the entire ciphertext tuple and the proof π (along with pp) to verify that the encrypted plaintexts indeed belong to the relation $\mathfrak{R}$.

They satisfy the following requirements:

Correctness. For any sufficiently large security parameter $\lambda \in \mathbb{N}$, any $\mathbf{m} := (m_1, \ldots, m_\tau) \in \mathfrak{R}$ and any set of identities $\mathbf{id} := id_1, \ldots, id_\tau$ the following probability is at least $1 - \mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c|c} & (\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{KeyGen}(1^\lambda); \\ \mathbf{m}' = \mathbf{m}; & (\mathbf{c}, \pi) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id}); \\ \mathsf{Ver}(\mathsf{pp}, \mathbf{c}, \pi) = 1; & \{\mathsf{sk}_{id_i} \leftarrow \mathsf{KeyExt}(\mathsf{pp}, \mathsf{msk}, id_i)\}_{i \in [\tau]}; \\ & \{m'_i := \mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_i, (c_{\mathsf{aux}}, c_i))\}_{i \in [\tau]} \end{array}\right]$$

Soundness. For any sufficiently large security parameter $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$ the following probability is at most $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c|c} & (\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{KeyGen}(1^\lambda); \\ (m_1, \ldots, m_\tau) \notin \mathfrak{R}; & (\mathbf{id}, \mathbf{c}, \pi) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \mathsf{Ver}(\mathsf{pp}, \mathbf{c}, \pi) = 1 & \{\mathsf{sk}_{id_i} \leftarrow \mathsf{KeyExt}(\mathsf{pp}, \mathsf{msk}, id_i)\}_{i \in [\tau]}; \\ & \{m_i := \mathsf{Dec}(\mathsf{pp}, \mathsf{sk}_i, (c_{\mathsf{aux}}, c_i))\}_{i \in [\tau]} \end{array}\right]$$

Zero-knowledge. For any sufficiently large security parameter $\lambda \in \mathbb{N}$ and any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that the following probability is at most $\mathsf{negl}(\lambda)$:

$$\left|\Pr\left[\begin{array}{c|c} & (\mathsf{msk}, \mathsf{pp}, \mathsf{td}_{\mathsf{svme}}) \leftarrow \mathsf{KeyGen}(1^\lambda), \\ & ((\mathbf{m} \in \mathfrak{R}), \mathbf{id}, \mathcal{C}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b = b' & (\mathbf{c}_0, \pi_0) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id}), \\ & (\mathbf{c}_1, \pi_1) \leftarrow \mathcal{S}(\mathsf{pp}, \{m_i\}_{i \in \mathcal{C}}, \mathsf{td}_{\mathsf{svme}}), \\ & \{\mathsf{sk}_{id_i} \leftarrow \mathsf{KeyExt}(\mathsf{pp}, \mathsf{msk}, id_i)\}_{i \in [\mathcal{C}]}, \\ & b \leftarrow \{0,1\}, b' \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, \{\mathsf{sk}_{id_i}\}_{i \in \mathcal{C}}, (\mathbf{c}_b, \pi_b)) \end{array}\right] - \frac{1}{2}\right|$$

Succinctness. For any security parameter $\lambda \in \mathbb{N}$, any $\mathbf{m} = (m_1 \ldots, m_\tau) \in \mathfrak{R}$ and any set of identities $\mathbf{id} = (id_1, \ldots, id_\tau)$, let $(\mathbf{c}, \pi) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id})$. Then, the size of π is independent of τ and $|\mathcal{C}_{\mathfrak{R}}|$, where $\mathcal{C}_{\mathfrak{R}}$ is the circuit for checking the membership in $\mathfrak{R}$.

Next, we present the building blocks for constructing our specific SVME scheme for permutations.

5.1 Building Blocks for SVME

In this section we present the syntax of three building blocks required to construct an SVME.

5.1.1 Multi-identity based encryption (MIBE)

A MIBE scheme is similar to an IBE scheme, except that it encrypts multiple messages under multiple identities to produce a ciphertext with a common part, which encodes the encryption randomness (and is independent of the message) and an individual part which encodes the message. An instantiation, based on Boneh-Franklin IBE [13] scheme is provided in Section 5.2.

Definition 2 (MIBE)**.** A multi-identity based encryption scheme consists of four algorithms:

- $\mathsf{MIBE.KeyGen}(1^\lambda) \to (\mathsf{msk}, \mathsf{mpk})$. The key-generation algorithm generates a master public-secret key pair.
- $\mathsf{MIBE.Enc}(\mathsf{mpk}, id_1, \dots, id_n, m_1, \dots, m_n) \to (\tilde{e}, e_1, \dots, e_n)$. The encryption algorithm takes n identities and n messages to produce a $n+1$ tuple, with element $\tilde{e}$ being common to all messages, and each e_i encoding the message m_i under the identity id_i.
- $\mathsf{MIBE.KeyExt}(\mathsf{msk}, id) \to \mathsf{sk}_{id}$. The key-extraction algorithm produces an identity-specific decryption key using the master secret key.
- $\mathsf{MIBE.Dec}(\mathsf{sk}_{id}, (\tilde{e}, e)) \to m$. The decryption algorithm uses a decryption-key to decrypt a ciphertext pair $(\tilde{e}, e)$.

We require the following properties to hold:

Correctness. For any sufficiently large security parameter $\lambda \in \mathbb{N}$, any set of messages $m_1, \dots, m_n$ any set of identities $\mathbf{id} := id_1, \dots, id_n$ the following probability is at most $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c|c} & (\mathsf{msk},\mathsf{mpk}) \leftarrow \mathsf{MIBE.KeyGen}(1^\lambda); \\ \mathbf{m}' \neq \mathbf{m}; & (\tilde{e}, e_1, \dots, e_n) \leftarrow \mathsf{MIBE.Enc}(\mathsf{mpk}, \mathbf{id}, \mathbf{m}); \\ \{\mathsf{MIBE.KeyVer}(\mathsf{pp}, \mathsf{sk}_{id_i}, id_i)\}_{i \in [n]} & \{\mathsf{sk}_{id_i} \leftarrow \mathsf{MIBE.KeyExt}(\mathsf{msk}, id_i)\}_{i \in [n]}; \\ & \{m'_i := \mathsf{Dec}(\mathsf{sk}_i, (\tilde{e}, e_i))\}_{i \in [n]} \end{array}\right]$$

CPA-security. For any sufficiently large security parameter $\lambda \in \mathbb{N}$, any set of identities $\mathbf{id}$, and any PPT adversary $\mathcal{A}$ then the following probability is bounded by $1/2 \pm \mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c|c} & (\mathsf{msk},\mathsf{mpk}) \leftarrow \mathsf{MIBE.KeyGen}(1^\lambda); \\ & (\mathbf{m}_0, \mathbf{m}_1, \mathcal{C}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{mpk}); \\ b = b'; & b \leftarrow \{0,1\}; \\ \{m_{0,i} = m_{1,i}\}_{i \in \mathcal{C}} & (\tilde{e}, \mathbf{e}) \leftarrow \mathsf{MIBE.Enc}(\mathsf{mpk}, \mathbf{id}, \mathbf{m}_b); \\ & \{\mathsf{sk}_{id_i} \leftarrow \mathsf{KeyExt}(\mathsf{msk}, id_i)\}_{i \in [\mathcal{C}]}; \\ & b' \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, \{\mathsf{sk}_{id_i}\}_{i \in \mathcal{C}}, (\tilde{e}_b, \mathbf{e}_b)) \end{array}\right]$$

5.1.2 (Trapdoor) Polynomial Commitments

We use a polynomial commitment scheme, which has succinctness and supports a trapdoor opening. This can be instantiated by KZG [42] commitments (details in Appendix 5.3).

Definition 3 ((Trapdoor) Polynomial Commitments)**.** A polynomial commitment scheme for polynomials of maximum degree d over field $\mathbb{F}$ consists of the following algorithms:

- $\mathsf{PC.Setup}(1^\lambda) \to (\mathsf{pp}_{\mathsf{pc}}, \mathsf{td}_{\mathsf{pc}})$. The setup algorithm generates public parameters and a trapdoor.
- $\mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, p(X)) \to P$. The commit algorithm commits a polynomial $p(X) \in \mathbb{F}[X]^{\leq d}$ to commitment P.

- $\mathsf{PC.Open}(\mathsf{pp_{pc}}, p(X), x) \to W$. On input a polynomial $p(X) \in \mathbb{F}[X]^{\leq d}$ and an input $x \in \mathbb{F}$, this algorithm outputs an opening W of $y = p(x)$.
- $\mathsf{PC.Ver}(\mathsf{pp_{pc}}, P, W, x, y) \to 1/0$. Given a commitment P of a polynomial $p(X) \in \mathbb{F}[X]^{\leq d}$ and an opening W with respect to input x and output y, this algorithm verifies whether W is indeed a correct opening of $p(x) = y$.
- $\mathsf{PC.TD.Open}(\mathsf{pp_{pc}}, \mathsf{td_{pc}}, P, x, y) \to W$. Given the trapdoor $\mathsf{td_{pc}}$, this procedure equivocates an opening W which asserts $p(x) = y$, when $P \leftarrow \mathsf{PC.Com}(\mathsf{pp_{pc}}, p(X))$.

We require them to satisfy the following properties:

Correctness. For sufficiently large $\lambda \in \mathbb{N}$, any $p(X) \in \mathbb{F}[X]^{\leq d}$, any $x \in \mathbb{F}$ the following probability is 1:

$$\Pr\left[\mathsf{PC.Ver}(\mathsf{pp_{pc}}, P, W, x, p(x)) = 1 \;\middle|\; \begin{array}{c} (\mathsf{td_{pc}}, \mathsf{pp_{pc}}) \leftarrow \mathsf{PC.Setup}(1^\lambda); \\ P \leftarrow \mathsf{PC.Com}(\mathsf{pp_{pc}}, p(X)); \\ W \leftarrow \mathsf{PC.Open}(\mathsf{pp_{pc}}, p(X), x) \end{array}\right]$$

Polynomial Binding. For a sufficiently large $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$ the following probability is at most $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c} p_1(X), p_2(X) \in \mathbb{F}[X]^{\leq d}; \\ \mathsf{PC.Com}(\mathsf{pp_{pc}}, p_1(X)) = P; \\ \mathsf{PC.Com}(\mathsf{pp_{pc}}, p_2(X)) = P \end{array} \;\middle|\; \begin{array}{c} (\mathsf{td_{pc}}, \mathsf{pp_{pc}}) \leftarrow \mathsf{PC.Setup}(1^\lambda); \\ (P, p_1(X), p_2(X)) \\ \leftarrow \mathcal{A}(\mathsf{pp_{pc}}) \end{array}\right]$$

Evaluation Binding. For a sufficiently large $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$ the following probability is at most $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c} (x_1, y_1) \neq (x_2, y_2); \\ \mathsf{PC.Ver}(\mathsf{pp_{pc}}, P, W_1, x, y_1) = 1; \\ \mathsf{PC.Ver}(\mathsf{pp_{pc}}, P, W_2, x, y_2) = 1 \end{array} \;\middle|\; \begin{array}{c} (\mathsf{td_{pc}}, \mathsf{pp_{pc}}) \leftarrow \mathsf{PC.Setup}(1^\lambda); \\ (P, W_1, W_2, x, y_1, y_2) \leftarrow \mathcal{A}(\mathsf{pp_{pc}}) \end{array}\right]$$

Trapdoor Opening. For any $\lambda \in \mathbb{N}$, any polynomial $p(X) \in \mathbb{F}[X]^{\leq d}$, and any $x, y \in \mathbb{F}$ the following probability is 1.

$$\Pr\left[W = W' \;\middle|\; \begin{array}{c} (\mathsf{td_{pc}}, \mathsf{pp_{pc}}) \leftarrow \mathsf{PC.Setup}(1^\lambda); \\ P \leftarrow \mathsf{PC.Com}(\mathsf{pp_{pc}}, p(X)); \\ W \leftarrow \mathsf{PC.Open}(\mathsf{pp_{pc}}, p(X), x); \\ W' \leftarrow \mathsf{PC.TD.Open}(\mathsf{pp_{pc}}, \mathsf{td_{pc}}, P, x, y) \end{array}\right]$$

Succinctness. For any $\lambda \in \mathbb{N}$ and any polynomial $p(X)$ let P is a commitment, and for any point $x \in \mathbb{F}$, W is an opening for $y = p(x)$. Then the size of P, W and the running time of algorithm $\mathsf{PC.Ver}$ are all $O(1)$.

5.1.3 Succinct Proofs

We use non-interactive succinct proof systems crucially in our construction.

Definition 4 (Succinct Proofs)**.** A succinct proof system for any NP binary relation $\mathfrak{R}$, consisting of instance-witness pairs $(\mathsf{inst}, \mathsf{wit})$, is a tuple of algorithms described as:

- $\mathsf{SP.Setup}(1^\lambda, \mathsf{aux}) \to (\mathsf{td_{sp}}, \mathsf{pp_{sp}})$. The setup algorithm outputs the public parameters and optionally a trapdoor for simulation. It may take some auxiliary information aux as an additional input.
- $\mathsf{SP.Prove}(\mathsf{pp_{sp}}, \mathsf{inst}, \mathsf{wit}) \to \pi$. This algorithm outputs (non-interactively) a proof π using a witness.
- $\mathsf{SP.Ver}(\mathsf{pp_{sp}}, \mathsf{inst}, \pi) \to 1/0$. This algorithm verifies legitimacy of the (public) instance inst using a proof π.

We require them to satisfy the following properties:

Completeness. For any sufficiently large $\lambda \in \mathbb{N}$, for any $(\mathsf{inst}, \mathsf{wit}) \in \mathfrak{R}$, the following probability is 1:

$$\Pr\left[\mathsf{SP.Ver}(\mathsf{pp_{sp}}, \mathsf{inst}, \pi) = 1 \;\middle|\; \begin{array}{c} (\mathsf{td_{sp}}, \mathsf{pp_{sp}}) \leftarrow \mathsf{SP.Setup}(1^\lambda, \mathsf{aux}); \\ \pi \leftarrow \mathsf{SP.Prove}(\mathsf{pp_{sp}}, \mathsf{inst}, \mathsf{wit}) \end{array}\right]$$

Existential Soundness. For any sufficiently large security parameter $\lambda \in \mathbb{N}$, any PPT adversary $\mathcal{A}$ the following probability is at most $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c} (\mathsf{inst}, \cdot) \notin \mathfrak{R}; \\ \mathsf{Ver}(\mathsf{pp_{sp}}, \mathsf{inst}, \pi) = 1 \end{array} \;\middle|\; \begin{array}{c} (\mathsf{td_{sp}}, \mathsf{pp_{sp}}) \leftarrow \mathsf{SP.Setup}(1^\lambda, \mathsf{aux}); \\ (\mathsf{inst}, \pi) \leftarrow \mathcal{A}(\mathsf{pp_{sp}}) \end{array}\right]$$

Zero-knowledge For any sufficiently large security parameter $\lambda \in \mathbb{N}$, and any PPT adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$, such that the following probability is at most $\mathsf{negl}(\lambda)$:

$$\left|\Pr\left[b = b' \;\middle|\; \begin{array}{c} (\mathsf{td_{sp}}, \mathsf{pp_{sp}}) \leftarrow \mathsf{SP.Setup}(1^\lambda, \mathsf{aux}); \\ ((\mathsf{inst}, \mathsf{wit}) \in \mathfrak{R}, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{pp_{sp}}); \\ \pi_0 \leftarrow \mathsf{SP.Prove}(\mathsf{pp_{sp}}, \mathsf{inst}, \mathsf{wit}) \\ \pi_1 \leftarrow \mathcal{S}(\mathsf{pp_{sp}}, \mathsf{td_{sp}}, \mathsf{inst}); \\ b \leftarrow \{0,1\}; \\ b' \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A}, \pi_b) \end{array}\right] - \frac{1}{2}\right|$$

Succinctness For any security parameter $\lambda \in \mathbb{N}$ and for any $(\mathsf{inst}, \mathsf{wit}) \in \mathfrak{R}$ let $(\mathsf{td_{sp}}, \mathsf{pp_{sp}}) \leftarrow \mathsf{SP.Setup}(1^\lambda, \mathsf{aux})$ and $\pi \leftarrow \mathsf{SP.Prove}(\mathsf{pp_{sp}}, \mathsf{inst}, \mathsf{wit})$. Then the size of π is independent of $|\mathcal{C}_\mathfrak{R}|$, where $\mathcal{C}_\mathfrak{R}$ is a circuit used for checking membership $(\mathsf{inst}, \mathsf{wit}) \in \mathfrak{R}$.

5.2 Our MIBE Construction

Based on Boneh-Franklin's IBE scheme [13] we put forward the following MIBE construction to encrypt any n vector $(m_1, \dots, m_n)$. We also assume that each m_i is from a set of size at most $\phi(\lambda)$ to enable efficient (brute force) decryption. Recall that, for the SVME construction we need each $m_i \in \{1 \dots, 64\}$.

The scheme is based on bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$[8] where each group has (prime) order p and a hash function (modeled as random oracles) $\mathsf{H} : \{0,1\}^* \to \mathbb{G}_1$. Suppose n is a parameter denoting the number of messages encrypted.

- $\mathsf{MIBE.KeyGen}(1^\lambda) \to (\mathsf{msk}, \mathsf{mpk})$: This algorithm samples $\mathsf{msk} \leftarrow_\$ \mathbb{Z}_p$ and set $\mathsf{mpk} := g_1^{\mathsf{msk}}$.
- $\mathsf{MIBE.Enc}(\mathsf{mpk}, id_1, \dots, id_n, m_1, \dots, m_n) \to (\tilde{e}, e_1, \dots, e_n)$. The multi-identity encryption scheme encrypts message m_i with respect to id_i and outputs a ciphertext that consists of $n+1$ elements, with $\tilde{e} \in \mathbb{G}_1$ and each $e_i \in \mathbb{G}_2$ constructed as follows:
  - Sample $\rho \leftarrow_\$ \mathbb{Z}_p$ and compute $\tilde{e} := g_1^\rho$.
  - Compute $e_i := e(\mathsf{H}(id)^\rho, \mathsf{mpk}) \cdot g_2^{m_i}$.
- $\mathsf{MIBE.KeyExt}(\mathsf{msk}, id) \to \mathsf{sk}_{id}$. Similar to the Boneh-Franklin key-extraction algorithm this uses the master secret msk to derive the decryption key as $\mathsf{sk}_{id} := \mathsf{H}(id)^{\mathsf{msk}}$.
- $\mathsf{MIBE.Dec}(\mathsf{sk}_{id}, (\tilde{e}, e)) \to m$. It decrypts in the exponent as $M := E/e(\mathsf{sk}_{id}, \tilde{e})$ and then solving discrete log $m := \mathsf{DLog}_{g_2}(M)$ by brute force.[9]

We show via the next theorem that the construction satisfies our MIBE definition for m_i in small sets.

Theorem 1 (MIBE Construction)**.** *The above construction is a secure MIBE scheme in the random oracle model as long as bilinear DDH holds over underlying pairing group.*

[8]We use symmetric pairing notation for simplicity. We stress that it can easily be extended to support asymmetric pairing for efficiency, which we adapt in our implementation.

[9]In practice one may use a pre-computed look up table for fast decryption.

Proof. Correctness is easy to see, since by construction,

$$\begin{aligned}\frac{E}{e(\mathsf{sk}_{id}, \tilde{e})} &= \frac{e(H(id)^{\rho}, \mathsf{mpk}) g_2^{m_i}}{e(H(id)^{\mathsf{msk}}, \tilde{e})} \\ &= \frac{e(H(id), g_1)^{\rho \mathsf{msk}} g_2^{m_i}}{e(H(id), g_1)^{\rho \mathsf{msk}}} = g_2^{m_i}\end{aligned}$$

and this has the correct discrete logarithm. Note that decryption is efficient as long as the messages are chosen from a small space, since DLog is hard in this group.

For CPA-security recall the multi-identity security notion first. For any arbitrary set of n identities the adversary is allowed to obtain keys for upto $n-1$ identities – without loss of generality, assume that it obtains keys for the first $n-1$ identities (Note that any additional keygen queries that do not involve any of these n identities give no information to the adversary due to the random oracle H). We will show that the adversary cannot distinguish the encryption of any m_0 and m_1 encrypted with the n-th identity (for which it does not know a secret key).
Suppose the encrypted message is m_0. Then, the adversary's view in the CPA game is:

$$\begin{aligned}&(\mathsf{mpk}, \mathsf{sk}_{id_1}, \ldots, \mathsf{sk}_{id_{n-1}}, \tilde{e}, E_n) \\ &\equiv (g_2^{\mathsf{msk}}, \mathsf{H}(id_1)^{\mathsf{msk}}, \ldots, \mathsf{H}(id_{n-1})^{\mathsf{msk}}, \\ &\qquad g_2^{\rho}, e(\mathsf{H}(id_n), g_2)^{\rho \cdot \mathsf{msk}} \cdot g_T^{m_0}) \\ &\equiv (g_2^{\mathsf{msk}}, (g_1^{\mathsf{msk}})^{r_1}, \ldots, (g_1^{\mathsf{msk}})^{r_{n-1}}, \\ &\qquad g_2^{\rho}, e(g_1, g_2)^{r_n \cdot \rho \cdot \mathsf{msk}} \cdot g_T^{m_0}) \\ &\approx_{\mathsf{c}} \left(g_2^{\mathsf{msk}}, (g_1^{\mathsf{msk}})^{r_1}, \ldots, (g_1^{\mathsf{msk}})^{r_{n-1}}, g_2^{\rho}, u_T \cdot g_T^{m_0}\right)\end{aligned}$$

Above, we show certain equivalences between the distributions of the view of the adversary. The first equivalence holds by definition of the terms, and the second equivalence follows since H is a random oracle, and $H(id_i)$ can be written as $g_1^{r_i}$ for random r_i. The final computational indistinguishability step follows from BDDH, which says that given $g_1^{\mathsf{msk}}, g_2^{\mathsf{msk}}, g_1^{r_n}(= H(id_n)), g_2^{\rho}$ for uniformly random msk, r_n, ρ in $\mathbb{Z}_p$, it is computationally hard to distinguish between $e(g_1, g_2)^{r_n \rho \mathsf{msk}}$ and a uniform random element $u_T \leftarrow_{\$} \mathbb{G}_T$. Replacing the pairing with a random element in $\mathbb{G}_2$, we observe that the last term is now a uniformly random element in the group and independent of the value of m_0.
Similarly, the analysis when the encrypted message is m_1 leads to the same conclusion and hence by transitivity, the encryptions are indistinguishable. □

5.3 Instantiating Polynomial Commitments with KZG [42]

The KZG polynomial commitment scheme given below works for all polynomials of degree upto τ over $\mathbb{Z}_p$. It is based on bilinear pairing $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, where each $\mathbb{G}_i$ is a cyclic group of order p, and is generated by g_i. The algorithm specifications are as follows:

- $\mathsf{PC.Setup}(1^{\lambda}) \to (\mathsf{pp}_{\mathsf{pc}}, \mathsf{td}_{\mathsf{pc}})$. Sample a uniform random $\imath \leftarrow_{\$} \mathbb{Z}_p$ and publish $\mathsf{pp}_{\mathsf{pc}} := \{h_i := g_1^{\imath^i}\}_{i \in \{0,\ldots,\tau\}}$ and the trapdoor $\mathsf{td}_{\mathsf{pc}} := \imath$.
- $\mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, p(X)) \to P$. To commit to a τ-degree polynomial $p(X) = c_0 + c_1 x + \ldots + c_{\tau} x^{\tau}$ output the commitment $P = \prod_{i=0}^{d} h_i^{c_i} \in \mathbb{G}_1$ where $\mathsf{pp}_{\mathsf{pc}} = (h_0, \ldots, h_{d+1})$.
- $\mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, p(X), x) \to W$. Compute the quotient polynomial $w(X) := \frac{p(X)-p(x)}{X-x}$ and then output $W := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, w(X))$.
- $\mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, P, W, x, y) \to 1/0$. Return
$$(e(P/g_1^{y}, g_1) = e(W, h_0/g_1^{x}))$$
- $\mathsf{PC.TD.Open}(\mathsf{pp}_{\mathsf{pc}}, \mathsf{td}_{\mathsf{pc}}, P, x, y) \to W$. Parse $\imath := \mathsf{td}_{\mathsf{pc}}$, and output:
$$W := \left(\frac{P}{g_1^{y}}\right)^{(\imath - x)^{-1}}$$

Note that, the scheme maybe slightly altered by using evaluation representation of a polynomial instead, which we adapt in our implementation.

Also as shown in the original paper [42], this scheme is a secure polynomial commitment scheme according to Definition **??** as long as the DL and a variant of bilinear Diffie-Hellman assumptions hold.

5.4 Instantiating π_{perm} with Plonk [8]

Recall that this proof system is based on the KZG commitments and is for the language $\mathfrak{R}_{\mathsf{perm}}$ which contains instance-witness pair $(\mathsf{inst}, \mathsf{wit})$ where:

- $\mathsf{inst} := (M, V)$ and $\mathsf{wit} := (m(X), \rho_m)$ such that:
  - $M := \mathsf{PC.Com}(m'(X))$ where $m'(X) := m(X) + \rho_m z_\Omega(X)$;
  - $V := \mathsf{PC.Com}(v(X))$ where $v(X) := \mathsf{V2P}(1, 2, \ldots, \tau)$ is a fixed polynomial;
  - $\mathbf{m} = \mathsf{P2V}(m(X))$ is a permutation of $\mathbf{v} := (1, 2, \ldots, \tau)$.

We use Plonk [8] permutation check proof system – we include this for notational consistency and completeness. We assume the hash functions are chosen appropriately for the desired domains/ranges.

- $\mathsf{SP.PERM.Setup}(1^\lambda, \mathsf{aux}) \to (\mathsf{td}_{\mathsf{perm}}, \mathsf{pp}_{\mathsf{perm}})$.

  Let $(\mathsf{pp}_{\mathsf{pc}}, \mathsf{td}_{\mathsf{pc}}) := \mathsf{aux}$. Set $\mathsf{pp}_{\mathsf{perm}} := \mathsf{pp}_{\mathsf{pc}}$ and $\mathsf{td}_{\mathsf{perm}} := \mathsf{td}_{\mathsf{pc}}$.
- $\mathsf{SP.PERM.Prove}(\mathsf{pp}_{\mathsf{perm}}, (M, V), (m(X), \rho_m) \to \pi_{\mathsf{perm}}$:
  - Let $v(X) := \mathsf{V2P}(1, \ldots, \tau)$ and $V := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, v(X))$.
  - Compute the hash $\gamma_1 := \mathsf{H}_1(V, M)$.
  - Compute polynomials $g(X) := \gamma_1 + m(X) + \rho_m z_\Omega(X)$ and $h(X) := \gamma_1 + v(X)$.
  - Compute a $\tau - 1$-degree *rational* polynomial $t(X)$ such that $t(\omega^i) := \frac{g(\omega^i)}{h(\omega^i)}$ for all $i \in \{0, \ldots, \tau - 1\}$.
  - Compute $T := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, t(X))$.
  - Compute the quotient polynomial $q(X) := d(X)/(X^\tau - 1)$ where $d(X) := h(X) \cdot t(X) - g(X)t(X/\omega)$. If there is a non-zero remainder, stop and return $\perp$. Otherwise go to the next step.
  - Commit $Q = \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, q(X))$.
  - Compute the hash $\gamma_2 := \mathsf{H}_2(M, V, Q, T)$.
  - Finally compute the KZG openings:
    * $W_1 := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, t(X), \omega^{\tau-1})$;
    * $W_2 := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, t(X), \gamma_2)$;
    * $W_3 := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, t(X), \omega\gamma_2)$;
    * $W_4 := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, g(X), \omega\gamma_2)$;
    * $W_5 := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, q(X), \gamma_2)$;
  - Compute the polynomial evaluations: $y_1 := t(\omega^{\tau-1})$, $y_2 := t(\gamma_2)$, $y_3 := t(\omega\gamma_2)$, $y_4 := g(\omega\gamma_2)$ and $y_5 := q(\gamma_2)$.
  - Output π_{perm} where:

$$\begin{aligned}\pi_{\mathsf{perm}} :=& (Q, T, (y_1, W_1), (y_2, W_2),\\ & (y_3, W_3)(y_4, W_4), (y_5, W_5))\end{aligned}$$

- $\mathsf{SP.PERM.Ver}(\mathsf{pp}_{\mathsf{perm}}, (M, V), \pi_{\mathsf{perm}}) \to 1/0$

- Parse $\pi_{\mathsf{perm}} = (M, V, Q, T, (y_1, W_1), (y_1, W_2), (y_3, W_3), (y_4, W_4), (y_5, W_5))$
- Compute $\gamma_1 := \mathsf{H}_1(M, V)$, $G = M.g_1^{\gamma_1}$ and $\gamma_2 := \mathsf{H}_2(M, V, Q, T)$.
- Run KZG verification on the following:
  * $\mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, T, W_1, \omega^{\tau-1}, y_1)$.
  * $\mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, T, W_2, \gamma_2, y_2)$.
  * $\mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, T, W_3, \omega\gamma_2, y_3)$.
  * $\mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, G, W_4, \omega\gamma_2, y_4)$.
  * $\mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, Q, W_5, \gamma_2, y_5)$.
- Output 1 iff all verifications return 1, otherwise output 0.

First we note that the above proof system satisfies our Definition 4 for relation $\mathfrak{R}_{\mathsf{perm}}$ – this follows from [8] (the security argument works in algebraic group model assuming the hash functions as random oracles). It is worth noting that the zero-knowledge property is achieved as M is a commitment of $m'(X) = m(X) + \rho_m z_\Omega(X)$ instead of $m(X)$ itself; $m'(X)$ hides $m(X)$ unconditionally as long as no other information on ρ_m is given. This does not, however, affects the procedures, because when $t(X)$ is being defined, the ration $g(\omega^i)/h(\omega^i)$ remains unchanged, as $z_\Omega(\omega^i) = 0$ for all $i \in \{0, \ldots, \tau\}$ by definition. For more details we refer to the original paper [8]. We note that, the soundness should hold for any $m(X) \mod (z_\Omega(X))$, which suffices for our purpose due to the property of the zero polyonmial.

5.5 Chaum-Pedersen's (distributed) Proof of Equality of Discrete Log

We present Chaum-Pedersen's proof of equality of discrete log for completeness. We also present the distributed protocol for shared witness. We use a version that works over a symmetric bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_2$, where each group is of prime order p. The proof system would enable knowledge of witness ρ such that given the public instance $(x, y, X, Y) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_2$ $y = x^\rho$ and $Y = X^\rho$. The non-interactive (via Fiat-Shamir) variant has two algorithms:

- $\mathsf{DLEQ.Prove}((x, y, X, Y)(\rho))$: Compute:
  - $r \leftarrow_\$ \mathbb{Z}_p$;
  - $y' = x^r$ and $Y' = X^r$;
  - $c := \mathsf{H}(y', Y')$ $(c \in \mathbb{Z}_p)$;
  - $z := r + c\rho$;

  Output: $((y', Y'), c, z)$
- $\mathsf{DLEQ.Ver}((x, y, X, Y), ((y', Y'), c, z)$:

  Return $(x^z = y'.y^c) \wedge (X^z = Y'.Y^c)$

The completeness, soundness and zero-knowledge of the proof system is well established. For details we refer to, for example, [14].

5.5.1 Distributed DLEQ Proof

For the distributed SVME construction, we require a distributed proof system, in which n participants jointly posses secret share of the witness $[\rho]$. First we present an ideal functionality $\mathcal{F}_{\mathsf{DLEQ.Prove}}$:

- Upon $(\mathsf{gid}$, DLEQ-Prove, $((x, y, X, Y), ([\rho])))$ from all n parties, then reconstruct ρ and compute $\pi_{\mathsf{dleq}} := \mathsf{DLEQ.Prove}((x, y, X, Y), \rho)$ Return π_{dleq} to everyone.

We then present a protocol $\Pi_{\mathsf{DLEQ.Prove}}$:

- **Input:** Upon $(\mathsf{gid}, \text{DLEQ-Prove}, (\mathsf{PubIP}, \mathsf{PrivIP}))$ each party P_i has public input $\mathsf{PubIP} = (x, y, X, Y)$ and private input $\mathsf{PrivIP} = [\rho]$.
- **Step-1 (Non-interactive):** Sample $[r] \leftarrow_{\$} \mathbb{Z}_p$. Compute $[y'] := x^{[r]}$ and $[Y'] := X^{[r]}$.
- **Step-2 (Interactive):** Collaboratively reconstruct y' and Y'
- **Step-3 (Non-interactive):** Compute $c := \mathsf{H}(y', Y')$ and $[z] := [r] + c[\rho]$.
- **Step-4 (Interactive):** Collaboratively reconstruct z.
- **Output:** $((y', Y'), c, z)$.

Lemma 1. *The protocol $\Pi_{\mathsf{DLEQ.Prove}}$ securely realizes the ideal functionality $\mathcal{F}_{\mathsf{DLEQ.Prove}}$*

Proof (sketch). Essentially if there is at most $n-1$ corruption, the adversary still does not know anything about ρ. The simulator, with its private inputs $[\rho]_{\mathcal{C}}$ call the ideal functionality to obtain back (y', Y', z) as part of the output. In the interactive steps it samples the values $([y']_{\mathcal{H}}, [Y']_{\mathcal{H}}, [z]_{\mathcal{H}})$ on behalf of the honest parties so that the reconstructed values from $([y']_{\mathcal{H}}, [Y']_{\mathcal{H}}, [z]_{\mathcal{H}})$ and $([y']_{\mathcal{C}}, [Y']_{\mathcal{C}}, [z]_{\mathcal{C}})$ match the output (y', Y', z). If all n parties are corrupt the simulation is trivial. $\square$

5.6 Our SVME Construction

Notations. First recall from Sec. 4 the representation of an $(\tau-1)$-degree polynomial over sub-group $\Omega = \{1, \ldots, \omega^{|\Omega-1|}\}$ and the corresponding degree $|\Omega|$-degree vanishing polynomial z_Ω. Here we set $|\Omega| = \tau$ and $\mathbb{F} = \mathbb{Z}_p$. For $\mathbf{v} = (v_1, \ldots, v_\tau)$ where $v_i \in \mathbb{F}$, define a deterministic function $\mathsf{V2P} : \mathbb{F}^\tau \to \mathbb{F}[X]^{\tau-1}$ that converts a τ-dimensional vector $\mathbf{v}$ to a polynomial $v(X)$ of degree $\tau - 1$ over the same field $\mathbb{F}$ by setting the evaluations $v(\omega^i) = v_i$ and then interpolating using the Lagrange basis $\ell_i(x)$'s. Similarly we also define $\mathsf{P2V} : \mathbb{F}[X]^{\tau-1} \to \mathbb{F}^\tau$ which performs the inverse operation. Also we assume that there is a fixed dummy identity id_{pp}, such that for any $\mathbf{id}$ of size τ, $\hat{\mathbf{id}} := (\mathbf{id}, id_{\mathsf{pp}})$ has $\tau + 1$ elements.

Specific Succinct Proof Systems. For SVME construction we rely on two succinct proof systems $\mathsf{SP.PERM}$ and $\mathsf{SP.LEC}$ for the NP relations $\mathfrak{R}_{\mathsf{perm}}$ (a specific permutation relation, hence called perm) and $\mathfrak{R}_{\mathsf{lec}}$ (a relation linking encryption and commitment, hence called lec), respectively, as follows:

1. $\mathfrak{R}_{\mathsf{perm}}$ is based on a polynomial commitment scheme for polynomials in $\mathbb{Z}_p[X]^{\leq\tau}$. It captures that a polynomial $m(X) \in \mathbb{Z}_p[X]^{\tau-1}$ is such that $\mathbf{m} := \mathsf{P2V}(m(X))$ is a permutation of a fixed vector $\boldsymbol{v} = (1, \omega, \ldots, \omega^{\tau-1})$, where ω is a fixed public τ^{th} root of unity in $\mathbb{Z}_p$. For zero-knowledge a commitment to $m'(X) = m(X) + \rho_m z_\Omega(X)$ is provided (this is borrowed from Plonk [8]). This does not affect the instantiation, as this is essentially the (shifted) permutation check of Plonk.[10] It contains instance-witness pair $(\mathsf{inst}, \mathsf{wit})$ where:
   - $\mathsf{inst} := (M, V)$, $\mathsf{wit} := (m(X), \rho_m)$ such that:
     - $M := \mathsf{PC.Com}(m'(X))$ with $m'(X)$ as above;
     - $V := \mathsf{PC.Com}(v(X))$ where $v(X) := \mathsf{V2P}(\boldsymbol{v})$ is a fixed polynomial;
2. $\mathfrak{R}_{\mathsf{lec}}$ is based on a polynomial commitment scheme for polynomials in $\mathbb{Z}_p[X]^{\leq\tau}$ and an MIBE scheme for vectors of size $(\tau + 1)$ over $\mathbb{Z}_p$. Essentially, it captures that a polynomial commitment of $m'(X) = m(X) + \rho_m z_\Omega(X)$ and a MIBE ciphertext have the same values $\hat{\mathbf{m}} = (\mathsf{P2V}(m(X), \rho_m)$ inside them. Specifically, it contains instance-witness pairs $(\mathsf{inst}, \mathsf{wit})$ where:
   - $\mathsf{inst} = (M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk})$ and $\mathsf{wit} = (\rho, \rho_m, \mathbf{m})$:
     - $M = \mathsf{PC.Com}(m'(X))$ where $m'(X) := m(X) + \rho_m z_\Omega(X)$ and $m(X) = \mathsf{V2P}(\mathbf{m})$;
     - $\hat{\mathbf{m}} := (\mathbf{m}, \rho_m)$ and $(\tilde{e}, \mathbf{e}) = \mathsf{MIBE.Enc}(\mathsf{mpk}, \hat{\mathbf{id}}, \hat{\mathbf{m}}; \rho)$

The Construction. In Figure 2, we describe our SVME construction for relation $\mathfrak{R}_{\mathsf{main}}$ such that $(m_1, \ldots, m_\tau) \in \mathfrak{R}_{\mathsf{main}}$ if and only if $\mathbf{m} := (m_1, \ldots, m_\tau)$ is a permutation of $(1, \omega, \ldots \omega^{\tau-1})$.

[10]Note that this is simpler than Plonk since one of the polynomials V is fixed.

Ingredients: A MIBE scheme MIBE for $\tau + 1$ sized vector over $\mathbb{Z}_p$; a polynomial commitment PC for τ-degree polynomials over $\mathbb{Z}_p[X]$; two succinct proof systems $\mathsf{SP.LEC}$ and $\mathsf{SP.PERM}$ as described above.

– $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{msk}, \mathsf{pp}, \mathsf{td}_{\mathsf{svme}})$. On input the security parameter execute:
  - $(\mathsf{td}_{\mathsf{pc}}, \mathsf{pp}_{\mathsf{pc}}) \leftarrow \mathsf{PC.Setup}(1^\lambda)$; $\mathsf{aux} := (\mathsf{pp}_{\mathsf{pc}}, \mathsf{td}_{\mathsf{pc}})$; $(\mathsf{pp}_{\mathsf{perm}}, \mathsf{td}_{\mathsf{perm}}) \leftarrow \mathsf{SP.PERM.Setup}(1^\lambda, \mathsf{aux})$;
  - $(\mathsf{pp}_{\mathsf{lec}}, \mathsf{td}_{\mathsf{lec}}) \leftarrow \mathsf{SP.LEC.Setup}(1^\lambda, \mathsf{aux})$; $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{MIBE.KeyGen}(1^\lambda)$; $v(X) := \mathsf{V2P}(\boldsymbol{v})$;
  - Choose a random dummy id_{pp}; $\mathsf{pp} := (\mathsf{mpk}, id_{\mathsf{pp}}, \mathsf{pp}_{\mathsf{pc}}, \mathsf{pp}_{\mathsf{perm}}, \mathsf{pp}_{\mathsf{lec}}, v(X))$; $\mathsf{td}_{\mathsf{svme}} := (\mathsf{td}_{\mathsf{perm}}, \mathsf{td}_{\mathsf{lec}})$.

– $\mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id}) \to (\mathbf{c}, \pi)$. Parse $(\mathsf{mpk}, id_{\mathsf{pp}}, \mathsf{pp}_{\mathsf{pc}}, \mathsf{pp}_{\mathsf{perm}}, \mathsf{pp}_{\mathsf{lec}}) \leftarrow \mathsf{pp}$ and execute:
  - $m(X) := \mathsf{V2P}(\mathbf{m})$; $\rho_m \leftarrow_{\$} \mathbb{Z}_p$; $m'(X) := m(X) + \rho_m z_\Omega(X)$; $M := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, m'(x))$; $\hat{\mathbf{m}} := (\mathbf{m}, \rho_m)$;
  - $\hat{\mathbf{id}} := (\mathbf{id}, id_{\mathsf{pp}})$; $\rho \leftarrow_{\$} \mathbb{Z}_p$; $(\tilde{e}, \mathbf{e}) := \mathsf{MIBE.Enc}(\mathsf{mpk}, \hat{\mathbf{id}}, \hat{\mathbf{m}}; \rho)$; $V := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, v(x))$;
  - $\pi_{\mathsf{perm}} \leftarrow \mathsf{SP.PERM.Prove}(\mathsf{pp}_{\mathsf{perm}}, (M, V), (m(X), \rho_m)$; $\pi_{\mathsf{lec}} \leftarrow \mathsf{SP.LEC.Prove}(\mathsf{pp}_{\mathsf{lec}}, (M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), (\rho, \rho_m, \mathbf{m}))$;
  - $\pi := (M, V, \pi_{\mathsf{perm}}, \pi_{\mathsf{lec}})$; $\mathbf{c} := (\tilde{e}, \mathbf{e})$.

– $\mathsf{KeyExt}(\mathsf{msk}, id) \to \mathsf{sk}_{id}$. Execute $\mathsf{sk}_{id} \leftarrow \mathsf{MIBE.KeyExt}(\mathsf{msk}, id)$.

– $\mathsf{Dec}(\mathsf{sk}_{id}, (c_{\mathsf{aux}}, c_{id})) \to m$. Execute $m \leftarrow \mathsf{MIBE.Dec}(\mathsf{sk}_{id}, (\tilde{e}, e_{id}))$ where $\tilde{e} := c_{\mathsf{aux}}$ and $e_{id} := c_{id}$.

– $\mathsf{Ver}(\mathsf{pp}, \mathbf{c}, \mathbf{id}, \pi) \to 1/0$. Parse $(\tilde{e}, \mathbf{e}) := \mathbf{c}$ and $(M, V, \pi_{\mathsf{perm}}, \pi_{\mathsf{lec}}) := \pi$ and $(\mathsf{mpk}, \mathsf{pp}_{\mathsf{pc}}, \mathsf{pp}_{\mathsf{perm}}, \mathsf{pp}_{\mathsf{lec}}) \leftarrow \mathsf{pp}$. Then execute:
  - $d_{\mathsf{perm}} := \mathsf{SP.PERM.Ver}(\mathsf{pp}_{\mathsf{perm}}, (M, V), \pi_{\mathsf{perm}})$; $d_{\mathsf{lec}} := \mathsf{SP.LEC.Ver}(\mathsf{pp}_{\mathsf{lec}}, (M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), \pi_{\mathsf{lec}})$;
  - Return $(d_{\mathsf{perm}} \wedge d_{\mathsf{lec}})$.

Figure 2: Our SVME Construction for Permutation

We now argue that our construction is an SVME scheme for $\mathfrak{R}_{\mathsf{main}}$, formalized via the following theorem, a detailed proof for which is given in Appendix B.2

Theorem 2. *Our construction is an SVME for* $\mathfrak{R}_{\mathsf{main}}$ *according to Def. 1 as long as the underlying (i) MIBE scheme satisfies Def. 2; (ii) the polynomial commitment scheme satisfies Def. 3 and (iii) the proof systems* SP.PERM *and* SP.LEC *both satisfy Def. 4 for their resepective relations.*

Next, we provide the construction of the succinct proof system SP.LEC. The MIBE construction follows tweaking Boneh-Franklin's IBE [13]; SP.PERM is instantiated with Plonk [8] and PC with KZG [42]. We describe them in Appendix 5.1.3.

5.7 Instantiating SP.LEC

We provide an instantiation of SP.LEC in Figure 3. Recall that, this succinct proof system *links* a committed polynomial using PC (KZG commitment), and a MIBE encryption (described above) of the same (vector) form.

Note that, while a standard (non-interactive) Sigma protocol may suffice for this, it would not be succinct. Therefore, we design a proof system that combines the KZG verification and Chaum-Pedersen's proof of equality of discrete log (included in Appendix 5.5) – we call this DLEQ. The algorithms are described below (we assume that the hash functions, modeled as random oracles, have appropriate ranges):

We formally capture the properties via the following theorem, a proof for which is provided in Appendix 5.5 .

Theorem 3. SP.LEC *(Fig. 3) is a secure succinct proof system according to Def. 4 in the ROM.*

5.8 Distributed SVME

In our protocol, the encryption and key-extraction algorithms will be distributed, albeit in two different settings. We describe the settings and ideal functionalities below and the instantiation in Section 5.8.2.

- $\mathsf{SP.LEC.Setup}(1^\lambda, \mathsf{aux}) \to (\mathsf{pp}_{\mathsf{lec}}, \mathsf{td}_{\mathsf{lec}})$: Let $(\mathsf{pp}_{\mathsf{pc}}, \mathsf{td}_{\mathsf{pc}}) := \mathsf{aux}$. Set $\mathsf{pp}_{\mathsf{lec}} := \mathsf{pp}_{\mathsf{pc}}$ and $\mathsf{td}_{\mathsf{lec}} := \mathsf{td}_{\mathsf{pc}}$.
- $\mathsf{SP.LEC.Prove}(\mathsf{pp}_{\mathsf{lec}}, (M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), (\rho, \rho_m, \mathbf{m})) \to \pi_{\mathsf{lec}}$
  - $\delta := \mathsf{H}(M, \tilde{e}, \mathbf{e}, \mathbf{id}, \mathsf{mpk})$; $m(X) := \mathsf{V2P}(\mathbf{m})$; $m'(X) := m(X) + \rho_m z_\Omega(X)$; $z_m := m'(\delta)$; $W := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, m'(X), \delta)$;
  - $E := \prod_i e(\mathsf{H}'(id_i)^{\ell_i(\delta)}, \mathsf{mpk}) \cdot e(\mathsf{H}'(id_{\mathsf{pp}})^{z_\Omega(\delta)}, \mathsf{mpk})$; $T := E^\rho$; $\pi_{\mathsf{dleq}} \leftarrow \mathsf{DLEQ.Prove}((g_1, E, \tilde{e}, T), \rho)$.
  - Output: $\pi_{\mathsf{lec}} = (W, z_m, \pi_{\mathsf{dleq}})$
- $\mathsf{SP.LEC.Ver}(\mathsf{pp}_{\mathsf{lec}}, (M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), \pi_{\mathsf{lec}}) \to 1/0$. Parse: $(W, z_m, \pi_{\mathsf{dleq}}) := \pi_{\mathsf{lec}}$, and compute:
  - $\delta := \mathsf{H}(M, \tilde{e}, \mathbf{e}, \mathbf{id}, \mathsf{mpk})$; $T := (\prod_i e_i^{\ell_i(\delta)} \cdot e_{\tau+1}^{z_\Omega(\delta)})/(g_2^{z_m})$; $E := \prod_i e(\mathsf{H}'(id_i)^{\ell_i(\delta)}, \mathsf{mpk}) \cdot e(\mathsf{H}'(id_{\mathsf{pp}})^{z_\Omega(\delta)}, \mathsf{mpk})$
  - $d_{\mathsf{kzg}} := \mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, M, W, \delta, z_m)$; $d_{\mathsf{dleq}} := \mathsf{DLEQ.Ver}(\mathsf{pp}_{\mathsf{pc}}, (g_1, E, \tilde{e}, T), \pi_{\mathsf{dleq}})$; Return $d_{\mathsf{dleq}} \wedge d_{\mathsf{kzg}}$.

Figure 3: Our SP.LEC construction

5.8.1 Distributed Enc

Consider the following setting:

- There are n parties, among which an arbitrary large subset can be semi-honest corrupt. They communicate via point to point authenticated channel.
- Each party P_i holds $\tau+1$ shares $[\mathbf{m}]$ as private inputs. There are public inputs, such as pp and $(\tau+1)$-sized id-vector $\mathbf{id}$. The protocol depends on the specification of the SVME Enc algorithm.
- In the end each party gets public output $(\mathbf{c}, \pi)$.

We define the corresponding ideal functionality $\mathcal{F}_{\mathsf{SV.Enc}}$ as:

On $(\mathsf{gid}, \textsc{Encrypt}, (\mathsf{pp}, [\mathbf{m}], [\rho_{\mathsf{Enc}}], \mathbf{id}))$ from all n parties:

- Reconstruct $\mathbf{m}$ and ρ_{Enc}.
- Encrypt $(\mathbf{c}, \pi) := \mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id}; \rho_{\mathsf{Enc}})$, where $\rho_{\mathsf{Enc}} = (\rho, \rho_m)$ is the encryption randomness.
- Return $(\mathbf{c}, \pi)$ to everyone.

5.8.2 Distributed KeyExt

Let us lay out the setting:

- There are $\tilde{n}$ keypers $K_1, \ldots, K_{\tilde{n}}$, among them at most $< \tilde{n}/2$ can be maliciously corrupt. Guaranteed output delivery is required. There are also m players $P_1, \ldots, P_m$ who provide public inputs. Once the key-setup is done, the entire communication takes place over a blockchain.
- Each keyper has a private input msk_i, a $(\tilde{t}, \tilde{n})$-Shamir's share of msk of the SVME scheme. In each execution each player P_i requests for the j-th card in its deck. After each execution, a legitimate player P_i, whose legitimacy can be imposed by a smart contract, obtains a decryption key $\mathsf{sk}_{id_{i,j}}$. These executions can be easily merged to deliver the entire deck.

We define the ideal functionality $\mathcal{F}_{\mathsf{KeyExt}}$ below.

- Upon $(\textsc{SetUp}, \mathsf{pp})$ from anyone, skip if the status is $\textsc{Active}$, otherwise store pp, and change the status to $\textsc{Active}$.
- Upon $(\mathsf{gid}, \textsc{KeyExt}, j)$ from player P_i, send this message to all keypers. When at least $(\tilde{t}+1)$ keypers $\{K_k\}$ reply back with $\{\mathsf{msk}_k\}$:
  - Check if $(\mathsf{mpk}_k, \mathsf{msk}_k)$ is consistent, if not then skip; otherwise:
  - $\mathsf{sk}_{id_{i,j},k} := \mathsf{KeyExt}(\mathsf{msk}_k, id_{i,j})$, where $id_{i,j} = (\mathsf{gid}, i, j)$.

– Reconstruct $\mathsf{sk}_{id_{i,j}}$ using Lagrange in the exponent.

– Send back $\mathsf{sk}_{id_{i,j}}$ to P_i.

In the distributed setting, the encryption and key-extraction algorithms will be distributed, albeit in two different settings.

5.9 Instantiation of Distributed ENC

For our instantiation of Enc (Figure 2), we observe:

- The MIBE scheme is additively homomorphic over the message and randomness – both are distributed as witnesses. This can be easily seen from the structure: for example, for two ciphertexts $(g_1^{\rho_1}, e(\mathsf{H}(id)^{\rho_1}, \mathsf{mpk})\cdot g_2^{m_1})$ and $(g_1^{\rho_2}, e(\mathsf{H}(id)^{\rho_2}, \mathsf{mpk})\cdot g_2^{m_2})$, one can just multiply the group elements to obtain encryption of $m_1 + m_2$ as $(g_1^{\rho_1+\rho_2}, e(\mathsf{H}(id)^{\rho_1+\rho_2}, \mathsf{mpk})\cdot g_2^{m_1+m_2})$ – this uses the bilinear property. So for secret shares $[\mathbf{m}]$ and $[\rho]$ we can write $([\tilde{e}], [\mathbf{e}]) \leftarrow \mathsf{MIBE.Enc}(\cdots, [\mathbf{m}])$. Such that one can perform group operations to reconstruct $(\tilde{e}, \mathbf{e})$ from adequate number of shares $([\tilde{e}], [\mathbf{e}])$.
- Next, we note that the KZG commitments with which we instantiate PC are also homomorphic on the committed polynomial. Therefore, we can write $[P] := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, [p(X)])$, such that one can reconstruct P by linear operations.
- From [47] we observe that the Plonk proof $\mathsf{SP.PERM.Prove}$ can be implemented in a distributed fashion as well, when the witness, i.e. the polynomial $m(x)$ (plus the randomness ρ_m) is secret shared among n parties. This will be executed using an interactive protocol, which realizes an ideal functionality $\mathcal{F}_{\mathsf{Perm.Prove}}$ described as follows:
  - Upon $(\mathsf{gid}, \textsc{Perm-Prove}, ((\mathsf{pp}_{\mathsf{perm}}, M, V), ([m(X)]), [\rho_m]))$ from all n parties:
    - ∗ Reconstruct $(m(X), \rho_m)$.
    - ∗ Compute $\pi_{\mathsf{perm}} \leftarrow \mathsf{SP.PERM.Prove}(\mathsf{pp}_{\mathsf{perm}}, (M, V), (m(X), \rho_m))$.
    - ∗ Return π_{perm} to everyone.

  We refer to the paper [47] for detailed protocol.
- We design a protocol $\Pi_{\mathsf{SP.LEC.Prove}}$ in Figure 4 for distributed computation of the proof $\mathsf{SP.LEC.Prove}$ in $(\mathcal{F}_{\mathsf{DLEQ.Prove}}, \mathcal{F}_{\mathsf{Z\text{-}Share}})$-hybrid, where $\mathcal{F}_{\mathsf{DLEQ.Prove}}$ is described in Section 5.5.1and $\mathcal{F}_{\mathsf{Z\text{-}Share}}$ is described as:
  - Upon $(\mathsf{gid}, \textsc{Zero-share},)$ from all n parties, choose a random share of $[0]$ in $\mathbb{Z}_p$, and reply back with $g_1^{[0]}$.

  . One can instantiate this protocol with several existing approaches, such as everybody picking up a zero-share locally, and then adding.
- Finally let us describe the functionality $\mathcal{F}_{\mathsf{LEC.Prove}}$:
  - Upon $(\mathsf{gid}, \text{LEC-}\textsc{Prove}, ((\mathsf{pp}_{\mathsf{lec}}, M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), ([\rho], [\rho_m], [\mathbf{m}])))$ from all n parties:
    - ∗ Reconstruct $\rho, \rho_m, \mathbf{m}$.
    - ∗ Run $\pi_{\mathsf{lec}} \leftarrow \mathsf{SP.LEC.Prove}(\mathsf{pp}_{\mathsf{lec}}, (M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), (\rho, \rho_m, \mathbf{m}))$
    - ∗ Return π_{lec} to everyone.

Now we present our protocol $\Pi_{\mathsf{SV.Enc}}$ in Figure 5 in $\mathcal{F}_{\mathsf{LEC.Prove}}$-hybrid, which in turn is instantiated by protocol $\Pi_{\mathsf{SP.LEC.Prove}}$, described in Figure 4 in $(\mathcal{F}_{\mathsf{DLEQ.Prove}}, \mathcal{F}_{\mathsf{Z\text{-}Share}})$-hybrid. The security of protocol $\Pi_{\mathsf{SV.Enc}}$ and $\Pi_{\mathsf{SP.LEC.Prove}}$ are formalized via the following theorems, arguments for which are deferred to Appendix B.4 and Appendix B.5 .

Theorem 4. *The protocol $\Pi_{\mathsf{SV.Enc}}$, described in Figure 5 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{SV.Enc}}$ for arbitrary many semi-honest corruption in $\mathcal{F}_{\mathsf{LEC.Prove}}$-hybrid.*

Input: Each server S_i receives $(\mathsf{gid}, \text{LEC-PROVE}, (\mathsf{PubIP}, \mathsf{PrivIP}))$ with public input $\mathsf{PubIP} = (\mathsf{pp}_{\mathsf{lec}}, M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk})$, and private inputs $\mathsf{PrivIP} = ([\rho], [\rho_m], [\mathbf{m}])$.

Part-1 (non-interactive): Compute:

- $\delta := \mathsf{H}(M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk})$
- $[m(X)] := \mathsf{V2P}([\mathbf{m}])$.
- $[m'(X)] := [m(X)] + [\rho_m] z_\Omega(X)$
- $[z_m] := [m'(\delta)]$
- $[W] := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, [m'(X)], \delta)$
- $E :=$ $\prod_i e(\mathsf{H}'(id_i)^{\ell_i(\delta)}, \mathsf{mpk}) \cdot e(\mathsf{H}'(id_{\mathsf{pp}})^{z_\Omega(\delta)}, \mathsf{mpk})$

Part-2 (interactive): Call $\mathcal{F}_{\mathsf{Z\text{-}Share}}$ with message $(\mathsf{gid}, \text{ZERO-SHARE})$ to obtain back a secret sharing of zero in te exponent of $\mathbb{G}_1$, $[1]$ for $1 \in \mathbb{G}_1$. Let $[W] := [W] \cdot [1]$.

Part-3 (interactive): Reconstruct z_m, W by interaction.

Part-4 (interactive): Compute $T := (\prod_i e_i^{\ell_i(\delta)} \cdot e_{\tau+1}^{z_\Omega(\delta)})/(g_2^{z_m})$ and then call functionality $\mathcal{F}_{\mathsf{DLEQ.Prove}}$ by sending $(\mathsf{gid}, ((g_1, E, \tilde{e}, T), [\rho]))$ to obtain back π_{dleq}.

Output: $\pi_{\mathsf{lec}} := (W, z_m, \pi_{\mathsf{dleq}})$

Figure 4: Protocol $\Pi_{\mathsf{SP.LEC.Prove}}$ in $(\mathcal{F}_{\mathsf{DLEQ.Prove}}, \mathcal{F}_{\mathsf{Z\text{-}Share}})$-hybrid

Input: Upon $(\mathsf{gid}, \text{ENCRYPT}, (\mathsf{pp}, [\mathbf{m}], [\rho_{\mathsf{Enc}}], \mathbf{id}))$ each party S_i has public input $(\mathsf{pp}, \mathbf{id})$, and private inputs $[\mathbf{m}]$, parse $[\rho_{\mathsf{Enc}}]$ as $([\rho], [\rho_m])$.

Part-1 (non-interactive): Compute:

- $[m(X)] := \mathsf{V2P}([\mathbf{m}])$;
- $[m'(X)] := [m(X)] + [\rho_m] z_\Omega(X)$;
- $[M] := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, [m'(X)])$;
- $[\hat{\mathbf{m}}] := ([\mathbf{m}], [\rho_m])$;
- $\hat{\mathbf{id}} := (\mathbf{id}, id_{\mathsf{pp}})$;
- $([\tilde{e}], [\mathbf{e}]) := \mathsf{MIBE.Enc}(\mathsf{mpk}, \mathbf{id}, [\mathbf{m}]; [\rho])$;
- $V := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, v(X))$.

Part-2 (interactive): Reconstruct $(M, \tilde{e}, \mathbf{e})$ collectively by interaction.

Part-3 (interactive): Send $(\mathsf{gid}, \text{PERM-PROVE}, (\mathsf{pp}_{\mathsf{perm}}, ((M, V), ([m(X)], [\rho_m]))))$ to $\mathcal{F}_{\mathsf{Perm.Prove}}$ to get back π_{perm} and in parallel send $(\mathsf{gid}, \text{LEC-PROVE}, (\mathsf{pp}_{\mathsf{lec}}, ((M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), ([\mathbf{m}], [\rho_m]))))$ to $\mathcal{F}_{\mathsf{LEC.Prove}}$ to get back π_{lec}.

Output: Return $(\mathbf{c}, \pi)$ where $\mathbf{c} := (\tilde{e}, \mathbf{e})$ and $\pi := (M, V, \pi_{\mathsf{perm}}, \pi_{\mathsf{lec}})$.

Figure 5: Protocol $\Pi_{\mathsf{SV.Enc}}$ in $(\mathcal{F}_{\mathsf{Perm.Prove}}, \mathcal{F}_{\mathsf{LEC.Prove}})$-hybrid

Theorem 5. *The protocol $\Pi_{\mathsf{SP.LEC.Prove}}$, described in Figure 4 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{LEC.Prove}}$ for arbitrary many semi-honest corruption in $(\mathcal{F}_{\mathsf{DLEQ.Prove}}, \mathcal{F}_{\mathsf{Z\text{-}Share}})$-hybrid.*

5.10 Instantiation of Distributed KeyExt

We discuss the instantiation here. Note that, from our SVME construction, KeyExt algorithm is exactly the same as the MIBE.KeyExt algorithm, which is in turn the same as Boneh-Franklin's IBE scheme. Similar to threshold BLS, one can easily thresholdize this: for each i and any id if $\mathsf{sk}_{id,i} := \mathsf{KeyExt}(\mathsf{msk}_i, id)$, then sk_{id} can reconstructed by Lagrange in the exponent from any $(\tilde{t}+1)$ partial keys $\mathsf{sk}_{id,i}$.

However, one difficulty is that there is no secure channel between the players and the keypers. Communication takes place over the smart bulletin board. Over this channel, everyone can obtain any msk_{id}, if it is posted on the bulletin board. Nevertheless, this is exactly similar a recent construction put forward by [19], in that this is resolved by blinding the output with a random blind only known to the client (a similar approach is also taken in [41] in to enable output-privacy of VRFs via blinding). Furthermore, their construction also builds on the threshold BLS structure. So, here we can directly use that to instantiate the functionality $\mathcal{F}_{\mathsf{KeyExt}}$ (described in Section 5.8.2)

- This protocol will be over a smart bulletin board enabled with smart contracts that can verify:
  1. a proof of knowledge (of the blind) in the exponent verification algorithm for a player;
  2. a proof of knowledge (of the partial master secret key) in the exponent verification algorithm for a keyper.
- Each player sends $\mathsf{H}'(id)^{\nu}$ to the bulletin board, with a blind ν, along with a proof of knowledge of ν – this is verified on smart bulletin board and gets stored only when that succeeds. Note that this also ensures that a player has the legitimate id.[11]
- Each keyper, with master key share msk_i would retrieve $\mathsf{H}'(id)^{\nu\mathsf{msk}_i}$ and post that on to the smart bulletin board, along with another proof of knowledge of exponent msk_i (with respect to a $\mathsf{mpk}_i = g_1^{\mathsf{msk}_i}$). It will be then stored, only if the verification on smart bulletin board succeeds. Note that, to enable this particular verification, the setup ($\mathcal{F}_{\mathsf{Setup}}$ in Sec. 7) in the distributed setting must release partial verification keys $\mathsf{mpk}_i := g_1^{\mathsf{msk}_i}$.
- Each player now downloads the partial blinded responses, and there must be at least $\tilde{t}+1$ of them due to honest majority assumption. Then $\mathsf{H}'(id)^{\nu\mathsf{msk}}$ is constructed locally using Lagrange in the exponent from $\tilde{t}+1$ values $\mathsf{H}'(id)^{\nu\mathsf{msk}_i}$. Finally $\mathsf{sk}_{id} = \mathsf{H}'(id)^{\mathsf{msk}}$ is obtained by unblinding.

The protocol is very simple. Malicious security is guaranteed because everyone's response can be verified publicly. In fact, the simulator can extract the exponents form the zero-knowledge proofs – this can be done using a non-interactive Schnorr's proof. Guaranteed output delivery is guaranteed easily due to the honest majority assumption and the in-built verification of each individual keyper's response. This reduces to the same assumption as [41] – a threshold variant of bilinear one more Diffie-Hellman.

6 Protocol for Distributed Random Permutation

In this section, we construct an MPC protocol for generating a distributed random permutation within a small range τ. This is executed among the servers $S_1, \ldots, S_n$ – the same as for distributed Enc (cf. Section 5.8). The ideal functionality in Fig. 6 is parameterized by $\tau \in \mathbb{N}$. We provide a protocol in Figure 7, where B denotes the parameter for the number of repetitions.
Smaller values of B decrease the MPC cost (fewer parallel repetitions of $\mathcal{F}_{\mathsf{RAN}_\omega}$) but increase the likelihood that $\Pi_{\mathsf{R\text{-}Perm\text{-}Gen}}$ aborts (due to insufficient unique values). We can upper bound the probability of abort as $\tau(\frac{\tau-1}{\tau})^B$.

We describe the ideal functionalities $\mathcal{F}_{\mathsf{RAN}_\omega}$ and $\mathcal{F}_{\mathsf{DY\text{-}DPRF}}$ and their instantiations below.

[11]While the exact procedure may vary, we set $id_{i,j} = (\mathsf{gid}, i, j)$ for the j-th card of P_i.

– Upon $(\mathsf{gid}, \textsc{PermGen})$ from all servers send it to the adversary, once adversary sends back σ' execute:
  - If $|\mathcal{C}_S| < n$, sample a random permutation σ over $[\tau]$.
  - If $|\mathcal{C}_S| = n$, set $\sigma := \sigma'$ only if σ' is a permutation over $[\tau]$, else skip.
  - Let $(s_1, \ldots, s_\tau) := \mathbf{s}$ and $v_i := \omega^{s_i - 1}$.
  - Generate an additive sharing of $\mathbf{v} := (v_1, \ldots, v_\tau)$
  - Send share $[\mathbf{v}]$ to respective server.

Figure 6: Ideal functionality $\mathcal{F}_{\textsf{R-Perm-Gen}}$

– On input $(\mathsf{gid}, \textsc{PermGen})$, server S does as follows:
  - For $k \in B$ send $(\mathsf{gid}, \textsc{Invoke})$ to $\mathcal{F}_{\textsf{RAN}_\omega}$ (in parallel). Let $[\gamma_k]$ be the shared private output of the k-th iteration where $k \in [B]$. So, at the end of this step, the servers collectively hold $(\gamma_1, \ldots, \gamma_B)$ in additive secret-sharing.
  - Send $(\mathsf{gid}, \textsc{Key-Setup})$ to $\mathcal{F}_{\textsf{DY-DPRF}}$. For each $k \in [B]$ send $(\mathsf{gid}, \textsc{Eval}, [\gamma_k])$ to $\mathcal{F}_{\textsf{DY-DPRF}}$ (in parallel). Receive the outputs $Y = (y_1, \ldots, y_B)$, where y_k is the output of the k-th invocation.
  - If there are less than τ unique y_k in Y, then abort. Otherwise construct the ordered set $\{[\gamma_k]\}_{k \in I}$ of size $|I| = \tau$ by selecting τ-many $[\gamma_k]$ corresponding to *first* τ unique y_k from Y (each y_k corresponds to a $[\gamma_k]$ for known k). Denote $[\mathbf{v}] := ([v_1], \ldots, [v_\tau])$, where $v_i = \gamma_k$ and k is the i-th index in I. Let us call this *sanitization*.
  - Return $[\mathbf{v}]$

Figure 7: $\Pi_{\textsf{R-Perm-Gen}}$ in $(\mathcal{F}_{\textsf{RAN}_\omega}, \mathcal{F}_{\textsf{DY-DPRF}})$-hybrid

- $\mathcal{F}_{\textsf{RAN}_\omega}$: Upon $(\mathsf{gid}, \textsc{Invoke})$ from all n parties, sample $\gamma \leftarrow_\$ \{1, \omega, \ldots, \omega^{\tau-1}\}$, compute an additive sharing $[\gamma]$, send back the respective shares to the parties.

- $\mathcal{F}_{\textsf{DY-DPRF}}$:

  – Upon $(\mathsf{gid}, \textsc{Key-Setup})$ from all n parties sample a secret key $\mathsf{sk} \leftarrow_\$ \mathbb{Z}_p$. Store $(\mathsf{gid}, \mathsf{sk})$ and mark gid.

  – Upon $(\mathsf{gid}, \textsc{Eval}, [x])$ from all n servers, unless gid is marked skip, else reconstruct $x \in \mathbb{Z}_p$. Then return $y := g^{(sk+x)^{-1}}$ to everyone.

6.1 Protocols $\Pi_{\textsf{RAN}_\omega}$ and $\Pi_{\textsf{DY-DPRF}}$

$\mathcal{F}_{\textsf{RAN}_\omega}$ and $\mathcal{F}_{\textsf{DY-DPRF}}$ are instantiated by protocols $\Pi_{\textsf{RAN}_\omega}$ and $\Pi_{\textsf{DY-DPRF}}$ respectively.

Protocol $\Pi_{\textsf{RAN}_\omega}$:

1. Upon $(\mathsf{gid}, \textsc{Invoke})$ a server S:
2. Locally sample share $[x]$.
3. Sends $(\mathsf{gid}, \textsc{Exp-Reveal}, [x])$ to $\mathcal{F}_{\textsf{EXP-REVEAL}}$ to receive back x^τ.
4. If $x^\tau = 0 \bmod p$, then restart from Step 2 above. Otherwise output $\left(\sqrt[\tau]{x^\tau}\right)^{-1} \cdot [x]$

Where:

Functionality $\mathcal{F}_{\textsf{EXP-REVEAL}}$:

- Upon $(\mathsf{gid}, \textsc{Exp-Reveal}, [a])$ from all n parties, reconstruct a and send $a^\tau \bmod p$ to all parties and the adversary.

Protocol $\Pi_{\textsf{DY-DPRF}}$:

1. Upon $(\mathsf{gid}, \textsc{Key-Setup})$ if gid is *not* marked $\textsc{Live}$ skip, else each server locally samples a share $[\mathsf{sk}]$ uniformly at random from $\mathbb{Z}_p$. Store $(\mathsf{gid}, [\mathsf{sk}])$. Mark gid $\textsc{Key-Setup-Done}$.
2. Upon $(\mathsf{gid}, \textsc{Eval}, [x])$, unless gid is marked $\textsc{Key-Setup-Done}$ skip, else each server executes the following:
   (a) Send $(\mathsf{gid}, \textsc{Invert}, [x] + [\mathsf{sk}])$ to $\mathcal{F}_{\mathsf{INV}}$ to get back $g^{[\hat{x}]}$.
   (b) Set $[y] := g^{[\hat{x}]}$ and collaboratively reconstruct y; and return y

Where:

Functionality: $\mathcal{F}_{\mathsf{INV}}$

- Upon $(\mathsf{gid}, \textsc{Invert}, [x])$ from all n servers, if gid is unmarked skip, else reconstruct x, then compute the inverse $y = x^{-1}$ in $\mathbb{Z}_p$. Compute random additive sharing $[y]$. Send $g^{[y]}$ to the respective server.

6.2 Protocols $\Pi_{\mathsf{EXP\text{-}REVEAL}}$ and Π_{INV}

Now we instantiate $\mathcal{F}_{\mathsf{EXP-REVEAL}}$ by protocol $\Pi_{\mathsf{EXP\text{-}REVEAL}}$ and $\mathcal{F}_{\mathsf{INV}}$ by Π_{INV}.

Protocol $\Pi_{\mathsf{EXP\text{-}REVEAL}}$:

1. On $(\mathsf{gid}, \textsc{Exp-Reveal}, [a])$ each party initializes $[x_0] = [a]$ and executes the following loop for $k \in \{1, \ldots, \log(\tau)\}$
   – Send $(\mathsf{gid}, \textsc{Mult}, [x_{k-1}], [x_{k-1}])$ to $\mathcal{F}_{\mathsf{MULT}}$ and get back $[x_k]$ as output.
2. Once the loop is completed collaboratively reconstruct x_e, and return that.

Where:

Functionality $\mathcal{F}_{\mathsf{MULT}}$:

- Upon $(\mathsf{gid}, \textsc{Mult}, [x], [y])$ from all n parties, if gid is unmarked skip, otherwise send $(\mathsf{gid}, \textsc{Mult})$ to the adversary.
  1. Compute $z = xy \bmod p$, then compute an additive sharing $[z]$.
  2. Send z_i to each honest party i.

Finally we present Π_{INV} in $\mathcal{F}_{\mathsf{MULT}}$-hybrid.

Protocol Π_{INV}:

- On $(\mathsf{gid}, \textsc{Invoke}, [s])$ if gid is unmarked skip, otherwise each server S_i executes:
  1. Locally sample a share of uniform random value $[r]$ in $\mathbb{Z}_p$.
  2. Send $(\mathsf{gid}, \textsc{Mult}, [s], [r])$ to $\mathcal{F}_{\mathsf{MULT}}$ to get back $[u]$ then reconstruct collboratively u.
  3. Compute $[s^{-1}] := u^{-1} \cdot [r]$.
  4. Return $g^{[s^{-1}]}$

We note that $\mathcal{F}_{\mathsf{MULT}}$ can be realized by any MPC protocol. We omit an instantiation. In our implementation we use a variant of SPDZ [27].

We capture the security of $\Pi_{\mathsf{R\text{-}Perm\text{-}Gen}}$ via the following theorem, a proof of which is deferred to Appendix B.6.

Theorem 6. *The protocol* $\Pi_{\mathsf{R\text{-}Perm\text{-}Gen}}$ *securely realizes* $\mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}$ *in* $(\mathcal{F}_{\mathsf{RAN}_\omega}, \mathcal{F}_{\mathsf{DY\text{-}DPRF}})$*-hybrid.*

Parameters. This is specifically parameterized with SVME verification algorithm Ver (uploaded as smart contracts). It has public storage and is visible to everyone.
– Upon $(\textsc{SetUp}, \mathsf{pp})$, skip if the status is $\textsc{Active}$ otherwise send this to all parties, if everyone approves, then (publicly) store pp and change status to $\textsc{Active}$.
– Upon $(\mathsf{gid}, \textsc{Server}, \mathsf{msg})$ from a server S_i, skip if gid is marked $\textsc{Sampled}$ else send this to all servers, once they respond back with approval, if $\mathsf{msg} = (\mathbf{id}, \mathbf{c}, \pi)$ verify $\mathsf{Ver}(\mathsf{pp}, (\mathbf{id}, \mathbf{c}, \pi)$, if that succeeds, store $(\mathsf{gid}, (\mathbf{id}, \mathbf{c}, \pi))$ and mark gid $\textsc{Sampled}$, if $\mathsf{msg} = \perp$, then store $(\mathsf{gid}, \perp)$ and mark gid $\textsc{Sampling-Failed}$.

Figure 8: Functionality $\mathcal{F}_{\mathsf{SBB}}$, which captures smart bulletin board

7 Insta-Pok3r: Definition and Construction

Finally we are ready to present Insta-Pok3r – this is formally captured by $\mathcal{F}_{\mathsf{VES}}$ (Fig. 9), which interacts with m players $P_1, \ldots, P_m$, n servers $S_1, \ldots, S_n$, $\tilde{n}$ keypers $K_1, \ldots, K_{\tilde{n}}$, a public verifier V and the ideal adversary. It has four interfaces/phases:

1. **Setup.** This is run once to setup keys, and public parameters – only the keypers and servers are involved (not players, who arrive only in the online phase). One setup serves multiple games. Of course, in practice, we may need to redo this for rotation, etc., but for simplicity, we assume there is a single setup. We do not instantiate this.
2. **Sample (offline).** In the Sample (offline) phase only servers are involved in generating a verifiably encrypted random permutation. Each Sample (offline) phase is consumed by a single Retrieve (online) phase.
3. **Retrieve (online).** In this phase players retrieve their cards with the help of keypers.
4. **Public Verification.** At any point, once the Sample (offline) phase is completed, anyone, including a public verifier can verify whether the deck is correct.

Smart bulletin board. In our setting, different groups of parties communicate over a smart-contract-enabled blockchain. We describe a simple ideal functionality $\mathcal{F}_{\mathsf{SBB}}$ in Fig. 8 to formally capture that. Note that, this is specific to our requirement, and can be easily adapted to other requirements.

Parameters. It is parameterized with an integer $\tau \in \mathbb{N}$, where τ denotes the range of the permutation, and a status flag initialized with $\textsc{InActive}$.

Ideal Functionality $\mathcal{F}_{\mathsf{VES}}$

1. **Setup.** Upon $\textsc{SetUp}$ from all servers and all players: if the status is $\textsc{InActive}$, forward it to the adversary; else skip. When the adversary approves then set the status to $\textsc{Active}$ and send $\textsc{Active}$ to everyone.
2. **Sample (offline).** Upon $(\mathsf{gid}, \textsc{Sample})$ from all servers skip unless the status is $\textsc{Active}$, else send it to the adversary to get back a response. If the adversary replies with $\perp$, then mark gid $\textsc{Sampling-Failed}$ and send $(\mathsf{gid}, \perp)$ to the honest parties. Else if adversary's response is $(\mathsf{gid}, \mathbf{s}')$:
   (a) If $|\mathcal{C}_S| < n$ then sample a random permutation $\mathbf{s} = (s_1, \ldots, s_\tau)$ over $[\tau]$ (that is, $s_i \in [\tau]$).
   (b) Else when $|\mathcal{C}_S| = n$, then if $\mathbf{s}'$ is not a permutation, then mark gid $\textsc{Sampling-Failed}$ and send $(\mathsf{gid}, \perp)$ to the honest parties and exit. Else set $\mathbf{s} := \mathbf{s}'$.
   Mark gid $\textsc{Sampled}$ and send $(\mathsf{gid}, \textsc{Sampled})$ to all servers $S_1, \ldots, S_n$. Store $(\mathsf{gid}, \mathbf{s})$.
3. **Retrieve (online):** Upon $(\mathsf{gid}, \textsc{Card}, j)$ from P_i: skip if gid is not marked $\textsc{Sampled}$, else send that to the adversary and when the adversary returns the same message then forward the message to the keypers. When at least $(\tilde{t} + 1)$ keypers send approvals, retrieve $(\mathsf{gid}, \mathbf{s})$ and send $(\mathsf{gid}, s_{i,j})$ to player P_i.
4. **Public Verification.** Upon $(\mathsf{gid}, \textsc{Verify})$ from any party including V: if gid is marked $\textsc{Sampled}$, then reply $\textsc{Success}$; else if marked $\textsc{Sampling-Failed}$ reply $\textsc{Failure}$; in all other cases skip.

Figure 9: Ideal Functionality $\mathcal{F}_{\mathsf{VES}}$

Parameters. It is parameterized with the permutation range τ and a field element $\omega \in \mathbb{Z}_p$ of order τ. *(Parameters are common for all (sub-)functionalities (and (sub-)protocols) recursively called; we omit restating them further).*

Protocol Π_{VES}

1. **Setup.** On input $\textsc{SetUp}$ to a server or a player, if status is $\textsc{Inactive}$:
   - Each party forwards this to $\mathcal{F}_{\mathsf{Setup}}$. A keyper K_i gets back a private msk_i. Everyone gets back a public parameter pp of SVME, then it sends $(\textsc{SetUp}, \mathsf{pp})$ to $\mathcal{F}_{\mathrm{SBB}}$ and $\mathcal{F}_{\mathsf{KeyExt}}$. When $\mathcal{F}_{\mathrm{SBB}}$ seeks approval message for pp, send approval. Change status to $\textsc{Active}$.
2. **Sample (offline).** On input $(\mathsf{gid}, \textsc{Sample})$ to a server, skip unless the status is $\textsc{Active}$ *and* gid is unmarked:
   - Send $(\mathsf{gid}, \textsc{PermGen})$ to $\mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}$ to obtain back $[\mathbf{v}]$.
   - Set $\mathbf{id} := (id_1, \ldots, id\tau)$ where each $id_i := (\mathsf{gid}, i)$
   - Sample $[\rho_{\mathsf{Enc}}]$ uniformly at random.
   - Send $(\mathsf{gid}, \textsc{Encrypt}, (\mathsf{pp}, [\mathbf{v}], [\rho_{\mathsf{Enc}}], \mathbf{id}))$ to $\mathcal{F}_{\mathsf{SV.Enc}}$ to get back $(\mathbf{c}, \pi)$.
   - Verify $d := \mathsf{Ver}(\mathsf{pp}, \mathbf{c}, \mathbf{id}, \pi)$.
     - If $d = 1$: Mark gid $\textsc{Sampling-Failed}$ send $(\mathsf{gid}, \textsc{Server}, (\mathsf{gid}, \perp))$ to $\mathcal{F}_{\mathrm{SBB}}$.
     - If $d = 0$: Mark gid $\textsc{Sampled}$ and send $(\mathsf{gid}, \mathbf{id}, \mathbf{c}, \pi)$ to $\mathcal{F}_{\mathrm{SBB}}$.
   - Whenever $\mathcal{F}_{\mathrm{SBB}}$ seeks for a correct approval, send approval.
3. **Retrieve (online).** On input $(\mathsf{gid}, \textsc{Card}, j)$ a player P_i, skip unless gid is marked $\textsc{Sampled}$, else:
   (a) Send $(\mathsf{gid}, \textsc{KeyExt}, j)$ to $\mathcal{F}_{\mathsf{KeyExt}}$. On receiving the message from $\mathcal{F}_{\mathsf{KeyExt}}$, each keyper K_k sends msk_k.
   (b) Player P_i obtains $\mathsf{sk}_{id_{i,j}}$ then decrypts $v_{i,j} := \mathsf{Dec}(\mathsf{sk}_{id_{i,j}}, (c_{\mathsf{aux}}, c_{i,j}))$.
   (c) P_i outputs $s_{i,j}$ such that $v_{i,j} = \omega^{s_{i,j}}$ (note that, $s_{i,j} \in [\tau]$).
4. **Public Verify.** On input $(\mathsf{gid}, \textsc{Verify})$ to any party it retrieves $(\mathsf{gid}, \mathbf{id}, \mathbf{c}, \pi)$ from $\mathcal{F}_{\mathrm{SBB}}$, and returns $\mathsf{Ver}(\mathsf{pp}, \mathbf{c}, \mathbf{id}, \pi)$.

Figure 10: Our Main Protocol Π_{VES}, aka Insta-Pok3r, for sampling verifiable encrypted randomness

Setup. We describe the ideal functionality $\mathcal{F}_{\mathsf{Setup}}$:

- Upon $\textsc{SetUp}$ from all keypers:
  1. Run $(\mathsf{pp}, \mathsf{msk}, \mathsf{td}_{\mathsf{svme}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, where pp contains $\mathsf{mpk}_i = g_1^{\mathsf{msk}_i}$ for all $i \in [\tilde{n}]$.
  2. Create a random $\tilde{t}$ out of $\tilde{n}$ Shamir sharing of msk: $(\mathsf{msk}_1, \ldots, \mathsf{msk}_{\tilde{n}})$. Send msk_i to all each K_i and pp to everyone.

We describe Π_{VES} in Fig. 10 and formalize the security by the following theorem; a proof sketch is provided in appendix B.7.

Theorem 7. *The protocol Π_{VES} (Fig. 10) securely realizes $\mathcal{F}_{\mathsf{VES}}$ (Fig. 9) in $(\mathcal{F}_{\mathsf{Setup}}, \mathcal{F}_{SBB}, \mathcal{F}_{\mathsf{KeyExt}}, \mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}, \mathcal{F}_{\mathsf{SV.Enc}})$-hybrid.*

8 Empirical Evaluation

	Total MPC time				Player time
Size	$n = 4$	$n = 8$	$n = 16$	$n = 20$	all n
64	1.54	2.97	7.86	12.08	0.28
128	2.99	5.85	15.31	23.32	0.44
256	6.70	13.39	35.68	56.16	0.63

Figure 11: MPC and player running Time (in seconds) for varying permutation sizes and number of servers n. To account for collisions, we use $B = 400$, 800, 2000 samples for 64, 128, 256 cards, respectively.

Protocol	$n = 4$	$n = 8$	$n = 16$	$n = 20$
`beaver`	0.03	0.08	0.16	0.22
`shuffle`	0.65	1.41	4.36	7.20
`perm proof`	0.12	0.22	0.82	1.36
`encryption`	0.67	1.13	2.19	2.79
`ctxt proof`	0.05	0.1	0.21	0.30

Figure 12: Running Time (secs) for sub-protocols in the MPC phase, for varying number of servers and 64 cards.

We implement our entire system – i.e., the Π_{VES} protocol and all its sub-protocols – in Rust, building upon the `arkworks` libraries for finite fields, elliptic curves, and pairings, and the `libp2p` libraries for networking. Specifically, we use the BLS12-381 pairing-based curve [25], and the hashing to elliptic curve method defined in [34]. Our system is open-sourced at `https://github.com/rsinha/pok3r`.

For the MPC protocols in the shuffling phase, we use a dishonest majority MPC protocol based on additive secret sharing. We use the Low Gear [43] protocol for generating Beaver triples. We reiterate that our protocol can be implemented using a myriad of choices for the MPC protocol, and honest majority protocols can have even better performance (but require twice as many servers for the same corruption threshold).

While our protocols have several opportunities for parallelism, we avoid implementing them for two reasons: 1) we find it simpler to use the available CPU cores in concurrent instances of the MPC protocol, to shuffle decks in service of the concurrently running poker games; 2) by using a single processor for the duration of the protocol, we were able to do an apples-to-apples comparison while evaluating various choices in our protocol's design. For applications that need a single instance of a permutation, one can avail the various opportunities for parallelism to shorten the latency.

Experimental Setup

For evaluating the MPC phase, we use a cluster of 20 AWS c4.xlarge machines, with 4 vCPUs and 8 GB RAM each. We allocate them in the same AWS region, by instantiating a virtual private cloud (VPC) to place them in the same LAN subnet. The round-trip time between any two machines was found to be under 10 ms. We use a Macbook Pro for measuring the running time of the player's computation.

8.1 MPC Running Time

The end-to-end latency of the MPC phase is reported in Figure 11, where the permutation size is varied between 64 and 256, while the number of MPC servers ranges from 4 to 20.

We also measure the running time by the various sub-protocols that comprise the MPC phase. These are illustrated in Figure 12, where we vary the number of MPC servers between 4 and 20, while shuffling a deck of 64 cards. Observe that we use more samples (denoted B) than the permutation size (denoted τ), as a balancing act between efficiency and failure probability – in the event of failure, we simply run the protocol again. [12]

Our protocol consumes 3314, 7426, and 20050 Beaver triples for $B = 400$, 800, and 2000 respectively. As reported in Figure 12, we find that the computation to generate the triples is a relatively small fraction of the total running time.

The `shuffle` protocol includes the time to sample a shared, uniformly random permutation, and it takes roughly 65% of the total running time. Shuffling is expensive mainly for having to compute several extra samples to account for collisions – each sample requires $O(n)$ group operations (for the the Dodis-Yampolski PRF) in addition to the MPC multiplications.

[12]Here, failure is the case when we do not obtain all τ cards after sampling B cards. Explicitly, we can use the union bound to upper bound this probability as $\tau \cdot \left(\frac{\tau-1}{\tau}\right)^B$.

The protocol for computing the PLONK-style permutation argument is denoted by `perm check`, and it takes roughly 8% of the total running time. This step is relatively inexpensive because it only computes 3 polynomial commitments and 5 opening proofs, albeit distributed and therefore linear in n. That is, by encoding this check in terms of polynomial identities, we avoid incurring cost linear in the size of the permutation. The `encryption` protocol involves $O(n)$ pairing operations – 3 G_T group operations for each card – which results in about 25% of the total running time. Finally, `ctxt proof` denotes the sub-protocol which computes the LEC proof, and it occupies under 5% of the running time.

Not surprisingly, for all of the sub-protocols above, the running time grows linearly with the number of MPC servers, as we generally rely on homomorphism to aggregate group elements from all servers.

8.2 Player Running Time

The player's computation refers to decrypting the ciphertext corresponding to one card and verifying the proofs for the permutation argument and ciphertext validity argument. For a deck of 64 cards, we find the running time to be approximately 0.3 seconds, which is independent of the number of MPC servers, as expected and desired.

8.3 On-chain Space Requirement

For each hand of poker, we place the encrypted cards and the associated proofs on the chain. The breakdown is as follows:

1. **IBE ciphertexts**: 53 $\mathbb{G}_T$ elements and 1 $\mathbb{G}_2$ element, totalling 20448 bytes
2. **permutation argument**: 8 $\mathbb{G}_1$ elements and 5 $\mathbb{F}$ elements, totalling 544 B
3. **ciphertext validity (SP.LEC) argument**: 2 $\mathbb{G}_T$ elements, 1 $\mathbb{G}_2$ element, 1 $\mathbb{G}_1$ element, and 3 $\mathbb{F}$ elements, totalling 1008 B

The entire ciphertext is 21.5 KB in size and is independent of the number of MPC servers. In practice, to save on gas costs, we posit that the ciphertext we placed on cheaper mediums, such as on layer 2 side-chains or data availability layers, e.g. [1].

8.4 Discussion

We demonstrated some attractive aspects of our construction. Specifically, we showed that the ciphertext size and player's computation are independent of the number of MPC servers, as one would want from a poker service.

However, we find that the concrete running times are on the larger side, and studying further performance improvements is a valid exercise for future work. Specifically, we find that the dominant cost in all the sub-protocols is the need to do $O(n)$ group operations to aggregate "contributions" from all MPC servers, as each of them only operates over a share of any secret value in the protocol. This is a fundamental characteristic of our MPC protocols.

9 Conclusion and Future Work

This paper addresses the need for verifiable encrypted shuffle in Web3 gaming. It introduces a framework for generating publicly verifiable pseudorandom permutations. Our approach combines multiparty computation and distributed checks while optimizing for on-chain efficiency. The MPC-as-a-Web3-service concept introduced in the paper can be of interest to a few other MPC tasks. In the future, it can interesting to explore privacy-preserving computation tasks such as secure learning/inference.

References

[1] Avail: The Unification Layer for Web3. https://docs.availproject.org/docs/introduction-to-avail.

[2] Game3r. https://gam3r.org/.

[3] Geometry Research: Mental Poker. https://github.com/geometryresearch/mental-poker.

[4] Supra Whitepapers. https://supraoracles.com/whitepapers/.

[5] ZKShuffle: Mental Poker on SNARK for Ethereum. https://zkholdem.xyz/wp-content/themes/zkholdem-theme/zkshuffle.pdf.

[6] NBC: Poker site cheating plot a high-stakes whodunit. https://www.nbcnews.com/id/wbna26563848, 2008.

[7] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder - scalable, robust anonymous committed broadcast. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1233–1252. ACM Press, November 2020.

[8] Ariel Gabizon and Zachary J. Williamson and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. https://eprint.iacr.org/2019/953.

[9] Adam Barnett and Nigel P. Smart. Mental poker revisited. In *Cryptography and Coding*, pages 370–383, 2003.

[10] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. Cryptology ePrint Archive, Paper 2014/075, 2014. https://eprint.iacr.org/2014/075.

[11] Rohann Bella, Xavier Bultel, Céline Chevalier, Pascal Lafourcade, and Charles Olivier-Anclin. Practical construction for secure trick-taking games even with cards set aside. In *Financial Cryptography and Data Security (FC)*, 2023.

[12] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 410–440. Springer, Heidelberg, December 2017.

[13] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[14] Dan Boneh and Victor Shoup. A graduate course in applied cryptography.

[15] Xavier Bultel and Pascal Lafourcade. Secure trick-taking game protocols - how to play online spades with cheaters. In *Financial Cryptography and Data Security FC*, pages 265–281, 2019.

[16] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 331–345. Springer, Heidelberg, December 2000.

[17] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[18] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

[19] Andrea Cerulli, Aisling Connolly, Gregory Neven, Franz-Stefan Preiss, and Victor Shoup. vetkeys: How a blockchain can keep many secrets. Cryptology ePrint Archive, Paper 2023/616, 2023. https://eprint.iacr.org/2023/616.

[20] Chainlink. Chainlink VRF: On-Chain Verifiable Randomness. https://developer.wax.io/en/tutorials/create-wax-rng-smart-contract/rng_basics.html.

[21] David Chaum. Verification by anonymous monitors. In Allen Gersho, editor, *CRYPTO'81*, volume ECE Report 82-04, pages 138–139. U.C. Santa Barbara, Dept. of Elec. and Computer Eng., 1981.

[22] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[23] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.

[24] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 321–338. IEEE Computer Society, 2015.

[25] D. Boneh and S. Gorbunov and R. Wahby and H. Wee and C. Wood and Z. Zhang. BLS Signatures. draft-irtf-cfrg-bls-signature-05, 2022. `https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-bls-signature-05`.

[26] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, Heidelberg, March 2006.

[27] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[28] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. Verifiable secret sharing simplified. Cryptology ePrint Archive, Paper 2023/1196, 2023. `https://eprint.iacr.org/2023/1196`.

[29] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

[30] Yevgeniy Dodis, Aleksandr Yampolskiy, and Moti Yung. Threshold and proactive pseudo-random permutations. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 542–560. Springer, Heidelberg, March 2006.

[31] DRand. DRand - Distributed Randomness Beacon. `https://drand.love/`.

[32] Eric James Beyer. OpenSea's Insider Trading Case Is a Wake-Up Call for Web3. `https://nftnow.com/features/openseas-insider-trading-case-is-a-wake-up-call-for-web3/`.

[33] Saba Eskandarian and Dan Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. In *NDSS*. The Internet Society, 2022.

[34] A. Faz-Hernandez, S. Scott, N. Sullivan, R.S. Wahby, and C.A. Wood. Hashing to Elliptic Curves. Online, 2022. `https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/12/`.

[35] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[36] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *ACM STOC*, pages 365–377, 1982.

[37] Philippe Golle. Dealing cards in poker games. In *International Symposium on Information Technology: Coding and Computing (ITCC 2005), Volume 1,*, pages 506–511, 2005.

[38] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

[39] Jens Groth. Non-interactive distributed key generation and key resharing. *Cryptology ePrint Archive*, 2021.

[40] Sanket Kanjalkar, Ye Zhang, Shreyas Gandlur, and Andrew Miller. Publicly auditable mpc-as-a-service with succinct verification and universal setup. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 386–411. IEEE, 2021.

[41] Aniket Kate, Easwar Vivek Mangipudi, Siva Mardana, and Pratyay Mukherjee. FlexiRand: Output Private (Distributed) VRFs and Application to Blockchains. Cryptology ePrint Archive, Paper 2023/1339 (To appear in ACM CCS 2023), 2023. `https://eprint.iacr.org/2023/1339`.

[42] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010.

[43] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 158–189. Springer, Heidelberg, April / May 2018.

[44] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 195–206. ACM Press, October 2015.

[45] Donghang Lu and Aniket Kate. RPM: robust anonymity at scale. *Proc. Priv. Enhancing Technol.*, 2023(2):347–360, 2023.

[46] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew K. Miller. HoneyBadgerMPC and AsynchroMix: Practical asynchronous MPC and its application to anonymous communication. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 887–903. ACM Press, November 2019.

[47] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 4291–4308. USENIX Association, August 2022.

[48] A. Shamir, R. L. Rivest, and L. M. Adleman. Mental poker. Technical report LCS/ TM-125, Massachusetts Institute of Technology, 1979.

[49] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[50] Simran Jagdev. KZG Ceremony: Participate and Help Build Ethereum. Online, 2023. `https://consensys.io/blog/kzg-ceremony-participate-and-help-build-ethereum`.

[51] Heiko Stamer. Efficient electronic gambling: An extended implementation of the toolbox for mental card games. In *WEWoRC 2005 - Western European Workshop on Research in Cryptology*, volume P-74, pages 1–12, 2005.

[52] Akira Takahashi and Greg Zaverucha. Verifiable encryption from MPC-in-the-head. Cryptology ePrint Archive, Report 2021/1704, 2021. `https://eprint.iacr.org/2021/1704`.

[53] The Tor Project. https://www.torproject.org/. Accessed in Nov 2018.

[54] Valeria Nikolaenko and Sam Ragsdale and Joseph Bonneau and Dan Boneh. Powers-of-Tau to the People: Decentralizing Setup Ceremonies. Cryptology ePrint Archive, Paper 2022/1592, 2022. `https://eprint.iacr.org/2022/1592`.

A Additional Preliminaries

A.1 Universal Composability

We provide a brief description of Canetti's universal composability framework [17] – this is taken almost verbatim from [41].

In the UC framework, a PPT algorithm called the environment (which is adversarial) is trying to distinguish between a real and an ideal world. The adversary in the protocol can corrupt parties in the real world, whereas an ideal adversary, controls the corrupt parties in the ideal world. The ideal world comprises an ideal functionality (a.k.a. trusted third party) that is directly connected to all the parties plus the ideal adversary. The honest ideal world parties are called dummy parties because they are just interfaces between the environment and the ideal functionality. The real world does not have the trusted party, and the protocol is designed so that parties interacts to achieve the same effect as the ideal world. For proving a protocol *securely realizes* an ideal functionality, for every real world adversary we need to design a simulator in the ideal world, which acts as an ideal adversary and internally simulates the real world to a real world adversary such that no environment providing inputs to and observing the outputs from the computing entities can distinguish between the real world and the ideal world, given the adversary's view of both worlds. The simulator typically simulates the real world to an instance of the real-world adversary by providing messages on behalf of the honest parties while accessing the ideal functionality and finally outputs whatever the adversary outputs. We also use hybrid protocols, where part of the protocols are computed by a less powerful ideal functionality. In that case, the simulator also needs to simulate those ideal (sub-)functionalities. The simulator can schedule messaging and outputs in the ideal world to prevent trivial distinctions by timing. All entities are formally modeled as instances of an interactive Turing machine, or ITI. For a detailed formalization, we refer to [17, 18].

B Proofs

B.1 Proofs for $\Pi_{\mathsf{R\text{-}Perm\text{-}Gen}}$

In this section we provide proof sketches for relevant theorems about $\Pi_{\mathsf{R\text{-}Perm\text{-}Gen}}$.

Theorem 8. *The protocol $\Pi_{\mathsf{RAN}_\omega}$ securely realizes $\mathcal{F}_{\mathsf{RAN}_\omega}$ in the $\mathcal{F}_{\mathsf{EXP-REVEAL}}$-hybrid model.*

Proof Sketch. We describe a simulator which simulates the honest party's response plus the ideal functionality $\mathcal{F}_{\mathsf{EXP-REVEAL}}$'s response to the adversary. First, we observe that , when all parties are corrupt then the simulation is trivial, as it has full control over x. So we only focus on the case when $|\mathcal{C}_S| < n$, and the simulator does not know x. The simulator, in that case, works as follows:

- Receive query $[x]$ to $\mathcal{F}_{\mathsf{EXP-REVEAL}}$ for all corrupt parties from the adversary. Denote party i's ($i \in \mathcal{C}_S$) share $[x]$ as x_i. For each $i \in \mathcal{C}_S$, choose a uniform random $c_i \leftarrow_\$ \mathbb{Z}_p$. Then compute x^τ from the equation: $c_i = \left(\sqrt[\tau]{x^\tau}\right)^{-1} \cdot x_i$ and check if $x^\tau = 0 \mod p$, if not then reply back to the adversary as a response. Otherwise ask the adevrsary to restart.
- Finally send $\{c_i\}_{i\in\mathcal{C}_S}$ to $\mathcal{F}_{\mathsf{RAN}_\omega}$.

Essentially, the simulator here is leveraging the power of choosing its own share for the uniform random power of ω. And we note that, since x^τ is correctly computed using the actual equation, this holds for all $i \in [n]$. □

Theorem 9. *The protocol $\Pi_{\mathsf{DY\text{-}DPRF}}$ securely realizes $\mathcal{F}_{\mathsf{DY\text{-}DPRF}}$ in $\mathcal{F}_{\mathsf{INV}}$-hybrid.*

Proof (sketch). Intuitively the protocol is secure due to secret sharing. One can construct a simulator which gets y, and then creates a random additive sharing $[y]$, which it replies to emulate $\mathcal{F}_{\mathsf{INV}}$. □

Theorem 10. *The protocol $\Pi_{\mathsf{EXP\text{-}REVEAL}}$ securely realizes $\mathcal{F}_{\mathsf{EXP-REVEAL}}$ in $\mathcal{F}_{\mathsf{MULT}}$-hybrid.*

Proof (sketch). We describe a simulator which simulates the honest party's response plus the ideal functionality $\mathcal{F}_{\mathsf{MULT}}$'s response to the adversary. First, we observe that, when all parties are corrupt then the simulation is trivial, as it has full control over a. So we only focus on the case when $|\mathcal{C}_S| < n$, and the simulator does not know a. The simulator, in that case, works as follows:

- When the simulator receives the first call to $\mathcal{F}_{\mathsf{MULT}}$ from adversary, it gets a query $(\mathsf{gid}, \textsc{Mult}, [a], [a])$ for the corrupt servers. At this point the simulator sends $(\mathsf{gid}, \textsc{Exp-Reveal}, [a])$ to $\mathcal{F}_{\mathsf{EXP-REVEAL}}$ for each $i \in \mathcal{C}_S$ to get back $x_e = a^\tau$ in the clear.
- Then the simulator computes $x'_{e-1} := \sqrt{x_e}, x'_{e-2} := \sqrt{x'_{e-1}}, \dots, x'_1 := \sqrt{x'_2}$ by taking repeated square roots over $\mathbb{Z}_p$, and choosing *any* of the two roots at each step.
- Finally, constructs a random sharing of x'_1 for all corrupt parties and send back those in response to the query $(\mathsf{gid}, \textsc{Mult}, [a], [a])$ made in Step B.1.
- Then, once it receives the next query $(\mathsf{gid}, \textsc{Mult}, [x'_1], [x'_1])$ reply with random shares of x'_2 and continue like this until it reaches the final step.

To see the correctness of the simulation, observe that, each step the adversary receives only random shares, so even if it happens that the repeated square root does not lead to the actual value a (for example, $a^2 \neq x'_1$), it would not be detected by the adversary due to the additive sharing (and since $|\mathcal{C}_S| < n$). Eventually, the only value adversary sees in the clear is $x_e = a^\tau$, which is correctly computed. □

Theorem 11. *The protocol* Π_{INV} *securely realizes* $\mathcal{F}_{\mathsf{INV}}$ *in* $\mathcal{F}_{\mathsf{MULT}}$*-hybrid.*

Proof (sketch). A simulator emulates $\mathcal{F}_{\mathsf{MULT}}$ with uniform random values. This implicitly define s and r. Finally it obtains the $g^{[s^{-1}]}$ from the functionality $\mathcal{F}_{\mathsf{INV}}$, which it sends to the adversary. Since $s.r$ information theoretically hides both s and r when at most $n-1$ servers are corrupt, this does not leak individual values. □

B.2 SVME Security (Theorem 2)

We restate the theorem first.

Theorem 2. *Our construction is an SVME for* $\mathfrak{R}_{\mathsf{main}}$ *according to Def. 1 as long as the underlying (i) MIBE scheme satisfies Def. 2; (ii) the polynomial commitment scheme satisfies Def. 3 and (iii) the proof systems* $\mathsf{SP.PERM}$ *and* $\mathsf{SP.LEC}$ *both satisfy Def. 4 for their resepective relations.*

Proof. We show that our construction is an SVME for $\mathfrak{R}_{\mathsf{main}}$ by arguing the following *four* properties:

1. **Correctness**: Since $\mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id}) \to (\mathbf{c}, \pi)$, we have $\mathbf{c} \leftarrow \mathsf{MIBE.Enc}(\mathsf{mpk}, \mathbf{id}, \hat{\mathbf{m}})$ and $\pi_{\mathsf{perm}} \leftarrow \mathsf{SP.PERM.Prove}($ $(M, V), (m(X), \rho_m, \rho_v))$, $\pi_{\mathsf{lec}} \leftarrow \mathsf{SP.LEC.Prove}((M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk}), (\rho, \rho_m, \mathbf{m}))$; where M, V are honestly computed polynomial commitments. Since $\mathbf{m} \in \mathfrak{R}$, we can invoke the completeness properties of both $\mathsf{SP.PERM}$ and $\mathsf{SP.LEC}$ to claim that $d_{\mathsf{lec}} = 1$ and $d_{\mathsf{perm}} = 1$, and hence $\mathsf{Ver}(\mathbf{c}, \pi) = 1$. Finally using the correctness of MIBE we get correctness with a negligible loss.
2. **Soundness**: If $\mathsf{Ver}(\mathbf{c}, \pi) = 1$ then we have that both $d_{\mathsf{lec}} = 1$ and $d_{\mathsf{perm}} = 1$. Using the soundness of $\mathsf{SP.PERM}$ and $\mathsf{SP.LEC}$, we have that with overwhelming probability, $M = \mathsf{PC.Com}(m(X); \rho_m), V = \mathsf{PC.Com}(v(X); \rho_v)$ for polynomials $v(X)$ and $m(X)$ such that $\mathbf{v} = \mathsf{P2V}(v(X))$ and $\mathbf{m} = \mathsf{P2V}(m(X))$ are permutations of each other and $\mathbf{c}$ encrypts $\hat{\mathbf{m}}$ under $\hat{\mathbf{id}}$. Now, due to polynomial binding property of the underlying polynomial commitment scheme, $\mathbf{v} = (1 \dots, \tau)$ with overwhelming probability. Hence $\mathbf{m} \in \mathfrak{R}_{\mathsf{main}}$ with overwhelming probability over λ.
3. **Succinctness**: This follows directly from the succinctness of the proof systems $\mathsf{SP.LEC}$ and $\mathsf{SP.PERM}$ plus succinctness of the polynomial commitment scheme PC.
4. **Zero-knowledge**: Define a simulator $\mathcal{S}_{\mathrm{SVME}}(\mathsf{pp}, \{m_i\}_{i \in \mathcal{C}}, \mathsf{td}_{\mathsf{svme}}) \to (\mathbf{c}', \pi')$ based on the simulators $\mathcal{S}_{\mathsf{SP.PERM}}$ and $\mathcal{S}_{\mathsf{SP.LEC}}$ for the proof systems $\mathsf{SP.PERM}$ and $\mathsf{SP.LEC}$ as follows:
   - Parse $\mathsf{td}_{\mathsf{svme}}$ as $(\mathsf{td}_{\mathsf{lec}}, \mathsf{td}_{\mathsf{perm}})$.

- Sample a random vector $\tilde{\mathbf{m}} \in \mathfrak{R}_{\mathsf{main}}$ such that $\tilde{m}_i = m_i$ for all $i \in \mathcal{C}$.
- $\tilde{m}(X) := \mathsf{V2P}(\tilde{\mathbf{m}})$.
- $\hat{\mathbf{m}} := (\tilde{\mathbf{m}}, 0)$
- $(\tilde{e}', \mathbf{e}') \leftarrow \mathsf{MIBE.Enc}(\mathsf{mpk}, \hat{\mathbf{id}}, \hat{\mathbf{m}})$.
- Sample a uniform random $\rho_{\tilde{m}} \in \mathbb{Z}_p$.
- $\tilde{m}'(X) := \tilde{m}(x) + \rho_{\tilde{m}} z_\Omega(X)$
- $\tilde{M} := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, \tilde{m}'(X))$ and
  $V := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, v(X))$.
- $\pi'_{\mathsf{perm}} \leftarrow \mathcal{S}_{\mathsf{SP.PERM}}(\mathsf{td}_{\mathsf{perm}}, \tilde{M}, V)$.
- $\pi'_{\mathsf{lec}} \leftarrow \mathcal{S}_{\mathsf{SP.LEC}}(\mathsf{td}_{\mathsf{lec}}, \tilde{M}, \tilde{e}', \mathbf{e}', \hat{\mathbf{id}}, \mathsf{mpk})$.
- Return $((\tilde{e}' m \mathbf{e}'), (\tilde{M}, V, \pi'_{\mathsf{perm}}, \pi'_{\mathsf{lec}}))$

To prove the zero knowledge property, we use a hybrid argument. Consider the initial game (or 0^{th} hybrid) where the adversary sees honestly generated $(\mathbf{c}_0, \pi_0) \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathbf{m}, \mathbf{id})$. As given in the definition, $\mathbf{m} \in \mathfrak{R}$ and the corrupted set $\mathcal{C}$ is output by the adversary earlier.

- Hybrid 1: Replace the generation of π_{perm} with the output of the zero knowledge simulator $\mathcal{S}_{\mathsf{SP.PERM}}$'s output with (M, V) as input.
  This hybrid only differs from hybrid 0 in the π_{perm} part and is hence indistinguishable due to the zero knowledge property of $\mathsf{SP.PERM}$.
- Hybrid 2: Replace the generation of π_{lec} with the output of $\mathcal{S}_{\pi_{\mathsf{lec}}}$.
  This only differs from the previous hybrid in the π_{lec} part and is hence also indistinguishable due to the zero knowledge property of $\mathsf{SP.LEC}$.
- Hybrid 3: Sample a $\tilde{\mathbf{m}} \in \mathfrak{R}_{\mathsf{main}}$, and set $\hat{\mathbf{m}} := (\tilde{\mathbf{m}}, 0)$ as in the simulator and use that to generate the ciphertext $(\tilde{e}', \mathbf{e}')$.
  This hybrid is indistinguishable from the previous one due to the CPA security of the MIBE scheme.
- Hybrid 4: Finally replace the commitment M with $\tilde{M}$ as done in the simulator.
  This hybrid is statistically indistinguishable from the previous hybrid as since now the ciphertext $(\tilde{e}, \mathbf{e})$ is independent of ρ_m, the distribution of $m(X) + \rho_m z_\Omega(X)$ and $\tilde{m}(X) + \rho_{\tilde{m}} z_\Omega(X)$ are statistically indistinguishable.

Notice that hybrid-4 is the same as the ideal world – this concludes the proof.

□

B.3 Security of SP.LEC (Theorem 3)

Theorem 3. *The construction described in Figure 3 is a secure succinct proof system according to Definition 4 in the random oracle model.*

Proof. We show correctness, succinctness, soundness and zero-knowledge below:

- *Correctness* follows straightforwardly from the correctness of the polynomial commitment.
- *Succinctness* too is easy to see, observing that the proof π_{dleq} is only of constant size, as it works on a constant number of constraints and witnesses.
- *Soundness* can be argued by the soundness of DLEQ proof, binding of the polynomial commitment and an information theoretic argument. To argue this, first note that, the binding of polynomial commitment implies that M is a commitment of a polynomial, say $p(X)$ for which $p(\delta) = z_m$ – this holds with overwhelming probability over λ if $d_{\mathsf{kzg}} = 1$. Furthermore, if $d_{\mathsf{dleq}} = 1$, that would imply (with

overwhelming probability) the discrete logs $\mathsf{DLog}_{g_1}(\tilde{e})$ and $\mathsf{DLog}_E(T)$ both are equal to ρ – this follows from the soundness of DLEQ proof. Now, note that each $e_i = e(\mathsf{H}'(id_i), \mathsf{mpk})^\rho \cdot g_2^{m_i}$. The computation of T along with the fact that $\rho = \mathsf{DLog}_E(T)$ implies that

$$g_2^{m(\delta)-z_m} \cdot g_2^{\rho_m z_\Omega(\delta)} = 1$$

where $m(X) = \mathsf{V2P}(\mathbf{m})$. Alternatively we can write the exponents:

$$m(\delta) - p(\delta) + \rho_m z_\Omega(\delta) = 0 \bmod p.$$

Now, since δ is chosen as a output of random oracle, by Schwartz-Zippel with probability $O(1/p)$ we conclude $p(X) = m(X) + \rho_m z_\Omega(X) = m'(X)$ over $\mathbb{Z}_p$.

- *Zero knowledge* follows from the following simulation strategy:
  - The simulator gets input the instance $(M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk})$ and the trapdoor $\mathsf{td}_{\mathsf{pc}}$.
  - Sample uniform random $z_m \leftarrow_\$ \mathbb{Z}_p$.
  - Compute $\delta := \mathsf{H}(M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk})$.
  - Run $W := \mathsf{PC.TD.Open}(\mathsf{pp}_{\mathsf{pc}}, \mathsf{td}_{\mathsf{pc}}, M, \delta, z_m)$
  - Compute $E := \prod_i e(\mathsf{H}'(id_i)^{\ell_i(\delta)}, \mathsf{mpk})$ and $T := (\prod_i e_i^{\ell_i(\delta)} \cdot e_{\tau+1}^{z_\Omega(\delta)})/(g_2^{z_m})$
  - Run the simulator of DLEQ with instance $(g_1, E, \tilde{e}, T)$ to get the simulated proof π_{dleq}.
  - Output $(W, z_m, \pi_{\mathsf{dleq}})$.

  We can use a number of hybrids to show that the output distribution of the simulator is indistinguishable from the actual transcript output from a legitimate prover. The first hybrid would switch the DLEQ prover to the DLEQ simulator for generating π_{dleq}, rest remains unchanged – indistinguishability follows from the statistical zero knowledge of DLEQ proof system. In the next hybrid trapdoor opening is used instead of actual opening for the polynomial commitment – the indistinguishability follows from the KZG commitment properties. Finally, it is made independent of the witness by sampling z_m uniformly at random and deriving T without using ρ – this is just a syntactical change.

□

B.4 Security of $\Pi_{\mathsf{SV.Enc}}$ (Theorem 4)

Theorem 4. *The protocol $\Pi_{\mathsf{SV.Enc}}$, described in Figure 5 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{SV.Enc}}$ for arbitrary many semi-honest corruption in $\mathcal{F}_{\mathsf{LEC.Prove}}$-hybrid.*

Proof. To argue Theorem 4 we construct a simulator as follows:

1. Input: Public inputs $(\mathsf{pp}, \mathbf{id})$ and private input of the corrupt parties $([\mathbf{m}]_\mathcal{C}, [\rho]_\mathcal{C}, [\rho_m]_\mathcal{C})$, where $\mathcal{C} \subseteq [n]$ denotes the set of corrupt parties.
2. Send $([\mathbf{m}]_\mathcal{C}, [\rho]_\mathcal{C}, [\rho_m]_\mathcal{C})$ to the ideal functionality $\mathcal{F}_{\mathsf{SV.Enc}}$ to obtain $(\mathbf{c}, \pi)$ where $\mathbf{c} = (\tilde{e}, \mathbf{e})$ and $\pi = (M, V, \pi_{\mathsf{perm}}, \pi_{\mathsf{lec}})$
3. Choose $([M]_\mathcal{H}, [\tilde{e}]_\mathcal{H}, [e]_\mathcal{H})$ such that together with $([M]_\mathcal{C}, [\tilde{e}]_\mathcal{C}, [e]_\mathcal{C})$ they reconstruct to $(M, \tilde{e}, \mathbf{e})$. In Step-2 use these values on behalf of each honest party.
4. For each corrupt party, compute:
   (a) $[m'(X)] := [m(X)] + [\rho_m] z_\Omega(X)$ where $[m(X)] := \mathsf{V2P}([\mathbf{m}])$.
   (b) $[M] := \mathsf{PC.Com}(\mathsf{pp}_{\mathsf{pc}}, [m'(x)])$;
   (c) $([\tilde{e}], [\mathbf{e}]) := \mathsf{MIBE.Enc}(\mathsf{mpk}, \hat{\mathbf{id}}, [(\mathbf{m}, \rho_m)]; [\rho])$;

5. Simulate the ideal functionalities $\mathcal{F}_{\mathsf{Perm.Prove}}$ and $\mathcal{F}_{\mathsf{LEC.Prove}}$ using π_{perm} and π_{lec} obtained from the ideal functionality $\mathcal{F}_{\mathsf{SV.Enc}}$.

Now we argue the simulation is correct with overwhelming probability. This can be argued by a number of hybrids, starting from the real world, where the protocol Π_{Enc} is executed in presence of an adversary, and then gradually moving towards an ideal world, in presence of the above simulator. The hybrids are as follows:

Hybrid-1: This hybrid has everything same as the real world except the values $([\tilde{e}], [\mathbf{e}]$ from an honest party is replaced by uniform random values subject to the reconstructed value $(\tilde{e}, \mathbf{e})$. We note that this is indistinguishable from the real world due to BDDH over the bilinear group. Given $(\mathsf{mpk} = g_1^{\mathsf{msk}}, \tilde{e} = g_1^{\rho}, g_1^{\alpha})$ where $\mathsf{msk}, \rho, \alpha$ are all uniform random over $\mathbb{Z}_p$, by BDDH we can argue that each $e(\mathsf{H}'(id_i) = e(g_1, g_1)^{r_i \mathsf{msk} \alpha}, \mathsf{mpk})^{\rho})$ is indistinguishable from $h_2^{r_i}$ where $h_2 \leftarrow_{\$} \mathbb{G}_2$ and $r_i \leftarrow_{\$} \mathbb{Z}_p$ as long as each $\mathsf{H}(id_i)$ is programmed to $g_1^{r_i \alpha}$ while assuming H' is a programmable random oracle (here we denote $id_{\mathsf{pp}} = id_{\tau+1}$).

Hybrid-2: In this hybrid, the only change we make is that instead of giving $h_2^{r_{\tau+1}} \cdot g_2^{\rho_m}$, we just give $h_2^{r_{\tau+1}} \leftarrow_{\$} \mathbb{G}_2$ – this is perfectly indistinguishable from previous

Hybrid-3: This is same as the previous one except that each honest $[M]$ is now chosen uniformly at random subject to the reconstruction value equal to M. This is again perfectly indistinguishable from the previous hybrid by construction. This is now same as the ideal world.

□

B.5 Security of $\Pi_{\mathsf{SP.LEC.Prove}}$

Theorem 5. *The protocol $\Pi_{\mathsf{SP.LEC.Prove}}$, described in Figure 4 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{LEC.Prove}}$ for arbitrary many semi-honest corruption in $(\mathcal{F}_{\mathsf{DLEQ.Prove}}, \mathcal{F}_{\mathsf{Z\text{-}Share}})$-hybrid.*[13]

Proof. To argue Theorem 5 we construct a simulator, that has access to $\mathsf{td}_{\mathsf{pc}}$ as follows:

1. Input: public inputs $(\mathsf{pp}_{\mathsf{lec}}, M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk})$ and private input $([\rho]_{\mathcal{C}}, [\rho_m]_{\mathcal{C}}, [\mathbf{m}]_{\mathcal{C}})$.
2. Send $([\rho]_{\mathcal{C}}, [\rho_m]_{\mathcal{C}}, [\mathbf{m}]_{\mathcal{C}})$ to the ideal functionality $\mathcal{F}_{\mathsf{LEC.Prove}}$ to obtain output $\mathsf{SP.LEC.Prove} = (W, z_m, \pi_{\mathsf{dleq}})$.
3. For each corrupt party, compute:
   (a) $\delta := \mathsf{H}(M, \tilde{e}, \mathbf{e}, \hat{\mathbf{id}}, \mathsf{mpk})$;
   (b) $[m(X)] := \mathsf{V2P}([\mathbf{m}])$.
   (c) $[m'(X)] := [m(X)] + [\rho_m] z_{\Omega}(X)$
   (d) $[z_m] := [m'(\delta)]$
   (e) $[W] := \mathsf{PC.Open}(\mathsf{pp}_{\mathsf{pc}}, [m'(X)], \delta)$
   (f) $[W] := [W] \cdot [1]$, where each $[1]$ is sampled randomly, and used in simulating $\mathcal{F}_{\mathsf{Z\text{-}Share}}$ when queried by the adversary.
4. Choose uniform random $([z_m], [W])$ on behalf of honest parties, such that together with the corrupt party's shares they reconstruct to z_m and W respectively.
5. Finally in response to messsage to functionality $\mathcal{F}_{\mathsf{DLEQ.Prove}}$ from the adversary, returns π_{dleq}.

We show that the above simulation is correct via a number of hybrid, stating from real world to ideal world as follows:

Hybrid-1: In this hybrid $[z_m]$ from a honest party is replaced by a random value without using $[m'(X)]$. This is perfectly indistinguishable as the operation on a uniform random point δ by a random polynomial $m'(X)$ (since each $[\rho_m]$ is random).

[13]See Section 5.5 for descriptions of $\mathcal{F}_{\mathsf{DLEQ.Prove}}$.

Hybrid-2: In this hybrid $[W]$ for each honest party is picked randomly subject to the reconstruction to W. Now, here crucially the secret-sharing of zero becomes crucial. It ensures that each simulated $[W]$ has the same distribution as real ones, unconditionally. Without this, each $[W]$ can be individually verified using $\mathsf{PC.Ver}(\mathsf{pp}_{\mathsf{pc}}, [M], [W], \delta, [z_m])$. However, masking with a random value, which does not perturb the final evaluation breaks this correlation. However, such secret-share of zero does not work when exactly $n-1$ parties are corrupt, because adversary can calculate the exact share of zero of the honest party subtracting from 0. Nevertheless, in that case $[W]_{\mathcal{H}}$ becomes deterministic anyway, so simulator does not need to choose random values in behalf of honest parties. Observe that this hybrid is the same as ideal world – this conclude the proof.

□

B.6 Security of $\Pi_{\mathsf{R\text{-}Perm\text{-}Gen}}$ (Theorem 6)

Theorem 6. *The protocol $\Pi_{\mathsf{R\text{-}Perm\text{-}Gen}}$ securely realizes $\mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}$ in $(\mathcal{F}_{\mathsf{RAN}_\omega}, \mathcal{F}_{\mathsf{DY\text{-}DPRF}})$-hybrid.*

We describe a simulator which simulates the honest server's communication plus all the ideal functionalities' response to the adversary. A key observation is that for $\mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}$, there is no private input, but each party gets a private output, which is additive share of a permutation. Now let us consider two cases:

- Case-1: When $|\mathcal{C}_S| = n$. In this case, the simulation is straightforward as it gets to choose the permutation $\mathbf{s}$. It is important to note that, in this case, the adversary too knows the permutation. So the simulation strategy is to use the same permutation, which is chosen by the adversary. The simulator extracts the permutation from adversary's interaction with $\mathcal{F}_{\mathsf{RAN}_\omega}$, which is now simulated by the simulator. Note that, in this case the permutation may not be random.
- Case-2: When $|\mathcal{C}_S| < n$. The simulator does not get to choose $\mathbf{s}$, however it can leverage the fact the adversary does not know $\mathbf{s}$ either. So the strategy it follows is:
  - Invoke $\mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}$ to obtain back $[\mathbf{c}]$.
  - Choose uniform random permutation $\mathbf{s}'$ of size τ $\{1, \omega, \ldots, \omega^{\tau-1}\}$ and compute their random additive shares, such that for each corrupt party $[\mathbf{s}'_i] = [c_i]$.
  - Now expand that to a set of B values $\gamma_1, \ldots, \gamma_B$, each $\gamma_i \in \{1, \ldots, \omega^{\tau-1}\}$ such that the sanitization would result in exactly $\mathbf{s}'$.
  - Simulate $\mathcal{F}_{\mathsf{RAN}_\omega}$ using $[\gamma]_{\mathcal{C}_S}$.
  - Simulate $\mathcal{F}_{\mathsf{DY\text{-}DPRF}}$ by using a secret key sk and compute on the γ_i values.

Intuitively, the above simulation strategy is correct as the marginal distribution of adversary's view in the simulated execution, which is consistent with $\mathbf{s}'$ is statistically indistinguishable from the marginal distribution of adversary's view in the real execution conditioned consistent with another random permutation $\mathbf{s}$. Precisely, since the output is secret shared, as long as a random permutation is chosen, which may be different from the one chosen by the ideal functionality, the adversary's view is the same.

B.7 Security of Π_{VES} (Theorem 7)

Theorem 7. *The protocol Π_{VES} (Fig. 10) securely realizes $\mathcal{F}_{\mathsf{VES}}$ (Fig. 9) in $(\mathcal{F}_{\mathsf{Setup}}, \mathcal{F}_{SBB}, \mathcal{F}_{\mathsf{KeyExt}}, \mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}, \mathcal{F}_{\mathsf{SV.Enc}})$-hybrid.*

Proof (Sketch). The theorem essentially follows from the universal composability, as the only job of the simulator is to simulate the hybrid functionalities correctly. Intuitively, since the permutation sampled is always secret shared, the only time a card is known to the adverasry in the clear is by corrupting the corresponding player. Hence, the only challenge is to ensure that consistency is maintained between the cards obtained by the corrupt players, and the prior interactions.

In particular, when $|\mathcal{C}_S| < n$ the simulator works as follows:

- Throughout the protocol simulate $\mathcal{F}_{\text{SBB}}$ correctly, using the appropriate verification algorithm. The correctness is guaranteed by the soundness of the SVME.
- In the Setup phase simulate $\mathcal{F}_{\mathsf{Setup}}$ by running KeyGen of SVME to obtain pp, all msk_i and $\mathsf{td}_{\mathsf{svme}}$.
- In the Sample (offline) phase it makes call $(\mathsf{gid}, \textsc{Sample})$ and subsequently $(\mathsf{gid}, \textsc{Card}, j)$ to the functionality $\mathcal{F}_{\mathsf{VES}}$ to obtain back the corrupt player's deck $\{s_{i,j}\}_{i \in \mathcal{C}_P}$. It obtains $[\mathbf{v}]_{\mathcal{C}_S}$ from the adversary. For each corrupt player $i \in \mathcal{C}_P$ it samples $[v_{i,j}]$ subject to reconstruction being equal to $v_{i,j} = \omega^{s_{i,j}}$. This is the consistency requirement.
- In the Retrieve (online) phase when the adversary requests a card, say $s_{i,j}$, on behalf of a corrupt player, simply forward that.

It is not hard to argue that the simulation is correct. Note that, since in the real world (in $(\mathcal{F}_{\mathsf{Setup}}, \mathcal{F}_{\text{SBB}}, \mathcal{F}_{\mathsf{R\text{-}Perm\text{-}Gen}}, \mathcal{F}_{\mathsf{SV.Enc}})$-hybrid), each party's only interaction happens via these ideal functionality, usage of the above simulator instead makes only syntactic changes. Security follows directly from composability. We omit the details.

$\square$

C A simplified solution with player's pre-registration

We note that, since we attempt to capture a realistic setting, where players are unknown until they are available to play, we needed to deal with the IBE setting and also a keyper's committee. However, one could imagine a setting where players pre-register for the game ahead of time (maybe an hour before) for a game to take place at a later time – this gives the framework to invoke a public-key infrastructure. In this setting, we can easily replace the IBE by standard PKE – ElGamal encryption would just do the job, while everything else stays the same during the offline phase, except now the ciphertext is encrypted with player's public key directly instead of identities. The major difference comes into the online retrieval phase – now each player can immediately decrypt their deck with the secret keys – in other words, there is no key-derivation step. This makes the online phase totally off-chain and truly instantaneous. Of course, this severely relies on all players being able to register ahead of time, which is a major restriction and therefore we do not focus on this. It is an interesting open question, if a truly instantaneous solution (where the entire key-derivation procedure takes place off-chain) is possible in our IBE setting, or more technically, without assuming PKI.