Efficient Multi-Key Verifiable Shuffles from Short Arguments for Randomized Algorithms

Benedikt Bünz, Mariana Raykova, and Jayshree Sarathy

Abstract. Verifiable shuffles are a key building block for mixnets, which are used to provide anonymity in electronic communication, payment, and voting systems. Existing constructions of verifiable shuffles are only able to shuffle ciphertexts encrypted under the *same* public key, which limits the functionality of the mix-net to one-way communication. We introduce the first *multi-key verifiable shuffle*, which shuffles ciphertexts encrypted under *different* public keys, along with the public keys themselves. This shuffle enables a powerful, bi-directional mixnet, which allows users to participate in a protocol even after the mixing is complete. For instance, users can use the output of the multi-key shuffle to authenticate, send, and receive private messages, and perform zero-knowledge proofs about their ciphertexts.

We provide a zero-knowledge argument for the correctness of the multi-key shuffle that has $O(\log(n))$ proof size and $O(n)$ prover and verifier time when shuffling n k-bit elements. This improves upon the previous state-of-the-art, Bulletproofs (Bünz et al. S&P2018), which has $O(\log(kn\log(n)))$ proof size and $O(kn\log(n))$ prover and verifier time.

In addition, we present an improved non-interactive zero-knowledge argument protocol for arbitrary arithmetic circuits that inherits the short proofs and lack of trusted setup from Bulletproofs, and additionally offers the new ability to perform proofs on randomized algorithms, yielding concrete improvements in proof size for the class of problems with faster randomized verification. The protocol also enables proofs over committed vectors, which was previously not possible in a black-box manner, and maintains zero-knowledge even under subversion of the common reference string.

1 Introduction

The notion of a cryptographic shuffle was introduced by Chaum [1] in the context of mix networks (mixnets), where multiple parties have messages and the goal is to output these messages in a shuffled order such that they are unlinkable from the inputs. A verifiable shuffle additionally guarantees that all inputs (and only real inputs) appear in the output list. Mixnets are widely used in systems that require anonymity and unlinkability, including anonymous communication systems [2, 3], private bulletin boards [4], anonymous payment schemes [5–7], anonymous reputation systems [8, 9], and electronic voting schemes [10, 11]. These systems typically leverage several mixnet servers or shufflers, each of which shuffles the inputs in turn. As the content of the inputs needs to remain hidden during this process (otherwise, the unlinkability property cannot be achieved), the shuffle algorithm must work with the encrypted inputs. Moreover, the input and output ciphertexts from each shuffle cannot be identical, as this will reveal the permutation. There are two main types of mixnets depending on how the latter is achieved: decryption mixnets [2], where each shuffler peels off a layer of encryption, and re-randomization mixnets [10], where each shuffler re-randomizes the input ciphertexts before applying its permutation.

The security properties of a mixnet rely on the shuffle applied by each mix server. While the unlinkability can be guaranteed as long as at least one of the servers acts honestly and does not reveal its permutation, correctness requires that all mix servers output the encryptions of the same messages that were encrypted in their input. The goal of verifiable shuffles [10, 12–16] is to guarantee this correctness property. More formally, a verifiable shuffle requires the shuffler to prove that two lists of ciphertexts, $\mathsf{ct}_1, ..., \mathsf{ct}_n$ and $\mathsf{ct}'_1, ..., \mathsf{ct}'_n$, encrypt the same set messages up to a permutation, i.e. there exists a permutation $\pi \in \Sigma_n$ such that ct'_i and $\mathsf{ct}_{\pi(i)}$ encrypt the same message for all $i \in [n]$.

Bi-directional Mixnet and Multi-Key Shuffle. Anonymous bulletin boards are widely used in sensitive communication settings [2–11] to allow users to post on a public bulletin board without revealing their identities. Such constructions can be enabled by decryption mixnets, where clients encrypt their messages under layers of keys corresponding to the mixnet servers. However, a major limitation of anonymous bulletin boards is that it is difficult for communications to occur beyond the initial posting on the board.

Therefore, we consider a bi-directional mixnet, in which users are able to both send and receive messages. For this, each message is accompanied by a message key, which is a public key for which the sender knows the private key and using which the receiver can encrypt a reply to the sender. Importantly, the message key is unlinkable to the sender's main public key and the sender's identity. The architecture can also be used for anonymous reputation systems [8, 9]. Users can vote on messages, and in each round a message receives a set of scores. A user can convincingly prove that her aggregate score over multiple rounds is within some reputable range. The mix-net ensures that the message content and votes on that message are unlinkable to the sender's identity and main public key.

In order to enable this new powerful functionality we require a new notion of a verifiable shuffle with the following capabilities. The shuffle needs to output shuffled and randomized encrypted messages, along with a randomized public key associated with the message. We call this a *multi-key verifiable shuffle*. The key difference to prior verifiable shuffles [10, 12, 14–17] is that the messages can be encrypted under different keys rather than just one single key.

Concretely, our new multi-key verifiable shuffle takes as inputs several ciphertexts encrypted under different keys $\mathsf{ct}_1 = \mathsf{Enc}(\mathsf{pk}_1, m_1), ..., \mathsf{ct}_n = \mathsf{Enc}(\mathsf{pk}_1, m_1)$ together with their corresponding public encryption keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and outputs new pairs of ciphertexts and encryption keys $\{(\mathsf{pk}'_i, \mathsf{ct}'_i = \mathsf{Enc}(\mathsf{pk}'_i, m'_i))\}_{i=1}^n$ such that there exists a permutation π such that $m'_i = m_{\pi(i)}$, and additionally there exists an inverse operation of the shuffle that takes any ciphertext $\mathsf{ct}'_j = \mathsf{Enc}(\mathsf{pk}'_j, x)$ and transforms it into a ciphertext $\mathsf{ct}_j = \mathsf{Enc}(\mathsf{pk}'_{\pi^{-1}(j)}, x)$. The shuffled encryption keys enable a reader to post responses encrypted under the corresponding key pk'_j, which can be shuffled back to the author and will be encrypted under her corresponding key and only she can read them. If the encryption scheme provides homomorphic properties, then the responses, which can be just votes, can be first aggregated before being sent back to the author. We could further require that the transformation between $\mathsf{ct}'_j = \mathsf{Enc}(\mathsf{pk}'_j, x; r)$ and $\mathsf{ct}_j = \mathsf{Enc}(\mathsf{pk}'_{\pi^{-1}(j)}, x; r)$ preserves the encryption randomness. This enables the anonymous sender to not only be able to decrypt the responses sent back to her, but further to be able to provide zero-knowledge proofs for the encrypted messages in these responses that have been recorded on the bulletin board, for example, give a range proof for the aggregated votes that the sender received.

We present a new multi-key verifiable shuffle that can support both of the above features.

Proofs for Randomized Functionalities Beyond Shuffles. Most verifiable shuffle protocols that do not use generic zero-knowledge arguments for circuits leverage a randomized verification algorithm[10, 16]. This algorithm uses a representation of the shuffled set as a polynomial whose roots are the set elements. The polynomial is invariant under permutation of the elements. Verifying that two sets are equal becomes a polynomial identity test for which there are efficient randomized techniques [18–20]. Our multi-key shuffle protocol provides an argument for satisfiability of this randomized verification algorithm. We further consider the general question of zero-knowledge arguments for randomized functionalities, where the verifier provides challenge randomness after the prover has committed to its witness, and we present such a protocol. This stands in contrast to proof systems for NP-relations that are defined by a deterministic circuit for checking the relation. Enabling randomized verification not just theoretically increases the expressiveness of statements but can also practically lead to efficiency gains. For example randomized primality check [21, 22] is significantly more efficient than the deterministic variant[23].

Our Contributions. We introduce a new notion of a *multi-key verifiable shuffle* that extends the capabilities of existing verifiable shuffle constructions, which work only for shuffling ciphertexts under the same key. The multi-key verifiable shuffle allows permuting ciphertexts under different keys together with their corresponding public encryption keys $\{\mathsf{pk}_i, \mathsf{Enc}(\mathsf{pk}_i, m_i)\}$. The output of the multi-key shuffle is a new set of ciphertexts and public keys $\{\mathsf{pk}'_{\pi(i)}, \mathsf{Enc}(\mathsf{pk}'_{\pi(i)}, m_{\pi(i)})\}$, such that the ciphertexts are encryptions of the original messages, now under the corresponding public keys included in the shuffle output. This functionality provides enhanced communication properties for anonymous bulletin boards.

We present a multi-key verifiable shuffle construction which provides a zero-knowledge argument for the statements that two sets of public keys and ciphertexts, $\{\mathsf{pk}_i,\ \mathsf{Enc}(\mathsf{pk}_i, m_i)\}$, and $\{\mathsf{pk}'_{\pi(i)},\ \mathsf{Enc}(\mathsf{pk}'_{\pi(i)}, m_{\pi(i)})\}$, satisfy the multi-key relation defined above. Our construction is based on the hardness of discrete log and does not require any trusted setup. The size of the proof is logarithmic and the prover and verification time in linear in the size of the shuffle set.

This is the first verifiable shuffle of public keys along with ciphertexts encrypted under these keys. Our shuffle enables users of an anonymous communication system to participate in secure interactions even after the shuffle is complete. These interactions include authenticating, sending, and receiving secure private messages, as well as proving statements about their ciphertexts in zero-knowledge.

Our verifiable shuffle argument protocol builds upon Bulletproofs [24, 25], but yields new functionality which we are able to generalize to arbitrary arithmetic circuits. In particular, we present a general zero-knowledge argument system that offers the following properties.

Proofs over Vector Commitments. Bulletproofs enables proofs on Pedersen committed values. This means that the proof circuit can directly take as input committed values without implementing the opening function of the commitment directly. This has many applications, such as efficient range proofs [26, 27]. We significantly extend this functionality such that the proof can take multiple Pedersen committed vectors as input. These vectors could be the result of an outside protocol as is the case in the multi-key shuffle. With this extension, the proof circuit can directly take the committed vectors as input rather than implementing the opening function of the commitment. For m vectors of size n the proof size is only $O(\sqrt(m) + \log(m \cdot n))$.

Subversion Zero-Knowledge. We show that Bulletproofs with a simple extension satisfies the notion of subversion zero-knowledge [28]. That is, even for a maliciously generated reference string, the zero-knowledge property is preserved. In the multi-key shuffle, we use the users' public keys as a CRS and, thus, have to deal with a potentially maliciously generated CRS.

Proofs for Randomized Functionalities. We extend Bulletproofs to enable proofs for randomized functionalities with the randomness provided by the verifier. To do this, we insert additional rounds to the Bulletproofs protocol. Using the ability to efficiently perform proofs on vector committed values, the prover can commit to part of the verification computation. The verifier then sends a random challenge, and the prover proves correctness of the rest of the computation using the challenge. This can be done multiple times.

Shuffles of Public Keys and Ciphertexts. Our protocol provides the ability to shuffle both public keys and ciphertexts. This is significantly different from shuffling committed or encrypted values. Our shuffle does this using algebraic techniques, i.e. only operating on the keys and ciphertexts as group elements, rather than an encoding of the elements. This has significant efficiency benefits as implementing group operations inside a circuit is very costly.

Efficiency. The proof system inherits the efficient properties of Bulletproofs. The proofs are logarithmic and require no trusted setup. The proof generation and verification time are linear in n, the witness size. Security is based on the discrete logarithm assumption. The new ability to do proofs on randomized verification leads to a shuffle with linear prover and verifier time, as opposed to $n \log(n)$ for Bulletproofs' deterministic shuffle. Along with a concurrent work [29] (which supports only ciphertext under the same key), we present the first logarithmic size shuffle of encrypted values. The protocol can be made made non-interactive in the random oracle model using the Fiat-Shamir transform [30].

Technical Overview. Existing verifiable shuffle constructions [10, 31, 15] work only over ciphertexts encrypted under the same key. They are limited to this functionality, because their ciphertext re-randomization techniques rely on knowledge of a single global public key of the encryption scheme. In our setting, each ciphertext is associated with a different public key. The shuffle has to not only permute the ciphertexts, but also the public keys, in order to break the linkability between the original public keys and the final output of the mixnet.

In our approach, the shuffle applies a transformation that maps all public keys to a new set of randomized keys and re-encrypts the ciphertext under these new keys, which guarantees unlinkability between the inputs and outputs consisting of key-ciphertext pairs. Neff's 'Simple k-Shuffle' [10], which is a building block in his shuffle of El Gamal ciphertexts, also applies masks on the shuffled messages as we do, but the corresponding zero-knowledge argument of correctness requires the server to know the shuffled messages in the clear.

Our multi-key shuffle argument leverages ideas from previous verifiable shuffles. In particular, it uses the insight that polynomials are invariant under permutation of their roots. Thus, we can check that two lists are permutations of each other by checking that the polynomials they induce are equal, for which we have an efficient randomized verification algorithm based on polynomial evaluation at a random point [19, 20]. Another technique that we borrow from previous work is probabilistic checking that two lists contain the same elements by comparing a random linear combination of the elements in each list.

Multi-Key Verifiable Shuffle. Our construction of the multi-key shuffle uses the additively homomorphic El Gamal encryption scheme: informally, for public generator g, $\mathsf{Enc}(\mathsf{pk}, m; r) = (g^r, g^m \mathsf{pk}^r)$. The shuffler must argue the following: for shuffle inputs $\{(\mathsf{pk}_i, \mathsf{ct}_i)\}_{i=1}^n$ and outputs $\{(\mathsf{pk}'_i, \mathsf{ct}'_i)\}_{i=1}^n$, it knows a permutation π and a secret s such that, for all $i \in [n]$, $\mathsf{pk}'_i = \mathsf{pk}^s_{\pi(i)}$ and $\mathsf{ct}'_i = \mathsf{ct}^s_{\pi(i)}$. By the properties of additively homomorphic El Gamal, this implies that for all $i \in [n]$, $m'_i = s \cdot m_{\pi(i)}$. The shuffler's commitment to s enables each user to compute her new public key and recover her original message. We describe the argument protocol at a high level. First, the verifier computes implicitly three commitments, $A_{R,h}, A_{R,b}, A_{R,c}$, to random linear combinations of the output committed values, $\{x'_i, \gamma'_i, d'_i\}_{i=1}^n$ with respect to public commitment parameter g. Crucially, the verifier creates these commitments using the *outputs* of the shuffle as commitment parameters, and using its random challenges r, u.

$$A_{R,h} = \prod_i {\mathsf{pk}'}_i^{(r-u^i)} = \mathsf{Com}_g(\sum_i x'_i(r-u^i))$$
$$A_{R,b} = \prod_i {\mathsf{ct}'}_{1,i}^{(r-u^i)} = \mathsf{Com}_g(\sum_i \gamma'_i(r-u^i))$$
$$A_{R,c} = \prod_i {\mathsf{ct}'}_{2,i}^{(r-u^i)} = \mathsf{Com}_g(\sum_i d'_i(r-u^i))$$

where $\mathsf{ct}'_i = (\mathsf{ct}'_{1,i}, \mathsf{ct}'_{2,i})$.

Now, to prove a relation between the inputs and outputs of the shuffle, the prover must argue that $A_{R,h}, A_{R,b}, A_{R,c}$ have the correct form with respect to the *inputs* of the shuffle. First, the prover computes $\mathbf{a}_R = (a_{R1}, \ldots, a_{Rn})$, which are the committed values in $A_{R,h}, A_{R,b}, A_{R,c}$ with respect to *inputs* of the shuffle as commitment parameters. Then, the prover must argue the following three properties about $\mathbf{a}_R$.

1. $\prod_i a_{Ri} = s^n \prod_i (r - u^i)$.
   Using a polynomial identity test, this shows that there exists $\pi \in \Sigma_n$ such that a_{Ri} contains $c_i \cdot (r - u^{\pi^{-1}(i)})$,where c_i is as of now unknown.
2. $\sum_i a_{Ri} = s \sum_i (r - u^i)$.
   Using a random linear combination check, this shows that there exists a secret s such that for all a_{Ri}, $c_i = s$.
3. $\mathbf{a}_R$ is the committed value in $A_{R,h}$ with respect to bases $\{\mathsf{pk}_i\}_{i=1}^n$, in $A_{R,b}$ with respect to bases $\{\mathsf{ct}_{1,i}\}_{i=1}^n$, and in $A_{R,c}$ with respect to bases $\{\mathsf{ct}_{2,i}\}_{i=1}^n$.

We will leave the discussion of how the prover argues these properties to later in the overview. If $\mathbf{a}_R$ satisfies these constraints, then with high probability, each commitment is now a commitment, under parameter g, to not only a random linear combination of the output values, but also to a *permuted* random linear combination of the *input* values.

$$A_{R,h} = \mathsf{Com}_g(\sum_i x_i(r - u^{\pi^{-1}(i)}))$$
$$A_{R,b} = \mathsf{Com}_g(\sum_i \gamma_i(r - u^{\pi^{-1}(i)}))$$
$$A_{R,c} = \mathsf{Com}_g(\sum_i d_i(r - u^{\pi^{-1}(i)}))$$

Therefore, the verifier can deduce that, with overwhelming probability, $x'_i = s \cdot x_{\pi(i)}$, $\gamma'_i = s \cdot \gamma_{\pi(i)}$, and $d'_i = s \cdot d_{\pi(i)} \forall i \in [n]$, for $d'_i = m'_i + x'_i \gamma_{\pi(i)}$ All private keys, messages and randomizers are therefore shifted by the same random shift s and the output ciphertext is an encryption using the output public key and input randomizer. The latter equality implies that $s \cdot m_{\pi(i)} = d'_i - x'_i \cdot \gamma_{\pi(i)} = m'_i$. Thus, we can deduce that $s \cdot m_{\pi(i)} = m'_i$.

In order to prove the constraints on $\mathbf{a}_R$, our protocol will invoke a sub-routine of Bulletproofs with modifications. Our first modification provides zero-knowledge under subverted parameters. Note that in our shuffle argument, the commitments $A_{R,h}, A_{R,b}, A_{R,c}$ use the inputs of the shuffle as commitment parameters. To ensure that zero-knowledge will hold under potentially subverted parameters, the prover must run an algorithm to verify that the commitment parameters are valid. Our second modification arises from using multiple vector commitments $A_{R,h}, A_{R,b}, A_{R,c}$ in the shuffle proof. To incorporate these into the Bulletproofs protocol, the prover must argue that these commitments use non-intersecting commitment parameters. The third modification is that the permutation argument uses additional random challenges. We modify Bulletproofs to work for more general randomized functionalities, using a more powerful version of the forking lemma of Bootle et al.[24] to proof security.

These modifications are explained in more detail below in the overview of our general zero-knowledge argument.

Comparison to Bulletproofs' verifiable shuffle protocol. The Bulletproofs' verifiable shuffle protocol takes as input two lists of n commitments. It runs both lists through a sorting circuit and checks that the outputs are equal. The proof size is $O(k\log(n\log(n)))$ (assuming the shuffled elements have k bits), and proof generation and verification runs in $O(kn\log(n))$ time. Using randomized verification algorithms, we are able to shuffle multiple keys and ciphertexts, rather than just commitments, while improving the proof size to $O(\log(n))$ and the prover and verifier time to $O(n)$.

Generalizing our Techniques for Arithmetic Circuits. We describe our starting point, Bulletproofs [25], and our three major extensions. Following [24], Bulletproofs represents an arithmetic circuit with n multiplication gates (each with fan-in 2) as a Hadamard product (entry-wise multiplication) and a set of linear constraints. For each multiplication gate, let $\mathbf{a}_L$ be the vector of left wire values, $\mathbf{a}_R$ be the vector of right wire values, and $\mathbf{a}_O$ be the vector of output wires. Then, the multiplication gates can be represented using the Hadamard product relation, $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$, and the rest of the circuit, including constraints on inputs $\{p_i\}_{i=1}^{l}$, can be captured using $Q \leq 2n$ linear constraints of the form $\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}$, for $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q\times n}$, and $\mathbf{d} \in \mathbb{Z}_p$. The last constraint is that the inputs $\{p_i\}_{i=1}^{n}$ must be the committed values within the l Pedersen scalar inputs commitments from the prover, $\{P_i = \mathsf{Com}(p_i;\rho_i)\}_{i=1}^{l}$. Thus, the arithmetic circuit is reduced to the following constraints.

$$\{P_i\}_{i=1}^{l} = \{\mathsf{Com}(p_i;\rho_i)\}_{i=1}^{l} \quad \wedge \quad \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \quad \wedge$$
$$\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}$$

Our first extension of Bulletproofs is to make the protocol more expressive by allowing proofs statements not only on committed scalars, but also on committed vectors. To the best of our knowledge, proving statements on committed vectors could not be done previously in a black-box way.

To enable vector-committed inputs, we modify the Bulletproofs protocol as follows. Let $\{\mathbf{v}_i\}_{i=1}^{m}$ be the vector-valued inputs to the circuit. The linear constraints that describe the circuit now include constraints $\{\mathbf{W}_{V\,i}\}_{i=1}^{m}$ on the input vectors as follows. The prover must argue that the following constraints are satisfied:

$$\{V_i\}_{i=1}^{m} = \{\mathsf{Com}(\mathbf{v}_i;\gamma_i)\}_{i=1}^{m} \quad \wedge \quad \{P_i\}_{i=1}^{l} = \{\mathsf{Com}(p_i;\rho_i)\}_{i=1}^{l} \quad \wedge \quad \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \quad \wedge$$
$$\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O + \sum_{i=1}^{m} \mathbf{W}_{V\,i} \cdot \mathbf{v}_i = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}.$$

Let m be the number of vector commitments, each of size n. The incorporation of vector commitments and corresponding constraints yields polynomials $l(X), r(X)$, and $t(X)$ of degree $O(m)$. Using the polynomial commitment scheme from [24], we incur a cost in proof size of $O(\sqrt{m})$ but are able to reduce input size by a factor of m. When $\sqrt{m} < \log(m \cdot n)$, we are able to provide more expressive proofs with the same communication complexity.

Our second extension is to enable proofs for randomized verification circuits. Bulletproofs can only handle deterministic circuits. For these randomized circuits the verifier is able to supply randomness and the circuit computation can depend on that randomness. This is powerful as many important problems such as polynomial identity testing [19, 20], matrix multiplication [32], and primality testing [21, 22] can be more efficiently verified using randomized algorithms. Our main technique is to allow the verifier to adaptively define the circuit. We encode a universal circuit that accepts as input a set of functions, and that will evaluate the functions on the prover's inputs and the verifier's random challenges. The prover begins by providing commitments to the witness as inputs to the protocol. Then, the verifier samples a random challenge $\mathbf{c} \in \mathbb{Z}_p^n$, with which, the prover and verifier can compute the randomized linear constraints of the circuit, $\mathbf{W}_L, \mathbf{W}_R$, $\mathbf{W}_O, \{\mathbf{W}_{Vi}\}_{i=1}^m$,, and $\mathbf{W}_P$ using a set of linear functions. In addition, using $\mathbf{c}$ and the witness, the prover can compute the randomized wire values for the multiplication gates of the circuit, $\mathbf{a}_L, \mathbf{a}_R$, and $\mathbf{a}_O$. Once the constraints and wire values are fixed, then the protocol proceeds as before. Given a randomized verification algorithm, our argument can be used to show that with high probability, the prover's inputs satisfy the randomized verification. The question of whether certain randomized verification algorithms imply soundness for deterministic statements is outside the scope of our general zero-knowledge argument. However, we provide an extension of the forking lemma used in [24, 25] to show that witness-extended emulation holds even for probabilistic circuits.

With the addition of this functionality for randomized algorithms, our proof size remains logarithmic in the size of the circuit, but the *circuits themselves* are often significantly smaller. Thus, we achieve concrete improvements in communication complexity for the class of problems with more efficient randomized verification.

Our third extension is allowing the CRS to be generated by an untrusted verifier or third-party, rather than from a trusted setup algorithm. The motivation for this comes from the shuffle application; there, the inputs and outputs of the shuffle, which are generated by the individual users of the bulletin board and the shuffler, are used as commitment parameters for the argument protocol. In such settings, when the CRS could be subverted, we must ensure that zero-knowledge still holds. We use the definition of subversion zero-knowledge recently proposed by Bellare et al. [28].

To ensure subversion zero-knowledge, the prover must run an algorithm to verify that the CRS looks honestly generated and only proceeds if the algorithm accepts the CRS. If there exists such an algorithm, then the prover's protocol will remain zero-knowledge. We show that such an algorithm exists for Pedersen commitments. Note that the prover is not allowed to subvert the CRS, and we assume that the prover does not know a discrete log relation between the commitment parameters. Thus, soundness is not affected by the subverted CRS.

2 Related Work

Zero-Knowledge Proofs. Zero-knowledge proofs were invented by Goldwasser et al. [33], and later shown by Goldreich et al. [34] to exist for all languages in NP. Zero-knowledge arguments are similar to zero-knowledge proofs, but they have computational rather than statistical soundness. Kilian [35] showed that unlike zero-knowledge proofs, zero-knowledge arguments can be much smaller in size than the corresponding witness. Many zero-knowledge arguments are based upon the discrete log assumption. Schnorr presented the first such protocol [36], which was improved by Cramer and Damgard [37] to have linear communication complexity. Groth [31], Bayer and Groth [16] and Seo [38] have all constructed square root-size arguments. Bootle et al. [24] used techniques in [16] to reduce the communication complexity even further. The most efficient protocol currently is Bünz et al.'s Bulletproofs [25], which enables log-size arguments for arithmetic circuits on committed inputs, and has applications to range proofs, verifiable shuffles, and confidential transactions.

Some other approaches for Zero-Knowledge proofs include zk-SNARKS [39], zk-STARKS [40], Ligero [41], Hyrax [42], and ZKBoo [43]. zk-SNARKs, or Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge [39], enable constant-sized proofs and verification within a few milliseconds even for large programs. However, zk-SNARKs have significant drawbacks: they unavoidably rely on strong non-falsifiable assumptions such as the knowledge-of-exponent assumption, use a long and complex common reference string that requires a trusted third party or a computationally-intensive multi-party protocol, and are prover-inefficient.

In contrast, zk-STARKS, or Zero-Knowledge Scalabale Transparent Arguments of Knowledge [40], do not require a trusted setup and have logarithmic communication and verification complexity. However, they require a large Fast-Fourier Transform to make proving efficient and are thus quite memory-intensive. Hyrax [42], another variation of zk-SNARKs, achieves sublinear proof size and verification time and linear proof generation time. It relies on the discrete log assumption and does not require trusted setup.

Ligero [41] is another zero-knowledge argument protocol that is lightweight and does not require a trusted setup. Ligero enables proof size that is square-root in the verification circuit size and linear verification time. It only requires the assumption of collision-resistant hash functions and uses efficient symmetric-key cryptography. Ligero improves on ZKBoo [43] and uses the same technique of 'MPC in the Head.'

Spartan [44] is a zk-SNARK without trusted setup that only relies on standard assumptions. Verifying a proof incurs sub-linear costs without additional requirements on the structure of the circuit. However, as with all of the works described including our own, Spartan is not able to achieve logarithmic size proofs and sub-linear prover and verification cost at the same time.

Yun and Oleg [45] provide an implementation for Bulletproofs for adaptively-defined arithmetic circuits using a random challenge.

Recently, Hoffman et al. [29] constructed a zero-knowledge argument for satisfiability of a set of quadratic equations. Additionally, they identify techniques that have been implicitly used throughout this line of work in zero-knowledge arguments, including probabilistic verification and random linear combinations, which we employ as well.

Verifiable Shuffles. David Chaum introduced the idea of a shuffle in 1981 [1]. Sako and Killian [46] and Abe [47–49] were the firsts to present zero-knowledge arguments for the correctness of a shuffle. Furukawa and Sako [12] and Neff [10] concurrently proposed the first linear-size arguments for El Gamal-based shuffles. Their constructions used different approaches: Furukawa and Sako utilized permutation matrices, while Neff relied on the invariance of polynomials under permutation of its roots. Both of these ideas have been carried further in subsequent years: Wikstrom [14] refined the permutation matrices approach, while Groth and Ishai [13, 15] extended the polynomial invariance approach. Groth and Ishai presented the first sublinear arguments for a shuffle. In 2012, Bayer and Groth [16] used a new multi-exponentiation argument to produce arguments of size $O(\sqrt{n})$, where n is the size of the lists to be shuffled. In 2017, Bünz et al [25] introduced Bulletproofs, which enable zero-knowledge arguments that are logarithmic in the size of the circuit. For the application of verifiable shuffles, Bulletproofs employs a sorting circuit of size $O(n \log(n)$. Our verifiable shuffle uses a randomized polynomial identity test of circuit size $O(n)$, yielding concrete improvements in proof size. Concurrent work by Hoffman et al [29] also employs randomized functionality to construct a $O(\log(n))$-size verifiable shuffle argument. However, the verifiable shuffle they consider is the traditional single-key shuffle, whereas we consider a new multi-key shuffle of both keys and ciphertexts.

3 Preliminaries

3.1 Notation

We write $c = A(x; r)$ when the algorithm on input x and randomness r outputs c. We write $c \leftarrow A(x)$ to denote the process of choosing randomness r and setting $c = A(x; r)$, and also write $b \leftarrow S$ to denote sampling b uniformly at random from the set S. We will assume that one can sample uniformly at random from sets $\mathbb{Z}_p$ and $\mathbb{Z}_p^*$. Throughout the paper, we let $\mathbb{G}$ be a group of prime order p. We use bold font to denote vectors, $\mathbf{g} = (g_1, \ldots, g_n)$.

3.2 Assumptions

Let Setup be an algorithm that on input 1^λ returns $(\mathbb{G}, p, g)$ such that $\mathbb{G}$ is the description of a finite cyclic group of prime order p, where $|p| = \lambda$, and g is a generator of $\mathbb{G}$.

Definition 1 (Discrete Logarithm Assumption). *For all PPT adversaries $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$ such that*

$$\Pr\left[\begin{array}{l}(\mathbb{G}, p, g) \leftarrow_{\$} \mathsf{Setup}(1^\lambda), h \leftarrow_{\$} \mathbb{G}; \\ a \leftarrow \mathcal{A}(\mathbb{G}, p, g, h) : g^a = h\end{array}\right] \leq \mu(\lambda)$$

In this definition, a is called the discrete logarithm of h with respect to base g. It is well known that the discrete log assumption is equivalent to the following assumption.

Definition 2 (Discrete Log Relation). *For all PPT adversaries $\mathcal{A}$ and for all $n \geq 2$ there exists a negligible function $\mu(\lambda)$ such that*

$$\Pr\left[\begin{array}{l}\mathbb{G} = \mathsf{Setup}(1^\lambda), g_1 \ldots g_n \leftarrow_{\$} \mathbb{G}; \\ a_1 \ldots a_n \in \mathbb{Z}_p \leftarrow \mathcal{A}(\mathbb{G}, g_1, \ldots, g_n) : \exists a_i \neq 0 \wedge \prod_{i=1}^n g^{a_i} = 1\end{array}\right] \leq \mu(\lambda)$$

The Discrete Logarithm Relation assumption states that it is infeasible for an adversary to find a non-trivial discrete log relation between randomly chosen group elements, where a non-trivial relation between $g_1, \ldots, g_n$ is $\prod_{i=1}^n g_i^{a_i} = 1$.

3.3 Commitments

A non-interactive commitment scheme allows a sender to commit to a secret value by sending a commitment. Later, the sender may open the commitment and reveal the value in a verifiable manner. A commitment has two security properties: it should hide the secret value (hiding), and the sender should be able to open the commitment to only one value (binding).

Definition 3 (Commitment). *A non-interactive commitment scheme consists of a pair of probabilistic polynomial time algorithms* $(\mathsf{Setup}, \mathsf{Com})$. *The setup algorithm* $\mathsf{Setup}(1^\lambda)$ *generates public parameters* pp *for the scheme for security parameter* λ. *The commitment algorithm* $\mathsf{Com}_{\mathsf{pp}}$ *defines a function* $M_{\mathsf{pp}} \times R_{\mathsf{pp}} \to C_{\mathsf{pp}}$ *for message space* M_{pp}*, randomness space* R_{pp} *and commitment space* C_{pp} *determined by* pp. *For a message* $x \in M_{\mathsf{pp}}$*, the algorithm draws* $r \leftarrow_{\$} R_{\mathsf{pp}}$ *uniformly at random, and computes a commitment,* $C = \mathsf{Com}_{\mathsf{pp}}(x; r)$.

Definition 4 (Perfectly Hiding). *A non-interactive commitment scheme (*$\mathsf{Setup}, \mathsf{Com}$*) is perfectly hiding if for all PPT interactive adversaries* $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow_{\$} \mathsf{Setup}(1^\lambda), (m_0, m_1) \leftarrow_{\$} \mathcal{A}(\mathsf{pp}), \\ b \leftarrow_{\$} \{0,1\}, r \leftarrow_{\$} R_{\mathsf{pp}}, C = \mathsf{Com}_{\mathsf{pp}}(m_b; r) : \mathcal{A}(C) = b\end{array}\right] = \frac{1}{2}$$

where $m_0, m_1 \in M_{\mathsf{pp}}$.

Definition 5 (Computationally Binding). *A non-interactive commitment scheme is computationally binding if for all PPT interactive adversaries* $\mathcal{A}$,

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow_{\$} \mathsf{Setup}(1^\lambda), (m_0, m_1, r_0, r_1) \leftarrow_{\$} \mathcal{A}(pp) : \\ \mathsf{Com}_{\mathsf{pp}}(m_0; r_0) = \mathsf{Com}_{\mathsf{pp}}(m_1; r_1) \wedge m_0 \neq m_1\end{array}\right] \leq \mu(\lambda)$$

where $m_0, m_1 \in M_{\mathsf{pp}}$ *and* $r_0, r_1 \in R_{\mathsf{pp}}$.

Definition 6 (Homomorphic Commitments). *A homomorphic commitment scheme is a non-interactive commitment scheme such that* $M_{\mathsf{pp}}, R_{\mathsf{pp}}$*, and* C_{pp} *are all abelian groups, and for all* $x_1, x_2 \in M_{\mathsf{pp}}, r_1, r_2 \in R_{\mathsf{pp}}$,

$$\mathsf{Com}_{\mathsf{pp}}(x_1; r_1) + \mathsf{Com}_{\mathsf{pp}}(x_2; r_2) = \mathsf{Com}_{\mathsf{pp}}(x_1 + x_2; r_1 + r_2)$$

The Pedersen commitment scheme is a prominent instantiation of a homomorphic, perfectly hiding commitment scheme. We distinguish between the usual Pedersen scheme for commitments of single values, which we call Pedersen scalar commitments, and a variant that allows us to commit to multiple values at once, which we call Pedersen vector commitments.

Definition 7 (Pedersen Scalar Commitment). *For a Pedersen scalar commitment, let* $M_{\mathsf{pp}}, R_{\mathsf{pp}} = \mathbb{Z}_p, C_{\mathsf{pp}} = \mathbb{G}$ *of order p, and define* $(\mathsf{Setup}, \mathsf{Com})$ *as follows.*

$$\mathsf{Setup} : g, h \leftarrow_{\$} \mathbb{G}$$
$$\mathsf{Com}_{\mathsf{pp}}(x; r) = (g^x h^r)$$

Definition 8 (Pedersen Vector Commitment). *For a Pedersen vector commitment, let* $M_{\mathsf{pp}} = \mathbb{Z}_p^n, R_{\mathsf{pp}} = \mathbb{Z}_p, C_{\mathsf{pp}} = \mathbb{G}$ *of order p, and define* $(\mathsf{Setup}, \mathsf{Com})$ *as follows.*

$$\mathsf{Setup} : \mathbf{g} = (g_1, \ldots, g_n), h \leftarrow_{\$} \mathbb{G} \qquad \mathsf{Com}_{\mathsf{pp}}(\mathbf{x} = (x_1, \ldots, x_n); r) = h^r \prod_{i=1}^{n} g_i^{x_i} \in \mathbb{G}$$

Pedersen scalar and vector commitments are homomorphic, perfectly hiding, and computationally binding under the discrete log and discrete log relation assumptions, respectively. In our constructions, we will rely on specific properties of the Pedersen schemes.

3.4 Zero-Knowledge Arguments of Knowledge

A zero-knowledge proof is a protocol by which a prover convinces a verifier that some statement holds without revealing the witness for the statement. The protocol is an argument rather than a proof if the prover is computationally bounded and some computational hardness assumptions are used.

We will consider arguments consisting of three interactive algorithms, $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$, all running in probabilistic polynomial time. These are the common reference string generator Setup, the prover $\mathcal{P}$, and the verifier $\mathcal{V}$. On input 1^λ, algorithm Setup produces a common reference string σ. The transcript produced by $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs s and t is denoted by $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$. We write $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = b$ depending on whether the verifier rejects, $b = 0$, or accepts, $b = 1$. Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time ternary relation. Given σ, we call w a witness for a statement u if $(\sigma, u, w) \in \mathcal{R}$, and define the CRS-dependent language

$$L_\sigma = \{x \mid \exists w \;:\; (\sigma, x, w) \in \mathcal{R}\}$$

as the set of statements x that have a witness w in the relation $\mathcal{R}$.

Definition 9 (Argument of Knowledge). *The triple* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *is an argument of knowledge for a relation* $\mathcal{R}$ *if it has perfect completeness and computational witness-extended emulation, both defined below.*

Definition 10 (Public Coin). *An argument of knowledge* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *is called public coin if all messages sent from the verifier to the prover are chosen uniformly at random and independently of the prover's messages. In other words, the challenges correspond only to the verifier's randomness* ρ.

Definition 11 (Perfect Completeness). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *has perfect completeness if for all non-uniform polynomial time adversaries* $\mathcal{A}$

$$P\left[(\sigma, u, w) \notin \mathcal{R} \;\; or \;\; \langle \mathcal{P}(\sigma, u, w), \mathcal{V}(\sigma, u) \rangle = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda) \\ (u, w) \leftarrow \mathcal{A}(\sigma) \end{array} \right] = 1$$

where the or is exclusive.

The standard argument of knowledge definition of Bellare and Goldreich demands that a witness be extracted for all sufficiently long inputs. It cannot be used when the public keys are generated before the argument of knowledge (CRS model). This is because there is a nonzero probability that a cheating prover can compute a trapdoor from the public keys, so the prover could create common inputs (such as commitments using the public keys) for which it may be impossible to extract a witness.

Thus, to define an argument of knowledge, we use the notion of computational witness-extended emulation, which was used in [25]. It was also used in a statistical sense in [24] and originated from Groth and Ishai [13], who borrowed the term witness-extended emulation from Lindell [50].

Informally, whenever an adversary produces an argument that is accepted by the verifier with some probability, there exists an emulator $\mathcal{E}$ that produces a similar argument but also extracts a witness with almost the same probability. $\mathcal{E}$ can rewind the interaction and resume with the same internal state s for the prover, but with fresh randomness for the verifier. Whenever $\mathcal{P}$ produces a satisfying argument in state s, $\mathcal{E}$ can extract a witness, which implies knowledge soundness.

Definition 12 (Computational Witness-Extended Emulation). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *has witness-extended emulation if for all deterministic polynomial time $\mathcal{P}^*$ there exists an expected polynomial time emulator $\mathcal{E}$ such that for all pairs of interactive adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$, there exists a negligible function $\mu(\lambda)$ such that*

$$\left| \begin{array}{l} P\left[\mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, s) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, u, s), \mathcal{V}(\sigma, u)\rangle \end{array}\right] - \\ \\ P\left[\begin{array}{l} \mathcal{A}_1(tr) = 1 \\ \wedge (tr \textit{ is accepting } \rightarrow (\sigma, u, w) \in R \end{array} \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), \\ (u, s) \leftarrow \mathcal{A}_2(\sigma), \\ (tr, w) \leftarrow \mathcal{E}^{\mathcal{O}}(\sigma, u) \end{array}\right] \end{array} \right| \le \mu(\lambda)$$

where the oracle is given by $\mathcal{O} = \langle \mathcal{P}^*(\sigma, u, s), \mathcal{V}(\sigma, u)\rangle$ and permits rewinding to a specific point and resuming with fresh randomness for the verifier from this point onwards. We can also define computational witness-extended emulation by restricting to non-uniform polynomial time adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$.

The argument of knowledge that we build on [25] is special honest-verifier zero-knowledge, meaning that given the verifier's challenge values, it is possible to efficiently simulate the entire argument without knowing the witness.

Definition 13 (Perfect Special Honest-Verifier Zero-Knowledge). *A public coin argument of knowledge* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *is a perfect special honest verifier zero knowledge (SHVZK) argument of knowledge for $\mathcal{R}$ if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all pairs of interactive adversaries $\mathcal{A}_1, \mathcal{A}_2$*

$$P\left[(\sigma, u, w) \in R \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, u, w), \mathcal{V}(\sigma, u; \rho)\rangle \end{array}\right] =$$

$$P\left[(\sigma, u, w) \in R \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \mathcal{S}(\sigma, u, \rho) \end{array}\right]$$

In our constructions, we consider the setting in which the common reference string (CRS) may be generated adversarially by the verifier. Our arguments provide subversion zero-knowledge [28], which is zero-knowledge under subversion of the trusted parameters. Informally, the definition requires that for any adversary $\mathcal{A}_3$ creating a malicious CRS, there exists a simulator $\mathcal{S}$ returning a simulated transcript, such that no adversary $\mathcal{A}_1$ can distinguish between being given a malicious CRS and simulated transcript, and being given an honest CRS and real transcript.

Definition 14 (Subversion Zero-Knowledge). *A public coin argument of knowledge* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *is a subversion (perfect special honest-verifier) zero knowledge (SZK) argument of knowledge for* $\mathcal{R}$ *if for all triples of interactive adversaries* $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ *there exists a probabilistic polynomial time simulator* $\mathcal{S}$ *such that*

$$P\left[(\sigma, u, w) \in R \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, u, w), \mathcal{V}(\sigma, u; \rho) \rangle \end{array}\right] =$$

$$P\left[(\sigma, u, w) \in R \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathcal{A}_3(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \mathcal{S}(\sigma, u, \rho) \end{array}\right]$$

Now, we describe a simple algorithm to check whether the commitment parameters for a Pedersen commitment scheme are valid and well-formed.

```
if the CRS satisfies the following conditions: $\mathbb{G}$ is a group of prime order, $g \in \mathbb{G}$
  and $g \neq 1$ for all group elements $g$ in the CRS then
    Accept;
else
    Reject;
end
```

Algorithm 1: CRSCheck

The CRSCheck algorithm is trivial, but it ensures that Pedersen commitments remain hiding and subversion zero-knowledge holds. This is good news;it means that we only need a trivial algorithm to get a strong guarantee.

Lemma 1. *All Pedersen commitments or Pedersen vector commitments made by the prover using commitment parameters that pass the CRSCheck algorithm are perfectly hiding.*

We assume that the prover is not the one subverting the CRS, and that the prover does not know a discrete log relation between the commitment parameters [1]. Thus, soundness is not affected by the subverted CRS.

[1] In other words, our protocol assumes that there is no collusion between two or more users and the shuffle prover. We believe there are applications where such an assumption is a better match than the assumption of trusted setup for CRS generation, which is the existing alternative. The work of [28] shows that we cannot achieve subversion soundness along with zero-knowledge for any non-trivial relation, which makes our assumption necessary in an approach that relies on user-generated parameters in the CRS. It is an interesting and non-trivial open question whether we can match the efficiency of our current construction without using user-generated parameters in the CRS.

3.5 Verifiable Shuffles

Verifiable shuffles are cryptographic protocols that permute a list of ciphertexts and provide a zero-knowledge argument that the permutation was applied honestly. In this section, we formally define verifiable shuffles and describe the ciphertext shuffle most commonly seen in the literature [10]. We begin by defining a language for the shuffle relation.

Definition 15. *Suppose* $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is a public-key cryptosystem. Let* pp *be public parameters generated by* $\mathsf{Setup}(1^\lambda)$*,* pk *be a public key generated by* $\mathsf{KGen}(\mathsf{pp})$*, and* $\pi \in \Sigma_n$ *a permutation.* $L = (\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$ *and* $L' = (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_n)$ *are two lists of ciphertexts such that for all* $i \in [n]$*,* ct_i *and* $\mathsf{ct}'_{\pi(i)}$ *decrypt to the same message. Then, we define a language* $\mathcal{L}_{CS}$ *as follows.*

$$\mathcal{L}_{\mathsf{PKE}} = \left\{ (\mathsf{pp}, \mathsf{pk}, L, L') : \exists \pi \textit{ st. } \forall i, \mathsf{Dec}_{\mathsf{sk}}(\mathsf{pp}, \mathsf{ct}'_i) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{pp}, \mathsf{ct}_{\pi(i)}) \right\}$$

Next, we define correctness, verifiability, and privacy of the shuffle. As above, let the shuffle take as input a public key pk and list of input ciphertexts $L = \mathsf{ct}_1, \ldots, \mathsf{ct}_n$ and outputs a list of ciphertexts $L' = \mathsf{ct}'_1, \ldots, \mathsf{ct}'_n$. The shuffle is correct if for all $i \in [n]$, $m'_i = m_{\pi(i)}, r'_i = r_{\pi(i)}, \mathsf{Dec}_{\mathsf{sk}}(\mathsf{pp}, \mathsf{ct}'_i) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{pp}, \mathsf{ct}_{\pi(i)})$. The shuffle is verifiable if it runs an argument system to prove that the relation holds. Finally, the shuffle is private if it is infeasible for an adaptive adversary to distinguish transcripts of two shuffle executions that correspond to two different permutations. These properties are defined formally below.

Definition 16 (Verifiable Shuffle). *A verifiable shuffle* VS *is a triple,* $(\mathsf{PKE}, \mathsf{Shuffle}, \langle \mathcal{P}, \mathcal{V} \rangle)$*, such that*

- PKE *is the public-key cryptosystem defined in Definition 15.*
- $\mathsf{Shuffle}_\pi$ *is a probabilistic polynomial-time algorithm that takes as input public parameters* pp*, a public key* pk*, and a list of* n *ciphertexts* $L = \mathsf{ct}_1, \ldots, \mathsf{ct}_n$*, applies the permutation* π*, and outputs a list of* n *ciphertexts* $L' = \mathsf{ct}'_1, \ldots, \mathsf{ct}'_n$*.*
- $\langle \mathcal{P}, \mathcal{V} \rangle$ *is an interactive argument system that, on public input* $(\mathsf{pp}, \mathsf{pk}, L, L')$ *and private input* π *for* $\mathcal{P}$*, proves that* $(\mathsf{pp}, \mathsf{pk}, L, L') \in \mathcal{L}_{\mathsf{PKE}}$*. Note that the private input to* $\mathcal{P}$ *does not include the secret key* sk*.*

Definition 17 (Correctness). *The verifiable shuffle* $\mathsf{VS} = (\mathsf{PKE}, \mathsf{Shuffle}, \langle \mathcal{P}, \mathcal{V} \rangle)$ *is correct if: whenever* $(\mathsf{pp}, \mathsf{pk}, L, L') \in \mathcal{L}_{\mathsf{PKE}}$*,* $\langle \mathcal{P}(\mathsf{pp}, \mathsf{pk}, L, L'; \pi), \mathcal{V}(\mathsf{pp}, \mathsf{pk}, L, L') \rangle$ *returns* 1*.*

Definition 18 (Security). *The verifiable shuffle* $\mathsf{VS} = (\mathsf{PKE}, \mathsf{Shuffle}, \langle \mathcal{P}, \mathcal{V} \rangle)$ *is secure if*

- $\langle \mathcal{P}, \mathcal{V} \rangle$ *is an argument for the language* $\mathcal{L}_{\mathsf{PKE}}$ *with computational witness-extended emulation.*
- $\mathsf{Shuffle}_\pi$ *satisfies* IND-CPA$_{\mathsf{VS}}$ *privacy, as defined below.*

We define privacy of the shuffle using Nguyen et al.'s [51] notion of indistinguishability under chosen permutation attack (IND-CPA$_{\mathsf{VS}}$ security). The security game is detailed below.

Definition 19 (Privacy). *Let L, L' be defined as in Definition 15, and let $\mathcal{A}$ be an interactive adversary.*

```
IND-CPA$^{\mathcal{A}}_{\mathsf{VS}}$
1 :  $\mathsf{pp} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
2 :  $\mathsf{pk} \leftarrow_\$ \mathsf{KGen}(\mathsf{pp})$
3 :  $(m_1, \ldots, m_n), L, \pi_0, \pi_1 \leftarrow \mathcal{A}^{\mathsf{KGen},\mathsf{Enc}}(\mathsf{pp}, \mathsf{pk})$
4 :  $b \leftarrow_\$ \{0, 1\}$
5 :  $L' \leftarrow_\$ \mathsf{Shuffle}_{\pi_b}(\mathsf{pp}, \mathsf{pk}, L)$
6 :  $b' \leftarrow_\$ \mathcal{A}(\mathsf{pp}, \mathsf{pk}, L, L')$
7 :  $\mathbf{return}\ b = b'$
```

The adversary's advantage in this game is $\left|P[b = b'] - \frac{1}{2}\right|$. We say the verifiable shuffle has privacy if the adversary's advantage is negligible in the security parameter λ.

4 Multi-Key Verifiable Shuffle

In this section, we introduce a novel shuffle primitive and construction that enables shuffling of public keys, along with shuffling of ciphertexts encrypted under these public keys. We call this multi-key shuffling.

4.1 Shuffle Primitive

We begin by defining a language for the shuffle relation.

Definition 20. *Suppose* $\mathsf{RPKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKey}, \mathsf{ReEnc})$ *is a public-key cryptosystem with re-key and re-encryption functions. Let* M_{pp} *be the message space,* R_{pp} *the randomness space, and* C_{pp} *the ciphertext space determined by public parameters* pp. *Let L, L' be two lists of tuples*

$$L_i = (\mathsf{pk}_i, \mathsf{Enc}_{\mathsf{pk}_i}(m_i, r_i))$$
$$L'_i = (\mathsf{pk}'_i, \mathsf{Enc}_{\mathsf{pk}'_i}(m'_i, r'_i))$$

where $m_i \in \mathsf{M}_{\mathsf{pp}}, m'_i \in \mathsf{M}_{\mathsf{pp}'}$ *are messages and* $r_i \in \mathsf{R}_{\mathsf{pp}'}, r'_i \in \mathsf{R}_{\mathsf{pp}'}$ *are randomness for the encryptions. Let* $\pi \in \Sigma_n$ *be a permutation and* $s \in \mathsf{R}_{\mathsf{pp}}$ *be randomness for the re-key and re-encryption algorithms. Then, we define a language* $\mathcal{L}_{\mathsf{RPKE}}$ *of tuples* $(\mathsf{pp}, \mathsf{pp}', L, L')$ *such that for all* $i \in [n]$, $\mathsf{pk}'_i = \mathsf{ReKey}(\mathsf{pk}_{\pi(i)}, s)$ *and* $\mathsf{Enc}_{\mathsf{pk}_{\pi(i)}}(m_{\pi(i)}, r_{\pi(i)})$ *and* $\mathsf{Enc}_{\mathsf{pk}'_i}(m'_i, r'_i)$ *are encryptions of the same message with the same randomization.*

$$\begin{aligned}\mathcal{L}_{\mathsf{RPKE}} = \big\{(&\mathsf{pp}, \mathsf{pp}', \{\mathsf{pk}_i, \mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{pp}, m_i, r_i)\}_{i=1}^n, \{\mathsf{pk}'_i, \mathsf{Enc}_{\mathsf{pk}'_i}(\mathsf{pp}, m'_i, r'_i)\}_{i=1}^n : \exists \pi, s \textit{ st.} \\ &\forall i, \mathsf{pk}'_i = \mathsf{ReKey}(\mathsf{pp}, \mathsf{pk}_{\pi(i)}, s), m'_i = m_{\pi(i)}, r'_i = r_{\pi(i)}, \\ &\mathsf{Enc}_{\mathsf{pk}'_i}(\mathsf{pp}, m'_i, r'_i) = \mathsf{ReEnc}_{\mathsf{pp},\mathsf{pk}'_i}(\mathsf{Enc}_{\mathsf{pk}_{\pi(i)}}(\mathsf{pp}, m_{\pi(i)}, r_{\pi(i)}), s)\big\}\end{aligned}$$

Note that the relation presented here strictly implies the relation in Definition 15, used for single-key verifiable shuffles. In this relation, we are constraining not only the output plaintexts m'_i, but also the randomness r'_i used in the output ciphertexts.

Definition 21 (Multi-Key Verifiable Shuffle). *Let us assume access to* RPKE *and zero-knowledge schemes. A multi-key verifiable shuffle* MKS *is a triple,* $(\mathsf{RPKE}, \mathsf{Shuffle}, \langle \mathcal{P}, \mathcal{V} \rangle)$*, such that*

- RPKE *is the public-key cryptosystem with re-encrypt and re-encryption functions defined in Definition 20.*
- $\mathsf{Shuffle}_\pi$ *is a probabilistic polynomial-time algorithm that takes as input public parameters* pp *and a list of n tuples* $L = \{(\mathsf{pk}_i, \mathsf{ct}_i)_{i=1}^n\}$*, applies the permutation π, and outputs new public parameters* pp' *and a list of n ciphertexts* $L = \{(\mathsf{pk}'_i, \mathsf{ct}'_i)_{i=1}^n\}$.
- $\langle \mathcal{P}, \mathcal{V} \rangle$ *is an argument system that, on public input* $(\mathsf{pp}, \mathsf{pp}', L, L')$ *and private input π to $\mathcal{P}$, proves that* $(\mathsf{pp}, \mathsf{pp}', L, L') \in \mathcal{L}_{\mathsf{RPKE}}$. *The private input to $\mathcal{P}$ does not include the secret key* sk.

MKS is correct if for all $i \in [n]$, pk'_i is a re-key of $\mathsf{pk}_{\pi(i)}$, and ct'_i and $\mathsf{ct}_{\pi(i)}$ decrypt to the same message and have the same randomness when viewed as encryptions under their respective public keys, pk'_i and $\mathsf{pk}_{\pi(i)}$. MKS is verifiable if the argument system $\langle \mathcal{P}, \mathcal{V} \rangle$ satisfies completeness, soundness, and zero-knowledge properties. Finally, MKS is private if it is infeasible for a probabilistic polynomial-time adversary to distinguish transcripts of two shuffle executions that correspond to two different permutations. These properties are defined formally below.

Definition 22 (Correctness). $\mathsf{MKS} = (\mathsf{RPKE}, \mathsf{Shuffle}, \langle \mathcal{P}, \mathcal{V} \rangle)$ *is correct if* $(\mathsf{pp}, \mathsf{pp}', L, L') \in \mathcal{L}_{\mathsf{RPKE}}$.

Definition 23 (Security). $\mathsf{MKS} = (\mathsf{RPKE}, \mathsf{Shuffle}, \langle \mathcal{P}, \mathcal{V} \rangle)$ *is secure if*

- $\langle \mathcal{P}, \mathcal{V} \rangle$ *satisfies perfect completeness, subversion zero-knowledge, and computational witness-extended emulation.*
- $\mathsf{Shuffle}_\pi$ *satisfies* IND-CPA$_{\mathsf{MKS}}$ *privacy, as defined below.*

For privacy, we modify Nguyen et al.'s IND-CPA$_{\mathsf{VS}}$ notion for permutation indistinguishability. Our security notion allows the adversary to select a subset of public keys in addition to the messages and permutations. The security game for IND-CPA$_{\mathsf{MKS}}$ is detailed below.

Definition 24 (Privacy). *Let L, L' be defined as in Definition 20 and let $\mathcal{A}$ be an interactive adversary. The security parameter is λ.*

```
IND-CPA$^{\mathcal{A}}_{\mathsf{MKS}}$
1 :  $\mathsf{pp} \leftarrow_\$ \mathsf{Setup}(1^\lambda)$
2 :  $T \subset [n] \leftarrow_\$ \mathcal{A}(n)$
3 :  $m_{i\in[n]}, \{\mathsf{pk}_i, \mathsf{ct}_i\}_{i\in T}, \pi_0, \pi_1 : \forall i \in T, \pi_0(i) = \pi_1(i) \leftarrow \mathcal{A}^{\mathsf{KGen},\mathsf{Enc}}(\mathsf{pp})$
4 :  $(\mathsf{pk}_i)_{i\notin T} \leftarrow_\$ \mathsf{KGen}(\mathsf{pp})$
5 :  $(\mathsf{ct}_i)_{i\notin T} \leftarrow_\$ \mathsf{Enc}_{\mathsf{pk}_i}(\mathsf{pp}, m_i)$
6 :  $L = \{\mathsf{pk}_i, \mathsf{ct}_i\}_{i\in[n]}$
7 :  $b \leftarrow_\$ \{0,1\}$
8 :  $\mathsf{pp}', L', = \mathsf{Shuffle}_{\pi_b}(\mathsf{pp}, L)$
9 :  $b' \leftarrow_\$ \mathcal{A}(\mathsf{pp}, \mathsf{pp}', L, L')$
10 : $\mathbf{return}\ b = b'$
```

The adversary's advantage of winning the game is $\mathsf{Adv}^{\text{ind-cpa}_{\mathsf{mks}}}_{\mathcal{A}}(\lambda) = \left|P[b = b'] - \frac{1}{2}\right|$. *The multi-key verifiable shuffle* $\mathsf{MKS} = (\mathsf{RPKE}, \mathsf{Shuffle}, \langle \mathcal{P}, \mathcal{V}\rangle)$ *is private if, for all interactive adversaries* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\text{ind-cpa}_{\mathsf{mks}}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$$

4.2 Shuffle Construction in the Discrete Log Setting

We describe our instantiation of the multi-key verifiable shuffle primitive in the discrete log setting. Recall that the primitive consists of a cryptosystem RPKE, a shuffle algorithm $\mathsf{Shuffle}$, and an argument system $\langle \mathcal{P}, \mathcal{V}\rangle$.

We begin by describing the cryptosystem
$\mathsf{RPKE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKey}, \mathsf{ReEnc})$, which is the El Gamal additively homomorphic encryption scheme with re-key and re-encryption functions. The cryptosystem is described below.

```
Setup(1^λ):
    Return $\mathsf{pp} := (G, p, g)$

KGen(pp):
    $x \leftarrow_\$ \mathbb{Z}_p$
    $\mathsf{sk} := x$
    $\mathsf{pk} := g^x$
    Return $(\mathsf{pk}, \mathsf{sk})$

Enc_{pk}(pp, m):
    $r \leftarrow_\$ \mathbb{Z}_p^*$
    $\mathsf{ct}_1 := g^r$
    $\mathsf{ct}_2 := g^m \cdot \mathsf{pk}^r$
    Return $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2)$

Dec_{sk}(pp, ct):
    $y := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}}$
    $m := \log_g(y)$
    Return $m$

ReKey(pp, pk, s):
    $\mathsf{pk}' := \mathsf{pk}^s$
    Return $\mathsf{pk}'$

ReEnc_{pk'}(pp, ct, s):
    $\mathsf{ct}' := \mathsf{ct}^s$
    Return $\mathsf{ct}'$
```

Note that, identical to the additively homomorphic El Gamal scheme, decryption requires brute force computation of a discrete log. Typically, this is made possible by restricting the message space such that brute force computation is feasible. In our application, however, the holder of the secret key will always know the randomness applied to the ciphertext, so she will be able to recover the message easily.

Next, we describe the shuffle algorithm Shuffle for our instantiation. Shuffle takes as input (pp, L), where pp includes g, the generator for the scheme, and L is a list of tuples, each consisting of a public key, h_i, and an El Gamal additively homomorphic ciphertext, (b_i, c_i), which is an encryption under h_i, and outputs (g', L'), a new generator and list of tuples.

```
Shuffle_π(pp, L):
  – $s \leftarrow_\$ \mathbb{Z}_p^*$
  – $\forall i, h'_i := \mathsf{ReKey}(\mathsf{pp}, h_{\pi(i)}, s)$
  – $\forall i, (b'_i, c'_i) := \mathsf{ReEnc}_{h'_i}(\mathsf{pp}, (b_{\pi(i)}, c_{\pi(i)}), s)$
  – $L' := \{h'_i, b'_i, c'_i\}_{i=1}^n$
  – $g' := g^s$
  – $\mathsf{pp}' = (G, p, g')$
  – Output (pp', L')
```

The construction of the argument system $\langle \mathcal{P}, \mathcal{V} \rangle$ is detailed in Section 4.3.

Correctness. We show that $(\mathsf{pp}, \mathsf{pp}', L, L') \in \mathcal{L}_{\mathsf{RPKE}}$. First, for all i, $h'_i = h^s_{\pi(i)} = \mathsf{ReKey}(h_{\pi(i)}, s)$. Next, for all i, $(b'_i, c'_i) = (b^s_{\pi(i)}, c^s_{\pi(i)})$. Therefore,

$$\begin{aligned}(b'_i, c'_i) &= (b^s_{\pi(i)}, c^s_{\pi(i)})\\ &= ((g^{r_{\pi(i)}})^s, (g^{m_{\pi(i)}} h^{r_{\pi(i)}}_{\pi(i)})^s)\\ &= ((g^s)^{r_{\pi(i)}}, (g^s)^{m_{\pi(i)}} (h^s_{\pi(i)})^{r_{\pi(i)}}\\ &= ((g')^{r'_i}, (g')^{m'_i} (h'_i)^{r'_i})\end{aligned}$$

Thus, for all i, $m'_i = m_{\pi(i)}, r'_i = r_{\pi(i)}$, and $(b'_i, c'_i) = \mathsf{ReEnc}_{h'_i}((b_{\pi(i)}, c_{\pi(i)}), s)$. Therefore, $(\mathsf{pp}', \mathsf{pp}, L, L') \in \mathcal{L}_{\mathsf{RPKE}}$ and correctness holds.

Verifiability. Let g, g' and the lists L, L' be public inputs, and let π, s be private inputs to $\mathcal{P}$. The discrete logarithms of any of the group elements in the lists are unknown to the $\mathcal{P}$. $\mathcal{P}$ must produce an argument claiming that she knows a randomizer s and a permutation π such that for all $1 \leq i \leq n$,

$$(h'_i, b'_i, c'_i) = (h^s_{\pi(i)}, b^s_{\pi(i)}, c^s_{\pi(i)}) \tag{1}$$

The argument system must have perfect completeness, subversion zero-knowledge, and computational witness-extended emulation. In the next section (Section 4.3), we show that the argument system for this shuffle satisfies these properties.

Privacy. Intuitively, the privacy of the multi-key shuffle follows from the semantic security of El Gamal encryption and the hiding property of Pedersen commitments, or more generally, from the hardness of Decisional Diffie-Hellman (DDH).

Theorem 1. *Under the DDH hardness assumption, the multi-key shuffle presented in Section 4.2 has* IND-CPA$_{\mathsf{MKS}}$ *privacy, as defined in Definition 24.*

4.3 Shuffle Argument Protocol

In Section 4.2, we introduced an instantiation of the multi-key verifiable shuffle in the discrete log setting. We detailed the first two components of the shuffle: the cryptosystem RPKE, a variation on the El Gamal additively homomorphic scheme, and the Shuffle algorithm. In this section, we will describe the third component of the multi-key verifiable shuffle: the argument system $\langle \mathcal{P}, \mathcal{V} \rangle$.

Recall that the input and output lists of the Shuffle algorithm are $g, \{h_i, b_i, c_i\}_{i=1}^n$ and $g', \{h'_i, b'_i, c'_i\}_{i=1}^n$, where g, g' are generators, h_i, h'_i are public keys, and $(b_i, c'_i), (b'_i, c'_i)$ are El Gamal ciphertexts (with messages in the exponent). Thus, the argument system $\langle \mathcal{P}, \mathcal{V} \rangle$ should argue knowledge of a permutation $\pi \in \Sigma_n$ and randomness $s \in \mathbb{Z}_p^*$ such that $g' = g^s$ and for all $i \in [n]$, $h'_i = h^s_{\pi(i)}, b'_i = b^s_{\pi(i)}$, and $c'_i = c^s_{\pi(i)}$.

Note that when we expand our notation of the public keys and ciphertexts, we have that: $h_i = g^{x_i}, b_i = g^{\gamma_i}, c_i = g^{m_i} h_i^{\gamma_i}$, and $h'_i = g^{x'_i}, b'_i = g^{\gamma'_i}, c'_i = g^{m'_i} h'^{\gamma_i}_i$. Thus, we want to show that for all i, $x'_i = s \cdot x_{\pi(i)}, \gamma'_i = s \cdot \gamma_{\pi(i)}$, and $m'_i + x'_i \cdot \gamma_i = s \cdot m_{\pi(i)} + s \cdot x_{\pi(i)} \gamma_i$.

Below, we give an overview of how we transform the inputs to the argument protocol into general linear constraints that can be passed into a modified Bulletproofs[2] sub-routine.

Overview. The first step is for $\mathcal{V}$ to sample random challenges $r, u \leftarrow_{\$} \mathbb{Z}_p^*$. Using the outputs of the shuffle, $\mathcal{V}$ can create a commitment, $A_{R,h}$, to a random linear combination of the output secret keys, $\{x'_i\}_{i=1}^n$.

Let $\mathbf{k} = (k_1, k_2, \ldots, k_n)$, where $k_j = r - u^j$. Let $\hat{k} = \prod_{j=0}^{n-1} k_j$ and $k^* = \sum_{j=1}^n k_j$.

$$A_{R,h} = \prod_i {h'_i}^{k_i} = g^{x'_1 k_1 + \ldots + x'_n k_n}$$

Similarly, $\mathcal{V}$ can create a commitment, $A_{R,b}$, to a random linear combination of the randomizers for the output ciphertexts, $\{\gamma'_i\}_{i=1}^n$.

$$A_{R,b} = \prod_i {b'_i}^{k_i} = g^{\gamma'_1 k_1 + \ldots + \gamma'_n k_n}$$

Finally, let d'_i be the discrete log of c_i with respect to the public generator g. $\mathcal{V}$ can create a commitment, $A_{R,c}$, to a random linear combination of $\{d'_i\}_{i=1}^n$.

$$A_{R,c} = \prod_i {c'_i}^{k_i} = g^{d'_1 k_1 + \ldots + d'_n k_n}$$

Now, let $\mathbf{a}_R$ be the vector of discrete logs in $A_{R,h}$ with respect to bases $\{h_i\}_{i=1}^n$. $\mathcal{P}$ needs to argue that $\mathbf{a}_R$ satisfies two conditions:

1. $\prod_i a_{Ri} = s^n \prod_i k_i$. This shows, using a polynomial identity test, that there exists $\pi \in \Sigma_n$ such that a_{Ri} contains $c_i \cdot k_{\pi^{-1}(i)})$,where c_i is any constant.
2. $\sum_i a_{Ri} = s \sum_i k_i$. This shows, using a random linear combination check, that there exists an $s \in \mathbb{Z}_p^*$ such that for all a_{Ri}, c_i is equal to s.
3. $\mathbf{a}_R$ is the committed value in $A_{R,h}$ with respect to bases $\{h_i\}_{i=1}^n$, as well as the committed value in $A_{R,b}$ with respect to bases $\{b_i\}_{i=1}^n$, and in $A_{R,c}$ with respect to bases $\{c_i\}_{i=1}^n$.

If $\mathbf{a}_R$ satisfies these conditions, $\mathcal{V}$ can deduce that $\mathbf{a}_R$ takes the following form: $(sk_{\pi^{-1}(1)}, ..., sk_{\pi^{-1}(n)})$. This implies that $A_{R,h}$ is a commitment to $\sum_i x_i \cdot s \cdot k_{\pi^{-1}(i)}$ using bases $\{h_i\}_{i=1}^n$, which is a random linear combination, using permuted coefficients, of $\{x_i \cdot s\}_{i=1}^n$. In addition, the third condition implies that $A_{R,b}$ and $A_{R,c}$ are commitments to a random linear combination (using the same permuted coefficients) of $\{\gamma_i \cdot s\}_{i=1}^n$ using bases $\{b_i\}_{i=1}^n$, and $\{(m_i + x_i\gamma_i) \cdot s\}_{i=1}^n$ using bases $\{c_i\}_{i=1}^n$, respectively.

We now know that $A_{R,h}, A_{R,b}, A_{R,c}$ are commitments with two representations each. The first representation holds due to $\mathbf{a}_R$ being well-formed with respect to bases

[2] A brief overview of Bulletproofs is given in Section 5, but for a full description, see the original paper [25].

from the input lists. The second holds due to the verifier's construction of the commitment using bases from the output lists.

$$A_{R,h} = \prod_i h_i^{a_{Ri}} = g^{\sum_i x_i \cdot s \cdot k_{\pi^{\text{-}1}(i)}} \qquad A_{R,b} = \prod_i b_i^{a_{Ri}} = g^{\sum_i x'_i \cdot k_{\pi^{\text{-}1}(i)}}$$

$$A_{R,c} = \prod_i c_i^{a_{Ri}} = g^{\sum_i (m_i + x_i \gamma_i) \cdot s \cdot k_{\pi^{\text{-}1}(i)}}$$

First, by equating the exponents in the two representations of $A_{R,h}$, we obtain that a random linear combination of $\{x'_i\}_{i=1}^n$, using coefficients $\{k_i\}_{i=1}^n$, is equal to a random linear combination of $\{x_i \cdot s\}_{i=1}^n$, using permuted coefficients $\{k_{\pi^{\text{-}1}(i)})\}_{i=1}^n$.

$$\sum_i x_i \cdot s \cdot k_{\pi^{\text{-}1}(i)}) = \sum_i x'_i \cdot k_i$$
$$\sum_i x_{\pi(i)} \cdot s \cdot k_i = \sum_i x'_i \cdot k_i$$

which means that with overwhelming probability, $s \cdot x_{\pi(i)} = x'_i$. By similar arguments, we can deduce that, with overwhelming probability, $s \cdot \gamma_{\pi(i)} = \gamma'_i$ and that $s \cdot (m_{\pi(i)} + x_{\pi(i)} \cdot \gamma_{\pi(i)}) = d'_i$. If we let $d'_i = m'_i + x'_i \gamma_{\pi(i)}$ (ie. the output ciphertext is an encryption using the output public key and input randomizer), the latter equality yields the following.

$$s \cdot (m_{\pi(i)} + x_{\pi(i)} \cdot \gamma_{\pi(i)}) = d'_i$$
$$s \cdot m_{\pi(i)} = d'_i - x'_i \cdot \gamma_{\pi(i)} = m'_i$$

Thus, we can deduce that $s \cdot m_{\pi(i)} = m'_i$.

Arguing that $\mathbf{a}_R$ is well-formed. To argue that $\mathbf{a}_R$ satisfies the sum and product conditions listed above, our protocol will invoke the Bulletproofs sub-routine, described below under the heading, 'Bulletproofs sub-routine.' However, we will first need to reduce the conditions to equations that are either a Hadamard (entry-wise) product or a linear combination of vectors. The sum condition, $\sum_i a_{Ri} = s \cdot \sum_i k_i$, is already in the latter form. However, the product condition, $\prod_i a_{Ri} = s^n \prod_i k_i$ requires some manipulation.

To represent the product condition as a Hadamard product of two vectors, we will use a helper vector, $\mathbf{a}_L \in \mathbb{Z}_p^n$. $\mathbf{a}_L$ is constructed recursively as follows.

$$a_{L1} = 1 \quad \wedge \quad a_{Li} = a_{Li-1} \cdot a_{Ri-1}$$

Using $\mathbf{a}_L$, we can now represent $\prod_i a_{Ri}$ as $a_{Ln} \cdot a_{Rn} = \prod_i a_{Ri}$. Putting it all together, the following equations capture the sum and product conditions on $\mathbf{a}_R$, and can be passed to the Bulletproofs sub-routine.

$$a_{L_1} = 1 \ \wedge \ a_{L_i} = a_{L_{i-1}} \cdot a_{R_{i-1}} \ \wedge$$
$$a_{L_n} a_{R_n} = s^n \cdot \prod_i (k_i) \ \wedge \ \sum_{i=1}^n a_{R_i} = s \cdot \sum_i (k_i)$$

Note that the argument system will need to receive commitments to s and s^n as public input (denoted by W and V). A simple protocol, similar to the shuffle protocol but not shown here, can be used to argue that V is well-formed with respect to W. //
Bulletproofs sub-routine. The Bulletproofs sub-routine works as follows: given linear constraints for vectors $\mathbf{a}_L$ and $\mathbf{a}_R$, along with commitments to $\mathbf{a}_L$, $\mathbf{a}_R$, and any private target values used in the constraints, the sub-routine will run a logarithmic-size zero-knowledge argument that the committed vectors and values satisfy the linear constraints. Note that this sub-routine is slightly modified from the Bulletproofs paper [25] in order to fit our setting.

The full interactive protocol between $\mathcal{P}$ and $\mathcal{V}$ is given below:

Statement: $\forall i \in [n]\ (h^s_{\pi(i)}, b^s_{\pi(i)}, c^s_{\pi(i)}) = (h'_i, b'_i, c'_i)$
CRS: $g \in \mathbb{G}, \mathbf{g} \in \mathbb{G}^n, \mathbf{h}, \mathbf{b}, \mathbf{c} \in \mathbb{G}^n$
Public input: $\mathbf{h}', \mathbf{b}', \mathbf{c}' \in \mathbb{G}^n, W, V \in \mathbb{G}, n \in \mathbb{Z}$
Private input: $s \in \mathbb{Z}_p^*, \pi \in \Sigma_n$

1. $\mathcal{P}$ **:** If CRS does not pass CRSCheck, return $\perp$.
2. $\mathcal{V} \rightarrow \mathcal{P}$ **:** $r, u \leftarrow_\$ \mathbb{Z}_p^*$
3. $\mathcal{P}$ **and** $\mathcal{V}$ **:**
   - $k_i = (r - u^i) \quad i \in [n]$
   - $A_{R,h} = \prod_{i=1}^n {h'_i}^{k_i}$
   - $A_{R,b} = \prod_{i=1}^n {b'_i}^{k_i}$
   - $A_{R,c} = \prod_{i=1}^n {c'_i}^{k_i}$
4. $\mathcal{P}$ **:**
   - $\mathbf{a}_R = (sk_{\pi^{-1}(1)}, \dots, sk_{\pi^{-1}(n)})$
   - $\mathbf{a}_L = (1, sk_{\pi^{-1}(1)}, s^2 k_{\pi^{-1}(1)} k_{\pi^{-1}(2)}, \dots, s^n k_{\pi^{-1}(1)} \dots k_{\pi^{-1}(n)})$
   - $\alpha, \beta, \rho_h, \rho_b, \rho_c \leftarrow_\$ \mathbb{Z}_p$
   - $\mathbf{s}_L, \mathbf{s}_R \leftarrow_\$ \mathbb{Z}_p^n$
5. $\mathcal{P} \rightarrow \mathcal{V}$ **:**
   - $A_L = \mathbf{g}^{\mathbf{a}_L} h^\alpha$
   - $S_L = \mathbf{g}^{\mathbf{s}_L} h^\beta$
   - $S_{R,h} = \mathbf{h}^{\mathbf{s}_R} h^{\rho_h}$
   - $S_{R,b} = \mathbf{b}^{\mathbf{s}_R} h^{\rho_b}$
   - $S_{R,c} = \mathbf{c}^{\mathbf{s}_R} h^{\rho_c}$
6. $\mathcal{V} \rightarrow \mathcal{P}$ **:** $y, z \leftarrow_\$ \mathbb{Z}_p^*$
7. $\mathcal{P}$ **and** $\mathcal{V}$ **:**
   - $\mathbf{y}^n = (1, y, y^2, ..., y^n)$
   - $\mathbf{y}_z^{n-2} = (z, 1, \dots, y^{n-2})$
   - $\delta(y, z) =< z^3 \mathbf{y}^{-n}, \mathbf{y}_z^{n-2} > -z$
8. $\mathcal{P}$ **:**
   - $l(X) = \mathbf{a}_L + z^3 \mathbf{y}^{-n} + \mathbf{s}_L X^2$
   - $r(X) = (\mathbf{a}_R X + \mathbf{s}_R X^3) \circ \mathbf{y}^n - X \mathbf{y}_z^{n-2}$
   - $t(X) =< l(X), r(X) >= \sum_i t_i X^i$

- $t_1 = s^n \hat{k} y^{n-1} - sz^3 < \mathbf{1}, \mathbf{k} > +\delta(y,z)$
- $\tau_i \leftarrow_\$ \mathbb{Z}_p,\ i = \{0,2,3,4,5\}$
- $T_i = g^{t_i} h^{\tau_i},\ i = \{0,2,3,4,5\}$

9. $\boldsymbol{\mathcal{V} \rightarrow \mathcal{P}}$ **:** $x \leftarrow_\$ \mathbb{Z}_p^*$
10. $\boldsymbol{\mathcal{P} \rightarrow \mathcal{V}}$ **:**
    - $\mathbf{l} = l(x)$
    - $\mathbf{r} = r(x)$
    - $\hat{t} =< \mathbf{l}, \mathbf{r} >$
    - $\tau_x = \tau_0 + \tau_2 x^2 + \tau_3 x^3 + \tau_4 x^4$
    - $\mu_h = \alpha + \beta x^2 + \rho_h x^3$
    - $\mu_b = \alpha + \beta x^2 + \rho_b x^3$
    - $\mu_c = \alpha + \beta x^2 + \rho_c x^3$
11. $\boldsymbol{\mathcal{V}}$ **:**
    - $h_i^* = h_i^{y^{-i}} \quad i \in [n]$
    - $b_i^* = b_i^{y^{-i}} \quad i \in [n]$
    - $c_i^* = c_i^{y^{-i}} \quad i \in [n]$
    - $g^{\hat{t}} \cdot h^{\tau_x} \stackrel{?}{=} V^{xy^{n-1}\hat{k}} \cdot g^{x\delta(y,z)} \cdot T_o \cdot T_2^{x^2} \cdot T_3^{x^3} \cdot T_4^{x^4} \cdot T_5^{x^5}$
    - $h^{\mu_h} \cdot \mathbf{g}^{\mathbf{l}} \cdot (\mathbf{h}^*)^{\mathbf{r}} \stackrel{?}{=} A_L \cdot A_{R,h}^x \cdot S_L^{x^2} \cdot S_{R,h}^{x^3} \cdot \mathbf{g}^{z^3 \mathbf{y}^{-n}} \cdot (\mathbf{h}^*)^{-x \cdot \mathbf{y}_z^{n-2} \circ \mathbf{y}^n}$
    - $h^{\mu_b} \cdot \mathbf{g}^{\mathbf{l}} \cdot (\mathbf{b}^*)^{\mathbf{r}} \stackrel{?}{=} A_L \cdot A_{R,b}^x \cdot S_L^{x^2} \cdot S_{R,b}^{x^3} \cdot \mathbf{g}^{z^3 \mathbf{y}^{-n}} \cdot (\mathbf{b}^*)^{-x \cdot \mathbf{y}_z^{n-2} \circ \mathbf{y}^n}$
    - $h^{\mu_c} \cdot \mathbf{g}^{\mathbf{l}} \cdot (\mathbf{c}^*)^{\mathbf{r}} \stackrel{?}{=} A_L \cdot A_{R,c}^x \cdot S_L^{x^2} \cdot S_{R,c}^{x^3} \cdot \mathbf{g}^{z^3 \mathbf{y}^{-n}} \cdot (\mathbf{c}^*)^{-x \cdot \mathbf{y}_z^{n-2} \circ \mathbf{y}^n}$
    - $\hat{t} \stackrel{?}{=} \langle \mathbf{l}, \mathbf{r} \rangle$
12. $\boldsymbol{\mathcal{V}}$ **:** Accept or Reject.

Multi-Key Shuffle Argument

Theorem 2. *Under the discrete log hardness assumption, the multi-key shuffle argument presented in this section has perfect completeness, perfect honest-verifier zero-knowledge, and computational witness extended emulation.*

Comparison with Bulletproofs. Bulletproofs' verifiable shuffle protocol takes as input two lists of commitments. It runs both lists through a sorting circuit and checks that the outputs are equal. The proof size is $O(\log(n \log(n)))$, and proof generation and verification run in $O(n \log(n))$ time.

Our shuffle proof protocol can be implemented using Bulletproofs, but it is complicated and less efficient. Since $\mathbf{h}$ and $\mathbf{h}'$ are not independent commitment parameters, we cannot use them as the two lists of commitments. Thus, the prover needs to commit to $\mathbf{h}$ and $\mathbf{h}'$ themselves; for example, embedding $\mathbf{h}$ and $\mathbf{h}'$ into elliptic curve points and setting $L_1 = \mathbf{g}^{\mathbf{h}} h^{\alpha}$ and $L_2 = \mathbf{g}'^{\mathbf{h}'} h^{\beta}$. Then, the prover would run our power argument to commit to $\mathbf{h}^s$. Finally, both lists will be run through the sorting circuit and checked for equality.

In our argument, we eliminate the complicated commitments to $\mathbf{h}$ and $\mathbf{h}'$ and the implementation of the sorting circuit. By doing so, we are able to lower our computation and communication costs: our proof size is $O(\log(n))$ and our proof generation and verification time is $O(n)$.

5 General Argument Protocol

In Section 4.3, we presented a zero-knowledge argument protocol for a multi-key verifiable shuffle with improved efficiency. Now, we will expand this protocol to present a general zero-knowledge argument protocol for arbitrary arithmetic circuits. The argument works directly over committed vectors for proving satisfiability of adaptively-defined randomized verification circuits, and remains secure even when the CRS is maliciously generated. We begin with a brief overview of our techniques.

Overview. Just as in the multi-key verifiable shuffle argument protocol, the general argument protocol in this section will also use a modified Bulletproofs sub-routine. Although our general protocol is an extension of Bulletproofs, we cannot use Bulletproofs in a black-box way because our techniques require modifications throughout the Bulletproofs protocol. Therefore, we present Bulletproofs as a starting point and describe our new functionality in hybrid steps.

We define four protocols, $\mathsf{P}_0, \mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$. Each protocol adds a new technique or functionality to the previous one, such that P_3 is the composition of all of our techniques. We describe each protocol below:

Protocol P_0. Our starting point is Bulletproofs. The Bulletproofs protocol enables the prover to argue that committed inputs that satisfy an arithmetic circuit C. The crux of the protocol is reducing satisfiability of the arithmetic circuit to satisfiability of a set of constraints, and then reducing satisfiability of the constraints to satisfiability of a single inner product relation.

We first describe the reduction from arithmetic circuit to constraints. Following [24], Bulletproofs represents an arithmetic circuit with n multiplication gates (each with fan-in 2) as a Hadamard product (entry-wise multiplication) and a set of linear constraints. For each multiplication gate, let $\mathbf{a}_L$ be the vector of left wire values, $\mathbf{a}_R$ be the vector of right wire values, and $\mathbf{a}_O$ be the vector of output wires. Then, satisfiability of the multiplication gates can be captured using the Hadamard product relation, $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$. Satisfiability of the rest of the circuit, including constraints on inputs $\{p_i\}_{i=1}^{l}$, can be captured using $Q \leq 2n$ linear constraints of the form $\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}$, for $\mathbf{w}_{L,q}, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}$, and $\mathbf{d} \in \mathbb{Z}_p$.

Finally, the inputs $\{p_i\}_{i=1}^{n}$ must be the committed values within the l Pedersen scalar inputs commitments from the prover, $\{P_i = \mathsf{Com}(p_i, \rho_i)\}_{i=1}^{l}$. Thus, satisfiability of the arithmetic circuit is reduced to satisfiability of the following constraints.

$$\{P_i\}_{i=1}^{l} = \{\mathsf{Com}(p_i, \rho_i)\}_{i=1}^{l} \quad \wedge \quad \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \quad \wedge$$
$$\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}$$

Now, we describe the reduction from the above constraints to a single inner product relation. First, $\mathcal{P}$ creates commitments A_R, A_L, A_O to the wire values $\mathbf{a}_R, \mathbf{a}_L, \mathbf{a}_O$, respectively. $\mathcal{P}$ also commits to $\mathbf{s}_L$ and $\mathbf{s}_R$, which will be used as blinding values for $\mathbf{a}_L$ and $\mathbf{a}_R$.

Then, $\mathcal{V}$ uses its random challenges y, z to reduce the Hadamard product relation and linear constraints into a *single* inner product relation. This is done by taking a random linear combination of all the equations, using coefficients $\mathbf{y}^n = (1, y, \ldots, y^{n-1})$

and $\mathbf{z}_{[1:]}^{Q+1} = (z, z^2, \ldots, z^Q)$. If the inner product relation holds, then the individual equations will all hold with overwhelming probability.

Using the inner product relation, $\mathcal{V}$ defines three polynomials $l(X), r(X) \in \mathbb{Z}_p^n[X]$, and $t(X) \in \mathbb{Z}_p[X]$ with respect to the left vector, right vector, and output vectors in the inner product relation. Now, $\mathcal{P}$ must argue that the polynomials are constructed correctly. This sub-argument uses a polynomial commitment scheme from [24], and applies an inner product optimization for logarithmic communication complexity.

$\mathcal{V}$ evaluates the polynomials in the exponent, using the public parameters, challenges x, y, z, and commitments to A_R, A_L, A_O, S_L, and S_R and checks that they match the evaluations of the polynomials submitted by the prover. Finally, $\mathcal{V}$ checks that $t(x) = \langle l(x), r(x) \rangle$. If all these checks hold, then the inner product relation holds.

Protocol P_1. Our first extension is to allow the protocol to prove statements not only on committed scalars, but also on committed vectors. The benefits of this are twofold. First, we are able to achieve concretely smaller proof size when the length of the input is greater than the log of the size of the circuit. Second, the prover will be able to directly integrate commitments from previous protocols that use the same CRS, whether these commitments are in scalar or vector form.

We modify the Bulletproofs protocol as follows. Let $\{\mathbf{v}_i\}_{i=1}^m$ be the vector-valued inputs to the circuit. The linear constraints that describe the circuit now include constraints $\{\mathbf{W}_{V\,i}\}_{i=1}^m$ on the input vectors. Thus, $\mathcal{P}$ must argue that the following constraints are satisfied.

$$\{V_i\}_{i=1}^m = \{\mathsf{Com}(\mathbf{v}_i, \gamma_i)\}_{i=1}^m \;\wedge\; \{P_i\}_{i=1}^l = \{\mathsf{Com}(p_i, \rho_i)\}_{i=1}^l \;\wedge\; \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \;\wedge$$
$$\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O + \sum_{i=1}^m \mathbf{W}_{V\,i} \cdot \mathbf{v}_i = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}$$

for $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q\times n}, \{\mathbf{W}_{V\,i}\}_{i=1}^m \in \mathbb{Z}_p^{Q\times m}, \mathbf{W}_P \in \mathbb{Z}_p Q \times l$, and $\mathbf{d} \in \mathbb{Z}_p$. When we reduce these constraints to a single inner product relation, the constraints $\mathbf{W}_{V\,i}$ appear in the left vector and the commitments V_i appear in the right vector. Then, when defining the corresponding polynomials $l(X)$ and $r(X)$, the constraints and commitments for the vector inputs must all be raised to different powers of X to ensure that the commitments are non-intersecting. The verifier needs to include these constraints and commitments as well in the final verification step.

The incorporation of vector commitments and corresponding constraints yields polynomials $l(X), r(X)$, and $t(X)$ of degree $O(m)$. We use the polynomial commitment scheme directly from [24], which is described below.

Definition 25 (PolyCommit [24]). *Let* pp *be the public commitment parameters of the commitment scheme, let* $t(X) = \prod_{i=-m_1}^{m_2} t_i X^i$ *be a polynomial of degree* $m_1 + m_2$*, with a constant* t_0 *term.*

- $\mathsf{PolyCommit}(\mathsf{pp}, m_1, m_2, 1, t(X)) \rightarrow (\mathsf{pc}, \mathsf{st})$*, where* $\mathsf{pc} = (\{T_i'\}_{i=1}^{m_1}, \{T_i'\}_{i=1}^{m_2})$ *and* $\mathsf{st} = (t(X), \tau)$
- $\mathsf{PolyEval}(\mathsf{st}, x) \rightarrow \mathsf{pe}$*, where* x *is a challenge from the verifier, and* $\mathsf{pe} = (\hat{t}, \tau_x)$
- $\mathsf{PolyVerify}(\mathsf{pp}, m_1, m_2, \mathsf{pc}, \mathsf{pe}, x) \rightarrow \mathsf{Accept}$ *or* Reject.

This scheme has communication complexity logarithmic in the degree of the polynomial $t(X)$. Here, the polynomial has degree $m + 2$, so the complexity of this step is $O(\sqrt{m})$. For witness size $O(m \cdot n)$, the overall proof size is $O(\sqrt{m} + \log(m \cdot n))$.

Protocol P_2. Our second extension is to allow statements with randomized verification. Bulletproofs only considers deterministic arithmetic circuits. However, many problems including polynomial identity testing, matrix multiplication, and primality testing, can be efficiently verified using randomized circuits.

Our main technique is to allow the verifier to adaptively define the circuit using random challenges. This is possible by encoding a universal circuit that accepts as input a set of functions, and that will evaluate the functions on the prover's inputs and the verifier's challenges.

First, as before, the prover provides commitments to the witness as inputs to the protocol. Then, the verifier samples a random challenge $\mathbf{c} \in \mathbb{Z}_p^n$. Using $\mathbf{c}$, the prover and verifier can compute the randomized linear constraints of the circuit, $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O$, $\{\mathbf{W}_{V\,i}\}_{i=1}^m$,, and $\mathbf{W}_P$ as follows.

$$\mathbf{W}_L = f_{\mathbf{W}_L}(\mathbf{c}) \in \mathbb{Z}_p^{Q \times n} \qquad \mathbf{W}_R = f_{\mathbf{W}_L}(\mathbf{c}) \in \mathbb{Z}_p^{Q \times n} \qquad \mathbf{W}_O = f_{\mathbf{W}_L}(\mathbf{c}) \in \mathbb{Z}_p^{Q \times n}$$
$$\mathbf{W}_{V\,i} = f_{\mathbf{W}_{V\,i}}(\mathbf{c}) \in \mathbb{Z}_p^{Q \times n} \ \ \forall i \in [1, m] \qquad \mathbf{W}_P = f_{\mathbf{W}_P}(\mathbf{c}) \in \mathbb{Z}_p^{Q \times l}$$

In addition, using $\mathbf{c}$ and the witness, the prover can compute the randomized wire values for the multiplication gates of the circuit, $\mathbf{a}_L, \mathbf{a}_R$, and $\mathbf{a}_O$.

$$\mathbf{a}_L = f_{\mathbf{a}_L}(\mathbf{c}, \mathbf{w}) \in \mathbb{Z}_p^n \quad \mathbf{a}_R = f_{\mathbf{a}_R}(\mathbf{c}, \mathbf{w}) \in \mathbb{Z}_p^n \quad \mathbf{a}_O = f_{\mathbf{a}_O}(\mathbf{c}, \mathbf{w}) \in \mathbb{Z}_p^n$$

Once the constraints and wire values are fixed, then the protocol proceeds as before.

Note that we do not consider how to construct probabilistic verification circuits for statements; this is outside the scope of the zero-knowledge argument. However, randomized verification is well-studied in the literature.

When enabling arguments for probabilistic statements, our argument size remains logarithmic in the size of the circuit, but the circuits themselves are often significantly smaller! Thus, we achieve concrete improvements in communication complexity for the class of problems with more efficient randomized verification.

Protocol P_3. Our third extension is allowing the CRS to come from an untrusted verifier or third-party, rather than from a trusted setup algorithm. The motivation for this comes from the shuffle application; there, the inputs and outputs of the shuffle, which are generated by the individual users of the bulletin board and the shuffler, are used as commitment parameters for the argument protocol. In such settings, when the CRS could be subverted, we must ensure that zero-knowledge still holds.

To do so, the prover must run the algorithm CRSCheck to verify that the CRS looks honestly generated. If the check passes, we use our claim from an earlier section that Pedersen commitments remain hiding, so the protocol has subversion zero-knowledge.

Note that the prover is not allowed to subvert the CRS, and we assume that the prover does not know a discrete log relation between the commitment parameters. Thus, soundness is not affected by the subverted CRS.

P_0, $\boxed{\mathsf{P}_1}$, $\dbox{\mathsf{P}_2}$, $\colorbox{lightgray}{$\mathsf{P}_3$}$

Statement: $\forall i \in [1, l], P_i = g_0^{p_i} h_0^{\rho_i} \ \wedge \ \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \ \wedge \ \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O + \boxed{\sum_{i=1}^{m} \mathbf{W}_{V\,i} \cdot \mathbf{v}_i} = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}$

CRS: $\colorbox{lightgray}{$\mathbf{g}, \mathbf{h} \leftarrow_{\$} \mathbb{G}^n, g_0, h_0, h \leftarrow_{\$} \mathbb{G}$}$

Public input: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n, g_0, h_0, h \in \mathbb{G}, \mathbf{P} \in \mathbb{G}^l$, $\boxed{\mathbf{V} \in \mathbb{G}^m}$, $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}$, $\boxed{\forall i \in [1, m]\ \mathbf{W}_{V\,i} \in \mathbb{Z}_p^{Q \times n}}$, $\mathbf{W}_P \in \mathbb{Z}_p^{Q \times l}, \mathbf{d} \in \mathbb{Z}_p^Q$, $\dbox{$f_{\mathbf{W}_L}, f_{\mathbf{W}_R}, f_{\mathbf{W}_O}, f_{\mathbf{W}_{V\,1}}, \ldots, f_{\mathbf{W}_{V\,m}} \in f : \mathbb{Z}_p^n \to \mathbb{Z}_p^{Q \times n}),$}$ $\dbox{$f_{\mathbf{W}_P} \in f : \mathbb{Z}_p^n \to \mathbb{Z}_p^{Q \times l}, f_{\mathbf{a}_L}, f_{\mathbf{a}_R}, f_{\mathbf{a}_O} \in f : \mathbb{Z}_p^n \times \mathbb{Z}_p^n \to \mathbb{Z}_p^n$}$

Private input: $\dbox{$\mathbf{w} \in \mathbb{Z}_p^n$}$, $\boxed{\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{Z}_p^n, \gamma \in \mathbb{Z}_p^m}$, $\mathbf{p} \in \mathbb{Z}_p^l, \rho \in \mathbb{Z}_p^l$

1. $\mathcal{P}$ **:** $\colorbox{lightgray}{If CRS does not pass CRSCheck, return $\perp$}$.

2. $\mathcal{P} \to \mathcal{V}$ **:** $\dbox{$\forall i \in [n], W_i = \mathsf{Com}(h_i, w_i)$}$

3. $\mathcal{V} \to \mathcal{P}$ **:** $\dbox{$\mathbf{c} \leftarrow_{\$} \mathbb{Z}_p^n$}$

$$\dbox{$\mathbf{W}_L = f_{\mathbf{W}_L}(\mathbf{c})$} \quad \dbox{$\mathbf{W}_R = f_{\mathbf{W}_L}(\mathbf{c})$} \quad \dbox{$\mathbf{W}_O = f_{\mathbf{W}_L}(\mathbf{c})$} \\ \dbox{$\mathbf{W}_{V\,i} = f_{\mathbf{W}_{V\,i}}(\mathbf{c}) \forall i \in [1, m]$} \ \dbox{$\mathbf{W}_P = f_{\mathbf{W}_P}(\mathbf{c}$} \tag{2}$$

4. $\mathcal{P}$ **:**

$$\dbox{$\mathbf{a}_L = f_{\mathbf{a}_L}(\mathbf{c}, \mathbf{w})$} \dbox{$\mathbf{a}_R = f_{\mathbf{a}_R}(\mathbf{c}, \mathbf{w})$} \dbox{$\mathbf{a}_O = f_{\mathbf{a}_O}(\mathbf{c}, \mathbf{w})$} \tag{3}$$

5. $\mathcal{P} \to \mathcal{V}$ **:** $\beta, \kappa, \phi, \lambda \leftarrow_{\$} \mathbb{Z}_p^*, \mathbf{s}_L, \mathbf{s}_R \leftarrow_{\$} \mathbb{Z}_p^n$

$$A_R = \mathbf{h}^{\mathbf{a}_R} h^{\alpha} \qquad A_L = \mathbf{g}^{\mathbf{a}_L} h^{\beta} \qquad A_O = \mathbf{g}^{\mathbf{a}_O} h^{\kappa} \tag{4}$$

$$S_R = \mathbf{h}^{\mathbf{s}_R} h^{\phi} \qquad S_L = \mathbf{g}^{\mathbf{s}_L} h^{\lambda} \tag{5}$$

6. $\mathcal{V} \to \mathcal{P}$ **:** $y, z \leftarrow_{\$} \mathbb{Z}_p^*$

7. $\mathcal{P}$ **and** $\mathcal{V}$ **:**

$$\mathbf{y}^n = (1, y, y^2, \ldots, y^{n-1}) \in \mathbb{Z}_p^n \tag{6}$$

$$\mathbf{z}_{[1:]}^{Q+1} = (z, z^2, \ldots, z^Q) \in \mathbb{Z}_p^Q \tag{7}$$

$$\delta(y, z) = \langle \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R), \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L \rangle \tag{8}$$

8. $\mathcal{P}$ **:**

$$l(X) = \mathbf{a}_L \cdot X^{-1} + \mathbf{a}_O \cdot X^{-2} + \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R) \cdot X^{-1} + \mathbf{s}_L \cdot X^{-3} + \\ \boxed{\sum_{i=1}^{m} \mathbf{W}_{V\,i} \cdot X^{-(i+3)}} \in \mathbb{Z}_p^n[X] \tag{9}$$

$$\tag{10}$$

$$r(X) = \mathbf{y}^n \circ \mathbf{a}_R \cdot X - \mathbf{y}^n \cdot X^2 + \mathbf{z}_{[1:]}^{Q+1}(\mathbf{W}_L \cdot X + \mathbf{W}_R \cdot X^2) + \mathbf{y}^n \circ \mathbf{s}_R \cdot X^3 + \boxed{\sum_{i=1}^m V_i \cdot X^{i+3}} \in \mathbb{Z}_p^n[X] \tag{11}$$

$$t(X) = \langle l(X), r(X) \rangle = \prod_{i=-(m+2)}^{m+2} t_i X^i \in \mathbb{Z}_p[X] \tag{12}$$

$$\mathbf{u} = \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O + \boxed{\sum_{i=1}^m \mathbf{W}_{V\,i} \cdot \mathbf{v}_i} \tag{13}$$

$$t_0 = \langle \mathbf{a}_L, \mathbf{a}_R \cdot \mathbf{y}^n \rangle - \langle \mathbf{a}_O, \mathbf{y}^n \rangle + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{u} \rangle + \delta(y,z) = \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d} \rangle + \delta(y,z) \tag{14}$$

9. $\mathcal{P} \rightarrow \mathcal{V}$ **:**

$$(\{T_i\}_{i=-(m+2)}^{-1}, \{T_i\}_{i=1}^{m+2}; t(X), \tau) \leftarrow \mathsf{PolyCommit}((g_0, h_0), m+2, m+2, 1, t(X)) \tag{15}$$

10. $\mathcal{V} \rightarrow \mathcal{P}$ **:** $x \leftarrow_{\$} \mathbb{Z}_p^*$
11. $\mathcal{P} \rightarrow \mathcal{V}$ **:**

$$(\hat{t}, \mathbf{l}, \mathbf{r}, \tau_x, \mu) \leftarrow \mathsf{PolyEval}((t(X), \tau), x) \tag{16}$$

12. $\mathcal{V}$ **:**

$$\mathsf{PolyVerify}((g_0, h_0), m+2, m+2, 1, T_i, \hat{t}, \tau_x, x) \tag{17}$$

$$h_i^* = h_i^{y^{-i+1}} \quad \forall i \in [1, n] \tag{18}$$

$$W_L = \mathbf{g}^{\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L} \tag{19}$$

$$W_R = \mathbf{h}^{\mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R)} \tag{20}$$

$$W_O = \mathbf{g}^{\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_O} \tag{21}$$

$$\boxed{W_{V,i} = \mathbf{g}^{\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_{V\,i}} \quad \forall i \in [1, m]} \tag{22}$$

$$h^\mu \mathbf{g}^{\mathbf{l}} \mathbf{h}^{*\mathbf{r}} \stackrel{?}{=} A_L^{x^{-1}} \cdot A_R^x \cdot A_O^{x^{-2}} \cdot \boxed{\prod_{i=1}^m V_i^{x^{i+3}}} \cdot \mathbf{h}^{*-\mathbf{y}^n} \cdot W_L^x \cdot W_R^{x^{-1}} \cdot W_O^{x^2} \cdot \boxed{\prod_{i=1}^m {W_V}_i^{-(i+3)}} \cdot S_L^{x^{-3}} \cdot S_R^{x^3} \tag{23}$$

13. $\mathcal{V}$ **:** Accept or Reject

P_3: Composition of protocols

Theorem 3. *Assuming hardness of discrete log, the general argument protocol* P_3 *presented above has perfect completeness, subversion zero-knowledge, and computational witness-extended emulation.*

6 Acknowledgements

We would like to thank Oleg Andreev, Preetum Nakkiran, Salil Vadhan, and Cathie Yun for helpful discussions, and the anonymous reviewers for their valuable comments.

References

1. Chaum DL. Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM. 1981;24(2):84–90.
2. Dingledine R, Mathewson N, Syverson P. Tor: The second-generation onion router. Naval Research Lab Washington DC; 2004.
3. Ren J, Wu J. Survey on anonymous communications in computer networks. Computer Communications. 2010;33(4):420–431.
4. Juang WS, Lei CL, Chang CY. Anonymous channel and authentication in wireless communications. Computer communications. 1999;22(15-16):1502–1511.
5. Jacobson M, M'Raïhi D. Mix-based electronic payments. In: International Workshop on Selected Areas in Cryptography. Springer; 1998. p. 157–173.
6. Choi S, Kim K. Authentication and payment protocol preserving location privacy in mobile IP. In: GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489). vol. 3. IEEE; 2003. p. 1410–1414.
7. Androulaki E, Raykova M, Srivatsan S, Stavrou A, Bellovin SM. PAR: Payment for anonymous routing. In: International Symposium on Privacy Enhancing Technologies Symposium. Springer; 2008. p. 219–236.
8. Androulaki E, Choi SG, Bellovin SM, Malkin T. Reputation systems for anonymous networks. In: International Symposium on Privacy Enhancing Technologies Symposium. Springer; 2008. p. 202–218.
9. Zhai E, Wolinsky DI, Chen R, Syta E, Teng C, Ford B. AnonRep: Towards Tracking-Resistant Anonymous Communication. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI). CA, USA: Santa Clara; 2016. .
10. Neff ACA. verifiable secret shuffle and its application to e-voting. In: 8th ACM Conference on Computer and Communications Security (CCS; 2001. .
11. Jakobsson M, Juels A, Rivest RL. Making mix nets robust for electronic voting by randomized partial checking. In: USENIX security symposium. San Francisco, USA; 2002. p. 339–353.
12. Furukawa J, Sako K. An efficient scheme for proving a shuffle. In: Annual International Cryptology Conference. Springer; 2001. p. 368–387.
13. Groth J, Ishai Y. Sub-linear zero-knowledge argument for correctness of a shuffle. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer; 2008. p. 379–396.
14. Wikström D. A commitment-consistent proof of a shuffle. In: Australasian Conference on Information Security and Privacy. Springer; 2009. p. 407–421.
15. Groth J. A verifiable secret shuffle of homomorphic encryptions. Journal of Cryptology. 2010;23(4):546–579.
16. Bayer S, Groth J. Efficient zero-knowledge argument for correctness of a shuffle. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer; 2012. p. 263–280.
17. Groth J. Non-interactive zero-knowledge arguments for voting. In: Heidelberg SB, editor. International Conference on Applied Cryptography and Network Security; 2005. .
18. DeMillo RA, Lipton RJ. A Probabilistic Remark on Algebraic Program Testing. Georgia Inst of Tech Atlanta School of Information and Computer Science; 1977.
19. Schwartz JT. Probabilistic algorithms for verification of polynomial identities. In: International Symposium on Symbolic and Algebraic Manipulation. Springer; 1979. p. 200–215.
20. Zippel R. Effective polynomial computation. vol. 241. Springer Science & Business Media; 2012.

21. Miller GL. Riemann's hypothesis and tests for primality. Journal of computer and system sciences. 1976;13(3):300–317.
22. Rabin MO. Probabilistic algorithm for testing primality. Journal of number theory. 1980;12(1):128–138.
23. Agrawal M, Kayal N, Saxena N. PRIMES is in P. Annals of mathematics. 2004;p. 781–793.
24. Bootle J, Cerulli A, Chaidos P, Groth J, Petit C. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer; 2016. p. 327–357.
25. Bünz B, Bootle J, Boneh D, Poelstra A, Wuille P, Maxwell G. Bulletproofs: Efficient range proofs for confidential transactions. Cryptology ePrint Archive, Report 2017/1066, 2017. https://eprint. iacr. org/2017/1066; 2017.
26. Maxwell G. Confidential transactions. URL: https://people xiph org/˜ greg/confidential_values txt (Accessed 09/05/2016). 2015;.
27. Bünz B, Agrawal S, Zamani M, Boneh D. Zether: Towards Privacy in a Smart Contract World. IACR Cryptology ePrint Archive. 2019;2019:191.
28. Bellare M, Fuchsbauer G, Scafuro A. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer; 2016. p. 777–804.
29. Hoffmann M, Klooß M, Rupp A. Efficient zero-knowledge arguments in the discrete log setting, revisited (Full version). 2019;.
30. Bellare M, Rogaway P. Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM conference on Computer and communications security. ACM; 1993. p. 62–73.
31. Groth J. Linear algebra with sub-linear zero-knowledge arguments. In: Advances in Cryptology-CRYPTO 2009. Springer; 2009. p. 192–208.
32. Freivalds R. Fast probabilistic algorithms. In: International Symposium on Mathematical Foundations of Computer Science. Springer; 1979. p. 57–69.
33. Goldwasser S, Micali S, Rackoff C. The knowledge complexity of interactive proof systems. SIAM Journal on computing. 1989;18(1):186–208.
34. Goldreich O, Micali S, Wigderson A. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM (JACM). 1991;38(3):690–728.
35. Kilian J. A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. ACM; 1992. p. 723–732.
36. Schnorr CP. Efficient signature generation by smart cards. Journal of cryptology. 1991;4(3):161–174.
37. Cramer R, Damgård I. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In: Annual International Cryptology Conference. Springer; 1998. p. 424–441.
38. Seo JH. Round-efficient sub-linear zero-knowledge arguments for linear algebra. In: International Workshop on Public Key Cryptography. Springer; 2011. p. 387–402.
39. Ben-Sasson E, Chiesa A, Genkin D, Tromer E, Virza M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Annual Cryptology Conference. Springer; 2013. p. 90–108.
40. Ben-Sasson E, Bentov I, Horesh Y, Riabzev M. Scalable, transparent, and post-quantum secure computational integrity. IACR Cryptology ePrint Archive. 2018;2018:46.
41. Ames S, Hazay C, Ishai Y, Venkitasubramaniam M. Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM; 2017. p. 2087–2104.

42. Wahby RS, Tzialla I, Shelat A, Thaler J, Walfish M. Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). IEEE; 2018. p. 926–943.
43. Giacomelli I, Madsen J, Orlandi C. Zkboo: Faster zero-knowledge for boolean circuits. In: 25th {USENIX} Security Symposium ({USENIX} Security 16); 2016. p. 1069–1083.
44. Setty S. Spartan: Efficient and general-purpose zkSNARKs without trusted setup;.
45. Yun dVHAO Cathie. Programmable Constraint Systems for Bulletproofs;.
46. Sako K, Kilian J. Receipt-free mix-type voting scheme. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer; 1995. p. 393–403.
47. Abe M. Universally verifiable mix-net with verification work independent of the number of mix-servers. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer; 1998. p. 437–447.
48. Abe M. Mix-networks on permutation networks. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer; 1999. p. 258–273.
49. Abe M, Hoshino F. Remarks on mix-network based on permutation networks. In: International Workshop on Public Key Cryptography. Springer; 2001. p. 317–324.
50. Lindell Y. Parallel coin-tossing and constant-round secure two-party computation. Journal of Cryptology. 2003;16(3).
51. Nguyen L, Safavi-Naini R, Kurosawa K. Verifiable shuffles: A formal model and a paillier-based efficient construction with provable security. In: ACNS; 2004. .

A Proof of Lemma 1

Let $g, h \in G$ be part of a CRS that has passed the CRSCheck algorithm, and let $c = g^m h^r$, $m, r \in \mathbb{Z}_p^*$, be a Pedersen commitment for the message m. Since $h, g \in \mathbb{G}$, there must exist some $a \in \mathbb{Z}_p^*$ such that $h = g^a$. Now, we note that for every message m', there exists a unique r' such that $g^{m'} h^{r'} = c$. We can compute r' as

$$r' = \alpha^{-1}(m - m' + \alpha r) \in \mathbb{Z}_p^*$$

Thus, c does not reveal any information about m, so the commitment is perfectly hiding. A similar argument can be made for Pedersen vector commitments.

B Proof of Theorem 1

We prove permutation indistinguishability by reducing the IND-CPA_S security of our shuffle to the Decisional Diffie-Hellman (DDH) Hardness Assumption.

Definition 26 (Decisional Diffie-Hellman (DDH) Hardness Assumption). *Let $\mathbb{G}$ be a large cyclic group of prime order p. We consider the following two distributions:*

- (g, g^r, g^s, g^{rs})*, where $r, s \leftarrow_\$ \mathbb{Z}_p$ are chosen uniformly at random.*
- (g, g^r, g^s, g^t)*, where $r, s, t \leftarrow_\$ \mathbb{Z}_p$ are chosen uniformly at random.*

The DDH hardness assumption states that for all probabilistic polynomial-time algorithms $\mathcal{B}$, the probability that $\mathcal{B}$ efficiently distinguishes between these two distributions is negligible.

We show that if there exists an adversary $\mathcal{A}$ which has non-negligible advantage in the IND-CPA$_S$ security game, we can construct an adversary $\mathcal{B}$ that breaks the hardness of DDH. We denote the DDH challenger by $\mathcal{C}^{DDH}$.

```
DDH Adversary $\mathcal{B}$
1 :  $T \subset [n], |T| \leq n-2 \leftarrow\$ \mathcal{A}(n)$
2 :  $(\mathsf{pp} := (\mathbb{G}, p, g)) \leftarrow\$ \mathcal{C}^{DDH}$
3 :  $(m_1, \ldots, m_n; \{\mathsf{pk}_i, \mathsf{ct}_i\}_{i \in T}; \pi_0, \pi_1 : \pi_0(i) = \pi_1(i) \forall i \in T) \leftarrow \mathcal{A}^{\mathsf{KGen}, \mathsf{Enc}}(\mathsf{pp})$
4 :  $(g^r, g^s, g^t) \leftarrow\$ \mathcal{C}^{DDH}$
5 :  $\mathsf{pk}_1 := g^r; \mathsf{ct}_1 \leftarrow\$ \mathsf{C}_{\mathsf{pp}}$
6 :  $\mathsf{pk}_j \leftarrow\$ \mathsf{KGen}(\mathsf{pp}), \mathsf{ct}_j \leftarrow\$ \mathsf{C}_{\mathsf{pp}} \quad \forall j \notin T, j \neq 1$
7 :  $L := \{(\mathsf{pk}_i, \mathsf{ct}_i)\}_{i \in [n]}$
8 :  $\pi := \pi_1; \mathsf{pp}' := (\mathbb{G}, p, g^s)$
9 :  $\mathsf{pk}'_{\pi(1)} := g^t; \mathsf{ct}'_{\pi(1)} \leftarrow\$ \mathsf{C}_{\mathsf{pp}'}$
10 : $\mathsf{pk}'_{\pi(i)} \leftarrow\$ \mathsf{KGen}(\mathsf{pp}'), \mathsf{ct}'_{\pi(i)} \leftarrow\$ \mathsf{C}_{\mathsf{pp}'} \quad \forall i \neq 1$
11 : $L' := \{(\mathsf{pk}_{\pi(i)}, \mathsf{ct}_{\pi(i)})\}_{i \in [n]}$
12 : $b'' \leftarrow\$ \mathcal{A}(\mathsf{pp}, \mathsf{pp}', L, L')$
13 : $b' := b''$
14 : $\mathbf{return}\ b'$
```

Given the public parameters provided by $\mathcal{C}$, $\mathcal{A}$ specifies messages $m_1, \ldots, m_n$, a subset T of $[n]$, such that $|T| \leq n-2$, public keys pk_i for $i \in T$, and two distinct permutations π_0 and π_1 that contain the same mapping for all indices in T. Since the permutations are distinct, and the set T leaves out two indices in $[n]$, there exists at least one index in $[n]$ but not in T that is mapped differently by π_0 and π_1. Without loss of generality, let this index be $i = 1$.

Now, $\mathcal{C}$ samples a random bit $b \leftarrow\$ \{0,1\}$. If $b = 1$, it creates a valid DDH tuple; otherwise, it samples three random group elements from $\mathbb{G}$. $\mathcal{B}$ receives the DDH challenge (g^r, g^s, g^t) from $\mathcal{C}$.

$\mathcal{B}$ uses the DDH challenge as follows. It sets $\mathsf{pk}_1 := g^r$, $\mathsf{pk}_{\pi_1(1)} := g^t$, and $\mathsf{pp}' := (\mathbb{G}, p, g^s)$. Note that $\mathcal{B}$ could have used the DDH challenge within the ciphertexts, rather than the public keys [3], since the El Gamal ciphertext components are also group elements.

Then, $\mathcal{B}$ fills in the rest of the unspecified elements in L and L' with random group elements sampled from the appropriate domains. $\mathcal{B}$ sends $(\mathsf{pp}, \mathsf{pp}', L, L')$ to $\mathcal{A}$ and receives a guess b'' for which permutation was used.

If (g^r, g^s, g^t) was a valid DDH tuple, the placement of g^t in the output list L' would adhere to the mapping given by π_1. Thus, an $\mathcal{A}$ who can distinguish between permutations would have a non-negligible advantage in guessing $b'' = 1$. If the DDH challenge was not a valid DDH tuple, however, then the entire list L' contains elements

[3] For instance, let ct_1 denote either the first or second component of the ciphertext. Then, $\mathcal{B}$ could set $\mathsf{ct}_1 := g^r, \mathsf{ct}_{\pi_1(1)} := g^t$, and $\mathsf{pp}' := (\mathbb{G}, p, g^s)$.

that are selected randomly and that do not correspond to L, so $\mathcal{A}$ has no advantage in choosing the correct permutation.

Thus, if $\mathcal{A}$ returns $b'' = 1$, $\mathcal{B}$ returns $b' = 1$, and has the same advantage of $\mathcal{A}$. Otherwise, $\mathcal{B}$ returns $b' = 0$ and has no advantage. $\mathcal{B}$ will therefore win the IND-CPA_S game with half the advantage of $\mathcal{A}$. Assuming $\mathcal{A}$ has a non-negligible advantage in guessing the permutation, $\mathcal{B}$ has a non-negligible advantage in choosing the correct message. However, under the DDH hardness assumption, we should not be able to construct an adversary $\mathcal{B}$ that can win the game with non-negligible probability. Thus, there must not exist an adversary $\mathcal{A}$ that has non-negligible advantage in the $\text{IND-CPA}_S^{\mathcal{A}}$ security game.

C Proof of Theorem 2

C.1 Perfect Completeness

Perfect completeness follows from inspection. It can be verified that the relation holds for all valid witnesses.

C.2 Perfect Special Honest-Verifier Zero-Knowledge

Perfect SHVZK follows from the Bulletproofs' efficient simulator. In addition, our simulator can simulate a commitment to $\mathbf{a}_L$ by choosing a random group element in $\mathbb{G}$. A_L is a perfectly hiding commitment, so the simulator's choice of random independent elements produces an indistinguishable distribution from a real execution.

C.3 Computational Witness-Extended Emulation

To show computational witness extended emulation, we run the prover and verifier with random challenges. If we get an acceptable argument we have to extract a witness. We construct an extractor χ as follows. χ calls the extractor $\chi_{\text{Bulletproofs}}$, which runs the prover with n different values of r and u, n different values of y, 4 different values of z, and 6 different values of x, where x, y, and z are the verifier's random challenges within the Bulletproofs subroutine. $\chi_{\text{Bulletproofs}}$ produces $O(n^4)$ valid proof transcripts, using which it extracts witnesses $\mathbf{a}_R$ and $\mathbf{a}_L$.

$\chi_{\text{Bulletproofs}}$ takes linear combinations of the final verification equations using $A_{R,h}$, $A_{R,b}$, and $A_{R,c}$ to extract α, $\mathbf{a}_L$ and $\mathbf{a}_R$ such that

$$\begin{aligned} A_{R,h} &= \mathbf{h}^{\mathbf{a}_R} \\ A_{R,b} &= \mathbf{b}^{\mathbf{a}_R} \\ A_{R,c} &= \mathbf{c}^{\mathbf{a}_R} \\ A_L &= \mathbf{g}^{\mathbf{a}_L} h^{\alpha} \end{aligned}$$

and such that the following linear constraints hold.

$$\begin{aligned} a_{L0} &= 1 \\ a_{Li} &= a_{Li-1} \cdot a_{Ri-1} \\ a_{Ln-1} \cdot a_{Rn-1} &= s^n \prod_i k_i \\ \sum_{i=0}^{n-1} a_{Ri} &= s \sum_i k_i \end{aligned}$$

If $\chi_{\text{Bulletproofs}}$ can compute different $\alpha', \mathbf{a}'_R, \mathbf{a}'_L$ for any other set of challenges such that the above equations hold, then this yields a non-trivial discrete log relation between independent generators $\mathbf{h}$, $\mathbf{g}$, h, which contradicts the discrete log relation assumption.

Now, we argue that the extracted $\mathbf{a}_L$ and $\mathbf{a}_R$ are properly formed. This will allow χ to extract witnesses s and π. Before we begin, we will state the following lemma that will be used in the proof.

Lemma 2 (Schwartz-Zippel). *Let p be a non-zero multi-variate polynomial of degree d over $\mathbb{Z}_q$. Then, the probability of of $p(x_1, \ldots, x_n) = 0$ for randomly chosen $x_1, \ldots, x_n \leftarrow_{\$} \mathbb{Z}_q^*$ is at most $\frac{d}{q-1}$.*

To check whether two multivariate polynomials p_1, p_2 are equal, we can use a random $x_1, \ldots, x_n \leftarrow_{\$} \mathbb{Z}_q^*$ and test whether $p_1(x_1, \ldots, x_n) - p_2(x_1, \ldots, x_n) = 0$. If $p_1 = p_2$, this test will always pass. If $p_1 \neq p_2$, the test will pass with probability at most $\frac{max(d_1,d_2)}{q-1}$.

We give a high-level overview of our argument before proceeding in detail.

Overview. We show that if the extracted $\mathbf{a}_L$ and $\mathbf{a}_R$ satisfy the constraints, then $\mathbf{a}_R$ is properly formed. The first two constraints on $\mathbf{a}_L$ are trivial, so we focus on the latter two constraints. The first of these shows that there exists a permutation π mapping $\mathbf{h}$ to $\mathbf{h}'$, and the second shows that there exists an s such that all the discrete log between input-output pairs are equal to s.

To show that $\mathbf{a}_R$ is properly formed, we first express each h'_j as a commitment to a discrete log relation between bases $\mathbf{h} = h_0, \ldots, h_{n-1}$. Since the discrete log between bases $\mathbf{h}$ are not known to $\mathcal{P}$, the commitment is binding to a unique relation. This will allow us to re-write A_R in terms of bases $\mathbf{h}$ in two ways: one as a vector commitment to $\mathbf{a}_R$, and the other as a compilation of these discrete log relations.

Now, since we have written A_R in two distinct forms, we match the exponents, which will give us an equation for a_{Ri}. From our first constraint on $\mathbf{a}_R$, we find there exists a permutation π between the elements in $\mathbf{h}$ and $\mathbf{h}'$. From our second constraint, we find there exists an exponent s to which all input elements are raised. Finally, χ extracts π and s from $\mathbf{a}_R$.

Existence of permutation π. First, we show that every element of $\mathbf{a}_R$ contains a distinct k_j and that there exists a permutation mapping elements in $\mathbf{h}$ (indexed by i) to elements in $\mathbf{h}'$ (indexed by j).

Recall that a_{Ri} was constructed by $\mathcal{V}$ using bases $\mathbf{h}'$ as follows:

$$A_R = \prod_{j=1}^{n} {h'_j}^{k_j} \tag{24}$$

Next, we know that for all $j \in [n]$, h'_j can be expressed using the bases $\mathbf{h}$. This is because in a prime order group, all elements are generators.

$$h'_j = \prod_{i=1}^{n} h_i^{x_{i,j}} \tag{25}$$

where $x_{i,j} \in \mathbb{Z}_p$. Then, combining equations (24) and (25), we can re-write A_R using these $x_{i,j}$'s.

$$A_R = \prod_{j=1}^{n} \prod_{i=1}^{n} h_i^{x_{i,j} \cdot k_j} = \prod_{i=1}^{n} h_i^{\sum_{j=1}^{n}(x_{i,j} \cdot k_j)} \tag{26}$$

At the same time, A_R can also be expressed using the bases $\mathbf{h}$.

$$A_R = \prod_{i=1}^{n} {h_i}^{a_{Ri}} \tag{27}$$

If this equation does not hold, then we have a non-trivial discrete logarithm relation between independent bases h_i, which contradicts the discrete logarithm assumption. Thus, we can compare equations (26) and (27) to relate the a_{Ri}'s to the $x_{i,j}$'s.

$$a_{Ri} = \sum_{j=1}^{n} (x_{i,j} \cdot k_j) \tag{28}$$

Note that since the k_j terms were applied by $\mathcal{V}$, the prover could not have altered them or chosen other combinations of values that equal k_j. The prover would only have had control over altering the $x_{i,j}$ terms that relate h_i to h'_j.

Now, we turn to our first constraint on $\mathbf{a}_R$, which is the following.

$$\prod_{i=1}^{n} a_{Ri} = s^n \prod_{i=1}^{n} k_i$$

Using (28), we can replace the a_{Ri}'s with $\sum_{j=1}^{n}(x_{i,j} \cdot k_j)$ in this first constraint to get the following equation.

$$\prod_{i=1}^{n} \sum_{j=1}^{n} (x_{i,j} \cdot k_j) = s^n \prod_{i=1}^{n} k_i \tag{29}$$

Again, we note that the k_j's on the left hand side are applied to the prover's commitment to $x_{i,j}$'s, so the k_j's cannot have been altered by the prover.

We show that this equation enforces a permutation π from j to i. First, we claim that for every $j \in [n]$, there exists exactly one $i \in [n]$ such that $x_{i,j}$ is nonzero. By way of contradiction, assume that for some j, $x_{i,j}$ is zero for every i. Then, the left side of the equation will not contain a k_j term, yet there will be a k_j term on the right. Thus, there must be at least one i for every j such that $x_{i,j}$ is nonzero. Similarly, assume that for some j there exists $i, i' \in [n]$, where $i \neq i'$, such that $x_{i,j}$ and $x_{i',j}$ are both non-zero. Then, k_j will appear in two separate sums on the left side. When these sums are multiplied, this will result in a k_j^2 term on the left side; however, there is no such term on the right side. Thus, for every j, there must be exactly one i such that $x_{i,j}$ is non-zero. Next, we show that for every $j \in [n]$, there exists a distinct $i \in [n]$ such that $x_{i,j}$ is nonzero. Assume that there exists an i and $j \neq j'$ such that $x_{i,j}$ and $x_{i,j'}$ are both nonzero. Since we know that exactly one $x_{i,j}$ is nonzero for a given j, k_j and $k_{j'}$ will appear only within the same sum on the left side. Then, when expanded, the left side will not contain a $k_j k_{j'}$ term, yet the right side will. Thus, if $x_{i,j}$ and $x_{i,j'}$ are both nonzero, then $j = j'$. This shows that for every $j \in \mathbb{Z}_n$, there exists a distinct $i \in \mathbb{Z}_n$ such that $x_{i,j}$ is nonzero. If we define π as the function that maps from j to i for all nonzero $x_{i,j}$'s,

$$\pi(j) = i \quad i, j \in [n]$$

then, it is clear that π is a permutation. Now, we can re-write equation (29) as

$$\prod_{i=1}^{n} (x_{i,\pi^{-1}(i)} \cdot k_{\pi^{-1}(i)}) = s^n \prod_{i=1}^{n} k_i$$

Thus, we have shown that there exists a permutation π mapping elements of $\mathbf{h}'$ to elements of $\mathbf{h}$.

Existence of universal discrete log value s. Next, we turn to our second constraint on a_R, which is as follows.

$$\sum_{i=1}^{n} a_{Ri} = s \sum_{i=1}^{n} k_i$$

From the result of the first constraint, we can re-write the equation in terms of π.

$$\sum_{i=1}^{n} x_{i,\pi^{-1}(i)} \cdot k_{\pi^{-1}(i)} = s \sum_{i=1}^{n} k_i$$

Since the $k_{\pi^{-1}(i)}$'s were applied by the verifier (so they are truly equal to the k_i's), and π is a permutation, the k terms must be the same on left and right hand sides of the equation. Then, we see that the right side of this equation enforces that we must be able to factor out s. This means that, for all $i \in \mathbb{Z}_n$, the following must hold.

$$x_{i,\pi^{-1}(i)} = s$$

This shows that all the coefficients are equal to the committed value s, and thus, are all equal to each other.

Extracting witnesses s and π. Now that we know $\mathbf{a}_R$ is formed correctly, χ can easily extract witnesses s and π. First, χ can divide out $\prod_i k_i$ from $\prod_i a_{Ri}$ to obtain s^n. Then, it can take the nth root of s^n to obtain s.

$$s = \left(\frac{\prod_i a_{Ri}}{\prod_i k_i} \right)^{1/n}$$

Now that χ has extracted s, it can divide out s from each a_{Ri} to obtain $k_{\pi^{-1}(i)}$.

$$k_{\pi^{-1}(i)} = \frac{a_{Ri}}{s}$$

Finally, it can compare the list of $k_{\pi^{-1}(i)}$'s against the list of k_i's to extract the permutation π.

Extraction either returns a valid witness (s, π) or a discrete logarithm relation between independently chosen generators $g, h, \mathbf{h}$. By the Discrete Log Relation assumption, the latter scenario occurs with negligible probability. χ rewinds the prover $O(n^4)$ times, so extraction is efficient. We have shown that extraction is efficient and returns a valid witness with overwhelming probability, which proves that computational witness extended emulation holds.

D Proof of Theorem 3

D.1 Perfect Completeness

Perfect completeness follows from inspection. It can be verified that the relation holds for all valid witnesses.

D.2 Subversion Zero-Knowledge

We proceed via hybrid argument, by constructing simulators for each of the protocols above, P_0 through P_3. We claim that each protocol fulfills its respective zero-knowledge property, conditioned on the previous protocol doing the same. Note that P_0 corresponds to Bulletproofs, while P_3 is the composition of all our techniques. All the simulators discussed below take as input the proof protocol's public input pub and the verifier's randomness ρ.

Simulator S_0. This is the simulator for the Bulletproofs protocol.

- Inputs: pub $= (\mathbf{g}, \mathbf{h}, g_0, h_0, h, \mathbf{P}, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \mathbf{W}_P, \mathbf{d}); \rho$
- Compute x, y, z from the verifier's randomness ρ.
- $\mu, \tau_x \leftarrow_\$ \mathbb{Z}_p$
- $\mathbf{l}, \mathbf{r} \leftarrow_\$ \mathbb{Z}_p^n$
- $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle$
- $A_R, A_L, A_O, S_R \leftarrow_\$ \mathbb{G}$
- $S_L = (h^\mu \cdot \mathbf{g}^{\mathbf{l}} \cdot \mathbf{h}^{*\mathbf{r}} \cdot A_L^{x^{-1}} \cdot A_R^{x} \cdot A_O^{x^{-2}} \cdot \mathbf{h}^{*-\mathbf{y}^n} \cdot W_L^{x} \cdot W_R^{x^{-1}} \cdot W_O^{x^2} \cdot S_R^{x^3})^{x^3}$

- $T_i \leftarrow_{\$} \mathbb{G} \quad i = \{-2, 3\}, i \neq 0$
- $T_0 = g_0^{x^0(\delta(y,z)+\langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{d}\rangle)} \cdot P^{x^0(\mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_P \cdot \rho)} \cdot \prod_{i=-2, i\neq 0}^{3} T_i^{x^i}$
- Output: $(A_R, A_L, A_O, S_R, S_L; y, z; (T_i)_{-2}^{3}; x; \mu, \hat{t}, \mathbf{l}, \mathbf{r})$

Simulator S_1. This simulator invokes S_0. It adjusts some of the values returned by S_0, and uniformly samples the values not included in S_0's transcript.

- Inputs: $\text{pub} = (\mathbf{g}, \mathbf{h}, g_0, h_0, h, \mathbf{P}, \mathbf{V}, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \forall i \in [1, m]\, \mathbf{W}_{V\,i}, \mathbf{W}_P, \mathbf{d}); \rho$
- $\text{pub}' = (\mathbf{g}, \mathbf{h}, g_0, h_0, h, \mathbf{P}, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \mathbf{W}_P, \mathbf{d})$
- $(A'_R, A'_L, A'_O, S'_R, S'_L; y', z'; (T'_i)_{-2}^{3}; x'; \mu', \hat{t}', \mathbf{l}', \mathbf{r}') \leftarrow \mathsf{S}_0(\text{pub}', \rho)$
- $S_L = S'_L \cdot (\sum_{i=1}^{m} V_i^{x^{i+3}} \cdot \sum_{i=1}^{m} W_{V_i}^{-(i+3)})^{x^3}$
- $T_i \leftarrow_{\$} \mathbb{G}, \quad i \in \{-m+2, m+2\}/\{-2, 3\}$
- $T_0 = T'_0 \cdot \prod_{i=-(m+2), i\notin\{-2,3\}}^{m+2} T_i^{x^i} \cdot \prod_{i=-2, i\neq 0}^{3} T_i'^{x^i}$
- Output: $(A'_R, A'_L, A'_O, S'_R, S_L; y', z'; (T'_i)_{i\in\{-2,3\}/\{0\}}, (T_i)_{i\in\{-(m+2), m+2\}/\{-2,3\}},$ $T_0; x'; \mu', \hat{t}', \mathbf{l}', \mathbf{r}')$

Simulator S_2. This simulator first computes the weights $\mathbf{W}'_L, \mathbf{W}'_R, \mathbf{W}'_O$ using the verifier's challenge $\mathbf{c}$. Then, it invokes S_1 with these weights as part of the public input, and returns S_1's output directly.

- Inputs: $\text{pub} = (\mathbf{g}, \mathbf{h}, g_0, h_0, h, \mathbf{P}, \mathbf{V}, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \forall i \in [1, m]\, \mathbf{W}_{V\,i}, \mathbf{W}_P, \mathbf{d},$ $f_{W,R}, f_{W,L}, f_{W,O}, f_{a,R}, f_{a,L}, f_{a,O}); \rho$
- Compute $\mathbf{c}$ from the verifier's randomness.
- $\mathbf{W}'_R = f_{W,R}(\mathbf{W}_R, \mathbf{c}), \mathbf{W}'_L = f_{W,L}(\mathbf{W}_L, \mathbf{c}), \mathbf{W}'_O = f_{W,O}(\mathbf{W}_O, \mathbf{c})$
- $\text{pub}' = (\mathbf{g}, \mathbf{h}, g_0, h_0, h, \mathbf{P}, \mathbf{V}, \mathbf{W}'_L, \mathbf{W}'_R, \mathbf{W}'_O, \forall i \in [1, m]\, \mathbf{W}_{V\,i}, \mathbf{W}_P, \mathbf{d},$ $f_{W,R}, f_{W,L}, f_{W,O}, f_{a,R}, f_{a,L}, f_{a,O}); \rho$
- $(A'_R, A'_L, A'_O, S'_R, S'_L; y', z'; (T'_i)_{-m+2}^{m+2}; x'; \mu', \hat{t}', \mathbf{l}', \mathbf{r}') \leftarrow \mathsf{S}_1(\text{pub}', \rho)$
- Output: $(A'_R, A'_L, A'_O, S'_R, S'_L; y', z'; (T'_i)_{-m+2}^{m+2}; x'; \mu', \hat{t}', \mathbf{l}', \mathbf{r}')$

Simulator S_3. This simulator checks that the subverted commitment parameters are valid. If so, it invokes S_2 and returns S_2's output directly.

- Inputs: $\text{pub} = (\mathbf{g}, \mathbf{h}, g_0, h_0, h, \mathbf{P}, \mathbf{V}, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O, \forall i \in [1, m]\, \mathbf{W}_{V\,i}, \mathbf{W}_P, \mathbf{d},$ $f_{W,R}, f_{W,L}, f_{W,O}, f_{a,R}, f_{a,L}, f_{a,O}); \rho$
- If the CRS does not pass the CRSCheck verification algorithm, return $\perp$.
- $(A'_R, A'_L, A'_O, S'_R, S'_L; y', z'; (T'_i)_{-m+2}^{m+2}; x'; \mu', \hat{t}', \mathbf{l}', \mathbf{r}') \leftarrow \mathsf{S}_1(\text{pub}, \rho)$
- Output $(A'_R, A'_L, A'_O, S'_R, S'_L; y', z'; (T'_i)_{-m+2}^{m+2}; x'; \mu', \hat{t}', \mathbf{l}', \mathbf{r}')$

Putting together Lemmas 3, 4, 5, and 6, we obtain the theorem.

Lemma 3 **(P_0 has Perfect SHVZK).** *For all pairs of interactive adversaries $\mathcal{A}_1, \mathcal{A}_2$*

$$P\left[(\sigma, u, w) \in R_0 \textit{ and } \mathcal{A}_1(tr) = 1 \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, u, w), \mathcal{V}(\sigma, u; \rho)\rangle \end{array}\right] =$$

$$P\left[(\sigma, u, w) \in R_0 \textit{ and } \mathcal{A}_1(tr) = 1 \;\middle|\; \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \mathsf{S}_0(\sigma, u, \rho) \end{array}\right]$$

Proof. Follows directly from the security of Bulletproofs.

Lemma 4 **(P_0 has Perfect SHVZK $\rightarrow$ P_1 has Perfect SHVZK).** *Let us assume that P_0 has Perfect SHVZK using simulator S_0. Then, for all pairs of interactive adversaries $\mathcal{A}_1, \mathcal{A}_2$*

$$P\left[(\sigma, u, w) \in R_1 \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, u, w), \mathcal{V}(\sigma, u; \rho)\rangle \end{array}\right] =$$

$$P\left[(\sigma, u, w) \in R_1 \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \mathsf{S}_1(\sigma, u, \rho) \end{array}\right]$$

Proof. By the assumption that P_0 has perfect SHVZK, the values produced by S_0 are indistinguishable from those in a valid proof produced by an honest prover interacting with an honest verifier. Now, we show that the values computed by S_1 are also indistinguishable from those in a valid proof from an honest prover. First, we note that S_L is fully defined by the following verification equation.

$$h^\mu \mathbf{g}^\mathbf{l} \mathbf{h}^{*\mathbf{r}} = A_L^{x^{-1}} \cdot A_R^x \cdot A_O^{x^{-2}} \cdot \prod_{i=1}^m V_i^{x^{i+3}} \cdot \mathbf{h}^{*-\mathbf{y}^n} \cdot W_L^x \cdot W_R^{x^{-1}} \cdot W_O^{x^2} \cdot \prod_{i=1}^m W_{V_i}^{-(i+3)} \cdot S_L^{x^{-3}} \cdot S_R^{x^3}$$

To ensure that this relation holds, S_1 modifies S_0's computation of S_L to incorporate the V_i and W_{V_i} terms. Specifically, S_1 multiplies both sides of the equation S_0 used to compute S_L by $(\sum_{i=1}^m V_i^{x^{i+3}} \cdot \sum_{i=1}^m W_{V_i}^{-(i+3)})^{x^3}$ in order to satisfy the relation. Similarly, T_0 is uniquely defined by the following verification equation.

$$T_0 = g_0^{x^0(\delta(y,z) + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{d}\rangle)} \cdot P^{x^0(\mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_P \cdot \rho)} \cdot \prod_{i=-(m+2), i \neq 0}^{m+2} T_i^{x^i}$$

S_1 modifies S_0's computation of T_0 to incorporate the additional T_i terms and satisfy the relation. Thus, all relations between values that are visible to the verifier hold.

Finally, S_1 creates the T_i's that were not included in the output of S_0 by sampling random group elements. Since the honestly produced T_i's, for $i \neq 0$, are perfectly hiding commitments, they are indistinguishable from random group elements in the verifier's view. This completes the simulation. S_1 is efficient and produces a transcript that is identically distributed to that of an honestly computed proof, so special honest-verifier zero-knowledge holds.

Lemma 5 **(P_1 has Perfect SHVZK $\rightarrow$ P_2 has Perfect SHVZK).** *Let us assume that P_1 has Perfect SHVZK using simulator S_1. Then, for all pairs of interactive adversaries $\mathcal{A}_1, \mathcal{A}_2$*

$$P\left[(\sigma, u, w) \in R_2 \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, u, w), \mathcal{V}(\sigma, u; \rho)\rangle \end{array}\right] =$$

$$P\left[(\sigma, u, w) \in R_2 \textit{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \mathsf{S}_2(\sigma, u, \rho) \end{array}\right]$$

Proof. The weights are uniquely defined by the verifier's challenge, so S_1's computation ensures that this relation holds. By the assumption that P_1 has perfect SHVZK, the values produced by S_1 are indistinguishable from those in a valid proof produced by an honest prover interacting with an honest verifier. Thus, special honest-verifier zero-knowledge holds for P_2.

Lemma 6 **(P_2 has Perfect SHVZK $\to$ P_3 has Subversion ZK).** *Let us assume that* P_2 *has Perfect SHVZK using simulator* S_2*. Then, for all triples of interactive adversaries* $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$

$$P\left[(\sigma, u, w) \in R_3 \text{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathsf{Setup}(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \langle \mathcal{P}^*(\sigma, u, w), \mathcal{V}(\sigma, u; \rho)\rangle \end{array}\right] =$$

$$P\left[(\sigma, u, w) \in R_3 \text{ and } \mathcal{A}_1(tr) = 1 \,\middle|\, \begin{array}{l} \sigma \leftarrow \mathcal{A}_3(1^\lambda), (u, w, \rho) \leftarrow \mathcal{A}_2(\sigma) \\ tr \leftarrow \mathsf{S}_3(\sigma, u, \rho) \end{array}\right]$$

Proof. If the CRS does not pass the CRSCheck algorithm, both an honest prover and S_3 will return $\perp$. Otherwise, S_3 invokes S_2, which by our assumption outputs transcripts that are identically distributed to those from an honest prover. Thus, subversion zero-knowledge holds for P_3.

D.3 Computational Witness-Extended Emulation

First, we state the following forking lemma that we will use in the proof of computational witness-extended emulation. It is build on the forking lemma by Bootle et al [24]. The main use of the forking lemma is that it abstracts any adversarial influence on the distribution of the challenges. It states that if there exists an efficient algorithm that can extract given *any* tree of challenges, then the protocol must have witness extended emulation. This extractor does not care about the distribution of challenges other than that it requires a certain number of distinct challenges per fork. We state an even more powerful version of the forking lemma. Our adapted forking lemma allows the extraction algorithm to have a small (negligble) failure probability on any transcript. Importantly the probability is independent of the distribution of the challenges. This modification enables us to have small failure probabilities, for example when verifying a proof for a randomized functionality.

Theorem 4 **(Forking Lemma).** *Let* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *be a* $(2k+1)$*-move, public coin interactive protocol. Let* χ *be a witness extraction algorithm that succeeds with probability* $1 - \mu(\lambda)$ *for some negligible function* $\mu(\lambda)$ *in extracting a witness from any* $(n_1, \ldots, n_k)$*-tree of accepting transcripts in probabilistic polynomial time. Assume that* $\prod_{i=1}^{k} n_i$ *is bounded above by a polynomial in the security parameter* λ*. Then,* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *has computational witness-extended emulation.*

Proof. We extend the forking lemma by Bootle et al [24]. In the proof of the lemma they built a tree finder algorithm which succeeds with overwhelming probability in finding a tree of transcripts with suitable challenges. We slightly relax the lemma by not requiring that the extractor χ succeeds with probability 1 but instead only with overwhelming probability. Note that this probability is independent of the challenges on which the extractor forks! The relaxed forking lemma still holds because the probability that the PPT witness extended emulation adversary $(\mathcal{P}^*, \mathcal{A}_1, \mathcal{A}_2)$ can create *any* tree of accepting transcripts on which χ fails to extract is less than or equal to the total number of transcripts the adversary produces times $\mu(\lambda)$. The total number of transcripts is polynomial in λ so the overall probability is still negligible.

In order to prove computational witness extended emulation, we construct an extractor χ. Our construction closely follows the construction of the Bulletproofs extractor, but with two main differences. The first is that χ needs to use a factor of $O(m)$ additional valid transcripts with different challenges in order to extract $\mathbf{v}_1, \ldots, \mathbf{v}_m$ and $\gamma_1, \ldots, \gamma_m$ along with the other intermediate values. The second difference is that once χ has extracted the description of the circuit, it must do an additional rewind of the prover and verifier due to the adaptively-defined, randomized circuit. Using several $\mathbf{c}$ challenges from the verifier, χ is able to extract the witness. χ inherits the additional failure probability from the probabilistic verification of the statement itself. However this failure probability is negligible and independent of the challenges on which χ forks. The forking lemma shows that χ still implies witness extended emulation. We describe the full construction of χ below.

χ runs the prover with n different values of y, $Q + 1$ different values of z, and $4m + 15$ different values of x. χ invokes the extractor $\chi_{\text{InnerProduct}}$ to extract the inner product vectors, as well as the extractor $\chi_{\text{PolyCommit}}$ to extract the coefficients of the polynomial, of each of the transcripts. Because $\chi_{\text{InnerProduct}}$ uses $O(n^2)$ transcripts, and $\chi_{\text{PolyCommit}}$ uses $O(m)$ transcripts, χ produces $O(n^3 \cdot m)$ valid proof transcripts.

The extraction proceeds in three stages. First, χ uses $\chi_{\text{InnerProduct}}$ and $\chi_{\text{PolyCommit}}$ to extract the inner product identity. Next, χ extracts the wire vectors for the multiplication gates as well as the weights for the linear constraints. Finally, χ extracts the witnesses for the proof statement.

In the first stage, $\chi_{\text{InnerProduct}}$ extracts a witness $\mathbf{l}, \mathbf{r}$ to the inner product argument such that

$$\begin{aligned} h^{\mu} \cdot \mathbf{g}^{\mathbf{l}} \cdot (\mathbf{h}^*)^{\mathbf{r}} &= P \\ \langle \mathbf{l}, \mathbf{r} \rangle &= \hat{t}. \end{aligned}$$

Using $2m + 10$ valid transcripts and $\mathbf{l}, \mathbf{r}$ for different x's, χ uses linear combinations of the following equation

$$h^{\mu}\mathbf{g}^{\mathbf{l}}\mathbf{h}^{*\mathbf{r}} \stackrel{?}{=} A_L^{x^{-1}} \cdot A_R^{x} \cdot A_O^{x^{-2}} \cdot \prod_{i=1}^{m} V_i^{x^{i+3}} \cdot \mathbf{h}^{*-\mathbf{y}^n} \cdot W_L^{x} \cdot W_R^{x^{-1}} \cdot W_O^{x^2} \cdot \prod_{i=1}^{m} {W_V}_i^{-(i+3)} \cdot S_L^{x^{-3}} \cdot S_R^{x^3}$$

to compute $\mathbf{v}_1, \ldots, \mathbf{v}_m$ and $\gamma_1, \ldots, \gamma_m$ along with $\mathbf{a}_R, \mathbf{a}_L, \mathbf{a}_O, \mathbf{s}_L, \mathbf{s}_R, \beta, \kappa, \phi, \lambda$ such that

$$\begin{aligned} A_R &= \mathbf{g}^{\mathbf{a}_R} h^{\alpha} \\ A_L &= \mathbf{h}^{\mathbf{a}_L} h^{\beta} \\ A_O &= \mathbf{g}^{\mathbf{a}_O} h^{\kappa} \\ S_R &= \mathbf{g}^{\mathbf{s}_L} h^{\lambda} \\ S_L &= \mathbf{h}^{\mathbf{s}_R} h^{\phi} \\ V_i &= \mathbf{h}^{\mathbf{v}_i} h^{\gamma} \;\; \forall i \in [m] \end{aligned}$$

If χ can compute different representations of any of these values for any other set of challenges (x, y, z), then we have a non-trivial discrete logarithm relation between independent generators $\mathbf{g}, \mathbf{h}, h$, which contradicts the discrete logarithm assumption.

Next, for all challenges (x, y, z), the following holds.

$$\mathbf{l} = \mathbf{a}_L \cdot x^{-1} + \mathbf{a}_O \cdot x^{-2} + \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R) \cdot x^{-1} + \mathbf{s}_L \cdot x^{-3} + \sum_{i=1}^{m} \mathbf{W}_{V\,i} \cdot x^{-(i+3)}$$

$$\mathbf{r} = \mathbf{y}^n \circ \mathbf{a}_R \cdot x - \mathbf{y}^n \cdot x^2 + \mathbf{z}_{[1:]}^{Q+1} (\mathbf{W}_L \cdot x + \mathbf{W}_R \cdot x^2) + \mathbf{y}^n \circ \mathbf{s}_R \cdot x^3 + \sum_{i=1}^{m} V_i \cdot x^{i+3}$$

If these equalities do not hold for all challenges and $\mathbf{l}, \mathbf{r}$ from the transcript, then we have two distinct representations of the same group element using the independent generators $\mathbf{g}, \mathbf{h}, h$, which is a non-trivial discrete logarithm relation. This contradicts the discrete logarithm assumption.

Next, χ invokes $\chi_{\mathsf{PolyCommit}}$, defined in [24], to extract $\{t_i, \tau_i\}_{i=-(m+2),\neq 0}^{m+2}$ such that $T_i = g_0^{t_i} h_0^{\tau_i}$. $\chi_{\mathsf{PolyCommit}}$ uses fixed values y, z and $2m+5$ transcripts with different x challenges. Additionally, χ can compute $(p_i, \rho_i)_{i=1}^{l}$ such that $P_i = g_0^{p_i} h_0^{\rho_i} \forall i \in [1, l]$.

For all transcripts, the following must hold.

$$\hat{t} = \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_P \cdot \mathbf{p} + d \rangle + \delta(y, z) + \sum_{i=-(m+2),\neq 0}^{m+2} t_i x^i$$

If this equality does not hold, we have a non-trivial discrete log relation between independent generators g, h, which contradicts the discrete logarithm assumption. Then, for all challenges of x, for all y, z, we have the following

$$\sum_{i=-(m+2)}^{m+2} t_i x^i - p(x) = 0$$

where $p(x) = \sum_{i=-(m+2)}^{m+2} p_i x^i =< l(x), r(x) >$.

Since the polynomial $t(x) - p(x)$ is of degree $2m + 4$ but has at least $2m + 5$ roots (the x challenges), it must be the zero polynomial. Thus, $t(x) = \langle l(x), r(x) \rangle$ and

$t_0 = p_0$. This means that the following holds for all y, z challenges.

$$t_0 = \langle \mathbf{a}_L, \mathbf{a}_R \cdot \mathbf{y}^n \rangle - \langle \mathbf{a}_O, \mathbf{y}^n \rangle + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{u} \rangle + \delta(y, z) = \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d} \rangle + \delta(y, z)$$

Now, using $n(Q+1)$ different transcripts (n different y challenges and $(Q+1)$ different z challenges), we can infer the following.

$$\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$$

$$\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O + \sum_{i=1}^{m} \mathbf{W}_{Vi} \cdot \mathbf{v}_i = \mathbf{W}_P \cdot \mathbf{p} + \mathbf{d}$$

Thus, we have recovered the Hadamard product relation and the linear constraints that represent our circuit.

Finally, χ runs the prover with $O(n)$ challenges $\mathbf{c}$ and uses linear combinations of the equations below to extract $\mathbf{w}$.

$$\mathbf{W}_L = f_{\mathbf{W}_L}(\mathbf{c}) \qquad \mathbf{W}_R = f_{\mathbf{W}_L}(\mathbf{c}) \qquad \mathbf{W}_O = f_{\mathbf{W}_L}(\mathbf{c})$$
$$\mathbf{W}_{Vi} = f_{\mathbf{W}_{Vi}}(\mathbf{c}) \forall i \in [1, m] \qquad \mathbf{W}_P = f_{\mathbf{W}_P}(\mathbf{c})$$
$$\mathbf{a}_L = f_{\mathbf{a}_L}(\mathbf{c}, \mathbf{w}) \in \mathbb{Z}_p^n \qquad \mathbf{a}_R = f_{\mathbf{a}_R}(\mathbf{c}, \mathbf{w}) \in \mathbb{Z}_p^n \qquad \mathbf{a}_O = f_{\mathbf{a}_O}(\mathbf{c}, \mathbf{w}) \in \mathbb{Z}_p^n$$

In this last step, χ has two failure modes: first, it may compute a non-trivial discrete log relation between independent generators independent generators $\mathbf{g}, \mathbf{h}, h$. This occurs with negligible probability due to the discrete log hardness assumption. Second, it may extract an invalid witness due to the probabilistic verification of the statement itself.

Let us assume that the failure probability of the probabilistic verification is a negligible function $\epsilon(\lambda)$ and that the failure probability of the rest of the extraction is a negligible function $\mu(\lambda)$. Since the extraction from the statement is the last step, by the extended forking lemma, the total failure probability of χ_2 is $p = \epsilon(\lambda) \cdot \mu(\lambda)$, which is negligible.

χ rewinds the prover $O(n^3 \cdot Q \cdot m)$ times. Q is the number of linear constraints, so $Q \leq 2n$. We assume that $m = O(poly(n))$. Thus, the number of transcripts is polynomial in the security parameter λ.

We have shown that extraction is efficient and returns a valid witness with overwhelming probability, which proves that computational witness-extended emulation holds.