Mutable Batch Arguments and Applications

Rishab Goyal
UW-Madison*

Abstract

Non-interactive batch arguments (BARGs) let a prover compute a single proof π proving validity of a 'batch' of k **NP** statements $x_1, \ldots, x_k$. The two central features of BARGs are succinctness and soundness. Succinctness states that proof size, $|\pi|$ does not grow with k; while soundness states a polytime cheating prover cannot create an accepting proof for any invalid batch of statements.

In this work, we put forth a new concept of mutability for batch arguments, called *mutable batch arguments*. Our goal is to re-envision how we think about and use BARGs. Traditionally, a BARG proof string π is an *immutable* encoding of k **NP** witness $\omega_1, \ldots, \omega_k$. In a mutable BARG system, each proof string π is a *mutable* encoding of original witnesses. Thus, a mutable BARG captures and enables computations over a batch proof π. We also study new privacy notions for mutable BARGs, guaranteeing that a mutated proof hides all non-trivial information. Such mutable BARGs are a naturally good fit for many privacy sensitive applications.

Our main contributions can be summarized as introducing the general concept of mutable BARGs, identifying new non-trivial classes of feasible mutations over BARGs, designing new constructions for mutable BARGs satisfying mutation privacy for these classes from standard cryptographic assumptions, and developing applications of mutable BARGs to advanced signatures such as homomorphic signatures, redactable signatures, and aggregate signatures. Our results improve state-of-the-art known for many such signature systems either in terms of functionality, efficiency, security, or versatility (in terms of cryptographic assumptions).

1 Introduction

Batch arguments (BARGs) enable a prover to compute a *succinct* proof certifying validity of k **NP** statements $x_1, \ldots, x_k \in \mathcal{L}$. Succinctness requires the proof size to be $\mathsf{poly}(\lambda, \log k)$, thus independent of batch size as $k \leq 2^\lambda$ (but could grow with the size of a single witness). The standard soundness notion states that no polynomial-time cheating prover can compute an accepting proof π for any batch of statements containing at least one invalid instance with non-negligible probability. To avoid trivial complexity-theoretic barriers, BARGs are usually defined in the common reference string (CRS) model.

Over the last few years, BARGs have emerged as a powerful tool in the study and applications of succinct proofs. They have led to a significant swell-up in current cryptographic capabilities leading to important progress on major longstanding open problems (see the non-exhaustive list [RRR16, BHK17, KPY19, CJJ21b, CJJ21a, KVZ21, WW22, HJKS22, DGKV22, PP22, KLVW23, GSWW22,

*Email: rishab@cs.wisc.edu. Support for this research was provided by OVCRGE at UW–Madison with funding from the Wisconsin Alumni Research Foundation.

CGJ+22, KLV22] and references therein). One of the reasons behind the success of BARGs is: unlike (general-purpose) succinct non-interactive arguments (SNARGs) [Kil92, Mic94], BARGs do not suffer from strong non-black-box proof barriers [GW11] that hinder SNARGs. Today, we have efficient standard model BARGs from a variety of standard falsifiable assumptions such as LWE, or DLIN, or sub-exponential DDH (and QR) [CJJ21b, CJJ21a, KVZ21, WW22, HJKS22, DGKV22, PP22, CGJ+22, KLV22, KLVW23].

This work. We put forth a new framework for batch arguments, called *mutable batch arguments*. Our goal is to re-envision how we think about succinct proof systems.

Traditionally, a succinct proof system is viewed as a cryptographic operation that encodes an **NP** witness ω into a short (proof) string π. Here proof π serves as a short, sound, and verifiable substitute for the actual witness ω. Unfortunately, such a cryptographic encoding operation (called the prover's algorithm) creates *immutable* encodings π. That is, they are frozen in time and do not support general *mutation* operations that an actual witness does. As a simple illustrative example, consider you have a witness ω for statement x. Given ω, you can use a portion of it (or even combine it with another witness ω' for statement x') to create a witness $\hat{\omega}$ for a new statement $\hat{x}$. Such operations are not natively supported over succinct proofs. In words, once we encode an **NP** witness ω into a succinct proof π, we can no longer compute on it. Because π does not contain enough information about ω due to the succinctness property.

Speaking more generally, we study a very natural question towards advancing succinct proofs– *"Can we compute over succinctly proven data?"* We introduce a new framework, called *mutable BARGs*, for capturing computations over batch arguments. We provide multiple constructions for mutable BARGs with differing capabilities from a variety of standard falsifiable assumptions such as LWE, DLIN, QR, DDH. Mutable BARGs turn out to be more useful cryptographic tools enabling a variety of new applications. To illustrate this, we design new constructions for redactable, aggregate, and homomorphic signatures [JMSW02, SBZ01, BGLS03, AB09, BFKW09, BF11, GVW15, GV22] with improved efficiency and additional features from mutable BARGs.

Our results. We introduce the concept of general mutability in BARGs, and formally define a new framework to capture mutable BARGs. We study three overlapping classes of mutation functions – identity mutations, subset mutations, monotone policy batchNP mutations. For all these classes, we provide new constructions of mutable BARGs and use them to enable new applications. We also study new privacy notions for mutable BARGs, which states that a mutated proof does not reveal any non-trivial information about the original batch proofs or the instances.

In the next section, we provide a technical explanation of mutability in BARGs, and expand on these mutation classes. Briefly, identity mutations enable a user to generate a 'pseudo-witness', for any instance in a batch, given just the corresponding batch proof π. Subset mutations enable a user to combine two or more batch proofs $\pi_1, \pi_2, \ldots$ for any (possibly non-overlapping) batches of instances $\{x_{1,i}\}_i, \{x_{2,i}\}_i, \ldots$ to create a new batch proof for any subset of the underlying instances. And, monotone policy batchNP mutations further generalize subset mutations by enabling a user to compute a succinct proof for any monotone circuit satisfiability language over the underlying instances. In summary, we show the following results in this work.

Informal theorem 1. Assuming LWE, or DLIN, or sub-exponential DDH (and QR), there exists mutable BARGs for *subset* mutation functions satisyfing succinctness, soundness, and privacy.

(As we explain next, *subset* mutation functions contain *identity* mutation functions. Thus, the above captures mutable BARGs for identity mutation functions.)

Informal theorem 2. Assuming LWE, there exists mutable BARGs for *monotone-policy* batchNP mutation functions satisyfing succinctness, soundness, and privacy.

Application 1. Assuming mutable BARGs for *identity* mutations with mutation privacy, there exists locally verifiable aggregate signatures with full privacy.

Application 2. Assuming mutable BARGs for *subset* mutations with mutation privacy, there exists redactable aggregate signatures with full deletion privacy.

Application 3. Assuming mutable BARGs for *monotone-policy* batchNP mutations with mutation privacy, there exists homomorphic signatures for all polynomial-sized boolean formuale (i.e., log-depth circuits) with context hiding.

Application 4. Assuming mutable BARGs for *monotone-policy* batchNP mutations with mutation privacy, there exists homomorphic signatures for all polynomial-sized monotone circuits with context hiding.

Related work. We add that computing over cryptographically proven data is a well-studied research topic. In early 90s, De Santis and Yung [DSY91] proposed proofs that enabled certain basic computations over proofs, and there has been a long line of follow-up works [BDI+99, Val08, BCC+09, DHLAW10, CT10, AN11, BSW12, DSY12, BCCT13, AGP14, ADKL19] studying numerous generalizations. A major drawback of these generalizations is that they lack proof succinctness[1].

We know how to design non-interactive proof systems that support homomorphic computations [ADKL19], *when succinctness is not a desiderata*. Our focus is to study and enable admissible computations over succinct proofs. As we discuss next, succinct mutable proofs that enable arbitrary computations are impossible. Therefore, we view *succinct mutable* proofs and *non-succinct homomorphic* proofs as related concepts, but non-overlapping research topics due to diverging techniques and desiderata.

2 Technical overview

In this section, we provide a detailed overview of mutable BARGs, our constructions, technical ideas, and new applications. We begin by recalling the standard syntax for BARGs, and follow it up by our framework for defining mutability in BARGs.

Reviewing BARGs. Let us recall BARGs as defined in the common reference string (CRS) model. Let $\mathcal{L}$ be any fixed **NP** language. A trusted party samples crs which is available to all users (provers and verifiers). Given crs, a prover $\mathcal{P}$ generates a short proof π, for a sequence of k satisfying instance-witness pairs $(x_1, \omega_1), \ldots, (x_k, \omega_k)$, such that a verifier $\mathcal{V}$ can check $\{x_i \in \mathcal{L}\}_i$ by inspecting a single proof π. Succinctness states that the CRS and proof sizes $|\mathsf{crs}|, |\pi| \leq \mathsf{poly}(\lambda, \log k)$. They

[1]While there has been research on *'succinct metaproofs'* [Val08, BSW12, CT10, BCCT13] (or recursive succinct proofs), they do not think mutation/computation as generally. Their goal is to purely design a '*succinct* proof that there is a *succinct* proof and so on'.

can grow with the size of a single witness, but not with the batch size. Soundness states that no polynomial time cheating prover can create an accepting proof π^* for some $x_1, \ldots, x_k$ such that $x_i \notin \mathcal{L}$ for some i. Many recent works have also defined and achieved a stronger extractability property called 'somewhere extractability' [CJJ21a]. It states that for any index i^*, we can sample crs with a trapdoor such that one can efficiently extract a witness ω_{i^*} for the i^{*th} statement x_{i^*} from any accepting proof π. Moreover, such a crs does not leak any information about extraction index i^*.

It has been used implicitly [CJJ21a] and observed explicitly in prior works [DGKV22, KLVW23] that somewhere extractability can be generically achieved by combining BARGs with any somewhere extractable hash (SEH) function [HW15, OPWW15]. For completeness, we show this generic transformation formally in Appendix B. BARGs with somewhere extractability are commonly referred as seBARGs, and these will be a central tool that we generalize and use. Next, we introduce the concept of mutable BARGs.

Mutable BARGs. Intuitively, a mutable BARG is just a regular BARG system with one special feature. It supports a set of pre-defined mutation operations on top of batch proofs (without knowing the original witnesses). A mutable BARG is associated with a class of mutation functions $\mathcal{P}_\ell$, where each function $P \in \mathcal{P}_\ell$ takes ℓ instances $x_1, \ldots, x_\ell$ as an input, and it outputs a mutated instance x_P. Let $\mathcal{L}$ denote the **NP** language for the BARG. We use $\mathcal{L}_P$ to denote the **NP** language associated with the mutation function P. We call $\mathcal{L}_P$ as the mutated language, and highlight that $\mathcal{L}_P$ could be different for different function choices $P \in \mathcal{P}_\ell$.

In addition to the standard prover/verifier algorithms, a mutable BARG system has a proof mutation algorithm that takes a (mutation) function $P \in \mathcal{P}_\ell$ as an input along with an ℓ-length sequence of tuples, where each tuple contains an index, a list of instances, and a corresponding batch proof. That is, the inputs are P, $\{(j_i, X_i, \pi_i)\}_{i \le \ell}$, where each instance list X_i contains k instances $(x_{i,1}, \ldots, x_{i,k})$ and π_i is supposed to be a valid batch proof for instances in X_i. The algorithm outputs a mutated proof $\widehat{\pi}$. Intuitively, the property we desire from mutable BARGs is that the proof $\widehat{\pi}$ should be a valid succinct proof for the mutated instance $x_P = P(x_{1,j_1}, \ldots, x_{\ell,j_\ell})$ as per language $\mathcal{L}_P$. To capture this last part, we consider an additional verifier algorithm. It takes a mutation function P, mutated instance x_P, and a mutated proof $\widehat{\pi}$ as inputs, and checks whether $x_P \in \mathcal{L}_P$ given proof $\widehat{\pi}$.

As in any typical succinct proof system, a mutable BARG must satisfy succinctness, completeness, and soundness. Succinctness and completeness can be naturally defined for mutated proofs. And, soundness states that it should be computational infeasible for any cheating prover to create an accepting *mutated* proof $\widehat{\pi}$ for any invalid mutation instance $x_P \notin \mathcal{L}_P$ for any mutation function $P \in \mathcal{P}_\ell$. As a natural extension, we can also define a knowledge soundness property for mutable proofs. However, we avoid discussing it here for simplicity. Later, for certain specific choices of mutation classes, we consider and achieve mutable BARGs with varying levels of knowledge soundness. In addition to the above, we also study a privacy property for mutated proofs. The goal behind privacy is to ensure a mutated proof hides all non-trivial information about the input batch proofs and instances. That is, a mutated proof $\widehat{\pi}$ for any function-instance pair (P, x_P) does not reveal any information about the input proofs and instances, $\{(j_i, X_i, \pi_i)\}_i$, except whatever is revealed by (P, x_P). As we elaborate later, mutable BARGs satisfying privacy are a naturally good fit for many privacy sensitive applications.

Summary and plan. In this work, we formally define the above framework for mutable BARGs as

an anchoring point. We study mutable BARGs for natural classes of mutation functions with two goals in mind– (1) we can design mutable BARGs for that particular class while proving security under standard falsifiable assumptions, (2) they are useful for new applications. In the remaining overview, we incrementally raise the complexity of class of mutation functions that we can support, and show how to design mutable BARGs for that class from standard falsifiable assumptions. Along the way, we also discuss new applications enabled by each mutable BARG system. Lastly, we discuss a broad feasibility result for mutable BARGs supporting general functions from idealized assumptions (such as SNARKs).

Feasibility and boundaries. We briefly remark that one cannot hope to design mutable proofs for all efficiently computable functions. Intuitively, this can be understood as follows– we can only build mutable proofs for a function class if one could efficiently decide membership of a mutated instance x_P in $\mathcal{L}_P$, given just the function P and sequence of index-instances-proof tuples $\{(j_i, X_i, \pi_i)\}_{i \leq \ell}$. If membership of x_P in language $\mathcal{L}_P$ cannot be efficiently decided, given the instance lists and their proofs, then mutable BARGs for such a function class will be impossible to design due to appropriate complexity-theoretic separations. In other words, a batch proof π_i cannot contain non-trivial information about every input witness due to succinctness of π_i. Thus, if any valid witness of a mutated instance x_P must contain (or non-trivially depends on) the witness for the j_i^{th} instance in list X_i, then it would be impossible to design mutable BARGs for such mutation functions. We elaborate on this later in Remark 4.4, and next, let us dive into our main results for mutable BARGs and their applications.

2.1 Identity Mutations

We start by identifying a core fundamental class of mutation functions for BARGs that we call identity mutations. Later we show that mutable BARGs for identity mutations can be used as a core component to design mutable BARGs for more complex function classes.

Defining the mutation class. The identity mutation class, denoted as $\mathcal{P}^{\mathsf{local}}$, contains a single 'identity' program $P = \mathbb{I}$, where the function is defined over a single index-instances-proof tuple (j, X, π) (thus $\ell = 1$, that is mutation function acts on a single batch proof). Further, the associated language is the same **NP** language, $\mathcal{L}_P = \mathcal{L}$. Thus, for $\mathcal{P}^{\mathsf{local}}$, the mutated proof $\widehat{\pi}$ can be viewed as a 'pseudo-witness' for $x_j \in \mathcal{L}$ (the j^{th} instance in instance list X). We routinely refer to mutable BARGs for identity mutations as *locally verifiable* BARGs (lv-BARGs).

For an easier exposition, we use a specialized syntax for lv-BARGs. We call the mutation algorithm to be the local proof opening algorithm LOpen. It reads k instances $\{x_i\}_i$, target index j, and a proof π. It outputs a mutated proof which is parsed as two separate components (for technical reasons discussed later)— opening information aux_j and a mutated batch proof π'. Next, we call the mutated proof verification algorithm to be the local verification algorithm LVfy. It takes a single instance x, index j, a mutated batch proof π', and opening information aux, and outputs a bit. In the main body, we discuss multiple notions of security for such lv-BARGs ranging from soundness of locally opened proofs, (somewhere) extractability w.r.t. mutated proofs, as well as a strong notion of full privacy (which says a mutated batch proof π and opening information aux) reveal nothing about original instance list or corresponding witnesses. For the purposes of this overview, it is sufficient to consider just the regular soundness property as discussed for mutable BARGs. We highlight that above changes are purely syntactic simplifications.

With the above formalism, we take a step-by-step approach to build lv-BARGs. First, we

provide a basic construction for lv-BARGs that does not achieve mutation privacy, and later we upgrade them to satisfy mutation privacy, while preserving many interesting features.

Local verifiability via Merkle Trees. Let us start by restating the desiderata. A mutable BARG for identity mutations should allow taking a batch proof π, for some k instances $x_1, \ldots, x_k$, and turning it into a 'pseudo-witness' for any instance x_j in that batch. While one could simply set the mutated proof as π along with the k instances $\{x_i\}_i$ as such a pseudo-witness, this is not succinct. We want a mutated proof to be *succinct*, thus we cannot give out the full batch of instances as part of the proof.

Our idea to solve this succinctness issue is to use the folklore commit-and-prove approach, where Merkle tree based hashing is used to ensure succinctness is no longer violated. Recall that in a Merkle tree based hash function, one can create a short digest for a large database as well as generate short proofs of membership for any data block in the original database. These proofs of membership are much smaller than the original database, and an attacker cannot create fake proofs of membership. That is, they provide short local openings (proofs of membership) of any specific data block w.r.t. its digest, and these short local openings can be verified succinctly w.r.t. its digest.

Getting back to our main idea, the prover simply commits the batch of instances using Merkle tree hashing and uses local openings for each instance x_i as an additional witness. Concretely, the prover hashes the instance list $X = (x_1, \ldots, x_k)$ to create a digest $h_x = \mathsf{H}(\mathsf{hk}, X)$. It creates a short opening op_i for each instance x_i w.r.t. h_x. Here op_i proves that x_i is the i^{th} instance block in h_x. Next, it creates a batch proof for slightly expanded language $\hat{\mathcal{L}}$ where each instance now contains the digest h_x and an index i. Here it uses the actual instance x_i, its witness ω_i, and the opening op_i as the new witness. In words, membership for language $\hat{\mathcal{L}}$ checks:

— ω_i is a valid witness for x_i (w.r.t. language $\mathcal{L}$), and

— op_i is a valid opening of x_i (w.r.t. h_x).

The new batch proof only contains the underlying batch proof. This is because the global verifier can re-compute the digest h_x at verification time from the batch of instances. Now to create efficient local openings for batch proofs (i.e., pseudo-witnesses), the local opening algorithm LOpen creates the digest h_x from the entire batch of instances, and generates an opening op_j for target instance x_j. It sets the auxiliary opening information as $\mathsf{aux}_j = (h_x, \mathsf{op}_j)$. The local verifier then simply checks:

— op_j is a valid opening for x_j w.r.t. h_x (i.e., x_j was hashed down to create h_x), and

— π is a valid BARG proof for instances $(h_x, 1), \ldots, (h_x, k)$.

Crucially, the local verifier just needs aux_j for verification, and not the full list of instances. Thus, the mutated proof no longer needs to grow with the full batch size. We also highlight that this construction satisfies a rather interesting property of 'proof-independent openings'. That is, the opening algorithm does not need batch proof π to generate auxiliary opening information aux_j, and only the batch of instances is sufficient. Coincidentally, the same approach was used in the [CJJ21a] BARG construction, but rather for proving an online-offline verification property. As we explain above, we can re-purpose it as an interesting local verifiability property, which simply corresponds to a fundamental mutation class of identity mutations.

In order to prove stronger security properties, we use a somewhere extractable hash (SEH) function [HW15, OPWW15] instead of plain Merkle trees in the main body. We provide further details in Section 6.

Identity mutations with full privacy. Our next goal is to upgrade our mutable BARGs for identity mutations (i.e., lv-BARGs) to satisfy *privacy*. Here by privacy, we mean that the mutated proof (along with the opening information) does not reveal any non-trivial information about the batch of instances that were not opened. In other words, any two mutated proofs for the same instance should look indistinguishable, even when they are created by mutating two distinguishable batch proofs.

Our strategy to achieve privacy is to ensure– (a) that a batch proof hides all information about the original witnesses, and (b) information about original instances can be efficiently hidden during mutation. Clearly, if we can ensure both of these properties, then mutation privacy should follow. Suppose that the BARG is already witness hiding (that is, it satisfies the first property), then we show that we can use it to hide any non-trivial information about instances resulting in full mutation privacy. The idea is to commit each instance individually to hide the instance, and use the commitment opening as part of the witness. As long as the commitment opening is hidden, the instance will be hidden by hiding property of commitments. The question becomes whether this is enough. While it seems yes, unfortunately this kills succinctness. We elaborate below.

Suppose the prover creates a fresh commitment to hide each instance x_i as $c_i \leftarrow \mathsf{Com}(x_i)$, and then use the batch of commitments $\{c_i\}_i$ as the batch of instances while using $(\omega_i, x_i, \mathsf{op}_i)$ as the corresponding witness. (Here op_i denotes the opening of x_i w.r.t. c_i, and ω_i is its witness.) Namely, an instance is a commitment of the actual instance, while the witness contains the commitment opening and the actual witness. Observe the membership check can simply be:

— ω_i is a valid witness for x_i, AND

— op_i is a valid opening of x_i (w.r.t. c_i).

At first glance, this seems to work as a cheating verifier cannot learn x_j from a commitment c_j, and batch proof π hides the rest. However, the problem is – *where does the verifier get these commitments from?* A verifier needs the instances which are the commitments in this case. So, either we must add the commitments to the proof, or make them deterministic. The first solution takes away succinctness, while the second takes away privacy. In our previous construction, we used a deterministic Merkle tree hash, thus it was not private.

While it seems we are back to square one, we have actually made progress. Recall that the local opening (proof mutation) algorithm receives the entire list of original instances $\{x_i\}_i$. Thus, the opening algorithm can re-compute the commitments and create a Merkle tree style opening of the commitments along with the commitment opening for just the target instance. Now we could reveal the random coins (i.e., openings) used for generating commitments as part of the original batch proof π, but that again makes it non-succinct. What we need is a succinct representation of all the randomness used in commitments while arguing that the resulting commitments still hide the instances.

This requirement coincidentally is what pseudorandom functions (PRFs) precisely provide. A succinct representation, in the form of a PRF key, of a long list of random values. PRFs exactly fit what we need. Namely, the short description of the commitment openings can be a PRF key. Formally, prover runs as:

1. Sample a PRF key K, and commit instance x_i using randomness generated by K. (e.g., $c_i = \mathsf{Com}(x_i; F_K(i))$)
2. Create a batch proof π using $\{c_i\}_i$ as instances, and $\{(\omega_i, x_i, \mathsf{op}_i = F_K(i))\}_i$ as the witnesses.
3. The new batch proof contains the above batch proof π and the PRF key K.

Clearly, the batch proof verifier can re-compute all the k commitments given the PRF key K, thus correctness is unaffected. Moreover, a local verifier can also verify the batch proof π as long as it receives a local opened batch proof for the underlying BARG scheme (defined w.r.t. commitments as instances) as well as the opening randomness to the target commitment c_j.

Concretely, the new local opening algorithm, LOpen, creates the local opening using the underlying BARG scheme with $\{c_i\}_i$ as the instances[2], and sets the opening information to be the underlying BARG local opening along with the commitment opening $\mathsf{op}_j = F_K(j)$ corresponding to x_j. That is, it omits the PRF key K from the batch proof, and instead adds the PRF evaluation as the opening for the target instance in the auxiliary information (along with opening information for the lv-BARG). The local verifier first verifies the validity of the commitment, and then runs the local verifier for the underlying BARG scheme. The intuition behind privacy is that we could replace the PRF with a truly random function (by a hybrid argument since a mutated proof only contains a PRF evaluation on a single point, not the full PRF key), and then using witness hiding property we can hide the commitment openings for unopened instances, and eventually replace all the unopened commitments to random values. There are some technical subtleties that have to be handled, but the above idea is sufficient for proving mutation privacy. More details provided later in Section 7.

Finally, to finish up our strategy for mutation privacy, we need to hide witnesses within BARGs. We show a rather simple approach for that[3]. The idea is to simply replace each witness with a non-interactive zero-knowledge (NIZK) proof [GMR89]. That is, a prover proceeds as follows:

1. Create a NIZK τ_i for each instance-witness pair (x_i, ω_i),
2. Create the batch proof π using the NIZK proofs $\{\tau_i\}_i$ as witnesses instead of the actual witnesses $\{\omega_i\}_i$.

The witness hiding property follows directly from the zero-knowledge property. That is, each τ_i hides the witness ω_i. Soundness follows by relying on somewhere extractability of BARGs, and moreover, we can prove somewhere extractability of the above scheme if the NIZK scheme also has an extractor[4]. One might wonder if there are alternate approaches to achieve mutation privacy for lv-BARGs. While we also studied alternate strategies (as we discuss in detail later in Section 2.4), they are quite limited. Alternate strategies do not satisfy many interesting properties and features that our current approach provides. We briefly discuss them next.

Complementary mutations (or all-but-one opening). So far we discussed a rather simple, yet fundamental, class of mutation operations called identity mutations. Consider its complementary class

[2]Note that since the local opening algorithm gets access to the full batch of instances $\{x_i\}_i$ and also has access to key K from the proof, thus it can easily compute the batch of committed instances.

[3]Prior works [CW23] did provide alternate strategies for getting witness hiding, but our solution is simpler and does not make any additional structural assumption about the BARG scheme

[4]In the CRS model, a knowledge extractor exists for a non-succinct proof system by using the FLS paradigm [FLS99] via any perfectly correct public-key encryption scheme for instance.

where the goal is to succinctly mutate a batch proof into a mutated proof that proves validity of *all-but-one* instances from the original batch. That is, the mutation algorithm takes the same set of input as for lv-BARGs, which is k instances $\{x_i\}_i$, target index j, and a proof π. But the output proof (aux_j, π') is viewed as a batch proof for instances $\{x_i\}_{i \neq j}$, that is all instances *except* the j^{th} instance x_j. We refer to this as mutable BARGs with complementary mutations as they provide the complementary functionality to lv-BARGs (identity mutable BARGs).

Our design approach for lv-BARGs can be easily generalized to design mutable BARGs for complementary mutations. Moreover, it still satisfies mutation privacy. The core observation is that we currently reveal the PRF evaluation for a single instance where we want to *locally open* the proof. Now if we replace this with a "punctured" PRF key $K\{j\} = \mathrm{Puncture}(K, j)$ [BW13, BGI14, KPTZ13], then a verifier could re-compute the commitment c_i for every instances x_i for $i \neq j$. This is due to correctness of puncturable PRFs which state that $\mathsf{Eval}(K\{j\}, i) = F_K(i)$ for all $i \neq j$. Thus, a mutated proof for all-but-one openings contains the punctured PRF key $K\{j\}$ and commitment c_j. A verifier simply re-computes all other commitments and uses those to verify the mutated batch proof. And, since the commitment c_j is hiding, thus mutation privacy of the above construction follows similar to the mutation privacy for lv-BARGs. The main difference is that we rely on punctured PRF security instead of regular pseudorandomness security. This highlights a distinct advantage of our design approach of starting with witness hiding BARGs. In the next section, we design mutable BARGs for more general mutation classes, thus we do not provide the above construction in the main body.

Fast (witness-independent) proof mutation. Additionally, we highlight that our mutable BARG schemes (both for identity and complementary mutations) have a special feature that the running time of their respective mutation algorithms does **not** depend on the size of original witnesses. Thus, the mutation operation is extremely efficient in our constructions. This is because to mutate a batch proof, we simply compute a hash function (and a hash opening) and re-compute commitments of instances and evaluate a PRF. All these operations do not operate on any witness-dependent proof component, thus are independent of the time needed to even read a single witness or check its validity. While designing mutable BARGs with optimal proof mutation complexity are not a focus of this work, we believe our techniques will be useful for future works interested in resolving such questions.

Next, we provide a natural application of mutable BARGs supporting identity mutations to design aggregate signatures with fast local verification [GV22].

Application 1: locally verifiable aggregate signatures. An aggregate signature [BGLS03] scheme allow public aggregation of a sequence of verification-key-message-signature tuples $\{(\mathsf{vk}_i, m_i, \sigma_i)\}_i$ into a single short aggregated signature $\widehat{\sigma}$. Such signatures prove possession of signatures for m_i under key vk_i (for all i) by just providing $\widehat{\sigma}$. In a recent work [GV22], Goyal and Vaikuntanthan extended the concept of aggregate signatures to a new local verifiability model. Their goal was to enable fast verification of an aggregate signature, where a local verifier can check signature validity for a single message in time and space independent of the number of signatures aggregated. To avoid trivial impossibility, they defined an additional algorithm referred to as the hint generator that creates a short hint for an aggregated signature. Given such a short hint, a verifier can locally and efficiently inspect whether a signature for a particular message was aggregated inside the aggregated signature without reading the full sequence of verification-key-message pairs.

As an application of mutable BARGs supporting identity mutations, we obtain the first locally

verifiable multi-signer aggregate signature satisfying message privacy under adversarial openings from standard assumptions. Prior works on aggregate signatures with local verifiability did not achieve such properties. Either they worked in the single-signer model [GV22], or they relied on general purpose SNARGs.

Our starting point are the recent works [WW22, DGKV22] which made an observation that BARGs can be used generically to obtain aggregate signature. The key intuition was that the aggregation procedure can be implemented as a BARG prover where the sequence of key-message pairs $\{(\mathsf{vk}_i, m_i)\}_i$ can be used as the instance, and their corresponding signatures $\{\sigma_i\}_i$ as the witnesses. We show that this core idea can be naturally extended to the local verifiability model. We show that by replacing the underlying BARG with a mutable BARG for identity mutations, the resulting aggregate signature scheme also satisfies desired local verifiability property.

At a high level, the main observation is that one could use the mutation algorithm to create a local opening for an aggregate signature since an aggregate signature is simply a batch proof. Now we can also show that unforgeability of this scheme follows from an appropriate notion of somewhere extractability for mutable BARGs that our constructions satisfy. Moreover, the resulting signatures satisfy a strong notion of message hiding which was not known previously. Briefly, message privacy for such aggregate signatures state that a local verifier cannot learn anything beyond the message that it is trying to verify. This directly follows from the mutation privacy of our mutable BARGs. We refer to Section 12 for more details.

2.2 Subset Mutations

The next class of mutation functions that we consider is the subset mutation class. The subset mutation class, denoted as $\mathcal{P}^{\mathsf{del}}$, is a generalization of the identity class. In a few words, it is a generalized family of 'identity' functions. Each function $P_\ell = \mathbb{I}_\ell$ (for every $\ell \in \mathbb{N}$) in this class works on a tuple of instances rather than a single instance (as in $\mathcal{P}^{\mathsf{local}}$). Basically, P_ℓ takes as input (i.e., mutates) an ℓ-sequence of index-instances-proof tuples $(j_1, X_1, \pi_1), \ldots, (j_\ell, X_\ell, \pi_\ell)$ (where $\ell \in \mathbb{N}$ denotes the arity of the function), and it computes a proof for a *subset* of instances corresponding to indices $j_1, \ldots, j_\ell$. The associated language is also a batch language, $\mathcal{L}_{P_\ell} = \mathcal{L}^{\otimes \ell}$. That is, $(x_{1,j_1}, \ldots, x_{\ell,j_\ell}) \in \mathcal{L}_{P_\ell}$ iff $x_{i,j_i} \in \mathcal{L}$ for all i. It is straightforward to check that $\mathcal{P}^{\mathsf{local}} \subset \mathcal{P}^{\mathsf{del}}$ as $\mathcal{P}^{\mathsf{local}}$ is just the function $P_1 = \mathbb{I}_1$.

We routinely refer to mutable BARGs for subset mutations as *deletable* BARGs (de-BARGs). For an easier exposition, we again provide a specialized syntax for de-BARGs. We call the mutation algorithm to be the proof deletion algorithm Delete. It reads k instances $\{x_i\}_i$, deletion set S, and a proof π. It outputs a deleted/redacted proof π^{red}. In this case, the mutated proof verification algorithm is really defined to be the same algorithm as for non-mutated proof verification. In the main body, we discuss multiple notions of security for such de-BARGs ranging from soundness of deleted proofs, (somewhere) extractability w.r.t. deleted proofs, as well as multiple notions of deletion privacy. We want to point out that our formulation for de-BARGs slightly deviates from the subset mutation class as described above. The main difference is that, in our formulation, we consider the input tuples $(j_1, X_1, \pi_1), \ldots, (j_\ell, X_\ell, \pi_\ell)$ to have the additional property that all ℓ X_i's and π_i's are identical. This seems to more naturally capture the intuition of subset mutations, and contains the same technical ideas as what we require for a more general class of subset mutations.

Deletable BARGs via batching identity mutations. At first glance, it might appear that designing mutable BARGs for subset mutations could be more challenging than designing it for

identity mutations. Interestingly, we show this is not the case, and identity mutations is certainly a central and fundamental class of mutation operations.

Our main observation is that a locally verifiable BARG is really a deletable BARG, where the deletion operation only supports deletion of all-but-one instances. That is, if the goal is to delete all but the j^{th} instance, then we could simply run the LOpen algorithm on the batch proof π for instances $\{x_i\}_i$ with target index j to generate a local opening aux_j for x_j. Here opening aux_j along with the mutated proof π' can be viewed as a redacted proof for instance x_j. Moreover, if the local opening satisfies mutation privacy (which guarantees that the opening does not reveal any information about unopened instances), then this implies that the redacted proof (aux_j, π') satisfies deletion privacy. Here by deletion privacy, we mean that the deleted/redacted proof does not contain any non-trivial information about the deleted instances and/or witnesses.

With the above observation, our strategy for building de-BARGs is to lift the mutable BARG scheme supporting all-but-one deletion into a scheme that supports arbitrary deletion. To execute this, we use a rather simple idea. The general deletion algorithm now works in two phases– (1) it simply generates local openings for every instance that is not being deleted (i.e., $\{x_i\}_{i\notin S}$), and (2) then generates a fresh BARG proof for all these instances $\{x_i\}_{i\notin S}$ *using the individual all-but-one deleted proofs as the new witnesses*. Basically, we are recursively generating BARGs of BARGs to perform deletion. The core insight here is that the local verifiability feature enables translating a short proof π with large verification time (due to having to read entire batch of instances) into another short proof aux with small verification time (as LVfy only reads a single instance). Thus, while BARGing of BARGs could have been very inefficient (and also not necessarily privacy preserving), introducing local verifiability as an intermediate operation before BARGing a proof again solves the efficiency problem. Moreover, as long as the local openings also satisfy mutation privacy, this approach ensures full deletion privacy nearly generically. We discuss the above (and achieving more properties) in detail in Section 9.

Extending to general subset mutations. We highlight that the above approach directly extends to the more general setting of subset mutations. That is, even when the instance lists X_i's and corresponding proofs π_i's are no longer identical, the above strategy is sufficient for handling subset mutations. Although there is one subtle technical difference. When all the instance lists and proofs are the same, then our approach guarantees that the size of the mutated (deleted) proof grows only by an additive factor that can be made *independent of the original witness size*. This is because in our formulation locally opened proofs have two components– auxiliary opening information and a mutated batch proof. Now for a fixed batch proof, the actual mutated batch proof portion for different target indices are the same. Thus, they can be simply kept as part of the instance or the **NP** language instead. This optimization suggests that the locally opened proofs for a single batch proof can be batched more efficiently, but this cannot be guaranteed when X_i's and π_i's are no longer identical. It is an interesting problem to design mutable BARGs for general subset mutations with only a fixed polynomial additive proof growth. We believe that this might either need to develop new algebraic techniques for composing batch arguments, or a careful use of optimal-rate batch arguments [DGKV22, PP22].

Application 2: redactable signatures. Redactable signatures [JMSW02, SBZ01] allow a signature holder to publicly censor parts of a signed document such that the corresponding signature σ can be efficiently updated without the secret signing key, and the updated signature can still be verified given only the redacted document. These signatures have many real-world ap-

plications in privacy-preserving authentication as they can be used to sanitize digital signatures. (See [DPSS15, DKS16] for a detailed overview.)

We show that deletable BARGs provide a clean and natural framework to design redactable signatures with many interesting properties. Next, we summarize our construction for redactable signatures. In our redactable signature scheme, a signer partitions the document into equal sized chunks and signs each chunk individually. Here by a chunk we refer to the smallest portion of the message that a user might want to redact. For simplicity, one could consider breaking a long message bit-by-bit, and signing each bit individually along with its position in the message. However, the signer might break messages into larger chunks depending upong the application (e.g., partitioning it word-by-word or sentence-by-sentence).

Next, using de-BARGs, a signer can create a succinct batch proof proving knowledge of valid signatures for each chunk. In order to prevent trivial forgery attacks, the signer alsos sample a random tag and adds the same tag to each chunk before signing. This avoids trivial mix-and-match forgery attacks. We show by a simple reduction to de-BARGs and signature security that the above scheme satisfies stronger notions of unforgeability. More importantly, this also enables a simple approach to redact signatures. Whenever a user has to redact a signature (i.e., a batch proof as per our dsign), then it can run the proof deletion algorithm to delete the corresponding chunks. The resulting redacted/deleted proof is then set as the redacted signature. Any user can verify the redacted proof given just the non-redacted message chunks and their locations. Moreover, deletion privacy of de-BARGs also guarantees that the resulting scheme guarantees privacy of redacted messages.

We want to point out that in all known redactable signatures, the size of a redacted signature scales at least linearly with the length of non-redacted message. However, our redactable signatures are truly optimal since both the non-redacted and redacted signatures are of fixed size. Thus, to the best of our knowledge, this gives the first truly compact redactable signature scheme where all system parameter sizes are asymptotically optimal. We refer to Section 13 for more details.

2.3 Monotone Policy batchNP Mutations

The final class that we consider in this work are a special class of non-deterministic mutations, that we call monotone policy batchNP mutations. The mutation class is inspired by the recent work on SNARGs for monotone policy batchNP by Brakerski et al. [BBK+23]. For any **NP** language $\mathcal{L}$ and circuit family $\mathcal{C} = \{\mathcal{C}_k\}_k$, the monotone policy batchNP language $\mathcal{L}_{\mathcal{C}}^{(k)}$ is defined as:

$$\mathcal{L}_{\mathcal{C}}^{(k)} = \{(C, x_1, \ldots, x_k) : C \in \mathcal{C}_k \text{ and } C(\mathbb{I}_{x_1 \in \mathcal{L}}, \ldots, \mathbb{I}_{x_k \in \mathcal{L}}) = 1\},$$

where $\mathbb{I}$ is an indicator variable defined as $\mathbb{I}_{x \in \mathcal{L}} = 1$ iff $x \in \mathcal{L}$. [BBK+23] considered above languages where $\mathcal{C}$ contained all monotone circuits.[5]

In this work, we design mutable BARGs supporting monotone policy batchNP mutation functions. The monotone policy batchNP mutation class, denoted as $\mathcal{P}^{\mathsf{mtone}}$, is associated with a class

[5]We remark that prior works considered the language to be parameterized by a single monotone circuit C rather than a class of circuits $\mathcal{C}$. However, we find it to be cleaner to define the language w.r.t. a class of circuits rather than a single circuit. That is, we consider the circuit C to be part of the instance as well. While this is a non-trivial adjustment to the definition of batchNP languages, it turns out that this merely resorts to a syntactic change in the underlying constructions. In other words, existing constructions [BBK+23, NWW23] already satisfy the above formulation. Thus, we use the above more general notation for batchNP languages.

of monotone circuits $\mathcal{C} = \{\mathcal{C}_\ell\}_\ell$, where circuit $C \in \mathcal{C}_\ell$ is a monotone circuit that takes ℓ bits as inputs. Each function P_ℓ is associated with a monotone circuit $C \in \mathcal{C}_\ell$, and it takes as input/mutates an ℓ-sequence of index-instances-proof tuples $(j_1, X_1, \pi_1), \ldots, (j_\ell, X_\ell, \pi_\ell)$. Here the associated mutated language $\mathcal{L}_{P_\ell}$ is the monotone policy batchNP language $\mathcal{L}^{(k)}_{\mathcal{C}}$, as defined above. That is, $(x_{1,j_1}, \ldots, x_{\ell,j_\ell}) \in \mathcal{L}_{P_\ell}$ iff $(C, x_{1,j_1}, \ldots, x_{\ell,j_\ell}) \in \mathcal{L}^{(\ell)}_{\mathcal{C}}$. It is important to note that some of the proofs π_i can also be empty since the monotone circuit C might be satisfiable even when some of the input statements $\{x_{i,j_i}\}_i$ are not in the language $\mathcal{L}$. This is unlike previous mutation classes, where all the input batch proofs must be valid for the mutation operation to create a valid mutated proof.

Next, we describe our construction for mutable BARGs for aforementioned mutation class. Our construction relies on two central pieces: (a) mutable BARGs for identity mutations (i.e., locally verifiable BARGs), and (b) SNARGs for monotone policy batchNP.

Monotone policy batchNP mutations via batchNP SNARGs and lv-BARGs. We start by recalling the notion of SNARGs for monotone policy batchNP language from the recent work of Brakerski et al. [BBK+23]. In such SNARGs, the prover takes description of a monotone circuit $C \in \mathcal{C}$ along with a batch of k instance-witness pairs $\{(x_i, \omega_i)\}_i$. As long as $(C, x_1, \ldots, x_k) \in \mathcal{L}^{(k)}_{\mathcal{C}}$, the prover generates a valid succinct proof π that can be verified given the monotone circuit C and k instances $\{x_i\}_i$. Succinctness only guarantees that the proof size does not grow with the batch size k, but can grow with the length of a single witness.

Our main strategy for enabling monotone policy batchNP mutations is to combine above SNARGs with any locally verifiable BARG scheme (BARGs supporting identity mutations). This further illustrates the fundamental nature of the identity mutation class as it enables such a wide array of mutation functions via simple generic compilers. We describe the main approach next. To set up an lv-BARG system as well as a monotone policy batchNP SNARG system, and combine their individual CRS as the joint CRS for the new mutable BARG system. The regular prover and verifier algorithms for the mutable BARG system simply use the lv-BARG prover and verifier algorithms as is. Now, to mutate a sequence of k index-instances-proof tuples $\{(j_i, X_i, \pi_i)\}_i$ w.r.t. monotone circuit C, one performs a two-step approach: first, each batch proof π_i is locally opened for index j_i; second, run the batchNP SNARG prover for circuit C and instances $\{x_{i,j_i}\}_i$ using the locally opened proofs as the corresponding witnesses. The output proof is simply set to be the succinct mutated proof. Succinctness follows from succinctness of lv-BARGs and underlying SNARGs.

The soundness of the above design is not as straightforward. This is because the underlying lv-BARG scheme is only computationally sound, thus to argue soundness of the above design we need a reasonable notion of extraction for the underlying SNARGs. So far, for other mutation classes, the notion of somewhere extractability for BARGs was sufficient. However, here we need an appropriate notion of extractability for the batchNP SNARGs. A starting point would be to rely on the somewhere argument of knowledge property for such SNARGs as defined in [BBK+23], but it is unclear how to select the "necessary subset" non-adaptively to enable somewhere extraction. Fortunately, Brakerski et al. [BBK+23] also proved a much stronger notion of non-adaptive full extractability for their SNARGs. They proved their SNARG to be an argument of knowledge under the hardness of learning with errors assumption [Reg05]. By relying on full extractability, we can bypass the above technical issue and prove non-adaptive soundness of our mutable BARG system. Furthermore, we can combine this with somewhere extractability of our underlying lv-BARG system to get obtain an appropriate notion of somewhere extractability for our mutable

BARG proof system. This property will be later useful for our final application to homomorphic signatures.

Later in Section 10, we provide the above construction in full detail. Next, we overview the key ideas behind our homomorphic signature schemes based on mutable BARGs for monotone policy batchNP mutations.

Application 3: homomorphic signatures. Homomorphic signatures [AB09, BFKW09, BF11, GVW15] allow a signature holder to publicly run homomorphic computations on a signature, without the knowledge of the signing key. As discussed in many prior works [AB09, BFKW09, BF11, CF13, CFW14, GVW15, FMNP16], homomorphic signatures are very useful for numerous applications as well as for generalizing the functionality of public-key signatures.

We start by recalling a simplified syntax for homomorphic signatures, informally referred to as single-dataset fixed-length dataset model. In this model, the signature scheme supports signing of datasets of fixed size, say ℓ. To enable full generality, the signer can sign the bit value at each index location of the dataset individually. That is, the signing algorithm takes a bit b and index i as inputs, and generates a corresponding signature σ. In this model, the signer signs at most one bit for index $i \leq \ell$, and there exists a special homomorphic evaluation algorithm, Eval, that takes as input a circuit C, an-ℓ sequence of signatures σ_i for message bits m_i, and it generates an evaluated signature σ' that acts as a signature for message the evaluated $C(m_1, \ldots, m_\ell)$. The special property of the evaluated signature is that it can be verified given only the description of circuit C and the circuit output on the dataset. That is, a verifier does not need the underlying dataset $m = (m_1, \ldots, m_\ell)$. As noted in prior works [GVW15], the above formalization can be generically upgraded to homomorphic signatures to general models such as multi-dataset model.

Our main observation is that any mutable BARG system supporting monotone policy batchNP mutations can be used to design homomorphic signatures for both monotone as well as non-monotone circuits. That is, even when the mutable BARG system only support mutation of monotone circuits, we can still design homomorphic signatures to evaluate non-monotone circuits.

Homomorphically evaluating monotone circuits. As a starting point, let us consider the simpler case of evaluating *monotone* circuits with single-bit output. And, for simplicity, consider that the attacker's goal is to create a forgery σ' w.r.t. a circuit C^*, such that the circuit's output is 0 on the queried dataset, yet the verifier accepts it as a valid signature for output bit 1. We informally refer to this as 1-sided forgery. This simplifying assumption about forgery type follows without loss of generality. This is because to protect from attackers who want to claim the opposite (i.e., claim forgery for output bit 0 when circuit output is actually 1), one could repeat a homomorphic scheme with above guarantee twice (one for actual circuit and one for its complementary circuit). That is, simply consider the negated circuit $\overline{C^*}$ which outputs the complementary bit of C^*. Now if C^* is monotone, then by applying De Morgan's identities we can move the negation to the input wires and deterministically build another monotone circuit that will be applied on the negated input. Thus, a signer would sign the actual dataset bits under the first homomorphic signature scheme, and also sign its complement under the second homomorphic signature scheme. Now if it wants to claim the evaluated circuit output is 1, then it uses the first system for evaluation, else it uses the second system for evaluation.

With the above observation, the idea for designing homomorphic signatures for monotone circuits (with 1-sided forgery as describe above) is to simply start with an aggregate signature scheme from BARGs that we discussed earlier, and use the monotone policy batchNP mutation feature

for BARGs to compute the evaluated signature as a mutated batch proof. In words, the idea is to generate signatures individually (using the underlying standard signature scheme), and then aggregate them using the BARG prover's algorithm. That is, each dataset bit-value pair (i, m_i) is individually signed using a regular signing key sk. Then all these bit-by-bit signatures σ_i are aggregated in the form of a batch proof π. Now since we only want to guarantee 1-sided forgery (i.e., only want to prevent forgery for output bit being incorrectly claimed to be 1), we do not aggregate all signatures σ_i but only those where the corresponding message bit $m_i = 1$. That is, we only aggregate a signature if it can be useful in proving the evaluated circuit output to be 1. Note that since we are only considering monotone circuits here, thus only an input wire with value 1 can be useful in proving the output of the circuit is 1.

Now the idea for evaluating a monotone circuit on any given signature σ of some dataset $M = (m_1, \ldots, m_\ell)$ is to simply run the proof mutation algorithm on the signature σ (interpreted as a batch proof) where the mutation function is set to be C. To fit the notation of the mutation function, we have to set the index-instances-proof tuples $\{(j_i, X_i, \pi_i)\}_i$ appropriately. That is, each π_i is either σ or (depending upon i^{th} message bit m_i), and the index-instances are set as $j_i = i$ and $X_i = (1, \ldots, \ell)$.[6] We refer to Section 14.2 for more details. The proof of unforgeability boils down to the mutation soundness of the underlying mutable BARG as well as the signature scheme also satisfies context hiding if the BARG scheme satisfies mutation privacy. Since both of these propoerties are satisfied by our design of mutable BARGs, thus our homomorphic signatures are secure and context hiding.

Interestingly, our approach of using mutable BARGs for monotone policy $\mathsf{batchNP}$ mutations enables us to view and design far more general notions of homomorphic signatures. This is in part because our above approach does not use the full power of mutable BARGs. While it is not a focus of this work, we briefly sketch on how we could construct non-trivial generalizations of homomorphic signatures such as multi-key homomorphic signatures [FMNP16]. In homomorphic signatures, one could only evaluate a single 'fixed' dataset signed by a single user. However, one could consider a multi-signer setting where different dataset portions could be signed by asynchronously by multiple different signers. Such a multi-key setting was considered in the work of Fiore et al. [FMNP16], where they defined homomorphic signatures for this more general multi-signer model. It is straightforward to see that our aforementioned homomorphic signature construction can be naturally extended to the multi-key setting. This is because our mutable BARGs support mutation of batch proofs generated by different provers thus, by appropriately re-defining the index-instances-proof tuples, we can obtain multi-key homomorphic signatures for monotone circuits as well. We believe this to be an interesting direction, and leave further analysis for future work.

Beyond monotone circuits. While at first it might seem that designing homomorphic signature for classes beyond monotone circuits will be difficult following the above approach (unless we can design non-monotone $\mathsf{batchNP}$ SNARGs), we show this is not the case. Fortunately, we are able to extend the ideas behind the above construction for monotone circuits to be able to evaluate non-monotone circuits. Our intuition is to rely on folklore approaches to transform any non-monotone boolean circuits into monotone boolean circuits by introducing more input wires. Concretely, our approach is to use well-known and commonly-used technique of translating any non-monotone boolean formulae into a monotone boolean formulae of similar size. Please refer [GLW21, §8.2] and [GPSW06] for more details.

We know that any log-depth ($\mathbf{NC}^1$) circuit can be written as a polynomial-sized (non-monotone)

[6] In the main body, we set the values a bit differently for technical reasons. However, the intuition stays the same.

boolean formulae, which itself can be converted into a polynomial-sized monotone boolean formulae. Basically, the trick is to use De Morgan's identities to deterministically translate a non-monotone formula C with n-bit inputs into a monotone formula $C^{\neg}$ that reads $2n$-bits of inputs, but *does not contain any negation gates*. Here each input wire for formula C is encoded as two distinct input wires in formula $C^{\neg}$. Once we do this transformation, then we can just use the proof mutation algorithm as before to mutate a BARG proof (corresponding to the dataset that is signed) for *"monotonized"* circuit $C^{\neg}$. The soundness and context hiding proofs also can be naturally reduced to the security of underlying mutable BARGs and signature scheme. Later in Section 14.1, we describe this in full detail. We briefly remark that our usage of mutable BARGs as an abstraction also ensures that our homomorphic signatures satisfy many new interesting properties such as aggregatability of signatures and evaluation of aggregate signatures etc. This is an additional feature of using mutable batch arguments as a core cryptographic primitive for applications.

This concludes the overview of our main results and applications, and next we briefly discuss some other results and related work.

2.4 Auxiliary results and related work

Mutable Batch Arguments from SNARKs. As a final feasibility result, we outline a high level sketch for constructing most general notion of mutable BARGs from any succinct non-interactive arguments of knowledge (SNARKs) [Kil92, Mic94]. The idea quite simply is to use the mutation program P along with entire list of index-instances-proof tuples $\{(j_i, X_i, \pi_i)\}_i$ as the witness for showing the validity of the mutated instance $x_P = P(x_{1,j_1}, \ldots, x_{\ell,j_\ell})$ as per language $\mathcal{L}_P$. As we discussed earlier, mutable BARGs are only feasible when P and $\{(j_i, X_i, \pi_i)\}_i$ are sufficient to efficiently decide whether $x_P \in \mathcal{L}_P$ or not. If this is not feasible, then it is unclear whether such a mutable BARG can be designed without bypassing standard complexity-theoretic separations. Now, whenever P and $\{(j_i, X_i, \pi_i)\}_i$ are enough to efficiently decide $x_P \in \mathcal{L}_P$, then we can simply create a SNARK proof using this as the witness. And, any cheating prover can be caught by running the SNARK extractor, and using that to break soundness of the underlying BARG scheme. This merely suggests that under the assumption general-purpose SNARKs exist, we can design general-purpose mutable BARGs generically. As a simple corollary, this would imply that under existence of SNARKs, we get mutable BARGs for even non-monotone $\mathsf{batchNP}$ mutations. The focus of this work is to design succinct mutable proof systems without relying on idealized assumptions (such as those needed for SNARKs).

Adaptive soundness: leveraging SEH is enough. Additionally, we observed that we can prove adaptive soundness of BARGs from the polynomial harndess of any non-adaptively-sound BARG by relying on sub-exponential hardness of somewhere extractable hash (SEH) functions. Recall that adaptive soundness states that an attacker receives crs before selecting the batch of instances $\{x_i\}_i$ and the proof π it submits as its soundness attack. It is well known [BHK17] that there are significant technical challenges to adaptive soundness from falsifiable assumptions. However, we show these barriers can be bypassed if BARG is only somewhere extractable and not *everywhere* extractable.

Our observation is that the commit-and-prove construction for designing seBARGs is already adaptively sound by using complexity leveraging on the SEH component. Very briefly, the intuition is that an adaptive attacker must either break non-adaptive soundness of the underlying BARG or index hiding of the SEH function. This is because if there is a 'bad' soundness attacker that

breaks adaptive soundness but not non-adaptive soundness, then it must be able to break index hiding (moreover, a reduction algorithm can figure this out via a brute force search). While this does not impact any of our results in this paper, we formalize this observation later in Appendix C as it may be of independent interest.

Alternate approaches to identity mutation and limitations. As we hinted earlier, we also studied alternate approaches to design locally verifiable BARGs with mutation privacy. One successful strategy is to use NIZKs as a last step; that is, take a locally opened batch proof and then create a NIZK proof of it. Since all inputs to the local verifier are short, thus the size of the NIZK proof will still be short. Moreover, the resulting mutated proof satisfies mutation privacy because of the zero-knowledge property. Concretely, given a (non-private) local opening aux and batch proof π, we could generate a NIZK proving knowledge of aux and π such that they verify membership of target instance x.

While this also satisfies mutation privacy and seems simpler than our current approach, this has multiple limitations that we briefly summarize.

1. Complementary mutations. Using puncturable PRFs, we could easily handle complementary mutations. However, it seems unclear whether that follows from NIZKs-on-top approach.

2. Inefficient proof mutation. Further, by using NIZKs-on-top, the proof mutation algorithm would run in time polynomial in the witness size. This is because a batch proof size does grow with the size of a single witness and, if this batch proof is used as a witness by a NIZK prover, the proof mutation algorithm runtime would grow polynomially with the witness size. Our approach already gives us fast proof mutation.

3. Black-box vs. non-black-box. We wanted to avoid making non-black-box use of BARGs (if at all possible). Using NIZKs as a last step would mean making non-black-box use of cryptography, both inside and outside BARGs. Our current transformations only make black-box use of BARGs, and only within BARG themselves do we make non-black-box use of other cryptographic objects. We believe this will eventually lead to more practical constructions.

4. Minimizing assumptions. Moreover, from a theoretical view point, our approach requires far simpler assumptions. Observe that in our current construction (combining all individual transformations in one bigger transformation), we only need to rely on a NIZK proof system for language $\mathcal{L} \wedge \mathsf{COM.Verify}$. That is, we need a NIZK for just a slightly bigger language than $\mathcal{L}$. Whereas NIZKs-on-top would need a NIZK proof for language defined by the BARG verifier circuit which is far more complicated. E.g., if $\mathcal{L}$ is simply **UP**, then our approach only needs the minimal assumptions of BARGs and NIZKs for **UP** (since $\mathsf{COM.Verify}$ can be specified within **UP** as well). Thus, our current approach seems to be a better approach for understanding minimal cryptographic complexity of these concepts.

   Lastly, although currently we have a separate transformation for making our BARGs witness private, our hope is that concrete BARG constructions in the future might be directly zero-knowledge by exploiting the underlying mathematical structure in them. This would lead to concretely efficient constructions of instance private BARGs.

Related work. BARGs for **NP** follow directly from any SNARG for **NP**, and starting with the work of Micali [Mic94], there has been tremendous research advances in designing SNARGs

from a wide variety of assumptions. However, all such current constructions of SNARGs for **NP** rely on either idealized assumptions [Gro16, BSBHR18, COS20, CHM+20, Set20], or non-falsifiable assumptions [Gro10, BCCT12, DFH12, Lip13, PHGR16, GGPR13, BCI+13, BCPR14, BISW17, BCC+17], or obfuscation [SW14]. Moreover, due to Gentry-Wichs [GW11], we know that we cannot build an adaptively sound SNARG for **NP** (via a black-box reduction) to a falsifiable assumption [Nao03].

However, the landscape for BARGs for **NP** from standard assumptions is a lot more promising. The initial work by Kalai et al. [KPY19] on designing BARGs for **NP** relied on a non-standard pairing-based assumption. In recent outstanding results, [CJJ21a, CJJ21b] designed BARGs for NP with poly-logarithmic succinctness from LWE, and square-root succinctness from subexponentially-hard DDH and QR assumptions.[7] [KVZ21] provided a "BARG-to-SNARG compiler" to construct a SNARG for **P** and **NTISP**. Later on, [WW22] constructed a low-tech BARG construction from NIZK-like techniques achieving sublinear proof size from standard pairing-based assumptions, and [HJKS22] improved the proof size in [CJJ21b] to sublinear, and [CGJ+22, KLV22] constructed BARGs from sub-exponential Diffie-Hellman assumptions among other results. [DGKV22, PP22] constructed rate-1 BARGs based on standard cryptographic assumptions and provided numerous applications to such as aggregate signatures, incrementally verifiable computing and more, and [KLVW23] gave a compiler to boost weak succinctness to full succinctness via a rate-1 string OT, thereby making all prior weakly-succinct BARGs fully succinct. Furthermore, there has been tremendous progress on designing BARGs in many other settings, and for classes smaller than **NP**. References for those can be found within the above articles.

Recently, there has been great progress in understanding the connection between NIZKs and BARGs. Champion and Wu [CW23] as well as Bitansky et al. [BKP+23] designed NIZK arguments for **NP** from BARGs and dual-mode commitments (and local pseudorandom generators for [CW23]). Moreover, a concurrent work by Bradley et al. [BWW23] further reduced the assumptions to either sub-exponentially hard BARGs, or polynomially hard BARGs and public-key encryption. These results supplement our results very well, since we assume existence of NIZKs in many of our constructions. Thus, using these recent results, we can simplify our cryptographic assumptions to just seBARGs and PKE for many of our results (rather than having NIZKs as an extra assumption).

Concurrent works. In a concurrent work, Bradley et al. [BWW23] independently observed that sub-exponentially somewhere sound BARGs are also adaptively sound. Our transformation in Appendix C proves a slightly more general statement as we show that just sub-exponentially secure SEH is enough to go from somewhere sound BARGs to adaptively sound BARGs.

In two independent and concurrent works, Brodsky et al. [BCJP24] and Nassar et al. [NWW23] introduced and designed a new generalization of aggregate signatures called monotone-policy aggregate signatures. Such aggregate signatures enable aggregating signatures on a fixed message m coming from $\leq n$ different signers w.r.t. an n-bit policy predicate P. The policy predicates are general monotone circuits, and they generalize regular aggregate signatures where the policy predicate simply corresponds to 'AND' of all input bits. The functionality of monotone-policy aggregate signatures states that if a user has signatures on a message m from some subset $S \subset [n]$ of signers

[7] Coincidentally, main ideas behind our 'non-private' locally verifiable BARGs were already present in [CJJ21a], but used for a totally unrelated goal of building BARGs for **CSAT** from BARGs for index languages. It is an interesting coincidence that similar ideas are useful for local verifiability.

such that $P(S) = 1$ (i.e., the predicate is satisfied for the subset S of signers), then those can be aggregated into a short signature. And, this short signature can be verified given just P, and not the entire set of input signatures corresponding to set S. This captures the essence of aggregate signatures while giving more flexibility in terms of predicates satisfied over the group of signers. These works also relied on SNARGs for monotone-policy batchNP [BBK+23, NWW23].

In this work, we study and design homomorphic signatures which seem incomparable to monotone-policy aggregate signatures. Recall in homomorphic signatures, one perform computation on a signed message; whereas in monotone-policy aggregate signatures, the input message m is not evaluated but aggregated. Thus, on a technical level, there does not seem an immediate overlap, except the common use of monotone-policy batchNP SNARGs.

Lastly, we remark that Nassar et al. [NWW23] observed that by using 'all-but-one' (ABO) signatures [GVW19], one could rely on regular soundness of SNARGs rather than extractability notions. We believe similar ideas could be useful for generalizing the underlying assumptions for our construction of homomorphic signatures. Since Goyal et al. [GVW19] already designed ABO signatures from a wide variety of standard cryptographic assumptions, thus the above approach could possibly lead to new constructions of homomorphic signatures from more general cryptographic assumptions. We leave further exploration for future work.

3 Preliminaries

Notation. We will let PPT denote probabilistic polynomial-time. We denote the set of all positive integers up to n as $[n] := \{1, \ldots, n\}$. Also, we use $[0, n]$ to denote the set of all non-negative integers up to n, i.e. $[0, n] := \{0\} \cup [n]$.

Throughout this paper, unless specified, all polynomials we consider are positive polynomials. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of n components, where each component is drawn independently from the distribution $\mathcal{D}$.

3.1 Non-interactive Batch Arguments

Syntax. A non-interactive batch argument (BARG) scheme BARG for language $\mathcal{L}$ consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, 1^k, 1^n) \to \mathsf{crs}$. The setup algorithm takes as input the security parameter λ, number of instances k, instance length n, and outputs a crs crs.

$\mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[k]}) \to \pi$. The prover algorithm takes as input a crs and a sequence of k instance-witness pairs (x_i, ω_i) for $i \in [k]$. It outputs a proof π.

$\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i\in[k]}, \pi) \to 0/1$. The verification algorithm takes as input a crs, a sequence of k instances x_i for $i \in [k]$, and a proof π. It outputs a bit to signal whether the proof is valid or not.

In this work, we rely on somewhere extractable BARGs (seBARGs) for language $\mathcal{L}$ which are defined as above, except the setup algorithm also takes a special index as an input. And, there exists an additional algorithm called Extract that extracts an accepting witness for the special index

from any accepting batched proof. Below we provide the updated setup algorithm syntax along with the extraction algorithm.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$. The setup takes an index $i^* \in [k]$ as an additional input, and outputs a trapdoor td as well.

$\mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi) \to \omega$. The extraction algorithm takes as input the trapdoor td, k instances $\{x_i\}_i$, proof π, and outputs an extracted witness ω.

Correctness and succinctness. An seBARG is said to be correct and succinct if for every $\lambda, k, n \in \mathbb{N}$, index $i^* \in [k]$, setup parameters $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*)$, any k instances $x_1, \ldots, x_k \in \mathcal{L} \cap \{0,1\}^n$ and their corresponding witnesses ω_i for $i \in [k]$, and every proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_i)$, the following holds:

Completeness. $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$.

Extraction correctness. $\mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi) = \omega_{i^*}$.

Succinctness. $|\pi| \le \mathsf{poly}(\lambda, \log k, n, m)$. That is, the size of the batched proof is bounded by a fixed polynomial in λ, n, m, and $\log k$ where m is length of one witness.

Soundness. A BARG scheme is said to be sound if an attacker can not create a valid proof where one of the k instances being batch-proved do not belong to the language $\mathcal{L}$. For seBARGs, this can be indirectly captured by the following two properties.

Definition 3.1 (index hiding)**.** A somewhere extractable batch argument scheme seBARG satisfies index hiding if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{ll} \mathcal{A}(\mathsf{crs}) = b & (1^k, 1^n, i_0^*, i_1^*) \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ \wedge\ i_0^*, i_1^* \in [k] & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i_b^*) \end{array} : \right] \le \frac{1}{2} + \mathsf{negl}(\lambda).$$

Definition 3.2 (somewhere argument of knowledge)**.** A somewhere extractable batch argument scheme seBARG is a somewhere argument of knowledge if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1 \wedge\ i^* \in [k] \\ \wedge\ \omega^* \text{ is not a valid witness for } x_{i^*} \in \mathcal{L} \end{array} : \begin{array}{l} (1^k, 1^n, i^*) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \\ (\{x_i\}_{i \in [k]}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \omega^* \leftarrow \mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi) \end{array} \right] \le \mathsf{negl}(\lambda).$$

It was noted in prior works [DGKV22, Remark 3.3] that somewhere argument of knowledge property implies semi-adaptive soundness. For completeness, we provide the regular soundness definitions (adaptive and semi-adaptive variants) later in Appendix A.1.

3.2 Somewhere Extractable Hash

In this section, we define the notion of somewhere extractable hash schemes, which is essentially the same as a somewhere statistically binding hash function [HW15, OPWW15] except there is an explicit extraction algorithm.

Syntax. A somewhere extractable hash (SEH) scheme SEH consists of the following polynomial time algorithms:

$\mathsf{Setup}(1^\lambda, N, I) \to (\mathsf{hk}, \mathsf{td})$. The setup algorithm takes as input a security parameters λ, the input length N, and a subset $I \subset \mathbb{N}$. It outputs a hash key hk along with a trapdoor td.

$\mathsf{H}(\mathsf{hk}, x) \to h$. The hash function takes as input a hash key hk, an input $x \in \{0,1\}^N$, and outputs a hash value h.

$\mathsf{Open}(\mathsf{hk}, x, S) \to \pi$. The opening algorithm takes as input a hash key hk, an input $x \in \{0,1\}^N$, a set $S \subset [N]$, and outputs a proof π.

$\mathsf{Verify}(\mathsf{hk}, h, S, (b_j)_{j\in S}, \pi) \to 0/1$. The verification algorithm takes as input a hash key hk, a hash value h, a set $S \subset [N]$, a sequence of bits b_j for $j \in S$, a proof π, and outputs 0 or 1.

$\mathsf{Extract}(\mathsf{td}, h, S) \to (b_j)_j$. The extraction algorithm takes as input the trapdoor td, a hash value h, a set $S \subset [N]$, and outputs a sequence of extracted bits $(b_j)_{j\in S}$.

Correctness and compactness. A somewhere extractable hash scheme is said to be correct and compact if for every $\lambda, N \in \mathbb{N}$, every subset $I \subset [N]$, hash key and trapdoor $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, N, I)$, input $x \in \{0,1\}^N$, hash $h = \mathsf{H}(\mathsf{hk}, x)$, the following holds:

- **Correctness of opening.** For every set $S \subset [N]$, $\mathsf{Verify}(\mathsf{hk}, h, S, (x[j])_{j\in S}, \pi) = 1$, where $x[j]$ denotes the j-th bit of x and opening $\pi = \mathsf{Open}(\mathsf{hk}, x, S)$.
- **Correctness of extraction.** For every set $S \subseteq I$, $\mathsf{Extract}(\mathsf{td}, h, S) = (x[j])_{j\in S}$.
- **Compactness of hash key, hash value, and opening.** $|\mathsf{hk}|, |h| \le |I| \cdot \mathsf{poly}(\lambda, \log N)$, and $|\pi| \le |I| \cdot |S| \cdot \mathsf{poly}(\lambda, \log N)$.

Security. For security, we consider index hiding and somewhere statistical binding properties.

Definition 3.3 (index hiding)**.** A somewhere extractable hash scheme SEH satisfies index hiding if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{ll} \mathcal{A}(\mathsf{hk}) = b & (N, I_0, I_1) \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ \wedge\ |I_0| = |I_1| & (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, N, I_b) \end{array} : \right] \le \frac{1}{2} + \mathsf{negl}(\lambda).$$

Definition 3.4 (somewhere statistical binding)**.** A somewhere extractable hash scheme SEH satisfies somewhere statistical binding if for every stateful (computationally unbounded) attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{ll} \mathsf{Verify}(\mathsf{hk}, h, S, (b_j)_j, \pi) = 1 & (N, I) \leftarrow \mathcal{A}(1^\lambda),\ (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, N, I) \\ \wedge\ \mathsf{Extract}(\mathsf{td}, h, S) \neq (b_j)_j & (h, S, (b_j)_j, \pi) \leftarrow \mathcal{A}(\mathsf{hk}) \end{array} : \right] \le \mathsf{negl}(\lambda).$$

We note that such hash schemes have been constructed in [HW15, OPWW15] from assumptions such as LWE, DDH and DCR.

4 Mutable Batch Arguments

In this section, we introduce the concept of general mutability in batch proofs. Briefly, mutability property states that a sequence of batch proofs can be combined such that the resulting "mutated" proof is a fresh succinct proof of a "related" statement. Interestingly, this mutation operation: (1) can be performed without explicit knowledge of the original witnesses used to create the underlying batch proofs, and (2) ensures the resulting mutated proof will also not grow with original batch sizes.

Broadly, we view mutable BARGs as a BARG system which supports certain special mutation operations on top of batch proofs (without knowledge of original witnesses). One simplistic and generalized view would be to consider these as enabling "computation on *succinctly* proven data". Syntactically, we define them as follows.

A mutable BARG scheme for mutation class $\mathcal{P}$ contains the following two additional algorithms:

$\mathsf{Eval}(\mathsf{crs}, P, \{(j_i, X_i, \pi_i)\}_{i\leq\ell}) \to \widehat{\pi}$. The proof evaluation (or mutation) algorithm takes as input a crs, mutation function $P \in \mathcal{P}$, and an ℓ-length sequence of index-instances-proof tuple.

(That is, X_i consists of a sequence of instances $X_i = (x_{i,1}, \ldots)$ and π_i is a batch proof for the i^{th} sequence of instances, and j_i is an index corresponding to some instance in the i^{th} sequence.)

The algorithm outputs a mutated proof $\widehat{\pi}$. Proof $\widehat{\pi}$ is viewed as a succinct proof (SNARG) for the mutated instance $x_P = P(x_{1,j_1}, \ldots, x_{\ell,j_\ell})$ corresponding to mutated language $\mathcal{L}_P$. We use $\mathcal{P}$ to denote the class of supported mutation functions, and $\mathcal{L}_P$ denotes the **NP** language associated with each mutation function.

$\mathsf{Post\text{-}Verify}(\mathsf{crs}, P, x_P, \widehat{\pi}) \to 0/1$. The (post evaluation) verifier algorithm takes the crs, mutation function P, mutated instance x_P, and a mutated proof $\widehat{\pi}$. It outputs a single bit.

Correctness and succinctness of mutation. Informally, correctness for mutable BARGs states that the verifier accepts a mutated proof as long as the underlying batch proofs are valid. And, succinctness states that the size of a mutated proof grows only polynomially with the security parameter and maximum proofs size amongst the input batch proofs. We formalize it as follows.

Mutation correctness and succinctness. Let $\mathcal{P}$ denote the class of supported mutation functions. It states that for every $\lambda, k, n, \ell \in \mathbb{N}$, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n)$, every sequence of instances and corresponding proofs X_i and π_i for $\ell \in \mathbb{N}$ such that $\mathsf{Verify}(\mathsf{crs}, X_i, \pi_i) = 1$, and every sequence of indices $j_1, \ldots, j_\ell \in [\ell]$, and every function $P \in \mathcal{P}$, the following holds

$$\Pr\left[\mathsf{Post\text{-}Verify}(\mathsf{crs}, P, x_P, \widehat{\pi}) = 1 \ : \ \widehat{\pi} \leftarrow \mathsf{Eval}(\mathsf{crs}, P, \{(j_i, X_i, \pi_i)\}_i)\right] = 1,$$

where $x_P = P(x_{1,j_1}, \ldots, x_{\ell,j_\ell})$. Further, succinctness states that $|\widehat{\pi}| \leq \mathsf{poly}(\lambda, \max_i |\pi_i|, \log \ell)$.[8]

Multi-hop correctness and succinctness. More generally, in this work, we also consider notions of multi-hop mutation correctness and succinctness for certain specific mutation operations.

[8] Ideally, we would want that $|\widehat{\pi}| - \max_i |\pi_i| = 0$, or a fixed polynomial $\mathsf{poly}(\lambda)$. That is, the mutated proofs grow only additively by a fixed amount. However, this is not the focus of this work.

Intuitively, we define them as that a mutated proof can be used for further evaluations for appropriate mutation functions. One can very naturally extend the correctness and succinctness notions for the multi-hop variants as well.

While multi-hop mutation is not the main focus of this paper, we give constructions for multi-hop mutable batch proofs supporting deletion/redaction functions.

Soundness. For security, we consider natural post-mutation soundness properties. The intuition is that any polynomial-time attacker cannot create an accepting proof for any pair of mutation function and mutated instance (P, x_P). Formally, we define it as below.

Definition 4.1 (adaptive mutation soundness)**.** A mutable BARG scheme BARG for mutation class $\mathcal{P}$ satisfies semi-adaptive mutation soundness if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l|l} \mathsf{Post\text{-}Verify}(\mathsf{crs}, P, x_P, \widehat{\pi}) = 1 & (1^k, 1^n) \leftarrow \mathcal{A}(1^\lambda) \\ \wedge\ P \in \mathcal{P} \wedge\ x_P \notin \mathcal{L}_P & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n) \\ & (P, x_P, \widehat{\pi}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 4.2 (non-adaptive mutation soundness)**.** We say the scheme is non-adaptively mutation sound if the attacker $\mathcal{A}$ selects (both) the challenge mutation function P and instance x_P at the beginning of the security experiment.

Privacy. Additionally, we consider a novel privacy property for mutable batch proofs. The goal is to capture privacy of instances that were mutated to create a new proof. A mutated proof should not reveal any non-trivial information about the underlying batch proofs. That is, a mutated proof $\widehat{\pi}$ for any function-instance pair (P, x_P) does not reveal any non-trivial information about the input sequence of index-instances-proof tuples $\{(j_i, X_i, \pi_i)\}_i$. Below we provide a simulation based notion of mutation privacy, but one can also consider an indistinguishability based notion for mutation privacy. Later in this paper, we also formally define indistinguishability based mutation privacy notions for restricted class of mutation operations.

Definition 4.3 (mutation privacy)**.** A mutable BARG scheme $\mathsf{lv\text{-}BARG}$ for mutation class $\mathcal{P}$ satisfies mutation privacy if there exists a stateful PPT simulator $\mathtt{Sim}$ such that for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l|l} \mathcal{A}(\widehat{\pi}_b) = b \wedge & (1^k, 1^n) \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ \mathsf{Post\text{-}Verify}(\mathsf{crs}, P, x_P, \widehat{\pi}_0) = 1 & (\mathsf{crs}_0, \mathsf{td}_0) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n) \\ & \mathsf{crs}_1 \leftarrow \mathtt{Sim}(1^\lambda, 1^k, 1^n) \\ & (P, \{(j_i, X_i, \pi_i)\}_{i\in[\ell]}) \leftarrow \mathcal{A}(\mathsf{crs}_b) \\ & \widehat{\pi}_0 \leftarrow \mathsf{Eval}(\mathsf{crs}, P, \{(j_i, X_i, \pi_i)\}_i) \\ & \widehat{\pi}_1 \leftarrow \mathtt{Sim}(P, x_P := P(x_{1,j_1}, \ldots, x_{\ell,j_\ell})) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Remark 4.4 (Mutability vs. Homomorphism)**.** A mutable proof system can be viewed as a natural extension of homomorphism features to batch/succinct proofs. Thus, one might expect the techniques developed in related cryptographic systems such as homomorphic encryption/signatures/ZK-proofs to find new applications in this setting. However, we believe this might not be the case. This is because there is a major difference between these settings. One could easily consider mutation

classes such that it will be impossible to design mutable batch proofs supporting those mutations operation, but the same is not true for these other cryptographic systems.

As a simple example, consider a mutation function P that outputs a 3-SAT instance by interpreting the underlying instances as clauses. Clearly, if one could build a polynomial-time evaluation algorithm that supports such mutation operations, then this means one can efficiently solve 3-SAT arbitrarily. Therefore, unlike homomorphic encryption/signatures/non-succinct proofs, where homomorphic operations supported can be a *Turing-complete set*, mutable batch proofs supporting mutation operations that correspond to a Turing-complete set are impossible.

5 Identity Mutations = Local Proof Opening

We begin our study of mutable batch proofs by proposing a fundamental class of mutation operators. We refer to this class as *local proof opening*. The intuition behind the local proof opening operator is to mutate a batch proof π for (say) a sequence of k statements $x_1, \ldots, x_k \in \mathcal{L}$ into k equally short proofs $\pi'_1, \ldots, \pi'_j$, such that π'_j can be used to verify $x_j \in \mathcal{L}$ *in time and space independent of* k. Moreover, π'_j hides all information about all other instances which were not locally opened. Following our notation for mutable proofs from previous section, we define the 'local proof opening' mutation class $\mathcal{P}^{\mathsf{local}}$ as follows.

The mutation class $\mathcal{P}^{\mathsf{local}}$ contains only the 'identity' function $P = \mathbb{1}$. Further, it mutates only a single index-instances-proof tuple (j, X, π) (thus $\ell = 1$), and the associated language is the same language, $\mathcal{L}_P = \mathcal{L}$. Hence, for $\mathcal{P}^{\mathsf{local}}$, the mutated proof $\widehat{\pi}$ is regarded as a proof/pseudo-witness for x_j (the j^{th} instance in list X).

5.1 Specializing syntax and definition

In this work, we refer to mutable batch proofs for *identity mutation functions* as "locally verifiable BARGs" (lv-BARGs). Such lv-BARGs enable faster verification, in time independent of the batch size. These are going to be a central ingredient in the rest of our paper, thus we provide a specialized set of evaluation and verification algorithms tailored towards these mutation functions for ease of notation. Formally, an lv-BARG scheme is associated with the following additional algorithms–

$\mathsf{LOpen}(\mathsf{crs}, \{x_i\}_i, j, \pi) \to (\mathsf{aux}_j, \pi')$. This takes as input crs, k instances $\{x_i\}_i$, target index $j \in [k]$, and proof π. It outputs opening information aux_j corresponding to instance x_j, and a proof π'.

$\mathsf{LVfy}(\mathsf{crs}, x, j, \pi, \mathsf{aux}) \to 0/1$. This takes as input crs, instance x, index j, proof π, and opening information aux. It outputs one bit to denote acceptance/rejection.

Remark 5.1 (proof-independent openings)**.** We split up the proof π' and opening information aux into different components to distinguish the setting when the local opening can be computed independent of the the original batch proof. We call such lv-BARGs to have proof-independent local openings. That is, syntactically we have $\mathsf{LOpen}(\mathsf{crs}, \{x_i\}_i, j) \to \mathsf{aux}_j$. Thus, LOpen is oblivious to the batched proof, and the same local opening can be used for multiple independently computed batch proofs for the same group of instances. We do not consider this a core requirement for local verifiability, however it might be of independent interest.

The notion of correctness, succinctness, soundness, and privacy for lv-BARGs can be appropriately obtained by specializing the appropriate properties and definitions considered for general mutable batch proofs. Below we provide the specialized definitions (and some strengthenings) formally.

5.1.1 Correctness, succinctness, and security

Correctness and succinctness of local opening and verification. Informally, correctness for a locally verifiable BARG (lv-BARG) states that, given an accepting batch proof for a set of instances, the local opening algorithm generates a pair of batch proof and auxiliary information for each instance that can be efficiently verified by the local verifier. And, succinctness states that the size of the auxiliary opening as well as updated batch proof is poly-logarithmic in k. We formalize it as follows.

Local correctness. For every $\lambda, k, n \in \mathbb{N}$, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n)$, any k instances $\{x_i\}_i \in \mathcal{L} \cap \{0,1\}^n$ with corresponding witnesses ω_i, and every proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_i)$, we have that for every $j \in [k]$

$$\mathsf{LVfy}(\mathsf{crs}, x_j, j, \pi'_j, \mathsf{aux}_j) = 1, \qquad \text{where } (\mathsf{aux}_j, \pi'_j) = \mathsf{LOpen}(\mathsf{crs}, \{x_i\}_i, j, \pi).$$

Succinctness of opening. $|\mathsf{aux}_j|, |\pi'_j| \leq \mathsf{poly}(\lambda, \log k, n, m)$. That is, the size of the auxiliary information and the updated proof is bounded by a fixed polynomial in λ, n, m, and $\log k$ (where m is length of one witness).

Whenever crs is short, this implies that the running time of local verifier is independent of k (i.e., only grows poly-logarithmically in k). In situations where crs is not short, we can define an abridged version of crs such that that will be short, and local verifier only reads the abridged version.

Local soundness and extraction. In addition to the standard (extraction) soundness properties described in prior sections for regular BARGs, we propose stronger forms of local soundness properties with/out extraction guarantees. As discussed earlier, stronger soundness properties targeted to shield the local verifier are very useful for many applications (including deletion and redaction). The need for stronger soundness is highlighted by the fact that a local verifier can potentially be fooled in multiple disjoint ways– e.g., a cheating prover could create a purported batch proof that gets locally verified for an honest auxiliary opening, or it can create a malformed auxiliary information that gets verified for an honest batch proof, or it could mix-and-match these attack strategies arbitrarily. Thus, while defining soundness security for a local verifier, we consider, both, the prover and the user creating local opening as adversaries. Intuitively, we define *local* soundness property that says an adversary cannot find a valid proof π^* and auxiliary information aux^* for any invalid statement $x^* \notin \mathcal{L}$. We formalize it as follows.

Definition 5.2 (semi-adaptive local soundness with adversarial openings)**.** A locally verifiable BARG scheme $\mathsf{lv\text{-}BARG}$ satisfies semi-adaptive local soundness with adversarial openings if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l|l} \mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1 & (1^k, 1^n, i^*) \leftarrow \mathcal{A}(1^\lambda) \\ \wedge\ i^* \in [k] \wedge\ x \notin \mathcal{L} & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \\ & (x, \pi, \mathsf{aux}) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We also consider the following local extraction soundness property.

Definition 5.3 (local argument of knowledge with adversarial openings)**.** A locally verifiable BARG scheme lv-BARG satisfies local argument of knowledge with adversarial openings if there exists a local extraction algorithm LExtract such that for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l|l} \mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1 \wedge\ i^* \in [k] & (1^k, 1^n, i^*) \leftarrow \mathcal{A}(1^\lambda) \\ \wedge\ \omega^* \text{ is not a valid witness for } x \in \mathcal{L} & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \\ & (x, \pi, \mathsf{aux}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \omega^* \leftarrow \mathsf{LExtract}(\mathsf{td}, x, \pi, \mathsf{aux}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Remark 5.4 (Comparing local argument of knowledge and local soundness)**.** We want to point out that local argument of knowledge implies local soundness. If there exists a successful local soundness attacker, then that same attacker would be a successful local argument of knowledge attacker. Otherwise, the extractor will find an accepting witness, thereby invalidating the attacker being a valid soundness attacker.

Also, note that the above definitions are defined w.r.t. seBARGs since the setup algorithm takes a trapdoor index i^* as input. Alternatively, we could define local soundness for plain BARGs where the setup algorithm only produces a crs. However, later we provide a construction that achieves local extractability from any seBARG scheme. Thus, for simplicity, we define local soundness properties with a trapdoor index directly.

5.1.2 Instance Privacy

While local verifiability is a highly desirable feature on its own, the concept of a local verifier is very useful for privacy sensitive applications. As discussed earlier, the fact that, both, the batch proof π and opening aux are short (i.e., independent of batch size) implies that (at least on-average) they should hide information about other instances that were part of the initial set that was batched together. This opens the door for using lv-BARGs as a privacy preserving proof accumulator. By this we mean, that an lv-BARG system can accumulate a batch of classical **NP** proofs and store them succinctly. Moreover, the accumulated value can later be opened efficiently to obtain a verifiable yet private encoding of each individual **NP** proof without revealing anything about other proofs inside the accumulator.

Due to its many potential advantages and applications, we propose a strong worst-case instance privacy property for lv-BARGs. We view the local opening algorithm as generating an auxiliary opening aux along with a shielded batch proof π'. The intuition is that, from the local verifier's perspective, π' and aux hide everything about all the instances that were not locally opened. Formally, we capture it via the following game.

Definition 5.5 (instance privacy)**.** A locally verifiable BARG scheme lv-BARG satisfies instance privacy if there exists a PPT stateful simulator Sim such that for every stateful PPT attacker $\mathcal{A}$,

there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathcal{A}\left(\{(\mathsf{aux}_{j,b}, \pi'_{j,b})\}_{j\in S}\right) = b \\ \wedge \left(\begin{array}{l} \forall i \in [k],\ \omega_i \text{ is a valid} \\ \text{witness for } x_i \in \mathcal{L} \end{array}\right) \end{array} : \begin{array}{l} (1^k, 1^n, i^*) \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ (\mathsf{crs}_0, \mathsf{td}_0) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \\ \mathsf{crs}_1 \leftarrow \mathtt{Sim}(1^\lambda, 1^k, 1^n, i^*) \\ (S, \{(x_i, \omega_i)\}_{i\in[k]}) \leftarrow \mathcal{A}(\mathsf{crs}_b) \\ \pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_i) \\ \forall j \in S:\ (\mathsf{aux}_{j,0}, \pi'_{j,0}) \leftarrow \mathsf{LOpen}(\mathsf{crs}, \{x_i\}_i, j, \pi_0) \\ \{(\mathsf{aux}_{j,1}, \pi'_{j,1})\}_{j\in S} \leftarrow \mathtt{Sim}(S, \{x_j\}_{j\in S}, \{\omega_j\}_{j\in S}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

In this work, also we consider a stronger privacy notion called full privacy. It it nearly identical to the above instance privacy property, except the simulator $\mathtt{Sim}$ does not get witnesses for the opened instances $\{x_j\}_{j\in S}$ as well.

Definition 5.6 (full privacy)**.** We say an lv-BARG scheme satisfies full privacy if any PPT adversary does not win in the experiment described above even when $\mathtt{Sim}$ only gets $S, \{x_j\}_{j\in S}$ as inputs.

6 Building locally verifiable BARGs

In this section, we construct a locally verifiable BARG (lv-BARG) scheme for language $\mathcal{L}$. Let $\mathsf{seBARG} = (\mathsf{seB.Setup}, \mathsf{seB.Prove}, \mathsf{seB.Verify}, \mathsf{seB.Extract})$ be an seBARG for language $\hat{\mathcal{L}}$ (described Fig. 1), and $\mathsf{SEH} = (\mathsf{SEH.Setup}, \mathsf{SEH.H}, \mathsf{SEH.Open}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract})$ be a somewhere extractable hash scheme.

Set notation. Throughout this section, we will use the shorthand notation $\mathbb{I}_i$ to denote the set $\{(i-1)\cdot n + 1, \ldots, i \cdot n\}$. Here n is the length of an instance x in language $\mathcal{L}$. In words, $\mathbb{I}_i$ denotes the coordinates of the i^{th} block of size n.

Language $\hat{\mathcal{L}}$

Instance: $\hat{x} :=$ (index i, SEH key hk, hash value h_x).

Witness: $\hat{\omega} :=$ (instance x, witness ω w.r.t. $\mathcal{L}$, hash opening $\mathsf{seh}.\pi$).

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \hat{\mathcal{L}}$ if all of the following are satisfied:

- $\mathsf{seh}.\pi$ **is a valid opening for instance** x**.** Let $\mathbb{I}_i$ denote the indices corresponding to i^{th} instance block. This requires that $\mathsf{SEH.Verify}(\mathsf{hk}, h_x, \mathbb{I}_i, x, \mathsf{seh}.\pi) = 1$.
- ω **is a valid witness for** x**.** Namely, $\mathcal{R}_{\mathcal{L}}(x, \omega) = 1$ (where $\mathcal{R}_{\mathcal{L}}$ is $\mathcal{L}$'s relation).

Figure 1: Language $\hat{\mathcal{L}}$ for the seBARG.

Construction. Below we describe our locally verifiable seBARG scheme $\mathsf{lv\text{-}BARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract}, \mathsf{LOpen}, \mathsf{LVfy})$ for language $\mathcal{L}$.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$. It runs the setup algorithms for seBARG (for language $\hat{\mathcal{L}}$) and SEH schemes where the index i^* is used to *both* decide the subset of indices whose extraction is enabled by SEH scheme and the extraction index in seBARG. Namely, it generates the SEH hash key and trapdoor as

$$(\mathsf{seh.hk}, \mathsf{seh.td}) \leftarrow \mathsf{SEH.Setup}(1^\lambda, k \cdot n, \mathbb{I}_{i^*}),$$

where $\mathbb{I}_{i^*}$ is as defined above. It generates the seBARG parameters as

$$(\mathsf{sebg.crs}, \mathsf{sebg.td}) \leftarrow \mathsf{seB.Setup}(1^\lambda, 1^k, 1^{\hat{n}}, i^*),$$

where $\hat{n}$ is the length of instances in language $\hat{\mathcal{L}}$ (i.e., $\hat{n}$ is $\lceil \log k \rceil$ plus the sum of lengths of key seh.hk and hash value generated using SEH).

It outputs the lv-BARG crs and trapdoor as

$$\mathsf{crs} = (\mathsf{sebg.crs}, \mathsf{seh.hk}), \qquad \mathsf{td} = (\mathsf{sebg.td}, \mathsf{seh.td}).$$

$\mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[k]}) \to \pi$. It parses the crs as above. It first hashes the sequence of instances $(x_1, \ldots, x_k)$ using the SEH to obtain a digest/hash value h_x. Namely, it computes $h_x = \mathsf{SEH.H}(\mathsf{seh.hk}, (x_1, \ldots, x_k))$.

It then generates the opening for each instance x_i as follows

$$\forall i \in [k], \qquad \mathsf{seh}.\pi_i = \mathsf{SEH.Open}(\mathsf{seh.hk}, (x_1, \ldots, x_k), \mathbb{I}_i).$$

Next, it computes a BARG using the somewhere extractable BARG system where each instance now contains its index i, the hash key seh.hk and hash value h_ω, and the corresponding witness additionally contains the hash opening $\mathsf{seh}.\pi_i$ and the instance x_i. Formally, it generates the batch proof as

$$\mathsf{sebg}.\pi \leftarrow \mathsf{seB.Prove}(\mathsf{sebg.crs}, \{(\hat{x}_i, \hat{\omega}_i)\}_{i\in[k]}),$$

where $\hat{x}_i = (i, \mathsf{seh.hk}, h_x)$ and $\hat{\omega}_i = (x_i, \omega_i, \mathsf{seh}.\pi_i)$ for every $i \in [k]$.

Finally, it outputs $\pi = \mathsf{sebg}.\pi$.

$\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i\in[k]}, \pi) \to 0/1$. It parses the crs and proof as above. Note that crs contains the hash key seh.hk but the proof does not contain the hashed value h_x. However, since the verifier knows all the instances $\{x_i\}_i$, it recomputes h_x as $h_x = \mathsf{SEH.H}(\mathsf{seh.hk}, (x_1, \ldots, x_k))$.

The verifier then runs the seBARG verifier on $\mathsf{sebg}.\pi = \pi$ where the instances now contain i, seh.hk and h_x. Concretely, it outputs the following

$$\mathsf{seB.Verify}(\mathsf{sebg.crs}, \{\hat{x}_i = (i, \mathsf{seh.hk}, h_x)\}_{i\in[k]}, \mathsf{sebg}.\pi).$$

$\mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi) \to \omega$. It parses the trapdoor and proof as above. It first recomputes the hash value h_x as $h_x = \mathsf{SEH.H}(\mathsf{seh.hk}, (x_1, \ldots, x_k))$.

Here we assume that td contains the crs as well, thus seh.hk is also available to the extraction algorithm. It then runs the seBARG extraction algorithm to extract the witness $\hat{\omega}$ as

$$\hat{\omega} = \mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi), \qquad \text{where } \hat{x}_i = (i, \mathsf{seh.hk}, h_x)\ \forall\ i \in [k].$$

It then parses $\hat{\omega}$ as $(x, \omega, \mathsf{seh}.\pi)$ and outputs ω as the extracted witness.[9]

$\mathsf{LOpen}(\mathsf{crs}, \{x_i\}_i, j, \pi) \to (\mathsf{aux}_j, \pi')$. It generates the hash value h_x and the opening for the target instance x_j as:

$$h_x = \mathsf{SEH.H}(\mathsf{seh.hk}, (x_1, \ldots, x_k)), \qquad \mathsf{seh}.\pi = \mathsf{SEH.Open}(\mathsf{seh.hk}, (x_1, \ldots, x_k), \mathbb{I}_j).$$

It simply outputs the auxiliary information as $\mathsf{aux} = (h_x, \mathsf{seh}.\pi)$ and leaves the proof as is (i.e., $\pi' = \pi$).

$\mathsf{LVfy}(\mathsf{crs}, x, j, \pi, \mathsf{aux}) \to 0/1$. The local verifier parses the inputs as above, and performs the following two checks.

Verifying validity of instance. It runs the SEH verifier to check that

$$\mathsf{SEH.Verify}(\mathsf{seh.hk}, h_x, \mathbb{I}_j, x, \mathsf{seh}.\pi) = 1.$$

That is, x was the j^{th} instance in the instance batch hashed to create h_x.

Verifying validity of BARG. It runs the seBARG verifier to check that

$$\mathsf{seB.Verify}(\mathsf{sebg.crs}, \{\hat{x}_i\}_{i\in[k]}, \pi) = 1,$$

where $\hat{x}_i = (i, \mathsf{seh.hk}, h_x)$ for every i. Note that the local verifier only needs the hash value h_x to run the seBARG verifier.

It outputs 1 if both checks pass, otherwise it outputs 0.

6.1 Correctness and succinctness

We now show that the above locally verifiable BARG scheme satisfies completeness, correctness of extraction, local correctness, and succinctness of proof and opening if the underlying BARG and SEH schemes satisfy appropriate correctness and succinctness properties.

First, note that by the SEH correctness of opening we know that $\mathsf{SEH.Verify}(\mathsf{seh.hk}, h_x, \mathbb{I}_i, x_i, \mathsf{seh}.\pi_i) = 1$ for the opening $\mathsf{seh}.\pi_i$ as computed by the prover. Therefore, whenever ω_i is a valid witness for x_i (w.r.t. language $\mathcal{L}$), then we have that $\hat{\omega}_i$ is a valid witness for $\hat{x}_i$ (w.r.t. language $\hat{\mathcal{L}}$) for every i. Thus, by completeness of seBARG, the completeness of lv-BARG follows.

Second, note that by seBARG correctness of extraction we know that $\mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}_{i^*}$ for the digest h_x as computed by the prover and extraction algorithms. Thus, the correctness of lv-BARG extraction follows.

Third, note that the local verifier checks two conditions. It starts by checking that h_x is well formed in the auxiliary information, and then checks the validity of the BARG proof π. Observe that since the local opening and prover algorithms compute h_x identically, thus by SEH correctness of opening the first condition is always verified for honest executions. Next, observe that the seBARG verifier only needs h_x to specify the instances (which is computed correctly), thus the second condition is also verified. Hence, local correctness follows immediately.

[9] Interestingly, the extractor only requires the hash of all instances h_x for running the seBARG extractor, and it does not need to read all k instances for any other purposes. We use this property while proving local extraction security of our construction.

Finally, to argue succinctness of the lv-BARG scheme, note that it is sufficient to show that $|\hat{x}|$, $|\hat{\omega}|$, $|h_x|$, $\mathsf{seh}.\pi_j$ are all $\mathsf{poly}(\lambda, \log k, n, m)$. This is because then rest of the argument just relied on seBARG succinctness. Now observe that the above follows from compactness of SEH since $|\mathsf{seh}.\mathsf{hk}|$, $|\mathsf{seh}.\pi_i|$, $|h_\omega|$ are all individually bounded as polynomials in λ, n and m. Therefore, (proof and opening) succinctness of lv-BARG follows. Moreover, the local verifier's running time grows only poly-logarithmically with the batch size k. This follows from two facts– first, the time needed for hash verification performed by the local verifier depends only depends polynomially on $\log k$ by the efficiency of hash verification; second, the running time for an seBARG verifier also grows only as $\mathsf{poly}(\log k)$ since the underlying seBARG needed for this construction is only an 'index' seBARG [CJJ21a, KLVW23] where the verifier runtime is independent of the batch size.

Proof-independent openings. Interestingly, our construction naturally satisfies the proof-independent opening property. Note that the proof π is not used for any computation by the local verifier.

6.2 Security

Below we prove the security of our lv-BARG scheme.

Theorem 6.1. If the somewhere extractable BARG scheme seBARG satisfies index hiding and somewhere argument of knowledge (Definitions 3.1 and 3.2), and the SEH scheme SEH satisfies index hiding and somewhere statistical binding (Definitions 3.3 and 3.4), then the above scheme lv-BARG is a locally verifiable seBARG scheme satisfying index hiding, somewhere argument of knowledge, semi-adaptive local soundness with adversarial openings, and local argument of knowledge with adversarial openings (Definitions 3.1, 3.2, 5.2 and 5.3).

Proof. The proof is divided into five parts where we individually prove each security property.

Index hiding. This follows directly from the index hiding properties of the seBARG and SEH schemes. Concretely, we prove the following.

Lemma 6.2. If the seBARG scheme seBARG and SEH scheme SEH satisfy index hiding, then lv-BARG also satisfies index hiding.

Proof. This follows from a simple hybrid argument. We define an intermediate hybrid experiment in which the trapdoor index used while generating the seBARG parameters is i_1^* instead of i_1^*, while SEH parameters are still generated w.r.t. index set $\mathbb{I}_{i_0^*}$. Note that i_0^* and i_1^* are the two challenge indices chosen by the attacker. Observe that the hybrid experiment is indistinguishable from the experiment where the lv-BARG setup algorithm is run with index i_0^* by performing a simple reduction to the seBARG index hiding property. Similarly, the hybrid experiment is indistinguishable from the experiment where the lv-BARG setup algorithm is run with index i_1^* by performing a simple reduction to the SEH index hiding property. Thus, the lemma follows. □

Somewhere argument of knowledge. This follows directly from the somewhere statistical binding property of the SEH scheme and somewhere argument of knowledge of seBARG. Concretely, we prove the following.

Lemma 6.3. If the SEH scheme SEH satisfies somewhere statistical binding and the seBARG scheme seBARG satisfies somewhere argument of knowledge, then $\mathsf{lv\text{-}BARG}$ satisfies somewhere argument of knowledge.

Proof. This follows from a simple case by case reduction to the SEH somewhere statistical binding challenger and BARG somewhere argument of knowledge challenger. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks somewhere argument of knowledge of the $\mathsf{lv\text{-}BARG}$ scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases.

Type 1 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ but $\mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}$ is **not** a valid witness for $\hat{x}_{i^*}$. Note that a Type 1 attacker breaks the somewhere argument of knowledge property of the underlying seBARG scheme. This follows directly from a reduction to somewhere argument of knowledge of seBARG.

Briefly, the attacker $\mathcal{A}$ starts by outputting $(1^k, 1^n, i^*)$. The reduction algorithm $\mathcal{B}$ simply sends $(1^k, 1^{\hat{n}}, i^*)$ to the seBARG challenger, and the challenger sends the crs $\mathsf{sebg.crs}$ to $\mathcal{B}$. $\mathcal{B}$ then samples the SEH hash key $\mathsf{seh.hk}$ along with trapdoor $\mathsf{seh.td}$ as in the setup, and sends $(\mathsf{sebg.crs}, \mathsf{seh.hk})$ as the crs to $\mathcal{A}$. $\mathcal{A}$ then outputs a sequence of k instances $x_1, \ldots, x_k$ along with a proof $\pi = \mathsf{sebg}.\pi$. $\mathcal{B}$ then submits $\{\hat{x}_i\}_i$ and $\mathsf{sebg}.\pi$ as its argument of knowledge attack to the seBARG challenger (here $\hat{x}_i$ is as defined in the description of the scheme; note that this contains the hash value h_x which can be computed using $\{x_i\}_i$ and $\mathsf{seh.hk}$). Note that whenever $\mathcal{A}$ is a successful *type 1* attacker on somewhere argument of knowledge, then $\mathcal{B}$ is a successful attacker on seBARG somewhere argument of knowledge. This follows from the definition of a Type 1 attacker, and $\mathcal{A}$'s advantage in breaking somewhere argument of knowledge.

Type 2 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof $\pi = \mathsf{sebg}.\pi$ such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ *and* $\mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega} = (x, \omega, \mathsf{seh}.\pi)$ is a valid witness for $\hat{x}_{i^*}$, but ω is not a valid witness for x_{i^*}. Note that a Type 2 attacker breaks the somewhere statistical binding property of the SEH scheme. This relies on the fact that given $\hat{\omega} = (x, \omega, \mathsf{seh}.\pi)$ is a valid witness for $\hat{x}_{i^*} = (i^*, \mathsf{seh.hk}, h_x)$, then ω is a valid witness for x (w.r.t. language $\mathcal{L}$) and $\mathsf{seh}.\pi$ is a valid opening for x w.r.t. hash value h_x. That is, $\mathsf{SEH.Verify}(\mathsf{seh.hk}, h_x, \mathbb{I}_{i^*}, x, \mathsf{seh}.\pi) = 1$. Now by correctness of SEH extraction, we know that $\mathsf{SEH.Extract}(\mathsf{seh.td}, h_x, \mathbb{I}_{i^*}) = x_{i^*}$, thus by the somewhere statistical binding property of SEH, we know that it must be the case that $x = x_{i^*}$. This results in a contradiction. Thus, $\mathcal{A}$ can not be a successful Type 2 attacker even if is computationally unbounded.

Note that any successful somewhere argument of knowledge attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Semi-adaptive soundness. As noted in prior works [DGKV22, Remark 3.3], somewhere argument of knowledge property implies semi-adaptive soundness. Combining it with our somewhere argument of knowledge lemma proved above, we can prove it by combining the two lemmas. Alternatively, we can prove the following directly.

Lemma 6.4. If the SEH scheme SEH satisfies somewhere statistical binding and the seBARG scheme seBARG satisfies semi-adaptive soundness, then $\mathsf{lv\text{-}BARG}$ satisfies semi-adaptive soundness.

Proof. Note that the above proof of above lemma follows directly by combining proof of Lemma 6.3 with [DGKV22, Remark 3.3]. We could alternatively prove it directly by following the same strategy as for the proof of Lemma 6.3. The main difference will be in the way the attacker types are defined. The Type 1 attacker would now be defined as one who outputs a sequence of instances $\{x_i\}_i$ and a

proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ but $\hat{x}_{i^*} = (i^*, \mathsf{seh.hk}, h_x) \notin \hat{\mathcal{L}}$, where h_x as simply the hash of all instances combined. Note that Type 1 attacker directly break the semi-adaptive soundness of the seBARG scheme by a similar reduction to as described above.

And, by using the somewhere statistical binding property and extraction correctness of SEH, we get that a non-Type-1 attacker can not exist (with non-negligible probability) since if $\hat{x}_{i^*} \in \hat{\mathcal{L}}$ and $h_x = \mathsf{SEH.H}(\mathsf{seh.hk}, (x_1, \ldots, x_k))$, then x_{i^*} is the only valid instance at the i^{*th} location of input corresponding to h_x, therefore $x_{i^*} \in \mathcal{L}$. □

Local argument of knowledge with adversarial openings. This follows from the somewhere statistical binding property of the SEH scheme and somewhere argument of knowledge of seBARG. Concretely, we prove the following.

Lemma 6.5. If the SEH scheme SEH satisfies somewhere statistical binding and the seBARG scheme seBARG satisfies somewhere argument of knowledge, then lv-BARG satisfies local argument of knowledge with adversarial openings.

Proof. This follows from a simple case by case reduction to the SEH somewhere statistical binding challenger and BARG somewhere argument of knowledge challenger. To begin the argument, we define the local extraction algorithm LExtract as follows–

$\mathsf{LExtract}(\mathsf{td}, x, \pi, \mathsf{aux})$: It parses $\mathsf{aux} = (h_x, \mathsf{seh}.\pi)$, and sets the k instances $\hat{x}_i = (i, \mathsf{seh.hk}, h_x)$ for all $i \in [k]$. It then runs the seBARG extractor as $\hat{\omega} = \mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi)$. It parses $\hat{\omega}$ as $(x', \omega, \mathsf{seh}.\pi)$ and outputs ω as the extracted witness.

Suppose there exists a PPT attacker $\mathcal{A}$ that breaks the local argument of knowledge with adversarial opening of the lv-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases similar to the proof of Lemma 6.3.

Type 1 attacker: $\mathcal{A}$ outputs an instance x, proof π, and auxiliary information aux such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ but $\mathsf{seB.Extract}(\mathsf{sebg.td}, x, \pi, \mathsf{aux}) = \hat{\omega}^*$ is **not** a valid witness for $\hat{x}_{i^*} = (i^*, \mathsf{seh.hk}, h_x)$. Note that a Type 1 attacker breaks the somewhere argument of knowledge property of the underlying seBARG scheme. This follows directly from a reduction to somewhere argument of knowledge of seBARG.

Briefly, the attacker $\mathcal{A}$ starts by outputting $(1^k, 1^n, i^*)$. The reduction algorithm $\mathcal{B}$ simply sends $(1^k, 1^{\hat{n}}, i^*)$ to the seBARG challenger, and the challenger sends the crs sebg.crs to $\mathcal{B}$. $\mathcal{B}$ then samples the SEH hash key seh.hk along with trapdoor seh.td as in the setup, and sends $(\mathsf{sebg.crs}, \mathsf{seh.hk})$ as the crs to $\mathcal{A}$. $\mathcal{A}$ then outputs an instance x, proof π, and auxiliary information aux. $\mathcal{B}$ then submits $\{\hat{x}_i\}_i$ and π as its argument of knowledge attack to the seBARG challenger. (Here $\hat{x}_i$ is as defined in the description of the local extraction algorithm LExtract; note that aux contains the hash value h_x which is used to define each $\hat{x}_i$.) Note that whenever $\mathcal{A}$ is a successful *type 1* attacker on local argument of knowledge with adversarial opening, then $\mathcal{B}$ is a successful attacker on seBARG somewhere argument of knowledge. This follows from the definition of a Type 1 attacker, and $\mathcal{A}$'s advantage in breaking somewhere argument of knowledge. (Note that this is because since $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$, we have that $\mathsf{seB.Verify}(\mathsf{sebg.crs}, \{\hat{x}_i\}_i, \pi) = 1$.)

Type 2 attacker: $\mathcal{A}$ outputs an instance x, proof π, and auxiliary information $\mathsf{aux} = (h_x, \mathsf{seh}.\pi)$ such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ *and* $\mathsf{seB.Extract}(\mathsf{sebg.td}, x, \pi, \mathsf{aux}) = \hat{\omega}^* = (x^*, \omega^*, \mathsf{seh}.\pi^*)$ is a valid witness for $\hat{x}_{i^*} = (i^*, \mathsf{seh.hk}, h_x)$, but ω^* is not a valid witness for x. First, note that in this case we have that $\mathsf{SEH.Verify}(\mathsf{seh.hk}, h_x, \mathbb{I}_{i^*}, x, \mathsf{seh}.\pi) = 1$, $\mathsf{SEH.Verify}(\mathsf{seh.hk}, h_x, \mathbb{I}_{i^*}, x^*, \mathsf{seh}.\pi^*) = 1$,

and w^* is a valid witness for x^*. To get the desired contradiction, observe that it must be the case that $x = x^*$ as otherwise SEH does not even satisfy computational binding w.r.t. openings. This is because from above we get that both x and x^* have valid openings for $\mathbb{I}_{i^*}$ w.r.t. hash h_x. Now this directly follows from the somewhere statistical binding property. Note that the reduction algorithm can simply output one of $(x, \mathsf{seh}.\pi)$ and $(x^*, \mathsf{seh}.\pi^*)$ as the somewhere statistical binding attack as if $x \neq x^*$, then at most one of them is correctly extractable.

Note that any successful somewhere argument of knowledge attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Semi-adaptive local soundness with adversarial openings. As noted in Remark 5.4, this follows directly from Lemma 6.5. However, we prove below that this also follows from the somewhere statistical binding property of the SEH scheme and semi-adaptive soundness of seBARG. That is, the underlying BARG need not be somewhere extractable for our compiler to make it locally verifiable. Concretely, we prove the following.

Lemma 6.6. If the SEH scheme SEH satisfies somewhere statistical binding and the seBARG scheme seBARG satisfies semi-adaptive soundness, then $\mathsf{lv\text{-}BARG}$ satisfies semi-adaptive local soundness with adversarial openings.

Proof. The proof follows the same strategy as for the proof of Lemma 6.5. The main difference will be in the way the attacker types are defined. The Type 1 attacker would now be defined as one who outputs an instance x, proof π, and auxiliary information $\mathsf{aux} = (h_x, \mathsf{seh}.\pi)$ such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ but $\hat{x}_{i^*} = (i^*, \mathsf{seh.hk}, h_x) \notin \hat{\mathcal{L}}$. Now the Type 1 attacker directly breaks the semi-adaptive soundness of the seBARG scheme by a similar reduction to as described above.

And, by using the somewhere statistical binding property of SEH, we get that a non-Type-1 attacker can not exist (with non-negligible probability) since if $\hat{x}_{i^*} \in \hat{\mathcal{L}}$ and $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$, then x can be the only valid instance at the i^{*th} location of input corresponding to h_x, therefore $x_{i^*} \in \mathcal{L}$. □

This completes the proof of our main theorem. □

7 Fully Private Locally Verifiable Batch Arguments

In this section, we design a fully private lv-BARG scheme. Our idea is to split the process of hiding instances into two steps – first, we show that privacy can be reduced to a witness hiding property; second, we remark on how to make any (lv-)BARG into a witness hiding (lv-)BARG system. We start by recalling some useful preliminaries for this section.

7.1 Preliminaries

Pseudo Random Functions. A pseudorandom function (PRF) $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a keyed function[10], that takes a λ-bit key as input, and on input $x \in \{0,1\}^\lambda$, it outputs a value $y = F_K(x)$. *(Throughout the paper, we use $F_K(x)$ as a shorthand for PRF evaluation.)*

[10]For simplicity, we fix the key space, input space, and output space to be λ-bit strings.

Definition 7.1 (Pseudorandomness)**.** A PRF F is said to be secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all λ, the following holds:

$$\Pr\left[\mathcal{A}^{O_{K,b}(\cdot)}(1^\lambda) = b \ : \ K \leftarrow \{0,1\}^\lambda,\ b \leftarrow \{0,1\}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where oracle $O_{K,b}(\cdot)$ is defined as $F_K(\cdot)$ if $b = 0$, otherwise it is defined as a random function from $\{0,1\}^\lambda \to \{0,1\}^\lambda$.

Perfectly Binding Commitments. A commitment scheme COM consists of the following algorithms.

$\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{crs}$. The setup algorithm takes as input the security parameter λ, message length n, and outputs a crs crs.

$\mathsf{Com}(\mathsf{crs}, m; r) \to c$. The commit algorithm that takes as input the crs crs, message $m \in \{0,1\}^n$, and randomness $r \in \{0,1\}^\lambda$. It outputs a commitment c. *(We assume for simplicity that r is always a λ-bit string. Note that this follows w.l.o.g., and is not an extra assumption.)*

$\mathsf{Verify}(\mathsf{crs}, m, c, r) \to 0/1$. The verification algorithm takes as input the crs crs, message m, commitment c, and an opening/randomness r. It outputs either 0 or 1 to signal validity of the opening.

Correctness. A commitment scheme COM is correct if for every $\lambda, n \in \mathbb{N}$, crs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, any message $m \in \{0,1\}^n$, the following holds

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, m, c, r) = 1 \ : \ r \leftarrow \{0,1\}^\lambda,\ c = \mathsf{Com}(\mathsf{crs}, m; r)\right] = 1.$$

Perfect binding. A commitment scheme is said to be perfectly binding if no commitment can have valid openings for two different messages.

Definition 7.2 (Perfect binding)**.** A commitment scheme COM is perfectly binding if for all $\lambda, n \in \mathbb{N}$, every crs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, for every (c, m_1, r_1, m_2, r_2) such that $m_1 \neq m_2$, the following holds for at least one $i \in \{1,2\}$:

$$\Pr[\mathsf{Verify}(\mathsf{crs}, m_i, c, r_i) = 1] = 0.$$

Computational hiding. A commitment scheme is said to be computationally hiding if a commitment hides the messages when the openings are hidden.

Definition 7.3 (Hiding)**.** A commitment scheme COM is computationally hiding if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l|ll} \mathcal{A}(c) = b & & 1^n \leftarrow \mathcal{A}(1^\lambda),\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \wedge\ m_0, m_1 \in \{0,1\}^n & : & (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{crs}),\ b \leftarrow \{0,1\} \\ & & c \leftarrow \mathsf{Com}(\mathsf{crs}, m_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

7.2 Witness hiding BARGs

A BARG satisfies witness indistinguishability if an attacker cannot distinguish between batch proofs for any particular set of instances computed using two separate yet valid witnesses. Similarly, it is zero-knowledge if an attacker cannot distinguish between an honestly computed batch proof and simulated batch proof. It is well known that witness indistinguishability implies zero-knowledge for succinct proofs as well by the folklore OR trick [FLS99], whenever the goal is computational security. Thus, we directly zero-knowledge property below, and for completeness provide witness indistinguishability for BARGs in Appendix A.1.

Definition 7.4 (Zero-Knowledge)**.** A BARG scheme BARG satisfies zero-knowledge property if there exists a PPT stateful simulator $\mathcal{S}$ such that for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathcal{A}(\pi_b) = b \\ \wedge \left(\begin{array}{l} \forall i \in [k],\ \omega_i \text{ is a valid} \\ \text{witness for } x_i \in \mathcal{L} \end{array}\right) \end{array} : \begin{array}{l} (1^k, 1^n, i^*) \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ (\mathsf{crs}_0, \mathsf{td}_0) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \\ \mathsf{crs}_1 \leftarrow \mathcal{S}(1^\lambda, 1^k, 1^n, i^*) \\ \{(x_i, \omega_i)\}_{i\in[k]} \leftarrow \mathcal{A}(\mathsf{crs}_b) \\ \pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_i) \\ \pi_1 \leftarrow \mathcal{S}(\{x_i\}_i) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Clearly, a zk-BARG scheme implies a NIZK. Moreover, as shown in recent works [CW23, BKP+23], any seBARG implies a NIZK proof system, and therefore it also implies a zk-BARG scheme. Later, we show a simpler approach to designing zk-BARGs than previously considered in the literature. Moreover, we consider somewhere extractability property for such zk-BARGs.

7.3 Designing fully private lv-BARGs

Our strategy for hiding instances is to use perfectly binding commitments to hide instances, and use witness hiding BARGs to compute a batch proof using the commitments openings as the witness. The intution is that as long as the BARG scheme gives witness indistinguishability (or zero-knowledge), we can hide all unopened instances using a combination of commitment hiding property and PRF pseudorandomness.

Construction. Let F be a pseudorandom function, $\mathsf{COM} = (\mathsf{COM.Setup}, \mathsf{COM.Com}, \mathsf{COM.Verify})$ be a perfectly binding commitment scheme, and $\mathsf{lv\text{-}BARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract}, \mathsf{LOpen}, \mathsf{LVfy})$ be a locally verifiable seBARG for language $\hat{\mathcal{L}}$ (described in Fig. 2). Below we describe our private lv-BARG scheme $\mathsf{ih\text{-}BARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract}, \mathsf{LOpen}, \mathsf{LVfy})$ for language $\mathcal{L}$.

Language $\hat{\mathcal{L}}$

Instance: $\hat{x} :=$ (commitment c, commitment crs crs).

Witness: $\hat{\omega} :=$ (instance x, witness ω w.r.t. $\mathcal{L}$, opening randomness r).

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \hat{\mathcal{L}}$ if the following are satisfied:

- r **is a valid opening for** x **w.r.t.** c**.** Namely, $\mathsf{COM.Verify}(\mathsf{crs}, x, c, r) = 1$.
- ω **is a valid witness for** x**.** Namely, $\mathcal{R}_{\mathcal{L}}(x, \omega) = 1$ (where $\mathcal{R}_{\mathcal{L}}$ is $\mathcal{L}$'s relation).

Figure 2: Language $\hat{\mathcal{L}}$ for the lv-BARG.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$. It runs the setup algorithms for lv-BARG and COM schemes where the index i^* is used to set the extraction index in the lv-BARG scheme. Namely, it generates

$$\mathsf{com.crs} \leftarrow \mathsf{COM.Setup}(1^\lambda, 1^n), \quad (\mathsf{lvbg.crs}, \mathsf{lvbg.td}) \leftarrow \mathsf{lvB.Setup}(1^\lambda, 1^k, 1^{\hat{n}}, i^*),$$

where $\hat{n}$ is the length of instances in language $\hat{\mathcal{L}}$ (i.e., $\hat{n}$ is the sum of lengths of commitment and crs $\mathsf{com.crs}$). It outputs the crs and trapdoor as

$$\mathsf{crs} = (\mathsf{lvbg.crs}, \mathsf{com.crs}), \qquad \mathsf{td} = \mathsf{lvbg.td}.$$

$\mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[k]}) \to \pi$. It parses the crs as above. It first samples a random PRF key $K \leftarrow \{0,1\}^\lambda$, and encodes each instance x_i into an individual commitment c_i while using $F_K(i)$ as the randomness. Namely, it runs as follows

$$\forall i \in [k], \qquad c_i = \mathsf{COM.Com}(\mathsf{com.crs}, x_i; F_K(i)).$$

Next, it computes a proof using lv-BARG where each instance contains $\mathsf{com.crs}$ and commitment c_i, and the corresponding witness additionally contains the commitment opening $F_K(i)$ and the instance x_i. Formally, it generates the batch proof as

$$\mathsf{lvbg}.\pi \leftarrow \mathsf{lvB.Prove}(\mathsf{lvbg.crs}, \{(\hat{x}_i, \hat{\omega}_i)\}_{i\in[k]}),$$

where $\hat{x}_i = (c_i, \mathsf{com.crs})$ and $\hat{\omega}_i = (x_i, \omega_i, F_K(i))$ for every i. It outputs $\pi = (\mathsf{lvbg}.\pi, K)$.

$\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i\in[k]}, \pi) \to 0/1$. Note that neither π nor crs contain the commitments to the instances $\{x_i\}_i$ needed for running the lv-BARG verifier. But they can be recomputed by the verifier since π contains the PRF key K needed for computing the commitments c_i. Thus, the verifier starts by recomputing c_i as:

$$\forall i \in [k], \qquad c_i = \mathsf{COM.Com}(\mathsf{com.crs}, x_i; F_K(i)).$$

The verifier then runs the lv-BARG verifier on $\mathsf{lvbg}.\pi$ where the instances $\hat{x}_i$ are set as $(c_i, \mathsf{com.crs})$ for every i. Concretely, it outputs the following

$$\mathsf{lvB.Verify}(\mathsf{lvbg.crs}, \{\hat{x}_i\}_{i\in[k]}, \mathsf{lvbg}.\pi).$$

$\mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi) \to \omega$. It first recomputes the commitments c_i as above. That is,

$$\forall i \in [k], \qquad c_i = \mathsf{COM.Com}(\mathsf{com.crs}, x_i; F_K(i)).$$

It then runs the lv-BARG extraction algorithm to extract the witness $\hat{\omega}$ as

$$\hat{\omega} = \mathsf{lvB.Extract}(\mathsf{lvbg.td}, \{\hat{x}_i\}_i, \mathsf{lvbg}.\pi), \qquad \text{where } \hat{x}_i = (c_i, \mathsf{com.crs}) \; \forall \; i \in [k].$$

It then parses $\hat{\omega}$ as (x, r, ω) and outputs ω as the extracted witness.

$\mathsf{LOpen}(\mathsf{crs}, \{x_i\}_i, j, \pi) \to (\mathsf{aux}_j, \pi')$. It first recomputes the commitments c_i as above. That is,

$$\forall i \in [k], \qquad c_i = \mathsf{COM.Com}(\mathsf{com.crs}, x_i; F_K(i)).$$

It then runs the local opening algorithm for lv-BARG to generate appropriate auxiliary information for locally opening proof $\mathsf{lvbg}.\pi$. Namely, it runs

$$(\mathsf{lvbg.aux}_j, \mathsf{lvbg}.\pi') = \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, \{\hat{x}_i = (c_i, \mathsf{com.crs})\}_i, j, \mathsf{lvbg}.\pi).$$

Finally, it outputs the auxiliary opening and sanitized proof as

$$\mathsf{aux} = (\mathsf{lvbg.aux}_j, F_K(j)), \pi' = \mathsf{lvbg}.\pi'.$$

NOTE. *We want to point out that the reason we call π' as the sanitized proof is because we remove the PRF key from the original (unopened) proof. As we discuss later, this is crucial for guaranteeing instance privacy.*

$\mathsf{LVfy}(\mathsf{crs}, x, j, \pi, \mathsf{aux}) \to 0/1$. The local verifier parses the inputs as above. That is, it interprets $\mathsf{aux} = (\mathsf{lvbg.aux}, r)$ and $\pi = \mathsf{lvbg}.\pi$. It simply computes the commitment of instance x as $c = \mathsf{COM.Com}(\mathsf{com.crs}, x; r)$. It then simply runs the lv-BARG local verifier and outputs whatever it outputs. Namely, it outputs

$$\mathsf{lvB.LVfy}(\mathsf{lvbg.crs}, \hat{x} = (c, \mathsf{com.crs}), j, \mathsf{lvbg}.\pi, \mathsf{lvbg.aux}).$$

Correctness and succinctness. We now show that the above locally verifiable BARG scheme satisfies completeness, correctness of extraction, local correctness, and succinctness of proof and opening if the underlying BARG and commitment schemes satisfy appropriate correctness and succinctness properties.

First, note that by the COM correctness we know that $\mathsf{COM.Verify}(\mathsf{com.crs}, x_i, c_i, F_K(i)) = 1$ for every c_i and K as computed by the prover. Therefore, whenever ω_i is a valid witness for x_i (w.r.t. language $\mathcal{L}$), then we have that $\hat{\omega}_i$ is a valid witness for $\hat{x}_i$ (w.r.t. language $\hat{\mathcal{L}}$) for every i. Thus, by completeness of lv-BARG, the completeness of ih-BARG follows.

Second, note that by lv-BARG correctness of extraction we know that $\mathsf{lvB.Extract}(\mathsf{lvbg.td}, \{\hat{x}_i\}_i, \mathsf{lvbg}.\pi) = \hat{\omega}_{i^*}$ for the commitments $\{c_i\}_i$ as computed by the prover and extraction algorithms. Thus, the correctness of ih-BARG extraction follows.

Third, note that the local verifier simply checks that $\hat{x} = (c = \mathsf{COM.Com}(\mathsf{com.crs}, x; r), \mathsf{com.crs}) \in \hat{\mathcal{L}}$ by using $\mathsf{lvbg}.\pi$ and the auxiliary opening aux_j. By COM correctness, we have that $\mathsf{COM.Verify}($ $\mathsf{com.crs}, x, c, r) = 1$. Therefore, if $x \in \mathcal{L}$ and ω is a valid witness for it, then $\mathsf{lvB.LVfy}(\mathsf{lvbg.crs},$

$\hat{x}, j, \mathsf{lvbg}.\pi, \mathsf{lvbg}.\mathsf{aux}) = 1$ by correctness of local verification of lv-BARG. Hence, local correctness follows immediately.

Finally, to argue succinctness of the ih-BARG scheme, note that it is sufficient to show that $|\hat{x}|$, $|\hat{\omega}|$, $|\mathsf{com.crs}|$, $|K|$ are all $\mathsf{poly}(\lambda, \log k, n, m)$. This is because then rest of the argument just relied on seBARG succinctness. Now observe that the above follows from the fact that the running time of algorithms in the commitment scheme are all individually bounded as polynomials in λ and n. Therefore, (proof and opening) succinctness of ih-BARG follows. Further, the running time of a local verifier still grows poly-logarithmically with the batch size k since it runs the local verifier for the underlying lv-BARG after (re-)computing a single commitment string.

Remark 7.5 (Proof-Independent Openings)**.** We want to point out that unlike all the other compilers in this work, the instance privacy compiler described in this section does not preserve the proof-independent opening property. Moreover, it is extremely crucial for hiding the instances that the local opening algorithm has the ability to read and update/sanitize a batched proof.

7.4 Security

Below we prove the security of our ih-BARG scheme.

Theorem 7.6. If the lv-BARG scheme lv-BARG satisfies index hiding, somewhere argument of knowledge, semi-adaptive local soundness with adversarial openings, local argument of knowledge with adversarial openings, and zero-knowledge (Definitions 3.1, 3.2, 5.2, 5.3 and 7.4), and the commitment scheme COM satisfies perfect binding and computational hiding (Definitions 7.2 and 7.3), and the PRF F is a secure pseudorandom function (Definition 7.1), then the above scheme ih-BARG is an lv-BARG scheme satisfying index hiding, somewhere argument of knowledge, semi-adaptive local soundness with adversarial openings, local argument of knowledge with adversarial openings, and full privacy (Definitions 3.1, 3.2, 5.2, 5.3 and 5.6).

Proof. The proof is divided into multiple parts where we individually prove the desired properties.

Index hiding. This follows directly from the index hiding property of lv-BARG. Concretely, we prove the following.

Lemma 7.7. If the lv-BARG scheme lv-BARG satisfies index hiding, then ih-BARG also satisfies index hiding.

Proof. This directly follows from a straightforward reduction where the reduction algorithm simply generates the commitment crs com.crs on its own, and gets lvbg.crs from the lv-BARG challenger, and sends $(\mathsf{lvbg.crs}, \mathsf{com.crs})$ to the index hiding attacker on ih-BARG. The advantage of the reduction algorithm is the same as the attacker's advantage, thus the lemma follows. □

Somewhere argument of knowledge. This follows directly from the somewhere statistical binding property of the SEH scheme and somewhere argument of knowledge of seBARG. Concretely, we prove the following.

Lemma 7.8. If the commitment scheme COM satisfies (computational) binding and the lv-BARG scheme lv-BARG satisfies somewhere argument of knowledge, then ih-BARG satisfies somewhere argument of knowledge.

Proof. This follows from a simple case by case reduction to the commitment binding challenger and the lv-BARG somewhere argument of knowledge challenger. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks somewhere argument of knowledge of the ih-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases.

Type 1 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ but $\mathsf{lvB.Extract}(\mathsf{lvbg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}$ is **not** a valid witness for $\hat{x}_{i^*}$. Note that a Type 1 attacker breaks the somewhere argument of knowledge property of the underlying lv-BARG scheme. This follows directly from a reduction to somewhere argument of knowledge of seBARG.

Type 2 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof $\pi = (\mathsf{lvbg}.\pi, K)$ such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ *and* $\mathsf{lvB.Extract}(\mathsf{lvbg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega} = (x, r, \omega)$ is a valid witness for $\hat{x}_{i^*}$, but ω is not a valid witness for x_{i^*}. Note that a Type 2 attacker breaks the binding property of the commitment. This relies on the fact that given $\hat{\omega} = (x, r, \omega)$ is a valid witness for $\hat{x}_{i^*} = (c_{i^*}, \mathsf{com.crs})$ (where $c_{i^*} = \mathsf{COM.Com}(\mathsf{com.crs}, x_{i^*}; F_K(i^*))$), then ω is a valid witness for x (w.r.t. language $\mathcal{L}$) and r is a valid opening for x w.r.t. commitment c_{i^*} (that is, $\mathsf{COM.Verify}(\mathsf{com.crs}, x, c, r) = 1$). Since we know that $F_K(i^*)$ is a valid opening for x_{i^*} w.r.t. commitment c_{i^*}. Since $x \neq x_{i^*}$, thus the reduction algorithm can create valid openings for two different messages thereby breaking (computational) binding property of the commitment scheme. This results in a contradiction. Thus, $\mathcal{A}$ can not be a successful Type 2 attacker even if is computationally unbounded (assuming statistical binding).

Note that any successful somewhere argument of knowledge attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Semi-adaptive soundness. As noted previously, somewhere argument of knowledge property implies semi-adaptive soundness. Combining it with our somewhere argument of knowledge lemma proved above, we can prove it by combining the two lemmas. Alternatively, we can prove the following directly.

Lemma 7.9. If the commitment scheme COM satisfies (perfect) binding and the lv-BARG scheme lv-BARG satisfies semi-adaptive soundness, then ih-BARG satisfies semi-adaptive soundness.

Proof. The proof strategy is similar to that of Lemma 6.3, except the Type 1 attacker is defined as an attacker which creates a proof $\pi = (\mathsf{lvbg}.\pi, K)$ such that $\hat{x}_{i^*} \notin \hat{\mathcal{L}}$, where $\hat{x}_{i^*} = (c_{i^*} = \mathsf{COM.Com}(\mathsf{com.crs}, x_{i^*}; F_K(i^*)), \mathsf{com.crs})$. Now Type 1 attacker directly breaks the semi-adaptive soundness of the lv-BARG scheme by a similar reduction to as described above.

And, from the perfect binding and correctness properties of COM, we get that a non-Type-1 attacker can not exist. This is because since $c_{i^*} = \mathsf{COM.Com}(\mathsf{com.crs}, x_{i^*}; F_K(i^*))$ has $F_K(i^*)$ as a valid opening for x_{i^*} (by correctness), and by perfect binding, there can not be any other instance $x \neq x_{i^*}$ that has a valid opening w.r.t. commitment c_{i^*}. Thus, a non-Type-1 attacker will never be successful semi-adaptive soundness attacker. □

Local argument of knowledge with adversarial openings. This follows from the (computational) binding property of COM and local argument of knowledge of lv-BARG. Concretely, we prove the following.

Lemma 7.10. If the commitment scheme COM satisfies (computational) binding and the lv-BARG scheme lv-BARG satisfies local argument of knowledge with adversarial openings, then ih-BARG satisfies local argument of knowledge with adversarial openings.

Proof. The proof is very similar to the proof of Lemma 7.8. It follows from a simple case by case reduction to the commitment challenger and lv-BARG local argument of knowledge challenger. To begin the argument, suppose lv-BARG.LExtract is the local extractor for lv-BARG, then we define the local extraction algorithm LExtract for ih-BARG as follows–

LExtract$(\mathsf{td}, x, \pi, \mathsf{aux})$: It parses $\mathsf{aux} = (\mathsf{lvbg.aux}, r)$, and sets the instance $\hat{x} = (c, \mathsf{com.crs})$ where $c = \mathsf{COM.Com}(\mathsf{com.crs}, x; r)$. It then runs the lv-BARG local extractor as $\hat{\omega} = \mathsf{lv\text{-}BARG.LExtract}(\mathsf{td}, \hat{x}, \pi, \mathsf{lvbg.aux})$. It parses $\hat{\omega}$ as (x', r', ω) and outputs ω as the extracted witness.

Suppose there exists a PPT attacker $\mathcal{A}$ that breaks the local argument of knowledge with adversarial opening of the ih-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases as before.

Type 1 attacker: $\mathcal{A}$ outputs an instance x, proof π, and auxiliary information aux such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ but $\mathsf{lv\text{-}BARG.LExtract}(\mathsf{td}, \hat{x}, \pi, \mathsf{lvbg.aux}) = \hat{\omega}$ is **not** a valid witness for $\hat{x} = (c, \mathsf{com.crs})$. Note that a Type 1 attacker breaks the local argument of knowledge with adversarial openings property of the underlying lv-BARG scheme. This follows directly from a reduction to somewhere argument of knowledge of seBARG.

Type 2 attacker: $\mathcal{A}$ outputs an instance x, proof π, and auxiliary information $\mathsf{aux} = (\mathsf{lvbg.aux}, r)$ such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ *and* $\mathsf{lv\text{-}BARG.LExtract}(\mathsf{td}, \hat{x}, \pi, \mathsf{lvbg.aux}) = \hat{\omega} = (x', r', \omega)$ is a valid witness for $\hat{x} = (c, \mathsf{com.crs})$, but ω is not a valid witness for x. Note that since $\hat{\omega}$ is a valid witness for $\hat{\omega}$, thus $\mathsf{COM.Verify}(\mathsf{com.crs}, x', c, r') = 1$ where $c = \mathsf{COM.Com}(\mathsf{com.crs}, x; r)$. By completeness of the commitment scheme, we know that r is valid opening of x w.r.t. commitment c, therefore either $x = x'$, or this breaks (computational) binding property of the commitment scheme. This results in a contradiction. Thus, $\mathcal{A}$ can not be a successful Type 2 attacker even if is computationally unbounded (assuming statistical binding).

Note that any successful somewhere argument of knowledge attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Semi-adaptive local soundness with adversarial openings. As noted in Remark 5.4, this follows directly from Lemma 7.10. However, we prove below that this also follows from the perfect binding property of COM and semi-adaptive local soundness of lv-BARG. That is, the underlying BARG need not be locally extractable. Concretely, we prove the following.

Lemma 7.11. If the commitment scheme COM satisfies (perfect) binding and the lv-BARG scheme lv-BARG satisfies semi-adaptive local soundness with adversarial openings, then ih-BARG satisfies semi-adaptive local soundness with adversarial openings.

Proof. The proof follows the same strategy as for the proof of Lemma 7.10. The main difference will be in the way the attacker types are defined. The Type 1 attacker would now be defined as one who outputs an instance x, proof π, and auxiliary information $\mathsf{aux} = (\mathsf{lvbg.aux}, r)$ such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ but $\hat{x} = (\mathsf{COM.Com}(\mathsf{com.crs}, x; r), \mathsf{com.crs}) \notin \hat{\mathcal{L}}$. Now the Type 1 attacker directly breaks the semi-adaptive local soundness of the seBARG scheme by a similar reduction as described above.

And, by using the perfect binding property of COM, we get that a non-Type-1 attacker can not exist. This is because since $c = \mathsf{COM.Com}(\mathsf{com.crs}, x; r)$ has r as a valid opening for x (by correctness), and by perfect binding, there can not be any other instance $x' \neq x$ that has a valid opening w.r.t. commitment c. Thus, a non-Type-1 attacker will never be successful semi-adaptive local soundness attacker. □

Instance privacy. This follows from a combination of the PRF pseudorandomness property, commitment hiding security, and zero-knowledge property of lv-BARG. Concretely, we prove the following.

Lemma 7.12. If the pseudorandom function F is secure, amd the commitment scheme COM satisfies hiding, and the lv-BARG scheme lv-BARG satisfies (witness) zero-knowledge, then ih-BARG satisfies full privacy.

Proof. Let lv-BARG.Sim be the zero-knowledge simulator (see Definition 7.4) for the lv-BARG scheme. We use lv-BARG.Sim to design the (privacy) simulator Sim for ih-BARG described in this section. The simulator Sim works as follows:

1. On inputs $(1^\lambda, 1^k, 1^n, i^*)$, it runs the lv-BARG.Sim to sample the lvbg.crs as $\mathsf{lvbg.crs} \leftarrow \mathsf{lv\text{-}BARG.Sim}(1^\lambda, 1^k, 1^n, i^*)$. It stores the state of the simulator lv-BARG.Sim for simulating other components.

2. On input $(S, \{x_j\}_{j\in S})$, it runs as follows–

   (a) It sample k random coins $\{r_i\}_i$ independently as $r_i \leftarrow \{0,1\}^\lambda$ for $i \in [k]$.

   (b) It creates k commitments $\{c_i\}_i$ as $c_i \leftarrow \mathsf{COM.Com}(\mathsf{com.crs}, 0; r_i)$ for $i \notin S$ and $c_j = \mathsf{COM.Com}(\mathsf{com.crs}, x_j; r_j)$ for $j \in S$.

   (c) It continues running the lv-BARG simulator to create a simulated proof as $\mathsf{lvbg}.\pi \leftarrow \mathsf{lv\text{-}BARG.Sim}(\{\hat{x}_i\}_i)$ where $\hat{x}_i = (c_i, \mathsf{com.crs})$ for $i \in [k]$. (Note that the simulator stores the internal state of lv-BARG.Sim and uses that to run the simulator for generating $\mathsf{lvbg}.\pi$.)

   (d) It computes the local openings for the proof $\mathsf{lvbg}.\pi$ as

   $$\forall j \in S: \quad \mathsf{lvbg.aux}_j = \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, \{\hat{x}_i\}_i, j, \mathsf{lvbg}.\pi).$$

   (e) Finally, it outputs the (simulated) auxiliary opening information as $\mathsf{aux}_j = (\mathsf{lvbg.aux}, r_j)$ and (simulated) proof as $\pi_j = \mathsf{lvbg}.\pi$ for all $j \in S$.

The proof of security follows from a sequence of simple hybrid experiments. Let **Hybrid** 0 denote the *real* privacy experiment where the challenger runs the ih-BARG scheme honestly to generate the crs, proof and opening. Consider the following hybrid experiments.

Hybrid 1. This is identical to hybrid 0, except the challenger creates the commitments c_i using fresh independent random coins rather than by using a PRF. That is, $c_i = \mathsf{COM.Com}(\mathsf{com.crs}, x_i; r_i)$ for all i where $r_i \leftarrow \{0,1\}^\lambda$ instead of computing it as $r_i = F_K(i)$ for a random key $K \leftarrow \{0,1\}^\lambda$. Now, since there does not exist a succinct representation of the random coins, it can not run the LOpen algorithm as described in the construction, however it can simply be computed by running lvB.LOpen algorithm as is (i.e., using $\hat{x}_i = (c_i, \mathsf{com.crs})$), and setting the opening commitment to be r_j in the auxiliary information. (Note that since the attacker never receives the original/pre-redacted proof, thus the PRF key is never required in this experiment.)

Hybrid 2. This is identical to hybrid 1, except the challenger computes the lv-BARG proof $\mathsf{lvbg}.\pi$ as a simulated proof instead of a honestly computed proof using $\hat{x}_i = (c_i, \mathsf{com.crs})$ as the instances. (Note that the instances at this point still have valid witnesses.)

Hybrid 3. This is identical to hybrid 2, except the challenger computes all commitments c_j for $j \notin S$ as commitments to the all zeros string rather than committing to the actual corresponding instance x_i. That is, it creates k commitments $\{c_i\}_i$ as $c_i \leftarrow \mathsf{COM.Com}(\mathsf{com.crs}, 0; r_i)$ for $i \notin S$ and $c_j = \mathsf{COM.Com}(\mathsf{com.crs}, x_j; r_j)$ for $j \in S$.

Observe that **Hybrid** 3 is identical to the *ideal* privacy experiment where the challenger runs the ih-BARG simulator described above. Below we show via a sequence of claims that all the adjacent hybrids are computationally indistinguishable. Let $p_i^{\mathcal{A}}(\lambda)$ denote the probability that the attacker $\mathcal{A}$ outputs 0 in **Hybrid** i.

Claim 7.13. Assuming pseudorandom function F is secure, for every PPT attacker $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_0^{\mathcal{A}}(\lambda) - p_1^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Proof. This follows from a simple reduction to the PRF challenger. Suppose $\mathcal{A}$ has non-negligible advantage $\epsilon = \epsilon(\lambda)$ in distinguishing between hybrids 0 and 1. We can design a reduction algorithm $\mathcal{B}$ that can break pseudorandomness of F with advantage ϵ. The reduction algorithm $\mathcal{B}$ simply runs the ih-BARG setup as described, and sends the crs crs to $\mathcal{A}$. $\mathcal{A}$ then sends a target set S and a sequence of k instance-witness pairs $\{(x_i, \omega_i)\}_i$ to $\mathcal{B}$. $\mathcal{B}$ then queries the PRF challenger on inputs $1, 2, \ldots, k$, and receives the corresponding inputs $r_1, \ldots, r_{instno}$ from the challenger. It uses $\{r_i\}_i$ to create the commitments $c_i = \mathsf{COM.Com}(\mathsf{com.crs}, x_i; r_i)$, and sets $\hat{x}_i = (c_i, \mathsf{com.crs})$ for all i. It then runs the lv-BARG prover on instance-witness pairs $\{(\hat{x}_i, \hat{\omega}_i)\}_i$, where $\hat{\omega}_i = (x_i, r_i, \omega_i)$ for all i, to compute the proof $\mathsf{lvbg}.\pi$. Note that $\hat{\omega}_i$ is a valid instance for $\hat{x}_i$ for every i, thus by completeness of lv-BARG, $\mathcal{B}$ can create an accepting proof $\mathsf{lvbg}.\pi$. Next, $\mathcal{B}$ runs the lv-BARG local opening algorithm to compute $\mathsf{lvbg.aux}_j$ as $\mathsf{lvbg.aux}_j = \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, \{\hat{x}_i\}_i, j, \mathsf{lvbg}.\pi)$ for $j \in S$, and sends $\mathsf{aux}_j = (\mathsf{lvbg.aux}_j, r_j)$ and $\pi_j = \mathsf{lvbg}.\pi$ for $j \in S$ to the attacker $\mathcal{A}$. If $\mathcal{A}$ guesses 0, then $\mathcal{B}$ also guesses 0 to signal the PRF challenger used a PRF, otherwise it guesses 1 to signal that the PRF challenger used a truly random function.

Observe that $\mathcal{B}$ perfectly simulates the distinguishing game between hybrids 0 and 1 for the attacker $\mathcal{A}$. This is mainly due to the fact that the PRF key K is not required for simluating the experiment, and only the function evaluation on inputs $1, \ldots, k$ are required in the experiment. Thus, if $\mathcal{A}$ wins with probability ϵ, then so does $\mathcal{B}$ thereby breaking pseudorandomness of PRF F. This completes the proof of the claim. □

Claim 7.14. Assuming the lv-BARG scheme lv-BARG is (witness) zero-knowledge, for every PPT attacker $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_1^{\mathcal{A}}(\lambda) - p_2^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Proof. This follows from a simple reduction to the ZK property of lv-BARG. Suppose $\mathcal{A}$ has non-negligible advantage $\epsilon = \epsilon(\lambda)$ in distinguishing between hybrids 1 and 2. We can design a reduction algorithm $\mathcal{B}$ that can distinguish between honest and simulated proofs in scheme lv-BARG with advantage ϵ. The reduction algorithm $\mathcal{B}$ starts by generating the $\mathsf{com.crs}$ honestly, and it receives a crs $\mathsf{lvbg.crs}$ from the lv-BARG challenger. (Note that this is either simulated or honestly computed crs.) $\mathcal{B}$ then sends $\mathsf{crs} = (\mathsf{lvbg.crs}, \mathsf{com.crs})$ to $\mathcal{A}$, and $\mathcal{A}$ then outputs a target set S and a sequence of k instance-witness pairs $\{(x_i, \omega_i)\}_i$. $\mathcal{B}$ then creates fresh (honest) commitments to each of the instances as $c_i = \mathsf{COM.Com}(\mathsf{com.crs}, x_i; r_i)$ where $r_i \leftarrow \{0,1\}^\lambda$ for every $i \in [k]$. It sets $\hat{x}_i = (c_i, \mathsf{com.crs})$ and $\hat{\omega}_i = (x_i, r_i, \omega_i)$ for all i, and sends it to the lv-BARG challenger which outputs a proof $\mathsf{lvbg}.\pi$. (Again, note that the proof is either simulated or honestly computed.) Next, $\mathcal{B}$ runs the lv-BARG local opening algorithm to compute $\mathsf{lvbg.aux}_j$ as $\mathsf{lvbg.aux}_j = \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, \{\hat{x}_i\}_i, j, \mathsf{lvbg}.\pi)$ for $j \in S$, and sends $\mathsf{aux}_j = (\mathsf{lvbg.aux}_j, r_j)$ and

$\pi_j = \mathsf{lvbg}.\pi$ for $j \in S$ to the attacker $\mathcal{A}$. If $\mathcal{A}$ guesses 0, then $\mathcal{B}$ also guesses 0 to signal the lv-BARG challenger created the proof $\mathsf{lvbg}.\pi$ honestly, otherwise it guesses 1 to signal that the proof was simulated.

Observe that $\mathcal{B}$ perfectly simulates the distinguishing game between hybrids 1 and 2 for the attacker $\mathcal{A}$. Thus, if $\mathcal{A}$ wins with probability ϵ, then so does $\mathcal{B}$ thereby breaking zero-knowledge property of lv-BARG. This completes the proof of the claim. □

Claim 7.15. Assuming the commitment scheme COM is (computationally) hiding, for every PPT attacker $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $p_2^{\mathcal{A}}(\lambda) - p_3^{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Proof. This follows from a simple reduction to the commitment challenger. We are going to use a multi-message version of the hiding security game where the attacker outputs a sequence of message pairs rather than just two messages, and the challenger picks a random bit for the entire sequence and sends back a fresh commitment of the corresponding message in each pair to the attacker. Note that this follows immediately from a single message pair hiding game by a simple hybrid argument.

Suppose $\mathcal{A}$ has non-negligible advantage $\epsilon = \epsilon(\lambda)$ in distinguishing between hybrids 2 and 3. We can design a reduction algorithm $\mathcal{B}$ that can break hiding security of COM with advantage ϵ. The reduction algorithm $\mathcal{B}$ receives crs $\mathsf{com.crs}$ from the commitment challenger, and runs the lv-BARG simulator $\mathsf{lv\text{-}BARG.Sim}$ to compute $\mathsf{lvbg.crs}$ (and it stores the internal state of $\mathsf{lv\text{-}BARG.Sim}$ for further computation), and sends $\mathsf{crs} = (\mathsf{lvbg.crs}, \mathsf{com.crs})$ to $\mathcal{A}$. $\mathcal{A}$ then sends a target set S and a sequence of k instance-witness pairs $\{(x_i, \omega_i)\}_i$ to $\mathcal{B}$. $\mathcal{B}$ then sends $\{(x_i, 0)\}_{i \notin S}$ as the sequence of message pairs, and receives the corresponding commitments $\{c_i\}_{i \notin S}$. For $j \in S$, it creates commitments $c_j = \mathsf{COM.Com}(\mathsf{com.crs}, x_j; r_j)$ honestly for some $r_j \leftarrow \{0,1\}^\lambda$. It sets $\hat{x}_i = (c_i, \mathsf{com.crs})$ for all i, and runs the lv-BARG simulator to compute the simulated proof as $\mathsf{lvbg}.\pi \leftarrow \mathsf{lv\text{-}BARG.Sim}(\{\hat{x}_i\}_i)$. Next, $\mathcal{B}$ runs the lv-BARG local opening algorithm to compute $\mathsf{lvbg.aux}_j$ as $\mathsf{lvbg.aux}_j = \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, \{\hat{x}_i\}_i, j, \mathsf{lvbg}.\pi)$, and sends $\mathsf{aux}_j = (\mathsf{lvbg.aux}_j, r_j)$ and $\pi_j = \mathsf{lvbg}.\pi$ for $j \in S$ to the attacker $\mathcal{A}$. If $\mathcal{A}$ guesses 0, then $\mathcal{B}$ also guesses 0 to signal the commitment challenger committed the left messages, otherwise it guesses 1 to signal that the challenger committed the right messages.

Observe that $\mathcal{B}$ perfectly simulates the distinguishing game between hybrids 2 and 3 for the attacker $\mathcal{A}$. Thus, if $\mathcal{A}$ wins with probability ϵ, then so does $\mathcal{B}$ thereby breaking hiding of commitment COM. This completes the proof of the claim. □

Combining above claims, the lemma follows. □

This completes the proof of our main theorem. □

7.5 Hiding witnesses within BARGs

Next, we provide a general approach to hide witnesses in any BARG scheme. The naive approach is to compute a NIZK proof over a batch proof. That is, take a batch proof, and use it as a witness for NIZK/WI proof system to make it zero-knowledge/witness-indistinguishable. However, as noted in [CW23], this does not work as it takes away the succinctness property of the underlying BARG scheme. This is due to the fact that the instance used in the NIZK/WI proof system will contain all k instances. To that end, Champion and Wu [CW23], proposed a work-around. Their approach was to rely on the split-state verification property of BARGs, where the verification procedure can be split into an inefficient offline and efficient online phase. They suggested to use a NIZK

system only for hiding online portion of the verification protocol, thereby upgrading a BARG with split-state verification to zero-knowledge BARG with split-state verification.

In this work, we suggest a much simpler and more general approach for turning a BARG into zero-knowledge (or witness indistinguishable). Our idea is to, first, create a NIZK/WI proof π_i for each witness ω_i of the underlying BARG. Next, the BARG prover uses $\{\pi_i\}_i$ as the witnesses for generating the batch proof. The ZK/WI property of the resulting BARG follows directly from the ZK/WI property of the underlying non-succinct proof system. Due to its simplicity and space limitations, we describe the transformation in detail later in Appendix D. Below we state the theorem informally. (See Theorem D.2 for a formal version.)

Theorem 7.16 (informal)**.** If a proof system Π satisfies soundness and witness indistinguishability (or zero-knowledge), then any seBARG scheme seBARG can be upgraded to be a witness indistinguishable (or zero-knowledge) proof system. Moreover, if BARG is locally verifiable, then the resulting scheme also satisfies the same properties[11].

Recall that recent works [CW23, BKP+23] show that any seBARG implies a NIZK proof system, thus the above theorem show that any seBARG scheme can be upgraded into a zk-seBARG scheme. Furthermore, in the CRS model, a proof system that satisfies computational WI/ZK are equivalent assumptions. Thus, the above approach gives us a zk-seBARG proof system which, as discussed in Section 7.3, can be turned into a instance private proof system.

8 Subset Mutations = Deleting Proofs

As a next natural expansion of the class of supported mutation functions, we consider a new model for batch proofs that we refer to as *deletable proofs*. The intuition behind deletable proofs is to enable a user holding a batch proof π for a batch of instances $X = \{x_i\}_i$ to create a *redacted* batch proof π^{red} for any subset batch Y of instances such that $Y \subseteq X$. In words, it allows a user to delete instances (and their corresponding witnesses) from an existing batch proof π. Moreover, an additional interesting goal is to achieve privacy for deleted instances and witnesses. Following our notation for mutable proofs from Section 4, we define the 'proof deletion' mutation class $\mathcal{P}^{\mathsf{del}}$ as follows.

The mutation class $\mathcal{P}^{\mathsf{del}}$ still contains the family of 'identity' functions, however the identity functions $P_\ell = \mathbb{I}$ (for every $\ell \in \mathbb{N}$) work on a tuple of instances rather than a single instance (as in $\mathcal{P}^{\mathsf{local}}$). Basically, P_ℓ takes as input (i.e., mutates) an ℓ-sequence of index-instances-proof tuples $(j_1, X_1, \pi_1), \ldots, (j_\ell, X_\ell, \pi_\ell)$ (where $\ell \in \mathbb{N}$ denotes the arity of the function). Further, the associated language is also a batch language, $\mathcal{L}_{P_\ell} = \mathcal{L}^{\otimes \ell}$. That is, $(x_1, \ldots, x_\ell) \in \mathcal{L}_{P_\ell}$ iff $x_i \in \mathcal{L}$ for all i.

In summary, for $\mathcal{P}^{\mathsf{del}}$, the mutated proof $\widehat{\pi}$ is viewed as a redacted/deleted proof for some subset of instances $x_{1,j_1}, \ldots, x_{1,j_\ell}$ (the j_1^{th} instance in list X_1, and so on).

8.1 Specializing syntax and definition

In this work, we refer to mutable batch proofs for *subset mutation functions* as "deletable BARGs" (de-BARGs). Below we provide a specialized set of evaluation and verification algorithms tailored towards such mutation functions for ease of notation. Formally, a de-BARG scheme is associated with the following additional algorithm–

[11] Technically, we need Π to have an extractor to make the final proof somewhere extractable.

$\mathsf{Delete}(\mathsf{crs}, \{x_i\}_{i\in[k]}, S \subset [k], \pi) \to \pi^{\mathsf{red}}$. The deletion algorithm is a randomized algorithm that takes as input crs, a set of k instances $\{x_i\}_i$, a deletion set $S \subset [k]$, and outputs a redacted proof π^{red}.

The redacted proof π^{red} is viewed as a batch proof for the set of instances $\{x_i\}_{i\in[k]\setminus S}$. That is, it deletes instances $\{x_i\}_{i\in S}$ (and their witnesses) from the original batch proof π.

NOTE. The deletion algorithm can be applied on an already redacted proof π^{red} as well. In that case, the set of input instances to the deletion algorithm will be less than k as well as the deletion set S will a subset of a smaller set. We refer this as multi-hop deletion property of BARGs. In terms of verification, we consider that the existing verification algorithm also works for redacted batch proofs as well.

The notion of correctness, succinctness, soundness, and privacy for de-BARGs can be appropriately obtained by specializing the appropriate properties and definitions considered for general mutable batch proofs. Below we provide the specialized definitions (and some strengthenings) formally.

8.1.1 Correctness, succinctness, and security

Correctness and succinctness of deletion. Informally, correctness for de-BARGs states that the verifier accepts a redacted batch proof as long as the verifier accepts the input batch proof to be valid. And, succinctness states that the size of a redacted batch proof is poly-logarithmic in k (and the batch size of instances after deletion). We formalize it as follows.

Deletion correctness and succinctness. It states that for every $\lambda, k, n \in \mathbb{N}$, $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n)$, any k instances $\{x_i\}_i \in \mathcal{L} \cap \{0,1\}^n$ with corresponding witnesses $\{\omega_i\}_i$, and every proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_i)$, deletion correctness says that for every $S \subset [k]$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i\notin S}, \pi^{\mathsf{red}}) = 1 \ : \ \pi^{\mathsf{red}} \leftarrow \mathsf{Delete}(\mathsf{crs}, \{x_i\}_i, S, \pi)\right] = 1.$$

And, deletion succinctness states that $|\pi^{\mathsf{red}}| \le \mathsf{poly}(\lambda, |\pi|, \log(k - |S|))$.[12]

Multi-hop correctness and succinctness. More generally, we also consider multi-hop deletion correctness and succinctness which says that given any $\ell \le k$ instances $\{x_i\}_i$ with witnesses $\{\omega_i\}_i$, and any (possibly redacted) proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$, we have that for every $S \subset [\ell]$, the aforementioned correctness and succinctness properties still hold.

Soundness and Privacy. Next, we consider post-deletion soundness property. As noted earlier, in our formalization of de-BARGs, the verification algorithm can take as input, either of, unredacted and redacted batch proofs. Therefore, soundness of de-BARGs can be defined as in Definitions A.1 and A.2. Furthermore, we can also naturally extend the argument of knowledge property Definition 3.2 for de-BARG schemes.

In addition to the above properties, we consider a new privacy property for such proof systems. The goal is to capture the following: Given a proof π for a batch of instances $\{x_i\}_i$, a redacted

[12] Ideally, we would want that $|\pi^{\mathsf{red}}| - |\pi| = 0$, or a fixed polynomial $\mathsf{poly}(\lambda)$. That is, the redacted proofs grow only additively by a fixed amount. However, this is not the focus of this work.

proof π^{red}, for any choice of deletion set S, does not reveal any information about the deleted instances $\{x_i\}_{i\in S}$ (or their corresponding witnesses). We are not trying to hide the fact that π^{red} is a redacted proof or the deletion set S that was used to compute π^{red}, but merely the instances that were deleted. This is inspired by similar notions in the context of redactable and homomorphic signatures.

Definition 8.1 (deletion privacy)**.** A de-BARG scheme $\mathsf{de\text{-}BARG}$ satisfies deletion privacy if for every stateful admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}(\pi^{\mathsf{red}}) = \beta : \begin{array}{l} (1^k, 1^n) \leftarrow \mathcal{A}(1^\lambda),\ \beta \leftarrow \{0,1\} \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n) \\ (S, \{(x_{i,b}, \omega_{i,b})\}_{i\in[k], b\in\{0,1\}}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_{i,\beta}, \omega_{i,\beta})\}_i) \\ \pi^{\mathsf{red}} \leftarrow \mathsf{Delete}(\mathsf{crs}, \{x_{i,\beta}\}_i, S, \pi) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible iff $(x_{i,0}, \omega_{i,0}) = (x_{i,1}, \omega_{i,1})$[13] for all $i \notin S$, and $\omega_{i,b}$ is a valid witness for $x_{i,b} \in \mathcal{L}$ for every i, b.

Definition 8.2 (multi-hop deletion privacy)**.** A de-BARG scheme $\mathsf{de\text{-}BARG}$ satisfies *multi-hop* deletion privacy if every PPT attacker's advantage in the above distinguishing game (described in Definition 8.3) is negligible, even when the final redacted proof (provided as the challenge) is computed via two different sequence of deletion operations as long as the final set of non-deleted instances are identical. More concretely, the attacker additionally outputs a sequence of deletion set pairs $\{(S_{i,0}, S_{i,1})\}_{i\le\ell}$ such that $\cup_i S_{i,0} = \cup_i S_{i,1}$. And, the challenger computes the final challenge proof by first computing $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_{i,\beta}, \omega_{i,\beta})\}_i)$, and then running the deletion algorithms for sets $S_{1,\beta}, \ldots, S_{\ell,\beta}$ successively.

Beyond deletion privacy. We also propose a stronger notion of deletion security where we require a redacted proof is indistinguishable from a fresh proof. This goes beyond deletion privacy. It implies that a redacted proof also hides the set S used for deletion, thereby hiding the fact whether a batch proof is a redacted proof or not. We call this as *deletion anonymity*.

Definition 8.3 (deletion anonymity)**.** A de-BARG scheme $\mathsf{de\text{-}BARG}$ satisfies deletion anonymity if for every stateful admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}(\pi_b) = b : \begin{array}{l} (1^k, 1^n) \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n) \\ (S \subset [\ell], \{(x_i, \omega_i)\}_{i\in[\ell]}) \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[\ell]}) \\ \pi_0 := \pi^{\mathsf{red}} \leftarrow \mathsf{Delete}(\mathsf{crs}, \{x_i\}_i, S, \pi) \\ \pi_1 \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[\ell]\setminus S}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible iff $\ell \le k$ and ω_i is a valid witness for $x_i \in \mathcal{L}$ for every $i \in [\ell]$.

[13] One could define a slightly stronger property where the witnesses could be different, but only $x_{i,0} = x_{i,1}$ for $i \notin S$. Note that this already implies that the resulting BARG would satisfy witness indistinguishability too.

NOTE. We would like to point out that, in the above game, we extend the prover's algorithm to take a batch of instances of size fewer than k. That is, the Prove algorithm supports variable batch sizes. This is useful to define deletion anonymity in full generality, since this captures multi-hop deletion anonymity as well as multi-hop deletion indistinguishability via standard hybrid arguments.

9 Deletable BARGs via Private Local Verifiability

Next, we construct deletable BARGs (de-BARGs) from any locally verifiable BARGs (lv-BARGs). Our deletable BARGs naturally support multi-hop deletion. Moreover, we show that if lv-BARGs satisfy instance privacy, then our de-BARG scheme satisfies multi-hop deletion privacy. To design a de-BARG scheme for language $\mathcal{L}$, we require two lv-BARG schemes $\mathsf{lv\text{-}BARG} = (\mathsf{lvB.Setup}, \mathsf{lvB.Prove}, \mathsf{lvB.Verify}, \mathsf{lvB.Extract}, \mathsf{lvB.LOpen}, \mathsf{lvB.LVfy})$ for languages $\mathcal{L}$ and $\hat{\mathcal{L}}$ (described Fig. 3).

Language $\hat{\mathcal{L}}$

Instance: $\hat{x} :=$ (instance x, CRS crs).

Witness: $\hat{\omega} :=$ (proof π w.r.t. crs, auxiliary opening aux, index j).

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \hat{\mathcal{L}}$ if (π, aux) are a valid batch proof and local opening for index j, under crs, proving $x \in \mathcal{L}$. That is, $\mathsf{lvB.LVfy}(\mathsf{crs}, x, j, \pi, \mathsf{aux}) = 1$.

Figure 3: Language $\hat{\mathcal{L}}$ for the lv-BARG.

9.1 Construction

Below we describe our single-hop de-BARG scheme $\mathsf{de\text{-}BARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Delete}, \mathsf{Verify})$ for language $\mathcal{L}$.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$. It runs the setup algorithm for $\mathsf{lv\text{-}BARG}$ (for languages $\mathcal{L}$ and $\hat{\mathcal{L}}$) with extraction index i^*. Namely, it generates

$$(\mathsf{lvbg.crs}, \mathsf{lvbg.td}) \leftarrow \mathsf{lvB.Setup}^{\mathcal{L}}(1^\lambda, 1^k, 1^n, i^*),$$
$$(\widehat{\mathsf{lvbg.crs}}, \widehat{\mathsf{lvbg.td}}) \leftarrow \mathsf{lvB.Setup}^{\hat{\mathcal{L}}}(1^\lambda, 1^k, 1^{\hat{n}}, i^*),$$

where $\hat{n}$ is the length of instances in language $\hat{\mathcal{L}}$ (i.e., $\hat{n}$ is $n + |\mathsf{lvbg.crs}|$). It outputs

$$\mathsf{crs} = (\mathsf{lvbg.crs}, \widehat{\mathsf{lvbg.crs}}), \qquad \mathsf{td} = (\mathsf{lvbg.td}, \widehat{\mathsf{lvbg.td}}).$$

$\mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[k]}) \to \pi$. Let $\mathsf{crs} = (\mathsf{lvbg.crs}, \widehat{\mathsf{lvbg.crs}})$. It generates a batch proof under $\mathsf{lvbg.crs}$ for the input instance-witness pairs as $\pi \leftarrow \mathsf{lvB.Prove}(\mathsf{lvbg.crs}, \{(x_i, \omega_i)\}_i)$.

$\mathsf{Delete}(\mathsf{crs}, \{x_i\}_i, S, \pi) \to \pi^{\mathsf{red}}$. It generates a local opening for each instance x_j such that $j \notin S$. That is, it computes

$$\forall j \in [k] \setminus S, \quad (\mathsf{aux}_j, \pi'_j) \leftarrow \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, \{x_i\}_i, j, \pi).$$

Observe that lvB.LVfy can be used to verify each such opening-proof pair. The deletion algorithm then creates a fresh batch proof for instances $\{x_j\}_{j\notin S}$[14]. Concretely, it outputs the proof as

$$\pi^{\mathsf{red}} \leftarrow \mathsf{lvB.Prove}(\widehat{\mathsf{lvbg.crs}}, \{(\hat{x}_j, \hat{\omega}_j)\}_{j\notin S})), \text{ where } \hat{x}_j = (x_j, \mathsf{lvbg.crs}), \hat{\omega}_j = (\pi'_j, \mathsf{aux}_j, j).$$

$\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i\in[\ell]}, \pi) \to 0/1$. If $\ell = k$, then the prover runs the lv-BARG verifier w.r.t. lvbg.crs since π is a proof of a full batch of instances. That is, if $\ell = k$, it outputs $\mathsf{lvB.Verify}(\mathsf{lvbg.crs}, \{x_i\}_i, \pi)$. Otherwise, it outputs $\mathsf{lvB.Verify}(\widehat{\mathsf{lvbg.crs}}, \{x_i\}_i, \pi)$.

Preserving local verifiability and somewhere extractability. Observe that the above construction is also locally verifiable since we could simply use the corresponding local opening algorithm for a unredacted batch proof π and redacted batch proof π^{red} to create relevant local openings.

$\mathsf{LOpen}(\mathsf{crs}, \{x_i\}_{i\in[\ell]}, j, \pi) \to (\mathsf{aux}_j, \pi')$. Based on the batch size, it runs the local opening algorithm w.r.t. lvbg.crs or $\widehat{\mathsf{lvbg.crs}}$. That is, if $\ell = k$, it outputs $(\mathsf{aux}_j, \pi') \leftarrow \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, \{x_i\}_i, j, \pi)$. Else, it outputs $(\mathsf{aux}_j, \pi') \leftarrow \mathsf{lvB.LOpen}(\widehat{\mathsf{lvbg.crs}}, \{x_i\}_i, j, \pi)$.

$\mathsf{LVfy}(\mathsf{crs}, x, j, \pi, \mathsf{aux}) \to 0/1$. The local verifier can simply run lvB.LVfy on both lvbg.crs and $\widehat{\mathsf{lvbg.crs}}$, and output 1 if either execution accepts the proof. (Alternatively, we could add an indicator bit in the auxiliary opening to indicate whether it is a local opening of a unredacted or a redacted proof.)

Moreover, if the lv-BARG scheme satisfies somewhere extractability as well as local extractability properties, then the above de-BARG scheme also satisfies somewhere extractability as well as local extractability. For ease of exposition, we assume that the deletion algorithm replaces each deleted instance with a dummy[15] satisfying instance at the time of computing the redacted proof π^{red}. Thus, Delete runs lvB.Prove on a batch of size k after creating local openings. This assumption makes it easier to explain somewhere and local extractability. This is due to the fact that this strategy of replacing each deleted instance with dummy (satisfying instances) maintains the invariant that every instance that does not get deleted stays in the same index position. Thus, a somewhere/local extractor for above de-BARG scheme can simply be run as follows – (1) for a unredacted proof, run the somewhere extractor directly using lvbg.td to extract the witness ω_{i^*}, (2) for a redacted proof, first run the somewhere extractor using $\widehat{\mathsf{lvbg.td}}$ to extract a tuple of proof π and auxiliary opening aux, and then run the local extractor using lvbg.td on the extracted values to extract the underlying witness ω_{i^*}.

From single-hop to multi-hop deletion. We remark that one can generically construct a multi-hop de-BARG scheme from a single-hop de-BARGs, with the restriction that the number of

[14] Note that here we are assuming that the prover algorithm can compute a batch proof for any batch size $\leq k$. This follows w.l.o.g. since a prover could either add dummy valid instances to pad the batch size to k, or the BARG prover can simply natively support varying batch sizes. We point out that all existing BARG schemes already support varying length batch sizes. Here we assume that the algorithm adds dummy instances in place of every deleted instance.

[15] A dummy instance could simply be either the first instance that is not deleted, or some global instance.

hops is constant. For building a ℓ-hop de-BARG system (for some constant $\ell > 0$), the approach is to sample a sequence of ℓ single-hop de-BARG systems such that a redacted proof generated using the i^{th} single-hop de-BARG system can be further redacted by the $(i+1)^{th}$ single-hop system. Intuitively, we can generate the local openings for the i^{th} redacted proof, and use them as witnesses for $(i+1)^{th}$ single-hop system. Now by using deletion property of the $(i+q)^{th}$ instance, we can perform another level of deletion. This is reminscient to Gentry's bootstrapping [Gen09], but extended to deletable BARG schemes. This can be slightly optimized by modifying the above construction and sampling a sequence of $\ell+1$ lv-BARG systems, and performing the above deletion bootstrapping (or, sequential deletion) slightly more efficiently. However, the above approaches only support constant number of deletion hops due to potential exponential (in number of hops) growth in the proof size. This is because the proof size after each deletion could be polynomially larger than initial proof. However, if we start with a rate-1 de-BARG scheme, then they would enable unbounded number of multi-hop deletion operations.

Correctness and succinctness. We now show that the above scheme satisfies deletion correctness and succinctness. (It already satisfies regular correctness and succinctness due to regular correctness and succinctness of the base BARG scheme for language $\mathcal{L}$.) Deletion correctness follows similarly from the local correctness of the BARG scheme corresponding to languages $\mathcal{L}$, and regular correctness property for the BARG scheme corresponding to languages $\hat{\mathcal{L}}$. Moreover, the deletion succinctness follows from opening succinctness of BARG scheme (for language $\hat{\mathcal{L}}$) and proof succinctness of BARG scheme (for language $\mathcal{L}$).

In further detail, first note that by local correctness of the base BARG scheme, we have that (aux_j, π'_j) is a valid proof for instance x_j w.r.t. language $\hat{\mathcal{L}}$ due to the local correctness property. Moreover, by succinctness of opening we have that $|\mathsf{aux}_j|, |\pi'_j|$ are succinct (i.e., grow as $\mathsf{poly}(\lambda, \log k)$). Therefore, any proof π^{red} obtained using the deletion algorithm for a set S is accepted by the verifier since $\hat{x}_j$ and $\hat{\omega}_j$ will be a vali instance-witness pair w.r.t. $\hat{\mathcal{L}}$. Further, by succinctness of the second BARG scheme, we also get that $|\pi^{\mathsf{red}}| = \mathsf{poly}(\lambda, \log(k - |S|), |\hat{\omega}_j|)$ and since $|\hat{\omega}_j|$ is the same as the size of a locally opened barg proof, thus deletion succinctness follows. The proof of multi-hop deletion also follows analogously.

Lastly, we point out that if both the BARG scheme are rate-1, then the proof growth is only additive.

9.2 Security

Below we prove the security of our scheme.

Theorem 9.1. If the lv-BARG scheme $\mathsf{lv\text{-}BARG}$ satisfies index hiding, somewhere argument of knowledge, local argument of knowledge with adversarial openings, and instance privacy (Definitions 3.1, 3.2, 5.3 and 5.5), then the above scheme is a *deletable* BARG scheme satisfying index hiding, somewhere argument of knowledge, local argument of knowledge with adversarial openings, and deletion privacy (Definitions 3.1, 3.2, 5.3 and 8.3).

Proof. The proof is divided into four parts where we individually prove the desired properties.

Index hiding. This follows directly from the index hiding properties of the two $\mathsf{lv\text{-}BARG}$ schemes. Concretely, we prove the following.

Lemma 9.2. If the BARG scheme lv-BARG satisfies index hiding, then de-BARG also satisfies index hiding.

Proof. This follows from a simple hybrid argument. We define an intermediate hybrid experiment in which the trapdoor index used while generating the lv-BARG parameters for language $\mathcal{L}$ is i_1^* instead of i_0^*, while parameters for lv-BARG for language $\hat{\mathcal{L}}$ are still generated as before. Note that i_0^* and i_1^* are the two challenge indices chosen by the attacker. Observe that the hybrid experiment is indistinguishable from the experiment where the de-BARG setup algorithm is run with index i_0^* by performing a simple reduction to the lv-BARG index hiding property. Similarly, the hybrid experiment is indistinguishable from the experiment where the de-BARG setup algorithm is run with index i_1^* by performing a simple reduction to the lv-BARG index hiding property. Thus, the lemma follows. □

Somewhere argument of knowledge. This follows from a combination of the somewhere argument of knowledge property and the local argument of knowledge with adversarial openings property of lv-BARG for languages $\hat{\mathcal{L}}$ and $\mathcal{L}$, respectively. Concretely, we prove the following.

Lemma 9.3. If the BARG scheme lv-BARG satisfies somewhere argument of knowledge and local argument of knowledge with adversarial openings, then de-BARG satisfies somewhere argument of knowledge.

Proof. This follows from a simple recursive extraction procedure for the lv-BARG. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks somewhere argument of knowledge of the de-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in three cases.

Type 1 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_{i\in[\ell]}$ and a proof π such that $\ell = k$ and $\mathsf{lvB.Verify}(\mathsf{lvbg.crs}, \{x_i\}_i, \pi) = 1$ but $\mathsf{lvB.Extract}(\mathsf{lvbg.td}, \{x_i\}_i, \pi) = \omega$ is **not** a valid witness for x_{i^*}. Note that a Type 1 attacker breaks the somewhere argument of knowledge property of the inner BARG scheme. This follows directly from a reduction to somewhere argument of knowledge of lv-BARG.

Type 2 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_{i\in[\ell]}$ and a proof π such that $\ell < k$ and $\mathsf{lvB.Verify}(\widehat{\mathsf{lvbg.crs}}, \{\hat{x}_i\}_i, \pi) = 1$ but $\mathsf{lvB.Extract}(\widehat{\mathsf{lvbg.td}}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}$ is **not** a valid witness for $\hat{x}_{i^*}$. Note that a Type 2 attacker breaks the somewhere argument of knowledge property of the outer BARG scheme. This follows directly from a reduction to somewhere argument of knowledge of lv-BARG.

Type 3 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_{i\in[\ell]}$ and a proof π such that $\ell < k$ and $\mathsf{lvB.Verify}(\widehat{\mathsf{lvbg.crs}}, \{\hat{x}_i\}_i, \pi) = 1$, and $\mathsf{lvB.Extract}(\widehat{\mathsf{lvbg.td}}, \{\hat{x}_i\}_i, \pi) = \hat{\omega} = (\pi'_j, \mathsf{aux}_j, j)$ is a valid witness for $\hat{x}_{i^*} = (x_{i^*}, \mathsf{lvbg.crs})$, but $\mathsf{lvB.LExtract}(\mathsf{lvbg.td}, x_{i^*}, j, \pi'_j, \mathsf{aux}_j) = \omega$ is **not** a valid witness for x_{i^*}. Note that a Type 3 attacker breaks the local argument of knowledge with adversarial openings property of the inner BARG scheme. This follows directly from a reduction to local argument of knowledge of lv-BARG.

We want to highlight that here we are assuming that the deletion algorithm simply replaces each deleted instance with a dummy instance. This ensures that the index/location of an instance before and after deletion in the full batch of instances stays the same. This ensures that $j = i^*$, thus extraction can be recursively performed. In case, the deletion algorithm changes the order of the instances or reduces the batch size, then we will need to update the notion of somewhere extraction for deleted proofs where an aversary must commit to not only the location of the instance in the

original batch of instances, but also its location (relative to the batch of deleted instances) after the delete operation.

Finally, note that any successful somewhere argument of knowledge attacker $\mathcal{A}$ must be either Type 1, 2, or 3. Thus, combining the above arguments, the lemma follows. □

Semi-adaptive soundness. As noted earlier, somewhere argument of knowledge property implies semi-adaptive soundness. Combining this with our somewhere argument of knowledge lemma proved above, we achieve semi-adaptive soundness as well.

Local argument of knowledge with adversarial openings. This is a natural extension of the somewhere argument of knowledge proved in Lemma 9.3. Concretely, we prove the following.

Lemma 9.4. If the BARG scheme lv-BARG satisfies local argument of knowledge with adversarial openings, then de-BARG satisfies local argument of knowledge with adversarial openings.

Proof. The proof is similar to the proof of Lemma 9.3 which is by relying on a recursive extraction strategy, except now we use a local extractor for both outer and inner BARG schemes. Previously, we used a mixture of local extractor and somewhere extractor. However, now we simply recurse using the local extractor. Basically, a type 1 attacker is defined as before, except we run $\mathsf{LExtract}$ for the inner BARG to extract the witness. Similarly, for type 2 and 3 attacker, we first run $\mathsf{LExtract}$ for outer BARG, and then run $\mathsf{LExtract}$ for inner BARG as well (if needed, i.e. only for type 3 attacker). □

Deletion privacy. This follows from the instance privacy property of the lv-BARG scheme. Concretely, we prove the following.

Lemma 9.5. If the BARG scheme lv-BARG satisfies instance privacy, then de-BARG satisfies deletion privacy.

Proof. This follows immediately from the instance privacy property of the lv-BARG scheme. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks deletion privacy of the de-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. Then we construct another PPT attacker $\mathcal{B}$ that breaks instance privacy property of lv-BARG.

Briefly, the attacker $\mathcal{A}$ starts by outputting $(1^k, 1^n, i^*)$.[16] The reduction algorithm $\mathcal{B}$ simply sends $(1^k, 1^n, i^*)$ to the lv-BARG challenger, and the challenger sends $\mathsf{lvbg.crs}$ to $\mathcal{B}$. $\mathcal{B}$ then samples parameters for the outer BARG scheme to sample $\widehat{\mathsf{lvbg.crs}}$ and $\widehat{\mathsf{lvbg.td}}$ as in the actual scheme. (That is, $(\widehat{\mathsf{lvbg.crs}}, \widehat{\mathsf{lvbg.td}}) \leftarrow \mathsf{lvB.Setup}^{\hat{\mathcal{L}}}(1^\lambda, 1^k, 1^{\hat{n}}, i^*)$). It then sends $(\mathsf{lvbg.crs}, \widehat{\mathsf{lvbg.crs}})$ as the crs to $\mathcal{A}$. $\mathcal{A}$ then outputs a set S along with two sequences of k instance-witness pairs $(x_{i,b}, \omega_{i,b})$ for $i \in [k]$ and $b \in \{0,1\}$. $\mathcal{B}$ then samples a random bit $\beta \leftarrow \{0,1\}$, and then forwards $\overline{S} = [k] \setminus S$ and instance-witness pairs $\{(x_{i,\beta}, \omega_{i,\beta})\}_{i\in[k]}$ to the instance privacy challenger. Next, the challenger replies with a set of opening and (sanitized) batch proof pairs $\{(\mathsf{aux}_j, \pi'_j)\}_{j\notin S}$. Finally, $\mathcal{B}$ uses $(\pi'_j, \mathsf{aux}_j, j)$ as witness for each instance $\hat{x}_j = (x_j, \mathsf{lvbg.crs})$ for every $j \notin S$, and computes a batch proof using $\widehat{\mathsf{lvbg.crs}}$. That is, it computes a deleted proof π^{red} as in the Delete algorithm, except

[16] Technically, in the deletion privacy definition, the adversary only outputs $1^k, 1^n$, but here we expect it outputs i^* as well. We point out that this is simply due to the fact that our design de-BARG scheme also satisfies somewhere extractability, thus we have used an extended syntax.

that it receives the opening from the challenger rather than computing on its own. $\mathcal{B}$ then sends π^{red} to $\mathcal{A}$, and $\mathcal{A}$ outputs its guess b^*. If $\beta = b^*$, then $\mathcal{B}$ outputs 0 as its guess (i.e., opening and sanitized batch proofs were computed honestly). Otherwise, $\mathcal{B}$ outputs 1 to signal that they were simulated.

Using a standard probability analysis, we obtain that $\mathcal{B}$'s advantage is at least $\epsilon/2$, thus this breaks the instance privacy property of the lv-BARG scheme. This completes the proof of the lemma. □

This completes the proof of our main theorem. □

10 Mutable Batch Proofs for $\mathcal{C}$-batchNP mutation

In this section, we provide a constructions for mutable batch proofs for a special class of non-deterministic mutations, that we call $\mathcal{C}$-batchNP mutations where $\mathcal{C} = \{\mathcal{C}_k\}_{k\in\mathbb{N}}$ is a family of circuits (where $\mathcal{C}_k : \{0,1\}^k \to \{0,1\}$). We start by defining the language associated with such mutation operations. For simplicity, we call it $\mathcal{C}$-batchNP languages.

$\mathcal{C}$-batchNP languages. For any **NP** language $\mathcal{L}$, the $\mathcal{C}$-batchNP language $\mathcal{L}_{\mathcal{C}}^{(k)}$ is defined as:

$$\mathcal{L}_{\mathcal{C}}^{(k)} = \{(C, x_1, \ldots, x_k) : C \in \mathcal{C}_k \text{ and } C(\mathbb{I}_{x_1\in\mathcal{L}}, \ldots, \mathbb{I}_{x_k\in\mathcal{L}}) = 1\}\,,$$

where indicator variable is defined as $\mathbb{I}_{x\in\mathcal{L}} = 1$ iff $x \in \mathcal{L}$.

In recent works [BBK+23], such languages were considered where $\mathcal{C}$ contained all monotone circuits. In this work, we consider a more general abstraction for such languages, and provide new constructions for mutable batch proofs where the mutated language corresponds to appropriate $\mathcal{C}$-batchNP language. We start by recalling some useful preliminaries.

10.1 Preliminaries

SNARGs. A SNARG for **NP** language $\mathcal{L}$ consists of the following algorithms.

$\mathsf{Setup}(1^\lambda, 1^n, \mathcal{L}) \to \mathsf{crs}$. The setup algorithm takes as input the security parameter λ, instance length n, and language description $\mathcal{L}$. It outputs crs.

$\mathsf{Prove}(\mathsf{crs}, x, \omega) \to \pi$. The prover algorithm takes as input a crs, instance-witness pair (x, ω), and outputs a proof π.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 0/1$. The verification algorithm takes as input a crs, an instance x, and a proof π. It outputs a bit denoting proof validity.

Correctness and succinctness. A SNARG is said to be correct and succinct if for every $\lambda, n \in \mathbb{N}$, any instance $x \in \mathcal{L} \cap \{0,1\}^n$ and corresponding witness ω, and every proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \omega)$, the following holds:

Completeness. $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$.

Succinctness. $|\pi| \le \mathsf{poly}(\lambda, n)$. That is, the size of the batched proof is bounded by a fixed polynomial in λ, n.

Soundness. A SNARG scheme is said to be non-adaptively sound if an attacker can not create a valid proof for any invalid instance fixed before receiving the CRS.

Definition 10.1 (non-adaptive soundness)**.** A SNARG satisfies non-adaptive soundness if for any polynomial $n = n(\lambda)$, every PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any instance $x \notin \mathcal{L}$ where $|x| = n$, the following holds for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \mathcal{L}) \\ \pi \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Extractability. Furthermore, we consider the notion of extractability considered and achieved in prior works. Full extractability states that it an attacker breaks non-adaptive soundness, then there exists an extractor which can extract a sequence of valid witnesses given only oracle access to the attacker.

Definition 10.2 (argument of knowledge)**.** A SNARG is an argument of knowledge if for every constant $c \in \mathbb{N}$, there exists a PPT oracle machine $\mathcal{E}_c$ such that for every polynomial $n = n(\lambda)$, every PPT attacker $\mathcal{A}$, if for infinitely many $\lambda \in \mathbb{N}$, there exists instance x where $|x| = n$ such that

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \mathcal{L}) \\ \pi \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \geq \frac{1}{\lambda^c}.$$

then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every such λ, the following holds (where $\mathcal{R}$ denotes the relation for language $\mathcal{L}$),

$$\Pr\left[\mathcal{R}(x, \omega) = 1 : \omega \leftarrow \mathcal{E}_c^{\mathcal{A}}(1^\lambda, 1^n, \mathcal{L})\right] = 1 - \mathsf{negl}(\lambda).$$

Prior works also considered a weaker somewhere extractability notion. However, in this work, we stick to the just the regular extractability notion.

10.2 Mutable proofs via locally verifiable BARGs and SNARGs

Here we give a construction for mutable proofs for $\mathcal{C}$-batchNP mutation operations by combining any locally verifiable BARG scheme with any extractable SNARG scheme for language $\mathcal{C}$-batchNP. By combining it with recent constructions for extractable SNARGs, we obtain a mutable proofs for monotone policy batchNP mutation operations under learning with errors assumption.

Construction. Let $\mathcal{C} = \{\mathcal{C}_\ell\}_{\ell \in \mathbb{N}}$ be a family of circuits as discussed above (where ℓ denotes the input length of circuits in class $\mathcal{C}_\ell$), lv-BARG $= (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract}, \mathsf{LOpen}, \mathsf{LVfy})$ be a locally verifiable seBARG for language $\mathcal{L}$, and SNARG $= (\mathsf{SNARG.Setup}, \mathsf{SNARG.Prove}, \mathsf{SNARG.Verify})$ for language $\mathcal{C}$-batchNP. Below we describe our mutable BARG scheme for language $\mathcal{L}$ and mutation class $\mathcal{C}$.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, \mathcal{C}_\ell) \to (\mathsf{crs}, \mathsf{td})$. It runs the setup algorithms for lv-BARG and SNARG schemes. Namely, it outputs $\mathsf{crs} = (\mathsf{crs}', \mathsf{lvbg.crs})$ where

$$\mathsf{crs}' \leftarrow \mathsf{SNARG.Setup}(1^\lambda, 1^n, \mathcal{C}_\ell), \quad \mathsf{lvbg.crs} \leftarrow \mathsf{lvB.Setup}(1^\lambda, 1^k, 1^n).$$

$\mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i \in [k]}) \to \pi$. It generates the proof as $\pi \leftarrow \mathsf{lvB.Prove}(\mathsf{lvbg.crs}, \{(x_i, \omega_i)\}_i)$.

$\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i\in[k]}, \pi) \to 0/1$. It runs lv-BARG verifier to output $\mathsf{lvB.Verify}(\mathsf{lvbg.crs}, \{x_i\}_i, \pi)$.

$\mathsf{Eval}(\mathsf{crs}, C, \{(j_i, X_i, \pi_i)\}_{i\le\ell}) \to \widehat{\pi}$. Let $X_i = (x_{i,1}, \ldots, x_{i,k})$. It starts by running the local opening algorithm for each proof π_i corresponding to index j_i as follows

$$\forall i \in [\ell], \quad (\mathsf{lvbg.aux}_i, \mathsf{lvbg}.\pi'_i) = \mathsf{lvB.LOpen}(\mathsf{lvbg.crs}, X_i, j_i, \pi_i).$$

Next, it runs the SNARG prover to create mutated proof as $\widehat{\pi} \leftarrow \mathsf{SNARG.Prove}(\mathsf{crs}', x_C, \omega_C)$, where the instance-witness pair is defined as

$$x_C = (C, x_{1,j_1}, \ldots, x_{\ell,j_\ell}), \quad \omega_C = ((\mathsf{lvbg.aux}_1, \mathsf{lvbg}.\pi'_1), \ldots, (\mathsf{lvbg.aux}_\ell, \mathsf{lvbg}.\pi'_\ell)).$$

Note that here the **NP** language we use for the $\mathsf{batchNP}$ portion of the $\mathcal{C}$-$\mathsf{batchNP}$ language for SNARG corresponds to $\mathsf{lvB.LVfy}_{\mathsf{lvbg}}.\mathsf{crs}$, where message is the instance and the local opening and proof are the witnesses.

$\mathsf{Post\text{-}Verify}(\mathsf{crs}, C, x_C, \widehat{\pi}) \to 0/1$. The (post evaluation) verifier algorithm runs the SNARG verifier to output $\mathsf{SNARG.Verify}(\mathsf{crs}', x_C, \widehat{\pi})$.

Remark 10.3. Note that the above construction still preserves the local verifiability feature for the underlying BARG scheme. Thus, it would be more appropriate to define the class of supported mutation functions for this proof system to contain both $\mathcal{C}$-$\mathsf{batchNP}$ mutation as well as (private) local opening mutations.

Correctness and succinctness. The basic correctness and succinctness (about non-mutated proofs) in the above scheme follow directly from correctness and succinctness of the lv-BARG scheme since the Prove and Verify algorithms are just $\mathsf{lvB.Prove}$ and $\mathsf{lvB.Verify}$. Next, by correctness of $\mathsf{lvB.LVfy}$ and SNARG, we get that the mutation correctness also holds. This is because ω_C as described in the construction will be a valid witness for x_C as long as $C(\mathbb{I}_{x_{1,j_1}\in\mathcal{L}}, \ldots, \mathbb{I}_{x_{k,j_k}\in\mathcal{L}}) = 1$. Furthermore, succinctness of mutation proof follows from succinctness of SNARG succinctness and the fact that the output of the local opening algorithm is also succinct.

Soundness. Below we prove soundness of our mutable BARGs.

Theorem 10.4. If the locally verifiable BARG scheme $\mathsf{lv\text{-}BARG}$ satisfies semi-adaptive local soundness with adversarial openings (Definition 5.2), and the SNARG scheme SNARG satisfies argument of knowledge property (Definition 10.2), then the above scheme is a mutable BARG scheme satisfying non-adaptive soundness (Definition 4.2).

Proof. The proof follow from a simple type-based reduction to $\mathsf{lv\text{-}BARG}$ and SNARG. We provide a brief sketch below.

To begin the argument, consider a successful attacker $\mathcal{A}$ on mutation privacy. Then, suppose $\mathcal{E}$ is the extractor for SNARG, then we define a partial extraction oracle algorithm $\mathsf{Extract}$ as follows–

$\mathsf{Extract}^{\mathcal{A}}(1^\lambda, 1^n, \mathcal{C}_\ell)$: It runs the oracle machine $\mathcal{E}^{\mathcal{A}}$ (with oracle access to $\mathcal{A}$), and uses it to extract the SNARG witness ω_C.

Next, We divide the analysis in two cases.

Type 1 attacker: ω_C is a not valid witness for instance x_C. In this case, we reduce the attack to breaking argument of knowledge property of SNARG.

Type 2 attacker: ω_C is a not valid witness for instance x_C, but one of the underlying instances $x_i \notin \mathcal{L}$, where $x_C = (C, x_1, \ldots, x_\ell)$. In this case, we break semi-adaptive local soundness with adversarial openings. To design a reduction algorithm, the reduction algorithm guesses the index i such that $x_i \notin \mathcal{L}$, and uses as its attack on the local soundness. The reduction suffers from a polynomial loss in the advantage, but still results in a contradiction.

Note that any successful non-adaptive soundness attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Remark 10.5 (somewhere extraction)**.** In our applications to homomorphic signatures, we need to rely on a slightly stronger somewhere extractability property of our mutable batch proofs. Somewhere extractability for mutable BARGs states that for every index i, there exists an oracle PPT machine $\mathcal{E}^i$ that is able to extract a valid witness for the i^{th} instance in the mutated instance x_C. Furthermore, the adversary cannot distinguish the CRS generated by any two different oracle machines $\mathcal{E}^i$ and $\mathcal{E}^j$. This can be achieved by combining the above notion of SNARG extractor with the local somewhere extractor for underlying batch proofs. To avoid compicating the syntax, we keep this as a remark.

Privacy. Similarly, we can prove privacy of our mutable BARGs scheme.

Theorem 10.6. If the locally verifiable BARG scheme lv-BARG satisfies full privacy (Definition 5.6), then the above scheme is a mutable BARG scheme satisfying mutation privacy (Definition 4.3).

Proof. The proof follow from a straightforward reduction to lv-BARG privacy property. □

10.2.1 Instantiating from LWE

Given the recent work by Brakerski et al. [BBK+23], we know the following.

Theorem 10.7 (Paraphrased [BBK+23])**.** Assuming the polynomial hardness of learning with errors (LWE), there exist non-adaptively sound SNARGs for monotone policy batchNP for all polynomial-size monotone circuit policies, and the SNARG is an argument of knowledge.

Combining this with our construction of locally verifiable BARGs and Theorems 10.4 and 10.6, we obtain the following corollary.

Corollary 10.8. Assuming the polynomial hardness of learning with errors (LWE), there exist non-adaptively sound and private mutable batch proofs for any **NP** language with mutation class is $\mathcal{C}$-batchNP where $\mathcal{C}$ contains all polynomial-size monotone circuits.

We leave it as an interesting open problem to design mutable batch proofs for such $\mathcal{C}$-batchNP mutations from other standard assumptions.

11 Advanced Signature Schemes

In this section, we recall three types of advanced signature schemes– redactable signatures, aggregate signatures with local verifiability, and homomorphic signatures. Later we provide constructions of each of these signature schemes based on our mutable BARG schemes with appropriate mutation class.

For example, we show that a de-BARG directly implies a redactable signature with optimal parameters. Prior to this work, all known redactable signatures were restricted in one of many ways. Such as either they had sub-optimal parameters (in terms of sizes of signatures and public keys), or relied on idealized assumptions (such as random oracles, or knowledge-based assumptions), or could not support multi-hop deletions, or provided only weak forms of deletion privacy or unforgeability. In this work, we show a general template from de-BARGs to design redactable signatures that avoids all such limitations.

Further, we discuss how BARGs with different features and security properties proposed in this work can be used to design signatures with a variety of different features. In this section, we provide the definitional framework for different types of signature schemes. In following sections, we provide new constructions for these signatures based on standard assumptions.

11.1 Redactable Signatures

A redactable signature scheme [JMSW02, SBZ01] consists of the following polynomial time algorithms.

$\mathsf{Setup}(1^\lambda, L) \to (\mathsf{vk}, \mathsf{sk})$. It takes as input the security parameter λ and message length L, outputs a signing-verification key pair $(\mathsf{vk}, \mathsf{sk})$. (Note that message length is given in binary, thus can be as large as 2^λ.)

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. The signing algorithm takes as input a signing key sk and message $m \in \{0,1\}^L$, and computes a signature σ.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 0/1$. The verification algorithm takes as input a verification key vk, message $m \in \{0, 1, \perp\}^L$, and signature σ. It outputs a bit.

NOTE. The '$\perp$' symbol denotes a redacted bit. One can alternatively represent a redacted message as a sequence of bits $(b_i)_{i \in T}$ along with a set T, where T denotes all the messages that have not been redacted. We use '$\perp$' above for notational clarity, but our constructions can work with either representation.

$\mathsf{Redact}(\mathsf{vk}, m, S, \sigma) \to \sigma'$. It takes as input key vk, a message $m \in \{0, 1, \perp\}^L$, and a redaction set $S \subset [|m|]$, and outputs a redacted signature σ'.

NOTE. We say the scheme is multi-hop redactable if Redact can be used to further redact a redacted signature σ'.

Notation. For any message m and set S, we use $m^{\{S\}}$ as a shorthand notation to denote a message m redacted at places in S. That is, $m^{\{S\}}[i] = m[i]$ if $i \notin S$, else $m^{\{S\}}[i] = \perp$. In words, the i-th bit of $m^{\{S\}}$ matches i-th bit of m if $i \notin S$, otherwise it is set as $\perp$.

Correctness and succintness. A redactable scheme satisfies correctness if Verify accepts an honestly generated signature σ or any honestly computed redacted signature σ'. Moreover, the scheme is succinct if the size of a redacted signature does not grow with the maximum message length or the size of redacted message. Formally, it means that for every $\lambda \in \mathbb{N}$, $L \leq 2^\lambda$, key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, L)$, messages $m \in \{0,1\}^L$, every signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$, every redaction set $S \subset [L]$, and redacted signature $\sigma' \leftarrow \mathsf{Redact}(\mathsf{vk}, m, S, \sigma)$, the following holds:

$$\mathsf{Verify}(\mathsf{vk}, m, \sigma) = \mathsf{Verify}(\mathsf{vk}, m^{\{S\}}, \sigma') = 1, \qquad |\sigma'| = \mathsf{poly}(\lambda).$$

We also consider succinctness for regular (unredacted) messages which says $|\sigma| = \mathsf{poly}(\lambda)$. Further, we can naturally generalize correctness and succintness for multi-hop redaction. In that case, we require that the size of even a multi-hop redacted signature will be $\mathsf{poly}(\lambda)$, and Verify also accepts a multi-hop redacted signature.

Basic unforgeability. Next, recall the standard unforgeability security for plain signatures.

Definition 11.1 (unforgeability)**.** A signature scheme is said to be unforgeable if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*) = 1 : \begin{array}{l} L \leftarrow \mathcal{A}(1^\lambda), (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, L) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

and $\mathcal{A}$ is admissible as long as it did not query m^* to the Sign oracle.

Unforgeability and unlinkability of redaction. For redactable signatures, there are two main security properties. The first is a extension of unforgeability for redactable signature. It says that an attacker cannot produce a forgery on a message (either *redacted* or unredacted) for which it did not receive a signature yet. This is the simplest formulation in which the attacker does not make queries to the redaction algorithm. We also consider a strong notion of redaction unforgeability where the attacker can also make queries to the redaction algorithm. We describe both versions below.

Definition 11.2 (redaction unforgeability)**.** A redactable signature scheme is said to be unforgeable if no PPT attacker wins in the unforgeability game described in Definition 11.1 even when the challenge message m^* could be a redacted message as long as $m^* \neq m_i^{\{S\}}$ for any queried message m_i and any redaction set S. That is, unforgeability of signatures holds as long as an attacker does not produce a trivial forgery by replaying a signature or by outputting redacted signature for a queried message.

Definition 11.3 (strong redaction unforgeability)**.** A redactable signature scheme is said to be strongly unforgeable if no PPT attacker wins in the unforgeability game described in Definition 11.1 where:

1. $\mathcal{A}$ can also make queries to two more oracles – $\mathsf{Redact}, \mathsf{Request}$. Sign oracle computes the signature, but only sends a signature handle (i.e., an index) to $\mathcal{A}$. Moreover, $\mathcal{A}$ can make queries to Redact where it provides a handle and a redaction set, and Redact oracle answers with a handle to the redacted signature. To request a signature, $\mathcal{A}$ provides a handle to $\mathsf{Request}$ oracle, and receives the corresponding redacted/unredacted signature from the challenger.

2. A forgery on m^* is defined as successful if none of the signatures received by $\mathcal{A}$ as part of $\mathsf{Request}$ oracle queries are valid for m^*.

Additionally, we consider unlinkability property for redacted signatures. The intuition is that two redacted signatures of the same message should look indistinguishable, and not reveal anything about their source messages. We also consider a stronger unlinkability property where a signature also hides whether it was a redacted signature or a fresh signature (in this case, we expect Sign to be able to sign redacted messages).

Definition 11.4 (redaction unlinkability)**.** A redactable signature scheme is said satisfy unlinkability if for every stateful admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}(\sigma') = \beta : \begin{array}{l} L \leftarrow \mathcal{A}(1^\lambda), (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, L) \\ (S, m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(\mathsf{vk}) \\ \sigma_0 \leftarrow \mathsf{Sign}(\mathsf{sk}, m_0),\ \sigma_0 \leftarrow \mathsf{Sign}(\mathsf{sk}, m_0) \\ b \leftarrow \{0,1\},\ \sigma' \leftarrow \mathsf{Redact}(\mathsf{vk}, m_b, S, \sigma_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible iff $m_0^{\{S\}} = m_1^{\{S\}}$ (that is, redacted messages are identical for all message bits that are not redacted).

Definition 11.5 (strong redaction unlinkability)**.** A redactable signature scheme is said satisfy strong unlinkability if a redacted signature as defined in Definition 11.4 is indistinguishable from a fresh signature on message $m_0^{\{S\}}$.

11.2 Locally Verifiable Aggregate Signatures

A locally verifiable aggregate signature scheme [BGLS03, GV22] consists of the following polynomial time algorithms.

$\mathsf{CRS}(1^\lambda, 1^k) \to \mathsf{crs}$. On input the security parameter λ and upper bound on number of aggregations k, the CRS generation algorithm samples global parameters crs. *(All the remaining algorithms take* crs *as input, and for ease of notation we do not write it explicitly.)*

$\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$. The setup algorithm, on input the security parameter λ, outputs a pair of signing and verification keys $(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. The signing algorithm takes as input a signing key sk and message $m \in \{0,1\}^\lambda$, and computes a signature σ.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 0/1$. The verification algorithm takes as input a verification key vk, message $m \in \{0,1\}^\lambda$, and signature σ. It outputs a bit to signal signature validity.

$\mathsf{Agg}(\{(\mathsf{vk}_i, m_i, \sigma_i)\}_i) \to \widehat{\sigma}/\perp$. The signature aggregation algorithm takes as input a sequence of tuples, each containing a verification key vk_i, a message m_i, a signature σ_i, and it outputs either an aggregated signature $\widehat{\sigma}$ or a special abort symbol $\perp$.

$\mathsf{AggV}(\{(\mathsf{vk}_i, m_i)\}_i, \widehat{\sigma}) \to 0/1$. The aggregate verify algorithm takes as input a sequence of tuples, each containing a verification key vk_i, a message m_i, and it outputs a bit to signal whether the aggregated signature $\widehat{\sigma}$ is valid or not.

$\mathsf{LOpen}(\widehat{\sigma}, \{(\mathsf{vk}_i, m_i)\}_i, j) \to \mathsf{aux}_j$. The local opening algorithm takes as input an aggregated signature $\widehat{\sigma}$, a sequence of tuples (each containing a verification key vk_i and a message m_i for $i \in [k]$), and an index $j \in [k]$. It outputs auxiliary information aux_j.

$\mathsf{LVfy}(\widehat{\sigma}, \mathsf{vk}, m, j, \mathsf{aux}) \to 0/1$. The local verification algorithm takes as input an aggregated signature $\widehat{\sigma}$, a verification key vk, a message m, index j, and auxiliary information aux. It outputs a bit to signal whether the aggregate signature $\widehat{\sigma}$ contains a signature for message m under verification key vk, or not.

Correctness and Compactness. A locally verifiable aggregate signature scheme is said to be correct and compact if for all $\lambda, k \in \mathbb{N}$, parameters $\mathsf{crs} \leftarrow \mathsf{CRS}(1^\lambda, 1^k)$, verification-signing key pairs $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$ for $i \in [k]$, messages m_i for $i \in [k]$, every signature $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m_i)$ for $i \in [k]$, the following holds:

Correctness of signing. For all $i \in [k]$, $\mathsf{Verify}(\mathsf{vk}_i, m_i, \sigma_i) = 1$.

Correctness of aggregation. If $\widehat{\sigma} = \mathsf{Agg}(\{(\mathsf{vk}_i, m_i, \sigma_i)\}_i)$, then

$$\mathsf{AggV}(\{(\mathsf{vk}_i, m_i)\}_i, \widehat{\sigma}) = 1.$$

Correctness of local opening. For every $j \in [k]$, $\mathsf{LVfy}(\widehat{\sigma}, \mathsf{vk}_j, m_j, j, \mathsf{aux}_j) = 1$ where opening and signature are computed as $\mathsf{aux}_j = \mathsf{LOpen}(\widehat{\sigma}, \{(\mathsf{vk}_i, m_i)\}_i, j)$.

Compactness of aggregation and opening. $|\widehat{\sigma}|, |\mathsf{aux}| \leq \mathsf{poly}(\lambda)$. That is, the sizes of an aggregated signature and auxiliary opening information are polynomially bounded in λ.

Remark 11.6 (Fully Public Openings)**.** An aggregate signature scheme is said to have fully public local openings if the algorithm LOpen does not need read the aggregated signature $\widehat{\sigma}$. That is, it has the syntax — $\mathsf{LOpen}(\{(\mathsf{vk}_i, m_i)\}_i, j) \to \mathsf{aux}_j$. That is, LOpen is oblivious to the aggregated signature.

Security. In addition to regular unforgeability Definition 11.1, we consider aggregated unforgeability with(/out) adversarial openings.

Definition 11.7 (aggregated unforgeability)**.** An aggregate signature scheme satisfies aggregated unforgeability if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\mathsf{AggV}(\{(\mathsf{vk}_i^*, m_i^*)\}_i, \widehat{\sigma}^*) = 1 : \begin{array}{l} 1^k \leftarrow \mathcal{A}(1^\lambda), \mathsf{crs} \leftarrow \mathsf{CRS}(1^\lambda, 1^k) \\ (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\{(\mathsf{vk}_i^*, m_i^*)\}_{i \in [k]}, \widehat{\sigma}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{vk}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible if there exists $i \in [k]$ such that $\mathsf{vk}_i^* = \mathsf{vk}$ and m_i^* was not queried by $\mathcal{A}$ to the $\mathsf{Sign}(\mathsf{sk}, \cdot)$ oracle.

Definition 11.8 (aggregated unforgeability with adversarial opening)**.** An aggregate signature scheme satisfies aggregated unforgeability *with adversarial openings* if no PPT adversary $\mathcal{A}$ wins in the unforgeability game described in Definition 11.1 where a successful forgery can be respresented as a tuple of an aggregated signature $\widehat{\sigma}$, local opening aux, message m^*, and index j such that LVfy accepts it under challenge key vk, and m^* was never sign queried.

In this work, we also propose a strong notion of message privacy for locally verifiable aggregate signatures, called local opening simulatability. It says that a locally opened aggregate signature hides complete information about any aggregate signature (and corresponding messages) that were not opened. As in the case of instance privacy for lv-BARGs, we need to update the syntax of LOpen where it now also outputs a sanitized aggregate signature $\widehat{\sigma}'$.

Definition 11.9 (local opening simulatability)**.** A locally verifiable aggregate signature scheme $\mathcal{S}$ is said to satisfy message privacy with local opening if there exists a PPT stateful simulator $\mathtt{Sim}$ such that for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathcal{A}(\mathsf{aux}_b, \widehat{\sigma}'_b) = b \\ \wedge \left(\begin{array}{l} \forall i \in [k], \\ \mathsf{Verify}(\mathsf{vk}_i, m_i, \sigma_i) = 1 \end{array}\right) \end{array} : \begin{array}{l} 1^k \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ \mathsf{crs}_0 \leftarrow \mathsf{CRS}(1^\lambda),\ \mathsf{crs}_1 \leftarrow \mathtt{Sim}(1^\lambda, 1^k) \\ (j, \{(\mathsf{vk}_i, m_i, \sigma_i)\}_{i\in[k]}) \leftarrow \mathcal{A}(\mathsf{crs}_b) \\ \widehat{\sigma}_0 \leftarrow \mathsf{Agg}(\{(\mathsf{vk}_i, m_i, \sigma_i)\}_i) \\ (\mathsf{aux}_0, \widehat{\sigma}'_0) \leftarrow \mathsf{LOpen}(\widehat{\sigma}_0, \{(\mathsf{vk}_i, m_i)\}_i, j) \\ (\mathsf{aux}_1, \widehat{\sigma}'_1) \leftarrow \mathtt{Sim}(j, \mathsf{vk}_j, m_j, \sigma_j) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

11.3 Homomorphic Signatures

Homomorphic signatures [BFKW09, BF11] have been defined with many different variations such as single/multi-dataset, conjoined/independent signing, bounded length datasets, etc. In this work, we consider schemes with the following polynomial time algorithms. These directly capture (or generically imply) all existing formalizations of homomorphic signatures. Consider a circuit family $\mathcal{C} = \{\mathcal{C}_\ell\}_{\ell\in\mathbb{N}}$, where circuits in $\mathcal{C}_\ell$ take ℓ-bits as inputs.

$\mathsf{Setup}(1^\lambda, \mathcal{C}_\ell) \to (\mathsf{vk}, \mathsf{sk})$. On input the security parameter λ and a circuit class $\mathcal{C}_\ell$, it outputs a key pair $(\mathsf{vk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, i, b) \to \sigma$. The signing algorithm takes as input a signing key sk, index $i \in [\ell]$, and a message bit $b \in \{0,1\}$. It outputs a signature σ.

$\mathsf{Eval}(\mathsf{vk}, \{(m_i, \sigma_i)\}_i, C) \to \sigma'$. It takes as input key vk, a sequence of ℓ message-signature pairs m_i, σ_i, and a circuit $C \in \mathcal{C}_\ell$. It outputs an evaluated signature σ' corresponding to message $m' = C(m_1, \ldots, m_\ell)$.

NOTE. We say the scheme has multi-hop evaluation if Eval can also run on an evaluated signature σ'.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma, C) \to 0/1$. The verification algorithm takes as input key vk, message $m \in \{0,1\}^*$, signature σ, and circuit C. It outputs a bit to denote whether σ is a valid evaluated signature of message m w.r.t. circuit C.

NOTE. One can verify a signature that has not been evaluated by setting $C = \mathbb{I}$, an identity circuit.

Correctness and succintness. A homomorphic signature scheme for circuit class $\mathcal{C}$ satisfies correctness and succintness if for every $\lambda, \ell \in \mathbb{N}$, key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{C}_\ell)$, any circuit

$C \in \mathcal{C}_\ell$, any ℓ-bit message $m = (m_1, \ldots, m_\ell) \in \{0,1\}^\ell$, every signature $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, i, m_i)$, and evaluated signature $\sigma' \leftarrow \mathsf{Eval}(\mathsf{vk}, \{(m_i, \sigma_i)\}_i, C)$, the following holds:

$$\mathsf{Verify}(\mathsf{vk}, C(m_1, \ldots, m_\ell), \sigma', C) = 1, \qquad |\sigma_i|, |\sigma'| = \mathsf{poly}(\lambda).$$

We want to highlight that we consider a stronger notion of succinctness than prior works as we require the size of signatures to be independent of the input/output size of the circuit that is being evaluated. For instance, C could have a really long output, and we still require the evaluated signature to have size independent of $|C(M)|$.

Further, we can naturally generalize correctness and succintness for multi-hop evaluation. In that case, we require that the size of even a multi-hop evaluated signature will be $\mathsf{poly}(\lambda)$, and Verify also accepts all such multi-hop evaluated signatures.

Unforgeability. For unforgeability of homomorphic signatures, we define the most general fully adaptive model where the adversay can query on arbitrary datasets for arbitrary tags, and eventually the adversary wins if it produces a valid signature for any output value w.r.t. any adversarially selected circuit where the signature could not be computed by evaluating any combination of queried messages. Formally, we define it as follows.

Definition 11.10 (homomorphic unforgeability)**.** A homomorphic signature scheme for circuit class $\mathcal{C}$ satisfies unforgeability if for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, \ell \in \mathbb{N}$, the following holds

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, m^*, \sigma^*, C^*) = 1 : \begin{array}{l}(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{C}_\ell) \\ (m^*, \sigma^*, C^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot, \cdot)}(1^\lambda, \mathsf{vk})\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ is admissible if for any index i it makes at most one signing query, and $m^* \neq C^*(m)$ for message m created by combining the dataset queried (bit-by-bit) to the Sign oracle.

Definition 11.11 (circuit-selective homomorphic unforgeability)**.** A homomorphic signature scheme for circuit class $\mathcal{C}$ satisfies circuit-selective unforgeability if no PPT adversary $\mathcal{A}$ wins in the unforgeability game described in Definition 11.10 where $\mathcal{A}$ must declare C^* at the beginning of the game.

Context Hiding. As in prior works, we also consider the notion of context hiding for homomorphic signatures. It says that there exists a simulator that can simulate an evaluated signature given only the output message and the circuit used during evaluation.

Definition 11.12 (context hiding)**.** A homomorphic signature scheme for circuit class $\mathcal{C}$ satisfies context hiding if there exists a PPT stateful simulator $\mathtt{Sim}$ such that for every admissible PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda, \ell \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l}\mathcal{A}(\sigma'_b) = b \wedge \\ \mathsf{Verify}(\mathsf{vk}, y, \sigma'_0, C) = 1\end{array} : \begin{array}{l}(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{C}_\ell),\ b \leftarrow \{0,1\} \\ (\{(m_i, \sigma_i)\}_i, C) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot, \cdot)}(1^\lambda, \mathsf{vk}) \\ \text{Let } y = C(m_1, \ldots, m_\ell) \\ \sigma'_0 \leftarrow \mathsf{Eval}(\mathsf{vk}, \{(m_i, \sigma_i)\}_i, C) \\ \sigma'_1 \leftarrow \mathtt{Sim}(\mathsf{vk}, y, C)\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Note that the above property only guarantees to hide non-trivial information about the input datasets, and not the homomorphic circuit or the fact that it is an evaluated circuit. Such stronger forms of context hiding are called strong context hiding, and they are not the focus of this work.

12 Application 1: Locally Verifiable Aggregate Signatures

In this section, we provide a locally verifiable aggregate signature scheme from any lv-BARG scheme. Below we provide our construction.

Construction. Let $\mathcal{S} = (\mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify})$ be any regular signature scheme, and $\mathsf{lv\text{-}BARG} = (\mathsf{lvB.Setup}, \mathsf{lvB.Prove}, \mathsf{lvB.Verify}, \mathsf{lvB.Extract}, \mathsf{lvB.LOpen}, \mathsf{lvB.LVfy})$ be a locally verifiable seBARG scheme for language $\mathcal{L}_{\mathcal{S}}$ (Fig. 4). Below we describe our locally verifiable aggregate signature scheme Agg.

Language $\mathcal{L}_{\mathcal{S}}$

Instance: $\hat{x} :=$ (key vk, message m).

Witness: $\hat{\omega} :=$ signature σ w.r.t. vk.

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \mathcal{L}_{\mathcal{S}}$ if σ is a valid signature of m under vk. That is, $\mathcal{S}.\mathsf{Verify}(\mathsf{vk}, m, \sigma) = 1$.

Figure 4: Language $\mathcal{L}_{\mathcal{S}}$ for the lv-BARG.

$\mathsf{CRS}(1^\lambda, 1^k) \to \mathsf{crs}$. It samples lv-BARG parameters as $(\mathsf{lvbg.crs}, *) \leftarrow \mathsf{lvB.Setup}(1^\lambda, 1^k, 1^n, \perp)$. Here n is signature length in $\mathcal{S}$, $\perp$ denotes a dummy/random target index, and $*$ means it ignores the trapdoor. It outputs $\mathsf{crs} = \mathsf{lvbg.crs}$.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$. Same as $\mathcal{S}.\mathsf{Setup}$. That is, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)$.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. Same as $\mathcal{S}.\mathsf{Sign}$. That is, $\sigma \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, m)$.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 0/1$. Same as $\mathcal{S}.\mathsf{Verify}$. That is, it outputs $\mathcal{S}.\mathsf{Verify}(\mathsf{vk}, m, \sigma)$.

$\mathsf{Agg}(\{(\mathsf{vk}_i, m_i, \sigma_i)\}_i) \to \widehat{\sigma}$. It computes the aggregated signature as a BARG proof with instance $x_i = (\mathsf{vk}_i, m_i)$, and corresponding witness $\omega_i = \sigma_i$ for all $i \in [k]$. Formally, it outputs the aggregate signature as a batch proof $\widehat{\sigma} = \pi \leftarrow \mathsf{lvB.Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_i)$.

$\mathsf{AggV}(\{(\mathsf{vk}_i, m_i)\}_i, \widehat{\sigma}) \to 0/1$. It runs $\mathsf{lvB.Verify}$ on $\widehat{\sigma}$ with instances $\{x_i = (\mathsf{vk}_i, m_i)\}_{i \in [k]}$. Concretely, it outputs $\mathsf{lvB.Verify}(\mathsf{crs}, \{x_i\}_i, \widehat{\sigma})$.

$\mathsf{LOpen}(\widehat{\sigma}, \{(\mathsf{vk}_i, m_i)\}_i, j) \to (\mathsf{aux}_j, \widehat{\sigma}')$. It runs $\mathsf{lvB.LOpen}$, and outputs the auxiliary hint aux_j and updated signature $\widehat{\sigma}'$ as $(\mathsf{aux}_j, \widehat{\sigma}') \leftarrow \mathsf{lvB.LOpen}(\mathsf{crs}, \{x_i = (\mathsf{vk}_i, m_i)\}_i, j, \widehat{\sigma})$.

$\mathsf{LVfy}(\widehat{\sigma}, \mathsf{vk}, m, j, \mathsf{aux}) \to 0/1$. The local verifier runs $\mathsf{lvB.LVfy}$, and outputs $\mathsf{lvB.LVfy}(\mathsf{crs}, x = (\mathsf{vk}, m), j, \widehat{\sigma}, \mathsf{aux})$.

Multi-hop aggregation. We want to point out that unlike the recent work of Devadas et al. [DGKV22], our construction described above doesn't directly satisfy unbounded multi-hop aggregation. However, by relying on a rate-1 batch arguments as a starting point, our construction satisfies multi-hop aggregation as well.

Correctness and Compactness. The correctness and compactness follows directly from the correctness and compactness properties of the signature scheme $\mathcal{S}$ and BARG scheme lv-BARG. Note that in the compactness definition, we drop the dependence upon k since our batch arguments incur only poly-logarithmic dependence on k, and since k will always be a polynomial in λ, thus we simply write it as a fixed polynomial in λ.

12.1 Security

Theorem 12.1. If the signature scheme $\mathcal{S}$ is unforgeable as per Definition 11.1, and locally verifiable BARG scheme lv-BARG satisfies index hiding, somewhere argument of knowledge, local argument of knowledge with adversarial opening, and instance privacy properties (Definitions 3.1, 3.2, 5.3 and 5.5), then the designed scheme Agg is an aggregate signature scheme satisfying (plain) unforgeability, aggregated unforgeability, aggregated unforgeability with adversarial openings, and message privacy with local openings as per Definitions 11.1 and 11.7 to 11.9.

Proof. The proof is divided into multiple parts where we individually prove the desired properties.

(Plain) Unforgeability. This follows directly from the unforgeability of $\mathcal{S}$. Concretely, we have the following.

Lemma 12.2. If the base signature scheme $\mathcal{S}$ satisfies unforgeability, then Agg also satisfies (plain) unforgeability.

Proof. Since the Agg scheme is identical to $\mathcal{S}$ w.r.t. $\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}$, thus the unforgeability of Agg follows from a straightforward reduction to $\mathcal{S}$. □

Aggregated Unforgeability. This follows from the unforgeability of $\mathcal{S}$ and somewhere argument of knowledge and index hiding of lv-BARG. Concretely, we have the following.

Lemma 12.3. If the base signature scheme $\mathcal{S}$ satisfies unforgeability and the BARG scheme lv-BARG is a somewhere argument of knowledge and satisfies index hiding, then Agg satisfies aggregated unforgeability.

Proof. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks aggregated unforgeability of the Agg scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. That is, $\mathcal{A}$ finds a valid forgery $(\{(\mathsf{vk}_i^*, m_i^*)\}_i, \widehat{\sigma}^*)$ with probability ϵ such that there exists an index $j^* \in [k]$ where $\mathsf{vk}_{j^*}^*$ is the challenger's verification key, and $m_{j^*}^*$ was not queried by $\mathcal{A}$ to the challenger. While there might exist multiple such indices, we use j^* to denote the smallest such index. We start by defining simple hybrid experiments to complete the proof.

Experiment 0. This is the aggregate unforgeability security game, and we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins.

Experiment 1. In this experiment, the challenger samples a random index $\hat{i} \leftarrow [k]$, and plays the rest of aggregate unforgeability game with $\mathcal{A}$ as is. In the end, we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins *and* $\hat{i} = j^*$ (that is, the challenger guesses the forgery index correctly).

Experiment 2. This is identical to the previous experiment, except the challenger uses $\hat{i}$ as the extraction index in the lv-BARG setup. Namely, it runs $(\mathsf{lvbg.crs}, \mathsf{lvbg.td}) \leftarrow \mathsf{lvB.Setup}(1^\lambda, 1^k, 1^n, \hat{i})$.

It plays the rest of the experiment as before. As in previous experiment, we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins and $\hat{i} = j^*$.

Experiment 3. This is same as previous experiment, except if $\mathcal{A}$ wins and $\hat{i} = j^*$, then the challenger runs $\mathsf{lvB.Extract}$, the $\mathsf{lv\text{-}BARG}$ extractor, as follows:

$$\sigma^* = \mathsf{lvB.Extract}(\mathsf{lvbg.td}, \{x_i = (\mathsf{vk}_i^*, m_i^*)\}_i, \widehat{\sigma}^*).$$

We say the output of the experiment is 1 if and only if $(m_{\hat{i}}^*, \sigma^*)$ is a valid forgery w.r.t. challenge key vk.

Let $\text{EXPT}_i^{\mathcal{A}}(1^\lambda)$ denote the output of the experiment i. Next, we prove the following sequence of claims regarding the above experiments.

Claim 12.4. For every adversary $\mathcal{A}$, for every $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_1^{\mathcal{A}}(1^\lambda)\right] \geq \frac{\Pr\left[\text{EXPT}_0^{\mathcal{A}}(1^\lambda)\right]}{k}.$$

Proof. This is an information theoretic argument, and follows directly from the fact that the challenger samples $\hat{i}$ as a random index, and the guess is correct with probability at least $1/k$. Thus, the output of the experiment 1 will be as 1 with probability at most k times smaller than the probability in experiment 0. □

Claim 12.5. If the batch argument scheme $\mathsf{lv\text{-}BARG}$ satisfies index hiding, then for every PPT $\mathcal{A}$ playing the above aggregated unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_1^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\text{EXPT}_2^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. This follows directly from the index hiding property. Note that in both the experiments, the challenger samples two random indices i^* and $\hat{i}$. In experiment 1, it uses i^* as the extraction index in $\mathsf{lv\text{-}BARG}$, while in experiment 2, it uses $\hat{i}$. The reduction algorithm simply samples two random indices i^* and $\hat{i}$, and send them to the index hiding challenger. It receives a crs $\mathsf{lvbg.crs}$, and runs the rest of the experiment honestly. In the end, if $\mathcal{A}$ outputs a valid forgery such that $j^* = \hat{i}$ then it guesses i^* was used, otherwise it guesses $\hat{i}$ was used. Note that the challenger can find j^* since it is simply the smallest index such that $\mathsf{vk}_{j^*}^*$ is the challenge verification key and $m_{j^*}^*$ was not sign queried. Thus, if $\mathcal{A}$ can distinguish between the two experiment with non-negligible probability, then the reduction algorithm wins in the index hiding game with same probability. □

Claim 12.6. If the batch argument scheme $\mathsf{lv\text{-}BARG}$ satisfies somewhere argument of knowledge property, then for every PPT $\mathcal{A}$ playing the above aggregated unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_2^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. This follows directly from the somewhere argument of knowledge property. Note that the only difference in the two games is the way the winning condition is defined. In experiment 2, it is defined as the attacker finding an accepting batch proof $\widehat{\sigma}^*$ where the $\hat{i}^{th}$ instance contans the

challenge verification key and a non-sign-queried message. Whereas in experiment 3, it is changed so that the attacker wins if the extracted witness σ^* is an accepting signature for the non-sign-queried message under the challenge verification key. Observe that if the output of the experiment is noticeably different, then $\mathcal{A}$ can be used to break somewhere argument of knowledge property directly. We can design a reduction algorithm $\mathcal{B}$ as follows– $\mathcal{B}$ samples $\hat{i}$ at random, and sends $\hat{i}$ as the challenge index to the $\mathsf{lv\text{-}BARG}$ challenger; $\mathcal{B}$ receives $\mathsf{lvbg.crs}$ from the challenger, and generates a challenge verification-signing key pair $(\mathsf{vk}, \mathsf{sk})$, and sends $\mathsf{lvbg.crs}, \mathsf{vk}$ to $\mathcal{A}$; $\mathcal{B}$ then answers each sign query honestly; and finally, when the attacker outputs a forgery, $\mathcal{B}$ checks that $\hat{i} = j^*$ (i.e., $\hat{i}$ was the correct guess) and submits $(\{x_i = (\mathsf{vk}_i^*, m_i^*)\}_i, \pi = \widehat{\sigma}^*)$ as its attack on the somewhere argument of knowledge property. Note that whenever $\mathcal{A}$ distinguishes between the two experiment with non-negligible probability, then the reduction algorithm wins in the somewhere argument of knowledge game with same probability. Thus, the claim follows. □

Claim 12.7. If the signature scheme $\mathcal{S}$ satisfies unforgeability, then for every $\mathcal{A}$ playing the above aggregated unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. Suppose $\Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] = \delta$ for some non-negligible probability $\delta = \delta(\lambda)$. We now describe a reduction algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the unforgeability property of the signature scheme $\mathcal{S}$ with probability at least δ.

The challenger corresponding to scheme $\mathcal{S}$ samples a verification-signing key pair, and sends vk to the reduction algorithm $\mathcal{B}$. $\mathcal{B}$ then samples a random index $\hat{i} \leftarrow [k]$, and samples the crs as $(\mathsf{lvbg.crs}, \mathsf{lvbg.td}) \leftarrow \mathsf{lvB.Setup}(1^\lambda, 1^k, 1^n, \hat{i})$. It sends $\mathsf{lvbg.crs}, \mathsf{vk}$ to the adversary. $\mathcal{B}$ then answers $\mathcal{A}$'s signing queries by forwarding them to the signature scheme challenger, and relaying the challenger's response back to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a forgery $(\{(\mathsf{vk}_i^*, m_i^*)\}_i, \widehat{\sigma}^*)$, and sends it to $\mathcal{B}$.

$\mathcal{B}$ computes the forgery as $\sigma^* = \mathsf{lvB.Extract}(\mathsf{lvbg.td}, \{x_i = (\mathsf{vk}_i^*, m_i^*)\}_i, \widehat{\sigma}^*)$. The reduction algorithm $\mathcal{B}$ finally submits $m_{\hat{i}}^*$ and σ^* as its forgery.

Note that $\mathcal{B}$ wins the unforgeability game with the challenger for signature scheme $\mathcal{S}$ with probability δ. This is because, by definition of experiment 3, the experiment outputs 1 if and only if (m^*, σ^*) is a valid forgery w.r.t. challenge key vk. Thus, the claim follows. □

Combining all the above claims, we obtain that $\Pr\left[\text{EXPT}_0^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda)$ as k is a polynomial in the security parameter. Thus, proof of aggregate unforgeability follows □

Aggregated Unforgeability with Adversarial Openings. This follows from the unforgeability of $\mathcal{S}$ and local argument of knowledge with adversarial openings and index hiding of $\mathsf{lv\text{-}BARG}$. Concretely, we have the following.

Lemma 12.8. If the base signature scheme $\mathcal{S}$ satisfies unforgeability and the BARG scheme $\mathsf{lv\text{-}BARG}$ is a local argument of knowledge with adversarial openings and satisfies index hiding, then Agg satisfies aggregated unforgeability with adversarial openings.

Proof. The proof of this lemma is similar to the proof of Lemma 12.3, where the somewhere extraction step is replaced with the local extraction provided by $\mathsf{lv\text{-}BARG}$. Below we highlight the main changes.

Suppose there exists a PPT attacker $\mathcal{A}$ that breaks aggregated unforgeability with adversarial openings of the Agg scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. That is, $\mathcal{A}$ finds a valid forgery $(\widehat{\sigma}, \mathsf{aux}, m, j)$ with probability ϵ where m was not queried by $\mathcal{A}$ to the challenger. Below we describe hybrid experiments to complete the proof.

Experiment 0. This is the aggregate unforgeability with adversarial opening security game, and we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins.

Experiment 1. In this experiment, the challenger samples a random index $\hat{i} \leftarrow [k]$, and plays the rest of aggregate unforgeability game with $\mathcal{A}$ as is. In the end, we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins *and* $\hat{i} = j$ (that is, the challenger guesses the local forgery index correctly).

Experiment 2. This is identical to the previous experiment, except the challenger uses $\hat{i}$ as the extraction index in the $\mathsf{lv\text{-}BARG}$ setup. Namely, it runs $(\mathsf{lvbg.crs}, \mathsf{lvbg.td}) \leftarrow \mathsf{lvB.Setup}(1^\lambda, 1^k, 1^n, \hat{i})$. As before, we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins and $\hat{i} = j$.

Experiment 3. This is same as previous experiment, except if $\mathcal{A}$ wins and $\hat{i} = j$, then the challenger runs $\mathsf{lv\text{-}BARG.LExtract}$, the $\mathsf{lv\text{-}BARG}$ local extractor, as follows:

$$\sigma = \mathsf{lv\text{-}BARG.LExtract}(\mathsf{lvbg.td}, x = (\mathsf{vk}, m), \widehat{\sigma}, \mathsf{aux}).$$

We say the output of the experiment is 1 if and only if (m, σ) is a valid forgery w.r.t. challenge key vk.

Let $\text{EXPT}_i^{\mathcal{A}}(1^\lambda)$ denote the output of the experiment i. Next, we prove the following sequence of claims regarding the above experiments.

Claim 12.9. For every adversary $\mathcal{A}$, for every $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_1^{\mathcal{A}}(1^\lambda)\right] \geq \frac{\Pr\left[\text{EXPT}_0^{\mathcal{A}}(1^\lambda)\right]}{k}.$$

Proof. This is an information theoretic argument, and follows directly from the fact that the challenger samples $\hat{i}$ as a random index, and the guess is correct with probability at least $1/k$. □

Claim 12.10. If the batch argument scheme $\mathsf{lv\text{-}BARG}$ satisfies index hiding, then for every PPT $\mathcal{A}$ playing the above aggregated unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_1^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\text{EXPT}_2^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. This is similar to the proof of Claim 12.5. □

Claim 12.11. If the batch argument scheme $\mathsf{lv\text{-}BARG}$ satisfies local argument of knowledge property, then for every PPT $\mathcal{A}$ playing the above aggregated unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_2^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. This is similar to the proof of Claim 12.6, with the only difference that the reduction algorithm submits $(x = (\mathsf{vk}, m), \pi = \widehat{\sigma}, \mathsf{aux})$ as its attack on the local argument of knowledge property. □

Claim 12.12. If the signature scheme $\mathcal{S}$ satisfies unforgeability, then for every $\mathcal{A}$ playing the above aggregated unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. This is similar to the proof of Claim 12.7, with the only difference that the reduction algorithm runs the lv-BARG local extractor instead to extract the forged signature from the batch proof. □

□

Message Privacy with Local Openings. This follows from the instance privacy of lv-BARG. Concretely, we have the following.

Lemma 12.13. If the BARG scheme lv-BARG satisfies instance privacy, then Agg satisfies message privacy with local openings.

Proof. This follows immediately from a straightforward reduction to the instance privacy security of lv-BARG. The simulator for message privacy simply runs the instance privacy simulator with instance $x_j = (\mathsf{vk}_j, m_j)$ and $\omega_j = \sigma_j$. Also, note that the reduction algorithm chooses a random index i^* and sends it to lv-BARG challenger. Note that i^* does not affect any other part of the reduction. □

This completes the proof of our main theorem. □

13 Application 2: Redactable Signatures

In this section, we construct a redactable signature scheme from any de-BARG scheme.

Construction. Let $\mathcal{S} = (\mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify})$ be any plain signature scheme with message space $\{0,1\}^{2\lambda}$, and de-BARG $= (\mathsf{deB.Setup}, \mathsf{deB.Prove}, \mathsf{deB.Delete}, \mathsf{deB.Verify})$ be a deletable BARG scheme for language $\mathcal{L}_{\mathcal{S}}$ (Fig. 5). Below we describe our redactable signature scheme $\mathcal{S}^{\mathsf{red}}$.

Language $\mathcal{L}_{\mathcal{S}}$

Instance: $\hat{x} :=$ (key vk, index i, bit b, tag τ).

Witness: $\hat{\omega} :=$ signature σ w.r.t. vk.

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \mathcal{L}_{\mathcal{S}}$ if σ is a valid signature of message (τ, i, b) under vk. That is, $\mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (\tau, i, b), \sigma) = 1$.

Figure 5: Language $\mathcal{L}_{\mathcal{S}}$ for the de-BARG.

$\mathsf{Setup}(1^\lambda, L) \to (\mathsf{vk}, \mathsf{sk})$. It runs the signature setup algorithm to compute $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)$. It also samples parameters for the de-BARG scheme as $(\mathsf{crs}, *) \leftarrow \mathsf{deB.Setup}(1^\lambda, k = L, n = |\mathsf{vk}| + 2\lambda, \perp)$[17].

[17]Here by $\perp$ we mean that it picks a dummy/random extraction index, and $*$ means it ignores the trapdoor.

NOTE. Technically, the verification and signing keys contain the crs too, however we do not write it explicitly and drop it for ease of notation. Further, one could remove de-BARG setup entirely from the setup of redactable signatures, and keep it as global CRS. However, we avoid it as this would change the traditional syntax of redactable signatures.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. The signer first samples a random tag $\tau \leftarrow \{0,1\}^\lambda$, and then creates L signatures on messages $(\tau, i, m[i])$. That is, $\sigma_i \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\tau, i, m[i]))$. Next, it outputs the actual signature $\sigma = (\pi, \tau)$ where the batch proof π is computed as follows

$$\pi \leftarrow \mathsf{deB.Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[L]}), \quad \text{where } x_i = (\mathsf{vk}, i, m[i], \tau),\ \omega_i = \sigma_i.$$

$\mathsf{Redact}(\mathsf{vk}, m, S, \sigma) \to \sigma'$. Let $\sigma = (\pi, \tau)$. The redaction algorithm runs the deletion algorithm of de-BARG to compute the redacted signature as $\sigma' = (\pi^{\mathsf{red}}, \tau)$ where

$$\pi^{\mathsf{red}} \leftarrow \mathsf{deB.Delete}(\mathsf{crs}, \{x_j\}_{j\in[L]}, S, \sigma), \quad \text{where } x_j = (\mathsf{vk}, j, m[j], \tau).$$

NOTE. Observe that the schem satisfies multi-hop redaction naturally if the underlying de-BARG scheme satisfies multi-hop deletion.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma) \to 0/1$. Let $\sigma = (\pi, \tau)$, $\ell = |\{i : m[i] \neq \perp\}|$, and $i_j \in [L]$ denote the index of the j^{th} non-$\perp$ bit of m. For $j \in [\ell]$, let $x_j = (\mathsf{vk}, i_j, m[i_j], \tau)$. The verifier runs the de-BARG verifier as $\mathsf{deB.Verify}(\mathsf{crs}, \{x_j\}_j, \pi)$.

Correctness and succinctness. We now show that the above scheme satisfies correctness and succinctness. This follows directly from the regular and deletion correctness and succinctness of the BARG scheme.

In detail, first note that by correctness of the signature scheme, we have that, for every i, $\mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (\tau, i, m[i]), \sigma_i) = 1$ as defined in the signing algorithm. Therefore, the final signature which is computed as a batch proof will be accepted by the verifier as a valid signature since it simply runs the BARG verifier for the same batch of instances (since the tag τ is included as part of the signature). Moreover, by BARG succinctness and the fact that each $|\sigma_i| = \mathsf{poly}(\lambda)$, we have that $|\pi| = \mathsf{poly}(\lambda, \log L)$. Next, note that the redaction algorithm simply runs the BARG deletion algorithm, thus by deletion correctness we have that π^{red} is accepted by the BARG verifier. And, by deletion succinctness, we obtain succinctness of the redacted proof (i.e., $|\pi^{\mathsf{red}}| = \mathsf{poly}(\lambda)$) since the size of each batch proof is also only $\mathsf{poly}(\lambda)$ due to the fact that $\log L \leq \lambda$ and $|\sigma_i| = \mathsf{poly}(\lambda)$. The proof of multi-hop redaction follows analogously.

13.1 Security

Below we prove the security of our scheme.

Theorem 13.1. If the signature scheme $\mathcal{S}$ is unforgeable as per Definition 11.1, and BARG scheme $\mathsf{de\text{-}BARG}$ satisfies index hiding, somewhere argument of knowledge, zero-knowledge, and deletion privacy properties (Definitions 3.1, 3.2, 7.4 and 8.3), then the above scheme is a redactable signature scheme satisfying unforgeability, redaction unforgeability, and redaction unlinkability as per Definitions 11.1, 11.2 and 11.4.

Proof. The proof is divided into three parts where we individually prove the desired properties.

Unforgeability. This follows directly from the unforgeability of $\mathcal{S}$ and index hiding and extraction security of $\mathsf{de\text{-}BARG}$. Concretely, we have the following.

Lemma 13.2. If the base signature scheme $\mathcal{S}$ satisfies unforgeability and the BARG scheme $\mathsf{de\text{-}BARG}$ is a somewhere argument of knowledge and satisfies index hiding, then $\mathcal{S}^{\mathsf{red}}$ also satisfies (plain) unforgeability.

Proof. Suppose there exists an attacker $\mathcal{A}$ than wins in the unforgeability game with non-negligible probability $\epsilon = \epsilon(\lambda)$. That is, $\mathcal{A}$ finds a valid forgery m^*, σ^* with probability ϵ such that m^* was not queried by $\mathcal{A}$ to the challenger. Note that $\sigma^* = (\pi^*, \tau^*)$ such that $\mathsf{deB.Verify}(\mathsf{crs}, \{x_j^*\}_j, \pi^*)$ where $x_j^* = (\mathsf{vk}, j, m^*[j], \tau^*)$. Let j^* denote the smallest index such that $(\tau^*, j^*, m^*[j^*])$ was not signed by the challenger. Clearly, if $\mathcal{A}$ is an admissible attacker, then such an index j^* exists (with all-but-negligible probability). This is because with all-but-negligible probability, the tag values $\{\tau_i\}$ selected by the challenger to answer any signing query do not have a collision. That is, $\tau_{i_1} = \tau_{i_2}$ iff $i_1 = i_2$. Since the tags are randomly sampled λ-bit strings, thus this holds with probability at least $1 - \frac{q^2}{2^\lambda}$, where q is the totla number of signing queries. As q is a polynomial, we get that such a j^* exists with all-but-negligible probability. Let j^* be the smallest such index if it exists. Next, we start by defining simple hybrid experiments to complete the proof. In the following analysis, we condition on the event that such a j^* exists.

Experiment 0. This is the plain unforgeability security game, and we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins.

Experiment 1. In this experiment, the challenger samples a random index $\hat{j} \leftarrow [L]$, and plays the rest of the unforgeability game with $\mathcal{A}$ as is. In the end, we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins *and* $\hat{j} = j^*$ (that is, the challenger guesses the forgery index correctly). Since the challenger stores all the state from eevry signing query, thus it can efficiently check this.

Experiment 2. This is identical to the previous experiment, except the challenger uses $\hat{j}$ as the extraction index in the $\mathsf{de\text{-}BARG}$ setup. Namely, it runs $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{deB.Setup}(1^\lambda, k = L, n = |\mathsf{vk}| + 2\lambda, \hat{j})$. It plays the rest of the experiment as before. As in previous experiment, we say the output of the experiment is 1 if and only if $\mathcal{A}$ wins and $\hat{j} = j^*$.

Experiment 3. This is same as previous experiment, except if $\mathcal{A}$ wins and $\hat{j} = j^*$, then the challenger runs $\mathsf{deB.Extract}$, the $\mathsf{de\text{-}BARG}$ somewhere extractor, as follows:

$$\sigma^* = \mathsf{deB.Extract}(\mathsf{td}, \{x_j^* = (\mathsf{vk}, j, m^*[j], \tau^*)\}_j, \pi^*).$$

We say the output of the experiment is 1 if and only if σ^* is a valid forgery for $(\tau^*, j^*, m^*[j^*])$ w.r.t. challenge key vk.

Let $\text{EXPT}_i^{\mathcal{A}}(1^\lambda)$ denote the output of the experiment i. Next, we prove the following sequence of claims regarding the above experiments.

Claim 13.3. For every adversary $\mathcal{A}$, for every $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_1^{\mathcal{A}}(1^\lambda)\right] \geq \frac{\Pr\left[\text{EXPT}_0^{\mathcal{A}}(1^\lambda)\right]}{L}.$$

Proof. This is an information theoretic argument, and follows directly from the fact that the challenger samples $\hat{j}$ as a random index, and the guess is correct with probability at least $1/L$. Thus, the output of the experiment 1 will be as 1 with probability at most L times smaller than the probability in experiment 0. □

Claim 13.4. If the batch argument scheme $\mathsf{de\text{-}BARG}$ satisfies index hiding, then for every PPT $\mathcal{A}$ playing the above unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_1^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\text{EXPT}_2^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. This follows directly from the index hiding property. Note that in both the experiments the only difference is that in experiment 1, challenger uses $\perp$ (dummy index) as the extraction index in $\mathsf{de\text{-}BARG}$, while in experiment 2, it uses $\hat{j}$ which is sample randomly. A reduction algorithm can simply sample a random index $\hat{j}$, and send this along with the dummy index to the index hiding challenger. It receives a crs crs, and runs the rest of the experiment honestly. In the end, if $\mathcal{A}$ outputs a valid forgery such that $j^* = \hat{j}$ then it guesses $\perp$ was used, otherwise it guesses $\hat{j}$ was used. As discussed previously, the reduction algorithm can efficiently find j^*, thus if $\mathcal{A}$ can distinguish between the two experiments with non-negligible probability, then the reduction algorithm wins in the index hiding game with same probability. □

Claim 13.5. If the batch argument scheme $\mathsf{de\text{-}BARG}$ satisfies somewhere argument of knowledge property, then for every PPT $\mathcal{A}$ playing the above unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_2^{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. This follows directly from the somewhere argument of knowledge property. Note that the only difference in the two games is the way the winning condition is defined. In experiment 2, it is defined as the attacker finding an accepting batch proof π^*; while in experiment 3, it is changed so that the attacker wins if the extracted witness σ^* is a valid forgery. Observe that if the output of the experiment is noticeably different, then $\mathcal{A}$ can be used to break somewhere argument of knowledge property directly. □

Claim 13.6. If the signature scheme $\mathcal{S}$ satisfies unforgeability, then for every $\mathcal{A}$ playing the above unforgeability game, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that

$$\Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda).$$

Proof. Suppose $\Pr\left[\text{EXPT}_3^{\mathcal{A}}(1^\lambda)\right] = \delta$ for some non-negligible probability $\delta = \delta(\lambda)$. We now describe a reduction algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the unforgeability property of the signature scheme $\mathcal{S}$ with probability at least δ.

The challenger corresponding to scheme $\mathcal{S}$ samples a verification-signing key pair, and sends vk to the reduction algorithm $\mathcal{B}$. $\mathcal{B}$ then samples a random index $\hat{j} \leftarrow [L]$, and samples the crs as in experiment 3. It sends $\mathsf{crs}, \mathsf{vk}$ to the adversary. $\mathcal{B}$ then answers $\mathcal{A}$'s signing queries by first sampling a random tag τ_i for each query, and then forwarding them to the signature scheme challenger by splitting the message bit-by-bit, and after receing the challenger's response, it creates batch proof π_i honestly and sends back the proof π_i and τ_i to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs a forgery, and sends it to $\mathcal{B}$.

$\mathcal{B}$ computes the forgery as σ^* as in experiment 3. The reduction algorithm $\mathcal{B}$ finally submits $(\tau^*, j^*, m^*[j^*])$ and σ^* as its forgery.

Note that $\mathcal{B}$ wins the unforgeability game with the challenger for signature scheme $\mathcal{S}$ with probability δ. This is because, by definition of experiment 3, the experiment outputs 1 if and only if this is a valid forgery w.r.t. challenge key vk. Thus, the claim follows. □

Combining all the above claims, we obtain that $\Pr\left[\text{EXPT}_0^{\mathcal{A}}(1^\lambda)\right] \leq \mathsf{negl}(\lambda)$ as k is a polynomial in the security parameter. Thus, proof of (plain) unforgeability follows □

Redaction Unforgeability. This follows from the unforgeability of $\mathcal{S}$ and somewhere argument of knowledge, index hiding, and zero-knowledge of de-BARG. Concretely, we have the following.

Lemma 13.7. If the base signature scheme $\mathcal{S}$ satisfies unforgeability and the BARG scheme de-BARG is a somewhere argument of knowledge and satisfies index hiding and zero-knowledge, then $\mathcal{S}^{\mathsf{red}}$ satisfies redacted unforgeability.

Proof. The proof is nearly identical to the proof of Lemma 13.2, except the challenger now also answers queries to the Redact and Request oracle. In order to answer such queries, we first use the zero-knowledge property of the de-BARG scheme to replace the output of each Request oracle with a simulated proof. Crucially, the challenger does not perform any action for a Sign or Redact query, but it simply stored the query in its internal state and answers with a random handle to the query. Whenever $\mathcal{A}$ makes a Request query, at that point the challenger answers with a simulated proof, and stores the proof (along with the lazily sampled tag) in its memory for that handle. First, we show that no PPT attacker can distinguish between the original security game where queries are answered honestly, and this intermediate simulated game where the queries are answered lazily and using a simulator. Once we make this change, we follow the proof strategy as in Lemma 13.2. Note that since the queries are answered lazily and after simulation, thus the same definition of j^* still holds and proof goes as before. □

Redaction Unlinkability. This follows from the deletion privacy of de-BARG. Concretely, we have the following.

Lemma 13.8. If the BARG scheme de-BARG satisfies deletion privacy, then $\mathcal{S}^{\mathsf{red}}$ satisfies redaction unlinkability.

Proof. This follows immediately from the deletion privacy property of the de-BARG scheme. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks redaction unlinkability of the $\mathcal{S}^{\mathsf{red}}$ scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. Then we construct another PPT attacker $\mathcal{B}$ that breaks deletion privacy property of lv-BARG.

Briefly, the attacker $\mathcal{A}$ starts by outputting the message length L. The reduction algorithm $\mathcal{B}$ samples a signing key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)$, and sends $(k = L, n = |\mathsf{vk}| + 2\lambda)$ to the de-BARG challenger. The challenger sends crs to $\mathcal{B}$, and it then sends $\mathsf{crs}, \mathsf{vk}$ to $\mathcal{A}$. $\mathcal{A}$ then makes polynomially many signing queries $m_1, \ldots, m_q$ for some polynomial function $q = q(\lambda)$. $\mathcal{B}$ answers each query honestly, which is it samples a random tag τ_j, and creates L signatures on messages $(\tau_j, i, m_j[i])$, and computes the final signature as $\sigma_j = (\pi_j, \tau_j)$ where the batch proof π_j is a batch proof as computed in the scheme. Next, $\mathcal{A}$ outputs a redaction set S and two messages m_0^* and m_1^*. $\mathcal{B}$ then samples a random tag τ^*, and creates signatures as follows

$$\forall i \in S, b \in \{0,1\}, \quad \sigma_{i,b} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\tau^*, i, m_b^*[i]))$$
$$\forall i \notin S, b \in \{0,1\}, \quad \sigma_{i,0} = \sigma_{i,1} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\tau^*, i, m_0^*[i]))$$

Note that by admissibility of the attacker, we know that $m_0^{*\{S\}} = m_1^{*\{S\}}$, thus $m_0^*[i] = m_1^*[i]$ for $i \notin S$. Next, $\mathcal{B}$ sends the instance-witness pairs $\{(x_{i,b}, \omega_{i,b})\}_{i,b}$ to the challenger, where $x_{i,b} =$

$(\mathsf{vk}, i, m_b^*[i], \tau^*)$ and $\omega_{i,b} = \sigma_{i,b}$. Note that by our construction, we have that $(x_{i,0}, \omega_{i,0}) = (x_{i,1}, \omega_{i,1})$ for $i \notin S$. Thus, $\mathcal{B}$'s choice of instance-witness pairs are admissible as per the deletion privacy game. The challenger then sends a redacted proof π^{red}, and $\mathcal{B}$ then sends $(\pi^{\mathsf{red}}, \tau^*)$ as the redacted signature to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs its guess, and $\mathcal{B}$ forwards the same bit as its own guess.

We claim that if $\mathcal{A}$ wins with non-negligible probability, then so does $\mathcal{B}$ in the deletion privacy game. This follows from a simple observation that since a tag τ is randomly selected during the signing process (independent of the input message), thus we could prove a simple statistical intermediate lemma where we argue that $\mathcal{A}$'s advantage is identical even when the challenger uses a single tag τ^* to create original (unredacted) signatures for challenge messages. Next, since we can argue in an additional intermediate step that $\mathcal{A}$ cannot distinguish in the cases where the actual signatures on the unredacted portions of the challenge messages are the exavtly same signature. This can be proved by either derandomizing the signing process (which follow w.l.o.g.), or from the statistical fact that no (even unbounded time) adversary can distinguish between two i.i.d. random variables. Note that the distribution of signatures $\sigma_{i,0}$ and $\sigma_{i,1}$ are i.i.d. since the tag value is the same. This completes the proof of the lemma. □

This completes the proof of our main theorem. □

14 Application 3: Homomorphic Signatures

Finally, we provide new constructions for homomorphic signatures as a consequence of mutable batch proofs for $\mathcal{C}$-batchNP mutation operations. We start by describing a homomorphic signature scheme that can evaluate all polynomial-size *formulae* (thus, all log-depth circuits) from any mutable batch proof for $\mathcal{C}$-batchNP mutation class, where $\mathcal{C}$ contains all monotone circuits. Next, we show how to modify the construction to support evaluation of all polynomial-size monotone circuits using the same tool of mutable batch proofs.

14.1 Homomorphic signatures for log-depth circuits

Here we provide a construction for homomorphic signatures for the family of all monotone *formulae* of fixed size. Our construction naturally provides many new desirable features for homomorphic signatures such as the signatures can be aggregated as well as locally verified. This leads to much smaller signatures than previously known. This is because our construction builds on mutable batch proofs for $\mathcal{C}$-batchNP which, as discussed in Remark 10.3, provides batch proving as well as local verifiability of batch proofs.

Important notation. In order to explain our construction, we need to set up some useful notation. Consider any boolean *formula* family $\mathcal{C} = \{\mathcal{C}_k\}_{k\in\mathbb{N}}$, where each boolean formula in $\mathcal{C}_k$ take k-bits as inputs. Note that any log-depth ($\mathbf{NC}^1$) circuit can be written as a polynomial-sized (non-monotone) boolean formulae. (See [GLW21, §8.2] for further discussion.) This is well-known folklore transformation.

One of our main technique is to rely on another folklore approach to transform any non-monotone boolean formulae can be turned into a monotone boolean formulae by introducing more input variables. Concretely, it is known that, by using De Morgan's identities, one can deterministically translate a non-monotone formula C with n-bit inputs into a monotone formula $C^{\neg}$ that reads $2n$-bits of inputs, but *does not contain any negation gates*. This property has been used in

many prior works such as attribute-based encryption systems [GPSW06]. (See [GLW21, §8.2] for further discussion.)

The special property of $C^{\neg}$ is that each original input bit is now encoded as two inputs, thus leading to doubling the input length. More formally, we know that

$$\forall x \in \{0,1\}^k, C \in \mathcal{C}_k, \qquad C(x) = C^{\neg}(x_1, x_1 \oplus 1, \ldots, x_k, x_k \oplus 1).$$

Basically, if the i^{th} input bit of C is zero, then the $2i^{th}$-bit of expanded input is 0, while $(2i-1)^{th}$-bit of expanded input is 1, and vice versa. In our homomorphic signature construction, we use such "monotone formula representations" for every non-monotone formula.

Language $\mathcal{L}_{\mathsf{vk}}$

Instance: $\hat{x} :=$ (index i, message bit b).

Witness: $\hat{\omega} :=$ signature σ w.r.t. vk.

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \mathcal{L}_{\mathsf{vk}}$ if σ is a valid signature of (i, b) under vk. That is, $\mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, b), \sigma) = 1$.

Figure 6: Language $\mathcal{L}_{\mathsf{vk}}$ for the BARG.

Construction. Let $\mathcal{S} = (\mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify})$ be any regular signature scheme, and $\mathsf{mut\text{-}BARG} = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{Eval}', \mathsf{Post\text{-}Verify}')$ be a mutable batch proof for language $\mathcal{L}_{\mathsf{vk}}$ (described in Fig. 7) with mutation class monotone-batchNP. We construct homomorphic (aggregate) signatures for all (non-monotone) formulae $\mathcal{C}$ below.

$\mathsf{Setup}(1^\lambda, \mathcal{C}_k) \to (\mathsf{vk}, \mathsf{sk})$. It runs BARG setup to compute $\mathsf{crs} \leftarrow \mathsf{Setup}'(1^\lambda, 1^{2k}, 1^n, \mathcal{C}_k^{\neg})$, where n is signature length in $\mathcal{S}$, and $\mathcal{C}_k^{\neg}$ denotes the monotone representation of $\mathcal{C}_k$. It also samples a key pair as $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)$. Set $\mathsf{vk} = (\mathsf{crs}, \mathsf{vk}')$ and $\mathsf{sk} = \mathsf{sk}'$.

$\mathsf{Sign}(\mathsf{sk}, i, b) \to \sigma$. Same as $\mathcal{S}.\mathsf{Sign}$. That is, $\sigma \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (i, b))$.

$\mathsf{Agg}(\mathsf{vk}, \{(m_i, \sigma_i)\}_i) \to \widehat{\sigma}$. It computes the aggregated signature as a BARG proof with instance $x_i = (i, m_i)$, and corresponding witness $\omega_i = \sigma_i$ for all i. Thus, it outputs $\widehat{\sigma} \leftarrow \mathsf{Prove}'(\mathsf{crs}, \{(x_i, \omega_i)\}_i)$.

$\mathsf{Eval}(\mathsf{vk}, M = (m_i)_i, \widehat{\sigma}, C) \to \sigma'$. Here we assume that the evaluator gets an aggregated signature, instead of plain signatures for each message bit. This is not an additional assumption as the evaluator could simply run the Agg algorithm first.

Let $C^{\neg}$ correspond to the monotone representation of C as we discussed above. Also, let $X_M = ((1, m_1), \ldots, (k, m_k))$ and $X_{\overline{M}} = ((1, m_1 \oplus 1), \ldots, (k, m_k \oplus 1))$. It runs BARG proof evaluation to compute evaluated signature as $\sigma' \leftarrow \mathsf{Eval}'(\mathsf{crs}, C^{\neg}, \{(\lceil j/2 \rceil, X_j, \widehat{\sigma})\}_{j \le 2k})$, where $j \in [2k]$ because $C^{\neg}$ has double the input length, and the instance list X_j is defined as:

$$X_j = \begin{cases} X_M & \text{if } m_{\lceil j/2 \rceil} = j \bmod 2, \\ X_{\overline{M}} & \text{otherwise.} \end{cases}$$

In words, the instance list X_{2i-1} is set to be the input sequence of signed index-message pairs X_M when i^{th} message bit m_i is 1, else it is set as the complement of the signed index-message

pairs X^*. Similarly, X_{2i} is set in the reverse manner. This ensure that the non-monotone message gets correctly encoded into its monotone representation. The main observation is that the mutated instance x_C is simply the fixed sequence $(C,(1,0),(1,1),(2,0),\ldots,(k,1))$ for every choice of M and C.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma', C) \to 0/1$. The verification algorithm simply runs the (post-mutation) verifier to output $\mathsf{Post\text{-}Verify}'(\mathsf{crs}, C, x_C, \sigma')$, where $x_C = (C,(1,0),(1,1),(2,0),\ldots,(k,1))$.

Correctness and compactness. This follows directly from the correctness and compactness of the underlying signature and mutable BARG scheme. The central fact we use is that for every j, the $\lceil j/2 \rceil^{th}$ instance in list X_j is $(\lceil j/2 \rceil, j \bmod 2)$, and that $C(m) = C^{\neg}(m_1, m_1 \oplus 1, \ldots, m_k, m_k \oplus 1)$ for every message m.

Homomorphic unforgeability. Below we prove homomorphic unforgeability of our scheme.

Theorem 14.1. If the mutable BARG scheme $\mathsf{mut\text{-}BARG}$ satisfies non-adaptive soundness (Definition 4.2) and somewhere extractability (Remark 10.5), and the signature scheme $\mathcal{S}$ unforgeable (Definition 11.1), then the above scheme is a homomorphic signature scheme satisfying circuit-selective homomorphic unforgeability (Definition 11.11).

Proof. The proof is similar to the proof of Theorem 12.1, with minor changes. Let us first describe the main difference. Experiment 0 is defined as the original homomorphic unforgeability experiment. Next, in experiment 1, we use partial extractability property of our mutable batch proofs (which is implied by the knowledge soundness of the underlying SNARG scheme) and extract the intermediate sequence of witnesses which correspond to the local opening proofs for the appropriate group of instances, and we abort the experiment if any of the individual extracted witnesses are invalid. As mentioned above, in our mutable batch proof construction, this property follows from the knowledge soundness property.

The rest of the proof is same as that of Theorem 12.1. That is, we define experiment 2, where the challenger guesses the index of the instance for which the challenger did not create a signature for. Such an index exists since otherwise the adversary is not an admissible adversary as per the homomorphic unforgeability game. We say that if our guess is wrong, then we abort. One can argue that this only causes a polynomial loss in the advantage. Next, we define experiment 3, where we use the somewhere extractability feature as discussed in Remark 10.5. (Note that this can be reduced to the index hiding of lv-BARGs used in our mutable BARG scheme.) Finally, we define experiment 4, where we use such a somewhere extractor to find a forgery and break signature unforgeability. This concludes the proof sketch. We want to highlight this analysis can only prove circuit-selective homomorphic unforgeability since the extractor needs to know the exact circuit at the beginning (refer to [BBK+23] for more details). □

Privacy. Similarly, we can prove privacy of our mutable BARGs scheme.

Theorem 14.2. If the mutable BARG scheme $\mathsf{mut\text{-}BARG}$ satisfies mutation privacy (Definition 4.3), then the above scheme is a homomorphic signature scheme satisfying context hiding (Definition 11.12).

Proof. The proof follow from a straightforward reduction to $\mathsf{mut\text{-}BARG}$ mutation privacy property. □

Instantiating under LWE. Again by above, we obtain the following.

Corollary 14.3. Assuming the polynomial hardness of learning with errors (LWE), there exist homomorphic signatures satisfying circuit-selective homomorphic unforgeability and context hiding for all polynomial-size formulae (thus, NC^1 circuits).

14.2 Homomorphic signatures for monotone circuits

Here we provide a construction for homomorphic signatures for the family of all monotone circuits.

Language $\mathcal{L}_{\mathsf{vk}}$

Instance: $\hat{x} :=$ index i.

Witness: $\hat{\omega} :=$ signature σ w.r.t. vk.

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \mathcal{L}_{\mathsf{vk}}$ if σ is a valid signature of $(i, 1)$ under vk. That is, $\mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, 1), \sigma) = 1$.

Figure 7: Language $\mathcal{L}_{\mathsf{vk}}$ for the BARG.

Construction. Consider a monotone circuit family $\mathcal{C} = \{\mathcal{C}_k\}_{k\in\mathbb{N}}$, where each monotone circuit in $\mathcal{C}_k$ take k-bits as inputs. Let $\mathcal{S} = (\mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify})$ be any regular signature scheme, and $\mathsf{mut\text{-}BARG} = (\mathsf{Setup}', \mathsf{Prove}', \mathsf{Verify}', \mathsf{Eval}', \mathsf{Post\text{-}Verify}')$ be a mutable batch proof for language $\mathcal{L}_{\mathsf{vk}}$ (described in Fig. 4) with mutation class $\mathcal{C}\text{-}\mathsf{batchNP}$. We construct homomorphic (aggregate) signatures for monotone circuit class $\mathcal{C}$ below.

$\mathsf{Setup}(1^\lambda, \mathcal{C}_k) \to (\mathsf{vk}, \mathsf{sk})$. It runs BARG setup to compute $\mathsf{crs} \leftarrow \mathsf{Setup}'(1^\lambda, 1^k, 1^n, \mathcal{C}_k)$, where n is signature length in $\mathcal{S}$. It also samples a key pair as $(\mathsf{vk}', \mathsf{sk}') \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)$. Set $\mathsf{vk} = (\mathsf{crs}, \mathsf{vk}')$ and $\mathsf{sk} = \mathsf{sk}'$.

$\mathsf{Sign}(\mathsf{sk}, i, b) \to \sigma$. Same as $\mathcal{S}.\mathsf{Sign}$. That is, $\sigma \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (i, b))$.

$\mathsf{Agg}(\mathsf{vk}, \{(m_i, \sigma_i)\}_i) \to \widehat{\sigma}$. It computes the aggregated signature as a BARG proof with instance $x_i = i$, and corresponding witness $\omega_i = \sigma_i$, for all i such that $m_i = 1$. Thus, it outputs $\widehat{\sigma} \leftarrow \mathsf{Prove}'(\mathsf{crs}, \{(i, \sigma_i)\}_{i:m_i=1})$.

$\mathsf{Eval}(\mathsf{vk}, M = (m_i)_i, \widehat{\sigma}, C) \to \sigma'$. Here we assume that the evaluator gets an aggregated signature, instead of plain signatures for each message bit. This is not an additional assumption as the evaluator could simply run the Agg algorithm first.

It runs BARG proof evaluation to compute evaluated signature as $\sigma' \leftarrow \mathsf{Eval}'(\mathsf{crs}, C, \{(j_i, X_i, \widehat{\sigma})\}_{i\le k})$, where the index and instance list are defined as follows. If $m_i = 0$, then $j_i = i, X_i = (1, 2, \ldots, k)$. Else if $m_i = 1$, then $X_i = X_M$ where X_M denotes the ordered list of all indices $i \in [k]$ such that $m_i = 1$, and j_i is the index of the i in that list. As in previous construction, the main feature of defining indices and instance list this way is that the mutated instance x_C is simply the fixed sequence $(C, 1, 2, \ldots, k)$ for every choice of M and C.

$\mathsf{Verify}(\mathsf{vk}, m, \sigma, C) \to 0/1$. The verification algorithm simply runs the (post-mutation) verifier to output $\mathsf{Post\text{-}Verify}'(\mathsf{crs}, C, x_C, \sigma')$, where $x_C = (C, 1, 2, \ldots, k)$.

The correctness, compactness, homomorphic unforgeability, and privacy of the above construction follows similar to the previous construction. This gives the following result by combining known results.

Corollary 14.4. Assuming the polynomial hardness of learning with errors (LWE), there exist homomorphic signatures satisfying circuit-selective homomorphic unforgeability and context hiding for all polynomial-size monotone circuits.

Acknowledgements

We thank Vinod Vaikuntanathan and Yael Kalai for initial discussions on the topic of local verification of BARGs. We also thank anonymous reviewers.

References

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic macs: Mac-based integrity for network coding. In *Applied Cryptography and Network Security: 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings 7*, pages 292–305. Springer, 2009.

[ADKL19] Prabhanjan Ananth, Apoorvaa Deshpande, Yael Tauman Kalai, and Anna Lysyanskaya. Fully homomorphic nizk and niwi proofs. In *Theory of Cryptography Conference*, pages 356–385. Springer, 2019.

[AGP14] Prabhanjan Ananth, Vipul Goyal, and Omkant Pandey. Interactive proofs under continual memory leakage. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34*, pages 164–182. Springer, 2014.

[AN11] Tolga Acar and Lan Nguyen. Homomorphic proofs and applications, 2011.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. Snargs for monotone policy batch np. In *Annual International Cryptology Conference*, pages 252–283. Springer, 2023.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 108–125. Springer, 2009.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349. ACM, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 111–120. ACM, 2013.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 315–333. Springer, 2013.

[BCJP24] Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. Springer-Verlag, 2024.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 505–514, 2014.

[BDI+99] Mike Burmester, Yvo G Desmedt, Toshiya Itoh, Kouichi Sakurai, and Hiroki Shizuya. Divertible and subliminal-free zero-knowledge proofs for languages. *Journal of cryptology*, 12:197–223, 1999.

[BF11] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *International Workshop on Public Key Cryptography*, 2011.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In Stanisław Jarecki and Gene Tsudik, editors, *Public Key Cryptography – PKC 2009*, pages 68–87, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*. 2014.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures. In *Eurocrypt*, 2003.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Kalai. Non-interactive delegation and batch np verification from standard computational assumptions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 474–482, 2017.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J Wu. Lattice-based snargs and their application to more efficient obfuscation. In *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part III*, pages 247–277. Springer, 2017.

[BKP+23] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. Cryptology ePrint Archive, Paper 2023/754, 2023. https://eprint.iacr.org/2023/754.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, 2018.

[BSW12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 350–366, 2012.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT 2013*. 2013.

[BWW23] Eli Bradley, Brent Waters, and David J Wu. Batch arguments to nizks from one-way functions. *Cryptology ePrint Archive*, 2023.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 336–352. Springer, 2013.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Annual Cryptology Conference*, pages 371–389. Springer, 2014.

[CGJ+22] Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and snargs from sub-exponential ddh. *Cryptology ePrint Archive*, 2022.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 738–768. Springer, 2020.

[CJJ21a] Arka Rai Choudhuri, Abhihsek Jain, and Zhengzhong Jin. Snargs for p from lwe. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–79. IEEE, 2021.

[CJJ21b] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 394–423. Springer, 2021.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications*

of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I, pages 769–793, 2020.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 310–331, 2010.

[CW23] Jeffrey Champion and David J. Wu. Non-interactive zero-knowledge from non-interactive batch arguments. Cryptology ePrint Archive, Paper 2023/695, 2023.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *Theory of Cryptography: 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings 9*, pages 54–74. Springer, 2012.

[DGKV22] Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-np and applications. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, 2022.

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 511–520. IEEE, 2010.

[DKS16] David Derler, Stephan Krenn, and Daniel Slamanig. Signer-anonymous designated-verifier redactable signatures for cloud-based data sharing. In *CANS*, 2016.

[DPSS15] David Derler, Henrich C Pöhls, Kai Samelin, and Daniel Slamanig. A general framework for redactable signatures and new constructions. In *ICISC*, 2015.

[DSY91] Alfredo De Santis and Moti Yung. Cryptographic applications of the non-interactive metaproof and many-prover systems. In *Advances in Cryptology-CRYPTO'90: Proceedings 10*, pages 366–377. Springer, 1991.

[DSY12] Alfredo De Santis and Moti Yung. " metaproofs"(and their cryptographic applications). *Cryptology ePrint Archive*, 2012.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs under general assumptions. *SIAM Journal of Computing*, 29(1):1–28, 1999.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In *International conference on the theory and application of cryptology and information security*, pages 499–530. Springer, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 626–645. Springer, 2013.

[GLW21] Rishab Goyal, Jiahui Liu, and Brent Waters. Adaptive security via deletion in attribute-based encryption: Solutions from search assumptions in bilinear groups. In *ASIACRYPT*, 2021.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS '06*, 2006.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Asiacrypt*, volume 6477, pages 321–340. Springer, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

[GSWW22] Rachit Garg, Kristin Sheridan, Brent Waters, and David J Wu. Fully succinct batch arguments for np from indistinguishability obfuscation. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part I*, pages 526–555. Springer, 2022.

[GV22] Rishab Goyal and Vinod Vaikuntanathan. Locally verifiable signature and key aggregation. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference*, 2022.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 469–477, 2015.

[GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. Collusion resistant broadcast and trace from positional witness encryption. In *PKC*, 2019.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 99–108, 2011.

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. Snargs for p from sub-exponential ddh and qr. In *Advances in Cryptology–EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30–June 3, 2022, Proceedings, Part II*, pages 520–549. Springer, 2022.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015*, 2015.

[JMSW02] Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, 2002.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992.

[KLV22] Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. Snargs and ppad hardness from the decisional diffie-hellman assumption. *Cryptology ePrint Archive*, 2022.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and ram delegation. 2023.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 669–684, New York, NY, USA, 2013. ACM.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1115–1124. ACM, 2019.

[KVZ21] Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and snargs. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 330–368. Springer, 2021.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I 19*, pages 41–60. Springer, 2013.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 436–453. IEEE Computer Society, 1994.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pages 96–109. Springer, 2003.

[NWW23] Shafik Nassar, Brent Waters, and David J Wu. Monotone policy bargs from bargs and additively homomorphic encryption. *Cryptology ePrint Archive*, 2023.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *Advances in Cryptology - ASIACRYPT 2015*, 2015.

[PHGR16] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

[PP22] Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1045–1056. IEEE, 2022.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[RRR16] Omer Reingold, Guy N Rothblum, and Ron D Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 49–62, 2016.

[SBZ01] Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In *ICISC*, 2001.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, pages 704–737. Springer, 2020.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, pages 1–18, 2008.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. *IACR Cryptol. ePrint Arch.*, page 336, 2022.

A Preliminaries (Cont'd)

Here we recall more cryptographic concepts that we use in this work. First, we provide the standard soundness definitions and witness indistinguishability definitions for BARGs. Later, we define non-succinct non-interactive proofs.

A.1 BARGs: soundness and witness indistinguishability

Definition A.1 (adaptive soundness)**.** A batch argument scheme BARG satisfies adaptive soundness if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1 & (1^k, 1^n) \leftarrow \mathcal{A}(1^\lambda) \\ \wedge\ \exists i \in [k],\ x_i \notin \mathcal{L} & : \ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n) \\ & (\{x_i\}_{i\in[k]}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition A.2 (semi-adaptive soundness)**.** A batch argument scheme BARG satisfies semi-adaptive somewhere soundness if no PPT attacker has non-negligible advantage in the security game described in Definition A.1, where the attacker commits to the cheating index i^* at the beginning of

the game. That is, $\mathcal{A}$ outputs $i^* \in [k]$ before receiving crs, and it wins the game iff it creates an accepting proof π where the i^*-th instance is not in the language $\mathcal{L}$.

Definition A.3 (witness indistinguishability)**.** A BARG scheme BARG satisfies witness indistinguishability if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathcal{A}(\pi) = b \\ \wedge \left(\begin{array}{l} \forall (i,b) \in [k] \times \{0,1\}, \\ \omega_{i,b} \text{ is a valid witness for } x_i \in \mathcal{L} \end{array}\right) \end{array} : \begin{array}{l} (1^k, 1^n, i^*) \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \\ \{(x_i, \omega_{i,0}, \omega_{i,1})\}_{i \in [k]} \leftarrow \mathcal{A}(\mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_{i,b})\}_i) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

A.2 Non-Succinct Non-Interactive Proofs in the Common Reference String Model

A standard non-interactive proof system consists of the following polynomial time algorithms.

$\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{crs}$. The setup algorithm takes as input the security parameter λ, instance length n, and outputs a crs crs.

$\mathsf{Prove}(\mathsf{crs}, x, \omega) \to \pi$. The prover algorithm takes as input a crs, an instance x, and a witness ω. It outputs a proof π.

$\mathsf{Verify}(\mathsf{crs}, x, \pi) \to 0/1$. The verification algorithm takes as input a crs, an instance x, and a proof π. It outputs a bit to signal whether the proof is valid or not.

Completeness. A proof system is complete if for every $\lambda, n \in \mathbb{N}$, crs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, any instance $x \in \mathcal{L} \cap \{0,1\}^n$ with corresponding witness ω, the following holds

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \ : \ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \omega)\right] = 1.$$

Soundness. A proof system is said to be computationally sound if an attacker can not create accepting proofs of non-instances.

Definition A.4 (soundness)**.** A proof system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is computationally sound if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \\ \wedge\ x \notin \mathcal{L} \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

If the soundness condition holds for computationally unbounded attackers, then we say the scheme is statistically sound.

Witness Indistinguishability. A proof system is said to have (computationally) witness indistinguishability property if an attacker can not distinguish between proofs created using two different valid witnesses.

Definition A.5 (witness indistinguishability)**.** A proof system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is computationally witness indistinguishable if for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathcal{A}(\pi) = b \\ \wedge \left(\begin{array}{l} \omega_0, \omega_1 \text{ are valid} \\ \text{witnesses for } x \in \mathcal{L} \end{array}\right) \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ (x, \omega_0, \omega_1) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow \{0,1\},\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, \omega_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

If the above condition holds for computationally unbounded attackers, then we say the scheme is statistically witness indistinguishable.

(Multi-theorem) Zero-Knowledge. A proof system is said to be (computationally) zero-knowledge if a proof for any accepting instance can be simulated using just the instance. And, it is called multi-theorem zero-knowledge if no PPT distinguisher can distinguish between a polynomial number of simulated and honestly computed proofs.

Definition A.6 (zero-knowledge)**.** A proof system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is computationally zero-knowledge if there exists a stateful PPT simulator $\mathcal{S}$ such that for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \\ \mathcal{A}(\{\pi_{i,b}\}_i) = b \\ \wedge \left(\begin{array}{l} \forall i \in [k],\ \omega_i \text{ is a valid} \\ \text{witness for } x_i \in \mathcal{L} \end{array}\right) \\ \\ \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}(1^\lambda),\ b \leftarrow \{0,1\} \\ \mathsf{crs}_0 \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ \mathsf{crs}_1 \leftarrow \mathcal{S}(1^\lambda, 1^n) \\ \{(x_i, \omega_i)\}_{i\in[k]} \leftarrow \mathcal{A}(\mathsf{crs}_b) \\ \forall i \in [k],\ \pi_{i,0} \leftarrow \mathsf{Prove}(\mathsf{crs}_0, x_i, \omega_i) \\ \{\pi_{i,1}\}_i \leftarrow \mathtt{Sim}(\mathsf{crs}_0, \{x_i\}_i) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

If the above condition holds for computationally unbounded attackers, then we say the scheme is statistically zero-knowledge.

Knowledge Extractor. A proof system is said to have a knowledge extractor if there exists an efficient extractor such that given access to the private random coins of the setup, it can extract a valid witness from any accepting proof.

Definition A.7 (knowledge extractor)**.** A proof system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ has a knowledge extractor if there exists a PPT extractor $\mathcal{E}$ such that for every stateful PPT attacker $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the following holds

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \\ \wedge \left(\begin{array}{l} \omega = \mathcal{E}(\tau, x, \pi) \text{ is not a} \\ \text{valid witness for } x \in \mathcal{L} \end{array}\right) \end{array} : \begin{array}{l} 1^n \leftarrow \mathcal{A}(1^\lambda),\ \tau \leftarrow \{0,1\}^\lambda \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n; \tau) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where τ denotes the randomness used for running the setup algorithm, and we assume (w.l.o.g.) that $|\tau| = \lambda$.

If the above condition holds for computationally unbounded attackers, then we say the scheme is statistically knowledge extractable. We remark that any NIZK scheme can be made to possess an efficient knowledge extractor by using plain public-key encryption.

Remark A.8 (Multi-theorem setting)**.** Observe that we only define the zero-knowledge property in the multi-theorem setting, while other properties are defined only for the single-theorem setting. This is because it is well known that by a standard hybrid argument, the single-theorem versions of other properties are equivalent to their corresponding multi-theorem versions.

B From Soundness to Somewhere Extractability

In this section, we provide our general compiler for upgrading any BARG scheme into a somewhere argument of knowledge. The construction relies on a (plain, non-extractable) BARG $\mathsf{BARG} = (\mathsf{BARG.Setup}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify})$ for language $\hat{\mathcal{L}}$ (described in Fig. 8), and a somewhere extractable hash family $\mathsf{SEH} = (\mathsf{SEH.Setup}, \mathsf{SEH.H}, \mathsf{SEH.Open}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract})$. Below we describe our somewhere extractable BARG scheme $\mathsf{seBARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract})$ for language $\mathcal{L}$.

Language $\hat{\mathcal{L}}$

Instance: $\hat{x} :=$ (instance x, index i, SEH key hk, hash value h_ω).

Witness: $\hat{\omega} :=$ (witness ω w.r.t. $\mathcal{L}$, hash opening $\mathsf{seh}.\pi$).

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \hat{\mathcal{L}}$ if the following are satisfied:

- $\mathsf{seh}.\pi$ **is a valid opening for** x**.** Let $\mathbb{I}_i = \{(i-1)\cdot m + 1, \ldots, i \cdot m\}$ (i.e., coordinates corresponding to i^{th} instance). This requires that $\mathsf{SEH.Verify}(\mathsf{hk}, h_x, \mathbb{I}_i, x, \mathsf{seh}.\pi) = 1$.
- ω **is a valid witness for** x**.** Namely, $\mathcal{R}_{\mathcal{L}}(x, \omega) = 1$ (where $\mathcal{R}_{\mathcal{L}}$ is $\mathcal{L}$'s relation).

Figure 8: Language $\hat{\mathcal{L}}$ for the BARG.

Set notation. Throughout this section, we will use the shorthand notation $\mathbb{I}_i$ to denote the set $\{(i-1)\cdot m + 1, \ldots, i \cdot m\}$. Here m is the length of a single witness corresponding to length-n instances of language $\mathcal{L}$. In words, $\mathbb{I}_i$ denotes the coordinates of the i^{th} block of size m.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$. It runs the setup algorithms for BARG and SEH schemes where the index i^* is used to decide the subset of indices whose extraction is enabled by SEH scheme. Namely, it generates the SEH hash key and trapdoor as

$$(\mathsf{seh.hk}, \mathsf{seh.td}) \leftarrow \mathsf{SEH.Setup}(1^\lambda, k \cdot m, \mathbb{I}_{i^*}),$$

where m is the length of witness corresponding to length n instances of language $\mathcal{L}$ and $\mathbb{I}_{i^*}$ is as defined above. It generates the BARG crs as

$$\mathsf{barg.crs} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^k, 1^{\hat{n}}),$$

where $\hat{n}$ is the length of instances in language $\hat{\mathcal{L}}$ (i.e., $\hat{n}$ is $(n + \lceil \log k \rceil)$ plus the sum of lengths of key $\mathsf{seh.hk}$ and hash value generated using SEH).

It outputs the seBARG crs and trapdoor as

$$\mathsf{crs} = (\mathsf{barg.crs}, \mathsf{seh.hk}), \qquad \mathsf{td} = \mathsf{seh.td}.$$

$\mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i \in [k]}) \to \pi$. It parses the crs as above. It first hashes the sequence of witnesses $(\omega_1, \ldots, \omega_k)$ using the SEH to obtain a digest/hash value h_ω. Namely, it runs as follows

$$h_\omega = \mathsf{SEH.H}(\mathsf{seh.hk}, (\omega_1, \ldots, \omega_k)).$$

It then generates the opening for each witness ω_i as follows

$$\forall i \in [k], \qquad \mathsf{seh}.\pi_i = \mathsf{SEH.Open}(\mathsf{seh.hk}, (\omega_1, \ldots, \omega_k), \mathbb{I}_i).$$

Next, it computes a BARG using the (non-extractable) BARG system where each instance now additionally contains its index i, the hash key $\mathsf{seh.hk}$ and hash value h_ω, and the corresponding witness additionally contains the hash opening $\mathsf{seh}.\pi_i$. Formally, it generates the batch proof as

$$\mathsf{barg}.\pi \leftarrow \mathsf{BARG.Prove}(\mathsf{barg.crs}, \{(\hat{x}_i, \hat{\omega}_i)\}_{i \in [k]}),$$

where $\hat{x}_i = (x_i, i, \mathsf{seh.hk}, h_\omega)$ and $\hat{\omega}_i = (\omega_i, \mathsf{seh}.\pi_i)$ for every $i \in [k]$.

Finally, it outputs $\pi = (h_\omega, \mathsf{barg}.\pi)$.

$\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i \in [k]}, \pi) \to 0/1$. It parses the crs and proof as above. Note that crs contains the hash key $\mathsf{seh.hk}$ and proof contains the hashed value h_ω along with a BARG $\mathsf{barg}.\pi$. The verifier simply runs the (non-extractable) BARG verifier on $\mathsf{barg}.\pi$ where the instances now also include $\mathsf{seh.hk}$ and h_ω. Concretely, it outputs the following

$$\mathsf{BARG.Verify}(\mathsf{barg.crs}, \{\hat{x}_i\}_{i \in [k]}, \mathsf{barg}.\pi)$$

where $\hat{x}_i = (x_i, i, \mathsf{seh.hk}, h_\omega)$ for every $i \in [k]$.

$\mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi) \to \omega$. It parses the trapdoor and proof as above, and runs the SEH extraction algorithm to extract the witness ω as

$$\omega = \mathsf{SEH.Extract}(\mathsf{seh.td}, h_\omega, \mathbb{I}_{i^*}).$$

Correctness and Compactness. We now show that the above somewhere extractable BARG scheme satisfies completeness, correctness of extraction, and compactness if the underlying BARG and SEH schemes satisfy appropriate correctness and compactness properties.

First, note that by the SEH correctness of opening we know that $\mathsf{SEH.Verify}(\mathsf{seh.hk}, h_\omega, \mathbb{I}_i, \omega_i, \mathsf{seh}.\pi_i) = 1$ for the opening $\mathsf{seh}.\pi_i$ as computed by the prover. Therefore, whenever ω_i is a valid witness for x_i (w.r.t. language $\mathcal{L}$), then we have that $\hat{\omega}_i$ is a valid witness for $\hat{x}_i$ (w.r.t. language $\hat{\mathcal{L}}$) for every i. Thus, by completeness of BARG, the completeness of seBARG follows.

Second, note that by SEH correctness of extraction we know that $\mathsf{SEH.Extract}(\mathsf{seh.td}, h_\omega, \mathbb{I}_{i^*}) = \omega_{i^*}$ for the digest h_ω as computed by the prover. Thus, the correctness of seBARG extraction follows.

Finally, to argue compactness of the seBARG scheme, note that it is sufficient to show that $|\hat{x}|$ and $|\hat{\omega}|$ are both $\mathsf{poly}(\lambda, \log k, n, m)$ as that implies compactness by BARG compactness. This follows from the fact due to the compactness of SEH we get that $|\mathsf{seh.hk}|$, $|\mathsf{seh}.\pi_i|$, $|h_\omega|$ are all individually bounded as polynomials in λ, n and m. Thus, compactness of seBARG follows.

Security. Below we prove the security of our seBARG scheme.

Theorem B.1. If the (non-extractable) BARG scheme BARG satisfies semi-adaptive soundness (Definition A.2), and the SEH scheme SEH satisfies index hiding and somewhere statistical binding (Definitions 3.3 and 3.4), then the above scheme seBARG is a somewhere extractable BARG scheme satisfying index hiding, semi-adaptive soundness, and somewhere argument of knowledge (Definitions 3.1, 3.2 and A.2).

Proof. The proof is divided into three parts where we individually prove the desired properties.

Index hiding. This follows directly from the index hiding property of the SEH scheme. Concretely, we prove the following.

Lemma B.2. If the SEH scheme SEH satisfies index hiding, then seBARG also satisfies index hiding.

Proof. This follows from a simple reduction to the SEH index hiding challenger. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks index hiding of the seBARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We can design a reduction algorithm $\mathcal{B}$ that break index hiding of the SEH scheme with probability ϵ.

Briefly, the attacker $\mathcal{A}$ starts by outputting $(1^k, 1^n, i_0^*, i_1^*)$. The reduction algorithm $\mathcal{B}$ simply sends $k \cdot m$ as the input length and sets $\mathbb{I}_{i_0^*}, \mathbb{I}_{i_1^*}$ to the SEH challenger. (Note that the sets $\mathbb{I}_{i_0^*}, \mathbb{I}_{i_1^*}$ are as defined in the construction.) The challenger sends a hash key $\mathsf{seh.hk}$ to $\mathcal{B}$, and $\mathcal{B}$ then runs the rest of the setup algorithm to sample a crs $\mathsf{barg.crs} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^k, 1^{\hat{n}})$. $\mathcal{B}$ then sends $(\mathsf{barg.crs}, \mathsf{seh.hk})$ as the crs to $\mathcal{A}$. And, finally $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Note that $\mathcal{B}$ wins the SEH index hiding game whenever $\mathcal{A}$ wins the seBARG index hiding since $\mathcal{B}$ simulates the environment perfectly for $\mathcal{A}$. Thus, the lemma follows. □

Somewhere argument of knowledge. This follows directly from the somewhere statistical binding property of the SEH scheme and semi-adaptive soundness of BARG. Concretely, we prove the following.

Lemma B.3. If the SEH scheme SEH satisfies somewhere statistical binding and the BARG scheme BARG satisfies semi-adaptive soundness, then seBARG satisfies somewhere argument of knowledge.

Proof. This follows from a simple case by case reduction to the SEH somewhere statistical binding challenger and BARG semi-adaptive soundness challenger. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks somewhere argument of knowledge of the seBARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases.

Type 1 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ but $\hat{x}_{i^*} \notin \hat{\mathcal{L}}$. Note that a Type 1 attacker breaks the semi-adaptive soundness property of the underlying (non-extractable) BARG scheme. This follows directly from a reduction to semi-adaptive soundness of BARG.

Briefly, the attacker $\mathcal{A}$ starts by outputting $(1^k, 1^n, i^*)$. The reduction algorithm $\mathcal{B}$ simply sends $(1^k, 1^{\hat{n}}, i^*)$ to the BARG challenger, and the challenger sends the crs $\mathsf{barg.crs}$ to $\mathcal{B}$. $\mathcal{B}$ then samples the SEH hash key $\mathsf{seh.hk}$ along with trapdoor $\mathsf{seh.td}$ as in the setup, and sends $(\mathsf{barg.crs}, \mathsf{seh.hk})$ as the crs to $\mathcal{A}$. $\mathcal{A}$ then outputs a sequence of k instances $x_1, \ldots, x_k$ along with a proof $\pi = (h_\omega, \mathsf{barg}.\pi)$. $\mathcal{B}$ then submits $\{\hat{x}_i\}_i$ and $\mathsf{barg}.\pi$ as its soundness attack to the BARG challenger (here $\hat{x}_i$ is as

defined in the description of the scheme). Note that whenever $\mathcal{A}$ is a successful *type 1* attacker on somewhere argument of knowledge, then $\mathcal{B}$ is a successful semi-adaptive soundness attacker on BARG. This follows from the definition of a Type 1 attacker, and $\mathcal{A}$'s advantage in breaking somewhere argument of knowledge.

Type 2 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof $\pi = (h_\omega, \mathsf{barg}.\pi)$ such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ *and* $\hat{x}_{i^*} \in \hat{\mathcal{L}}$, but $\mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi)$ is not a valid witness for x_{i^*}. Note that a Type 2 attacker breaks the somewhere statistical binding property of the SEH scheme. This relies on the fact that given $\hat{x}_{i^*} \in \hat{\mathcal{L}}$, it holds that there is a witness $\hat{\omega}_{i^*}$ of the form $(\omega^*, \mathsf{seh}.\pi^*)$ for instance $\hat{x}_{i^*} = (x_{i^*}, i^*, \mathsf{seh}.\mathsf{hk}, h_\omega)$. Since $\hat{\omega}_{i^*} = (\omega^*, \mathsf{seh}.\pi^*)$ is a valid witness for $\hat{x}_{i^*}$, thus we get that ω^* is a valid witness for x_{i^*} (as per language $\mathcal{L}$) *and* $\mathsf{seh}.\pi^*$ is a valid opening for ω^* for set $\mathbb{I}_{i^*}$ and hash value h_ω w.r.t. key $\mathsf{seh}.\mathsf{hk}$. That is, $\mathsf{SEH.Verify}(\mathsf{seh}.\mathsf{hk}, h_\omega, \mathbb{I}_{i^*}, \omega^*, \mathsf{seh}.\pi^*) = 1$. Now by the somewhere statistical binding property of SEH, we know that it must be the case that $\mathsf{SEH.Extract}(\mathsf{seh}.\mathsf{td}, h_\omega) = \omega^*$. This results in a contradiction. Thus, $\mathcal{A}$ can not be a successful Type 2 attacker even if is computationally unbounded.

Note that any successful somewhere argument of knowledge attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Semi-adaptive soundness. As noted in prior works [DGKV22, Remark 3.3], somewhere argument of knowledge property implies semi-adaptive soundness. Combining it with our somewhere argument of knowledge lemma proved above, we get the following.

Lemma B.4. If the SEH scheme SEH satisfies somewhere statistical binding and the BARG scheme BARG satisfies semi-adaptive soundness, then seBARG satisfies semi-adaptive soundness.

Note that the above proof of above lemma follows directly by combining proof of Lemma B.3 with [DGKV22, Remark 3.3]. We could alternatively prove it directly by following the same strategy as for the proof of Lemma B.3.
This completes the proof of our main theorem. □

C Semi-Adaptive Soundness to Adaptive Soundness

In this section, we show how to upgrade any polynomially-hard non-interactive batch argument scheme with semi-adaptive soundness to (full) adaptive soundness by relying on exponential hardness of a somewhere extractable hash function.

Our compiler is basically identical to our compiler from Appendix B, except the security parameters used for the underlying batch argument scheme and the hash function are set differently. In a few words, the security parameter λ_h for SEH is set such that SEH is still secure even from attackers running in time $\mathsf{poly}(2^m, \lambda)$ (where m is the length of witnesses). Below we describe it briefly for completeness.

Construction. Let $\mathsf{BARG} = (\mathsf{BARG.Setup}, \mathsf{BARG.Prove}, \mathsf{BARG.Verify})$ be a (non-extractable) BARG scheme, and $\mathsf{SEH} = (\mathsf{SEH.Setup}, \mathsf{SEH.H}, \mathsf{SEH.Open}, \mathsf{SEH.Verify}, \mathsf{SEH.Extract})$ be a somewhere extractable hash scheme. Below we describe our adaptively sound BARG scheme $\mathsf{adaptive\text{-}BARG}$. As in Appendix B, we will use the same language $\hat{\mathcal{L}}$ as described in Fig. 8, with the difference that the security parameter for SEH is differently selected.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$. It runs the setup algorithms for BARG and SEH schemes where the index i^* is used to decide the subset of indices whose extraction is enabled by SEH scheme. Namely, it generates the SEH hash key and trapdoor as

$$(\mathsf{seh.hk}, \mathsf{seh.td}) \leftarrow \mathsf{SEH.Setup}(1^{\lambda_h}, k \cdot m, \mathbb{I}_{i^*}),$$

where $\lambda_h = \lambda \cdot m$, and m is the length of the witness (corresponding to length n instances of language $\mathcal{L}$). It generates the BARG crs as

$$\mathsf{barg.crs} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^k, 1^{\hat{n}}),$$

where $\hat{n}$ is the length of instances in language $\hat{\mathcal{L}}$ (i.e., $\hat{n}$ is $(n + \lceil \log k \rceil)$ plus the sum of lengths of key $\mathsf{seh.hk}$ and hash value generated using SEH).

It outputs the crs and trapdoor as

$$\mathsf{crs} = (\mathsf{barg.crs}, \mathsf{seh.hk}), \qquad \mathsf{td} = \mathsf{seh.td}.$$

$\mathsf{Prove}, \mathsf{Verify}, \mathsf{Extract}$. The prover, verifier, and extraction algorithms are identical as in Appendix B.

Correctness and Compactness. The correctness and compactness argument is same as before except, due to the change in selecting the security parameter λ_h, there is an additional polynomial dependence on the witness length m.

Security. Below we prove the adaptive security of our adaptive-BARG scheme. Before providing the main theorem, we state the stronger (sub-exponential) version of the index hiding security for SEH hash functions that we need for proving security.

Definition C.1 (Sub-Exponential Index Hiding)**.** A somewhere extractable hash scheme SEH satisfies *sub-exponential* index hiding if for every admissible attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all (large enough) $\lambda_1, \lambda_2 \in \mathbb{N}$ where $\lambda_1 > \lambda_2$, the following holds

$$\Pr\left[\begin{array}{ll} \mathcal{A}_2(\mathsf{st}, \mathsf{hk}) = b & (\mathsf{st}, N, I_0, I_1) \leftarrow \mathcal{A}_1(1^{\lambda_1}, 1^{\lambda_2}),\ b \leftarrow \{0,1\} \\ \wedge\ |I_0| = |I_1| & : \\ & (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^{\lambda_1 + \lambda_2}, N, I_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda_1 + \lambda_2),$$

where attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is admissible as long as the running time of $\mathcal{A}_1$ is $\mathsf{poly}(\lambda_1, \lambda_2)$ and $\mathcal{A}_2$ is $\mathsf{poly}(\lambda_1, 2^{\lambda_2})$.

In words, there are two security parameters λ_1, λ_2, and the attacker is allowed to run in time exponential in λ_2. This is a slightly weaker version of sub-exponential security definition for index hiding, and is sufficient for our construction.

Theorem C.2. If the (non-extractable) BARG scheme BARG satisfies semi-adaptive soundness (Definition A.2), and the SEH scheme SEH satisfies *sub-exponential* index hiding and somewhere statistical binding (Definitions 3.4 and C.1), then the above scheme adaptive-BARG is a somewhere extractable BARG scheme satisfying index hiding, *adaptive* soundness, and somewhere argument of knowledge (Definitions 3.1, 3.2 and A.1).

Proof. The proof of index hiding and somewhere argument of knowledge follows directly from Lemmas B.2 and B.3. Also, note that sub-exponential index hiding security is not needed for proving these properties. The proof of adaptive soundness is also similar, except now the reduction algorithm for breaking index hiding security of SEH has to be designed more carefully.

Adaptive soundness. This follows from the sub-exponential index hiding security and somewhere statistical binding security of SEH, and semi-adaptive soundness of BARG. Concretely, we prove the following.

Lemma C.3. If the SEH scheme SEH satisfies sub-exponential index hiding and somewhere statistical binding, and the BARG scheme BARG satisfies semi-adaptive soundness, then $\mathsf{adaptive\text{-}BARG}$ satisfies adaptive soundness.

Proof. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks adaptive soundness of the $\mathsf{adaptive\text{-}BARG}$ scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We claim show that there exists a PPT attacker $\hat{\mathcal{A}}_{i^*}$ that break semi-adaptive soundness of $\mathsf{adaptive\text{-}BARG}$ with non-negligible probability $\delta = \delta(\lambda)$ for some $i^* \in [k]$.

Formally, for $i^* \in [k]$, let $\hat{\mathcal{A}}_{i^*}$ be the attacker that sends i^* to the semi-adaptive soundness challenger. After receiving the crs crs, it runs the attacker $\mathcal{A}$ with crs and outputs whatever $\mathcal{A}$ outputs. We claim the following.

Claim C.4. Fix a non-negligible function ϵ. Assuming the SEH scheme SEH satisfies (sub-exponential) index hiding security, if there exists a PPT attacker $\mathcal{A}$ whose advantage in the $\mathsf{adaptive\text{-}BARG}$ adaptive soundness game is $\epsilon(\lambda)$, then there exists an index $i^* \in [k]$ and non-negligible function δ such that the advantage of the PPT attacker $\hat{\mathcal{A}}_{i^*}$ (described above) in the $\mathsf{adaptive\text{-}BARG}$ semi-adaptive soundness game is at least $\delta(\lambda)$.

Proof. This follows from a simple reduction to the SEH index hiding challenger. Suppose there exists a PPT attacker $\mathcal{A}$ whose advantage in the adaptive soundness game is ϵ, but for all $i^* \in [k]$, $\hat{\mathcal{A}}_{i^*}$'s advantage in the semi-adaptive soundness game is negligible.

Briefly, the adaptive soundness attacker $\mathcal{A}$ starts by outputting $(1^k, 1^n)$. The reduction algorithm $\mathcal{B}$ chooses two random indices $i_0^* \neq i_1^* \leftarrow [k]$, and submits $k \cdot m$ as the input length and sets $\mathbb{I}_{i_0^*}, \mathbb{I}_{i_1^*}$ to the SEH challenger. The challenger sends back hash key $\mathsf{seh.hk}$, and $\mathcal{B}$ then runs the rest of the setup algorithm to sample a crs $\mathsf{barg.crs} \leftarrow \mathsf{BARG.Setup}(1^\lambda, 1^k, 1^{\hat{n}})$. $\mathcal{B}$ then sends $(\mathsf{barg.crs}, \mathsf{seh.hk})$ as the crs to $\mathcal{A}$. And, $\mathcal{A}$ then outputs a sequence of instances $\{x_i\}_i$ along with a proof π. $\mathcal{B}$ then checks whether π is an accepting proof or not. That is, it checks $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$. If the check fails, then it aborts and guesses randomly. Otherwise, it continues as follows.

For every $i \in [k]$, $\mathcal{B}$ performs brute force search over all possible witness strings $\omega_i \in \{0,1\}^m$ to check whether $x_i \in \mathcal{L}$ or $x_i \notin \mathcal{L}$. If there does not exist an unsatisfying instance x_i, then it aborts and guesses randomly. Otherwise, if $x_{i_0^*} \notin \mathcal{L}, x_{i_1^*} \in \mathcal{L}$, then it outputs 1 to guess that SEH challenger used $\mathbb{I}_{i_1^*}$. Similarly, if $x_{i_0^*} \in \mathcal{L}, x_{i_1^*} \notin \mathcal{L}$, then it outputs 0 to guess that SEH challenger used $\mathbb{I}_{i_0^*}$. And, if $x_{i_0^*}, x_{i_1^*}$ are both either YES or NO instances, then it aborts and guesses randomly.

Note that the running time of $\mathcal{B}$ is $\mathsf{poly}(2^m, \lambda)$ due to the brute force search. Now note that since $\hat{\mathcal{A}}_{i^*}$'s advantage is negligible for all i^*, thus if $\mathcal{A}$ creates an accepting (unsound) proof in the above reduction such that $x_{i_0^*} \notin \mathcal{L}$ but $x_{i_1^*} \in \mathcal{L}$, then it must be the case that the SEH hask key $\mathsf{seh.hk}$ is created for set $\mathbb{I}_{i_1^*}$. And, similarly we can argue for the case $x_{i_0^*} \in \mathcal{L}$ but $x_{i_1^*} \notin \mathcal{L}$. Now since i_0^* and i_1^* are sampled completely independently and uniformly at random, and $\mathcal{A}$'s behavior depends upon at most one of the indices (i.e., it is independent of the other index since only one index is encoded in the hask key), thus whenever $\mathcal{A}$'s creates an accepting unsound proof, then with at least $1/(k-1)$ probability, $\mathcal{B}$ does not abort. Therefore, if $\mathcal{A}$'s advantage in the adaptive soundness game is ϵ, and $\hat{\mathcal{A}}_{i^*}$'s advantage in the semi-adaptive soundness game is negligible (for

every i^*), then $\mathcal{B}$'s advantage in the index hiding game is at least $\epsilon/(k-1)$. Thus, the lemma follows. □

Claim C.5. If the SEH scheme SEH satisfies somewhere statistical binding and the BARG scheme BARG satisfies semi-adaptive soundness, then adaptive-BARG satisfies semi-adaptive soundness.

Proof. This follows from the proof of Lemma B.4. □

□

This completes the proof of our main theorem. □

D Making BARGs witness indistinguishable

In this section, we describe the general approach to make a BARG proof system zero-knowledge. We construct a witness hiding BARG scheme wh-BARG = (Setup, Prove, Verify) for language $\mathcal{L}$. Our construction relies on a (non-interactive, non-succinct) WI proof system Π = (Π.Setup, Π.Prove, Π.Verify) for language $\mathcal{L}$, and a somewhere extractable BARG scheme seBARG = (Setup, Prove, Verify, Extract) for language $\hat{\mathcal{L}}$ (described in Fig. 9).

Language $\hat{\mathcal{L}}$

Instance: $\hat{x} :=$ (instance x, NIWI crs).

Witness: $\hat{\omega} :=$ NIWI proof π.

Membership: $\hat{\omega}$ is a valid witness for $\hat{x} \in \hat{\mathcal{L}}$ if π is an accepting WI proof for instance x w.r.t. crs. Namely,

$$\Pi.\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \text{ implies that } \hat{x} \in \hat{\mathcal{L}} \text{ and } \hat{\omega} \text{ is a valid witness.}$$

Figure 9: Language $\hat{\mathcal{L}}$ for the seBARG.

$\mathsf{Setup}(1^\lambda, 1^k, 1^n, i^*) \to (\mathsf{crs}, \mathsf{td})$. It runs the setup algorithms for seBARG and Π schemes where the index i^* is used to fix the extraction index in seBARG scheme. Namely, it generates $\Pi.\mathsf{crs} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, 1^k, 1^n)$. Next, it generates the seBARG parameters as

$$(\mathsf{sebg.crs}, \mathsf{sebg.td}) \leftarrow \mathsf{seB.Setup}(1^\lambda, 1^k, 1^{\hat{n}}, i^*),$$

where $\hat{n}$ is the length of instances in language $\hat{\mathcal{L}}$ (i.e., $\hat{n}$ is n plus the length of crs $\Pi.\mathsf{crs}$). It outputs the wh-BARG crs and trapdoor as $\mathsf{crs} = (\mathsf{sebg.crs}, \Pi.\mathsf{crs})$, $\mathsf{td} = \mathsf{sebg.td}$.

$\mathsf{Prove}(\mathsf{crs}, \{(x_i, \omega_i)\}_{i\in[k]}) \to \pi$. It parses the crs as above. It first creates a WI proof for each instance individually as

$$\forall i \in [k],\ \Pi.\pi_i \leftarrow \Pi.\mathsf{Prove}(\Pi.\mathsf{crs}, x_i, \omega_i).$$

Next, it computes a BARG using the seBARG system where each instance now additionally contains $\Pi.\mathsf{crs}$, and the corresponding witness contains only the WI proof $\Pi.\pi_i$. Formally, it generates the batch proof as

$$\mathsf{sebg}.\pi \leftarrow \mathsf{seB.Prove}(\mathsf{sebg.crs}, \{(\hat{x}_i, \hat{\omega}_i)\}_{i\in[k]}),$$

where $\hat{x}_i = (x_i, \Pi.\mathsf{crs})$ and $\hat{\omega}_i = \Pi.\pi_i$ for every $i \in [k]$. Finally, it outputs $\pi = \mathsf{sebg}.\pi$.

$\mathsf{Verify}(\mathsf{crs}, \{x_i\}_{i\in[k]}, \pi) \to 0/1$. It parses the crs and proof as above. It simply runs the seBARG verifier on $\mathsf{sebg}.\pi = \pi$ where the instances now also contain $\Pi.\mathsf{crs}$. Concretely, it outputs the following

$$\mathsf{seB.Verify}(\mathsf{sebg.crs}, \{\hat{x}_i\}_{i\in[k]}, \mathsf{sebg}.\pi)$$

where $\hat{x}_i = (x_i, \Pi.\mathsf{crs})$ for every $i \in [k]$.

Preserving local verifiability. We want to point out that if the underlying somewhere extractable BARG scheme is locally verifiable, then the upgraded witness hiding BARG scheme is also locally verifiable. That is, if seBARG.LOpen and seBARG.LVfy algorithms exist satisfying the corresponding properties, then using the following algorithms we can make our above construction to be locally verifiable as well.

$\mathsf{LOpen}(\mathsf{crs}, \{x_i\}_i, j, \pi) \to (\mathsf{aux}_j, \pi')$. It parses the crs and proof as above. It computes the auxiliary information and updated proof using seBARG's local opening algorithm directly. Namely, it computes

$$(\mathsf{aux}_j, \pi') = \mathsf{seBARG.LOpen}(\mathsf{sebg.crs}, \{\hat{x}_i\}_i, j, \pi),$$

where $\hat{x}_i = (x_i, \Pi.\mathsf{crs})$ for every $i \in [k]$.

$\mathsf{LVfy}(\mathsf{crs}, x, j, \pi, \mathsf{aux}) \to 0/1$. The local verifier parses the inputs as above, and simply runs the seBARG local verifier on π and aux where the instance now also contain $\Pi.\mathsf{crs}$. Concretely, it outputs the following

$$\mathsf{seBARG.LVfy}(\mathsf{sebg.crs}, \hat{x} = (x, \Pi.\mathsf{crs}), j, \pi, \mathsf{aux}).$$

Remark D.1 (Proof-Independent Openings)**.** Note that if the underlying BARG scheme seBARG has proof-independent openings, then our transformation preserves it.

Preserving somewhere extractability. We want to point out that if the underlying non-succinct Π scheme has a knowledge extractor, then the upgraded witness hiding BARG scheme maintains its somewhere extractability property. That is, if a knowledge extractor $\Pi.\mathcal{E}$ exists, then it can be used to extract the witness at location i^* as follows.

$\mathsf{Extract}(\mathsf{td}, \{x_i\}_i, \pi) \to \omega$. Here we assume that the trapdoor td also contains the random coins τ used during $\Pi.\mathsf{Setup}$. It first runs the seBARG extractor to obtain the witness $\hat{\omega}$ as

$$\hat{\omega} = \mathsf{seB.Extract}(\mathsf{td}, \{\hat{x}_i\}_i, \pi),$$

where $\hat{x}_i = (x_i, \Pi.\mathsf{crs})$ for every $i \in [k]$. Next, it runs the Π extractor to extract the actual witness ω as

$$\omega = \Pi.\mathcal{E}(\tau, x_{i^*}, \hat{\omega}).$$

Correctness and Compactness. Note that completeness and local correctness of our designed BARGs follow directly from the completeness of Π proof system combined with completeness and local correctness of seBARG (respectively). Additionally, extraction correctness follows directly from the extraction correctness of seBARG and Π. Also, compactness of the wh-BARG scheme follows directly from the compactness of seBARG scheme and the fact that the running time of Π.Setup and Π.Prove is a fixed polynomial.

Security. Below we prove the security of our lv-BARG scheme.

Theorem D.2 (Index hiding, soundness, witness indistinguishability)**.** If the somewhere extractable BARG scheme seBARG satisfies index hiding, semi-adaptive soundness, and somewhere argument of knowledge (Definitions 3.1, 3.2 and A.2), and the proof system Π satisfies soundness and witness indistinguishability (Definitions A.4 and A.5), then the above scheme wh-BARG is a BARG scheme satisfying index hiding, semi-adaptive soundness, and witness indistinguishability (Definitions 3.1, A.2 and A.3).

Proof. The proof is divided into multiple parts where we individually prove the desired properties.

Index hiding. This follows directly from the index hiding properties of the seBARG scheme. Concretely, we can prove the following.

Lemma D.3. If the seBARG scheme seBARG satisfies index hiding, then wh-BARG also satisfies index hiding.

This follows from a simple reduction to seBARG index hiding.

Semi-adaptive soundness. This follows directly from the soundness property of the Π system and somewhere argument of knowledge of seBARG. Concretely, we prove the following.

Lemma D.4. If the proof system Π satisfies soundness and the seBARG scheme seBARG satisfies somewhere argument of knowledge, then wh-BARG satisfies semi-adaptive soundness.

Proof. This follows from a simple case by case reduction to the proof system soundness challenger and the BARG somewhere argument of knowledge challenger. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks soundness of wh-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases.

Type 1 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ but $\mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}$ is **not** a valid witness for $\hat{x}_{i^*}$. Note that a Type 1 attacker breaks the somewhere argument of knowledge property of the underlying seBARG scheme. This follows directly from a reduction to somewhere argument of knowledge of seBARG.

Type 2 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ *and* $\mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}$ is a valid witness for $\hat{x}_{i^*}$, but $x_{i^*} \notin \mathcal{L}$. Note that a Type 2 attacker breaks the soundness property of the Π proof system. This relies on the fact that given $\hat{\omega}$ is a valid witness for $\hat{x}_{i^*}$, then $\hat{\omega}$ is an accepting Π proof for instance x_{i^*} (w.r.t. language $\mathcal{L}$). This results in a contradiction that Π satisfies soundness. Thus, $\mathcal{A}$ can not be a successful Type 2 attacker.

Note that any successful soundness attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Witness Indistinguishability. This follows directly from the witness indistinguishability property of the Π system. Concretely, we prove the following.

Lemma D.5. If the proof system Π satisfies witness indistinguishability, then wh-BARG satisfies witness indistinguishability.

Proof. This follows from a sequence of $k-1$ intermediate hybrid experiments, where in the i^{th} intermediate hybrid, the challenger uses $\omega_{1,1}, \ldots, \omega_{i,1}, \omega_{i+1,0}, \ldots, \omega_{k,0}$ as the witnesses for creating the batch proof. Now observe that the only difference between any two consecutive hybrid experiments (say $i-1$ and i) is that the i^{th} witness is switched from $\omega_{i,0}$ to $\omega_{i,1}$. The reduction algorithm can simply forward this to the witness indistinguishability challenger, and use attacker's guess to win the witness indistinguishability game w.r.t. Π. Alternatively, one could directly use the multi-theorem version of the witness indistinguishability property to prove it directly. □

This completes the proof of our theorem. □

Theorem D.6 (Zero-Knowledge)**.** If the proof system Π is zero-knowledge (Definition A.6), then the above scheme wh-BARG satisfies BARG zero-knowledge property (Definition 7.4).

Proof. This follows from a straightforward reduction to the (multi-theorem) zero-knowledge property of the Π proof system. Note that the reduction algorithm simply samples everything except the Π crs $\Pi.\mathsf{crs}$ and proofs $\Pi.\pi_i$ (to be used as witnesses for $\hat{x}_i$), and it receives these from the proof system challenger (where either they are simulated or computed honestly). If the attacker can break zero-knowledge property for wh-BARG, then the reduction algorithm will break zero-knowledge property of Π as well. Thus, the theorem follows. □

Theorem D.7 (Somewhere extractability)**.** If the somewhere extractable BARG scheme seBARG satisfies somewhere argument of knowledge (Definition 3.2), and the proof system Π *has a knowledge extractor* (Definition A.7), then the above scheme wh-BARG is a BARG scheme satisfying somewhere argument of knowledge as well.

Proof. This follows from a simple case by case reduction to the proof system knowledge extraction challenger and the BARG somewhere argument of knowledge challenger. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks somewhere argument of knowledge of the wh-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases.

Type 1 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ but $\mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}$ is **not** a valid witness for $\hat{x}_{i^*}$. Note that a Type 1 attacker breaks the somewhere argument of knowledge property of the underlying seBARG scheme. This follows directly from a reduction to somewhere argument of knowledge of seBARG.

Type 2 attacker: $\mathcal{A}$ outputs a sequence of instances $\{x_i\}_i$ and a proof π such that $\mathsf{Verify}(\mathsf{crs}, \{x_i\}_i, \pi) = 1$ *and* $\mathsf{seB.Extract}(\mathsf{sebg.td}, \{\hat{x}_i\}_i, \pi) = \hat{\omega}$ is a valid witness for $\hat{x}_{i^*}$, but $\Pi.\mathcal{E}(\tau, x_{i^*}, \hat{\omega}) = \omega$ is **not** a valid witness for $x_{i^*} \in \mathcal{L}$. Note that a Type 2 attacker breaks the knowledge extraction property of the Π proof system. This relies on the fact that given $\hat{\omega}$ is a valid witness for $\hat{x}_{i^*}$, then $\hat{\omega}$ is an accepting Π proof for instance x_{i^*} (w.r.t. language $\mathcal{L}$). This results in a contradiction that Π has a knowledge extractor. Thus, $\mathcal{A}$ can not be a successful Type 2 attacker.

Note that any successful somewhere argument of knowledge attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Theorem D.8 (Local soundness)**.** If the somewhere extractable BARG scheme seBARG satisfies local argument of knowledge with adversarial openings (Definition 5.3), and the proof system Π satisfies soundness (Definition A.4), then the above scheme wh-BARG is a BARG scheme satisfying semi-adaptive local soundness with adversarial openings.

Proof. This follows from a simple case by case reduction to the proof system soundness challenger and the BARG local argument of knowledge challenger. Suppose there exists a PPT attacker $\mathcal{A}$ that breaks local soundness of the wh-BARG scheme with non-negligible probability $\epsilon = \epsilon(\lambda)$. We divide the analysis in two cases.

Type 1 attacker: $\mathcal{A}$ outputs an instance x, proof π, and auxiliary information aux such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ but $\mathsf{sebg.LExtract}(\mathsf{sebg.td}, x, \pi, \mathsf{aux}) = \hat{\omega}$ is **not** a valid witness for $\hat{x} = (x, \Pi.\mathsf{crs})$. (Here sebg.LExtract is the local argument of knowledge extractor.) Note that a Type 1 attacker breaks the local argument of knowledge property of the underlying seBARG scheme. This follows directly from a reduction to somewhere argument of knowledge of seBARG.

Type 2 attacker: $\mathcal{A}$ outputs an instance x, proof π, and auxiliary information aux such that $\mathsf{LVfy}(\mathsf{crs}, x, i^*, \pi, \mathsf{aux}) = 1$ *and* $\mathsf{sebg.LExtract}(\mathsf{sebg.td}, x, \pi, \mathsf{aux}) = \hat{\omega}$ is a valid witness for $\hat{x} = (x, \Pi.\mathsf{crs})$, but $x \notin \mathcal{L}$. Note that a Type 2 attacker breaks the soundness property of the Π proof system. This relies on the fact that given $\hat{\omega}$ is a valid witness for $\hat{x}$, then $\hat{\omega}$ is an accepting Π proof for instance x (w.r.t. language $\mathcal{L}$). This results in a contradiction that Π satisfies soundness. Thus, $\mathcal{A}$ can not be a successful Type 2 attacker.

Note that any successful local soundness attacker $\mathcal{A}$ must be either Type 1 or 2. Thus, combining the above arguments, the lemma follows. □

Theorem D.9 (Local extractability)**.** If the somewhere extractable BARG scheme seBARG satisfies local argument of knowledge with adversarial openings (Definition 5.3), and the proof system Π *has a knowledge extractor* (Definition A.7), then the above scheme wh-BARG is a BARG scheme satisfying local argument of knowledge with adversarial openings.

Proof. This is similar to the proof of Theorem D.8, with the only change that for showing that there does not exist an appropriate Type 2 attacker we rely on the knowledge extractor for the proof system Π. □

Contents