Mutable Batch Arguments and Applications

Rishab Goyal
University of Wisconsin-Madison, WI, USA

Abstract

We put forth a new concept of mutability for batch arguments (BARGs), called *mutable batch arguments*. Our goal is to re-envision how we view and use BARGs. Traditionally, a BARG proof π is an *immutable* encoding of k **NP** witness $\omega_1, \ldots, \omega_k$. A mutable BARG system captures the notion of computations over BARGs, where each proof string π is treated as a *mutable* encoding of original witnesses. We also study strong privacy notions for mutable BARGs, with the goal of hiding all non-trivial information about witnesses from a mutated proof. Such mutable BARGs are a naturally good fit for many privacy sensitive applications. Our main contributions include introducing the concept of mutable BARGs, identifying non-trivial classes of feasible mutations, designing mutable BARGs with varying capabilities satisfying mutation privacy from standard cryptographic assumptions, and enabling new applications while improving state-of-the-art known for many signature systems.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases BARGs, Mutable proofs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.101

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://eprint.iacr.org/2024/737

Funding *Rishab Goyal*: Support for this research was provided by OVCRGE at UW-Madison with funding from the Wisconsin Alumni Research Foundation.

1 Introduction

Batch arguments (BARGs) enable computations of *succinct* proofs for certifying validity of k **NP** statements $\{x_i \in \mathcal{L}\}_{i \leq k}$. Succinctness requires the proof size to be $\mathsf{poly}(\lambda, \log k)$, while soundness states that no polynomial-time cheating prover can create an accepting proof π for an unsatisfying batch of statements.

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 many longstanding open problems (see the non-exhaustive list [73, 24, 60, 38, 37, 61, 78, 56, 44, 70, 58, 47, 36, 59]). Today, we have numerous BARG designs from a variety of standard assumptions such as LWE, DLIN, sub-exponential DDH (and QR) [38, 37, 61, 78, 56, 44, 70, 36, 59, 58]. One of the reasons behind this success is: unlike (general-purpose) succinct non-interactive arguments (SNARGs) [63, 66], BARGs do not suffer from strong black-box barriers [49].

This work. We put forth a new notion of batch arguments, called *mutable batch arguments*. Our goal is to re-envision how we think about batch arguments. A batch argument encodes a sequence of k **NP** witnesses $\{\omega_i\}_i$ into a single short (proof) string π. Here proof π serves as a short, sound, and verifiable substitute for all witnesses $\{\omega_i\}_i$. Unfortunately, such a proof process creates *immutable* encodings π. That is, proofs are frozen in time and do not support general *mutation* operations that an actual witness does.


© Rishab Goyal;
licensed under Creative Commons License CC-BY 4.0


An example. Consider a user wants to store a large database of pictures on a cloud server (e.g., Alice storing pictures on Google drive). Suppose a server wants to maintain a "proof of legitimacy" for every user. That is, the server wants to maintain a proof certifying none of the pictures contain any illicit material (such as CSAM, copyrighted photos, etc). Such a feature would be tremendously useful for secure and efficient auditing purposes (e.g., FBI wants to verify that Google is not storing any illicit material on its servers). Although BARGs would seem a great tool for maintaining such proofs, they have serious limitations!

The problem. Consider that the user wants to edit some of its pictures by running a popular image filter on them. The server can apply such a filter efficiently, but then it must also generate a fresh proof of legitimacy for the user. This is highly inefficient as even a minor change to some data would require the server to re-generate a new cryptographic proof. Thus, a fascinating question is: can BARGs be used in applications where the data is *dynamic* and proofs need to be efficiently mutable?

Speaking more generally, we study a natural question towards advancing succinct proofs –

"Can we compute over succinctly proven data?"

Our results. We introduce the concept of *mutable BARGs*, and formally define a framework to capture computations over batch arguments. We have identified three non-trivial mutation classes (i.e., computation classes) – identity mutations, subset mutations, monotone policy batchNP mutations. Briefly, identity mutations enable a user to generate a "pseudo-witness" for any instance x_i from a batch proof π for a batch $\{x_j\}_j$. 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 $\{x_{1,j}\}_j, \{x_{2,j}\}_j, \ldots$ to create a new batch proof for any subset S of the underlying instances $\{x_{i,j}\}_{(i,j)\in S}$. And, monotone policy batchNP mutations generalize subset mutations by enabling a user to compute any monotone circuit over satisfiability of underlying instances. (We expand on these mutation classes and provide a detailed technical explanation of mutability in the next section.)

We design mutable BARGs for all three mutation class from standard assumptions and provide new applications. We also study strong privacy notions for mutable BARGs, which (informally) guarantee that a "mutated" proof does not reveal any non-trivial information about the original batch proofs or the instances. We provide a short summary of our results below:

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

Theorem 2 (informal). Assuming LWE, there exists mutable BARGs for *monotone-policy* batchNP mutation functions satisyfing succinctness, soundness, and privacy.

Applications. Assuming mutable BARGs for "X" mutations with mutation privacy, there exists "Y" signatures with privacy/context hiding. (see Table 1)

Table 1 Applications.

X (Mutation Class)	Y (Signatures)	Multi-Signer
Identity	Locally Verifiable Aggregate [52]	Yes
Subset	Redactable [57, 76]	Yes
Monotone-policy batchNP	Homomorphic [3, 15, 16]	Yes

We highlight that all our signature schemes can be instantiated in the multi-signer setting from standard assumptions. This gives the first constructions for: (1) multi-key homomorphic signatures with short signatures and verification key, (2) (post-quantum) redactable signatures with short signatures and verification key, and (3) multi-signer locally verifiable aggregate signatures, from standard falsifiable assumptions. We further compare our results with prior works in Section 2.

Computing over proofs. Computing over cryptographically proven data is a well-studied research topic. In the early 90s, De Santis and Yung [40] proposed *metaproofs* that enabled basic computations over proofs (i.e., proofs of proofs), and there has been a long line of follow-up works [26, 77, 8, 45, 35, 1, 19, 41, 30, 31, 32, 12, 5, 4, 7] studying numerous generalizations such as proof malleability [30, 31, 32], succinct recursive proofs [12], homomorphic zero-knowledge [4], etc. We follow this long line of work, and keep our focus on batch arguments. We study new feasibilities for computations over batch arguments, give new constructions, and discuss multiple applications, while anchoring to standard falsifiable assumptions [67].

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 mutable BARG framework.

Reviewing BARGs. BARGs are typically defined in the common reference string (CRS) model. 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 $|\mathsf{crs}|, |\pi| \leq \mathsf{poly}(\lambda, \log k)$, but they can grow with the size of a single witness. Soundness states that no polynomial time cheating prover can create an accepting proof π^* for some $\{x_i\}_i$ such that $x_i \notin \mathcal{L}$ for some i. In this work, we rely on a stronger notion of knowledge soundness, called "somewhere extractability" [37][1]. 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 x_{i^*} from any accepting proof π. Moreover, the crs does not leak the extraction index i^*.

Mutable BARGs. In short, mutable BARGs are regular BARGs with one special feature. They support a set of pre-defined mutation operations on top of batch proofs *without knowing 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, $\{(j_i, X_i, \pi_i)\}_{i \leq \ell}$, where each instance

[1] It has been used implicitly [37] and observed explicitly in prior works [44, 58] that somewhere extractability can be generically achieved by combining BARGs with any somewhere extractable hash (SEH) function [55, 69]. For completeness, we show this generic transformation formally in the full version.

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 ease of exposition, but our construction do satisfy varying levels of knowledge soundness. Additionally, we also study privacy 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 the full version, 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. We highlight that above changes are purely syntactic simplifications.

Local verifiability via Merkle Trees. 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 contains the digest h_x and an index i, while the actual instance x_i, its witness ω_i, and opening op_i is the new witness. In words, membership for language $\hat{\mathcal{L}}$ checks: (1) ω_i is a valid witness for x_i (w.r.t. $\mathcal{L}$), and (3) 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 (vanilla) BARG verifier can re-compute the digest h_x at verification time from the batch of instances.

To create efficient local openings for batch proofs (i.e., pseudo-witnesses), LOpen creates the digest h_x, and generates an opening op_j for target instance x_j. It sets the auxiliary opening as $\mathsf{aux}_j = (h_x, \mathsf{op}_j)$. The local verifier (i.e., mutated proof verification) then simply checks: (1) op_j is a valid opening for x_j w.r.t. h_x, and (2) π is a valid BARG proof for instances $\{(h_x, i)\}_i$. 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.

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. Coincidentally, the same approach was used in the [37] BARG construction, but rather for proving an online-offline verification property. As we explain above, we can re-purpose it as local verifiability. In order to prove stronger security properties, we use a somewhere extractable hash (SEH) function [55, 69] in the main body. We provide further details in the full version.

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.

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 also be 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 (i.e., it satisfies (a)), then our

intuition is that this will be enough to hide any non-trivial information about instances. 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. Unfortunately, this does not work.

Suppose the prover creates a fresh commitment to hide each instance x_i as $c_i = \mathsf{Com}(x_i; r_i)$, and then use the batch of commitments $\{c_i\}_i$ as the batch of instances while using (ω_i, x_i, r_i) as the corresponding 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: (1) ω_i is a valid witness for x_i, and (2) r_i is a valid opening of x_i (w.r.t. c_i). While this seems to give privacy, 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.

We use pseudorandom functions (PRFs) to solve the succinctness problem while ensuring privacy. 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 $\{(\omega_i, x_i, r_i = F_K(i))\}_i$ as the witnesses.
3. The new batch proof contains the above batch proof π and the PRF key K.

Clearly, a (vanilla) BARG verifier can re-compute all the k commitments given the PRF key K, thus correctness is unaffected. Let us see how to locally open and verify such proofs.

The local opening algorithm creates a local opening for the underlying BARG using $\{c_i\}_i$ as the instances[2], and then sets the opening information to be the underlying local opening along with the commitment opening $r_j = F_K(j)$ corresponding to x_j. That is, it omits the PRF key K from the batch proof, and instead adds the PRF evaluation for the target instance as the additional opening information. A local verifier can first verify the validity of the commitment, and then run the underlying local BARG verifier. 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 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 the full version.

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 [50]. Later we also discuss an alternate approach (of applying zero-knowledge in the end instead) to achieve mutation privacy for lv-BARGs highlighting several limitations of doing that.

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

[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 [29] 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

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.

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 $K\{j\} = \text{Puncture}(K, j)$ [20, 21, 62], 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 above design approach.

Very fast proof mutation. We also 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 *very 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 in the future.

Next, we provide a natural application of mutable BARGs supporting identity mutations to design aggregate signatures with fast local verification [52].

Application 1: locally verifiable aggregate signatures. An aggregate signature [17] scheme allows 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 [52], Goyal and Vaikuntanthan studied a new generalization of aggregate signatures. 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 all the 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 [52], or relied on SNARGs for **NP**.

Our starting point is a recent observation [78, 44] 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. By replacing the underlying BARG with a mutable BARG for identity mutations, the resulting aggregate signature scheme also satisfies desired local verifiability property. The resulting construction also satisfies strong message privacy for such aggregate signatures, which was not known previously. We refer to the full version 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}. While the mutated proof verification algorithm is defined to be the same algorithm as for non-mutated proof verification. We 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 captures the intuition of subset mutations, and contains necessary technical ideas.

Deletable BARGs via batching identity mutations. At first glance, it might appear that designing mutable BARGs for subset mutations could be more challenging. Interestingly, we show this is not the case, and design a generic approach from identity mutations to subset mutations. 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 LOpen on 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.

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 the full version.

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 [44, 70].

Application 2: redactable signatures. Redactable signatures [57, 76] 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 applications in privacy-preserving authentication as they can be used to sanitize digital signatures. (See [43, 42] for a detailed overview.) We show that deletable BARGs provide a clean and natural framework to design redactable signatures with many interesting properties.

Our intuition is to make a signer partition the document into equal sized chunks and sign each chunk[4] individually. For simplicity, one could consider breaking a long message bit-by-bit, and signing each bit individually along with its position in the message. 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 design), then it can run the proof deletion algorithm to delete desired chunks. The resulting deleted proof is then set as the redacted signature. Any user can verify the redacted signature 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.

[4] By a chunk, we refer to the smallest portion of the message that a user might want to redact. E.g., depending upon application, one can partition word-by-word or sentence-by-sentence.

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 the full version 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. [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 defined as $\mathbb{I}_{x \in \mathcal{L}} = 1$ iff $x \in \mathcal{L}$. [23] considered above languages where $\mathcal{C}$ contained all monotone circuits, and designed SNARGs for $\mathcal{L}_{\mathcal{C}}^{(k)}$.[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 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 $\mathcal{L}_{\mathcal{C}}^{(k)}$, as defined above. That is, $(x_{1,j_1}, \ldots, x_{\ell,j_\ell}) \in \mathcal{L}_{P_\ell}$ iff $(C, x_{1,j_1}, \ldots, x_{\ell,j_\ell}) \in \mathcal{L}_{\mathcal{C}}^{(\ell)}$. 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 this mutation class. Our construction relies on two core ingredients: (a) mutable BARGs for identity mutations (i.e., lv-BARGs), and (b) SNARGs for monotone policy batchNP.

Monotone policy batchNP mutations via batchNP SNARGs and lv-BARGs. Our intuition is that, to support $\mathcal{L}_{\mathcal{C}}^{(k)}$ mutations, it is sufficient to combine SNARGs for $\mathcal{L}_{\mathcal{C}}^{(k)}$ with any lc-BARG scheme. 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. Our main idea is as follows. The 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.

[5] We remark that prior works [23, 68] 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.

Unfortunately, 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 a stronger 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 [23], but it is unclear how to select the "necessary subset" non-adaptively to enable somewhere extraction. Fortunately, Brakerski et al. [23] also proved a stronger notion of non-adaptive full extractability for their SNARGs. Specifically, they proved their SNARG to be an argument of knowledge under the hardness of learning with errors assumption [72]. 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 the full version, we provide the above construction in full detail. Next, we overview the key ideas behind our homomorphic signature schemes based on mutable BARGs.

Application 3: homomorphic signatures. Homomorphic signatures [3, 15, 16] 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 [3, 15, 16, 27, 28, 51, 46], homomorphic signatures are very useful for numerous applications.

Our observation is that any mutable BARG system supporting monotone policy batchNP mutations can be used to design homomorphic signatures for 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. Let us first explain the core idea by evaluating monotone circuits, and later we discuss how this can be generalized for arbitrary non-monotone circuits as well.

Homomorphically evaluating monotone circuits. As a starting point, let us consider the simpler case of evaluating *monotone* circuits with single-bit output, and consider that the attacker's goal is to create a forgery σ' w.r.t. a monotone circuit C^*, such that the circuit's output is 0 on the queried dataset $M = (m_1, \ldots, m_\ell)$, 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 follows without loss of generality.

Our core insight 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 a 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 a batch proof π. Since we only want to guarantee 1-sided forgery, 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[6] in proving the evaluated circuit output to be 1.

[6] 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, 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 $X_i = (1, \ldots, \ell)$.[7] The proof of unforgeability boils down to the mutation (knowledge) soundness of the underlying mutable BARG. Moreover, 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 unforgeable and context hiding.

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 batchNP SNARGs), we show this can be done. 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 circuits into a monotone boolean circuits of similar size.

Once we do such a transformation, then we can just use the proof mutation algorithm as before to mutate a BARG proof 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 the full version, 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 shows additional advantages of using mutable batch arguments as a core cryptographic primitive for applications. Moreover, our template for designing homomorphic signatures also readily extends to handle homomorphic computations over signatures created by multiple signers (i.e., multi-key setting [46]) as well as multi-hop evaluation of already evaluated signatures.

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

Concurrent works. In a concurrent work, Anthoine et al. [6] designed multi-key homomorphic signatures [46] by combining batch arguments and functional commitments [64]. Their scheme only supports chained multi-hop evaluation (where only a single evaluated signature can be further evaluated), whereas our homomorphic signatures support general multi-hop evaluation (where multiple evaluated signatures can be further evaluated). However, they could support a polynomial number of multi-hop evaluations, while we can only support constant number of multi-hop evaluations.

In two independent works, Brodsky et al. [25] and Nassar et al. [68] 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 monotone-policy predicate P. Although these signatures seem incomporable to out applications, we believe we can also design them from our mutable BARGs by adjusting our homomorphic signatures. In a follow-up work, Afshar et al. [2] extended our techniques to design (multi-key) homomorphic signatures that could support polynomial number of multi-hop evaluations.

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

Mutable Batch Arguments from SNARKs. As a feasibility result, we outline a high level sketch for constructing general mutable BARGs from any succinct non-interactive arguments of knowledge (SNARKs) [63, 66]. The idea is to simply 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 $\mathcal{L}_P$. 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 them as the witness[8]. And, any cheating prover can be caught by running the SNARK extractor, and using that to break soundness of the underlying BARG scheme. This suggests that assuming SNARKs, we can design general-purpose mutable BARGs. Our focus is on designing mutable BARGs from standard falsifiable assumptions.

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.

[8] 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.

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.

Adaptive soundness: leveraging SEH is enough. Additionally, we show 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 [24] 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. While this does not impact our results, we formalize this later in the full version as it may be of independent interest[9].

Related work. BARGs for **NP** follow directly from any SNARG for **NP**, and starting with the work of Micali [66], 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 [54, 9, 34, 33, 75], or non-falsifiable assumptions [53, 11, 39, 65, 71, 48, 14, 13, 18, 10], or obfuscation [74]. Moreover, due to Gentry-Wichs [49], we know that we cannot build an adaptively sound SNARG for **NP** (via a black-box reduction) to a falsifiable assumption [67]. The landscape for BARGs for **NP** from standard assumptions is a lot more promising due to several outstanding results [60, 37, 38, 61, 78, 56, 36, 59, 44, 70, 58].

3 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:

$\mathsf{Eval}(\mathsf{crs}, P, \{(j_i, X_i, \pi_i)\}_{i\le\ell}) \to \widehat{\pi}$. 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.

[9] In an independent work, Bradley et al. [22] also observed that sub-exponentially somewhere sound BARGs are also adaptively sound. Our transformation 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.

(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.

Post-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 $|\widehat{\pi}| \leq \mathsf{poly}(\lambda, \max_i |\pi_i|, \log \ell)$.[10]

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. 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 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*

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

[10] 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.

▶ **Definition 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 new 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[11]. 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 3** (mutation privacy)**.** *A mutable BARG scheme* lv-BARG *for mutation class $\mathcal{P}$ satisfies mutation privacy if there exists a stateful PPT simulator $\mathcal{Sim}$ 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*

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

▶ Remark 4 (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.

[11] One could define privacy as mutated proof being indistinguishable from a fresh proof for a mutated instance. However, there is a technical reason we did not pursue it. In our setting, we expect a prover to only take original (non-mutated) instances-witnesses as inputs. Note that a mutated language and original **NP** language could be different. Therefore, creating a fresh proof for a mutated instance directly could be an undefined operation. Due to this, we defined privacy a bit differently.

4 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 $P = \mathbb{I}$. Further, it mutates only a single index-instances-proof tuple (j, X, π) (thus $\ell = 1$), and the associated language is the same, $\mathcal{L}_P = \mathcal{L}$. 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).

4.1 Specializing syntax and definition

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 (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.

4.1.1 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 6** (semi-adaptive local soundness with adversarial openings)**.** *A locally verifiable BARG scheme* lv-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*

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

We also consider the following local extraction soundness property.

▶ **Definition 7** (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*

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

▶ Remark 8 (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.

4.1.2 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 9** (instance privacy). *A locally verifiable BARG scheme* lv-BARG *satisfies instance privacy if there exists a PPT stateful 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*

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

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 10** (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.*

References

1 Tolga Acar and Lan Nguyen. Homomorphic proofs and applications, 2011.

2 Abtin Afshar, Jiaqi Cheng, and Rishab Goyal. Leveled fully-homomorphic signatures from batch arguments. Cryptology ePrint Archive, Paper 2024/931, 2024. URL: https://eprint.iacr.org/2024/931.

3 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. doi:10.1007/978-3-642-01957-9_18.

4 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. doi:10.1007/978-3-030-36033-7_14.

5 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. doi:10.1007/978-3-662-44381-1_10.

6 Gaspard Anthoine, David Balbás, and Dario Fiore. Fully-succinct multi-key homomorphic signatures from standard assumptions. In *Annual International Cryptology Conference*, pages 317–351. Springer, 2024. doi:10.1007/978-3-031-68382-4_10.

7 Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction andÂ randomization of groth's zk-snark. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 457–475, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg. doi:10.1007/978-3-662-64322-8_22.

8 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. doi:10.1007/978-3-642-03356-8_7.

9 Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive*, 2018.

10 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. doi:10.1007/s00145-016-9241-9.

11 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. doi:10.1145/2090236.2090263.

12 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. doi:10.1145/2488608.2488623.

13 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. doi:10.1145/2591796.2591859.

14 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. doi:10.1007/978-3-642-36594-2_18.

15 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.

16 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.

17 Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures. In *Eurocrypt*, 2003.

18 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. doi:10.1007/978-3-319-56617-7_9.

19 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. doi:10.1145/2090236.2090264.

20 Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT 2013*, 2013.

21 Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC 2014*, 2014.

22 Eli Bradley, Brent Waters, and David J Wu. Batch arguments to nizks from one-way functions. *Cryptology ePrint Archive*, 2023.

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. doi:10.1007/978-3-031-38545-2_9.

24 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.

25 Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In *EUROCRYPT*. Springer-Verlag, 2024.

26 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. doi:10.1007/S001459900053.

27 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. doi:10.1007/978-3-642-38348-9_21.

28 Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Annual Cryptology Conference*, pages 371–389. Springer, 2014. doi:10.1007/978-3-662-44371-2_21.

29 Jeffrey Champion and David J. Wu. Non-interactive zero-knowledge from non-interactive batch arguments. Cryptology ePrint Archive, Paper 2023/695, 2023.

30 Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 281–300. Springer, 2012. doi:10.1007/978-3-642-29011-4_18.

31 Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: Complex unary transformations and delegatable anonymous credentials. *Cryptology ePrint Archive*, 2013.

32 Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable nizks and an application to compact shuffles. In *Theory of Cryptography Conference*, pages 100–119. Springer, 2013. doi:10.1007/978-3-642-36594-2_6.

33 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. doi:10.1007/978-3-030-45721-1_26.

34 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. doi:10.1007/978-3-030-45721-1_27.

35 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. URL: http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/25.html.

36 Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and snargs from sub-exponential ddh. *Cryptology ePrint Archive*, 2022.

37 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.

38 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. doi:10.1007/978-3-030-84259-8_14.

39 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. doi:10.1007/978-3-642-28914-9_4.

40 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.

41 Alfredo De Santis and Moti Yung. "Metaproofs" (and their cryptographic applications). *Cryptology ePrint Archive*, 2012.

42 David Derler, Stephan Krenn, and Daniel Slamanig. Signer-anonymous designated-verifier redactable signatures for cloud-based data sharing. In *CANS*, 2016.

43 David Derler, Henrich C Pöhls, Kai Samelin, and Daniel Slamanig. A general framework for redactable signatures and new constructions. In *ICISC*, 2015.

44 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.

45 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. doi:10.1109/FOCS.2010.56.

46 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. doi:10.1007/978-3-662-53890-6_17.

47 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. doi:10.1007/978-3-031-22318-1_19.

48 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. doi:10.1007/978-3-642-38348-9_37.

49 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. doi:10.1145/1993636.1993651.

50 S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989. doi:10.1137/0218012.

51 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. doi:10.1145/2746539.2746576.

52 Rishab Goyal and Vinod Vaikuntanathan. Locally verifiable signature and key aggregation. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference*, 2022.

53 Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Asiacrypt*, volume 6477, pages 321–340. Springer, 2010. doi:10.1007/978-3-642-17373-8_19.

54 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. doi:10.1007/978-3-662-49896-5_11.

55 Pavel Hubacek 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*, pages 163–172, 2015.

56 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. doi:10.1007/978-3-031-07085-3_18.

57 Robert Johnson, David Molnar, Dawn Song, and David Wagner. Homomorphic signature schemes. In *CT-RSA*, 2002.

58 Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and ram delegation. In *STOC*, 2023.

59 Yael Tauman Kalai, Alex Lombardi, and Vinod Vaikuntanathan. Snargs and ppad hardness from the decisional diffie-hellman assumption. *Cryptology ePrint Archive*, 2022.

60 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. doi:10.1145/3313276.3316411.

61 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. doi:10.1007/978-3-030-90459-3_12.

62 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. doi:10.1145/2508859.2516668.

63 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.

64 Benoît Libert, Somindu C Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, 2016.

65 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. doi:10.1007/978-3-642-42033-7_3.

66 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. doi:10.1109/SFCS.1994.365746.

67 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. doi:10.1007/978-3-540-45146-4_6.

68 Shafik Nassar, Brent Waters, and David J Wu. Monotone policy bargs from bargs and additively homomorphic encryption. *Cryptology ePrint Archive*, 2023.

69 Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 121–145. Springer, 2015. doi:10.1007/978-3-662-48797-6_6.

70 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. doi:10.1109/FOCS54457.2022.00102.

71 Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016. doi:10.1145/2856449.

72 Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

73 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. doi:10.1145/2897518.2897652.

74 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. doi:10.1145/2591796.2591825.

75 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. doi:10.1007/978-3-030-56877-1_25.

76 Ron Steinfeld, Laurence Bull, and Yuliang Zheng. Content extraction signatures. In *ICISC*, 2001.

77 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. doi:10.1007/978-3-540-78524-8_1.

78 Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. *IACR Cryptol. ePrint Arch.*, page 336, 2022. URL: https://eprint.iacr.org/2022/336.