Folding Schemes with Privacy Preserving Selective Verification

Joan Boyar and Simon Erfurth

University of Southern Denmark, Odense, Denmark

Abstract. Folding schemes are an exciting new primitive, transforming the task of performing multiple zero-knowledge proofs of knowledge for a relation into performing just one zero-knowledge proof, for the same relation, and a number of cheap inclusion-proofs. Recently, folding schemes have been used to amortize the cost associated with proving different statements to multiple distinct verifiers, which has various applications. We observe that for these uses, leaking information about the statements folded together can be problematic, yet this happens with previous constructions. Towards resolving this issue, we give a natural definition of *privacy preserving* folding schemes, and what security they should offer. To construct privacy preserving folding schemes, we first define a *statement hiders*, a primitive which might be of independent interest. In a nutshell, a statement hider hides an instance of a relation as a new instance in the same relation. The new instance is in the relation if and only if the initial instance is. With this building block, we can utilize existing folding schemes to construct a privacy preserving folding scheme, by first hiding each of the statements. Folding schemes allow verifying that a statement was folded into another statement, while statement hiders allow verifying that a statement was hidden as another statement.

Keywords: folding schemes · SNARKs · delegation of computation

1 Introduction

Suppose that N clients outsource some computations to an untrusted server. This server does the computation (possibly with some additional secret data or a proprietary algorithm) and then wishes to prove to the clients that each of their computations was done correctly. One way this could be done is that for each of the N clients, the server provides a (non-interactive) zero-knowledge proof that the client's computation was done correctly. However, this requires doing N (potentially expensive) proofs, one for each of the clients. A *folding scheme* [KST22] allows the server to combine the N statements into just one statement of the same size as the initial statements. Additionally, the folding scheme produces a *folding proof*, which proves that all the statements were folded into the final statement. Thus, the server can prove just the final statement, and distribute the non-interactive zero-knowledge proof of it being correct together with the folding proof to the clients, and all of them should be convinced that their computations were done correctly. We refer to the server as the *prover* and to each client as a *verifier*.

To be more specific, a folding scheme for an NP-language $\mathcal{L}$ with relation

$$\mathcal{R} = \{(x, w) \mid w \text{ is a proof that } x \in \mathcal{L}\},$$

can combine instances $(x_i, w_i) \in \mathcal{R}$ for $1 \leq i \leq N$ into one instance $(x, w) \in \mathcal{R}$. Intuitively, (x, w) is in $\mathcal{R}$ if and only if (x_i, w_i) is in $\mathcal{R}$ for $1 \leq i \leq N$. Additionally, a folding scheme

E-mail: joan@imada.sdu.dk (Joan Boyar), simon@serfurth.dk (Simon Erfurth)

Date of this document: 2024-09-30.

produces a folding proof π, that can be used to verify that x was produced by folding the x_i's, even though the verifiers do not necessarily learn w or any of the w_i's.

The folding proof proves that all the statements have been folded into the final statement, and hence, its size is generally $\Omega(N)$. There are two issues with this: (1) Sending a size $\Omega(N)$ folding proof to every verifier is wasteful, if each party only needs to verify that their statement was folded into the final statement. (2) Verifying a folding proof requires knowledge of all statements folded into the final statement, which in a multi-verifier setting has obvious privacy concerns. The first issue has led to the development of *folding schemes with selective verification* [RZ23]. These folding schemes support generating separate proofs of folding for each of the N statements folded together. Each proof of folding π_i should be of size $o(N)$, and need only prove that x_i was folded into the final statement.

However, the folding schemes with selective verification from [RZ23] do not resolve the second issue. Specifically, any folding proof for a specific statement includes either the statement before or the statement after that specific statement. Since each proof of folding is only required to prove that the corresponding statement was folded into the final statement, it is natural to require the proof of folding to preserve the privacy of all other statements, ensuring that no verifier learns anything about the other verifiers' statements.

In this work, we introduce *folding schemes with privacy preserving selective verification*, which resolves both issues (1) and (2). Privacy preserving should be understood as meaning that if a verifier's statement is folded together with other statements, and selective folding proofs are generated and distributed by the prover, other verifiers might learn that the statement is in the language (since the final statement is proven to be in the language), but they will have no idea which statement in the language it is. In practice, we work with an indistinguishability notion, where an adversary chooses two distinct indices i and ℓ and the entire input to the folding scheme, including two potential inputs for the i'th spot. One of the potential i'th inputs is then chosen at random, folding and proof generation is done, and the adversary is then given the statement obtained by folding, and the selective proof of folding for input ℓ. Finally, the adversary has to guess which of the statements was used as the i'th input. We say that the folding scheme is privacy preserving, if no adversary guesses correctly with probability more than $1/2 + \mathsf{negl}(\lambda)$, where λ is the security parameter.

Toward constructing folding schemes with privacy preserving selective verification, we define a new primitive, which we call an *NP-statement hider*. This primitive is used by the prover, hiding one instance, (x, w), as another instance, (x', w'), and producing a certificate for verifying that x' is hiding x. Using an NP-statement hider and a folding scheme with selective verification as building blocks, we present a generic construction of a folding scheme with privacy preserving selective verification, and show that it satisfies our definition of being privacy preserving. Thus, to extend a folding scheme with selective verification to one with privacy preserving selective verification, it is sufficient to construct a corresponding NP-statement hider. To facilitate this, we present a generic construction of an NP-statement hider, utilizing a folding scheme (which is not required to be privacy preserving). Essentially, the NP-statement hider works by folding the statement to be hidden with a randomly sampled statement. There is evidence that not all folding schemes will allow the required random sampling, but we demonstrate that there is one based on an NP-hard problem that does. Security of the constructed NP-statement hider follows from the security of the underlying folding scheme, and an additional property, which is essentially that for any three instances, there exists a fourth instance, such that the instance obtained by folding the first two instances, is the same as the instance obtained by folding the last two instances. Having this property results in information theoretically hiding NP-statement hiders. We informally state our results in the following theorem.

Theorem 1 (Combining Theorems 2 and 3)**.** *Let $\mathcal{L}$ be a language with relation $\mathcal{R}$. If there is a folding scheme for $\mathcal{L}$, $\mathcal{R}$ supports efficient sampling of instances, and for any* *three instances* $(x_1, w_1), (x_2, w_2), (x_3, w_3) \in \mathcal{R}$*, there exists a fourth instance* $(x_4, w_4) \in \mathcal{R}$*, such that folding* (x_1, w_1) *and* (x_2, w_2) *gives the same instance of* $\mathcal{R}$ *as folding* (x_3, w_3) *and* (x_4, w_4)*, then there is a folding scheme with privacy preserving selective verification for* $\mathcal{L}$*.*

We apply our constructions to some example folding schemes for algebraic NP-languages. As a warm-up problem, we consider the language *Inner Product Relation of Committed Values* [BCC+16, RZ23]. Then we consider the NP-complete language *Committed Relaxed R1CS* [KST22], which is also the original language used for folding schemes. We show that both these languages satisfy the conditions of Theorem 1.

1.1 Organization of paper

In Section 1.2 we review related work and in Section 1.3 we consider possible applications for our work. We review folding schemes in Sections 2 and 2.1, and folding scheme with selective verification in Section 2.2. After this, we define privacy preserving selective verification in Section 3 and NP-statement hiders in Section 3.1. We construct a privacy preserving folding scheme using an NP-statement hider in a black-box fashion in Section 3.2, and a NP-statement hider using a folding scheme in Section 3.3. Finally, in Section 4, we apply our constructions to two concrete languages.

1.2 Related Work

Folding schemes were introduced by Kothapalli, Setty, and Tzialla at CRYPTO'22 [KST22], as a tool to realize incrementally verifiable computation (IVC) [Val08]. IVC, as the name hints, is a method to do computations, such that the correctness of the entire computation can be verified by checking each increment of the computation. Historically, IVC has been constructed using recursive *succinct non-interactive arguments of knowledge* (SNARKs) to prove that each increment was computed correctly. More recently, *accumulators* have been developed [BGH19, BCMS20, BDFG21, BCL+21]. Rather than verifying a SNARK at every increment, an accumulator based scheme allows the SNARK check to be accumulated into the checks from previous increments. At a later time, all steps can be verified by checking a single SNARK, and that the accumulations has been performed correctly at each step. This can be significantly more efficient than checking a SNARK for each step, and communicating the single SNARK and the folding proof requires less communication than communicating a SNARK for each step. The most efficient type of accumulation schemes are folding schemes [NDC+24], and allow one to combine the proofs that each step was computed correctly into one single proof of the same size. Folding schemes yield IVC constructions where the recursive proof that folding (accumulation) was done correctly at each step is dominated by two elliptic curve scalar multiplications, and where the only needed assumption is the discrete logarithm assumption in the random oracle model [KST22]. Nova [KST22] introduced the notion of folding schemes. Since then, folding schemes have attracted much interest for IVC, leading to the development of many folding schemes, for example SuperNova/HyperNova [KS22, KS24], Protostar [BC23], LatticeFold [BC24], and Mangrove [NDC+24].

Recently, the *Reductions of Knowledge* framework [KP23, Kot24] was introduced by Kothapalli, one of the authors introducing folding schemes, and Parno. Reductions of knowledge generalizes many flavors of arguments of knowledge, including folding schemes. Generally, a reduction of knowledge reduces checking knowledge of a witness for a statement from one relation, to checking knowledge of a witness for a statement from another (usually simpler) relation. In this framework, a 2-folding scheme for a relation $\mathcal{R}$, is a reduction from $\mathcal{R} \times \mathcal{R}$ to $\mathcal{R}$. Specifically, knowing witnesses (w_1, w_2) to the instance $((x_1, w_1), (x_2, x_2)) \in \mathcal{R} \times \mathcal{R}$ is reduced to knowing witness w to instance $(x, w) \in \mathcal{R}$. While reductions of knowledge do not have folding proofs like folding schemes do, they instead have a requirement that the reduction is *publicly reducible*: given the initial statement(s) and the transcript, any party can reconstruct the final statement. We note that for the folding schemes we consider, the proof of folding is the first message from the prover, and checking it is exactly reconstructing the final statement.

Ràfols and Zacharakis [RZ23] considers a novel use of folding schemes, by modifying them to allow *selective verification*. Where the original version of folding schemes only considers a single verifier verifying all the proofs, and therefore only support verifying that *all* of the statements are folded into the final statement, folding schemes with selective verification instead consider multiple verifiers, where each verifier only needs to verify that a subset of the statements are folded into the final statement. Folding schemes with selective verification supports this by generating separate folding proofs for each statement, where each proof only verifies that the matching statement is folded into the final statement. A standard requirement is that each of these proofs should have size sub-linear in the total number of statements. Folding schemes with selective verification, are particularly useful in situations where folding proofs are not used as part of incrementally verifiable computations, but rather for verification of delegated computations. Specifically, if many clients outsource their distinct-but-similar computations to a server, and the server has to prove to the clients that it performed the correct computations, it might be more efficient to fold allof the proofs into one, rather than separately proving to each client that their computation was done correctly. In this case, rather than sending every client the full folding proof (and all other statements that are folded), the server can send each client only the proof that their statement was folded into the statement that was proven.

Related to folding schemes with selective verification, and hence also to our work, the polynomial commitment scheme, hbPolyCommit, from [YLF$^+$22] uses a Merkle tree structure to amortize the cost of batch processing multiple inner-product arguments, each corresponding to multiple verifiers. The commitment scheme uses the Merkle tree when combining multiple protocol transcripts to produce a challenge. The Merkle tree structure allows each party to verify that their transcript was considered, at a cost that is logarithmic in the number of transcripts. Folding schemes with selective verification differs by considering aggregation of multiple statements into one that is then proved, rather than aggregating multiple proofs together.

1.3 Applications

Folding schemes were initially developed for incrementally verifiable computing, but have since then had multiple other applications. We describe three applications, where folding schemes with privacy preserving selective verification might be useful.

One application, suggested in [RZ23], is for verification in *computation as a service*. Consider a case where many clients (verifiers) delegate similar computations on different inputs to a server (prover). In a trustless setting where interaction is very expensive or impossible, a standard solution is for the prover to use SNARKs to convince the verifiers, that their computations have been performed correctly. The application of folding schemes is straightforward: it amortizes the cost of proving a statement out over all the verifiers' computations, rather than having to prove a statement for each client. Selective verification reduces the communication to each verifier; they need only verify the correctness of their own computations. Privacy preserving selective verification additionally guarantees that the folding proofs do not leak information about other verifiers' computations.

A second application suggested in [RZ23], uses folding proofs with selective verification to share a *verifiable database*. In a verifiable database, clients (verifiers) outsource a database to a server (prover) in a trustless setting. Typically, the verifiers only store a short digest of the database, which allows querying and modifying the database. Viewing the database as a vector, the digest is a homomorphic vector commitment [CF13, CNR$^+$22], querying is simply opening the commitment at a specific location, and modifying is subtracting the original value from the commitment and adding the new one. Rather than opening a commitment for every query, the prover might batch up multiple proofs of opening, fold them together, and then send the SNARK for the folded statement to each verifier with a query in the batch. Again, privacy preserving selective verification both reduces the communication to each verifier and also guarantees that each verifier does not learn what data the other verifiers queried. Privacy preservation would be particularly important in a setting where verifiers might have different privileges, and hence be allowed to access different parts of the database.

Finally, a third application relates to *mitigating the effects of fake news*. While the traditional approach, has been to attempt to flag fake news as such, there has recently been a move towards also flagging authentic content as such. For images, the Adobe lead C2PA initiative [C2P], is currently starting to gain broader adaptation, with both Google and OpenAI having recently joined C2PA. Roughly, the solution proposed by C2PA is to have cameras sign images when they are captured, and then have C2PA compatible programs sign that the edits done to the image are legitimate. When viewing the image, the last signature can be verified, and, ideally, a chain of trust guarantees the authenticity of the image. However, C2PA's approach requires trusting the tools used to edit the image. One approach for resolving this issue, is to use image specific signatures allowing some modifications to be made to the image. However, these signatures comes with significant drawbacks, such as supporting only a very limited number of transformations [Erf24], having significant space overheads [JWL11], or only working with rarely used image formats [ZSL04]. Another common approach, is to use zero-knowledge SNARKs to prove that only certain edits have been applied to an image [NT16]. While this approach is more versatile in which edits it allows, imposes minimal space overhead, and allows efficient verification, it comes with a significant performance costs to the prover. Even the most recent construction takes time on the order of a few minutes to an hour, to generate proofs for a single image [DCB24]. Here, folding proofs with privacy preserving selective verification could be used to combine the proofs corresponding to many images together, potentially amortizing the cost of proving over many images. For example, suppose that a large (untrusted) social media wishes to support the C2PA approach, but still needs to compress the images uploaded to their platform. Rather than separately proving that each image was compressed by them, they could fold the proofs of many images uploaded in a small time-slot together, using a folding scheme. Selective verification would be sensible, since most likely a user only needs to verify one image at a time. Privacy preservation would be a necessity, since images that are not posted publicly (for example images sent in a private chat) should stay private.

We note that very recently, folding schemes have actually been used to in exactly this context [DEH24]. However, they focus on improving the costs associated with one image, whereas folding schemes with privacy preserving selective verification could be used to amortize this cost out over many images.

1.4 Notation

Generally, we denote single elements a using lowercase normal weight letters, vectors $\mathbf{a}$ using lowercase bold letters, and matrices A using uppercase normal weight letters. For tuples, we will occasionally be using notation of the form $(x, y, z = (a, b))$. This should be understood as the tuple (x, y, z) where $z = (a, b)$. Similarly, $\{y_i = (a_i, b_i)\}_{1 \leq i \leq n}$ should be understood as the set $\{y_i\}_{1 \leq i \leq n}$ where each $y_i = (a_i, b_i)$. For arrows, we use $x \leftarrow_{\$} X$ to denote that x is sampled uniformly from the set X, and $x \leftarrow \mathsf{A}(y)$ to denote that the output of algorithm A on input y is x.

We use additive group notation for cyclic groups, and let $\mathsf{gk} \leftarrow \mathcal{G}(1^\lambda)$ be the description of a group $\mathbb{G}$ over a field $\mathbb{F}$ sampled by a group generation algorithm. A description of a group is $\mathsf{gk} = (\mathbb{G}, \mathcal{P}, p)$, where $\mathbb{G}$ is a finite cyclic group of prime order p and $\mathcal{P}$ is a generator of $\mathbb{G}$. For $\mathcal{P}$ fixed, we denote with $[x]$ the element $x\mathcal{P}$, and let this notation extend naturally to vectors $[\mathbf{v}] \in \mathbb{G}^n$.

2 Folding Schemes

In this section we recall the definition of folding schemes [KST22] and folding schemes with selective verification [RZ23].

As mentioned in the introduction, folding schemes are schemes that allow folding two (or more) NP statements from a language $\mathcal{L}$ into one statement from $\mathcal{L}$. Intuitively, a statement produced by folding two or more statements, is in $\mathcal{L}$ if and only if all the individual statements are in $\mathcal{L}$. We begin with an informal description of 2-folding schemes, before moving on to a definition of N-folding schemes. Given a NP language $\mathcal{L}$ and a corresponding relation

$$\mathcal{R} = \{(x, w) \mid w \text{ is a witness for } x \in \mathcal{L}\},$$

a folding scheme allows efficiently reducing two instances $(x_1, w_1), (x_2, w_2)$ to one instance (x, w).We say that x is obtained by *folding* x_1 and x_2. The folding scheme is also required to output a *folding proof* π that x is the result of folding x_1 and x_2. This proof, together with x, x_1 and x_2, should be a convincing proof that x was formed by folding x_1 and x_2. Similar to standard proofs/arguments of knowledge, folding schemes should essentially have the following properties:

- **Completeness:** If $y_1 = (x_1, w_1) \in \mathcal{R}$ and $y_2 = (x_2, w_2) \in \mathcal{R}$, and folding y_1 and y_2 gives $y = (x, w)$, then $(x, w) \in \mathcal{R}$. Additionally, the folding proof π is accepted.

- **Knowledge soundness:** If (x, w) is the result of folding (x_1, w_1) and (x_2, w_2), and w is a witness that $x \in \mathcal{L}$, then w_i is a witness that $x_i \in \mathcal{L}$ for $i \in \{1, 2\}$.

Following the definition of [RZ23], we formally define N-folding schemes as follows.

Definition 1 (N-Folding Scheme)**.** For security parameter $\lambda \in \mathbb{N}$, NP language $\mathcal{L}_{\mathsf{pp}}$ parameterized by[1] $\mathsf{pp} \leftarrow \mathsf{pp}(\lambda)$, $\mathcal{R}_{\mathsf{pp}}$ the relation for $\mathcal{L}_{\mathsf{pp}}$, and $N = \mathsf{poly}(\lambda)$, an *$N$-folding scheme* FS for the language family $\{\mathcal{L}_{\mathsf{pp}}\}_{\mathsf{pp} \leftarrow \mathsf{pp}(\lambda)}$ is a tuple of algorithms $(\mathsf{Fold}, \mathsf{FoldVerify})$ which for $n \leq N$ operates as follows.

- $(x, w, \pi) \leftarrow \mathsf{Fold}(\mathsf{pp}, (x_1, w_1), \ldots, (x_n, w_n))$. On input parameters pp, and n instances $(x_i, w_i) \in \mathcal{R}_{\mathsf{pp}}$, Fold outputs an instance (x, w) from $\mathcal{R}_{\mathsf{pp}}$ and a folding proof π.

- $0/1 \leftarrow \mathsf{FoldVerify}(\mathsf{pp}, x_1, \ldots, x_n, x, \pi)$. On input parameters pp, $n + 1$ statements $x_1, \ldots, x_n$ and x, and a folding proof π, $\mathsf{FoldVerify}$ outputs 1 if x is the output of folding $x_1, \ldots, x_n$, and 0 otherwise.

Additionally, FS must satisfy the following properties:

- **Completeness:** For all adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \{y_i\}_{1 \leq i \leq n} \subseteq \mathcal{R}_{\mathsf{pp}} \wedge \\ ((x, w) \notin \mathcal{R}_{\mathsf{pp}} \vee b = 0) \end{array} \middle| \begin{array}{c} \{y_i = (x_i, w_i)\}_{1 \leq i \leq n} \leftarrow \mathcal{A}(\mathsf{pp}) \\ (x, w, \pi) \leftarrow \mathsf{Fold}(\mathsf{pp}, y_1, \ldots, y_n) \\ b \leftarrow \mathsf{FoldVerify}(\mathsf{pp}, x_1, \ldots, x_n, x, \pi) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

  Note that we allow $\mathcal{A}$ to be computationally unbounded.

[1] Here we abuse notation. When writing $\mathsf{pp}(\lambda)$ we refer to a (randomised and polynomial time) algorithm that on input the security parameter outputs parameter pp.

- **Knowledge soundness:** There exists a probabilistic polynomial time (PPT) extractor Ext, such that for all PPT adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c|c} (x,w) \notin \mathcal{R}_{\mathsf{pp}} \vee b = 0 \vee & (\{x_i\}_{1\le i\le n}, x, w, \pi) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \{(x_i,w_i)\}_{1\le i\le n} \subseteq \mathcal{R}_{\mathsf{pp}} & b \leftarrow \mathsf{FoldVerify}(\mathsf{pp}, x_1, \dots, x_n, x, \pi) \\ & \{w_i\}_{1\le i\le n} \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{pp}) \end{array}\right] \ge 1 - \mathsf{negl}(\lambda).$$

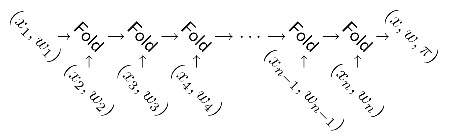
2.1 Bootstrapping from 2-folding to N-folding

Generally, folding schemes are constructed as 2-folding schemes, and then turned into N-folding schemes by recursive invocations. Let $\mathsf{FS} = (\mathsf{Fold}, \mathsf{FoldVerify})$ be any 2-folding scheme for a language $\mathcal{L}_{\mathsf{pp}}$ with relation $\mathcal{R}_{\mathsf{pp}}$. As an example, we construct a 3-folding scheme. Given 3 instances $(x_i, w_i) \in \mathcal{R}$, we fold the three instances into one by first folding two instances into one, and then folding this new instance and the third instance to obtain one final instance:

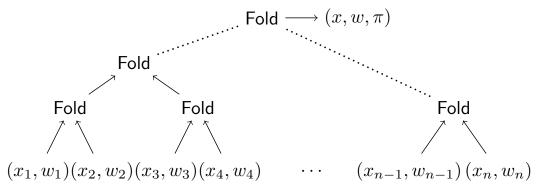
$$\begin{aligned}(x', w', \pi') &\leftarrow \mathsf{Fold}(\mathsf{pp}, (x_1, w_1), (x_2, w_2)),\\ (x, w, \pi'') &\leftarrow \mathsf{Fold}(\mathsf{pp}, (x', w'), (x_3, w_3)).\end{aligned}$$

Now the fold of all three instances is $(x, w, \pi = (\pi', \pi''))$. Observe that the folding proof of the 3-folding scheme consists of the folding proofs from both applications of FS.

The essential property making this construction possible, is that the statement generated by a folding scheme is in the same language and of the same size as the original statements. Thus, any 2-folding scheme can immediately be applied in a *bootstrap*-like way to create an N-folding scheme for $N = \mathsf{poly}(\lambda)$. This can be done in many ways, i.e., by chaining the statements together one after the other, or by creating a Merkle tree-like [Mer80, Mer89] structure, see Figures 1a and 1b. Regardless of which approach is used, the folding proof from the N-folding scheme is the accumulated folding proofs from the applications of the 2-folding scheme. Completeness of the N- folding scheme follows immediately from the construction, and observing that an adversary only has a negligible chance of cheating at each step and there are a polynomial number of folds[2], since $N = \mathsf{poly}(\lambda)$. Knowledge soundness takes more care, but essentially one can construct an extractor by recursively using the extractor for the scheme being bootstrapped. This method results in quasilinear overhead over the extractor for the 2-folding scheme, and once again the probability of extracting valid witnesses is polynomially related to the probability of the extractor for the 2-folding scheme extracting valid witnesses. In [RZ23], they give more details on bootstrapping with a Merkle tree-like structure, and shows that the bootstrapped N-folding scheme is both complete and has knowledge soundness.



(a) Chaining approach.


Fold
(x, w, π)
Fold
Fold
Fold
Fold
$(x_1, w_1)(x_2, w_2)(x_3, w_3)(x_4, w_4)$
$\cdots$
$(x_{n-1}, w_{n-1})\,(x_n, w_n)$

(b) Merkle tree approach.

Figure 1: The chaining and Merkle tree approaches to folding n instances (x_i, w_i) into one instance (x, w), using $n - 1$ applications of the 2-folding scheme $\mathsf{FS} = (\mathsf{Fold}, \mathsf{FoldVerify})$.

[2]Both constructions use exactly $N - 1$ folds. The Merkle tree approach has an advantage in that it can naturally be parallelized, both when folding and when verifying

2.2 Selective Verification

The folding scheme verification algorithm from Definition 1, takes as input all the folded statements. However, when the number of folded statements is large, this can be very costly if one only wishes to confirm that a single statement x_i was folded into the proven statement x. To solve this issue, [RZ23] introduces folding schemes with selective verification. Rather than having one folding proof π that verifies that all N statements $x_1, \ldots, x_N$ were folded into one statement x, they have N proofs $\pi_1, \ldots, \pi_N$. For each $i \in \{1, \ldots, N\}$, the i'th proof π_i together with x_i and x proves that x_i was folded into x. Note that the size of the proofs should be sublinear in N, since otherwise one could just set $\pi_i = (\pi, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N)$. Formally, a folding scheme with selective verification is defined as follows.

Definition 2 (Folding Scheme with Selective Verification)**.** For security parameter $\lambda \in \mathbb{N}$, NP language $\mathcal{L}_{\mathsf{pp}}$ parameterized by $\mathsf{pp} \leftarrow \mathsf{pp}(\lambda)$, $\mathcal{R}_{\mathsf{pp}}$ the relation for $\mathcal{L}_{\mathsf{pp}}$, $N = \mathsf{poly}(\lambda)$, an N-folding scheme $\mathsf{FS} = (\mathsf{Fold}, \mathsf{FoldVerify})$ for the language family $\{\mathcal{L}_{\mathsf{pp}}\}_{\mathsf{pp} \leftarrow \mathsf{pp}(\lambda)}$, has *selective verification*, if there is a tuple of algorithms $(\mathsf{SlctProve}, \mathsf{SlctVerify})$ which for $n \leq N$ operates as follows.

- $(\pi_1, \ldots, \pi_n) \leftarrow \mathsf{SlctProve}(\mathsf{pp}, x_1, \ldots, x_n, x, \pi)$. On input parameters pp, $n+1$ statements $x_1, \ldots, x_n$ and x, and folding proof π, $\mathsf{SlctProve}$ outputs proofs $\pi_1, \ldots, \pi_n$.
- $0/1 \leftarrow \mathsf{SlctVerify}(\mathsf{pp}, x, i, x_i, \pi_i)$. On input parameters pp, statements x and x_i, integer $i \in \{1, \ldots, n\}$, and folding proof π_i, $\mathsf{SlctVerify}$ outputs 1 if x_i is folded into x.

Additionally, the following properties must be satisfied.

- **Selective completeness:** For all adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c} \{y_i\}_{1 \leq i \leq n} \subseteq \mathcal{R}_{\mathsf{pp}} \\ \wedge\ (1 \leq j \leq n) \\ \wedge\ b = 0 \end{array} \middle| \begin{array}{c} (\{y_i = (x_i, w_i)\}_{1 \leq i \leq n}, j) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (x, w, \pi) \leftarrow \mathsf{Fold}(\mathsf{pp}, y_1, \ldots, y_n) \\ (\pi_1, \ldots, \pi_n) \leftarrow \mathsf{SlctProve}(\mathsf{pp}, x_1, \ldots, x_n, x, \pi) \\ b \leftarrow \mathsf{SlctVerify}(\mathsf{pp}, x, j, x_j, \pi_j) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- **Selective knowledge soundness:** There exists a PPT extractor Ext such that for all PPT adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c} (x, w) \notin \mathcal{R}_{\mathsf{pp}} \vee b = 0 \vee \\ (x_i, w_i) \in \mathcal{R}_{\mathsf{pp}} \end{array} \middle| \begin{array}{c} (i, x_i, \pi_i, x, w) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \leftarrow \mathsf{SlctVerify}(\mathsf{pp}, x, i, x_i, \pi_i) \\ w_i \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{pp}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

- **Efficiency:** The size of each π_i is sublinear in n, i.e., $|\pi_i| = o(n)$.

Remark 1. A folding scheme can be equipped with selective verification as follows. First, the Merkle tree-like bootstrapping construction, illustrated in Figure 1b, is used to get an N-folding scheme from a 2-folding scheme. Each π_i consists of the folding proofs for the 2-folding schemes used on the path between x_i and x, together with the statements these 2-folding schemes take as input, that are not already on the path between x_i and x.

For example, either x_{i-1} or x_{i+1} will be in π_i, since one of these is input to the 2-folding scheme taking x_i as input. However, x_i will not be in π_i, since it is on the path between x_i and x.

When this approach is used to make a folding scheme satisfying Definition 1 selective verifiable, it follows relatively straightforwardly that the bootstrapped construction satisfies Definition 2. Selective completeness follows immediately from the construction, and selective knowledge soundness from an argument similar to the argument for the bootstrapped construction having knowledge soundness, except that one now only need to follow one path from the root to x_i. Once again, more details can be found in [RZ23], where full algorithms for $\mathsf{SlctProve}$ and $\mathsf{SlctVerify}$ can also be found. For efficiency, it is easily observed that the path between x_i and x has length $O(\log n)$, and that each instance of the 2-folding scheme along the path results in 1 folding proof and requires 1 extra statement. Since both of these are constant sized with respect to the number of statements folded together, the size of π_i is $O(\log N)$, and hence sublinear in N. The notion of selective verification can be generalized to folding proofs for subsets of $\{x_i, \ldots, x_N\}$. From the Merkle tree literature [BELN23], it follows that in the case where one allows arbitrary subsets of the x_i, the number of additional statements one needs to provide might be linear in N, but if one requires the subset to be consecutive statements, the number of additional statements is still guaranteed to be logarithmic in N. However, one would still need to provide up to $2N - 1$ folding proofs from the underlying 2-folding scheme.

3 Privacy Preserving Selective Verification

The original definition of folding schemes has no notion of a folding scheme being "private", which makes sense since knowledge of all x_i's folded into x is required to verify the folding proof for x. Thus, there is in some sense nothing to be kept private, but the witnesses. However, for folding schemes with selective verification it seems natural to define *privacy preserving selective verification*, which, informally, extends Definition 2 with a guarantee that a folding proof π_i for x_i does not leak information about x_j for $j \neq i$. It can immediately be observed that the folding scheme with selective verification described in Remark 1 is not privacy preserving; any π_i includes either x_{i-1} or x_{i+1}. In this section, we first define *folding schemes with privacy preserving selective verification*, and then discuss a general approach to making folding schemes with selective verification privacy preserving, using a generic mechanism for hiding NP statements, which we formally define in Definition 4. This results in Construction 1 and Theorem 2. It is possible to construct hiding mechanisms for (some) NP languages from folding schemes, which we do in Construction 2 and Theorem 3. In Section 4, we give examples of folding schemes with privacy preserving selective verification, using the mechanisms from this section.

At the core of our definition of a folding scheme being privacy preserving, is a notion of indistinguishability under chosen-message attack. In our definition, we allow an adversary to choose an index j it wishes to attack, an index $\ell \neq j$ for which it will get the proof π_ℓ, the (valid) inputs (x_i, w_i) for all indices $i \neq j$, and two (valid) potential inputs (x_j^0, w_j^0) and (x_j^1, w_j^1) for j. For random $b \leftarrow_\$ \{0, 1\}$, folding is then done with (x_j^b, w_j^b), and the selective verification folding proofs are generated. Finally, the adversary is given x and π_ℓ, and has to guess b. For simplicity, we use a pair of algorithms as the adversary, but allow passing information from the first algorithm to the second through a state s. A folding scheme has privacy preserving selective verification if the probability of any adversary guessing correctly is only negligibly better than $\frac{1}{2}$. We formally define this in Definition 3.

Definition 3 (Folding Schemes with Privacy Preserving Selective Verification)**.** For security parameter $\lambda \in \mathbb{N}$, NP language $\mathcal{L}_{\mathsf{pp}}$ parameterized by $\mathsf{pp} \leftarrow \mathsf{pp}(\lambda)$, $\mathcal{R}_{\mathsf{pp}}$ the relation for $\mathcal{L}_{\mathsf{pp}}$, $N = \mathsf{poly}(\lambda)$, an N-folding scheme with selective verification $\mathsf{FS} = (\mathsf{Fold}, \mathsf{FoldVerify}, \mathsf{SlctProve}, \mathsf{SlctVerify})$ for the language family $\{\mathcal{L}_{\mathsf{pp}}\}_{\mathsf{pp} \leftarrow \mathsf{pp}(\lambda)}$, is said to be a *folding scheme with privacy preserving selective verification* if for $n \leq N$ and all adversaries $\mathcal{A}$ consisting of a pair of algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ is PPT,

$$\Pr\left[\begin{array}{c} \{y_i\}_{\substack{1\le i\le n\\ i\neq j}} \subseteq \mathcal{R}_{\mathsf{pp}} \wedge \\ \{(x_j^0, w_j^0), (x_j^1, w_j^1)\} \subseteq \mathcal{R}_{\mathsf{pp}} \\ \wedge\ \ell \neq j\ \wedge\ b' = b \end{array}\ \middle|\ \begin{array}{c} \begin{pmatrix} \ell, j, \{y_i = (x_i, w_i)\}_{\substack{1\le i\le n,\\ i\neq j}} \\ (x_j^0, w_j^0), (x_j^1, w_j^1), s \end{pmatrix} \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ b \leftarrow_{\$} \{0,1\} \\ (x, w, \pi) \leftarrow \mathsf{Fold}(\mathsf{pp}, y_1, \ldots, (x_j^b, w_j^b), \ldots, y_n) \\ (\pi_1, \ldots, \pi_n) \leftarrow \mathsf{SlctProve}(\mathsf{pp}, x_1, \ldots, x_j^b, \ldots, x_n, x, \pi) \\ b' \leftarrow \mathcal{A}_2(\mathsf{pp}, x, \ell, x_\ell, \pi_\ell, s) \end{array}\right] \le \frac{1}{2} + \mathsf{negl}(\lambda).$$

3.1 NP-statement hider

As previously mentioned, the construction from Remark 1 does not satisfy Definition 3. However, we observe that if the prover somehow "hides" the statements before folding them, we can reuse the construction. This motivates the following definition of a hiding mechanism, which on an instance $(x, w) \in \mathcal{R}$ and randomness $r \in \mathfrak{R}$ "hides" (x, w) as $(x', w') \in \mathcal{R}$. The hiding mechanism also outputs a certificate c, which can be used to verify that x' is hiding x. This certificate could, for example, include the randomness r. With such a mechanism, it is straightforward to get a folding scheme with privacy preserving selective verification. First, each instance is hidden, then all the hidden instances are folded, and finally the selective folding proofs π_i are updated to also include c_i. Crucially, π_i includes neither x_{i-1} nor x_{i+1}, but rather x'_{i-1} or x'_{i+1}, which, assuming the hiding mechanism is secure, do not reveal the original statements. We formalize this construction in Construction 1, but first formally define hiding mechanisms.

Definition 4 (NP-Statement Hider)**.** For security parameter $\lambda \in \mathbb{N}$ and NP language $\mathcal{L}_{\mathsf{pp}}$ parameterized by $\mathsf{pp} \leftarrow \mathsf{pp}(\lambda)$, with relation $\mathcal{R}_{\mathsf{pp}}$, an NP-statement hider for $\mathcal{L}_{\mathsf{pp}}$ is a pair of efficient algorithms $\mathsf{SH} = (\mathsf{Hide}, \mathsf{Check})$ such that for $(x, w) \in \mathcal{R}_{\mathsf{pp}}$ and random string $r \in \mathfrak{R}$ from randomness space $\mathfrak{R}$, SH acts as follows:

- $(x', w', c) \leftarrow \mathsf{Hide}(\mathsf{pp}, x, w, r)$ on input parameters pp, instance $(x, w) \in \mathcal{R}_{\mathsf{pp}}$ and randomness $r \in \mathfrak{R}$, Hide outputs $(x', w') \in \mathcal{R}_{\mathsf{pp}}$ and certificate c.

- $0/1 \leftarrow \mathsf{Check}(\mathsf{pp}, x, x', c)$ on input parameters pp, statements $\{x, x'\} \subseteq \mathcal{L}_{\mathsf{pp}}$, and certificate c, Check outputs 1 if the certificate shows that x' is hiding x.

Additionally, SH must satisfy the following properties.

- **Completeness:** For all adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c} (x, w) \in \mathcal{R}_{\mathsf{pp}} \wedge \\ ((x', w') \notin \mathcal{R}_{\mathsf{pp}} \vee b = 0) \end{array}\ \middle|\ \begin{array}{c} (x, w) \leftarrow \mathcal{A}(\mathsf{pp}) \\ r \leftarrow_{\$} \mathfrak{R} \\ (x', w', c) \leftarrow \mathsf{Hide}(\mathsf{pp}, x, w, r) \\ b \leftarrow \mathsf{Check}(\mathsf{pp}, x, x', c) \end{array}\right] \le \mathsf{negl}(\lambda).$$

- **Knowledge soundness:** There exists a PPT extractor Ext, such that for all PPT adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c} x \notin \mathcal{L}_{\mathsf{pp}} \vee (x', w') \notin \mathcal{R}_{\mathsf{pp}} \\ \vee\ b \neq 1 \vee (x, w) \in \mathcal{R}_{\mathsf{pp}} \end{array}\ \middle|\ \begin{array}{c} (x, x', w', c) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b \leftarrow \mathsf{Check}(\mathsf{pp}, x, x', c) \\ w \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{pp}) \end{array}\right] \ge 1 - \mathsf{negl}(\lambda).$$

- **Hiding:** For all adversaries $\mathcal{A}$ consisting of a pair of algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ is PPT,

$$\Pr\left[\begin{array}{c|c} \{(x_0, w_0), (x_1, w_1)\} \subseteq \mathcal{R}_{\mathsf{pp}} & (x_0, w_0, x_1, w_1, s) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ \wedge\ b' = b & b \leftarrow_\$ \{0,1\}, r \leftarrow_\$ \mathfrak{R} \\ & (x', w', c) \leftarrow \mathsf{Hide}(\mathsf{pp}, x_b, w_b, r) \\ & b' \leftarrow \mathcal{A}_2(x', w', s) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

3.2 Privacy preserving folding scheme from an NP-statement hider

In Construction 1, we construct a folding scheme with privacy preserving selective verification, using an NP-statement hider and a folding scheme with selective verification as building blocks. We show that the construction is secure in Theorem 2. At a high level, this construction does exactly what we previously described: each statement is hidden, then the statements hiding the original statements are folded together, and, finally, all proofs are updated to include some additional information, allowing checking that the statements hiding the original statements do indeed hide the statements they are claimed to be hiding. Similarly, the verification algorithms both check that the hiding(s) are as claimed, and that the folding is correct.

Construction 1 ($\mathsf{PrivateFS}$)**.** Let $\mathsf{SH} = (\mathsf{SH.Hide}, \mathsf{SH.Check})$ be an NP-statement hider and $\mathsf{FS} = (\mathsf{FS.Fold}, \mathsf{FS.FoldVerify}, \mathsf{FS.SlctProve}, \mathsf{FS.SlctVerify})$ be a folding scheme with selective verification. Then $\mathsf{PrivateFS} = (\mathsf{Fold}, \mathsf{FoldVerify}, \mathsf{SlctProve}, \mathsf{SlctVerify})$, constructed as follows, is a folding scheme with privacy preserving selective verification.

- $\mathsf{Fold}(\mathsf{pp}, (x_1, w_1), \ldots, (x_n, w_n))$:
  1. Generate randomness $r_1, \ldots, r_n \in \mathfrak{R}$.
  2. For $1 \leq i \leq n$: $(x'_i, w'_i, c_i) \leftarrow \mathsf{SH.Hide}(\mathsf{pp}, x_i, w_i, r_i)$.
  3. $(x, w, \pi') \leftarrow \mathsf{FS.Fold}(\mathsf{pp}, (x'_1, w'_i), \ldots (x'_n, w'_n))$.
  4. Output $(x, w, \pi = (\pi', (c_1, x'_1), \ldots, (c_n, x'_n)))$.
- $\mathsf{FoldVerify}(\mathsf{pp}, x_1, \ldots, x_n, x, \pi)$:
  1. Parse π as $(\pi', (c_1, x'_1), \ldots, (c_n, x'_n))$.
  2. For $1 \leq i \leq n$: if $\mathsf{SH.Check}(\mathsf{pp}, x_i, x'_i, c_i) = 0$, output 0 and abort.
  3. If $\mathsf{FS.FoldVerify}(\mathsf{pp}, x'_1, \ldots, x'_n, x, \pi') = 0$, output 0 and abort.
  4. Output 1.
- $\mathsf{SlctProve}(\mathsf{pp}, x_1, \ldots, x_n, x, \pi)$:
  1. Parse π as $(\pi', (c_1, x'_1), \ldots, (c_n, x'_n))$.
  2. $(\pi'_1, \ldots, \pi'_n) \leftarrow \mathsf{FS.SlctProve}(\mathsf{pp}, x'_1, \ldots, x'_n, x, \pi')$.
  3. Output $(\pi_i = (\pi'_i, c_i, x'_i))_{1 \leq i \leq n}$.
- $\mathsf{SlctVerify}(\mathsf{pp}, x, i, x_i, \pi_i)$:
  1. Parse π_i as (π'_i, c_i, x'_i).
  2. If $\mathsf{SH.Check}(\mathsf{pp}, x_i, x'_i, c_i) = 0$, output 0 and abort.
  3. If $\mathsf{FS.SlctVerify}(\mathsf{pp}, x, i, x'_i, \pi'_i) = 0$, output 0 and abort.
  4. Output 1.

Remark 2. It is immediate that the modifications in Construction 1 do not affect the asymptotic efficiency of the underlying folding scheme. In particular, the size of each proof π_i only grows by a constant amount in n.

Remark 3. Note that if a folding scheme with privacy preserving selective verification is used in a situation where it is frequent that a verifier will have to verify more than one proof of folding, generating the randomness r_1 to r_n with a seed-tree [BKP20] can result in less communication to each verifier. If a verifier has to verify the folding proofs of multiple consecutive statements, the randomness included in each of the folding proofs can often be replaced with fewer seeds from levels closer to the root of the seed-tree.

Theorem 2. *If* SH *is an NP-statement hider satisfying Definition 4 and* FS *is a folding scheme with selective verification satisfying Definition 2, then* $\mathsf{PrivateFS}$ *from Construction 1 is a folding scheme with privacy preserving selective verification, in the sense of Definition 3.*

Proof. We must show that

$$\mathsf{PrivateFS} = (\mathsf{Fold}, \mathsf{FoldVerify}, \mathsf{SlctProve}, \mathsf{SlctVerify}),$$

as constructed in Construction 1, has the properties described in Definitions 1 to 3. For brevity, we show selective completeness, selective knowledge soundness, and privacy preserving, i.e., the properties explicitly outlined in Definitions 2 and 3. Completeness and knowledge soundness (Definition 1) follows from very similar arguments.[3]

Selective completeness follows from showing that an adversary against $\mathsf{PrivateFS}$'s selective completeness implies an adversary against either the selective completeness of FS or the completeness of SH. Recall from Definition 2 that an adversary $\mathcal{A}$ against $\mathsf{PrivateFS}$'s selective completeness chooses a valid input $\{(x_i, w_i)\}_{1 \le i \le n}$ to $\mathsf{PrivateFS.Fold}$ and an index j, trying to make $b = 0$, where b is given by

$$\begin{aligned}(x, w, \pi) &\leftarrow \mathsf{PrivateFS.Fold}(\mathsf{pp}, y_1, \dots, y_n)\\ (\pi_1, \dots, \pi_n) &\leftarrow \mathsf{PrivateFS.SlctProve}(\mathsf{pp}, x_1, \dots, x_n, x, \pi)\\ b &\leftarrow \mathsf{PrivateFS.SlctVerify}(\mathsf{pp}, x, j, x_j, \pi_j).\end{aligned}$$

From Construction 1, it is clear that if $b = 0$, then either $\mathsf{SH.Check}(\mathsf{pp}, x_i, x_i', c_i) = 0$ or $\mathsf{FS.SlctVerify}(\mathsf{pp}, x, i, x_i', \pi_i') = 0$.

Consider first if $\mathsf{SH.Check}(\mathsf{pp}, x_i, x_i', c_i) = 0$. Since $(x_i, w_i) \in \mathcal{R}_{\mathsf{pp}}$, the randomness $r_i \in \mathfrak{R}$ was chosen at random, and x_i' and c_i generated as $(x_i', w_i', c_i) \leftarrow \mathsf{SH.Hide}(\mathsf{pp}, x_i, w_i, r_i)$, we are in exactly the situation described by SH's completeness definition (Definition 4). Thus, in this case, $\mathcal{A}$ being successful implies an adversary against SH being complete.

On the other hand, if $\mathsf{FS.SlctVerify}(\mathsf{pp}, x, i, x_i', \pi_i') = 0$, we observe that each (x_i', w_i') is in $\mathcal{R}_{\mathsf{pp}}$ (otherwise, we again have an adversary to SH being complete), and thus we are now in exactly the situation described by FS's selective completeness definition (Definition 2), where n instances (x_i', w_i') are first folded together using $\mathsf{FS.Fold}$, and selective proofs are then generated using $\mathsf{FS.SlctProve}$. Thus, if $\mathsf{FS.SlctVerify}(\mathsf{pp}, x, i, x_i', \pi_i') = 0$, we see that again $\mathcal{A}$ implies an adversary to either SH being complete or to FS being selective complete.

Selective knowledge soundness can be shown by constructing an extractor Ext for adversaries against $\mathsf{PrivateFS}$'s selective knowledge soundness, using the selective knowledge soundness extractor $\mathsf{FS.Ext}$ for FS, and the knowledge soundness extractor $\mathsf{SH.Ext}$ for SH. Essentially, we first use $\mathsf{FS.Ext}$ to extract a witness for x_i', and then $\mathsf{SH.Ext}$ to extract a witness for x_i.

Assume that an adversary $\mathcal{A}$ against the selective knowledge soundness of $\mathsf{PrivateFS}$ outputs (i, x_i, π_i, x, w), where π_i can be parsed as $\pi_i = (\pi_i', c_i, x_i')$. We construct Ext as follows.

[3] The main difference is that for selective completeness and selective knowledge soundness, we look at just one specific index of the input (and the output of hiding it). For completeness and knowledge soundness, however, we have to look at all inputs (either when finding an adversary to FS or SH, or when constructing an extractor against $\mathsf{PrivateFS}$). Since the number of inputs is polynomial in n, the constructed adversaries and extractor are still polynomial time.

1. To extract w'_i such that $(x'_i, w'_i) \in \mathcal{R}_{\mathsf{pp}}$, create an adversary $\mathcal{A}_{\mathsf{FS}}$, which itself queries $\mathcal{A}$, but then outputs (i, x'_i, π'_i, x, w). Note that if $\mathcal{A}$ is successful against $\mathsf{PrivateFS}$, then $\mathcal{A}_{\mathsf{FS}}$ is successful against FS, since $\mathsf{PrivateFS.SlctVerify}$ invokes $\mathsf{FS.SlctVerify}$.

2. Ext invokes $\mathsf{FS.Ext}$ with access to $\mathcal{A}_{\mathsf{FS}}$, obtaining w'_i.

3. To extract w_i such that $(x_i, w_i) \in \mathcal{R}_{\mathsf{pp}}$, create an adversary $\mathcal{A}_{\mathsf{SH}}$ which queries $\mathcal{A}$, extracts w'_i using $\mathsf{FS.Ext}$, and then outputs (x_i, x'_i, w'_i, c_i). Similarly to $\mathcal{A}_{\mathsf{FS}}$, we see that if $\mathcal{A}$ is successful against $\mathsf{PrivateFS}$, then $\mathcal{A}_{\mathsf{SH}}$ is successful against SH.

4. Ext invokes $\mathsf{SH.Ext}$ with access to $\mathcal{A}_{\mathsf{SH}}$, obtaining w_i, which Ext then outputs.

Since $\mathcal{A}_{\mathsf{FS}}$ and $\mathcal{A}_{\mathsf{SH}}$ are successful if $\mathcal{A}$ is successful, both $\mathsf{FS.Ext}$ and $\mathsf{SH.Ext}$ are successful with overwhelming probability if $\mathcal{A}$ is successful, and, hence, so is Ext.

Privacy preservation follows from showing that an adversary against $\mathsf{PrivateFS}$'s privacy preserving property implies an adversary against SH being hiding, similarly to how selective completeness was shown. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against $\mathsf{PrivateFS}$'s privacy preserving property (Definition 3). We now construct an adversary $\mathsf{SH}.\mathcal{A} = (\mathsf{SH}.\mathcal{A}_1, \mathsf{SH}.\mathcal{A}_2)$ against SH being hiding as follows.

- $\mathsf{SH}.\mathcal{A}_1$: Run $\mathcal{A}_1$ to get $(\ell, j, \{(x_i, w_i)\}_{\substack{1\le i\le n\\ i\ne j}}, (x_j^0, w_j^0), (x_j^1, w_j^1), s_{\mathsf{PrivateFS}})$. Output $(x_j^0, w_j^0, x_j^1, w_j^1, s)$, where $s = (\ell, j, \{(x_i, w_i)\}_{\substack{1\le i\le n\\ i\ne j}}, s_{\mathsf{PrivateFS}})$ is the state passed on to $\mathsf{SH}.\mathcal{A}_2$.

- $\mathsf{SH}.\mathcal{A}_2$: On input (x', w', s), hiding either (x_j^0, w_j^0) or (x_j^1, w_j^1), essentially emulate $\mathsf{PrivateFS.Fold}$ and $\mathsf{PrivateFS.SlctProve}$, to generate correct input to $\mathcal{A}_2$.

  1. For $i \neq j$, hide (x_i, w_i) using $\mathsf{SH.Hide}$ with randomness $r_i \leftarrow_\$ \mathfrak{R}$, obtaining (x'_i, w'_i, c_i). Then, fold all hidden statements using $\mathsf{FS.Fold}$ with (x', w') in the j'th spot;
  $$(x, w, \pi') \leftarrow \mathsf{FS.Fold}(\mathsf{pp}, (x'_1, w'_1), \ldots, (x', w'), \ldots, (x'_n, w'_n)).$$
  2. Next, run $\mathsf{FS.SlctProve}(\mathsf{pp}, x'_1, \ldots, x'_{j-1}, x', x'_{j+1}, \ldots, x'_n, x, \pi')$ to get $(\pi'_1, \ldots, \pi'_n)$. Generate π_ℓ by joining π'_ℓ and (c_ℓ, x'_ℓ). Since $\ell \neq j$, these are known.
  3. Run $\mathcal{A}_2(\mathsf{pp}, x, \ell, x_\ell, \pi_\ell, s_{\mathsf{PrivateFS}})$ to obtain b'. Output b'.

Observe that the input to $\mathcal{A}_2$ is *exactly* the same if $\mathcal{A}_2$ is running directly on $\mathsf{PrivateFS}$, since c_j is not part of the input, and is not used for generating any of the input, besides $(x'_j, w'_j) = (x', w')$, which is still generated using randomness sampled from $\mathfrak{R}$. Thus, $(\mathsf{SH}.\mathcal{A}_1, \mathsf{SH}.\mathcal{A}_2)$ is successful exactly when $(\mathcal{A}_1, \mathcal{A}_2)$ is successful, and hence it follows from the assumption that SH is hiding that $\mathsf{PrivateFS}$ is privacy preserving. □

3.3 NP-statement hider from a folding scheme

At this point, we have a folding scheme with selective verification from [RZ23], and we know that a folding scheme with selective verification together with an NP-statement hider is enough to give us a folding scheme with privacy preserving selective verification. Thus, the next question we ask is how to construct an NP-statement hider? One straightforward approach, is to hide an instance (x, w) by folding it with another instance $(x_\$, w_\$)$, using the folding scheme for the language, producing a new instance (x', w'), which is used as the output of the statement hider. The certificate c will then be the folding proof of folding (x, w) and $(x_\$, w_\$)$, together with either $x_\$$, or the seed used to generate $(x_\$, w_\$)$.Then, checking that x' is hiding x is just verifying that x was folded into x'.

To show that an NP-statement hider for a language, $\mathcal{L}$, with relation, $\mathcal{R}$, constructed in this fashion is secure, it is roughly sufficient that two properties hold: Let $\mathcal{R}' \subset \mathcal{R}$. We require that for any two instances $(x_0, w_0), (x_1, w_1) \in \mathcal{R}$ and any $(x_\$, w_\$) \in \mathcal{R}'$, there exists $(x'_\$, w'_\$) \in \mathcal{R}'$ such that

$$\mathsf{Fold}((x_0, w_0), (x_\$, w_\$)) = (x, w) = \mathsf{Fold}((x_1, w_1), (x'_\$, w'_\$)), \tag{1}$$

where we abuse notion and ignore the folding proof. In addition, we require that efficient sampling random instances from $\mathcal{R}'$ is possible, and we select $(x_\$, w_\$)$ randomly from $\mathcal{R}'$. If these properties hold, it is straightforward to see that since $(x_\$, w_\$)$ is sampled from $\mathcal{R}'$, it is equally likely that $(x'_\$, w'_\$)$ is sampled. Thus, (x, w) is just as likely to hide (x_1, w_1) as (x_0, w_0), and hence no adversary can do better than random guessing, showing hiding. Completeness and soundness follow directly from the underlying folding scheme. We present the construction of an NP-statement hider from a folding scheme in Construction 2, and show that it is secure in Theorem 3.

Construction 2 (NP-statement hider from folding)**.** Let $\mathsf{FS} = (\mathsf{Fold}, \mathsf{FoldVerify})$ be a folding scheme for a language $\mathcal{L}$ with relation $\mathcal{R}$, and $\mathcal{R}' \subseteq \mathcal{R}$ a subset used as the random space $\mathfrak{R}$. That is, Hide takes a random instance $(x_\$, w_\$) \in \mathcal{R}'$ as its randomness input. Then, $\mathsf{SH} = (\mathsf{Hide}, \mathsf{Check})$, constructed as follows, is an NP-statement hider.

- $\mathsf{Hide}(\mathsf{pp}, x, w, (x_\$, w_\$))$
  1. Fold (x, w) and $(x_\$, w_\$)$ together:
     $(x', w', \pi) \leftarrow \mathsf{FS.Fold}(\mathsf{pp}, (x, w), (x_\$, w_\$))$
  2. Output (x', w', c) where $c = (x_\$, \pi)$.
- $\mathsf{Check}(\mathsf{pp}, x, x', c)$
  1. Parse c as $(x_\$, \pi)$.
  2. Output the result of $\mathsf{FS.FoldVerify}(\mathsf{pp}, x, x_\$, x', \pi)$.

Theorem 3. *If* FS *satisfies Definition 1,* $\mathcal{R}'$ *allows efficient sampling, and for any two instances* $(x_0, w_0), (x_1, w_1) \in \mathcal{R}$ *and any* $(x_\$, w_\$) \in \mathcal{R}'$*, there exists* $(x'_\$, w'_\$) \in \mathcal{R}'$ *such that the outputs of* $\mathsf{Fold}((x_0, w_0), (x_\$, w_\$))$ *and* $\mathsf{Fold}((x_1, w_1), (x'_\$, w'_\$))$ *agree everywhere, except on the folding proofs. Then, the cryptographic scheme defined in Construction 2 is an NP-statement hider satisfying Definition 4.*

Proof. Completeness and knowledge soundness follow from FS satisfying Definition 1 for 2 statements. We give outlines for how they are argued.

For completeness, an adversary $\mathsf{FS}.\mathcal{A}$ against FS being complete, in the sense of Definition 1, can be constructed from an adversary $\mathcal{A}$ against Construction 2 being complete in the sense of Definition 4. $\mathsf{FS}.\mathcal{A}$ invokes $\mathcal{A}$ to get (x, w) and samples a random instance $(x_\$, w_\$) \in \mathfrak{R}$. It then outputs $((x, w), (x_\$, w_\$))$. By inspecting Hide and Check, it can be confirmed that from this point on, everything is computed the same way in FS's completeness definition and in Construction 2's completeness definition, and $\mathsf{FS}.\mathcal{A}$ is successful if $\mathcal{A}$ is successful.

For knowledge soundness, the extractor $\mathsf{FS.Ext}$ implied by FS having knowledge soundness can be used to construct an extractor Ext for Construction 2's knowledge soundness. To do this, we create an adversary $\mathsf{FS}.\mathcal{A}$ that $\mathsf{FS.Ext}$ queries. $\mathsf{FS}.\mathcal{A}$ runs $\mathcal{A}$ to get (x, x', w', c) and uses the information in c to derive $x_\$$, which was used to hide x. Finally, $\mathsf{FS}.\mathcal{A}$ outputs $(x, x_\$, x', w', \pi)$. Now, Ext runs $\mathsf{FS.Ext}^{\mathsf{FS}.\mathcal{A}}$, to obtain and output w.

For hiding, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any adversary. Consider the (x_0, w_0) and (x_1, w_1) output from $\mathcal{A}_1$. For $b \in \{0, 1\}$ and $(x_\$, w_\$) \in \mathfrak{R}$, both sampled at random, $\mathcal{A}_2$ receives (x', w') where

$$(x', w', c) \leftarrow \mathsf{Hide}(\mathsf{pp}, x_b, w_b, (x_\$, w_\$)) = \mathsf{FS.Fold}(\mathsf{pp}, (x, w), (x_\$, w_\$)).$$

The requirements of the theorem imply that there is $(x'_\$, w'_\$) \in \mathcal{R}' = \mathfrak{R}$ such that

$$\mathsf{Fold}((x_b, w_b), (x_\$, w_\$)) = \mathsf{Fold}((x_{1-b}, w_{1-b}), (x'_\$, w'_\$)),$$

and therefore, we could obtain the same (x', w') from (x_{1-b}, w_{1-b}). Thus, the only difference in the output of hiding (x_b, w_b) with $(x_\$, w_\$)$ and hiding (x_{1-b}, w_{1-b}) with $(x'_\$, w'_\$)$ is the certificate.

Since $\mathcal{A}_2$ does not receive c and $(x_\$, w_\$)$ is chosen randomly, $\mathcal{A}_2$ cannot distinguish between the two possible inputs. Hence, no adversary can do better than random guessing, showing hiding. □

With Theorems 2 and 3 and the results from Section 2, we note that a folding scheme with privacy preserving selective verification, is implied by the construction of a 2-folding scheme, and showing the additional few properties outlined in Theorem 3.

Remark 4. A folding scheme with privacy preserving selective verification can be obtained from the following 1-step transformation. Given a folding scheme with input N language instances, simply change the scheme into a scheme with input of length $2N$, where every second language instance is randomly sampled. Since this construction is essentially identical to Construction 1 with Construction 2 as the NP-statement hider, this construction is clearly also privacy preserving.

While the authors have not found any folding scheme where Construction 2 cannot be applied, we observe that in all examples we have investigated (see Section 4), we use the entire relation $\mathcal{R}$ for sampling random instances in order for Equation (1) to hold. Therefore, it is worth noting that it is an open question, if all languages in NP allow efficient sampling of random instances from the entire language. If they do, it implies that EXPTIME = NEXPTIME, which is not expected [SF90].

4 Examples

In this section, we consider concrete examples of NP-languages which already have folding schemes, and show that they also allow privacy preserving selective verification. Namely, we consider a folding scheme for Inner Product Relation of Committed Values from [RZ23] and the original folding scheme for Committed Relaxed R1CS from [KST22]. In [RZ23], they additionally construct folding schemes for Polynomial Commitment Openings [KZG10, BCL+21] and for Algebraic Vector Commitment Openings [CF13]. We do not consider either of these folding schemes, but note that the folding scheme for Algebraic Vector Commitment Openings is a reduction of an instance of Algebraic Vector Commitment Openings to an instance of Inner Product Relation of Committed Values, and hence our work also implies a folding scheme with privacy preserving selective verification for Algebraic Vector Commitment Openings.

Recently, folding schemes for other NP-complete languages have been proposed. Noticeably amongst them, [BC24] introduces LatticeFold, the first folding scheme that does not use a additively homomorphic commitment scheme based on the discrete logarithm problem, but rather Ajtai commitments which are based on the module SIS problem. Thus, LatticeFold is the first post-quantum secure folding scheme. LatticeFold is a general purpose scheme, and it supports folding of both low degree relations and high degree relations. In particular, it supports both R1CS and CCS [STW23].

4.1 Inner Product Relation of Committed Values

As a first example, we consider Inner Product Relation of Committed Values [BCC+16, RZ23], which are used in Bulletproofs [BBB+18].

For this example, we need Pedersen commitments for multiple values. We give details on Pedersen commitments in Appendix A.1, but the essentials are as follows. Using our group notation introduced in Section 1.4, a commitment key for a Pedersen commitment for multiple values is $[\mathbf{r}] \in \mathbb{G}^n$, and a commitment to $\mathbf{m} \in \mathbb{F}^n$ is $[c] \leftarrow [\mathbf{r}]^\top \mathbf{m}$. For $n \geq 2$, this commitment is information-theoretically hiding and computationally binding under the discrete logarithm assumption.

The language family of Inner Product Relation of Committed Values is parameterized by a group description gk and two Pedersen commitment keys $[\mathbf{r}], [\mathbf{s}] \in \mathbb{G}^n$. The language is then defined as

$$\mathcal{L}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]} = \left\{([c],[d],z) \mid \exists \mathbf{a}, \mathbf{b} \in \mathbb{F}^n \text{ s.t } [c] = [\mathbf{r}]^\top \mathbf{a}, [d] = [\mathbf{s}]^\top \mathbf{b}, z = \mathbf{a}^\top \mathbf{b}\right\},$$

and the corresponding relation as $\mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$. For $i \in \{1,2\}$ let $y_i = (([c_i],[d_i],z_i),(\mathbf{a}_i,\mathbf{b}_i))$, (supposedly) in $\mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$ with the witness being $(\mathbf{a}_i, \mathbf{b}_i)$. We now describe a public coin protocol for folding y_1 and y_2.

1. The prover sends $z_{1,2} = \mathbf{a}_1^\top \mathbf{b}_2$ and $z_{2,1} = \mathbf{a}_2^\top \mathbf{b}_1$.
2. The verifier sends $\rho \leftarrow_\$ \mathbb{F}$.
3. The prover and verifier each construct a new statement $([c],[d],z)$ as
$$\begin{aligned}[c] &= [c_1] + \rho[c_2]\\ [d] &= [d_1] + \rho^2[d_2]\\ z &= z_1 + \rho z_{2,1} + \rho^2 z_{1,2} + \rho^3 z_2,\end{aligned}$$
and the prover also constructs a new witness $(\mathbf{a}, \mathbf{b})$ as $\mathbf{a} = \mathbf{a}_1 + \rho\mathbf{a}_2, \mathbf{b} = \mathbf{b}_1 + \rho^2\mathbf{b}_2$.

Completeness of the protocol follows from straightforward calculation; if $y_1, y_2 \in \mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$, then $(([c],[d],z),(\mathbf{a},\mathbf{b})) \in \mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$. Knowledge soundness is a little more tricky, but, as outlined in [RZ23], it essentially follows from noticing that a prover who is able to open commitments of the form $[\alpha_1] + \rho[\alpha_2]$ should know openings to $[\alpha_1]$ and $[\alpha_2]$, since they are defined before the challenge ρ is given. Additionally, since $z_{1,2}$ and $z_{2,1}$ are defined before ρ is given, one could treat the relation that $\mathbf{a}^\top\mathbf{b} = z$ as a polynomial in ρ, that is, $\mathbf{a}(\rho)^\top\mathbf{b}(\rho) = z(\rho)$, and it can be shown that this implies both $\mathbf{a}_1^\top\mathbf{b}_1 = z_1$ and $\mathbf{a}_2^\top\mathbf{b}_2 = z_2$.

Since this protocol is public coin, the Fiat-Shamir heuristic [FS86] immediately transforms the protocol into a (non-interactive) 2-folding scheme for Inner Product Relation of Committed Values (2-IPRCV), essentially by replacing the random challenge ρ with the result of hashing the public inputs and the first message from the prover using a cryptographic hash function H. The folding proof π is simply the *cross terms* $z_{1,2}$ and $z_{2,1}$. For completeness, we write out 2-IPRCV as Construction 3 in Appendix A.2.

The following corollary follows from the bootstrap construction in Section 2.1 and the extension discussed in Remark 1. Theorems and proofs corresponding to the construction and discussion can be found as Theorems 1 and 2 in [RZ23].

Corollary 1 (IPRCV)**.** *2-IPRCV being a 2-folding scheme for Inner Product Relation of Committed Values implies the existence of a N-folding scheme IPRCV with selective verification for Inner Product Relation of Committed Values.*

In order to get a folding scheme that is also privacy preserving, we apply Construction 2 to get an NP-statement hider by folding with a randomly sampled instance from the entire space, i.e., $\mathcal{R}' = \mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$. Denote this instantiation of Construction 2 by $\mathsf{IPRCV\text{-}SH}$. For completeness, we write out $\mathsf{IPRCV\text{-}SH}$ as Construction 4 in Appendix A.3. The proof that this construction satisfies Theorem 3 is presented in Theorem 4.

Theorem 4. *If* 2-IPRCV is a 2-folding scheme for Inner Product Relation of Committed Values, then $\mathsf{IPRCV\text{-}SH}$ is an NP-statement hider for Inner Product Relation of Committed Values, in the sense of Definition 4.

Proof. It is sufficient to show that IPRCV-SH satisfies Theorem 3. By assumption 2-IPRCV satisfies Definition 1.

To see that $\mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$ supports efficient sampling, observe that we can sample a random instance in $\mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$ by sampling two vectors in $\mathbb{F}^n$, and constructing the rest of the random instance from these, as follows:

$$\begin{aligned} &\mathbf{a}_\$, \mathbf{b}_\$ \leftarrow_\$ \mathbb{F}^n && [d_\$] = [\mathbf{s}]^\top \mathbf{b}_\$ \\ &[c_\$] = [\mathbf{r}]^\top \mathbf{a}_\$ && z_\$ = \mathbf{a}_\$^\top \mathbf{b}_\$. \end{aligned} \tag{2}$$

Finally, we need to show for any three instances $(x_0, w_0), (x_1, w_1)$, and $(x_\$, x_\$)$, there exists an instance $(x'_\$, w'_\$)$ such that:

$$\text{2-IPRCV.Fold}(\mathsf{pp}, (x_0, w_0), (x_\$, w_\$)) = \text{2-IPRCV.Fold}(\mathsf{pp}, (x_1, w_1), (x'_\$, w'_\$)), \tag{3}$$

where we abuse notation by ignoring the folding proof. Denoting the output from folding (x_0, w_0) and $(x_\$, w_\$)$ by (x', w') and letting $\alpha \in \{0, 1, \$\}$, we use the following notation for the instances in consideration.

$$\begin{aligned} (x_\alpha, w_\alpha) &= (([c_\alpha], [d_\alpha], z_\alpha), (\mathbf{a}_\alpha, \mathbf{b}_\alpha)) \\ (x'_\$, w'_\$) &= (([c'_\$], [d'_\$], z'_\$), (\mathbf{a}'_\$, \mathbf{b}'_\$)). \\ (x', w') &= (([c], [d], z), (\mathbf{a}, \mathbf{b})). \end{aligned}$$

We prove the existence of $(x'_\$, w'_\$)$ in the random oracle model, where we replace the hash function H, used to find ρ, with a random oracle. By doing so, we can both use the same random value ρ in both folds, and assume it does not depend on the instances folded. Since the adversary will learn neither $x_\$$ nor $x'_\$$, one of which is part of the input to H, it is justified to model H as a random oracle.

When (x', w') is obtained by folding (x_0, w_0) and $(x_\$, w_\$)$, it has the following form:

$$\mathbf{a} = \mathbf{a}_0 + \rho \mathbf{a}_\$ \tag{4}$$
$$\mathbf{b} = \mathbf{b}_0 + \rho^2 \mathbf{b}_\$ \tag{5}$$
$$[c] = [c_0] + \rho[c_\$] = [c_0] + \rho[\mathbf{r}]^\top \mathbf{a}_\$ \tag{6}$$
$$[d] = [d_0] + \rho^2[d_\$] = [d_0] + \rho^2[\mathbf{s}]^\top \mathbf{b}_2 \tag{7}$$
$$z = z_0 + \rho z_{\$,0} + \rho^2 z_{0,\$} + \rho^3 z_\$ = z_0 + \rho \mathbf{a}_\$^\top \mathbf{b}_0 + \rho^2 \mathbf{a}_0^\top \mathbf{b}_\$ + \rho^3 \mathbf{a}_\$^\top \mathbf{b}_\$. \tag{8}$$

Where in Equations (6) to (8), we substitute in how $x_\$ = ([c_\$], [d_\$], z_\$)$ is derived from $w_\$ = (\mathbf{a}_\$, \mathbf{b}_\$)$, i.e., Equation (2), and the definition of the cross terms.

Since the goal is to have (x_1, w_1) and $(x'_\$, w'_\$)$ fold to $(([c], [d], z), (\mathbf{a}, \mathbf{b}))$, it follows from Equations (4) and (5) that we need

$$\begin{aligned} \mathbf{a}_1 + \rho \mathbf{a}'_\$ &= \mathbf{a} = \mathbf{a}_0 + \rho \mathbf{a}_\$ \\ \mathbf{b}_1 + \rho^2 \mathbf{b}'_\$ &= \mathbf{b} = \mathbf{b}_0 + \rho^2 \mathbf{b}_\$, \end{aligned}$$

and therefore we fix $\mathbf{a}'_\$$ and $\mathbf{b}'_\$$ such that $\rho \mathbf{a}'_\$ = \mathbf{a}_0 + \rho \mathbf{a}_\$ - \mathbf{a}_1$ and $\rho^2 \mathbf{b}'_\$ = \mathbf{b}_0 + \rho^2 \mathbf{b}_\$ - \mathbf{b}_1$. Deriving $x'_\$$ from the now fixed $w'_\$ = (\mathbf{a}'_\$, \mathbf{b}'_\$)$ by the same calculations as in Equation (2), we obtain the instance $(x'_\$, w'_\$) \in \mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$.

It now suffices to verify that when (x_1, w_1) is folded with $(\mathrm{x}'_\$, w'_\$)$ using randomness ρ, we get (x', w'). We first verify $[c]$:

$$\begin{aligned} [c_1] + \rho[c'_\$] &= [c_1] + \rho[\mathbf{r}]^\top \mathbf{a}'_\$ = [c_1] + [\mathbf{r}]^\top (\mathbf{a}_0 + \rho \mathbf{a}_\$ - \mathbf{a}_1) \\ &= [c_1] + [c_0] + \rho[c_\$] - [c_1] = [c_0] + \rho[c_\$] = [c]. \end{aligned}$$

A very similar calculation can be done to verify $[d]$, see Appendix A.4. Verifying z is done by showing that

$$z = z_1 + \rho z_{\$,1} + \rho^2 z_{1,\$} + \rho^3 z'_{\$}. \tag{9}$$

As a first step, we expand the four terms in Equation (9) separately.

$$\begin{aligned}
z_1 &= \mathbf{a}_1^\top \mathbf{b}_1 && (10)\\
\rho z_{\$,1} &= \rho(\mathbf{a}'_{\$})^\top \mathbf{b}_1 = (\mathbf{a}_0 + \rho\mathbf{a}_{\$} - \mathbf{a}_1)^\top \mathbf{b}_1 = \mathbf{a}_0^\top \mathbf{b}_1 + \rho\mathbf{a}_{\$}^\top \mathbf{b}_1 - \mathbf{a}_1^\top \mathbf{b}_1 && (11)\\
\rho^2 z_{1,\$} &= \mathbf{a}_1^\top(\rho^2\mathbf{b}'_{\$}) = \mathbf{a}_1^\top(\mathbf{b}_0 + \rho^2\mathbf{b}_{\$} - \mathbf{b}_1) = \mathbf{a}_1^\top\mathbf{b}_0 + \rho^2\mathbf{a}_1^\top\mathbf{b}_{\$} - \mathbf{a}_1^\top\mathbf{b}_1 && (12)\\
\rho^3 z'_{\$} &= (\rho\mathbf{a}'_{\$})^\top \rho^2\mathbf{b}'_{\$} = (\mathbf{a}_0 + \rho\mathbf{a}_{\$} - \mathbf{a}_1)^\top(\mathbf{b}_0 + \rho^2\mathbf{b}_{\$} - \mathbf{b}_1)\\
&= \mathbf{a}_0^\top(\mathbf{b}_0 + \rho^2\mathbf{b}_{\$} - \mathbf{b}_1) + \mathbf{a}_{\$}^\top(\rho\mathbf{b}_0 + \rho^3\mathbf{b}_{\$} - \rho\mathbf{b}_1) - \mathbf{a}_1^\top(\mathbf{b}_0 + \rho^2\mathbf{b}_{\$} - \mathbf{b}_1). && (13)
\end{aligned}$$

Inserting Equations (10) to (13) into Equation (9) gives

$$\begin{aligned}
(9) = \mathbf{a}_1^\top&(\mathbf{b}_1 - \mathbf{b}_1 + \mathbf{b}_0 + \rho^2\mathbf{b}_{\$} - \mathbf{b}_1 - \mathbf{b}_0 - \rho^2\mathbf{b}_{\$} + \mathbf{b}_1)\\
&+ \mathbf{a}_0^\top(\mathbf{b}_1 + \mathbf{b}_0 + \rho^2\mathbf{b}_{\$} - \mathbf{b}_1) + \mathbf{a}_{\$}^\top(\rho\mathbf{b}_1 + \rho\mathbf{b}_0 + \rho^3\mathbf{b}_{\$} - \rho\mathbf{b}_1),
\end{aligned} \tag{14}$$

and cancelling gives Equation (8). Calculations are in Appendix A.4. We have now shown that there is a $(x'_{\$}, w'_{\$})$ such that Equation (3) holds in the random oracle model. □

Corollary 2. *There is a folding scheme with privacy preserving selective verification for Inner Product Relation of Committed Values.*

4.2 Committed Relaxed R1CS

Committed Relaxed R1CS is the language used in the original paper introducing folding schemes [KST22]. The language is a folding amenable generalization of Rank One Constraint Systems (R1CS) [SBV+13, GGPR13], and a classical language used for many proof systems [Gro16, GWC19, BCR+19, KS22]. R1CS is a satisfiability flavored characterization of the complexity class NP. Roughly, R1CS works as follow. For the three parameters, $m \times m$ matrices $A, B, C \in \mathbb{F}^{m\times m}$, an instance of R1CS is $\mathbf{x} \in \mathbb{F}^n$ with $n < m$ for which there is a witness $\mathbf{w} \in \mathbb{F}^{m-n-1}$ such that with $\mathbf{z} = (\mathbf{w}, \mathbf{x}, 1)^\top$ we have $A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z}$, where $\circ$ denotes entry wise multiplication, also called the Hadamard product.

To make R1CS amenable to folding, the structure is modified to have $u \in \mathbb{F}$, rather than 1, as the last entry in $\mathbf{z}$, and as a scalar in front of $C\mathbf{z}$. Additionally, an error term $\mathbf{e}$ is introduced. To keep the protocol zero-knowledge, commitments for $\mathbf{w}$ and $\mathbf{e}$ are introduced, using an additively homomorphic commitment scheme, for example Pedersen commitments. For notation, we write $\overline{x} \leftarrow \mathsf{Com}(x, r_x)$, meaning that $\overline{x}$ is a commitment to x using randomness r_x. With this notation, the language Committed Relaxed R1CS is

$$\mathcal{L}_{A,B,C} = \left\{ (u, \mathbf{x}, \overline{\mathbf{e}}, \overline{\mathbf{w}}) \middle| \exists(\mathbf{e}, r_{\mathbf{e}}, \mathbf{w}, r_{\mathbf{w}}) \colon \begin{array}{c} \mathbf{z} := (\mathbf{w}, \mathbf{x}, u);\ A\mathbf{z} \circ B\mathbf{z} = uC\mathbf{z} + \mathbf{e}; \\ \overline{\mathbf{e}} \leftarrow \mathsf{Com}(\mathbf{e}, r_{\mathbf{e}});\ \overline{\mathbf{w}} \leftarrow \mathsf{Com}(\mathbf{w}, r_{\mathbf{w}}) \end{array} \right\},$$

with a corresponding relation $\mathcal{R}_{A,B,C}$. Note that since R1CS is NP-complete and contained in Committed Relaxed R1CS, Committed Relaxed R1CS is NP-complete [KST22].

Following [KST22], a public coin protocol for folding two instances of Committed Relaxed R1CS can be constructed as follows. For $i \in \{0, 1\}$, denote the two instances as

$$y_i = ((u_i, \mathbf{x}_i, \overline{\mathbf{e}_i}, \overline{\mathbf{w}_i}), (\mathbf{e}_i, r_{\mathbf{e}_i}, \mathbf{w}_i, r_{\mathbf{w}_i})) \in \mathcal{R}_{A,B,C}.$$

1. The prover sends $\overline{t} \leftarrow \mathsf{Com}(t, r_t)$ where $r_t \leftarrow_{\$} \mathbb{F}$ and
$$t = A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1C\mathbf{z}_2 - u_2C\mathbf{z}_1. \tag{15}$$
2. The verifier sends $\rho \leftarrow_{\$} \mathbb{F}$.

3. Both the prover and verifier construct the folded instance $(u, \mathbf{x}, \overline{\mathbf{e}}, \overline{\mathbf{w}})$ where[4]

$$u = u_1 + \rho u_2 \qquad \overline{\mathbf{e}} = \overline{\mathbf{e}_1} + \rho \overline{t} + \rho^2 \overline{\mathbf{e}_2}$$
$$\mathbf{x} = \mathbf{x}_1 + \rho \mathbf{x}_2 \qquad \overline{\mathbf{w}} = \overline{\mathbf{w}_1} + \rho \overline{\mathbf{w}_2}.$$

Additionally, the prover constructs a witness $(\mathbf{e}, r_{\mathbf{e}}, \mathbf{w}, r_{\mathbf{w}})$ for the folded instance, where

$$\mathbf{e} = \mathbf{e}_1 + \rho t + \rho^2 \mathbf{e}_2 \qquad \mathbf{w} = \mathbf{w}_1 + \rho \mathbf{w}_2$$
$$r_{\mathbf{e}} = r_{\mathbf{e}_1} + \rho r_t + \rho^2 r_{\mathbf{e}_2} \qquad r_{\mathbf{w}} = r_{\mathbf{w}_1} + \rho r_{\mathbf{w}_2}.$$

This protocol can be turned into a (non-interactive) 2-folding scheme using the Fiat-Shamir heuristic, and the 2-folding scheme can be bootstrapped using the techniques mentioned in Section 2.1 to an N-folding scheme with selective verification. Denote this folding scheme CR-R1CS. The security of CR-R1CS follows from the same type of arguments as the scheme in Section 4.1, and a proof can be found in [KST22]. It follows from the proof of the bootstrap construction in [RZ23] that CR-R1CS is selective verifiable.

In order to get a folding scheme with privacy preserving selective verification for Committed Relaxed R1CS, we first instantiate Construction 2 with CR-R1CS to get a statement hider SH-CR-R1CS. Similar to the IPRCV statement hider, we use the entire relation, $\mathcal{R}_{A,B,C}$, as the sample space. We can then instantiate Construction 1 with CR-R1CS and SH-CR-R1CS to get a folding scheme with privacy preserving selective verification for Committed Relaxed R1CS. We denote this instantiation of Construction 1 as PP-CR-R1CS. We now show that PP-CR-R1CS is a folding scheme with privacy preserving selective verification in the random oracle model.

Theorem 5. *Assuming that CR-R1CS is a folding scheme with selective verification, PP-CR-R1CS, constructed as described in the previous paragraph, is a folding scheme with privacy preserving selective verification for Committed Relaxed R1CS.*

Proof. From Theorems 2 and 3, it follows that in order to show that PP-CR-R1CS satisfies Definitions 1 to 3, it is sufficient to show that CR-R1CS satisfies Definition 2 and that Theorem 3 applies. By assumption, CR-R1CS satisfies Definitions 1 and 2.

Efficient sampling from the entire relation space can be obtained as follows: First, sample random vectors $\mathbf{x} \in \mathbb{F}^n, \mathbf{w} \in \mathbb{F}^{m-n-1}$ and $u \in \mathbb{F}$. Then, with $\mathbf{z} = (\mathbf{w}, \mathbf{x}, u)^\top$, set

$$\mathbf{e} := A\mathbf{z} \circ B\mathbf{z} - uC\mathbf{z}, \tag{16}$$

and generate commitments to $\mathbf{w}$ and $\mathbf{e}$ with randomness $r_{\mathbf{w}}, r_{\mathbf{e}} \leftarrow_\$ \mathbb{F}$:

$$\overline{\mathbf{e}} \leftarrow \mathsf{Com}(\mathbf{e}, r_{\mathbf{e}}) \qquad \overline{\mathbf{w}} \leftarrow \mathsf{Com}(\mathbf{w}, r_{\mathbf{w}}).$$

The random instance is now given by $((u, \mathbf{x}, \overline{\mathbf{e}}, \overline{\mathbf{w}}), (\mathbf{e}, r_{\mathbf{e}}, w, r_{\mathbf{w}}))$. It follows from Equation (16) that the instance by definition is in $\mathcal{R}_{A,B,C}$, and since $\mathbf{z}$ is a random vector in $\mathbb{F}^m$, the instance is chosen randomly from the entire space.

The final criterion we need to show is that for any two instances, y_1 and y_1', with

$$y_1 = ((u_1, \mathbf{x}_1, \overline{\mathbf{e}_1}, \overline{\mathbf{w}_1}), (\mathbf{e}_1, r_{\mathbf{e}_1}, \mathbf{w}_1, r_{\mathbf{w}_1}))$$
$$y_1' = ((u_1', \mathbf{x}_1', \overline{\mathbf{e}_1'}, \overline{\mathbf{w}_1'}), (\mathbf{e}_1', r_{\mathbf{e}_1'}, \mathbf{w}_1', r_{\mathbf{w}_1'})),$$

and third instance $y_2 = ((u_2, \mathbf{x}_2, \overline{\mathbf{e}_2}, \overline{\mathbf{w}_2}), (\mathbf{e}_2, r_{\mathbf{e}_2}, \mathbf{w}_2, r_{\mathbf{w}_2}))$, there is an instance $y_2' = ((u_2', \mathbf{x}_2', \overline{\mathbf{e}_i'}, \overline{\mathbf{w}_2'}), (\mathbf{e}_2', r_{\mathbf{e}_2'}, \mathbf{w}_2', r_{\mathbf{w}_2'}))$, such that the statement and witness obtained by folding y_1 and y_2 is the same as the statement and witness obtained by folding y_1' and y_2'. That is, abusing notation by ignoring the proof of folding, we find y_2' such that

$$\mathsf{Fold}(y_1, y_2) = \mathsf{Fold}(y_1', y_2'). \tag{17}$$

[4]Recall that the verifier knows the values committed to by $\overline{\mathbf{e}}$ and $\overline{\mathbf{w}}$, and Pedersen commitments allow noninteractive multiplying commitments by scalars and adding commitments.

We show this in the random oracle model, allowing us to use the same randomness ρ for both folds. To satisfy Equation (17), the following equations must hold:

$$\begin{aligned}\mathbf{x}_1 + \rho\mathbf{x}_2 &= \mathbf{x}'_1 + \rho\mathbf{x}'_2\\ u_1 + \rho u_2 &= u'_1 + \rho u'_2\\ \mathbf{w}_1 + \rho\mathbf{w}_2 &= \mathbf{w}'_1 + \rho\mathbf{w}'_2\\ \mathbf{e}_1 + \rho\cdot t + \rho^2\mathbf{e}_2 &= \mathbf{e}'_1 + \rho\cdot t' + \rho^2\mathbf{e}'_2,\end{aligned}$$

where t and t' are the cross terms from Equation (15), corresponding to $\mathsf{Fold}(y_1, y_2)$ and $\mathsf{Fold}(y'_1, y'_2)$, respectively. Isolating $\mathbf{x}'_2, u'_2, \mathbf{w}'_2, \mathbf{e}'_2$ and constructing commitments to $\mathbf{w}'_2$ and $\mathbf{e}'_2$ (and their randomness) from the initial instance gives an instance

$$\begin{aligned}y'_2 &= ((u'_2, \mathbf{x}'_2, \overline{\mathbf{e}'_1}, \overline{\mathbf{w}'_2}), (\mathbf{e}'_2, r_{\mathbf{e}'_2}, \mathbf{w}'_2, r_{\mathbf{w}'_2}))\\ &= ((\rho^{-1}(u_1 + \rho u_2 - u'_1), \rho^{-1}(\mathbf{x}_1 + \rho\mathbf{x}_2 - \mathbf{x}'_1),\\ &\quad \rho^{-2}(\overline{\mathbf{e}_1} + \rho\bar{t} + \rho^2\overline{\mathbf{e}_2} - \overline{\mathbf{e}'_1} - \rho\overline{t'}), \rho^{-1}(\overline{\mathbf{w}_1} + \rho\overline{\mathbf{w}_2} - \overline{\mathbf{w}'_1})),\\ &\quad (\rho^{-2}(\mathbf{e}_1 + \rho t + \rho^2\mathbf{e}_2 - \mathbf{e}'_1 - \rho t'), \rho^{-2}(r_{\mathbf{e}_1} + \rho r_t + \rho^2 r_{\mathbf{e}_2} - r_{\mathbf{e}'_1} - \rho r_{t'}),\\ &\quad \rho^{-1}(\mathbf{w}_1 + \rho\mathbf{w}_2 - \mathbf{w}'_1), \rho^{-1}(r_{\mathbf{w}_1} + \rho r_{\mathbf{w}_2} - r_{\mathbf{w}'_1}))),\end{aligned}$$

which by construction satisfies Equation (17). We need to verify that the instance is indeed in $\mathcal{R}_{A,B,C}$. By inspection, the commitments are correct, so it suffices to verify that

$$A\mathbf{z}'_2 \circ B\mathbf{z}'_2 = u'_2 C\mathbf{z}'_2 + \mathbf{e}'_2. \tag{18}$$

Essentially, this is done by expanding each side of Equation (18), and comparing the results. For each of the following 3 expansions, Appendix B includes more steps.

By construction $\mathbf{z}'_2 = (\mathbf{w}'_2, \mathbf{x}'_2, u'_2)^\top = \rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1)$. We first expand the left side of Equation (18) using the distributive laws for entry-wise multiplication.

$$\begin{aligned}A\mathbf{z}'_2 \circ B\mathbf{z}'_2 &= A(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1)) \circ B(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1))\\ &= \rho^{-2}(A\mathbf{z}_1 \circ B\mathbf{z}_1 + \rho A\mathbf{z}_1 \circ B\mathbf{z}_2 - A\mathbf{z}_1 \circ B\mathbf{z}'_1\\ &\qquad + \rho A\mathbf{z}_2 \circ B\mathbf{z}_1 + \rho^2 A\mathbf{z}_2 \circ B\mathbf{z}_2 - \rho A\mathbf{z}_2 \circ B\mathbf{z}'_1\\ &\qquad - A\mathbf{z}'_1 \circ B\mathbf{z}_1 - \rho A\mathbf{z}'_1 \circ B\mathbf{z}_2 + A\mathbf{z}'_1 \circ B\mathbf{z}'_1)\\ &= u_2 C\mathbf{z}_2 + \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - A\mathbf{z}_2 \circ B\mathbf{z}'_1 - A\mathbf{z}'_1 \circ B\mathbf{z}_2)\\ &\quad + \rho^{-2}(u_1 C\mathbf{z}_1 + \mathbf{e}_1 - A\mathbf{z}_1 \circ B\mathbf{z}'_1 - A\mathbf{z}'_1 \circ B\mathbf{z}_1 + u'_1 C\mathbf{z}'_1 + \mathbf{e}'_1).\end{aligned} \tag{19}$$

Before we expand the right side of Equation (18), we expand the error term $\mathbf{e}'_2$. This is done by inserting the cross terms t and t', multiplying out, inserting u'_2 and $\mathbf{z}'_2$, multiplying out again, and finally applying that $A\mathbf{z}'_1 \circ B\mathbf{z}'_1 = u'_1 C\mathbf{z}'_1 + \mathbf{e}'_1$ twice. We obtain that

$$\begin{aligned}\mathbf{e}'_2 &= \rho^{-2}(\mathbf{e}_1 + \rho t + \rho^2\mathbf{e}_2 - \mathbf{e}'_1 - \rho t')\\ &= \rho^{-2}(\mathbf{e}_1 + \rho(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1) + \rho^2\mathbf{e}_2 - \mathbf{e}'_1\\ &\qquad - \rho(A\mathbf{z}'_1 \circ B(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1)) + A(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1)) \circ B\mathbf{z}'_1\\ &\qquad - u'_1 C(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1)) - \rho^{-1}(u_1 + \rho u_2 - u'_1)C\mathbf{z}'_1))\\ &= \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1\\ &\qquad - A\mathbf{z}'_1 \circ B\mathbf{z}_2 - A\mathbf{z}_2 \circ B\mathbf{z}'_1 + u'_1 C\mathbf{z}_2 + u_2 C\mathbf{z}'_1)\\ &\quad + \rho^{-2}(\mathbf{e}_1 - \mathbf{e}'_1 - A\mathbf{z}'_1 \circ B\mathbf{z}_1 + A\mathbf{z}'_1 \circ B\mathbf{z}'_1 - A\mathbf{z}_1 \circ B\mathbf{z}'_1 + A\mathbf{z}'_1 \circ B\mathbf{z}'_1\\ &\qquad + u'_1 C\mathbf{z}_1 - u'_1 C\mathbf{z}'_1 + u_1 C\mathbf{z}'_1 - u'_1 C\mathbf{z}'_1).\end{aligned}$$

$$\begin{aligned}
&= \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1 \\
&\qquad - A\mathbf{z}_1' \circ B\mathbf{z}_2 - A\mathbf{z}_2 \circ B\mathbf{z}_1' + u_1' C\mathbf{z}_2 + u_2 C\mathbf{z}_1') \\
&\quad + \rho^{-2}(\mathbf{e}_1 + \mathbf{e}_1' - A\mathbf{z}_1' \circ B\mathbf{z}_1 - A\mathbf{z}_1 \circ B\mathbf{z}_1' + u_1' C\mathbf{z}_1 + u_1 C\mathbf{z}_1').
\end{aligned} \tag{20}$$

We are now ready to expand the right side of Equation (18). At Equation (21), we insert Equation (20) in place of $\mathbf{e}_2'$, and cancel out where applicable.

$$\begin{aligned}
u_2' C\mathbf{z}_2' + \mathbf{e}_2' &= \rho^{-1}(u_1 + \rho u_2 - u_1')C(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}_1')) + \mathbf{e}_2' \\
&= u_2 C\mathbf{z}_2 + \rho^{-1}(u_1 C\mathbf{z}_2 + u_2 C\mathbf{z}_1 - u_2 C\mathbf{z}_1' - u_1' C\mathbf{z}_2) \\
&\quad + \rho^{-2}(u_1 C\mathbf{z}_1 - u_1 C\mathbf{z}_1' - u_1' C\mathbf{z}_1 + u_1' C\mathbf{z}_1') + \mathbf{e}_2' \\
&= u_2 C\mathbf{z}_2 + \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - A\mathbf{z}_1' \circ B\mathbf{z}_2 - A\mathbf{z}_2 \circ B\mathbf{z}_1') \\
&\quad + \rho^{-2}(u_1 C\mathbf{z}_1 + \mathbf{e}_1 + u_1' C\mathbf{z}_1' + \mathbf{e}_1' - A\mathbf{z}_1' \circ B\mathbf{z}_1 - A\mathbf{z}_1 \circ B\mathbf{z}_1').
\end{aligned} \tag{21}$$

It can now be verified that Equation (18) holds, simply by comparing Equation (19) and Equation (21). Thus, we have shown the existence of $y_2' \in \mathcal{R}_{A,B,C}$ such that Equation (17) holds, which was the final requirement for Theorem 3, and hence finishes the proof of Theorem 5. □

References

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy - SP 2018*, pages 315–334. IEEE Computer Society, 2018. doi:10.1109/SP.2018.00020.

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In *Advances in Cryptology - ASIACRYPT 2023*, volume 14439 of *Lecture Notes in Computer Science*, pages 77–110. Springer, 2023. doi:10.1007/978-981-99-8724-5_3.

[BC24] Dan Boneh and Binyi Chen. LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. In *The Science of Blockchain Conference - SBC '24*, 2024. URL: https://eprint.iacr.org/2024/257.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology - EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 327–357. Springer, 2016. doi:10.1007/978-3-662-49896-5_12.

[BCL+21] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In *Advances in Cryptology - CRYPTO 2021*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710. Springer, 2021. doi:10.1007/978-3-030-84242-0_24.

[BCMS20] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In *Theory of Cryptography - TCC 2020*, volume 12551 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2020. doi:10.1007/978-3-030-64378-2_1.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for

R1CS. In *Advances in Cryptology - EUROCRYPT 2019*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019. doi:10.1007/978-3-030-17653-2_4.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Advances in Cryptology - CRYPTO 2021*, volume 12825 of *Lecture Notes in Computer Science*, pages 649–680. Springer, 2021. doi:10.1007/978-3-030-84242-0_23.

[BELN23] Joan Boyar, Simon Erfurth, Kim S. Larsen, and Ruben Niederhagen. Quotable signatures for authenticating shared quotes. In *Progress in Cryptology - LATINCRYPT 2023*, volume 14168 of *Lecture Notes in Computer Science*, pages 273–292. Springer, 2023. doi:10.1007/978-3-031-44469-2_14.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. *IACR Cryptol. ePrint Arch.*, page 1021, 2019. URL: https://eprint.iacr.org/2019/1021.

[BKP20] Ward Beullens, Shuichi Katsumata, and Federico Pintore. Calamari and Falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In *Advances in Cryptology - ASIACRYPT 2020*, volume 12492 of *Lecture Notes in Computer Science*, pages 464–492. Springer, 2020. doi:10.1007/978-3-030-64834-3_16.

[C2P] C2PA. Coalition for Content Provenance and Authenticity (C2PA). https://c2pa.org/.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *Public-Key Cryptography - PKC 2013*, volume 7778 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2013. doi:10.1007/978-3-642-36362-7_5.

[CNR$^+$22] Matteo Campanelli, Anca Nitulescu, Carla Ràfols, Alexandros Zacharakis, and Arantxa Zapico. Linear-map vector commitments and their practical applications. In *Advances in Cryptology - ASIACRYPT 2022*, volume 13794 of *Lecture Notes in Computer Science*, pages 189–219. Springer, 2022. doi:10.1007/978-3-031-22972-5_7.

[DCB24] Trisha Datta, Binyi Chen, and Dan Boneh. VerITAS: Verifying image transformations at scale, 2024. URL: https://eprint.iacr.org/2024/1066.

[DEH24] Stefan Dziembowski, Shahriar Ebrahimi, and Parisa Hassanizadeh. VIMz: Verifiable image manipulation using folding-based zkSNARKs. *IACR Cryptol. ePrint Arch.*, page 1063, 2024. URL: https://eprint.iacr.org/2024/1063.

[Erf24] Simon Erfurth. Digital signatures for authenticating compressed JPEG images. In *Security-Centric Strategies for Combating Information Disorder - SCID 2024*, page 4. ACM, 2024. doi:10.1145/3660512.3665522.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986. doi:10.1007/3-540-47721-7_12.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013. doi:10.1007/978-3-642-38348-9_37.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016. doi:10.1007/978-3-662-49896-5_11.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, page 953, 2019. URL: https://eprint.iacr.org/2019/953.

[JWL11] Rob Johnson, Leif Walsh, and Michael Lamb. Homomorphic signatures for digital photographs. In *Financial Cryptography and Data Security - FC 2011*, volume 7035 of *LNCS*, pages 141–157. Springer, 2011. doi:10.1007/978-3-642-27576-0_12.

[Kot24] Abhiram Kothapalli. *A Theory of Composition for Proofs of Knowledge*. PhD thesis, Carnegie Mellon University, 2024. URL: https://www.andrew.cmu.edu/user/bparno/papers/kothapalli_thesis.pdf.

[KP23] Abhiram Kothapalli and Bryan Parno. Algebraic reductions of knowledge. In *Advances in Cryptology - CRYPTO 2023*, volume 14084 of *Lecture Notes in Computer Science*, pages 669–701. Springer, 2023. doi:10.1007/978-3-031-38551-3_21.

[KS22] Abhiram Kothapalli and Srinath T. V. Setty. SuperNova: Proving universal machine executions without universal circuits. *IACR Cryptol. ePrint Arch.*, page 1758, 2022. URL: https://eprint.iacr.org/2022/1758.

[KS24] Abhiram Kothapalli and Srinath T. V. Setty. HyperNova: Recursive arguments for customizable constraint systems. In *Advances in Cryptology - CRYPTO 2024*, volume 14929 of *Lecture Notes in Computer Science*, pages 345–379. Springer, 2024. doi:10.1007/978-3-031-68403-6_11.

[KST22] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *Advances in Cryptology - CRYPTO 2022*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388. Springer, 2022. doi:10.1007/978-3-031-15985-5_13.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010. doi:10.1007/978-3-642-17373-8_11.

[Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy - S&P 1980*, pages 122–134. IEEE Computer Society, 1980. doi:10.1109/SP.1980.10006.

[Mer89] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89*, volume 435, pages 218–238. Springer, 1989. doi:10.1007/0-387-34805-0_21.

[NDC$^+$24] Wilson D. Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based SNARKs. In *Advances in Cryptology - CRYPTO 2024*, volume 14929 of *Lecture Notes in Computer Science*, pages 308–344. Springer, 2024. doi:10.1007/978-3-031-68403-6_10.

[NT16] Assa Naveh and Eran Tromer. PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *IEEE Symposium on Security and Privacy - S&P 2016*, pages 255–271. IEEE Computer Society, 2016. doi:10.1109/SP.2016.23.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991. doi:10.1007/3-540-46766-1_9.

[RZ23] Carla Ràfols and Alexandros Zacharakis. Folding schemes with selective verification. In *Progress in Cryptology - LATINCRYPT 2023*, volume 14168 of *Lecture Notes in Computer Science*, pages 229–248. Springer, 2023. doi:10.1007/978-3-031-44469-2_12.

[SBV+13] Srinath T. V. Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys '13*, pages 71–84. ACM, 2013. doi:10.1145/2465351.2465359.

[SF90] Laura A. Sanchis and Mark A. Fulk. On the efficient generation of language instances. *SIAM Journal on Computing*, 19(2):281–296, 1990. doi:10.1137/0219019.

[STW23] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Customizable constraint systems for succinct arguments. *IACR Cryptol. ePrint Arch.*, page 552, 2023. URL: https://eprint.iacr.org/2023/552.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography - TCC 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2008. doi:10.1007/978-3-540-78524-8_1.

[YLF+22] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. hbACSS: How to robustly share many secrets. In *Network and Distributed System Security Symposium - NDSS 2022*. The Internet Society, 2022. URL: https://www.ndss-symposium.org/ndss-paper/auto-draft-245/.

[ZSL04] Bin Benjamin Zhu, Mitchell D. Swanson, and Shipeng Li. Encryption and authentication for scalable multimedia: Current state of the art and challenges. *Internet Multimedia Management Systems V*, 5601:157–170, 2004. doi:10.1117/12.571869.

A Inner Product Relation of Committed Values (Additional Details)

A.1 Pedersen Commitments

Usually, a (single value) Pedersen commitment [Ped91] uses public parameters (or commitment key), G and H randomly sampled group elements from a group $\mathbb{G}$ over a field $\mathbb{F}$, and a commitment to $m \in \mathbb{F}$ using randomness $r \in \mathbb{F}$ is $c = rG + mH$. The opening of c to m is (m, r). A non-hiding Pedersen commitment simply does not use the randomness r, and also removes G from the public parameters. A non-hiding Pedersen commitment for multiple values uses public parameters $G_1, \dots, G_n \in \mathbb{G}$, and a commitment to $\mathbf{m} = (m_1, \dots, m_n)$

is $c = m_1G_1 + m_2G_2 + \cdots + m_nG_n$. Although this is called non-hiding, it is information-theoretically hiding for $n \geq 2$. The hiding versions of the Pedersen commitment scheme are perfectly hiding for $n \geq 1$, and all versions are computationally binding under the discrete logarithm assumption. As in [RZ23], we will work with the non-hiding version, but everything translates trivially to the hiding version.

Using our group notation from Section 1.4, a commitment key for a non-hiding Pedersen commitment for multiple values can be denoted as $[\mathbf{r}] \in \mathbb{G}^n$, and a commitment to $\mathbf{m} \in \mathbb{F}^n$ as $[c] = [\mathbf{r}]^\top \mathbf{m}$.

A.2 Non-interactive 2-folding scheme

This is the non-interactive 2-folding scheme for Inner Product Relation of Committed Values obtained by applying the Fiat-Shamir heuristic to the interactive protocol in Section 4.1.

Construction 3. (2-IPRCV) Let H denote a hash function sampled from a family of cryptographic hash functions. Construct 2-IPRCV = (Fold, FoldVerify) as follows.

- $\mathsf{Fold}((\mathsf{gk}, [\mathbf{r}], [\mathbf{s}]), ((([c_1], [d_1], z_1), (\mathbf{a}_1, \mathbf{b_1})), ((([c_2], [d_2], z_2), (\mathbf{a}_2, \mathbf{b_2})))$
  1. Compute the cross terms: $z_{1,2} = \mathbf{a_1}^\top \mathbf{b_2}$ and $z_{2,1} = \mathbf{a_2}^\top \mathbf{b}_1$.
  2. Compute a pseudo random challenge by hashing the parameters, public inputs, and cross terms: $\rho = H(\mathsf{gk}, [\mathbf{r}], [\mathbf{s}], [c_1], [d_1], z_1, [c_2], [d_2], z_2, z_{1,2}, z_{2,1})$.
  3. Construct the folded instance using ρ:
  $$\begin{aligned} [c] &= [c_1] + \rho[c_2] \\ [d] &= [d_1] + \rho^2[d_2] \\ z &= z_1 + \rho z_{2,1} + \rho^2 z_{1,2} + \rho^3 z_2 \\ \mathbf{a} &= \mathbf{a}_1 + \rho\mathbf{a}_2 \\ \mathbf{b} &= \mathbf{b}_1 + \rho^2\mathbf{b}_2, \end{aligned}$$
  4. Output $(([c], [d], z), (\mathbf{a}, \mathbf{b}), (z_{1,2}, z_{2,1}))$.
- $\mathsf{FoldVerify}((\mathsf{gk}, [\mathbf{r}], [\mathbf{s}]), ([c_1], [d_1], z_1), ([c_2], [d_2], z_2), ([c], [d], z), (z_{1,2}, z_{2,1}))$
  1. Compute $\rho = H(\mathsf{gk}, [\mathbf{r}], [\mathbf{s}], [c_1], [d_1], z_1, [c_2], [d_2], z_2, z_{1,2}, z_{2,1})$.
  2. Check that
  $$\begin{aligned} [c] =& [c_1] + \rho[c_2] \\ [d] =& [d_1] + \rho^2[d_2] \\ z =& z_1 + \rho z_{2,1} + \rho^2 z_{1,2} + \rho^3 z_2, \end{aligned}$$
  3. If so output 1, otherwise output 0.

A.3 Statement Hider

This is the statement hider for Inner Product Relation of Committed Values obtained by instantiating Construction 2 with Construction 3 and $\mathcal{R}' = \mathcal{R}_{\mathsf{gk},[\mathbf{r}],[\mathbf{s}]}$.

Construction 4. (IPRCV-SH) Let 2-IPRCV be constructed as in Construction 3 and $\mathsf{pp} = (\mathsf{gk}, [\mathbf{r}], [\mathbf{s}])$. Construct (Hide, Check) as follows.

- $\mathsf{Hide}(\mathsf{pp}, ([c_1], [d_1], z_1), (\mathbf{a}_1, \mathbf{b_1}), ((([c_\$], [d_\$], z_\$), (\mathbf{a}_\$, \mathbf{b}_\$))$
  1. Fold the two instances together:
  $(([c], [d], z), (\mathbf{a}, \mathbf{b}), (z_{1,2}, z_{2,1})) \leftarrow \text{2-IPRCV}.\mathsf{Fold}(\mathsf{pp}, ((([c_1], [d_1], z_1), (\mathbf{a}_1, \mathbf{b_1})),$
  $((([c_\$], [d_\$], z_\$), (\mathbf{a}_\$, \mathbf{b}_\$)))$
  2. Output $(([c], [d], z), (\mathbf{a}, \mathbf{b}), ((([c_\$], [d_\$], z_\$), z_{1,2}, z_{2,1}))$

- $\mathsf{Check}(\mathsf{pp}, ([c_1], [d_1], z_1), ([c], [d], z), (([c_\$], [d_\$], z_\$), z_{1,2}, z_{2,1}))$
  1. Output the result of
     $\mathsf{2\text{-}IPRCV.FoldVerify}(\mathsf{pp}, ([c_1], [d_1], z_1), ([c_\$], [d_\$], z_\$), ([c], [d], z), (z_{1,2}, z_{2,1}))$.

A.4 Additional Calculations for Theorem 4

From the following calculations, it can be seen that the $[d]$ obtained by folding (x_{1-b}, w_{1-b}) and $(x_\$, w_\$)$ are the same as in Equation (7).

$$\begin{aligned}
[d_1] + \rho^2[d_\$] &= [d_1] + [\mathbf{s}]^\top \rho^2 \mathbf{b}'_\$ \\
&= [d_1] + [\mathbf{s}]^\top (\mathbf{b}_0 + \rho^2\mathbf{b}_\$ - \mathbf{b}_1) \\
&= [d_1] + [d_0] + \rho^2[d_\$] - [d_1] \\
&= [d_0] + \rho^2[d_\$] = [d].
\end{aligned}$$

Recall that Equation (14) was obtained by inserting Equations (10) to (13) into Equation (9). Factoring out the $\mathbf{a}$ terms yields Equation (8), as follows.

$$\begin{aligned}
(9) &= \mathbf{a}_1^\top(\mathbf{b}_1 - \mathbf{b}_1 + \mathbf{b}_0 + \rho^2\mathbf{b}_\$ - \mathbf{b}_1 - \mathbf{b}_0 - \rho^2\mathbf{b}_\$ + \mathbf{b}_1) \\
&\quad + \mathbf{a}_0^\top(\mathbf{b}_1 + \mathbf{b}_0 + \rho^2\mathbf{b}_\$ - \mathbf{b}_1) + \mathbf{a}_\$^\top(\rho\mathbf{b}_1 + \rho\mathbf{b}_0 + \rho^3\mathbf{b}_\$ - \rho\mathbf{b}_1) \\
&= \mathbf{a}_1^\top\mathbf{0} + \mathbf{a}_0^\top(\mathbf{b}_0 + \rho^2\mathbf{b}_\$) + \mathbf{a}_\$^\top(\rho\mathbf{b}_0 + \rho^3\mathbf{b}_\$) \\
&= \mathbf{a}_0^\top\mathbf{b}_0 + \rho^2\mathbf{a}_0^\top\mathbf{b}_\$ + \rho\mathbf{a}_\$^\top\mathbf{b}_0 + \rho^3\mathbf{a}_\$^\top\mathbf{b}_\$ \\
&= z_0 + \rho^2 z_{0,\$} + \rho z_{\$,0} + \rho^3 z_\$ = z.
\end{aligned}$$

B Committed Relaxed R1CS (Additional Details)

We now present the proof that

$$A\mathbf{z}'_2 \circ B\mathbf{z}'_2 = u'_2 C\mathbf{z}'_2 + \mathbf{e}'_2 \tag{22}$$

in greater detail. Essentially, this is done by expanding each side, and obtaining the same. Note that by construction

$$\mathbf{z}'_2 := \begin{pmatrix} \mathbf{w}'_2 \\ \mathbf{x}'_2 \\ u'_2 \end{pmatrix} = \begin{pmatrix} \rho^{-1}(\mathbf{w}_1 + \rho\mathbf{w}_2 - \mathbf{w}'_1) \\ \rho^{-1}(\mathbf{x}_1 + \rho\mathbf{x}_2 - \mathbf{x}'_1) \\ \rho^{-1}(u_1 + \rho u_2 - u'_1) \end{pmatrix} = \rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1).$$

We first expand the left side of Equation (22) using the distributive laws for entry-wise multiplication.

$$\begin{aligned}
A\mathbf{z}'_2 \circ B\mathbf{z}'_2 &= A(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1)) \circ B(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}'_1)) \\
&= \rho^{-2}(A\mathbf{z}_1 \circ B\mathbf{z}_1 + \rho A\mathbf{z}_1 \circ B\mathbf{z}_2 - A\mathbf{z}_1 \circ B\mathbf{z}'_1 \\
&\qquad + \rho A\mathbf{z}_2 \circ B\mathbf{z}_1 + \rho^2 A\mathbf{z}_2 \circ B\mathbf{z}_2 - \rho A\mathbf{z}_2 \circ B\mathbf{z}'_1 \\
&\qquad - A\mathbf{z}'_1 \circ B\mathbf{z}_1 - \rho A\mathbf{z}'_1 \circ B\mathbf{z}_2 + A\mathbf{z}'_1 \circ B\mathbf{z}'_1) \\
&= A\mathbf{z}_2 \circ B\mathbf{z}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - A\mathbf{z}_2 \circ B\mathbf{z}'_1 - A\mathbf{z}'_1 \circ B\mathbf{z}_2) \\
&\quad + \rho^{-2}(A\mathbf{z}_1 \circ B\mathbf{z}_1 - A\mathbf{z}_1 \circ B\mathbf{z}'_1 - A\mathbf{z}'_1 \circ B\mathbf{z}_1 + A\mathbf{z}'_1 \circ B\mathbf{z}'_1) \\
&= u_2 C\mathbf{z}_2 + \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - A\mathbf{z}_2 \circ B\mathbf{z}'_1 - A\mathbf{z}'_1 \circ B\mathbf{z}_2) \\
&\quad + \rho^{-2}(u_1 C\mathbf{z}_1 + \mathbf{e}_1 - A\mathbf{z}_1 \circ B\mathbf{z}'_1 - A\mathbf{z}'_1 \circ B\mathbf{z}_1 + u'_1 C\mathbf{z}'_1 + \mathbf{e}'_1).
\end{aligned} \tag{23}$$

Before we expand the right side of Equation (22), we expand the error term $\mathbf{e}_2'$. This is done by inserting the cross terms t and t', and then inserting u_2' and $\mathbf{z}_2'$.

$$\begin{aligned}
\mathbf{e}_2' &= \rho^{-2}(\mathbf{e}_1 + \rho t + \rho^2\mathbf{e}_2 - \mathbf{e}_1' - \rho t') \\
&= \rho^{-2}(\mathbf{e}_1 + \rho(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1) + \rho^2\mathbf{e}_2 \\
&\qquad - \mathbf{e}_1' - \rho(A\mathbf{z}_1' \circ B\mathbf{z}_2' + A\mathbf{z}_2' \circ B\mathbf{z}_1' - u_1' C\mathbf{z}_2' - u_2' C\mathbf{z}_1')) \\
&= \rho^{-2}(\mathbf{e}_1 + \rho(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1) + \rho^2\mathbf{e}_2 - \mathbf{e}_1' \\
&\qquad - \rho(A\mathbf{z}_1' \circ B(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}_1')) + A(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}_1')) \circ B\mathbf{z}_1' \\
&\qquad\quad - u_1' C(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}_1')) - \rho^{-1}(u_1 + \rho u_2 - u_1')C\mathbf{z}_1')) \\
&= \rho^{-2}(\mathbf{e}_1 + \rho(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1) + \rho^2\mathbf{e}_2 - \mathbf{e}_1' \\
&\qquad - A\mathbf{z}_1' \circ B\mathbf{z}_1 - \rho A\mathbf{z}_1' \circ B\mathbf{z}_2 + A\mathbf{z}_1' \circ B\mathbf{z}_1' - A\mathbf{z}_1 \circ B\mathbf{z}_1' - \rho A\mathbf{z}_2 \circ B\mathbf{z}_1' \\
&\qquad + A\mathbf{z}_1' \circ B\mathbf{z}_1' + u_1' C\mathbf{z}_1 + \rho u_1' C\mathbf{z}_2 - u_1' C\mathbf{z}_1' + u_1 C\mathbf{z}_1' + \rho u_2 C\mathbf{z}_1' - u_1' C\mathbf{z}_1') \\
&= \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1 \\
&\qquad - A\mathbf{z}_1' \circ B\mathbf{z}_2 - A\mathbf{z}_2 \circ B\mathbf{z}_1' + u_1' C\mathbf{z}_2 + u_2 C\mathbf{z}_1') \\
&\quad + \rho^{-2}(\mathbf{e}_1 - \mathbf{e}_1' - A\mathbf{z}_1' \circ B\mathbf{z}_1 + A\mathbf{z}_1' \circ B\mathbf{z}_1' - A\mathbf{z}_1 \circ B\mathbf{z}_1' + A\mathbf{z}_1' \circ B\mathbf{z}_1' \\
&\qquad + u_1' C\mathbf{z}_1 - u_1' C\mathbf{z}_1' + u_1 C\mathbf{z}_1' - u_1' C\mathbf{z}_1'). \\
&= \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - u_1 C\mathbf{z}_2 - u_2 C\mathbf{z}_1 \\
&\qquad - A\mathbf{z}_1' \circ B\mathbf{z}_2 - A\mathbf{z}_2 \circ B\mathbf{z}_1' + u_1' C\mathbf{z}_2 + u_2 C\mathbf{z}_1') \qquad (24) \\
&\quad + \rho^{-2}(\mathbf{e}_1 + \mathbf{e}_1' - A\mathbf{z}_1' \circ B\mathbf{z}_1 - A\mathbf{z}_1 \circ B\mathbf{z}_1' + u_1' C\mathbf{z}_1 + u_1 C\mathbf{z}_1').
\end{aligned}$$

For Equation (24), we applied that $A\mathbf{z}_1' \circ B\mathbf{z}_1' = u_1' C\mathbf{z}_1' + \mathbf{e}_1'$ twice. We are now ready to expand the right side of Equation (22). At Equation (25), we insert Equation (24) in place of $\mathbf{e}_2'$, and cancel out where applicable.

$$\begin{aligned}
u_2' C\mathbf{z}_2' + \mathbf{e}_2' &= \rho^{-1}(u_1 + \rho u_2 - u_1')C(\rho^{-1}(\mathbf{z}_1 + \rho\mathbf{z}_2 - \mathbf{z}_1')) + \mathbf{e}_2' \\
&= \rho^{-2}(u_1 C\mathbf{z}_1 + \rho u_1 C\mathbf{z}_2 - u_1 C\mathbf{z}_1' + \rho u_2 C\mathbf{z}_1 + \rho^2 u_2 C\mathbf{z}_2 \\
&\qquad - \rho u_2 C\mathbf{z}_1' - u_1' C\mathbf{z}_1 - \rho u_1' C\mathbf{z}_2 + u_1' C\mathbf{z}_1') + \mathbf{e}_2' \\
&= u_2 C\mathbf{z}_2 + \rho^{-1}(u_1 C\mathbf{z}_2 + u_2 C\mathbf{z}_1 - u_2 C\mathbf{z}_1' - u_1' C\mathbf{z}_2) \\
&\quad + \rho^{-2}(u_1 C\mathbf{z}_1 - u_1 C\mathbf{z}_1' - u_1' C\mathbf{z}_1 + u_1' C\mathbf{z}_1') + \mathbf{e}_2' \\
&= u_2 C\mathbf{z}_2 + \mathbf{e}_2 + \rho^{-1}(A\mathbf{z}_1 \circ B\mathbf{z}_2 + A\mathbf{z}_2 \circ B\mathbf{z}_1 - A\mathbf{z}_1' \circ B\mathbf{z}_2 - A\mathbf{z}_2 \circ B\mathbf{z}_1') \qquad (25) \\
&\quad + \rho^{-2}(u_1 C\mathbf{z}_1 + \mathbf{e}_1 + u_1' C\mathbf{z}_1' + \mathbf{e}_1' - A\mathbf{z}_1' \circ B\mathbf{z}_1 - A\mathbf{z}_1 \circ B\mathbf{z}_1').
\end{aligned}$$

It can now be verified that Equation (22) holds, simply by comparing Equation (23) and Equation (25).