Circuit-Succinct Algebraic Batch Arguments from Projective Functional Commitments*

David Balbás[1], Dario Fiore[2], and Russell W. F. Lai[3]

[1] ETH Zurich, Zurich, Switzerland.
dbalbas@ethz.ch
[2] IMDEA Software Institute, Madrid, Spain.
dario.fiore@imdea.org
[3] Aalto University, Espoo, Finland.
russell.lai@aalto.fi

Abstract. Batch arguments for NP (BARGs) are non-interactive proof systems that allow a prover to convince a verifier that k NP statements $\mathtt{x}_1, \ldots, \mathtt{x}_k$ are valid relative to some circuit $\mathcal{C}$, i.e. there exist witnesses $\mathtt{w}_i$ such that $(\mathtt{x}_i, \mathtt{w}_i)$ satisfy $\mathcal{C}$ for all i, while the proof size remains sublinear in k. Most existing BARG constructions achieve a proof size of $|\pi| = \mathsf{poly}(\lambda, |\mathcal{C}|, \log k)$ for large or not explicitly specified poly acting on $|\mathcal{C}|$, with two exceptions:

1. Devadas et al. and Paneth and Pass's "rate-1" constructions [FOCS'22] achieve $|\pi| = |\mathtt{w}| + \mathcal{O}(|\mathtt{w}|/\lambda) + \mathsf{poly}(\lambda, \log k)$ (with matching verification time for Paneth and Pass), but for not explicitly specified poly due to non-black-box use of cryptographic primitives.
2. Waters and Wu's algebraic (pairing-based) construction [Crypto'22] achieves $|\pi| = \mathcal{O}(\lambda \cdot |\mathcal{C}|)$.

In this work, we give the first algebraic (pairing-based) construction of BARG that achieves proof size and online verifier runtime $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$. We achieve our result by means of a compiler which builds a BARG generically from a *projective chainable functional commitment* (PCFC), which supports somewhere extraction, subvector projection, and functional openings. We then construct a PCFC from the standard MDDH assumption in bilinear groups by building on top of the functional commitment for circuits by Wee and Wu [Eurocrypt'24]. Our black-box transformation may be of independent interest for understanding the connection between functional commitments and BARGs and towards obtaining other algebraic constructions of the latter.

1 Introduction

Succinct non-interactive arguments (SNARGs) [Mic94] allow a prover to convince a verifier of a statement's validity with a short and quickly verifiable proof. For example, they can prove an NP statement $\mathtt{x}$ checkable by a circuit $\mathcal{C}$, i.e., that $\exists \mathtt{w} : \mathcal{C}(\mathtt{x}, \mathtt{w}) = 1$, with a proof of size $\mathsf{poly}(\lambda, \log|\mathcal{C}|)$. They are a fundamental concept in the theory of proof systems and an essential tool in practical applications like outsourcing of computation, blockchains, and cryptocurrencies.

A central question about succinct arguments is the assumptions required for their construction. Gentry and Wichs provided a significant answer by proving that it is impossible to construct adaptively-sound SNARGs for NP with a black-box reduction to falsifiable assumptions [GW11].

Batch Arguments (BARGs). In search of stable foundations for this primitive, various lines of research have explored alternative methods to circumvent this impossibility result and to construct succinct arguments from standard assumptions. One notable approach is to *relax the full power of SNARGs*, while ensuring that they remain useful in applications. For example, a series of works considered SNARGs for the class P of *deterministic polynomial-time computations*, which have a

* An extended abstract of this paper appears in the proceedings of SCN 2026. This is the full version.

direct application to verifiable computation, and showed how to construct them from standard falsifiable assumptions [KPY19, GZ21, JKKZ21, CJJ22].

In this work, we focus on another restricted form of SNARGs — batch arguments for NP (BARGs) — which achieve a relaxed notion of succinctness. In a BARG scheme, one can prove a collection of NP statements, $\mathtt{x}_1, \ldots, \mathtt{x}_k$, i.e., $\forall i = 1, \ldots, k, \exists \mathtt{w}_i : \mathcal{C}(\mathtt{x}_i, \mathtt{w}_i) = 1$, with a proof that scales *sublinearly in the number of statements*, e.g., of size $\mathsf{poly}(\lambda, |\mathcal{C}|, \log k)$, but not necessarily sublinearly in the circuit size. Due to its distinctive properties – succinct proofs and realizations from standard falsifiable assumptions – BARGs are emerging as a useful tool to realise other cryptographic primitives from standard assumptions. Notably, for several primitives that have simple folklore realizations from SNARGs, recent works have shown that the SNARG can be replaced by a BARG. This approach is exemplified by applications of BARGs to non-interactive delegation [BHK17, KPY19, KVZ21, CJJ22], aggregate signatures [WW22, BCJP24], and homomorphic signatures [ABF24, Goy24, ACG24].

Non-Black-Box vs. Algebraic. In the state of the art, the main approach to construct BARGs involves non-black-box combinations of cryptographic primitives, with instantiations that often rely on probabilistically checkable proofs (PCPs) and correlation intractable (CI) hash functions [CGH98, CCH+19, PS19, JJ21]. This direction has led to constructions from LWE [CJJ22, DGKV22, PP22] or subexponential DDH [CGJ+23]. Notably, the rate-1 constructions of [DGKV22, PP22] achieve a proof size (and verification time for [PP22]) of $|\pi| = |\mathtt{w}| + \mathcal{O}(|\mathtt{w}|/\lambda) + \mathsf{poly}(\lambda, \log k)$, which is sublinear in $|\mathcal{C}|$. However, despite the near-optimal asymptotic efficiency, these constructions often have large concrete overheads, i.e. the poly functions in proof size and verification time expressions are often large or not explicitly specified.

An exception to the non-black-box approach is the work of Waters and Wu [WW22] who gave direct algebraic constructions over bilinear pairings based on the decision linear (DLin) assumption. Making black-box use of the underlying algebraic structure often translates into more concretely efficient and relatively simple constructions. Indeed, the scheme of [WW22] achieves a proof size of $|\pi| = \mathcal{O}(\lambda \cdot |\mathcal{C}|)$. The result is later improved by [GLWW24, CEW25], both of which reduce the public parameter size while keeping essentially the same design. However, in both [WW22, GLWW24] the verifier runs in time $\Omega(\lambda \cdot |\mathcal{C}|)$, even if offline preprocessing is allowed.

1.1 Our Contribution

In this work, we continue the line of work established by [WW22, GLWW24, CEW25] on obtaining algebraic constructions of BARGs. Specifically, we present a new construction of BARG directly over bilinear pairings under the matrix decisional Diffie-Hellman (MDDH) assumption, which achieves proofs of size $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$ with matching online verification time, improving upon [WW22, GLWW24, CEW25].

Additionally, our BARG natively supports proving k different circuits, one for each statement. In prior work, this property is achieved with the overhead of using a universal circuit $\mathcal{U}$ to define the batch language and including the description of each circuit in the statement, i.e., by proving that $\mathcal{U}((\mathcal{C}_i, \mathtt{x}_i), \mathtt{w}_i) = 1$.

To achieve the above result, we rely on a new notion that we call *projective chainable functional commitments (PCFC)*, which is a combination of projective commitments (PC) [GZ21, WW24] and chainable functional commitments (CFC) [BCFL23], together with two technical ingredients:

1. A black-box compiler turning PCFCs with certain properties to BARGs; and

2. An algebraic construction of such a PCFC based on the MDDH assumption.

Our black-box compiler is also a conceptual contribution, as it reveals new connections between functional commitments (FCs) [LRY16] and BARGs, which relax SNARGs by forgoing soundness and succinctness respectively. In short, our compiler identifies the properties that a functional commitment must have such that succinct BARGs can be built from it. We believe that this abstraction may also pave the way towards obtaining other direct algebraic constructions of BARGs from different assumptions, such as lattices, which is an open problem.

Limitations and Open Questions. Our BARG construction introduces a new trade-off between proof size and public parameter size. While the improvement in the proof size from $\mathcal{O}(\lambda \cdot |\mathcal{C}|)$ to $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$ is significant, it comes at the price of increasing the size of the public parameters to $\mathcal{O}(\lambda \cdot k^5 \cdot |\mathcal{C}|^5)$ (as opposed to $\mathcal{O}(\lambda \cdot k)$ in [CEW25]). Naturally, this is an important bottleneck if our scheme were to be used in practice. A natural open question is whether the public parameters of our scheme can be shortened, and whether this trade-off between proof size and public parameter size is necessary or can be circumvented.

Second, our techniques enable only computational somewhere extractability, whereas this property holds statistically in the scheme by Waters and Wu [WW22]. Since all known schemes anyway need to rely on computational assumptions for CRS indistinguishability, this does not seem to be an issue in most applications.[4] Yet, it is natural to ask whether our scheme can be lifted to achieve statistical somewhere extractability.

1.2 Other Related Work

The usual BARG succinctness notion requires that the proof size is bounded by $\mathsf{poly}(\lambda, |\mathcal{C}|, \log k)$, where k is the size of the batch. An interesting exception in the state of the art is the BARG for NP of Garg et al. [GSWW22] in which proofs have size $\mathsf{poly}(\lambda, \log|\mathcal{C}|, \log k)$, that is sublinear also in the size of the circuit. The scheme, built from indistinguishability obfuscation and one-way functions, is proven non-adaptively sound. Therefore, it does not bypass the impossibility result of [GW11]. In contrast, our result achieves adaptive soundness, in line with most of the BARG literature.

Another approach to build BARGs from other cryptographic primitives is that of Kalai et al. who show how to build BARGs for NP from *flexible SNARGs for RAM with partial input soundness*, additionally assuming the existence of rate-1 string OT [KLVW23]. On one hand, the spirit of this connection is similar to ours, especially after observing that their notion of SNARG for RAM is similar to that of functional commitments. On the other hand, their BARG construction differs from ours in two main aspects. First, it is non-black-box due to the recursive usage of the SNARG for RAM. Thus, it cannot yield algebraic BARGs, departing from one of the main motivations of our work. Second, they do not provide a direct construction of flexible SNARGs for RAM but show that, in turn, these can be built from BARGs (again, additionally assuming rate-1 string OT). While this result establishes an interesting connection between the two primitives, it ultimately does not yield new BARGs since the flexible SNARG for RAM should be instantiated from existing BARGs and thus from either PCPs and CI-hash, or from [WW22].

[4] Statistical somewhere extractability does play a role in constructions that combine BARGs with obfuscation [DJWW25], but for such 'galactic' constructions there is no benefit in relying on algebraic BARGs.

2 Technical Overview

The main result of this work is an algebraic construction of BARG with proof size and online verification time $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$. We obtain this result by compiling, in a black-box manner, a new commitment primitive that we call *projective chainable functional commitment* (PCFC), for which we give an algebraic construction over bilinear groups.

To give a high-level overview of our approach, in Section 2.1, we begin by recalling and generalizing the notions of projective commitments (PC), [GZ21, WW24] and chainable functional commitments (CFC) [BCFL23], and introducing the notion of PCFC. We then revisit in Section 2.2 the algebraic BARG construction of Waters and Wu [WW22] by viewing it as a black-box compilation from a PCFC to a BARG. From this abstract perspective, it becomes clear how, given a PCFC with stronger properties, the compiler can be tweaked to improve the proof size and online verification time from $\mathcal{O}(\lambda \cdot |\mathcal{C}|)$ to $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$. We thus present in Section 2.3 how to obtain such a PCFC from pairing groups based on the MDDH assumption. Finally, in Section 2.5 we conclude with a summary of our results and an outlook on future work.

2.1 Projective Chainable Functional Commitments (PCFC)

A (deterministic) commitment scheme allows the generation of a short commitment com_x which is computationally binding to a vector $\boldsymbol{x} \in \mathcal{M}^\ell$ potentially much longer than com_x. Many variations of the notion exist that extend the basic functionality and security of a commitment scheme. Our new notion, PCFC, which we formalise in Section 4, is a combination of two such variations: projective commitments (PC) and chainable functional commitments (CFC).

Projective Commitments (PC). In a *Projective commitment (PC)* scheme, introduced by Wee and Wu [WW24][5], the commitment key ck of the scheme can be sampled in two modes: the *normal mode* and the *projective mode*. Sampling the key in normal mode leads to a standard commitment key ck that computationally binds a vector $\boldsymbol{x} \in \mathcal{M}^\ell$ to a commitment com. In projective mode, the commitment key ck is sampled together with a trapdoor td that encodes a hidden index set $I \subseteq [\ell]$. Here, ck hides a *projective key*, which is essentially a secondary commitment key that is sampled only at the positions indexed by I. The trapdoor td allows one to project any commitment com to $\boldsymbol{x} \in \mathcal{M}^\ell$ to a commitment pcom to the subvector $\boldsymbol{x}_I \in \mathcal{M}^I$.[6] Without knowing td, both modes are computationally indistinguishable.

In more detail, a PC features the following algorithms (among others):

- $\mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$: Generates an ordinary-looking commitment key ck together with a trapdoor td which encodes a secret index set $I \subseteq [\ell]$.
- $\mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}) \to \mathsf{pcom}$: Generate a projective commitment pcom of the subvector $\boldsymbol{x}_I$.
- $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) \to \mathsf{pcom}$: Project a commitment com into a projective commitment pcom.

Projective commitments are closely related to somewhere extractable commitments [HW15, CJJ21], which essentially allow one to generate commitment keys so that a commitment com_x is *statistically binding* at some positions $I \subset [\ell]$. The trapdoor td allows one to actually extract $\boldsymbol{x}_I \in \bar{\mathcal{M}}$ (where $\bar{\mathcal{M}} \subseteq \mathcal{M}$ is a smaller extraction space) leveraging the statistical binding property. In this work, we enhance the original notion of projective commitments to also provide somewhere extractability. This is, our commitments also feature the following algorithm:

[5] Similar properties for commitments appear in [GZ21] under the name of G-extractability.
[6] The subvector $\boldsymbol{x}_I$ is obtained by taking the entries of $\boldsymbol{x}$ indexed by I.

– $\mathsf{ProjExt}(\mathsf{td}, \mathsf{com})$: Given com, which is guaranteed[7] to be committing to a vector whose positions indexed by I belong to the projective subspace $\bar{\mathcal{M}}^I$, extract $\boldsymbol{x}_I \in \bar{\mathcal{M}}^I$ such that $\mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}_I) = \mathsf{Proj}(\mathsf{td}, \mathsf{com})$.

To achieve the extractability outlined in the above simplified description, a commitment com has to be of size linear in $|I|$, which will be too large for our application of constructing circuit-succinct BARGs. Instead, we further augment the syntax of the commitment algorithm Com (among others) to take as input another index set J chosen from some admissible set $\mathcal{J} \subseteq 2^{[\ell]}$ so that the resulting commitment com is committing to a subsector $\boldsymbol{x}_J \in \mathcal{M}^J$. Now, the extractability guarantee is only required to hold at the positions $I \cap J$ for any $J \in \mathcal{J}$. By picking $\mathcal{J}$ appropriately, we can ensure that $|I \cap J|$ and hence $|\mathsf{com}|$ is never too large which eventually allows us to achieve circuit-succinctness.

Chainable Functional Commitments (CFC). While ordinary commitments are "all-or-nothing", functional commitments (FC) [LRY16] enable the generation of a succinct (weakly sound) proof asserting the correctness of $\boldsymbol{y} = f(\boldsymbol{x})$ for the vector $\boldsymbol{x}$ committed in com_x. *Chainable functional commitments (CFC)* [BCFL23] extend this functionality and allow to generate proofs for $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ where both the inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ and the output $\boldsymbol{y}$ are committed. A CFC is "chainable" in the sense that it allows one to prove relations between a sequence of committed inputs and outputs, in a "commit-and-prove" fashion. A (C)FC usually satisfies a relaxed notion of soundness called evaluation binding – no efficient prover could produce valid proofs for $(f, \mathsf{com}_x, \mathsf{com}_y)$ and $(f, \mathsf{com}_x, \mathsf{com}'_y)$ for $\mathsf{com}_y \neq \mathsf{com}'_y$. In this work, however, we deviate from this usual notion and require a stronger property that we introduce next.

Projective Chainable Functional Commitments (PCFC). Although PC and CFC are useful notions in their own rights, combining them into a PCFC makes the result more powerful than the sum of the two, since it allows us to reason about the functional relations between projective commitments and their extracted messages, whose spaces are hidden from a malicious prover. Such a property already resembles the functionality of a BARG, as both allow to extract part of the inputs to the batch computation while verifying its correctness.

In more detail, recall that a PC commitment key ck hides a secret index set $I \subseteq [\ell]$ while a CFC allows the generation of functional proofs. To glue these two notions together, we consider a class of functions called "I-separable functions" and define a property called *projective chain binding* with respect to these functions.

I-Separable Functions. We say that a function $f : (\mathcal{M}^\ell)^k \to \mathcal{M}^\ell$ is I-separable if the coordinates in I of the output vector only depend on the coordinates in I of each input vector.[8] A simple example is when f is the parallel application of another function $f_I : \mathcal{M}^{\ell/k} \to \mathcal{M}^{\ell/k}$ such that $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (f_I(\boldsymbol{x}_1), \ldots, f_I(\boldsymbol{x}_k))$.

Projective Chain Binding. We say that a PCFC is projective chain binding for an I-separable function f if no PPT adversary, given both a commitment key ck, which is trapdoored to be projective at I, *and the trapdoor* td, can produce tuples $(\mathsf{com}, \mathsf{com}_y, \pi)$ and $(\mathsf{com}', \mathsf{com}'_y, \pi')$ where

– π and π' are valid CFC proofs for $(f, \mathsf{com}, \mathsf{com}_y)$ and $(f, \mathsf{com}', \mathsf{com}'_y)$, respectively, i.e., the output committed in com_y (resp. com'_y) is the evaluation of f on the input committed in com (resp. com');

[7] To be (weakly) asserted by another algorithm called $\mathsf{ProjProve}$ and verified by $\mathsf{ProjVer}$.
[8] Our formal definition generalizes to allow different separability sets for inputs and outputs.

– $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}')$, i.e., the vectors committed in com and com' have the same values in the locations of the set I;
– but $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_y)$, i.e., the vectors committed in com_y and com'_y *differ* in the locations of the set I, which should not be possible given that f is I-separable.

Essentially, the projective chain binding property allows a security reduction to transfer the evaluation binding property of the CFC to hold also in the projective subspace. This is especially powerful if we leverage the somewhere extractability property of PC on the projective subspace. It is worth noting that some of the proof systems that are internally used to build the FC in [WW24] achieve a similar notion to projective chain binding (although the notion is not attained by the FC itself). In their case, however, the projection trapdoor is not available to the adversary. As we discuss in the next section, our strengthening is crucial when proving the security of our circuit-succinct BARG construction.

2.2 Black-Box Compiler from PCFC to BARG

Recall that in a BARG the prover aims to convince the verifier that k NP statements $\mathtt{x}_1, \ldots, \mathtt{x}_k$ are valid. That is, the prover knows witnesses $\mathtt{w}_1, \ldots, \mathtt{w}_k$ such that $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i) = 1$ for all $i \in [k]$. The main security property of BARGs, known as somewhere extractability, requires the existence of an efficient extractor which, given a valid proof, can extract a valid witness for the i^*-th instance $\mathtt{w}_{i^*}$, where i^* is a hidden index programmed into the common reference string.

Suppose the circuit $\mathcal{C}$ is such that the input-output relation of each gate $g \in \mathcal{C}$ is captured by a gate function χ_g represented as a quadratic polynomial over a field $\mathbb{F}$, and write $x_{i,g}$ for the output value from the g-th gate (this is, the g-th circuit wire) in the evaluation of $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i)$. In the following, let $\mathcal{G}_{\mathsf{stmt}}, \mathcal{G}_{\mathsf{wit}}, \mathcal{G}_{\mathsf{out}}$ be the sets of wires of $\mathcal{C}$ corresponding to the statement input, the witness input and all gate outputs, respectively.

Basic Compiler. To obtain an improved algebraic BARG, a natural starting point is the algebraic BARG of Waters and Wu [WW22]. Our first step is to view[9] their construction as black-box compiler which turns a PCFC into a BARG. We formalize this compiler in Appendix A and give a high-level overview here:

Basic Compiler (Abstraction of [WW22]):

1. The prover PCFC-commits to the evaluation of $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i)$ in a wire-by-wire fashion across all k circuits, i.e. for each wire $g \in \mathcal{G}_{\mathsf{wit}} \cup \mathcal{G}_{\mathsf{out}}$, they commit to the values $\boldsymbol{x}_g = (x_{1,g}, \ldots, x_{k,g})$ of the g-th wire as com_g.
2. For each witness wire $g \in \mathcal{G}_{\mathsf{wit}}$, compute a PC proof for the bariness of $\boldsymbol{x}_g$ committed in com_g, i.e. $\boldsymbol{x}_g \in \{0,1\}^k$, where $\{0,1\} = \bar{\mathcal{M}}$ plays the role of the projective subspace.
3. For each gate output wire $g \in \mathcal{G}_{\mathsf{out}}$, compute a CFC proof for $\boldsymbol{x}_g$ committed in com_g being computed correctly with χ_g w.r.t. the commitments of the input wires.

[9] Although Waters and Wu presented the same intuition (obliviously using different terminologies than PCFCs) when explaining their construction in [WW22], formally, they constructed the BARG directly in terms of algebraic operations. This results in a construction with many intertwined components, making it difficult to dissect and improve upon.

Instantiating the PCFC with the implicit PCFC for quadratic functions from [WW22] yields a BARG with proof size $\mathcal{O}(\lambda \cdot |\mathcal{C}|)$, where the linear dependency on $|\mathcal{C}|$ is due to the generation of commitments and proofs in a wire-by-wire fashion.[10]

Circuit-Succinct Compiler. Without going yet into formal details, one aspect of the security proof of the basic compiler stands out as an obstacle for reducing the proof size to be sublinear in $|\mathcal{C}|$: The security reduction has to extract the values of all $|\mathcal{C}|$ wires in one of the k circuits, and verify the functional proofs of all gate functions. The extractability of all $|\mathcal{C}|$ wires of a circuit information-theoretically implies that the BARG proof must have size linear in $|\mathcal{C}|$.

Towards improving the proof size and online verification time from $\mathcal{O}(\lambda \cdot |\mathcal{C}|)$ to $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$, a crucial observation is that the security reduction does not necessarily need to extract all $|\mathcal{C}|$ wires of (the i^*-th copy of) the circuit $\mathcal{C}$ but only its input $\mathtt{w}_{i^*}$. That is, if the PCFC is expressive enough to support generation of CFC proofs for arbitrary circuits, then the burden of checking that the circuit is evaluated correctly can be delegated to the prover. Based on this observation, we obtain the following circuit-succinct compiler, which is detailed in Section 5:

Circuit-Succinct Compiler:

1. The prover PCFC-commits to the *witness input wires* of $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i)$ across all k circuits, i.e. for each witness input wire $g \in \mathcal{G}_{\mathsf{wit}}$, they commit to the values $\boldsymbol{x}_g = (x_{1,g}, \ldots, x_{k,g})$ of the g-th wire as com_g.
2. For each witness wire $g \in \mathcal{G}_{\mathsf{wit}}$, compute a PC proof for the binariness of $\boldsymbol{x}_g$ committed in com_g, i.e. $\boldsymbol{x}_g \in \{0,1\}^k$, where $\{0,1\} = \bar{\mathcal{M}}$ plays the role of the projective subspace.
3. Compute a CFC proof for $f_{\mathcal{C}}(\boldsymbol{x}_g : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}) = \mathbf{1}_k$ where $f_{\mathcal{C}}$ consists of k parallel copies of $\mathcal{C}$.

Assuming that the commitments $(\mathsf{com}_g)_{g \in \mathcal{G}_{\mathsf{wit}}}$ and the CFC proof for $f_{\mathcal{C}}$ are sufficiently succinct, we can conclude by inspection that the BARG is circuit-succinct, i.e. a proof is of size $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$.

The main result is Theorem 1, which we state informally below:

Theorem 1 (simplified). *Let I_i for $i \in [k]$ be the sets corresponding to the positions of $\{x_{i,g} : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}\}$, and let $f_{\mathcal{C}} : \mathcal{C} : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}$ be the function defined by $f_{\mathcal{C}}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (\mathcal{C}(\boldsymbol{x}_1), \ldots, \mathcal{C}(\boldsymbol{x}_k))$. Let PCFC be a projective chainable functional commitment scheme that supports $f_{\mathcal{C}}$ as an I_i-separable function for every $i \in [k]$, and which satisfies:*

- *PC somewhere extractability for I_i on $\bar{\mathcal{M}}$.*
- *PCFC projective chain binding on I_i.*

Then, the obtained BARG is a somewhere extractable batch argument for NP whose proof size is $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$. Moreover, BARG admits efficient verification with preprocessing.

Security of the Circuit-Succinct Compiler. Assuming that the underlying PCFC is projective chain binding, we can prove that the above black-box compiler yields a somewhere-extractable BARG. In the security experiment, we consider a trapdoored setup which programs the PCFC commitment key so that it is projective on the subspace I_{i^*} defined by the hidden index i^* of the BARG extractor. The set I_{i^*} contains the location of the statement and witness of the i^*-th instance, i.e. $(\mathtt{x}_{i^*}, \mathtt{w}_{i^*})$.

[10] Here, we glossed over the detail that the somewhere extractability of the BARG obtained above is proven based on another property of PCFC that we call *functional extractability* instead of projective chain binding. We refer to Appendix A for details.

Recall that a BARG proof contains commitments com_g for each witness input wire $g \in \mathcal{G}_{\mathsf{wit}}$ across all k circuits, along with their projection proofs. The security proof proceeds in two broad steps.[11]

1. Appeal to PC somewhere extractability for each com_g, which ensures the extraction of a binary witness $\mathtt{w}_{i^*} \in \{0,1\}^{\ell_{\mathtt{w}}}$ and that $\mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}_g) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}_g)$ for each $g \in \mathcal{G}_{\mathsf{wit}}$.
2. Use projective chain binding to ensure that $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) = 1$.

To explain the second step in more detail, in the case $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) = 0$, we can construct a reduction $\mathcal{B}$ from projective chain binding. For simplicity, let us ignore the statement wires $g \in \mathcal{G}_{\mathsf{stmt}}$ (we can think of them being hardwired in $f_{\mathcal{C}}$) as they are not important for the reduction. The reduction $\mathcal{B}$ runs the malicious prover $\mathcal{A}$ to obtain the commitments com_g for all witness input wires $g \in \mathcal{G}_{\mathsf{wit}}$ and a valid CFC proof π for $f_{\mathcal{C}}(\boldsymbol{x}_g : g \in \mathcal{G}_{\mathsf{wit}}) = \mathbf{1}_k$. Using the trapdoor td, $\mathcal{B}$ can extract the i^*-th witness $\mathtt{w}_{i^*}$ from the commitments com_g. It then uses the extracted witness $\mathtt{w}_{i^*}$ together with dummy witnesses $\mathtt{w}_i = \mathbf{0}_{\ell_{\mathtt{w}}}$ for the other slots $i \neq i^*$ to compute the commitments $(\mathsf{com}'_g)_{g \in \mathcal{G}_{\mathsf{wit}}}$ together with another CFC proof π' for $f_{\mathcal{C}}(\boldsymbol{x}'_g : g \in \mathcal{G}_{\mathsf{wit}}) = \boldsymbol{b}$ for some $\boldsymbol{b} \in \{0,1\}^k$ with $b_{i^*} = 0$. By the correctness of the PCFC, it must hold that $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_g) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_g)$ for all $g \in \mathcal{G}_{\mathsf{wit}}$ and that π' is a valid CFC proof. The reduction $\mathcal{B}$ can thus output $(\mathsf{com}_g, \mathsf{com}'_g)_{g \in \mathcal{G}_{\mathsf{wit}}}, \mathsf{com}_y, \mathsf{com}'_y, \pi, \pi'$ where $\mathsf{com}_y = \mathsf{Com}(\mathsf{ck}, \mathbf{1}_k)$ and $\mathsf{com}'_y = \mathsf{Com}(\mathsf{ck}, \boldsymbol{b})$. Naturally, this violates projective chain binding as $b_{i^*} = 0$ and hence $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_y)$. We note that it is crucial in the projective chain binding definition that the adversary is given the trapdoor td, since the reduction $\mathcal{B}$ above needs it to extract the witness $\mathtt{w}_{i^*}$.

2.3 Algebraic PCFC from Pairings

To complete our circuit-succinct algebraic BARG construction, it remains to construct an algebraic PCFC for circuits which is compatible with our compiler. Here, we focus on structural-level ideas. Later in Section 2.4, we dive deeper and provide the core ideas for concretely instantiating projective commitment keys from bilinear groups. For the formal details, we refer to Section 6.

Our construction is loosely based on two existing schemes that we introduced earlier in the overview:

1. the implicit PCFC for separable quadratic functions extracted from the Waters-Wu BARG [WW22], and
2. the projective functional commitment (PFC) scheme for circuits by Wee and Wu [WW24], which is the only existing algebraic FC for circuits with constant size proofs.

Unfortunately, these schemes are not readily available to fulfill the requirements of our compiler. A major technical challenge is to achieve our projective chain binding property in which the adversary gets access to the projection trapdoor. Notably, this property is stronger than the chain binding of [WW24] where the adversary only sees the commitment key. Indeed, the Wee and Wu's PFC construction cannot be proven secure if the adversary has the trapdoor.

Melding Waters-Wu's PCFC with Wee-Wu's PFC. The PCFC implicit in Waters-Wu's BARG [WW22] and Wee-Wu's PFC [WW24] each already satisfy some of our requirements. In particular:

[11] In our formal construction there is one more step handling the aggregation of commitments. We gloss over this detail for now.

– Waters-Wu's PCFC supports chainable functional proofs for separable quadratic functions and allows to program the commitment key with a singleton set $I = \{i^*\}$ at which the committed message is extractable (given that it belongs to the projective subspace, which in their case is the boolean set $\bar{\mathcal{M}} = \{0, 1\}$).
– Wee-Wu's PFC supports non-chainable[12] functional proofs for arbitrary circuits and allows for projecting commitments on prefix sets $I = \{1, 2, \ldots, i^*\}$. The functional proofs are built from several specialized proof sub-systems that prove linear and quadratic relations between different types of commitments, ensuring consistency over the projective space. However, the scheme is not extractable and cannot achieve projective chain binding.

Conveniently, both schemes are pairing-based and are similarly structured. Our first step is therefore to meld these two schemes in a compatible way. At a high level, we proceed as follows.

– We redesign the Waters-Wu's PCFC to embed it with a projective key that allow us to reason about functional binding in the projective subspace. In the original scheme, the size of the commitments is $\kappa + 1$ group elements for a small parameter κ (which parameterizes the κ-Lin assumption). This is sufficient for somewhere extractability, but seems insufficient to reason about functional relations in the projective subspace. We extend it to size 2κ, where, intuitively, we need κ group elements acting on the entire set $[\ell]$ for the normal-mode key and κ group elements acting only on I for the projective key. We provide more concrete details in Section 2.4.
– We structure the 2κ-sized key to enable the generation of functional proofs for arbitrary circuits, making it fully compatible with the proof sub-systems that compose the Wee-Wu's PFC.[13]

Overall, we obtain a PCFC which supports functional proofs for arbitrary circuits, projecting commitments onto arbitrary (bounded cardinality) index sets $I \subseteq [\ell]$, and allows extraction at $I \cap J$ for commitments to subvectors given by J as long as $|I \cap J| = 1$. Note that although this PCFC is correct, it does not yet satisfy the projective chain binding property. To satisfy this property, we design a novel functional proof system that partially relies on two of the specialized proof sub-systems in the Wee-Wu's PFC.

Achieving Projective Chain Binding. As mentioned above, the main technical challenge that we tackle is to achieve projective chain binding against adversaries with access to the projection trapdoor td. In more detail, in the reduction from the MDDH assumption to projective chain binding, we need to employ a "sliding window" argument, which is quite common in the BARG literature (e.g. [CJJ21, KLVW23]), where we gradually shift the hidden index set I programmed in the trapdoor td from the input wires to the output wires. However, without modifying the construction, this technique contradicts with the requirement that the adversary is given access to td, since then the adversary can notice that the hidden index set I changes.

To circumvent this issue, our construction introduces three commitment sub-keys $\mathsf{ck} = (\mathsf{ck}_0, \mathsf{ck}_1, \mathsf{ck}_2)$ with their corresponding sub-trapdoors $\mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2$. The first key ck_0 is the primary commitment key that is trapdoored to be projective at I (and which also enables somewhere extractability). ck_1 and ck_2 are functional keys that correspond to the so-called type-I and type-II commitments, respectively, which are required by the functional proof sub-systems of the Wee-Wu's PFC. The idea is to hand td_0 to the adversary, while td_1 and td_2 are only used in the security proof and never

[12] The authors discussed chainability but did not formalise the idea.
[13] In turn, this would enable us to support the functional proofs of the Wee-Wu's PFC and obtain standard CFC evaluation binding, although this is not our end goal.

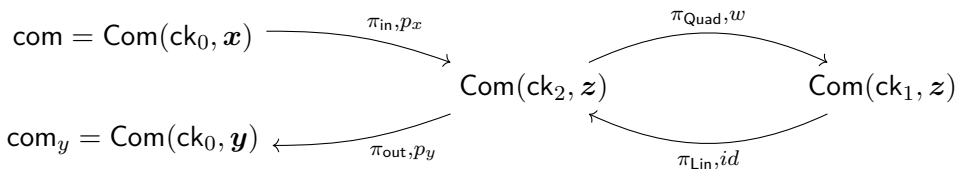


Fig. 1: Overview of the sub-proofs and commitments involved in the PCFC functional proofs.

given to the adversary. Then, we can enable the "sliding window" argument only over ck_1 and ck_2, without ever modifying ck_0. To execute this idea, we have to augment the PCFC construction to include additional consistency proofs between commitments generated against the primary key ck_0 and the functional keys $\mathsf{ck}_1, \mathsf{ck}_2$.

Constructing our Functional Proofs. In Figure 1 we give an overview of the functional proofs and commitments involved in our PCFC construction. Recall that the end goal is to prove the evaluation of a function f (interpreted as an arithmetic circuit) given an input commitment com and an output commitment com_y, both of which are committed under ck_0. For the description below, we let $\boldsymbol{z} \in \mathcal{M}^{|f|}$ be the computation trace of $f(\boldsymbol{x})$. This is, $\boldsymbol{z}$ contains the values of the $|f|$ circuit wires when f is evaluated on $\boldsymbol{x}$. For simplicity, we omit details related to f being a separable function that evaluates multiple copies of a circuit, and refer only to the single-circuit case. The proof contains four main components:

- A linear map sub-proof π_{in} that proves that a linear function s holds between $\mathsf{com} = \mathsf{Com}(\mathsf{ck}_0, \boldsymbol{x})$ and a type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$. The map s is a projection that maps the first ℓ coordinates of the input (i.e., $\boldsymbol{x}$) to the first ℓ coordinates of the output (i.e., the prefix of $\boldsymbol{z}$ corresponding to the input).
- A linear map sub-proof π_{Lin} for the identity function id between the type-I commitment $\mathsf{Com}(\mathsf{ck}_1, \boldsymbol{z})$ and the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$.
- A quadratic map sub-proof π_{Quad} for the *next wire* function w, which given the first t coordinates of $\boldsymbol{z}$ (i.e., the first t wires of $f(\boldsymbol{x})$), computes the $t+1$-th gate output wire. This is done between the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$ and the type-I commitment $\mathsf{Com}(\mathsf{ck}_1, \boldsymbol{z})$.
- Finally, a sub-proof π_{out} that proves an output-projection map p which maps the last ℓ coordinates of $\boldsymbol{z}$ to the first ℓ coordinates of $\boldsymbol{y}$, which is done between $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$ and $\mathsf{com}_y = \mathsf{Com}(\mathsf{ck}_0, \boldsymbol{y})$.

At a high level, the proof of projective chain binding goes as follows. The sub-proof π_{in} allows one to ensure matching projections between the input commitment com and the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$ on the first ℓ coordinates (i.e., $I = [\ell]$). Then, π_{Lin} and π_{Quad} enable a sliding window argument where I gradually increases from $[\ell]$ to $[|f|]$, ensuring matching projections of their respective commitments. Finally, π_{out} ensures that the projection (onto $[\ell]$) of the output commitment com_y matches the projection over the last ℓ coordinates of the type-II commitment $\mathsf{Com}(\mathsf{ck}_2, \boldsymbol{z})$.

2.4 Projective Key Structure

To conclude the presentation of the inner workings of our PCFC, we summarize the main algebraic structure of the projective commitment key. We focus on how to achieve projection and extraction on ck_0, as these are the properties that require the main structural ideas. Later, we need to augment the commitment key described below with cross-terms to enable the two main types of proofs needed

by the scheme: a proof that a committed value $\boldsymbol{x}$ is boolean, and a functional opening proof for the evaluation of f on $\boldsymbol{x}$. In this process, we also generate ck_1 and ck_2. However, we omit such details in this overview.

Let $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2, e)$ be a type-III pairing group setting, and define $\mathbb{F} := \mathbb{Z}_q$. We use the bracket additive notation of [EHK+13] for group elements: for $s \in \{1, 2, T\}$ and $x \in \mathbb{Z}_q$, $[x]_s$ denotes $g_s^x \in \mathbb{G}_s$. As we already discussed, committed vectors lie in $\mathcal{M} = \mathbb{F}$, and we restrict to $\bar{\mathcal{M}} = \{0, 1\}$ for extraction. All vectors $\boldsymbol{x} \in \mathbb{F}^\ell$ are column vectors.

Commitment Key Generation. The starting point is to sample random invertible matrices $\check{\mathbf{B}}, \hat{\mathbf{B}} \leftarrow\!\!\$\ \mathbb{F}^{2\kappa \times 2\kappa}$. These will be used to generate twin commitment keys in the source groups, where $\check{\mathbf{B}}$ generates the key in $\mathbb{G}_1$ and $\hat{\mathbf{B}}$ in $\mathbb{G}_2$. We denote the inverses of these matrices by $\check{\mathbf{B}}^*$ and $\hat{\mathbf{B}}^*$, respectively.

The next step is to split

$$\check{\mathbf{B}} = \begin{bmatrix} \check{\mathbf{B}}_1 \\ \check{\mathbf{B}}_2 \end{bmatrix}, \qquad \check{\mathbf{B}}^* = \begin{bmatrix} \check{\mathbf{B}}_1^* \ \check{\mathbf{B}}_2^* \end{bmatrix},$$

where $\check{\mathbf{B}}_1, \check{\mathbf{B}}_2 \in \mathbb{F}^{\kappa \times 2\kappa}$ and $\check{\mathbf{B}}_1^*, \check{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa \times \kappa}$. Essentially, the idea is that the 1-matrices generate an orthogonal space to the 2-matrices. The 1-matrices will correspond to the main commitment space, whereas the 2-matrices will be necessary only in projective mode, and can be ignored in normal mode. By construction, $\check{\mathbf{B}}_1 \cdot \check{\mathbf{B}}_1^* = \check{\mathbf{B}}_2 \cdot \check{\mathbf{B}}_2^* = \mathbf{I}_\kappa$ and $\check{\mathbf{B}}_1 \cdot \check{\mathbf{B}}_2^* = \check{\mathbf{B}}_2 \cdot \check{\mathbf{B}}_1^* = \mathbf{0}_\kappa$. Crucially both sets of matrices generate spaces of dimension κ to support functional evaluation. We define $\hat{\mathbf{B}}_1, \hat{\mathbf{B}}_2 \in \mathbb{F}^{\kappa \times 2\kappa}$ and $\hat{\mathbf{B}}_1^*, \hat{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa \times \kappa}$ analogously for the twin commitment key in $\mathbb{G}_2$.

Next, sampling two random, wide matrices $\check{\mathbf{S}}_1, \hat{\mathbf{S}}_1 \leftarrow\!\!\$\ \mathbb{F}^{\kappa \times \ell}$ we let

$$\check{\mathbf{T}} = \check{\mathbf{B}}_1^* \check{\mathbf{S}}_1, \qquad \hat{\mathbf{T}} = \hat{\mathbf{B}}_1^* \hat{\mathbf{S}}_1,$$

whose columns $\check{\boldsymbol{t}}_i, \hat{\boldsymbol{t}}_i \in \mathbb{F}^{2\kappa}$ for $i \in [\ell]$ serve as per-coordinate commitment generators. The (simplified) base commitment key then becomes

$$\left(\left[\check{\mathbf{T}}\right]_1, \left[\hat{\mathbf{T}}\right]_2, \left[\check{\mathbf{B}}_1^*\right]_1, \left[\hat{\mathbf{B}}_1^*\right]_2 \right).$$

Then, to commit to a vector $\boldsymbol{x} \in \mathbb{F}^\ell$ over a subset $J \subseteq [\ell]$, the committer outputs

$$[\hat{\boldsymbol{c}}]_2 = \sum_{i \in J} \left[\hat{\boldsymbol{t}}_i\right]_2 \cdot x_i,$$

and the same can be done in $\mathbb{G}_1$ using $\left[\check{\boldsymbol{t}}_i\right]_1$. Having one commitment in $\mathbb{G}_1$ and one in $\mathbb{G}_2$ turns out very useful for proving quadratic relations later on, such as $x_i \in \{0, 1\}$. Note that commitments are linear in the selected coordinates and naturally support arbitrary subvector sets J.

Generating the Key in Projective Mode. As we discussed, the normal mode does not use the 2-matrices yet. In projective mode, for a target set $I \subseteq [\ell]$, we modify $\check{\mathbf{T}}, \hat{\mathbf{T}}$ by adding components in the subspaces generated by $\check{\mathbf{B}}_2^*, \hat{\mathbf{B}}_2^*$ using fresh random matrices $\check{\mathbf{S}}_2, \hat{\mathbf{S}}_2 \leftarrow\!\!\$\ \mathbb{F}^{\kappa \times \ell}$. Intuitively, coordinates in I receive an additional hidden component in the 2-subspace, while coordinates outside I remain as before. We let

$$\check{\mathbf{T}} = \check{\mathbf{B}}_1^* \check{\mathbf{S}}_1 + \check{\mathbf{B}}_2^* \check{\mathbf{S}}_2 \mathbf{P}_I, \qquad \hat{\mathbf{T}} = \hat{\mathbf{B}}_1^* \hat{\mathbf{S}}_1 + \hat{\mathbf{B}}_2^* \hat{\mathbf{S}}_2 \mathbf{P}_I,$$

where $\mathbf{P}_I$ is a binary, diagonal *projection matrix* that selects the coordinates in I, that is, $\mathbf{P}_{Ii,i} = 1$ if $i \in I$, and is zero elsewhere.

The trapdoor then becomes $\mathsf{td} = (I, \check{\mathbf{B}}_2, \check{\mathbf{S}}_2, \hat{\mathbf{B}}_2, \hat{\mathbf{S}}_2)$.

Projecting a Commitment. Now, given a commitment $\mathsf{com} = [\hat{\boldsymbol{c}}]_2 \in \mathbb{G}_2^{2\kappa}$, we can easily project it using the trapdoor by computing

$$\left[\hat{\boldsymbol{d}}\right]_2 = \hat{\mathbf{B}}_2 \cdot [\hat{\boldsymbol{c}}]_2 \,.$$

Under projective setup, $\left[\hat{\boldsymbol{d}}\right]_2 \in \mathbb{G}_2^{\kappa}$ is a commitment to the coordinates in $I \cap J$ of $\boldsymbol{x}$ in a basis defined by $\hat{\mathbf{S}}_2$. To see this, note that in trapdoor mode

$$\begin{aligned}
\hat{\boldsymbol{d}} &= \hat{\mathbf{B}}_2 \cdot \hat{\boldsymbol{c}} \\
&= \hat{\mathbf{B}}_2 \cdot \sum_{i \in J} \hat{\boldsymbol{t}}_i \cdot x_i \\
&= \hat{\mathbf{B}}_2 \cdot \sum_{i \in J} \left(\hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i} \cdot \delta(I,i) \right) \cdot x_i \\
&= \sum_{i \in J} \hat{\boldsymbol{s}}_{2,i} \cdot \delta(I,i) \cdot x_i \\
&= \sum_{i \in I \cap J} \hat{\boldsymbol{s}}_{2,i} \cdot x_i
\end{aligned}$$

where $\delta(I,i) = 1$ if $i \in I$ and zero otherwise, and where we have used the orthogonality properties $\hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_1^* = \mathbf{0}_\kappa$ and $\hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_2^* = \mathbf{I}_\kappa$.

Thus, extraction simply enumerates $\tilde{\boldsymbol{x}} \in \{0,1\}^{I \cap J}$ and checks whether

$$\left[\hat{\boldsymbol{d}}\right]_2 = \sum_{i \in I \cap J} [\hat{\boldsymbol{s}}_{2,i}]_2 \cdot \tilde{x}_i .$$

This takes time $\mathcal{O}\left(2^{|I \cap J|} \cdot \mathsf{poly}(\lambda)\right)$. Hence, if the committed input is boolean, extraction is efficient when $|I \cap J|$ is small.

2.5 Discussion and Future Work

The techniques introduced above lead to a construction of PCFC over pairing groups that we summarize in the following (simplified) theorem:

Theorem 2 (simplified). *Assuming the Ker-DH assumption and the MDDH assumption over bilinear groups, the construction* PCFC *satisfies:*

- *Somewhere extractability for a message space $\bar{\mathcal{M}} = \{0,1\}$ if the subvector set J and the projective set I satisfy $|I \cap J| = 1$.*
- *Projective chain binding for the class of functions $\mathcal{F}_{k,\ell_\mathcal{C}}$ which evaluate up to k parallel circuits $\mathcal{C}_i$, each of size at most $\ell_\mathcal{C}$.*
- *The commitment key size and the worst-case prover running time are $\mathcal{O}(\lambda \cdot k^5 \cdot \ell_\mathcal{C}^5)$.*
- *Commitments and proofs have size $\mathcal{O}(\lambda)$.*
- *Efficient verification runs in time $\mathcal{O}(\lambda)$.*

Overall, by combining the PCFC with the circuit-succinct compiler, we obtain a fully algebraic construction of BARGs from bilinear pairings, which we summarize as follows:

Corollary 1. *Under the assumptions of Theorem 2, there exists an algebraic somewhere extractable BARG for NP such that:*

– *The setup size* $|\mathsf{crs}|$ *and the worst-case prover running time are* $\mathcal{O}(\lambda \cdot k^5 \cdot \ell_{\mathcal{C}}^5)$.
– *The proof size is* $|\pi| = \mathcal{O}(\lambda \cdot |\mathtt{w}|\cdot)$.
– *Efficient verification runs in time* $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$.

Our BARG construction is almost concretely efficient, in the sense that all complexity measures are explicitly expressible. In particular, the resulting online verification time and proof size are $\mathcal{O}(\lambda \cdot |\mathtt{w}|)$ with small constants. The main efficiency bottleneck that remains is the size of the commitment key, which also impacts the prover time. We therefore see shrinking the commitment key size as one of the next immediate future directions.

Besides providing a formal connection between FCs and BARGs, we highlight that our BARG construction is the first *black-box* construction from another cryptographic primitive with falsifiable security notions and standard-model realizations. We believe this connection enhances the understanding of BARGs and could result in new, more efficient BARG realizations. For example, currently, there is no direct algebraic construction of BARGs from lattices, although there are direct and algebraic lattice-based FCs [ACL+22, WW23b, dCP23, BCFL23, WW23a], in part due to the difficulty of achieving somewhere extractability in the lattice setting. Nevertheless, we believe that our approach provides valuable insights and a candidate construction blueprint towards this goal.

3 Preliminaries

We denote by $\mathbb{N}$ the set of natural numbers > 0. We denote the security parameter by $\lambda \in \mathbb{N}$. We call a function ϵ *negligible*, denoted $\epsilon(\lambda) = \mathsf{negl}(\lambda)$, if $\epsilon(\lambda) = O(\lambda^{-c})$ for every constant $c > 0$, and call a function $p(\lambda)$ *polynomial*, denoted poly, if $p(\lambda) = O(\lambda^c)$ for some constant $c > 0$. We say that an algorithm is *probabilistic polynomial time* (PPT) if it consumes randomness and its running time is bounded by some $p(\lambda) = \mathsf{poly}(\lambda)$. For a finite set S, $x \leftarrow\!\!\$\ S$ denotes sampling x uniformly at random in S. For an algorithm A, we write $y \leftarrow A(x)$ for the output of A on input x. For a positive $n \in \mathbb{N}$, $[n]$ is the set $\{1, \ldots, n\}$. We denote vectors $\boldsymbol{x}$ and matrices $\mathbf{M}$ using bold fonts. We write $\mathbf{0}_k$ and $\mathbf{1}_k$ for the vectors of k zeros and ones respectively. Given two strings x, y, we denote their concatenation by $x|y$. We use $\mathcal{M}$ to denote the message space, and sometimes $\bar{\mathcal{M}} \subseteq \mathcal{M}$ for some message subspace, for all cryptographic primitives considered. We denote the message space of dimension ℓ by $\mathcal{M}^\ell$. To denote a (sub)vector with coordinates in a position subset $J \subseteq [\ell]$, we write $\boldsymbol{x}_J \in \mathcal{M}^J$.

3.1 Circuit model

Throughout this work, we model a circuit $\mathcal{C} : \mathcal{M}^{\ell_{\mathtt{x},\mathtt{w}}} \to \mathcal{M}$, where $\ell_{\mathtt{x},\mathtt{w}} = \ell_{\mathtt{x}} + \ell_{\mathtt{w}}$, to take as input a statement $\mathtt{x} \in \mathcal{M}^{\ell_{\mathtt{x}}}$ and a witness $\mathtt{w} \in \mathcal{M}^{\ell_{\mathtt{w}}}$, for some message space $\mathcal{M}$ of cardinality $|\mathcal{M}| \leq 2^{\mathsf{poly}(\lambda)}$. A circuit can be arithmetic (e.g. $\mathcal{M} = \mathbb{Z}_q$) or Boolean (i.e. $\mathcal{M} = \{0, 1\}$). A circuit supports arbitrary fan-in and fan-out arithmetic (resp. Boolean) gates over $\mathcal{M}$. We adopt the convention that $1 \in \mathcal{M}$ and we say that an input $(\mathtt{x}, \mathtt{w})$ is accepted by $\mathcal{C}$ if $\mathcal{C}(\mathtt{x}, \mathtt{w}) = 1$.

We measure the size of a circuit by its total number of wires $\ell_{\mathcal{C}}$, which includes the statement and witness input wires, as well as the output wire. We associate each wire to the output of a gate $g \in [\ell_{\mathcal{C}}]$. For notational convenience, and such that the number of gates and that of wires are both $\ell_{\mathcal{C}}$, we consider that all inputs come from an "input gate" $g \in [\ell_{\mathtt{x},\mathtt{w}}]$ which simply outputs the hardwired g-th input. Given a circuit computation $\mathcal{C}(\mathtt{x}, \mathtt{w})$, the value of the g-th wire (or the output of the g-th gate) is denoted by x_g. Each gate $g \in [\ell_{\mathcal{C}}]$ computes a function $\chi_g : \mathcal{M}^{|T_g|} \to \mathcal{M}$ on an

arbitrary set of input *wires* denoted by T_g. Each gate has a single output wire, even if this can be input to multiple other gates. Finally, note that χ_g is a quadratic map, as it represents a single gate computation on a Boolean or arithmetic circuit.

To be descriptive, we use $\mathcal{G} = [\ell_{\mathcal{C}}] = \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}} \cup \mathcal{G}_{\mathsf{out}}$ to denote the set of all gates/wires partitioned into statement, witness, internal and output wires respectively. More precisely, we enumerate the circuit gates and wires as follows:

- $\mathcal{G}_{\mathsf{stmt}} := \{1, \ldots, \ell_{\mathtt{x}}\}$ are the input wires corresponding to the values in $\mathtt{x}$.
- $\mathcal{G}_{\mathsf{wit}} := \{\ell_{\mathtt{x}} + 1, \ldots, \ell_{\mathtt{x},\mathtt{w}}\}$ are the input wires corresponding to the values in $\mathtt{w}$.
- $\mathcal{G}_{\mathsf{out}} := \{\ell_{\mathtt{x},\mathtt{w}} + 1, \ldots, \ell_{\mathcal{C}}\}$ correspond to the output wires of all (non-input) gates. We arrange these gates that the g-th gate only takes input wires corresponding to smaller indices, namely $T_g \subseteq [g-1]$.

We note that the subscripts in $\ell_{\mathcal{C}}$, $\ell_{\mathtt{x}}$, $\ell_{\mathtt{w}}$, $\mathcal{G}_{\mathsf{stmt}}$, $\mathcal{G}_{\mathsf{wit}}$, $\mathcal{G}_{\mathsf{out}}$, etc. are symbolic, e.g. $\ell_{\mathcal{C}}$ should not be thought of as a function of a circuit $\mathcal{C}$. We sometimes abuse notation and write $\ell_{\mathcal{C}} = |\mathcal{C}|$, $\ell_{\mathtt{x}} = |\mathtt{x}|$, $\ell_{\mathtt{w}} = |\mathtt{w}|$, etc. to denote the quantities.

3.2 Batch Arguments for NP

A Batch Argument for NP (BARG) is a proof system for a particular subclass of NP, which is the conjunction of k NP statements corresponding to the same NP language. More precisely, given an NP language $\mathcal{L}$, the prover attests that k statements $\mathtt{x}_1, \ldots, \mathtt{x}_k$ belong to $\mathcal{L}$. In terms of efficiency, BARGs produce proofs π whose size is sublinear in the number of instances k. As opposed to SNARGs, the size of a BARG proof is only required to be sublinear in k, but could be linear in the size of the circuit $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i)$ that decides the language given an input $\mathtt{x}_i$ and an NP witness $\mathtt{w}_i$.

We adopt the notion of (somewhere extractable) BARGs for circuit satisfiability from [CJJ22, KLVW23], where we generically consider circuits $\mathcal{C} : \mathcal{M}^{\ell_{\mathtt{x},\mathtt{w}}} \to \mathcal{M}$. Such BARGs have two security properties. The first is setup indistinguishability: one can sample a trapdoored BARG crs at some index i^*, but this should be indistinguishable from a non-trapdoored crs. The second is somewhere extractability: given a valid proof for $\mathtt{x}_1, \ldots, \mathtt{x}_k$ on a crs trapdoored at i^*, the trapdoor allows one to extract a valid witness $\mathtt{w}^*$ such that $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}^*) = 1$.

Definition 1 (BARG for NP [CJJ22, KLVW23]). *Let statement length $\ell_{\mathtt{x}} = \mathsf{poly}(\lambda)$ and witness length $\ell_{\mathtt{w}} = \mathsf{poly}(\lambda)$ be fixed. A somewhere extractable batch argument (BARG) for NP is a tuple of algorithms* $\mathsf{BARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver}, \mathsf{SetupTd}, \mathsf{Ext})$*:*

$\mathsf{Setup}(1^\lambda, 1^k, 1^{\ell_{\mathcal{C}}}) \to \mathsf{crs}$: *on input the security parameter λ, a number of instances k, a circuit size $\ell_{\mathcal{C}}$, a statement length $\ell_{\mathtt{x}}$, and a witness length $\ell_{\mathtt{w}}$, outputs a common reference string* crs*. Below, we implicitly assume that circuits $\mathcal{C} : \mathcal{M}^{\ell_{\mathtt{x},\mathtt{w}}} \to \mathcal{M}$ are of size $\ell_{\mathcal{C}}$, statements $\mathtt{x} \in \mathcal{M}^{\ell_{\mathtt{x}}}$, and witnesses $\mathtt{w} \in \mathcal{M}^{\ell_{\mathtt{w}}}$.*

$\mathsf{Prove}(\mathsf{crs}, \mathcal{C}, \{(\mathtt{x}_i, \mathtt{w}_i)\}_{i\in[k]}) \to \pi$: *on input the common reference string* crs*, a circuit $\mathcal{C}$, and a batch of input-witness pairs $\{(\mathtt{x}_i, \mathtt{w}_i)\}_{i\in[k]}$, outputs a proof π.*

$\mathsf{Ver}(\mathsf{crs}, \mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \to b$: *on input* crs*, a circuit $\mathcal{C}$ of size $\ell_{\mathcal{C}}$, a batch of statements $\{\mathtt{x}_i\}_{i\in[k]}$, and a proof π, accepts ($b = 1$) or rejects ($b = 0$).*

$\mathsf{SetupTd}(1^\lambda, 1^k, 1^{\ell_{\mathcal{C}}}, i^*) \to (\mathsf{crs}, \mathsf{td})$ *works as* Setup*, and additionally outputs a trapdoor* td *associated to the index i^* that is given as input.*

$\mathsf{Ext}(\mathsf{td}, \mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \to \mathtt{w}^*$ *On input a trapdoor* td*, a circuit $\mathcal{C}$, a batch of statements $\{\mathtt{x}_i\}_{i\in[k]}$ and a proof π, outputs a witness $\mathtt{w}^*$ corresponding to the position specified by the trapdoor.*

A BARG must satisfy the following properties:

Completeness. *For any $\lambda, k, \ell_{\mathcal{C}} \in \mathbb{N}$, any circuit $\mathcal{C}$, all statement-witness pairs $(\mathtt{x}_i, \mathtt{w}_i)_{i\in[k]}$ such that $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i) = 1$ for all $i \in [k]$, $\mathsf{crs} \in \mathsf{Setup}(1^\lambda, 1^k, 1^{\ell_{\mathcal{C}}})$ and $\pi \in \mathsf{Prove}(\mathsf{crs}, \mathcal{C}, \{(\mathtt{x}_i, \mathtt{w}_i)\}_{i\in[k]})$, it holds that $\mathsf{Ver}(\mathsf{crs}, \mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi) = 1$.*

Succinctness. *For any admissible set of parameters as before, $|\pi| \le \mathsf{poly}(\lambda, \log k, \ell_{\mathcal{C}})$. Furthermore, we say that* BARG *is* circuit-succinct *if $|\pi| \le \mathsf{poly}(\lambda, \log k, \ell_{\mathtt{w}})$.*

Setup Indistinguishability. *For any PPT adversary $\mathcal{A}$, any $k, \ell_{\mathcal{C}} = \mathsf{poly}(\lambda)$, and index $i^* \in [k]$, the distributions of crs induced by $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{SetupTd}(1^\lambda, 1^k, 1^{\ell_{\mathcal{C}}}, i^*)$ and $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^k, 1^{\ell_{\mathcal{C}}})$ are computationally indistinguishable.*

Somewhere Extractability. *For any PPT adversary $\mathcal{A}$, any $k, \ell_{\mathcal{C}} = \mathsf{poly}(\lambda)$, and index $i^* \in [k]$,*

$$\Pr\left[\begin{array}{l|l} \mathsf{Ver}(\mathsf{crs}, \mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi) = 1 & (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{SetupTd}(1^\lambda, 1^k, 1^{\ell_{\mathcal{C}}}, i^*) \\ \wedge\ \mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}^*) \neq 1 & (\mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ & \mathtt{w}^* \leftarrow \mathsf{Ext}(\mathsf{td}, \{\mathtt{x}_i\}_{i\in[k]}, \mathcal{C}, \pi) \end{array}\right] \le \mathsf{negl}(\lambda).$$

Definition 2 (Efficient Verification for BARG). *A BARG for NP* BARG *has efficient verification with preprocessing (also called split verification in [WW22]) if there exists a pair of algorithms:*

$\mathsf{PreVer}(\mathsf{crs}, \mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}) \to \mathsf{vk}$: *On input the common reference string* crs*, a circuit $\mathcal{C}$, and a batch of k statements $\{\mathtt{x}_i\}_{i\in[k]}$, it creates a succinct verification key* vk *such that $|\mathsf{vk}| \le \mathsf{poly}(\lambda, \log k, \ell_{\mathcal{C}})$*

$\mathsf{EffVer}(\mathsf{vk}, \pi) \to b$ *On input the succinct verification key* vk *and a proof π, accepts ($b = 1$) or rejects ($b = 0$) in time bounded by $\mathsf{poly}(\lambda, \log k, \ell_{\mathcal{C}})$.*

Furthermore, if for $\ell_{\mathcal{C}} \le 2^{\mathsf{poly}(\lambda)}$ the size of the verification key $|\mathsf{vk}|$ and the runtime of $\mathsf{EffVer}(\mathsf{vk}, \pi)$ are bounded by $\mathsf{poly}(\lambda, \log k, \ell_{\mathtt{w}})$, we say that BARG *has a circuit-succinct verification with preprocessing.*

Remark 1. In Definition 2 we consider a stronger notion of efficient verification where the circuit $\mathcal{C}$ can be preprocessed by PreVer. This allows us to define the circuit-succinct verifier where the verifier runtime is sublinear in $|\mathcal{C}|$. In prior definitions (e.g. [WW22]), the circuit $\mathcal{C}$ is not preprocessed by PreVer but is instead passed as an input to EffVer. In this case, it is impossible for the online verifier to have circuit-succinctness.

3.3 Bilinear Groups and Assumptions

A *bilinear group generator* $\mathcal{BG}(1^\lambda)$ is an algorithm that returns $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of prime order q, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ are fixed generators, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable, non-degenerate, bilinear map. In our work we use Type-3 groups in which it is assumed that there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. We use the bracket notation of [EHK+13] for group elements: for $s \in \{1, 2, T\}$ and $x \in \mathbb{Z}_q$, $[x]_s$ denotes $g_s^x \in \mathbb{G}_s$. We use additive notation for all groups. We note that given an element $[x]_s \in \mathbb{G}_s$, for $s = 1, 2$, and a scalar a, one can efficiently compute $a \cdot [x] = [ax] = g_s^{ax} \in \mathbb{G}_s$; given group elements $[a]_1 \in \mathbb{G}_1$ and $[b]_2 \in \mathbb{G}_2$, one can efficiently compute $[ab]_T = [a]_1 \cdot [b]_2$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$, we represent a matrix of group elements $g_s^{\mathbf{A}}$ as $[\mathbf{A}]_s \in \mathbb{G}_s^{m\times n}$.

We recall the matrix Diffie-Hellman assumption (MDDH) and the kernel Diffie-Hellman assumption (KerDH), which we adapt from [MRV16]. For MDDH we directly introduce the bilateral variant where the challenge is encoded in both $\mathbb{G}_1$ and $\mathbb{G}_2$, following the notation from [GZ21, WW24].

Definition 3 (Kernel Diffie-Hellman Assumption). *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2, e)$ *be a bilinear group setting and let* $\kappa, \ell, d \in \mathbb{N}$*. We say that the kernel Diffie-Hellman assumption* $\mathsf{KerDH}_{\kappa,\ell}$ *holds in* $\mathbb{G}_1$ *for* bgp *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathbf{A}\boldsymbol{x} = \mathbf{0} \wedge \boldsymbol{x} \neq \mathbf{0} \;\middle|\; \begin{array}{l}\mathbf{A} \leftarrow\!\$\; \mathbb{F}_q^{\kappa\times\ell} \\ [\boldsymbol{x}]_2 \leftarrow \mathcal{A}(\mathsf{bgp}, [\mathbf{A}]_1)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

where the probability is taken over the choice of $\mathbf{A} \leftarrow\!\$\; \mathbb{F}_q^{\kappa\times\ell}$ *and the adversary* $\mathcal{A}$*'s random coins. We define* $\mathsf{KerDH}_{\kappa,\ell}$ *over* $\mathbb{G}_2$ *analogously.*

Definition 4 (Matrix Diffie-Hellman Assumption). *Let* $\mathsf{bgp} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2, e)$ *be a bilinear group setting and let* $\kappa, \ell, d \in \mathbb{N}$*. We say that the (bilateral) Matrix Diffie-Hellman assumption* $\mathsf{MDDH}_{\kappa,\ell,d}$ *holds over* bgp *if for any PPT adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|\begin{array}{l}\Pr\left[\mathcal{A}(\mathsf{bgp}, [\mathbf{A}]_1, [\mathbf{A}]_2, [\mathbf{S}\cdot\mathbf{A}]_1, [\mathbf{S}\cdot\mathbf{A}]_2) = 1\right] \\ -\ \Pr\left[\mathcal{A}(\mathsf{bgp}, [\mathbf{A}]_1, [\mathbf{A}]_2, [\mathbf{U}]_1, [\mathbf{U}]_2) = 1\right]\end{array}\right| \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the choice of $\mathbf{A} \leftarrow\!\$\; \mathbb{F}_q^{\kappa\times\ell}, \mathbf{S} \leftarrow\!\$\; \mathbb{F}_q^{d\times\kappa}, \mathbf{U} \leftarrow\!\$\; \mathbb{F}_q^{d\times\ell}$ *and the adversary* $\mathcal{A}$*'s random coins.*

4 Projective Chainable Functional Commitments

We define several variants of commitments, in particular projective chainable functional commitments (PCFC), and their properties. Extending the usual syntax of commitments, we allow the commitment algorithm to additionally input an index set $J \in \mathcal{J} \subseteq 2^{[\ell]}$ and commit to a (sub)vector $\boldsymbol{x}_J \in \mathcal{M}^J$, and extend the other algorithms accordingly. The index set J is taken as $J = [\ell]$ when omitted. Although not often considered explicitly, such a functionality is naturally supported by common commitment schemes.

Throughout this work, we assume that all commitments are deterministic and that the opening information is the committed vector itself. We also assume that all algorithms taking a commitment key ck as input has random access to ck. This is to capture schemes where the commitment key consists of many components and where not all algorithms need access to all components.

Definition 5 (Commitments). *A commitment scheme for an index set family* $\mathcal{J} \subseteq 2^{[\ell]}$ *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Com})$ *with the following syntax:*

$\mathsf{Setup}(1^\lambda, 1^\ell) \to \mathsf{ck}$**:** *On input the security parameter* λ *and the vector length* ℓ*, output a commitment key* ck*.*

$\mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}) \to \mathsf{com}$**:** *On input the commitment key* ck*, an index set* $J \in \mathcal{J}$ *and a vector* $\boldsymbol{x} \in \mathcal{M}^J$*, output a commitment* com*.*

The index set J *is taken as* $J = [\ell]$ *when omitted. A commitment scheme must satisfy the following property:*

Binding. *For any PPT adversary* $\mathcal{A}$ *and vector length* $\ell = \mathsf{poly}(\lambda)$*,*

$$\Pr\left[\begin{array}{l}\mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}) = \mathsf{Com}(\mathsf{ck}, J', \boldsymbol{x}') \\ \wedge\ (J, \boldsymbol{x}) \neq (J', \boldsymbol{x}')\end{array}\;\middle|\; \begin{array}{l}\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ (\mathsf{com}, J, \boldsymbol{x}, J', \boldsymbol{x}') \leftarrow \mathcal{A}(\mathsf{ck}) \\ J, J' \in \mathcal{J} \wedge \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{M}^J\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Succinctness. *For any* $\lambda, \ell \in \mathbb{N}$, $\boldsymbol{x} \in \mathcal{M}^\ell$, *it holds that* $|\mathsf{com}| \leq \mathsf{poly}(\lambda, \log \ell)$.

We define a notion of aggregatability which allows to aggregate commitments of subvectors with disjoint index sets into a commitment of the union.

Definition 6 (Aggregatability). *Let* $\hat{\mathcal{J}} \subset \mathcal{J}$. *A commitment scheme is aggregatable if*

- *there exists a deterministic aggregation algorithm* $\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \hat{\mathcal{J}}}) \to \mathsf{com}$, *which takes as input* ck *and tuples of commitments* $(\mathsf{com}_J)_{J \in \hat{\mathcal{J}}}$ *and outputs a commitment* com *in time at most* $\mathsf{poly}(\lambda, \log \ell) \cdot |\hat{\mathcal{J}}|$, *and*
- *if* $J \cap J' = \emptyset$ *for all distinct* $J, J' \in \hat{\mathcal{J}}$, *then for any* $\boldsymbol{x}_J \in \mathcal{M}^J$ *for* $J \in \hat{\mathcal{J}}$, *the following property is satisfied:*

$$\Pr\left[\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \hat{\mathcal{J}}}) = \mathsf{com} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \mathsf{com}_J \leftarrow \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}_J) \quad \forall J \in \hat{\mathcal{J}} \\ \mathsf{com} \leftarrow \mathsf{Com}\left(\mathsf{ck}, \bigcup_{J \in \hat{\mathcal{J}}} J, \bigcup_{J \in \hat{\mathcal{J}}} \boldsymbol{x}_J\right) \end{array}\right] = 1.$$

Remark 2. Our aggregatability definition (Definition 6) is quite strong as it makes sense only for deterministic commitments. We believe that the definition can be relaxed, but made more complicated, to allow for randomized commitments.

4.1 Chainable Functional Commitments (CFC)

A Functional Commitment (FC) [LRY16] is a powerful primitive that allows an entity to first commit to some input $\boldsymbol{x} \in \mathcal{M}^\ell$ (more generally $\boldsymbol{x} \in \mathcal{M}^J$ for some $J \in \mathcal{J}$) and then open the commitment to $f(\boldsymbol{x})$ for some admissible function $f \in \mathcal{F}$. Both the commitment com and the opening proof π are succinct.

Some FC schemes may offer useful composability properties such as *chainability*. A chainable functional commitment (CFC) [BCFL23] is an extension of FC that allows one to commit to multiple inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathcal{M}^\ell$ and then open to *a commitment of* $\boldsymbol{y} = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$, i.e. the output $\boldsymbol{y}$ remains committed. A CFC generically implies an FC by simply adding an opening proof that the output commitment indeed opens to $\boldsymbol{y}$. Below, we follow and extend the syntax from [BCFL23] to allow each of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y}$ to be indexed by a different index set.

Definition 7 (Chainable Functional Commitments (CFC)). *A chainable functional commitment (CFC) for a class of functions*[14] $\mathcal{F} = (\mathcal{F}_\ell)_\ell$ *and index set family* $\mathcal{J} = (\mathcal{J}_\ell)_\ell$ *consists of a commitment scheme* $(\mathsf{Setup}, \mathsf{Com})$ *for* $\mathcal{J}$ *and additionally PPT algorithms* $(\mathsf{FuncProve}, \mathsf{FuncVer})$ *with the following syntax:*

- $\mathsf{FuncProve}(\mathsf{ck}, (\boldsymbol{x}_i)_{i \in [m]}, f) \to \pi$**:** *given vectors* $\boldsymbol{x}_i \in \mathcal{M}^{J_i}$ *for* $i \in [m]$ *and a function* $f \in \mathcal{F}$ *where* $f : \prod_{i \in [m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$, *returns an opening proof* π.
- $\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i \in [m]}, \mathsf{com}_y, f, \pi) \to b \in \{0, 1\}$**:** *on input commitments* $(\mathsf{com}_i)_{i \in [m]}$ *for* $i \in [m]$ *to the* m *inputs and* com_y *to the output, opening proof* π, *and function* $f \in \mathcal{F}$ *where* $f : \prod_{i \in [m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$, *accepts* $(b = 1)$ *or rejects* $(b = 0)$.

A CFC must satisfy the following properties.

[14] We often omit the subscript ℓ of $\mathcal{F}_\ell$ and treat the length parameter implicitly.

Correctness. *For $\lambda, \ell, m, \in \mathbb{N}$, $J_1, \ldots, J_m, J_y \in \mathcal{J}$, $f \in \mathcal{F}$ where $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$, and $\boldsymbol{x}_i \in \mathcal{M}^{J_i}$ for $i \in [m]$, it holds that*

$$\Pr\left[\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \mathsf{com}_i \leftarrow \mathsf{Com}(\mathsf{ck}, J_i, \boldsymbol{x}_i) \quad \forall i \in [m] \\ \mathsf{com}_y \leftarrow \mathsf{Com}(\mathsf{ck}, J_y, f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)) \\ \pi \leftarrow \mathsf{FuncProve}(\mathsf{ck}, (\boldsymbol{x}_i)_{i\in[m]}, f) \end{array}\right] = 1.$$

Succinctness. *For any admissible set of parameters as before, $|\pi| \leq \mathsf{poly}(\lambda, \log \ell, \log m, \log |f|)$, where $|f|$ denotes the size of the circuit description of f.*

The security notion of CFC considered in [BCFL23] is evaluation binding, recalled for completeness in Definition 8. In this work, however, we will not rely on evaluation binding but other properties to be defined throughout this section.

Definition 8 (CFC Evaluation Binding). *A CFC satisfies evaluation binding if for any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l} \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 \\ \wedge\ \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}'_y, f, \pi') = 1 \\ \wedge\ \mathsf{com}_y \neq \mathsf{com}'_y \end{array} \;\middle|\; \begin{array}{l} \mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \begin{pmatrix} (\mathsf{com}_i)_{i\in[m]}, & f, \\ \mathsf{com}_y, & \pi, \\ \mathsf{com}'_y, & \pi' \end{pmatrix} \leftarrow \mathcal{A}(\mathsf{ck}) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 9 (CFC Efficient Verification). *A CFC admits efficient verification with preprocessing if there exists a pair of algorithms:*

$\mathsf{PreFuncVer}(\mathsf{ck}, f) \to \mathsf{ck}_f$ *on input the commitment key* ck *and a function $f \in \mathcal{F}$ where $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$, outputs a function key* ck_f *of size $|\mathsf{ck}_f| \leq \mathsf{poly}(\lambda, \log \ell, \log m, \log |f|)$.*

$\mathsf{EffFuncVer}(\mathsf{ck}_f, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \pi) \to b \in \{0, 1\}$ *on input a function key* ck_f*, commitments* $(\mathsf{com}_i)_{i\in[m]}$ *to the m inputs and* com_y *to the output, and an opening proof π, accepts ($b = 1$) or rejects ($b = 0$) in time bounded by* $\mathsf{poly}(\lambda, \log \ell, \log m, \log |f|)$.

Furthermore, the following function equivalence holds:

$$\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) \equiv \mathsf{EffFuncVer}(\mathsf{PreFuncVer}(\mathsf{ck}, f), (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \pi).$$

4.2 Projective Commitments (PC)

In [WW24], Wee and Wu introduce the notion of projective commitments. We generalise their notion to allow for arbitrary projective subspaces, but not only prefixes, along with the flexibility of committing to subvectors. We adopt the notation that $\mathcal{I}$ is a family of admissible index sets $I \in \mathcal{I} \subseteq 2^{[\ell]}$. We write $\boldsymbol{x}_I$ to refer to a subvector of $\mathcal{M}^\ell$ that contains only the coordinates in I, namely $\boldsymbol{x}_I \in \mathcal{M}^I$.

Definition 10 (Projective Commitments (PC)). *Let $\mathcal{I}, \mathcal{J}$ be families of index sets parametrised by $\ell \in \mathbb{N}$. A projective commitment (PC) scheme for $(\mathcal{I}, \mathcal{J})$ consists of a commitment scheme* $(\mathsf{Setup}, \mathsf{Com})$ *for $\mathcal{J}$ and additional PPT algorithms* $(\mathsf{ProjSetup}, \mathsf{ProjCom}, \mathsf{Proj})$ *with the following syntax:*

$\mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \to (\mathsf{ck}, \mathsf{td})$**:** *On input the security parameter λ, vector length ℓ, and index set $I \in \mathcal{I}$, output a commitment key* ck *and a trapdoor* td*.*

$\mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}) \to \mathsf{pcom}$**:** *On input the trapdoor* td*, an index set $J \in \mathcal{J}$ and a vector $\boldsymbol{x} \in \mathcal{M}^{I \cap J}$, output a projective commitment* pcom*.*

$\mathsf{Proj}(\mathsf{td}, \mathsf{com}) \to \mathsf{pcom}$**:** *On input the trapdoor* td *and a commitment* com*, output a projective commitment* pcom*.*[15]

The index set J is taken as $J = [\ell]$ when omitted. A PC must satisfy the following properties:

Projection Correctness. *For any index sets $I \in \mathcal{I}$, $J \in \mathcal{J}$, any vector $\boldsymbol{x} \in \mathcal{M}^{I \cap J}$, and any $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$, then $\Pr[\mathsf{Proj}(\mathsf{td}, \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})) = \mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x})] = 1$.*

Setup Indistinguishability. *For any PPT adversary $\mathcal{A}$, and subspace $I \in \mathcal{I}$, the distributions of* ck *induced by $(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$ and $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$ are computationally indistiguishable.*

Extending the notion of aggregatability, we define the notion of projective aggregatability which states that not only ordinary commitments but also projective commitments are aggregatable.

Definition 11 (PC Projective Aggregatability). *A PC* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{ProjSetup}, \mathsf{ProjCom}, \mathsf{Proj})$ *for $(\mathcal{I}, \mathcal{J})$ is projective aggregatable if, given $\hat{\mathcal{J}} \subset \mathcal{J}$,*

- $(\mathsf{Setup}, \mathsf{Com})$ *is aggregatable with an aggregation algorithm* Agg*,*
- *there exists a projective aggregation algorithm* $\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}) \to \mathsf{pcom}$ *inputting the commitment key* ck *and tuples of projective commitments* $(\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}$ *and outputs a projective commitment* pcom*,*
- *if $J \cap J' = \emptyset$ for all distinct $J, J' \in \hat{\mathcal{J}}$, then for any $I \in \mathcal{I}$ and any $\boldsymbol{x}_J \in \mathcal{M}^{I \cap J}$ for $J \in \hat{\mathcal{J}}$, the following property is satisfied:*

$$\Pr\left[\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}) = \mathsf{pcom} \;\middle|\; \begin{array}{l} (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ \mathsf{pcom}_J \leftarrow \mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}_J) \quad \forall J \in \hat{\mathcal{J}} \\ \mathsf{pcom} \leftarrow \mathsf{ProjCom}\left(\mathsf{ck}, \bigcup_{J \in \hat{\mathcal{J}}} J, \bigcup_{J \in \hat{\mathcal{J}}} \boldsymbol{x}_J\right) \end{array}\right] = 1.$$

Remark 3 (Projection commutativity). Projection correctness (Definition 10) and projective aggregatability (Definition 11), which implies aggregatability (Definition 6), together imply "projection commutativity". In more detail, suppose

$$\mathsf{com}_J = \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}_J), \qquad \mathsf{pcom}_J = \mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}_J),$$
$$\mathsf{com} = \mathsf{Com}\left(\mathsf{ck}, \bigcup_{J \in \hat{\mathcal{J}}} J, \bigcup_{J \in \hat{\mathcal{J}}} \boldsymbol{x}_J\right), \quad \text{and} \quad \mathsf{pcom} = \mathsf{ProjCom}\left(\mathsf{td}, \bigcup_{J \in \hat{\mathcal{J}}} J, \bigcup_{J \in \hat{\mathcal{J}}} \boldsymbol{x}_J\right).$$

The above three properties imply the following:

- projection correctness: $\mathsf{Proj}(\mathsf{td}, \mathsf{com}_J) = \mathsf{pcom}_J$ and $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{pcom}$.
- aggregatability: $\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \hat{\mathcal{J}}}) = \mathsf{com}$.
- projective aggregatability: $\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \hat{\mathcal{J}}}) = \mathsf{pcom}$.

[15] Note that we define Proj not to input the (alleged) index set J of the commitment com. This leads to stronger properties related to projection. Clearly, the syntax and properties can be relaxed to allow Proj input J.

Putting the above together, we obtain

$$\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{Proj}(\mathsf{td}, \mathsf{com}_J))_{J\in\hat{\mathcal{J}}}) = \mathsf{Proj}(\mathsf{td}, \mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J\in\hat{\mathcal{J}}})).$$

For certain message subspaces $\bar{\mathcal{M}} \subset \mathcal{M}$ and vectors $\boldsymbol{x} \in \bar{\mathcal{M}}^\ell$, it might be possible to augment a projective commitment scheme with an embedded proof system that provides somewhere extractability. It means that, if the commitment key is extractable at set I and an adversary provides a commitment com and a proof π that verifies, then one can extract a subvector $\boldsymbol{x}_I \in \bar{\mathcal{M}}^I$ such that $\mathsf{Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{ProjCom}(\mathsf{td}, \boldsymbol{x}_I)$ except with negligible probability.

To illustrate this property, consider the example of a commitment scheme such that the message space $\mathcal{M} = \mathbb{F}_q$ for a prime $q > 2$. Moreover, the commitment supports an embedded "booleanity" proof that ensures that a committed value $\boldsymbol{x}$ satisfies $x_i(x_i - 1) = 0 \ \forall i \in [\ell]$, namely that $x_i \in \{0, 1\} = \bar{\mathcal{M}}$. Then, our somewhere extractability notion ensures that we can extract a vector $\boldsymbol{x}_I \in \{0, 1\}^I$.

Definition 12 (PC Somewhere Extractability). *Let $\mathcal{I}, \mathcal{J}$ be index sets families parametrised by $\ell \in \mathbb{N}$ and $\bar{\mathcal{M}} \subset \mathcal{M}$ be a message subspace. A somewhere extractable PC for $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$ consists of a PC for $(\mathcal{I}, \mathcal{J})$ and additional PPT algorithms $(\mathsf{ProjProve}, \mathsf{ProjVer}, \mathsf{ProjExt})$ with the following syntax:*

$\mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x}) \to \pi$**:** *On input the commitment key ck, an index set $J \in \mathcal{J}$ and a vector $\boldsymbol{x} \in \mathcal{M}^J$, output a subspace proof π.*

$\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) \to 0/1$**:** *On input the commitment key ck, an index set $J \in \mathcal{J}$, a commitment com and a subspace proof π, accepts or rejects.*

$\mathsf{ProjExt}(\mathsf{td}, J, \mathsf{com}) \to \boldsymbol{x}$**:** *On input the trapdoor td, an index set $J \in \mathcal{J}$ and a commitment com, outputs a value $\boldsymbol{x} \in \bar{\mathcal{M}}^{I\cap J} \cup \{\perp\}$.*

Moreover, these algorithms must satisfy the following properties:

Succinctness. *For any honestly generated commitment key ck, index set $J \in \mathcal{J}$ and vector $\boldsymbol{x} \in \mathcal{M}^J$, a projection proof $\pi \in \mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x})$ satisfies $|\pi| \le \mathsf{poly}(\lambda, \log \ell, \log |J|)$.*

Somewhere Completeness. *For any index sets $I \in \mathcal{I}$ and $J \in \mathcal{J}$ and any vector $\boldsymbol{x} \in \mathcal{M}^J$:*

$$\Pr\left[\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) = 1 \;\middle|\; \begin{array}{l}(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)\\ \mathsf{com} \leftarrow \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})\\ \pi \leftarrow \mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x})\end{array}\right] = 1.$$

Somewhere Extractability. *For any PPT*[16] *adversary $\mathcal{A}$ and index set $I \in \mathcal{I}$, the advantage $\mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{A}}(\lambda)$ defined below is $\mathsf{negl}(\lambda)$:*

$$\Pr\left[\begin{array}{l}\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) = 1\\ \wedge\ \left(\mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x}) \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}) \vee\ \boldsymbol{x} \notin \bar{\mathcal{M}}^{I\cap J}\right)\end{array} \;\middle|\; \begin{array}{l}(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)\\ (J, \mathsf{com}, \pi) \leftarrow \mathcal{A}(\mathsf{ck})\\ \boldsymbol{x} \leftarrow \mathsf{ProjExt}(\mathsf{td}, J, \mathsf{com})\\ J \in \mathcal{J}\end{array}\right].$$

[16] The property may also hold statistically against unbounded adversaries.

Remark 4 (Somewhere extractability for aggregated commitments.). Naturally, the above definition of somewhere extractability (Definition 12) can be extended to aggregated commitments. If $\mathsf{com} = \mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J\in\hat{\mathcal{J}}})$ for some non-overlapping family $\hat{\mathcal{J}} \subset \mathcal{J}$, then it follows that $\mathsf{ProjExt}(\mathsf{td}, \bigcup_{J'\in\hat{\mathcal{J}}} J, \mathsf{com}) = \bigcup_{J\in\hat{\mathcal{J}}} \mathsf{ProjExt}(\mathsf{td}, J', \mathsf{com}_J)$. This may enable extraction on aggregated commitments com where running the extractor $\mathsf{ProjExt}$ on com directly is not possible, such as when $I \cap J$ is too large.

Definition 13 (PC Efficient Verification). *A somewhere extractable PC admits efficient verification with preprocessing if there exists a pair of algorithms:*

$\mathsf{PreProjVer}(\mathsf{ck}, J) \to \mathsf{ck}_J$ *on input the commitment key* ck *and an index set* $J \in \mathcal{J}$*, outputs a projective key* ck_J *of size* $|\mathsf{ck}_J| \leq \mathsf{poly}(\lambda, \log \ell, \log |J|)$.

$\mathsf{EffProjVer}(\mathsf{ck}_J, \mathsf{com}, \pi) \to b \in \{0,1\}$ *on input a projective key* ck_J*, a commitment* com*, and a projection proof* π*, accepts* $(b = 1)$ *or rejects* $(b = 0)$ *in time bounded by* $\mathsf{poly}(\lambda, \log \ell, \log |J|)$.

Furthermore, the following function equivalence holds:

$$\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi) \equiv \mathsf{EffProjVer}(\mathsf{PreProjVer}(\mathsf{ck}, J), \mathsf{com}, \pi).$$

4.3 Projective Chainable Functional Commitments (PCFC)

We consider projective commitments that also admit the syntax of chainable functional commitments (Definition 7). We call these commitments *projective chainable functional commitments* (PCFC).

Definition 14 (Projective Chainable Functional Commitment (PCFC)). *A (somewhere extractable) projective CFC ((se-)PCFC) for a class of functions* $\mathcal{F}$*, families of admissible index sets* $(\mathcal{I}, \mathcal{J})$ *(and a message subspace* $\bar{\mathcal{M}}$*) is a tuple of PPT algorithms consisting of some shared* $\mathsf{Setup}, \mathsf{Com}$ *and*

$$(\overbrace{\underbrace{\mathsf{ProjSetup}, \mathsf{ProjCom}, \mathsf{Proj}}_{\text{PC for } (\mathcal{I},\mathcal{J})}, (\mathsf{ProjProve}, \mathsf{ProjVer}, \mathsf{ProjExt})}^{\text{se-PC for } (\mathcal{I},\mathcal{J},\bar{\mathcal{M}})}, \underbrace{\mathsf{FuncProve}, \mathsf{FuncVer}}_{\text{CFC for } (\mathcal{F},\mathcal{J})}).$$

Definition 15 (PCFC Efficient Verification). *A PCFC admits efficient verification with preprocessing if the underlying CFC and PC admit efficient verification with preprocessing, i.e. there exists additional algorithms* $(\mathsf{PreFuncVer}, \mathsf{EffFuncVer}, \mathsf{PreProjVer}, \mathsf{EffProjVer})$ *with properties defined in Definitions 9 and 13.*

We introduce the two main security notions for PCFC: functional extractability and projective chain binding. To define these notions, we first introduce separable functions.

Definition 16 (Separable Function). *Let* $I_1, \ldots, I_\ell, I_y \subseteq [\ell]$*. We say that a function* $f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y}$ *is* $I = (I_1, \ldots, I_\ell, I_y)$*-separable whenever the coordinates in* $I_y \cap J_y$ *of the output vector only depend on the coordinates in* $I_i \cap J_i$ *of each of the* i*-th input vectors. The restriction of* f *to* I *is denoted by* $f_I : \prod_{i\in[m]} \mathcal{M}^{I_i\cap J_i} \to \mathcal{M}^{I_y\cap J_y}$*. For a class of functions* $\mathcal{F}$*, we write* $\mathcal{F}_I$ *for the set of functions* $f \in \mathcal{F}$ *that are* I*-separable. If* f *is* I*-separable for all* $I \in \mathcal{I}^{\ell+1}$ *for some set* $\mathcal{I}$*, we say that* f *is* $\mathcal{I}$*-separable. If* $I_1 = \ldots = I_\ell = I_y$*, we abuse notation and write* $I = I_1 = \ldots = I_\ell = I_y$.

Example 1. The function $f : \mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{Z}^2$ given by $f((x_1, x_2), (y_1, y_2)) = (x_1 + y_1, x_1 + x_2 + y_2)$ is $\mathcal{I}$-separable for $\mathcal{I} = \{\{1\}, \{1, 2\}\}$.

Intuitively, a PCFC is functional extractable if, given a CFC opening π for a separable function f, then the extracted values from the commitments are guaranteed to evaluate f_I correctly.

Definition 17 (PCFC Functional Extractability). *A somewhere extractable PCFC satisfies functional somewhere extractability for a class of $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ if for any PPT adversary $\mathcal{A}$ and set $I \in \mathcal{I}$, the advantage $\mathsf{Adv}^{\mathrm{fe}}_{\mathsf{PCFC},\mathcal{A}}(\lambda)$ defined below is $\mathsf{negl}(\lambda)$:*

$$\Pr\left[\begin{array}{l|l} \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 & (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ \wedge\ f_I(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) \neq \boldsymbol{y} & (f, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, \pi) \leftarrow \mathcal{A}(\mathsf{ck}) \\ \wedge\ f \in \mathcal{F}_I & f : \prod_{i\in[m]} \mathcal{M}^{J_i} \to \mathcal{M}^{J_y} \\ & \boldsymbol{x}_i \leftarrow \mathsf{ProjExt}(\mathsf{td}, J_i, \mathsf{com}_i) \quad \forall i \in [m] \\ & \boldsymbol{y} \leftarrow \mathsf{ProjExt}(\mathsf{td}, J_y, \mathsf{com}_y) \\ & \boldsymbol{x}_i \in \mathcal{M}^{I\cap J_i} \quad \forall i \in [m] \\ & \boldsymbol{y} \in \mathcal{M}^{I\cap J_y} \end{array}\right]$$

Remark 5. Due to the somewhere extractability of PC (Definition 12), an alternative way of defining Functional Extractability for PCFC is to let the adversary output $\boldsymbol{x}_i$ and $\boldsymbol{y}$, and to let the experiment run Proj instead of $\mathsf{ProjExt}$.

Finally, we introduce projective chain binding for PCFC which generalises evaluation binding (Definition 8). In particular, if a PCFC for a class of $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ is projective chain binding, where $\mathcal{I}$ contains $[\ell]$, then the PCFC is also evaluation binding for $\mathcal{F}$.

Definition 18 (PCFC Projective Chain Binding). *A PCFC for a class of $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ satisfies projective chain binding if for any PPT adversary $\mathcal{A}$ and set $I \in \mathcal{I}$, the advantage $\mathsf{Adv}^{\mathrm{pcb}}_{\mathsf{PCFC},\mathcal{A}}(\lambda)$ defined below is $\mathsf{negl}(\lambda)$.*

$$\Pr\left[\begin{array}{l|l} \mathsf{Proj}(\mathsf{td}, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_y) & (\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I) \\ \wedge\ f \in \mathcal{F}_I & (f, (\mathsf{com}_i, \mathsf{com}'_i)_{i\in[m]}, \mathsf{com}_y, \mathsf{com}'_y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}) \\ & \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in[m]}, \mathsf{com}_y, f, \pi) = 1 \\ & \mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}'_i)_{i\in[m]}, \mathsf{com}'_y, f, \pi') = 1 \\ & \mathsf{Proj}(\mathsf{td}, \mathsf{com}_i) = \mathsf{Proj}(\mathsf{td}, \mathsf{com}'_i) \quad \forall i \in [m] \end{array}\right].$$

Note that the adversary not only gets access to the commitment key ck, but also to the projection trapdoor td. One may consider a weaker variant where td is not given to the adversary.

On functional extractability, projective chain binding, and partial input soundness. Partial input soundness [KLVW23] is a security notion that arises in the context of RAM SNARGs, more specifically in *flexible RAM SNARGs*. A flexible RAM SNARG with partial input soundness intuitively guarantees soundness if some part of the RAM memory can be extracted from the hash

of the memory tape, as long as the RAM computation depends only on the part that was extracted. The notion is defined with respect to a somewhere extractable hash function that is programmed on a memory part I. Then, partial input soundness is stated roughly as follows: given a somewhere extractable hash key programmed on I, it is hard for any adversary to output a valid SNARG proof π and a hash digest rt such that the RAM machine $\mathcal{R}$ reads only I and the extracted $\boldsymbol{x}_I$ satisfies that $\mathcal{R}(\boldsymbol{x}_I) = 0$.

Functional extractability (Definition 17) and partial input soundness are essentially the same property in different environments, as both notions refer to local computations where a part of the input can be extracted. Their main differences are the following two. First, functional extractability is not defined with respect to any external building block such as a somewhere extractable hash function, as it is an intrinsic property of PCFC schemes. Second, the property is defined for separable functions over circuits, instead of RAM machines that read only a part of their memory tape.

On the other hand, projective chain binding (Definition 18) is a stronger notion than both functional extractability and partial input soundness, as it must hold even if the adversary knows the trapdoor of the scheme. Moreover, projective chain binding holds for projections, which may or may not be (somewhere) extractable. In case the projected input commitments are indeed extractable, it is not hard to see that projective chain binding implies functional extractability.

5 Circuit-Succinct Compiler from PCFC to BARG

In this section we construct our black-box compiler that turns a PCFC (Definition 14) into a circuit-succinct BARG for NP, this is, a BARG such that the proof size grows linearly on the size of the witness $\ell_{\mathtt{w}}$. Our BARG can prove batch relations $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i)$ for $i \in [k]$, where $\mathcal{C} : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}$ are (possibly distinct, see Remark 7) arithmetic circuits of bounded size $|\mathcal{C}| \leq \ell_{\mathcal{C}}$.

We briefly describe our circuit-succinct compiler in Figure 2. The prover starts by committing to each of the witness wires across the k batch instances, obtaining commitments $\mathsf{com}_{\ell_{\mathtt{x}}+1}, \ldots, \mathsf{com}_{\ell_{\mathtt{x},\mathtt{w}}}$. The prover also provides subspace proofs on each of these commitments. Until this point, the compiler is analogous to the first step of the BARG construction in [WW22] (which we abstract in our wire-by-wire compiler in Appendix A). Then, instead of carrying out functional proofs for each circuit wire, the prover generates a PCFC functional opening as follows. First, it defines a function $f_{\mathcal{C}}$ to be the parallel evaluation of $\mathcal{C}$ k times. Then, it sets an input $\boldsymbol{x}$ which is the concatenation of $(\mathtt{x}_i, \mathtt{w}_i)$ for every $i \in [k]$. Finally, it provides a functional proof of $f_{\mathcal{C}}(\boldsymbol{x}) = (\mathcal{C}(\mathtt{x}_1, \mathtt{w}_1), \ldots, \mathcal{C}(\mathtt{x}_k, \mathtt{w}_k)) = \mathbf{1}_k$. Along with this proof, the prover provides an input commitment com and an output commitment com_y which commits to $\mathbf{1}_k$. The validity of com can be verified as it must match the aggregation of the commitments to the statement wires, which are computed by the verifier, along with the aggregation of $\mathsf{com}_{\ell_{\mathtt{x}}+1}, \ldots, \mathsf{com}_{\ell_{\mathtt{x},\mathtt{w}}}$.

The proof of BARG somewhere extractability for instance i^* intuitively works as follows. First, PC somewhere extractability enables the extraction of a binary witness $\mathtt{w}_{i^*}$ from each of the $\ell_{\mathtt{w}}$ commitment wires, where $\mathtt{w}_{i^*}$ is guaranteed to match the projections of these commitments. Second, we apply PC projective aggregatability to ensure that the projection of com indeed commits to $(\mathtt{x}_{i^*}, \mathtt{w}_{i^*})$. Finally, we crucially rely on PCFC projective chain binding to ensure that $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) = 1$.

Notation and parameters. For an (arithmetic or Boolean) circuit $\mathcal{C} : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}$, we define $f_{\mathcal{C}} : \bar{\mathcal{M}}^{k \cdot \ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}^{k \cdot \ell_{\mathtt{x},\mathtt{w}}}$ as $f_{\mathcal{C}}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (\mathcal{C}(\boldsymbol{x}_1), \ldots, \mathcal{C}(\boldsymbol{x}_k))$. Let $\mathcal{F}$ be the family of such functions $\mathcal{F} = \{f_{\mathcal{C}} : \mathcal{C} : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}\}$, and we let the input length $\ell = k \cdot \ell_{\mathtt{x},\mathtt{w}}$.

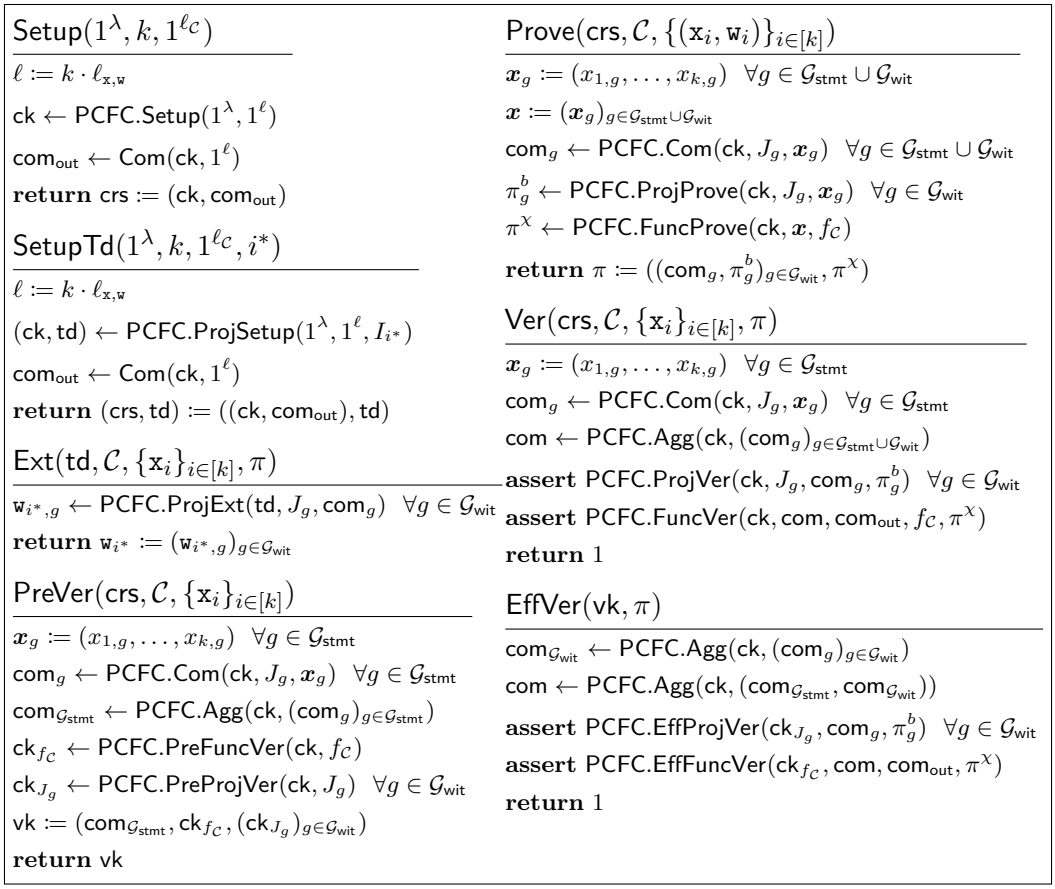


Fig. 2: Compiler from PCFC to fully succinct BARG.

To express $(i-1)\ell_{\mathtt{x},\mathtt{w}} + g \in [\ell]$ for $i \in [k], g \in \ell_{\mathtt{x},\mathtt{w}}$, we abuse notation and denote it by $(i, g) \in [\ell]$. That is, we use a bijective mapping between $[k] \times [\ell_{\mathtt{x},\mathtt{w}}]$ and $[\ell]$ given by $(i, g) \mapsto (i-1)\ell_{\mathtt{x},\mathtt{w}} + g$. This allows us to directly map values to their corresponding instance and wire. We define the index subsets $I_i := \{(i, g) : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}\}$ for $i \in [k]$ and $J_g := \{(i, g) : i \in [k]\}$ for $g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}$. Note that $I_i \cap J_g = \{(i, g)\}$ isolates a single index $(i, g) \in [\ell]$. By construction, all functions in $\mathcal{F}$ are $\mathcal{I}$-separable for the family of index sets $\mathcal{I} = \{I_i\}_{i\in[k]}$.

In Table 1, we show how the BARG inputs are arranged with respect to instances and subvectors. Recall that, for a given batch $i \in [k]$, we denote $x_{i,j} \leftarrow \mathtt{x}_{i,j}$ for $j = 1, \ldots, \ell_{\mathtt{x}}$, and $x_{i,j} \leftarrow \mathtt{w}_{i,(j-\ell_{\mathtt{x}})}$ for $j = \ell_{\mathtt{x}} + 1, \ldots, \ell_{\mathtt{x},\mathtt{w}}$.

Theorem 1 (Circuit-Succinct Compiler). *Let $\mathcal{I} := \{I_i : i \in [k]\}$ where $I_i := \{(i, g) : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}\}$, $\mathcal{J} := \{J_g : g \in \mathcal{G}_{\mathsf{stmt}} \cup \mathcal{G}_{\mathsf{wit}}\}$ where $J_g := \{(i, g) : i \in [k]\}$, $\mathcal{F} = \{f_C : \mathcal{C} : \bar{\mathcal{M}}^{\ell_{\mathtt{x},\mathtt{w}}} \to \bar{\mathcal{M}}\}$ where $f_C(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (\mathcal{C}(\boldsymbol{x}_1), \ldots, \mathcal{C}(\boldsymbol{x}_k))$, $\bar{\mathcal{M}} \supseteq \{0, 1\}$ be some message space,* PCFC *be a projective chainable functional commitment scheme (Definition 14) for the $\mathcal{I}$-separable functions $\mathcal{F}$ and index set families $(\mathcal{I}, \mathcal{J})$ which satisfies*

- *(Definition 11) PC projective aggregatability,*
- *(Definition 12) PC somewhere extractability for $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$ and*

	statements				witnesses					wires			
	J_1	J_2	$\cdots$	$J_{\ell_{\mathtt{x}}}$	$J_{\ell_{\mathtt{x}}+1}$	$\cdots$	$J_{\ell_{\mathtt{x}}+j}$	$\cdots$	$J_{\ell_{\mathtt{x},\mathtt{w}}}$	$J_{\ell_{\mathtt{x},\mathtt{w}}+1}$	$J_{\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	$J_{\ell_{\mathcal{C}}}$
I_1	$\mathtt{x}_{1,1}$	$\mathtt{x}_{1,2}$	$\cdots$	$\mathtt{x}_{1,\ell_{\mathtt{x}}}$	$\mathtt{w}_{1,1}$	$\cdots$	$\mathtt{w}_{1,j}$	$\cdots$	$\mathtt{w}_{1,\ell_{\mathtt{w}}}$	$x_{1,\ell_{\mathtt{x},\mathtt{w}}+1}$	$x_{1,\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	$x_{1,\ell_{\mathcal{C}}}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$
I_i	$\mathtt{x}_{i,1}$	$\mathtt{x}_{i,2}$	$\cdots$	$\mathtt{x}_{i,\ell_{\mathtt{x}}}$	$\mathtt{w}_{i,1}$	$\cdots$	$\mathtt{w}_{i,j}$	$\cdots$	$\mathtt{w}_{i,\ell_{\mathtt{w}}}$	$x_{i,\ell_{\mathtt{x},\mathtt{w}}+1}$	$x_{i,\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	$x_{i,\ell_{\mathcal{C}}}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$	$\vdots$		$\vdots$
I_k	$\mathtt{x}_{k,1}$	$\mathtt{x}_{k,2}$	$\cdots$	$\mathtt{x}_{k,\ell_{\mathtt{x}}}$	$\mathtt{w}_{k,1}$	$\cdots$	$\mathtt{w}_{k,j}$	$\cdots$	$\mathtt{w}_{k,\ell_{\mathtt{w}}}$	$x_{k,\ell_{\mathtt{x},\mathtt{w}}+1}$	$x_{k,\ell_{\mathtt{x},\mathtt{w}}+2}$	$\cdots$	$x_{k,\ell_{\mathcal{C}}}$
	$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$		$\downarrow$		$\downarrow$				
	com_1	com_2		$\mathsf{com}_{\ell_{\mathtt{x}}}$	$\mathsf{com}_{\ell_{\mathtt{x}}+1}$		$\mathsf{com}_{\ell_{\mathtt{x}}+j}$		$\mathsf{com}_{\ell_{\mathtt{x},\mathtt{w}}}$				

Table 1: Structure of the committed values in Figure 2. The rows are associated to projection sets $I_i \in \mathcal{I}$. The columns are associated to the subvector sets $J_j \in \mathcal{J}$.

– *(Definition 18) PCFC projective chain binding for* $(\mathcal{F}, \mathcal{I}, \mathcal{J})$.

Then, the construction BARG *in Figure 2 is a somewhere extractable batch argument (Definition 1) for NP whose proof size is* $|\pi| = \ell_{\mathtt{w}} \cdot (|\mathsf{com}| + |\pi^b|) + |\pi^\chi|$ *where* π^b *is a PC subspace proof, and* π^χ *is a PCFC functional proof for* $f \in \mathcal{F}$*. Moreover,* BARG *admits efficient verification with preprocessing.*

Proof. The completeness, full succinctness and setup indistinguishability of BARG follow by inspection, given the correctness, succinctness and setup indistinguishability of PCFC. The somewhere extractability of BARG follows from Lemma 1. Its efficient verification property follows from Lemma 5. □

Remark 6. The projective aggregatability property required by Theorem 1 is optional – its only purpose is so that a BARG proof does not need to include $|\mathcal{G}_{\mathsf{wit}}|$ commitments. If we were to drop the projective aggregatability property, the compiler needs to be slightly modified where FuncProve and FuncVer instead take as input multiple vectors and input commitments respectively.

Remark 7. Although we instantiate our compiler for the standard batch relation defined by $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i)$ for $i \in [k]$, where the circuit is the same for all batch instances, it actually supports batch proofs for distinct circuits $\mathcal{C}_i(\mathtt{x}_i, \mathtt{w}_i)$ for $i \in [k]$ *without any additional overhead.* The only requirements are that each $|\mathcal{C}_i| \leq \ell_{\mathcal{C}}$ and that the PCFC supports the consequently broader family of functions $\mathcal{F}_{k,\ell_f}$ (defined in Section 6), which is the case for our algebraic PCFC introduced there. This property can be useful in applications such as the aggregation of signature schemes with different verification equations.

5.1 Proof of Somewhere Extractability

Lemma 1. *Let* PCFC *and* BARG *be as defined in Theorem 1. It holds that* BARG *has somewhere extractability.*

Proof. We proceed by a series of hybrid games $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,0}, \ldots, \mathsf{Hyb}^{1,\ell_{\mathtt{w}}}, \mathsf{Hyb}^2$ which progressively include additional winning conditions. The formal descriptions of the games are given in Figure 3 with explanations given below.

$\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$:

$(\mathsf{ck}, \mathsf{td}) \leftarrow \mathsf{PCFC.ProjSetup}(1^\lambda, 1^k, I_{i^*})$

$\mathsf{com}_{\mathsf{out}} \leftarrow \mathsf{Com}(\mathsf{ck}, 1^\ell)$

$(\mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, ((\mathsf{com}_g, \pi^b_g)_{g\in\mathcal{G}_{\mathsf{wit}}}, \pi^\chi)) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{com}_{\mathsf{out}})$

$\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, J_g, \boldsymbol{x}_g) \quad \forall g \in \mathcal{G}_{\mathsf{stmt}}$

$\mathsf{com} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{stmt}}\cup\mathcal{G}_{\mathsf{wit}}})$

$x^*_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, J_g, \mathsf{com}_g) \quad \forall g \in \mathcal{G}_{\mathsf{wit}}$

$\mathtt{w}^* := (x^*_g)_{g\in\mathcal{G}_{\mathsf{wit}}}$

assert $\mathsf{PCFC.ProjVer}(\mathsf{ck}, J_g, \mathsf{com}_g, \pi^b_g) = 1 \quad \forall\, g \in \mathcal{G}_{\mathsf{wit}}$

assert $\mathsf{PCFC.FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_{\mathsf{out}}, f_{\mathcal{C}}, \pi^\chi) = 1$

assert $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}^*) \neq 1$

return 1

$\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathtt{w}}$:

// identical to $\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$ until before "**return** 1"

assert $x^*_g \in \bar{\mathcal{M}} \quad \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$

assert $\mathsf{PCFC.ProjCom}(\mathsf{td}, J_g, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}_g) \quad \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$

$\mathsf{Hyb}^2_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathsf{out}}$:

// identical to $\mathsf{Hyb}^{1,\ell_{\mathtt{w}}}_{\mathcal{A}}(\lambda)$ until before "**return** 1"

assert $\mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*)) = \mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com})$

Fig. 3: Games $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,j}, \mathsf{Hyb}^2$ for the proof of BARG somewhere extractability. All games output 1 by default. We highlight changes between games.

Hyb^0**:** This is the standard somewhere extractability game (Definition 1) with the algorithms in BARG inlined. To recall, in this game the adversary outputs a proof π and the experiment runs $\mathtt{w}_{i^*} \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, \mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi)$. The adversary wins if the proof verifies yet $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) \neq 1$.

$\mathsf{Hyb}^{1,j}$**:** $\mathsf{Hyb}^{1,0}$ is identical to Hyb^0. In $\mathsf{Hyb}^{1,j}$, the game additionally checks that the first j entries of the extracted witness, i.e. those indexed by g belonging to the first j values in $\mathcal{G}_{\mathsf{wit}}$, resides in the message subspace $\bar{\mathcal{M}}$, i.e. $x^*_g \in \bar{\mathcal{M}}^{\ell_{\mathtt{w}}}$, and that they are consistent with the projected commitments, i.e. $\mathsf{PCFC.ProjCom}(\mathsf{td}, J_g, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}_g)$.

Hyb^2**:** This game simply adds the winning condition that

$$\mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*)).$$

We now proceed to bound the advantage of any PPT adversary $\mathcal{A}$ against Hyb^0 by a series of lemmas.

Lemma 2. *For any $j \in [\ell_{\mathtt{w}}]$, there exists a PPT adversary $\mathcal{B}$ against the somewhere extractability for $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$ of* PCFC *such that*

$$\left|\Pr[\mathsf{Hyb}^{1,j-1}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda) = 1]\right| \le \mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. Fix $j \in [\ell_{\mathtt{w}}]$. We build an adversary $\mathcal{B}$ as follows. $\mathcal{B}$ receives input ck from its challenger and runs $\mathcal{A}(\mathsf{ck})$. Then, it parses the proof π of $\mathcal{A}$ and retrieves the tuple $(\mathsf{com}_{\ell_{\mathtt{x}}+j}, \pi^b_{\ell_{\mathtt{x}}+j})$. Finally, $\mathcal{B}$ simply forwards these to its challenger.

We argue that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j-1}$ but not in $\mathsf{Hyb}^{1,j}$, then $\mathcal{B}$ also wins the game. First, as $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j}$, it holds that $\mathsf{PCFC.ProjVer}(\mathsf{ck}, \mathsf{com}_{\ell_{\mathtt{x}}+j}, \pi^b_{\ell_{\mathtt{x}}+j}) = 1$. Also, note that $x^*_{\ell_{\mathtt{x}}+j} \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathtt{x}}+j})$. Therefore, all the preconditions for $\mathcal{B}$ in the PC somewhere extractability game are met. Finally, as $\mathcal{A}$ does not win in $\mathsf{Hyb}^{1,j}$, it must be that either $\mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathtt{x}}+j}) \neq \mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_{\ell_{\mathtt{x}}+j})$ or $x^*_{\ell_{\mathtt{x}}+j} \notin \bar{\mathcal{M}}$. □

Lemma 3. *It holds that*

$$\Pr[\mathsf{Hyb}^{1,\ell_{\mathtt{w}}}_{\mathcal{A}}(\lambda) = 1] = \Pr[\mathsf{Hyb}^{2}_{\mathcal{A}}(\lambda) = 1].$$

Proof. Immediate from projective aggregatability (Definition 11). □

Lemma 4. *There exists a PPT adversary $\mathcal{B}$ against projective chain binding for $(\mathcal{F}, \mathcal{I}, \mathcal{J})$ of* PCFC *such that*

$$\Pr[\mathsf{Hyb}^{2}_{\mathcal{A}}(\lambda) = 1] \leq \mathsf{Adv}^{\mathrm{pcb}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. We build the adversary $\mathcal{B}$ against projective chain binding from $\mathcal{A}$ against Hyb^2 as follows. Our adversary $\mathcal{B}$ receives $(\mathsf{ck}, \mathsf{td})$ from its challenger, computes $\mathsf{com}_{\mathsf{out}} = \mathsf{PCFC.Com}(\mathsf{ck}, 1^\ell)$ and runs $\mathcal{A}$ on $(\mathsf{ck}, \mathsf{com}_{\mathsf{out}})$ (note that $\mathcal{A}$ does not receive td as input). It obtains $(f_{\mathcal{C}}, \mathsf{com}_{\mathsf{out}}, (\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{wit}}}, \pi^{\chi})$ from $\mathcal{A}$.

Next, $\mathcal{B}$ extracts the witness values $\mathtt{w}^*_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, J_g, \mathsf{com}_g)$ one by one from the commitments $(\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{wit}}}$ and generates dummy circuit inputs $\tilde{\boldsymbol{x}}_i = \mathtt{x}_i|\tilde{\mathtt{w}}_i$ as follows:

$$\tilde{\boldsymbol{x}}_i \leftarrow \mathtt{x}_i|\mathbf{0}_{\ell_{\mathtt{w}}} \text{ if } i \neq i^*; \quad \mathtt{x}_i|\mathtt{w}^* \text{ if } i = i^*.$$

It commits to the dummy inputs, $\tilde{\mathsf{com}} \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_k))$. Let also $\tilde{y}_i \leftarrow \mathcal{C}(\tilde{\boldsymbol{x}}_i)$ for all $i \in [k]$, and let $\tilde{\mathsf{com}}_{\mathsf{out}} \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (\tilde{y}_1, \ldots, \tilde{y}_k))$. Finally, $\mathcal{B}$ generates an honestly computed opening proof that $f_{\mathcal{C}}(\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_k) = (y_1, \ldots, y_k)$ by running $\tilde{\pi^{\chi}} \leftarrow \mathsf{PCFC.FuncProve}(\mathsf{ck}, \tilde{\boldsymbol{x}}, f_{\mathcal{C}})$. Then, it returns the tuple

$$(f_{\mathcal{C}}, \mathsf{com}, \tilde{\mathsf{com}}, \mathsf{com}_{\mathsf{out}}, \tilde{\mathsf{com}}_{\mathsf{out}}, \pi^{\chi}, \tilde{\pi^{\chi}}).$$

Clearly, $\mathcal{B}$ runs in polynomial time. We next analyse its success probability of violating the projective chain binding property. Let $(\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{wit}}}, \pi^{\chi}$ be parsed from the outputs of $\mathcal{A}$. Let $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, J_g, \boldsymbol{x}_g)$ for $g \in \mathcal{G}_{\mathsf{stmt}}$ and $\mathsf{com} \leftarrow \mathsf{PCFC.Agg}(\mathsf{ck}, (\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{stmt}}\cup\mathcal{G}_{\mathsf{wit}}})$. By the correctness of PCFC,

- $\mathsf{PCFC.FuncVer}(\mathsf{ck}, \tilde{\mathsf{com}}, \tilde{\mathsf{com}}_{\mathsf{out}}, f_{\mathcal{C}}, \tilde{\pi^{\chi}}) = 1$,
- $\mathsf{PCFC.Proj}(\mathsf{td}, \tilde{\mathsf{com}}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*))$, and
- $\mathsf{PCFC.Proj}(\mathsf{td}, \tilde{\mathsf{com}}_{\mathsf{out}}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, 1^{\ell_{\mathtt{x},\mathtt{w}}} \cdot \tilde{y}_{i^*})$, $\tilde{y}_{i^*} = \mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}^*) \neq 1$.

On the other hand, with probability $\Pr[\mathsf{Hyb}^2_{\mathcal{A}}(\lambda) = 1]$, it holds that

- $\mathsf{PCFC.FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_{\mathsf{out}}, f_{\mathcal{C}}, \pi^{\chi}) = 1$,
- $\mathsf{PCFC.Proj}(\mathsf{td}, \mathsf{com}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, (\mathtt{x}_{i^*}, \mathtt{w}^*))$, and
- $\mathsf{PCFC.Proj}(\mathsf{td}, \tilde{\mathsf{com}}_{\mathsf{out}}) = \mathsf{PCFC.ProjCom}(\mathsf{td}, I_{i^*}, 1^{\ell_{\mathtt{x},\mathtt{w}}})$.

In other words, $\mathcal{B}$ successfully wins the PCFC projective chain binding game with probability $\Pr[\mathsf{Hyb}^2_{\mathcal{A}}(\lambda) = 1]$.

We collect the bounds of Lemma 2, Lemma 3, and Lemma 4, conclude that the advantage of $\mathcal{A}$ in the original Hyb^0 is bounded by

$$\Pr[\mathsf{Hyb}^0_{\mathcal{A}}(\lambda) = 1] \leq \ell_{\mathtt{w}} \cdot \mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathrm{pcb}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$ □

5.2 Efficient Verification

Lemma 5. *Let* PCFC *and* BARG *be as defined in Theorem 1. If further* PCFC *has efficient verification with preprocessing (Definition 15), then* BARG *has circuit-succinct verification with preprocessing (Definition 2).*

Proof. As per Definition 2, it suffices to show that $|\mathsf{vk}|$ and the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ are at most $\mathsf{poly}(\lambda, \log k, \ell_{\mathtt{w}})$. We actually prove below a stronger claim that these quantities are at most $\mathsf{poly}(\lambda, \log k) \cdot \ell_{\mathtt{w}}$. Note that since $\ell = k \cdot \ell_{\mathtt{x},\mathtt{w}}$ and $\ell_{\mathtt{x}}$ and $\ell_{\mathtt{w}}$ are fixed $\mathsf{poly}(\lambda)$, we have $\mathsf{poly}(\lambda, \log \ell) = \mathsf{poly}(\lambda, \log k)$. Therefore, in the below we express all complexities in terms of ℓ instead of k.

We first examine $|\mathsf{vk}|$. Recall that $\mathsf{vk} = (\mathsf{com}_{\mathcal{G}_{\mathsf{stmt}}}, \mathsf{ck}_{f_{\mathcal{C}}}, (\mathsf{ck}_{J_g})_{g \in \mathcal{G}_{\mathsf{wit}}})$. Note that by Definition 5 fresh commitments are of size at most $\mathsf{poly}(\lambda, \log \ell)$ and by Definition 6 aggregated commitments are identical to fresh ones. Therefore, $|\mathsf{com}_{\mathcal{G}_{\mathsf{stmt}}}| \leq \mathsf{poly}(\lambda, \log \ell)$. Next, by Definition 9 it holds that $|\mathsf{ck}_{f_{\mathcal{C}}}| \leq \mathsf{poly}(\lambda, \log \ell, \log m, \log |f_{\mathcal{C}}|)$ where $m = 1$ in the current context and $|f_{\mathcal{C}}| \leq 2^{\mathsf{poly}(\lambda)}$. Thus, $|\mathsf{ck}_{f_{\mathcal{C}}}| \leq \mathsf{poly}(\lambda, \log \ell)$. Finally, by Definition 13 it holds that $|\mathsf{ck}_{J_g}| \leq \mathsf{poly}(\lambda, \log \ell, \log |J_g|)$ for each $g \in \mathcal{G}_{\mathsf{wit}}$, where $|J_g| = k$ and $|\mathcal{G}_{\mathsf{wit}}| = \ell_{\mathtt{w}}$. Putting everything together, we conclude that $|\mathsf{vk}| \leq \mathsf{poly}(\lambda, \log k) \cdot \ell_{\mathtt{w}}$.

For the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$, by Definition 6, the two calls to PCFC.Agg take time at most $\mathsf{poly}(\lambda, \log \ell) \cdot |\mathcal{G}_{\mathsf{wit}}| = \mathsf{poly}(\lambda, \log \ell) \cdot \ell_{\mathtt{w}}$ and $\mathsf{poly}(\lambda, \log \ell) \cdot 2$ respectively. Then, each of the $\ell_{\mathtt{w}}$ calls to PCFC.EffProjVer runs in time at most $\mathsf{poly}(\lambda, \log \ell, |J_g|)$ according to Definition 13. Finally, the call to PCFC.EffFuncVer runs in time at most $\mathsf{poly}(\lambda, \log \ell, \log m, \log |f_{\mathcal{C}}|)$ which as analysed above is at most $\mathsf{poly}(\lambda, \log \ell)$. We thus conclude that $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ runs in time at most $\mathsf{poly}(\lambda, \log \ell) \cdot \ell_{\mathtt{w}}$. □

6 Algebraic PCFC from Bilinear Groups

In this section, we give an algebraic construction of a somewhere extractable projective chainable functional commitment scheme PCFC that is sufficient to instantiate Theorem 1, yielding a circuit-succinct BARG for circuit satisfiability.

Let $\mathcal{F}_{\mathcal{C}} = \{\mathcal{C} : \mathbb{F}^{\ell} \to \mathbb{F}^{\ell} : |\mathcal{C}| \leq \ell_{\mathcal{C}}\}$ be the family of arithmetic circuits of size bounded by $\ell_{\mathcal{C}}$, where without loss of generality we let the input space be equal to the output space. We define the function family

$$\mathcal{F}_{k,\ell_{\mathcal{C}}} = \{f : \mathbb{F}^{k\ell} \to \mathbb{F}^{k\ell}; f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (\mathcal{C}_1(\boldsymbol{x}_1), \ldots, \mathcal{C}_k(\boldsymbol{x}_k)) \ \wedge \ \mathcal{C}_i \in \mathcal{F}_{\mathcal{C}} \ \forall i \in [k]\}.$$

That is, $\mathcal{F}_{k,\ell_{\mathcal{C}}}$ consists of all functions which input k vectors $(\boldsymbol{x}_i)_{i \in [k]}$ and evaluate some circuit $\mathcal{C}_i \in \mathcal{F}_{\mathcal{C}}$ on $\boldsymbol{x}_i$ for each $i \in [k]$.

The section is structured in two parts, where we incrementally construct PCFC using only generic operations over pairing groups. We first construct a somewhere extractable PC in Section 6.1 and then equip it with CFC algorithms in Section 6.2 turning it into a PCFC, while ensuring the compatibility of all properties. We remark that we also provide an algebraic interpretation of the compiler in Appendix B when instantiated with the schemes in this section, which offers a different perspective on the scheme.

The results of this section are summarised in the following theorem:

Theorem 2. *Let* $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2)$ *be a pairing setting description, and let* $M = \mathbb{F} = \mathbb{Z}_q$*. Assume that the bilateral* $\mathsf{MDDH}_{\kappa,\ell,d}$ *assumption and the* $\mathsf{KerDH}_{\kappa,\ell}$ *assumption hold over* bgp *for some* $\kappa \geq 2$ *and* $\ell, d = \mathsf{poly}(\lambda)$*. Then, the construction* $\mathsf{PCFC} = (\mathsf{PC}, \mathsf{CFC})$ *is a somewhere extractable PCFC with the following properties:*

- *(Projective) aggregatable.*
- PC *is somewhere extractable for* $(\mathcal{I}, \mathcal{J}, \bar{\mathcal{M}})$*, where* $\bar{\mathcal{M}} = \{0,1\}$ *and* $\mathcal{I}, \mathcal{J} \subseteq 2^{[\ell]}$ *are arbitrary families of sets such that* $|I \cap J| \leq 1$ *for every* $I \in \mathcal{I}, J \in \mathcal{J}$*.*
- CFC *is projective chain binding for* $(\mathcal{F}_{k,\ell_{\mathcal{C}}}, \mathcal{I}')$ *where* $\mathcal{I}' = \{\{i(\ell-1)+1, \ldots, i\ell\} : i \in [k]\}$*.*
- *The commitment key size* $|\mathsf{ck}|$ *and the prover running time are* $\mathcal{O}(\lambda \cdot \kappa^3 \cdot k^5 \cdot \ell_{\mathcal{C}}^5)$*.*
- *Commitments have size* $|\mathsf{com}| = \mathcal{O}(\lambda \cdot \kappa)$*.*
- *The* PC *subspace proofs have size* $\left|\pi^b\right| = \mathcal{O}(\lambda \cdot \kappa^2)$*.*
- *The* PCFC *functional proofs have size* $\left|\pi^f\right| = \mathcal{O}(\lambda \cdot \kappa^2)$*.*
- PCFC *efficient verification runs in time* $\mathcal{O}(\lambda \cdot \kappa^2)$*.*

Proof. The proof follows from the constructions in Section 6.1 and Section 6.2 and from the series of lemmas proven there. We state the different properties: *a*) Binding follows from Lemma 6. *b*) Projection correctness follows from Lemma 7. *c*) Setup indistinguishability follows from both Lemma 8 and Lemma 14. *d*) Somewhere completeness follows from Lemma 9. *e*) Somewhere extractability follows from Lemma 10. *f*) Aggregatability and projective aggregatability follow from Lemma 12. *g*) Projective chain binding follows from Theorem 3. *h*) Efficient verification follows from both Lemma 13 and Lemma 16. □

As κ is a small constant (e.g., $\kappa = 2$), the combination of Theorem 1 and Theorem 2 yields the following corollary:

Corollary 1. *Under the assumptions of Theorem 2, there exists an algebraic somewhere extractable BARG for NP such that:*

- *The setup size* $|\mathsf{crs}|$ *and the prover running time are* $\mathcal{O}(\lambda \cdot k^5 \cdot \ell_{\mathcal{C}}^5)$*.*
- *The proof size is* $|\pi| = \mathcal{O}(\lambda \cdot \ell_{\mathtt{w}})$*.*
- *Efficient verification runs in time* $\mathcal{O}(\lambda \cdot \ell_{\mathtt{w}})$*.*

6.1 Algebraic PC from Bilinear Groups

We introduce a construction of a somewhere extractable PC. Conceptually, our PC is an adaptation of the proof systems in the WW22 BARG [WW22] to work over a larger projective commitment key that is compatible with the WW24 FC proof systems [WW24]. Notably, the projective space in our PC has dimension κ, as opposed to 1 in the commitments in the WW22 BARG. Therefore, our commitments are vectors of length 2κ instead of $\kappa + 1$. Besides, our scheme explicitly supports subvector commitments, as well as their aggregation. We remark that all algorithms are well defined for any $J \in \mathcal{J} = 2^{[\ell]}$. The message space is defined as $\mathcal{M} = \mathbb{Z}_q = \mathbb{F}$ and the small message space as $\bar{\mathcal{M}} = \{0,1\}$. All vectors $\boldsymbol{x} \in \mathbb{F}^\ell$ are column vectors. As in Section 6.2 we borrow algorithms from the [WW24] CFC, we follow their notation closely.

Our PC construction.

$\underline{\mathsf{Setup}(1^\lambda, 1^\ell)}$**:**

– Generate a bilinear group description $\mathsf{bgp} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, [1]_1, [1]_2) \leftarrow \mathcal{BG}(1^\lambda)$, and let $\mathbb{F} := \mathbb{Z}_q$ and κ be the κ-Lin parameter.

– Sample random invertible $\hat{\mathbf{B}}, \check{\mathbf{B}} \leftarrow\!\!\$\ \mathbb{F}^{2\kappa\times 2\kappa}$ and let $\hat{\mathbf{B}}^* = \hat{\mathbf{B}}^{-1}$ and $\check{\mathbf{B}}^* = \check{\mathbf{B}}^{-1}$.

– Split $\hat{\mathbf{B}} = \begin{bmatrix}\hat{\mathbf{B}}_1\\ \hat{\mathbf{B}}_2\end{bmatrix}$ where $\hat{\mathbf{B}}_1, \hat{\mathbf{B}}_2 \in \mathbb{F}^{\kappa\times 2\kappa}$ and split $\hat{\mathbf{B}}^* = \begin{bmatrix}\hat{\mathbf{B}}_1^* \ \hat{\mathbf{B}}_2^*\end{bmatrix}$ where $\hat{\mathbf{B}}_1^*, \hat{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa\times\kappa}$. By construction, $\hat{\mathbf{B}}_1 \cdot \hat{\mathbf{B}}_1^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_2^* = \mathbf{I}_\kappa$ and $\hat{\mathbf{B}}_1 \cdot \hat{\mathbf{B}}_2^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_1^* = \mathbf{0}_\kappa$. Define $\check{\mathbf{B}}_1, \check{\mathbf{B}}_2 \in \mathbb{F}^{\kappa\times 2\kappa}$ and $\check{\mathbf{B}}_1^*, \check{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa\times\kappa}$ analogously.

– Sample $\hat{\mathbf{S}}_1, \check{\mathbf{S}}_1 \leftarrow\!\!\$\ \mathbb{F}^{\kappa\times\ell}$, and define $\hat{\mathbf{T}} = \hat{\mathbf{B}}_1^* \cdot \hat{\mathbf{S}}_1 \in \mathbb{F}^{2\kappa\times\ell}$ and $\check{\mathbf{T}} = \check{\mathbf{B}}_1^* \cdot \check{\mathbf{S}}_1 \in \mathbb{F}^{2\kappa\times\ell}$. Let also $\hat{\boldsymbol{t}}_i \in \mathbb{F}^{2\kappa}$ be each of the column vectors in $\hat{\mathbf{T}}$, and similarly for $\check{\boldsymbol{t}}_i \in \mathbb{F}^{2\kappa}$ and $\check{\mathbf{T}}$.

– Finally, sample $\mathbf{R}_{i,j} \leftarrow\!\!\$\ \mathbb{F}^{\kappa\times\kappa}$ and set

$$\hat{\mathbf{W}}_{i,j} \leftarrow -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T, \quad \check{\mathbf{W}}_{i,j} \leftarrow \check{\mathbf{B}}_1^* (\check{s}_{1,i} \hat{\mathbf{s}}_{1,j}^T + \mathbf{R}_{i,j}).$$

Output the following commitment key:

$$\mathsf{ck} := \left(\left[\check{\mathbf{T}}\right]_1, \left[\hat{\mathbf{T}}\right]_2, \left[\check{\mathbf{B}}_1^*\right]_1, \left[\hat{\mathbf{B}}_1^*\right]_2, \left\{ \left[\check{\mathbf{W}}_{i,j}\right]_1, \left[\hat{\mathbf{W}}_{i,j}\right]_2 \right\}_{i,j\in[\ell]} \right).$$

Given any subset $J \subseteq [\ell]$, we let $\hat{\boldsymbol{t}}_J \leftarrow \sum_{i\in J} \hat{\boldsymbol{t}}_i$ and $\check{\boldsymbol{t}}_J \leftarrow \sum_{i\in J} \check{\boldsymbol{t}}_i$. Note that $\left[\check{\boldsymbol{t}}_J\right]_1, \left[\hat{\boldsymbol{t}}_J\right]_2$ can be computed from ck for any $J \subseteq [\ell]$.

$\underline{\mathsf{ProjSetup}(1^\lambda, 1^\ell, I)}$**:** Let $\mathbf{P}_I \in \mathbb{F}^{\ell\times\ell}$ be the (diagonal) projection matrix associated to I, defined by $\mathbf{P}_{Ii,i} = 1$ if $i \in I$ and $\mathbf{P}_{Ii,j} = 0$ elsewhere. Sample all matrices as before, and additionally sample $\hat{\mathbf{S}}_2, \check{\mathbf{S}}_2 \leftarrow\!\!\$\ \mathbb{F}^{\kappa\times\ell}$. Then, set

$$\hat{\mathbf{T}} \leftarrow \hat{\mathbf{B}}_1^* \cdot \hat{\mathbf{S}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\mathbf{S}}_2 \cdot \mathbf{P}_I, \quad \check{\mathbf{T}} \leftarrow \check{\mathbf{B}}_1^* \cdot \check{\mathbf{S}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\mathbf{S}}_2 \cdot \mathbf{P}_I$$

$$\hat{\mathbf{W}}_{i,j} \leftarrow \begin{cases} -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T + \hat{\boldsymbol{t}}_i \check{\boldsymbol{s}}_{1,j}^T & \text{if } j \in I \\ -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T & \text{if } j \notin I \end{cases}, \quad \check{\mathbf{W}}_{i,j} \leftarrow \begin{cases} \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} & \text{if } j \in I \\ \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} + \check{\boldsymbol{t}}_i \hat{\boldsymbol{s}}_{1,j}^T & \text{if } j \notin I \end{cases}.$$

Output the commitment key as before, and output $\mathsf{td} := (I, \check{\mathbf{B}}_2, \check{\mathbf{S}}_2, \hat{\mathbf{B}}_2, \hat{\mathbf{S}}_2)$.

$\underline{\mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})}$**:** Output $[\hat{\boldsymbol{c}}]_2 \leftarrow \sum_{i\in J} \left[\hat{\boldsymbol{t}}_i\right]_2 \cdot x_i$.

$\underline{\mathsf{ProjCom}(\mathsf{td}, J, \boldsymbol{x})}$**:** Output $\left[\hat{\boldsymbol{d}}\right]_2 \leftarrow \sum_{i\in I\cap J} [\hat{\mathbf{s}}_{2,i}]_2 \cdot x_i$.

$\underline{\mathsf{Proj}(\mathsf{td}, \mathsf{com})}$**:** Parse $\hat{\mathbf{B}}_2$ from td, and parse $\mathsf{com} = [\hat{\boldsymbol{c}}]_2$. Output $\left[\hat{\boldsymbol{d}}\right]_2 \leftarrow \hat{\mathbf{B}}_2 \cdot [\hat{\boldsymbol{c}}]_2$.

$\underline{\mathsf{ProjProve}(\mathsf{ck}, J, \boldsymbol{x})}$**:**

– Compute the twin commitment $[\check{\boldsymbol{c}}]_1 \leftarrow \sum_{i\in J} \left[\check{\boldsymbol{t}}_i\right]_1 x_i$.

– Compute the booleanity proofs:

$$\left[\check{\boldsymbol{\Pi}}_1\right]_1 \leftarrow \sum_{i,j\in J, i\neq j} \left[\check{\mathbf{W}}_{i,j}\right]_1 \cdot x_i(x_j - 1), \quad \left[\hat{\boldsymbol{\Pi}}_1\right]_2 \leftarrow \sum_{i,j\in J, i\neq j} \left[\hat{\mathbf{W}}_{i,j}\right]_2 \cdot x_i(x_j - 1),$$

$$\left[\check{\boldsymbol{\Pi}}_2\right]_1 \leftarrow \sum_{i,j\in J, i\neq j} \left[\check{\mathbf{W}}_{i,j}\right]_1 \cdot x_j(x_i - 1), \quad \left[\hat{\boldsymbol{\Pi}}_2\right]_2 \leftarrow \sum_{i,j\in J, i\neq j} \left[\hat{\mathbf{W}}_{i,j}\right]_2 \cdot x_j(x_i - 1).$$

– Output $\pi \leftarrow \left([\check{\boldsymbol{c}}]_1, \left[\check{\boldsymbol{\Pi}}_1\right]_1, \left[\hat{\boldsymbol{\Pi}}_1\right]_2, \left[\check{\boldsymbol{\Pi}}_2\right]_1, \left[\hat{\boldsymbol{\Pi}}_2\right]_2\right)$.

$\underline{\mathsf{ProjVer}(\mathsf{ck}, J, \mathsf{com}, \pi)\text{:}}$ Parse $\pi = \left([\check{\boldsymbol{c}}]_1, \left[\check{\boldsymbol{\Pi}}_1\right]_1, \left[\hat{\boldsymbol{\Pi}}_1\right]_2, \left[\check{\boldsymbol{\Pi}}_2\right]_1, \left[\hat{\boldsymbol{\Pi}}_2\right]_2\right)$ and $\mathsf{com} = [\hat{\boldsymbol{c}}]_2$. Compute $\left[\check{\boldsymbol{t}}_J\right]_1, \left[\hat{\boldsymbol{t}}_J\right]_2$. Output 1 if and only if both of the following hold:

$$[\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{t}}_J^T\right]_2 = [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\boldsymbol{\Pi}}_1^T\right]_2 + \left[\check{\boldsymbol{\Pi}}_1\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2. \tag{1}$$

$$\left[\check{\boldsymbol{t}}_J\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 = [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\boldsymbol{\Pi}}_2^T\right]_2 + \left[\check{\boldsymbol{\Pi}}_2\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2. \tag{2}$$

$\underline{\mathsf{PreProjVer}(\mathsf{ck}, J)\text{:}}$ Output $\mathsf{ck}_J \leftarrow \left(\left[\check{\boldsymbol{t}}_J\right]_1, \left[\hat{\boldsymbol{t}}_J\right]_2, \left[\check{\mathbf{B}}_1^*\right]_1, \left[\hat{\mathbf{B}}_1^{*T}\right]_2\right)$.

$\underline{\mathsf{EffProjVer}(\mathsf{ck}_J, \mathsf{com}, \pi)\text{:}}$ Check equations 1, 2 from $\mathsf{ProjVer}$ and output 1 if and only if both hold.

$\underline{\mathsf{ProjExt}(\mathsf{td}, J, \mathsf{com})\text{:}}$ Parse $(I, \hat{\mathbf{S}}_2)$ from td and compute $\left[\tilde{\boldsymbol{d}}\right]_2 \leftarrow \mathsf{Proj}(\mathsf{td}, \mathsf{com})$. Then, find a vector $\tilde{\boldsymbol{x}} \in \{0,1\}^{I \cap J}$ such that $\left[\tilde{\boldsymbol{d}}\right]_2 = \sum_{i \in I \cap J} [\hat{\boldsymbol{s}}_{2,i}]_2 \cdot \tilde{x}_i$. Output $\tilde{\boldsymbol{x}}$ if found. Otherwise, output $\perp$.

Note that this algorithm runs in time $\mathcal{O}\left(2^{|I \cap J|} \cdot \mathsf{poly}(\lambda)\right)$.

$\underline{\mathsf{Agg}(\mathsf{ck}, (\mathsf{com}_J)_{J \in \mathcal{J}'}) \to \mathsf{com}\text{ :}}$ Output $[\hat{\boldsymbol{c}}]_2 \leftarrow \sum_{J \in \mathcal{J}'} [\hat{\boldsymbol{c}}_J]_2$.

$\underline{\mathsf{ProjAgg}(\mathsf{ck}, (\mathsf{pcom}_J)_{J \in \mathcal{J}'}) \to \mathsf{pcom}\text{ :}}$ Output $\left[\hat{\boldsymbol{d}}\right]_2 \leftarrow \sum_{J \in \mathcal{J}'} \left[\hat{\boldsymbol{d}}_J\right]_2$.

Proofs of the properties of PC. Next, we prove that PC satisfies all the properties required by Theorem 2. We remark that most of the properties and the proofs follow the blueprints of [WW22] with minor modifications. Essentially, we adapt their proofs to work over a projective space of dimension κ, as the commitments in the WW22 BARG implicitly have a projective space of dimension 1.

Lemma 6. *Assume that the* $\mathsf{KerDH}_{\kappa,\ell}$ *assumption holds over* bgp*. Then,* se-PC *is binding (Definition 5).*

Proof. Let $\mathcal{A}(\mathsf{ck})$ be a PPT algorithm against binding that returns $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{F}^J$ such that $\boldsymbol{x} \neq \boldsymbol{x}'$ and $\mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}) = \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x}')$. Note that by construction $\left[\hat{\mathbf{T}}_J\right]_2 \cdot (\boldsymbol{x} - \boldsymbol{x}') = [\mathbf{0}]_2$.

We show how to use $\mathcal{A}$ to build an adversary $\mathcal{B}$ against $\mathsf{KerDH}_{\kappa,\ell}$. $\mathcal{B}$ receives $\left[\hat{\mathbf{S}}_1\right]_2 \in \mathbb{G}_2^{\kappa \times \ell}$ from its challenger, generates all other matrices involved in Setup including $\hat{\mathbf{B}}_1^* \in \mathbb{F}^{2\kappa \times \kappa}$, and generates a commitment key ck which sends to $\mathcal{A}$. $\mathcal{B}$ receives $\boldsymbol{x}, \boldsymbol{x}'$ from $\mathcal{A}(\mathsf{ck})$ and outputs the vector $\boldsymbol{u} = \boldsymbol{x} - \boldsymbol{x}' \neq \mathbf{0}$. We argue that $\boldsymbol{u}$ is in the kernel of $\left[\hat{\mathbf{S}}_1\right]_2$. Let $\boldsymbol{v} = \hat{\mathbf{S}}_1 \cdot \boldsymbol{u}$. Note that $\hat{\mathbf{B}}_1^* \boldsymbol{v} = \mathbf{0}$ if and only if $\boldsymbol{v} = \mathbf{0}$, as $\hat{\mathbf{B}}_1^*$ has maximal rank κ. Therefore, as $\left[\hat{\mathbf{T}}\right]_2 \boldsymbol{u} = \left[\hat{\mathbf{B}}_1^*\right]_2 \boldsymbol{v} = [\mathbf{0}]_2$, we conclude that $\boldsymbol{v} = \mathbf{0}$, and $\mathcal{B}$ successfully wins the $\mathsf{KerDH}_{\kappa,\ell}$ game.

Lemma 7. se-PC *satisfies projection correctness (Definition 10).*

Proof. Let $\boldsymbol{x} \in \mathbb{F}^J$, $J \subseteq [\ell]$, $I \subseteq [\ell]$ and $\mathsf{ck} \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell, I)$. Then,

$$\begin{aligned}
\mathsf{Proj}(\mathsf{td}, \mathsf{Com}(\mathsf{ck}, J, \boldsymbol{x})) &= \hat{\mathbf{B}}_2 \cdot \left[\sum_{i \in J} \hat{\boldsymbol{t}}_i \cdot x_i\right]_1 = \hat{\mathbf{B}}_2 \cdot \left[\hat{\mathbf{T}} \cdot \boldsymbol{x}\right]_1 \\
&= \hat{\mathbf{B}}_2 \cdot \left[\hat{\mathbf{B}}_1^* \cdot \hat{\mathbf{S}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\mathbf{S}}_2 \cdot \mathbf{P}_I\right]_1 \cdot \boldsymbol{x} = \left[\hat{\mathbf{S}}_2 \cdot \mathbf{P}_I \cdot \boldsymbol{x}\right]_1 \\
&= \left[\sum_{i \in I \cap J} \hat{\boldsymbol{s}}_{2,i} \cdot x_i\right]_1.
\end{aligned}$$

Where the simplification follows by construction of the commitment key, as $\hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_2^* = \mathbf{I}_k$ and $\hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_1^* = \mathbf{0}_k$.

Lemma 8. *Assume that the* $\mathsf{MDDH}_{\kappa,\ell,2\kappa}$ *assumption holds over* bgp*. Then,* PCFC *satisfies setup indistinguishability (Definition 10).*

Proof. We define a series of game hops as follows. Note that we display each columnn vectors $\check{\boldsymbol{t}}_i, \hat{\boldsymbol{t}}_i$ of the matrices $\check{\mathbf{T}}, \hat{\mathbf{T}}$ separately.

– In Hyb^0, we have the key in normal mode, where

$$\hat{\boldsymbol{t}}_i \leftarrow \hat{\mathbf{B}}_1^* \hat{\boldsymbol{s}}_{1,i}, \quad \check{\boldsymbol{t}}_i \leftarrow \check{\mathbf{B}}_1^* \check{\boldsymbol{s}}_{1,i}.$$

$$\hat{\mathbf{W}}_{i,j} \leftarrow -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T, \quad \check{\mathbf{W}}_{i,j} \leftarrow \check{\mathbf{B}}_1^* (\check{\boldsymbol{s}}_{1,i} \hat{\boldsymbol{s}}_{1,j}^T + \mathbf{R}_{i,j})$$

– In Hyb^1, we change the distribution of $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$. Namely, we sample $\hat{\mathbf{S}}_2, \check{\mathbf{S}}_2 \leftarrow\!\!\$\ \mathbb{F}^{\kappa \times \ell}$ and set

$$\hat{\mathbf{W}}_{i,j} \leftarrow \begin{cases} -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T + \hat{\boldsymbol{t}}_i \check{\boldsymbol{s}}_{1,j}^T & \text{if } j \in I \\ -\hat{\mathbf{B}}_1^* \mathbf{R}_{i,j}^T & \text{if } j \notin I \end{cases}, \quad \check{\mathbf{W}}_{i,j} \leftarrow \begin{cases} \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} & \text{if } j \in I \\ \check{\mathbf{B}}_1^* \mathbf{R}_{i,j} + \check{\boldsymbol{t}}_i \hat{\boldsymbol{s}}_{1,j}^T & \text{if } j \notin I \end{cases}.$$

– In Hyb^2, we change the way that $\hat{\boldsymbol{t}}_i$ is sampled. Namely, we set $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$ as in Hyb^1, sample $\hat{\boldsymbol{u}}_i \leftarrow\!\!\$\ \mathbb{F}^{2\kappa}$, and set

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\boldsymbol{u}}_i & \text{if } i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if } i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i}$$

– In Hyb^3, we sample $\hat{\boldsymbol{t}}_i$ in trapdoor mode. Namely, we set $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$ as in Hyb^2, and then

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i} & \text{if } i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if } i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i}$$

– In Hyb^4, we change the way that $\check{\boldsymbol{t}}_i$ is sampled. Namely, we set $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$ as in Hyb^3, sample $\check{\boldsymbol{u}}_i \leftarrow\!\!\$\ \mathbb{F}^{2\kappa}$, and set

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i} & \text{if } i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if } i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \begin{cases} \check{\boldsymbol{u}}_i & \text{if } i \in I \\ \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} & \text{if } i \notin I \end{cases}$$

– In Hyb^5, we sample the full key in trapdoor mode. Namely, we set $\hat{\mathbf{W}}_{i,j}, \hat{\mathbf{W}}_{i,j}$ as in Hyb^4, and set

$$\hat{\boldsymbol{t}}_i \leftarrow \begin{cases} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i} & \text{if } i \in I \\ \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if } i \notin I \end{cases}, \quad \check{\boldsymbol{t}}_i \leftarrow \begin{cases} \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{s}}_{2,i} & \text{if } i \in I \\ \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} & \text{if } i \notin I \end{cases}$$

We proceed to bound the advantage between the different game hops.

$\mathsf{Hyb}^0 \to \mathsf{Hyb}^1$**:** . This is a syntactic relabeling that follows from generating $\mathbf{R}_{i,j}$ as follows: Sample $\bar{\mathbf{R}}_{i,j} \leftarrow\!\!\$\ \mathbb{F}^{\kappa \times \kappa}$, and let

$$\mathbf{R}_{i,j} \leftarrow \begin{cases} \bar{\mathbf{R}}_{i,j} - \check{\boldsymbol{s}}_{1,i} \hat{\boldsymbol{s}}_{1,j}^T & \text{if } j \in I \\ \bar{\mathbf{R}}_{i,j} & \text{if } j \notin I \end{cases}$$

Clearly, $\mathbf{R}_{i,j}$ still follows a uniform distribution over $\mathbb{F}^{\kappa \times \kappa}$, and so the view of the adversary in both games is identical. Therefore,

$$\mathsf{Adv}^0_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^1_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^1 \to \mathsf{Hyb}^2$**:** This follows by a sequence of $|I|$ sub-hybrids where we change each $\hat{\boldsymbol{t}}_j$ at a time. In each of the sub-hybrids, we carry out a reduction to $\mathsf{MDDH}_{\kappa,1,2\kappa}$ where the challenge matrix is $\left[\hat{\mathbf{B}}_1^*\right]_2$. We specify the first sub-hybrid, where we change the smallest index $i^* \in I$. Let $\mathcal{A}$ be an adversary against the first sub-hybrid.

- Let $\mathcal{B}$ be an adversary against $\mathsf{MDDH}_{\kappa,1,2\kappa}$ which receives a challenge matrix $\left[\hat{\mathbf{B}}_1^*\right]_2 \in \mathbb{G}_2^{2\kappa\times\kappa}$ and a challenge vector $[\boldsymbol{z}]_2 \in \mathbb{G}_2^{2\kappa}$.
- $\mathcal{B}$ generates a ck as follows (we specify only the modified terms):

$$\left[\hat{\boldsymbol{t}}_i\right]_2 \leftarrow \begin{cases} [\boldsymbol{z}]_2 & \text{if } i = i^* \\ \left[\hat{\mathbf{B}}_1^*\right]_2 \cdot \hat{\boldsymbol{s}}_{1,i} & \text{if } i \neq i^* \end{cases},$$

$$\left[\hat{\mathbf{W}}_{i,j}\right]_2 \leftarrow \begin{cases} -\left[\hat{\mathbf{B}}_1^*\right]_2 \mathbf{R}_{i,j}^T + \left[\hat{\boldsymbol{t}}_i\right]_2 \check{\boldsymbol{s}}_{1,j}^T & \text{if } j = i^* \\ -\left[\hat{\mathbf{B}}_1^*\right]_2 \mathbf{R}_{i,j}^T & \text{if } j \neq i^* \end{cases}.$$

  Note, in particular, that the terms $\hat{\boldsymbol{s}}_{1,j}$ are always used in $\hat{\boldsymbol{t}}_i$ for $j \notin I$, and so $\check{\mathbf{W}}_{i,j}$ does never change its distribution.
- The distribution of the key is identical to Hyb^1 whenever $\boldsymbol{z} = \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i^*}$ as $\hat{\boldsymbol{s}}_{1,i^*}$ is sampled uniformly at random. On the other hand, it is identical to Hyb^2 whenever $\boldsymbol{z}$ is uniform.

Hence, $\mathcal{B}$ simply gives the key to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. We conclude that

$$\mathsf{Adv}^1_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^2_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{mddh}}_{\mathcal{B}}(\lambda).$$

$\mathsf{Hyb}^2 \to \mathsf{Hyb}^3$**:** This is just a syntactic relabeling, as the distribution of $\left[\hat{\boldsymbol{t}}_i\right]_2$ is identical in both games. To see this, note that $\hat{\mathbf{B}}_1^*$ and $\hat{\mathbf{B}}_2^*$ together form a basis of the 2κ-dimensional vector space, and that $\hat{\boldsymbol{s}}_{1,i}, \hat{\boldsymbol{s}}_{2,i}$ are sampled at random. Hence,

$$\mathsf{Adv}^2_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^3_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^3 \to \mathsf{Hyb}^4$**:** The argument is the same as in between Hyb^1 and Hyb^2, so we omit it. It follows that

$$\mathsf{Adv}^3_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^4_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{mddh}}_{\mathcal{B}}(\lambda).$$

$\mathsf{Hyb}^4 \to \mathsf{Hyb}^5$**:** The argument is the same as in between Hyb^2 and Hyb^3, so we omit it. It follows that

$$\mathsf{Adv}^4_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^5_{\mathsf{se\text{-}PC},\mathcal{A}}(\lambda).$$

The lemma follows by collecting the different advantages. □

Lemma 9. $\mathsf{se\text{-}PC}$ *satisfies somewhere completeness (Definition 12).*

Proof. Let $I, J \subseteq [\ell]$ and $\boldsymbol{x} \in \{0,1\}^J$. Note that $x_i^2 = x_i$ for every $i \in J$. First, note that for any $i, j \in [\ell]$, the following equality holds

$$\check{\mathbf{B}}_1^* \cdot \hat{\mathbf{W}}_{i,j}^T + \check{\mathbf{W}}_{i,j} \cdot \hat{\mathbf{B}}_1^{*T} = -\check{\mathbf{B}}_1^* \cdot \mathbf{R}_{i,j} \cdot \hat{\mathbf{B}}_1^{*T} + \check{\mathbf{B}}_1^* \cdot (\check{\boldsymbol{s}}_{1,i}\hat{\boldsymbol{s}}_{1,j}^T + \mathbf{R}_{i,j}^T) \cdot \hat{\mathbf{B}}_1^{*T} = \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T$$

We start on the RHS of the first verification equation in the $\mathsf{ProjVer}$ algorithm.

$$
\begin{aligned}
\check{\boldsymbol{c}} \cdot \hat{\boldsymbol{c}}^T + \hat{\mathbf{B}}_1^* \cdot \hat{\mathbf{\Pi}}_1^T + \check{\mathbf{\Pi}}_1 \cdot \hat{\mathbf{B}}_1^{*T} &= \\
&= \sum_{i,j \in J} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i x_j + \sum_{i,j \in J, i \neq j} \left(\check{\mathbf{B}}_1^* \cdot \hat{\mathbf{W}}_{i,j}^T + \check{\mathbf{W}}_{i,j} \cdot \hat{\mathbf{B}}_1^{*T} \right) x_i (x_j - 1) \\
&= \sum_{i,j \in J} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i x_j + \sum_{i,j \in J, i \neq j} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i (x_j - 1) \\
&= \sum_{i \in J} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_i^T x_i^2 + \sum_{i,j \in J, i \neq j} \check{\boldsymbol{t}}_i \hat{\boldsymbol{t}}_j^T x_i \\
&= \left(\sum_{i \in J} \check{\boldsymbol{t}}_i x_i \right) \cdot \left(\sum_{j \in J} \hat{\boldsymbol{t}}_j^T \right) \\
&= \check{\boldsymbol{c}} \cdot \hat{\boldsymbol{t}}_J^T
\end{aligned}
$$

□

Correctness for the second verification equation follows analogously.

Lemma 10. $\mathsf{se\text{-}PC}$ *satisfies statistical somewhere extractability (Definition 12) for any sets* $I \in \mathcal{I}$, $J \in \mathcal{I}$ *such that* $|I \cap J| = 1$.

Proof. Let $I, J \subseteq [\ell]$ such that $I \cap J = \{i^*\}$. Our goal is to prove that the extraction is correct and matches the projection of com except with negligible probability against unbounded adversaries. The proof follows the blueprint of the proof of [WW22, Lemma 4.12], although the trapdoor setup in our construction differs. We start by proving the following lemma.

Lemma 11. *With respect to the matrices defined in* $\mathsf{ProjSetup}$, *any vector* $\boldsymbol{v} \in \mathbb{F}^{2\kappa}$ *can be* uniquely *written as*

$$
\boldsymbol{v} = \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2, \quad \textit{and} \quad \boldsymbol{v} = \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2
$$

for some vectors $\hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2, \check{\boldsymbol{u}}_1, \check{\boldsymbol{u}}_2 \in \mathbb{F}^{\kappa}$.

Proof. Note that $\hat{\mathbf{B}}^* = \left[\hat{\mathbf{B}}_1^* \, \hat{\mathbf{B}}_2^* \right]$ where $\hat{\mathbf{B}}_1^*, \hat{\mathbf{B}}_2^* \in \mathbb{F}^{2\kappa \times \kappa}$ form a basis of $\mathbb{F}^{2\kappa \times 2\kappa}$. Hence, any vector can be written uniquely as a combination of 2κ basis vectors, i.e.,

$$
\boldsymbol{v} = \left[\hat{\mathbf{B}}_1^* \, \hat{\mathbf{B}}_2^* \right] \cdot \begin{bmatrix} \hat{\boldsymbol{u}}_1 \\ \hat{\boldsymbol{u}}_2 \end{bmatrix}.
$$

And similarly for the basis formed by $\check{\mathbf{B}}_1^*, \check{\mathbf{B}}_2^*$. □

Let $\left([\check{\boldsymbol{c}}]_1, [\hat{\boldsymbol{c}}]_2, \left[\check{\mathbf{\Pi}}_1 \right]_1, \left[\hat{\mathbf{\Pi}}_1 \right]_2, \left[\check{\mathbf{\Pi}}_2 \right]_1, \left[\hat{\mathbf{\Pi}}_2 \right]_2 \right)$ be commitments and proof terms that pass the two verification equations. Following the lemma, let $\check{\boldsymbol{u}}_1, \check{\boldsymbol{u}}_2, \hat{\boldsymbol{u}}_1, \hat{\boldsymbol{u}}_2 \in \mathbb{F}^{\kappa}$ be the unique vectors such that $\check{\boldsymbol{c}} = \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2$ and $\hat{\boldsymbol{c}} = \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2$. When we rewrite the following terms that

appear in the verification relation in the above form, we obtain

$$\check{\boldsymbol{c}} \cdot \hat{\boldsymbol{t}}_J^T = \left(\check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2\right) \cdot \left(\sum_{i \in J} \hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{s}}_{1,i} + \sum_{i \in I \cap J} \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{s}}_{2,i}\right)^T,$$
$$\check{\boldsymbol{c}} \cdot \hat{\boldsymbol{c}}^T = \left(\check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{u}}_1 + \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{u}}_2\right) \cdot \left(\hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2\right)^T,$$
$$\check{\boldsymbol{t}}_J \cdot \hat{\boldsymbol{c}}^T = \left(\sum_{i \in J} \check{\mathbf{B}}_1^* \cdot \check{\boldsymbol{s}}_{1,i} + \sum_{i \in I \cap J} \check{\mathbf{B}}_2^* \cdot \check{\boldsymbol{s}}_{2,i}\right) \cdot \left(\hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2\right)^T.$$

Now, consider the two verification equations:

$$[\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{t}}_J^T\right]_2 = [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\boldsymbol{\Pi}}_1^T\right]_2 + \left[\check{\boldsymbol{\Pi}}_1\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2,$$
$$\left[\check{\boldsymbol{t}}_J\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 = [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 + \left[\check{\mathbf{B}}_1^*\right]_1 \cdot \left[\hat{\boldsymbol{\Pi}}_2^T\right]_2 + \left[\check{\boldsymbol{\Pi}}_2\right]_1 \cdot \left[\hat{\mathbf{B}}_1^{*T}\right]_2.$$

If we now project each term to the projective subspace by multiplying each term by $\check{\mathbf{B}}_2$ on the left and $\hat{\mathbf{B}}_2^T$ on the right, we get

$$\check{\mathbf{B}}_2 [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{t}}_J^T\right]_2 \hat{\mathbf{B}}_2^T = \check{\mathbf{B}}_2 [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 \hat{\mathbf{B}}_2^T,$$
$$\check{\mathbf{B}}_2 \left[\check{\boldsymbol{t}}_J\right]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 \hat{\mathbf{B}}_2^T = \check{\mathbf{B}}_2 [\check{\boldsymbol{c}}]_1 \cdot \left[\hat{\boldsymbol{c}}^T\right]_2 \hat{\mathbf{B}}_2^T.$$

Where we have used that $\check{\mathbf{B}}_2 \cdot \check{\mathbf{B}}_1^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_1^* = \mathbf{0}_k$, and so the terms with a $\mathbf{B}_1^*$-type matrix are canceled out. Plugging in the expressions for the remaining terms, and using that $I \cap J = \{i^*\}$, we derive

$$\check{\boldsymbol{u}}_2 \cdot \hat{\boldsymbol{s}}_{2,i^*}^T = \check{\boldsymbol{u}}_2 \cdot \hat{\boldsymbol{u}}_2^T,$$
$$\check{\boldsymbol{s}}_{2,i^*} \cdot \hat{\boldsymbol{u}}_2^T = \check{\boldsymbol{u}}_2 \cdot \hat{\boldsymbol{u}}_2^T,$$

Where we have used that $\check{\mathbf{B}}_2 \cdot \check{\mathbf{B}}_2^* = \hat{\mathbf{B}}_2 \cdot \hat{\mathbf{B}}_2^* = \mathbf{1}_k$. For the above equations to hold, we must have that:

$$\check{\boldsymbol{u}}_2 \cdot (\hat{\boldsymbol{s}}_{2,i^*}^T - \hat{\boldsymbol{u}}_2^T) = \mathbf{0}_k = (\check{\boldsymbol{s}}_{2,i^*} - \check{\boldsymbol{u}}_2) \cdot \hat{\boldsymbol{u}}_2^T.$$

We analyze the first equality. As these are outer products, we have that, coordinate-wise, $(\check{u}_2)_i \cdot ((\hat{s}_{2,i^*})_j - (\hat{u}_2)_j) = 0$ for every $i, j \in [\kappa]$. Hence, we conclude that either $\check{\boldsymbol{u}}_2 = \mathbf{0}$, or $\hat{\boldsymbol{u}}_2 = \hat{\boldsymbol{s}}_{2,i^*}$. A similar analysis can be carried out for the second equation, which allows us to conclude that $\check{\boldsymbol{u}}_2 = \check{\rho} \cdot \check{\boldsymbol{s}}_{2,i^*}$ and that $\hat{\boldsymbol{u}}_2 = \hat{\rho} \cdot \hat{\boldsymbol{s}}_{2,i^*}$ for some $\check{\rho}, \hat{\rho} \in \{0,1\}$.

To conclude the proof, note that the extractor runs $\left[\hat{\boldsymbol{d}}\right]_1 \leftarrow \mathsf{Proj}(\mathsf{td}, \mathsf{com})$, which by definition of Proj and by the previous argument, is

$$\begin{aligned}\left[\hat{\boldsymbol{d}}\right]_2 &= \hat{\mathbf{B}}_2 \cdot [\hat{\boldsymbol{c}}]_2 \\ &= \hat{\mathbf{B}}_2 \cdot \left[\hat{\mathbf{B}}_1^* \cdot \hat{\boldsymbol{u}}_1 + \hat{\mathbf{B}}_2^* \cdot \hat{\boldsymbol{u}}_2\right]_2 \\ &= [\hat{\boldsymbol{u}}_2]_2 = \hat{\rho} \cdot [\hat{\boldsymbol{s}}_{2,i^*}]_2\end{aligned}$$

Hence, by construction of the extractor, the extracted value $x_{i^*} = \hat{\rho} \in \{0,1\}$. □

Lemma 12. se-PC *satisfies aggregatability (Definition 6) and projective aggregatability (Definition 11).*

Proof. Both properties follow from the algebraic structure of the commitments, which are linearly homomorphic. The proof is straightforward and is omitted. □

Lemma 13. se-PC *satisfies efficient verification (Definition 13) where* EffProjVer *runs in time* $\mathcal{O}(\lambda \cdot \kappa^2)$.

Proof. Follows directly from the descriptions of the algorithms $\mathsf{PreProjVer}(\mathsf{ck}, J)$ and $\mathsf{EffProjVer}(\mathsf{ck}_J, \mathsf{com}, \pi)$. □

Remark 8. Given $\mathcal{I} = \{\{i\} : i \in [\ell]\}$, it is straightforward to extend se-PC to build a PCFC for $\mathcal{I}$-separable quadratic functions that achieves functional extractability (Definition 17). This is necessary to build a BARG via Theorem 4. The extension consists of adding CFC prove and verify algorithms that run as described in [WW22, Remark 4.16], and we refer to their work for the details. Then, the proof of functional extractability follows a similar pattern as the proof of Lemma 10.

6.2 Algebraic PCFC from Bilinear Groups

We construct a (somewhere extractable) projective chainable functional commitment PCFC (Definition 14) by augmenting the PC from the previous section with CFC algorithms. Our starting point, which we argue is the most natural choice given that it is the only fully-succinct algebraic FC in the literature, is the WW24 CFC on its chainable form [WW24, Remark 5.18]. Unfortunately, this scheme does not satisfy projective chain binding (Definition 18), which we require for our circuit-succinct compiler to a BARG.

The WW24 scheme has multiple internal algorithms for committing and proving relations on commitments. Our PCFC will also build on (two of) their algorithms, although we will introduce modifications towards achieving projective chain binding. We start with a description of the internals of WW24, where we already introduce some generalizations. We remark that these algorithms only serve the purpose of modularizing the scheme and the proofs. When the PCFC is used in a black-box way as in our compiler, they are hidden behind the PCFC interface.

Base commitment scheme. There are two families of setup algorithms in WW24: those that generate "base" commitment keys, and those that extend the commitment key with additional terms to enable a proof system. The so-called base setup algorithm $\mathsf{ck}_{\mathsf{base}} \leftarrow \mathsf{SetupBase}(1^\lambda, 1^\ell, 1^\ell_{\mathcal{C}})$ is parameterized by the maximum input length ℓ and the maximum circuit size $\ell_{\mathcal{C}}$. Such a setup supports arbitrary arithmetic circuits $\mathcal{C} : \mathbb{F}^\ell \to \mathbb{F}^\ell$ that have at most $\ell_{\mathcal{C}}$ arithmetic gates.[17] The algorithm generates two independent projective commitment keys. The type-I key $\left[\hat{\mathbf{T}}\right]_2 \in \mathbb{G}_2^{2\kappa \times \ell_{\mathcal{C}}}$ intentionally coincides with the primary commitment key of PC, and its terms are always represented with a hat accent. The type-II key is defined by two random matrices $\mathbf{T}_A, \mathbf{T}_B \leftarrow\!\!\$\ \mathbb{F}^{2\kappa \times \ell_{\mathcal{C}}}$. To commit to an input $\boldsymbol{x} \in \mathbb{F}^\ell$, and an output $\boldsymbol{y} \in \mathbb{F}^\ell$ via PCFC.Com, one uses the type-I key on the set $[\ell]$ or $[\ell]$, respectively. The type-II key is used internally to enable interactions between the functional proof algorithms.

[17] We remind that, without loss of generality and to simplify the notation, we assume that the input space and the output space are the same.

We remark that, similarly to our PC construction, both keys are projective. One can sample them in trapdoor mode by specifying two (independent) projection sets $I_1, I_2 \subseteq [\ell_{\mathcal{C}}]$ for the type-I and type-II keys, respectively.[18] We denote this operation by $(\mathsf{ck}_{\mathsf{base}}, \mathsf{td}) \leftarrow \mathsf{ProjSetupBase}(1^\lambda, 1^\ell, 1^\ell_{\mathcal{C}}, (I_1, I_2))$.

Functional proof systems. For functional openings, there are three proof systems that build on top of the type-I and type-II keys: prefix proofs (Pre), linear map proofs (Lin), and quadratic map proofs (Quad). We describe the last two in the generalized form we use in our construction, where the separability sets can be arbitrary and not necessarily prefixes. The prefix proof system is not needed in our construction, so we omit its details.

Lin: is a tuple of algorithms $(\mathsf{SetupLin}, \mathsf{OpenLin}, \mathsf{VerLin})$ that extend the base FC algorithms. $\mathsf{SetupLin}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S})$ extends the base commitment key, requiring a separability set $\mathcal{S} \subseteq 2^{[\ell_{\mathcal{C}}]\times[\ell_{\mathcal{C}}]}$ and outputting an extended ck. Let σ_1 be a type-I commitment to a vector $\boldsymbol{x} \in \mathbb{F}^\ell_{\mathcal{C}}$ and let σ_2 be a type-II commitment to $y \in \mathbb{F}^\ell_{\mathcal{C}}$. Let also $f : \mathbb{F}^\ell_{\mathcal{C}} \to \mathbb{F}^\ell_{\mathcal{C}}$ be a (I_1, I_2)-separable linear function such that $f(\boldsymbol{x}) = \boldsymbol{y}$. Then, $\mathsf{OpenLin}(\mathsf{ck}, \boldsymbol{x}, f)$ outputs a linear map proof π. The proof can be verified using $\mathsf{VerLin}(\mathsf{ck}, \sigma_1, \sigma_2, f, \pi)$. The proof system also admits algorithms $(\mathsf{PreVerLin}, \mathsf{EffVerLin})$ for efficient verification with pre-processing, which run in time $\mathcal{O}(\lambda \cdot \kappa^2)$.

Its associated security notion is called *linear chain binding* (Definition 19) and can be seen as a restriction of our projective chain binding (Definition 18) for linear functions.[19] Given a trapdoored setup at $(I_1, I_2) \in \mathcal{S}$, it is hard for any PPT adversary to output two input commitments σ_1, σ_1', output commitments σ_2, σ_2', a (I_1, I_2)-local function f and proofs π, π' such that (1) both proofs verify, (2) $\mathsf{Proj}(\mathsf{td}, \sigma_1) = \mathsf{Proj}(\mathsf{td}, \sigma_1')$, and (3) $\mathsf{Proj}(\mathsf{td}, \sigma_2) \neq \mathsf{Proj}(\mathsf{td}, \sigma_2')$.

Quad: is a tuple of algorithms $(\mathsf{SetupQuad}, \mathsf{OpenQuad}, \mathsf{VerQuad})$. Its properties are identical to those of Lin, except that it admits (I_1, I_2)-separable quadratic functions $f : \mathbb{F}^\ell_{\mathcal{C}} \to \mathbb{F}^\ell_{\mathcal{C}}$, and except that the roles of σ_1 and σ_2 are reversed. Its security notion, *quadratic chain binding* (Definition 20, which generalizes [WW24, Definition 4.35]), is analogous to linear chain binding except that it considers quadratic functions.

In Appendix C, we introduce an explanation of how the different proof systems compose to achieve evaluation binding. We next introduce the formal definitions of our generalizations of linear chain binding and quadratic chain binding, as well as the corresponding definitions of projective matrices that we use in our construction.

Generalization of chain binding properties. For our generalizations of linear chain binding and quadratic chain binding, we refer to dual-type keys generated by ProjSetupBase as described before, where σ_1 is a type-I commitment and σ_2 is a type-II commitment. We also allow the adversary to have the trapdoor of the scheme, which is needed in our security reductions. Note that the definitions of linear and quadratic chain binding differ only on the ordering of the commitments.

Definition 19 (Generalized Linear Chain Binding for Lin). *Let $\mathcal{S}_{\mathsf{Lin}} \subseteq 2^{[\ell_{\mathcal{C}}]\times[\ell_{\mathcal{C}}]}$ and let $\mathcal{F}^{\mathcal{S}}_{\mathsf{Lin}}$ be the family of linear functions $f : \mathbb{F}^\ell_{\mathcal{C}} \to \mathbb{F}^\ell_{\mathcal{C}}$ such that f is $\mathcal{S}_{\mathsf{Lin}}$-separable. A PC augmented with* Lin *satisfies generalized linear chain binding if for any PPT adversary $\mathcal{A}$, for any $\ell \in \mathbb{N}$, and for*

[18] In the original WW24 scheme, I_1, I_2 are always prefix sets $[j_1], [j_2]$ for $j_1, j_2 \leq \ell_{\mathcal{C}}$.

[19] This is a generalization of the original chain binding notion [WW24, Definition 4.20] where I_1, I_2 are required to be prefixes.

any $(I_1, I_2) \in \mathcal{S}_{\mathsf{Lin}}$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{Proj}(\mathsf{td}, \sigma_2) & (\mathsf{ck}_{\mathsf{base}}, \mathsf{td}) \leftarrow \mathsf{ProjSetupBase}(1^\lambda, 1^\ell_{\mathcal{C}}, (I_1, I_2)) \\ \neq \mathsf{Proj}(\mathsf{td}, \sigma_2') & \mathsf{ck} \leftarrow \mathsf{SetupLin}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Lin}}) \\ \wedge\ f \in \mathcal{F}^{\mathcal{S}}_{\mathsf{Lin}} & (f, \sigma_1, \sigma_1', \sigma_2, \sigma_2', \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}) \\ & \mathsf{VerLin}(\mathsf{ck}, \sigma_1, \sigma_2, f, \pi) = 1 \\ & \mathsf{VerLin}(\mathsf{ck}, \sigma_1', \sigma_2', f, \pi') = 1 \\ & \mathsf{Proj}(\mathsf{td}, \sigma_1) = \mathsf{Proj}(\mathsf{td}, \sigma_1') \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Definition 20 (Generalized Quadratic Chain Binding for Quad). *Let* $\mathcal{S}_{\mathsf{Quad}} \subseteq 2^{[\ell_{\mathcal{C}}]\times[\ell_{\mathcal{C}}]}$ *and let* $\mathcal{F}^{\mathcal{S}}_{\mathsf{Quad}}$ *be the family of quadratic functions* $f : \mathbb{F}^\ell_{\mathcal{C}} \to \mathbb{F}^\ell_{\mathcal{C}}$ *such that* f *is* $\mathcal{S}_{\mathsf{Quad}}$*-separable. A PC augmented with* Lin *satisfies generalized quadratic chain binding if for any PPT adversary* $\mathcal{A}$*, for any* $\ell \in \mathbb{N}$*, and for any* $(I_1, I_2) \in \mathcal{S}_{\mathsf{Quad}}$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{Proj}(\mathsf{td}, \sigma_1) & (\mathsf{ck}_{\mathsf{base}}, \mathsf{td}) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^\ell_{\mathcal{C}}, I_1, I_2) \\ \neq \mathsf{Proj}(\mathsf{td}, \sigma_1') & \mathsf{ck} \leftarrow \mathsf{SetupQuad}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Quad}}) \\ \wedge\ f \in \mathcal{F}^{\mathcal{S}}_{\mathsf{Quad}} & (f, \sigma_1, \sigma_1', \sigma_2, \sigma_2', \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}) \\ & \mathsf{VerQuad}(\mathsf{ck}, \sigma_2, \sigma_1, f, \pi) = 1 \\ & \mathsf{VerQuad}(\mathsf{ck}, \sigma_2', \sigma_1', f, \pi') = 1 \\ & \mathsf{Proj}(\mathsf{td}, \sigma_2) = \mathsf{Proj}(\mathsf{td}, \sigma_2') \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Projection matrices. We describe the projection matrices $\mathbf{P}^{(\mathcal{S}_{\mathsf{Lin}})}, \mathbf{P}^{(\mathcal{S}_{\mathsf{Quad}})}$ for separable linear and quadratic functions below, which we adapt from [WW24, Definitions 4.21, 4.36]. Recall that P_I are diagonal projection matrices such that $P_{I_{i,i}} = 1$ if $i \in I$ and $P_{I_{i,i}} = 0$ otherwise.

Definition 21 (Projection matrix for a separable linear function). *Let* $\ell_{\mathcal{C}} \in \mathbb{N}$ *be an input length. For sets* $I_1, I_2 \subseteq [\ell_{\mathcal{C}}]$*, we define the projection matrix* $\mathbf{P}^{(I_1,I_2)}_{\mathsf{Lin}}$ *to be*

$$\mathbf{P}^{(I_1,I_2)}_{\mathsf{Lin}} := \mathbf{I}_{\ell^2_{\mathcal{C}}} - (\mathbf{I}_{\ell_{\mathcal{C}}} - \mathbf{P}_{I_1}) \otimes \mathbf{P}_{I_2} \in \{0,1\}^{\ell^2_{\mathcal{C}} \times \ell^2_{\mathcal{C}}}.$$

For a separability set $\mathcal{S} \subseteq 2^{[\ell_{\mathcal{C}}]\times[\ell_{\mathcal{C}}]}$*, we define the linear projection matrix for* $\mathcal{S}$ *to be*

$$\mathbf{P}^{(\mathcal{S})} := \prod_{(I_1,I_2)\in\mathcal{S}} \mathbf{P}^{(I_1,I_2)}_{\mathsf{Lin}} \in \{0,1\}^{\ell^2_{\mathcal{C}} \times \ell^2_{\mathcal{C}}}.$$

Definition 22 (Projection matrix for a separable quadratic function). *Let* $\ell_{\mathcal{C}} \in \mathbb{N}$ *be an input length. For sets* $I_1, I_2 \in [\ell_{\mathcal{C}}]$*, we define the projection matrix* $\mathbf{P}^{(I_1,I_2)}_{\mathsf{Quad}}$ *to be*

$$\mathbf{P}^{(I_1,I_2)}_{\mathsf{Quad}} := \mathbf{I}_{\ell^3_{\mathcal{C}}} - (\mathbf{I}_{\ell^2_{\mathcal{C}}} - (\mathbf{P}_{I_1} \otimes \mathbf{P}_{I_1})) \otimes \mathbf{P}_{I_2} \in \{0,1\}^{\ell^3_{\mathcal{C}} \times \ell^3_{\mathcal{C}}}.$$

For a separability set $\mathcal{S} \subseteq 2^{[\ell_{\mathcal{C}}]\times[\ell_{\mathcal{C}}]}$*, we define the quadratic projection matrix for* $\mathcal{S}$ *to be*

$$\mathbf{P}^{(\mathcal{S})} := \prod_{(I_1,I_2)\in\mathcal{S}} \mathbf{P}^{(I_1,I_2)}_{\mathsf{Quad}} \in \{0,1\}^{\ell^3_{\mathcal{C}} \times \ell^3_{\mathcal{C}}}.$$

Our PCFC construction. Our PCFC is designed for the function family $\mathcal{F}_{k,\ell_{\mathcal{C}}}$ defined in Theorem 2. We additionally define ProjSetup that takes two arguments (I_0, I_1, I_2) – a necessity for granularity

in the projective chain binding proof. In our construction, the setup of $\mathsf{Lin}, \mathsf{Quad}$ is modified with respect to WW24 to express functions in $\mathcal{F}_{k,\ell_\mathcal{C}}$ accurately, such that there are no cross-wires between any two parallel "circuit blocks".

The construction essentially boils down to three commitment keys and four functional proofs (recall Figure 1 from the technical overview). The primary commitment key ck_0 is the same as in PC described in Section 6.1. Then, we rely on auxiliary commitment keys $\mathsf{ck}_1, \mathsf{ck}_2$ to realise the following functional proofs:

- A linear proof π_{in} for an input-projection map s that maps (the first ℓ coordinates of) the input commitment com under ck_0 to a commitment σ_2 under ck_2 to the circuit trace.
- A linear proof π_{Lin} between σ_1 (under ck_1) and σ_2 (under ck_2) for the identity function id.
- A quadratic proof π_{Quad} between σ_2 and σ_1 for a next-wire function w.
- A quadratic proof π_{out} for an output-projection map p that maps (the last ℓ coordinates of) σ_2 into (the first ℓ coordinates of) com_y.

$\underline{\mathsf{Setup}(1^\lambda, 1^{k\ell})}$**:**

- Generate $\mathsf{ck}_0 \leftarrow \mathsf{PC.Setup}(1^\lambda, 1^{k\ell_\mathcal{C}})$.
- Sample matrices $\mathbf{T}_1, \mathbf{T}_A, \mathbf{T}_B \leftarrow\!\!\$\ \mathbb{F}^{2\kappa \times \ell_\mathcal{C}}$ at random. Let $\mathbf{T}_* \leftarrow \mathbf{T}_A \otimes \mathbf{T}_B \in \mathbb{F}^{4\kappa^2 \times \ell_\mathcal{C}^2}$ and let $\mathsf{ck}_{\mathsf{base}} \leftarrow ([\mathbf{T_1}]_2, [\mathbf{T}_A]_1, [\mathbf{T}_A]_2, [\mathbf{T}_B]_2, [\mathbf{T}_*]_2)$
  For convenience, we name $\mathsf{ck}' \leftarrow \left([\hat{\mathbf{T}}]_2, [\mathbf{T}_A]_1, [\mathbf{T}_A]_2, [\mathbf{T}_B]_2, [\mathbf{T}_*]_2\right) \subseteq \mathsf{ck}$. Note that the first term is (the type-I key) taken from ck_0 and the remaining terms correspond to the type-II key from $\mathsf{ck}_{\mathsf{base}}$.
- Define the following sets and families of sets:

$$\begin{aligned} I_{i,j} &= \{(i-1)\ell_\mathcal{C} + 1, \ldots, (i-1)\ell_\mathcal{C} + j\}, \\ \mathcal{S}_{\mathsf{in}} &= \{(I_{i,j}, I_{i,j}) : i \in [k], j \in [\ell]\}, \\ \mathcal{S}_{\mathsf{Lin}} &= \{(I_{i,j}, I_{i,j}) : i \in [k], j \in [\ell_\mathcal{C}]\}, \\ \mathcal{S}_{\mathsf{Quad}} &= \{(I_{i,j}, I_{i,j+1}) : i \in [k], j \in [\ell_\mathcal{C} - 1]\}, \\ \mathcal{S}_{\mathsf{out}} &= \{(I_{i,j+\ell_\mathcal{C}-\ell}, I_{i,j}) : i \in [k], j \in [\ell]\}. \end{aligned}$$

- Run $\mathsf{SetupLin}(\mathsf{ck}', \mathcal{S}_{\mathsf{in}})$ as follows. Let $\mathbf{P}_{\mathsf{Lin}} = \mathbf{P}^{(\mathcal{S}_{\mathsf{Lin}})}$ defined as in Definition 21. Then, run $\mathsf{SetupLin}$ as defined in [WW24, Construction 4.23] with the above $\mathbf{P}_{\mathsf{Lin}}$. Obtain $\mathsf{ck}_{\mathsf{in}}$.
- Run $\mathsf{SetupLin}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Lin}})$ as before but over all terms on $\mathsf{ck}_{\mathsf{base}}$. Obtain $\mathsf{ck}_{\mathsf{Lin}}$.
- Run $\mathsf{SetupQuad}(\mathsf{ck}_{\mathsf{base}}, \mathcal{S}_{\mathsf{Quad}})$ as follows. Let $\mathbf{P}_{\mathsf{Quad}} = \mathbf{P}^{(\mathcal{S}_{\mathsf{Quad}})}$ defined as in Definition 22. Then, run $\mathsf{SetupQuad}$ as defined in [WW24, Construction 4.38] with the above $\mathbf{P}_{\mathsf{Quad}}$. Obtain $\mathsf{ck}_{\mathsf{Quad}}$.
- Finally, run $\mathsf{SetupQuad}(\mathsf{ck}', \mathcal{S}_{\mathsf{out}})$ (again over ck', the of terms from ck_0 and $\mathsf{ck}_{\mathsf{base}}$), Obtain $\mathsf{ck}_{\mathsf{out}}$.

Output

$$\mathsf{ck} := \mathsf{ck}_0 \cup \mathsf{ck}_{\mathsf{base}} \cup \mathsf{ck}_{\mathsf{in}} \cup \mathsf{ck}_{\mathsf{Lin}} \cup \mathsf{ck}_{\mathsf{Quad}} \cup \mathsf{ck}_{\mathsf{out}}.$$

$\underline{\mathsf{ProjSetup}(1^\lambda, 1^{k\ell}, I)}$**:** As $I \in \mathcal{I}'$ as defined in Theorem 2, we have that $I = \{i(\ell-1)+1, \ldots, i\ell\}$ for some $i \in [k]$.
Run $(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell}, (I_{i,\ell}, I_{i,\ell}, I_{i,\ell}))$ as defined below. Then, output $(\mathsf{ck}, \mathsf{td} = \mathsf{td}_0)$.

$\underline{\mathsf{ProjSetup}(1^\lambda, 1^{k\ell}, (I_0, I_1, I_2))}$**:**

- Run $(\mathsf{ck}_0, \mathsf{td}_0) \leftarrow \mathsf{PC.ProjSetup}(1^\lambda, 1^{k\ell_\mathcal{C}}, I_0)$.
- Sample random $\mathbf{B}_1, \mathbf{B}_A, \mathbf{B}_B \leftarrow\!\!\$\ \mathbb{F}^{2\kappa \times 2\kappa}$ and let $\mathbf{B}_A^* = \mathbf{B}_A^{-1}$, $\mathbf{B}_B^* = \mathbf{B}_B^{-1}$ and $\mathbf{B}_1^* = \mathbf{B}_1^{-1}$

– Split $\mathbf{B}_A = \begin{bmatrix}\mathbf{B}_{A,1}\\ \mathbf{B}_{A,2}\end{bmatrix}$ where $\mathbf{B}_{A,1}, \mathbf{B}_{A,2} \in \mathbb{F}^{\kappa\times 2\kappa}$. Similarly, split $\mathbf{B}^*_A = \begin{bmatrix}\mathbf{B}^*_{A,1}\ \mathbf{B}^*_{A,2}\end{bmatrix}$ where $\mathbf{B}^*_{A,1}, \mathbf{B}^*_{A,2} \in \mathbb{F}^{2\kappa\times\kappa}$. Note that, by construction, $\mathbf{B}_{A,1}\cdot\mathbf{B}^*_{A,1} = \mathbf{B}_{A,2}\cdot\mathbf{B}^*_{A,2} = \mathbf{I}_\kappa$ and that $\mathbf{B}_{A,1}\cdot\mathbf{B}^*_{A,2} = \mathbf{B}_{A,2}\cdot\mathbf{B}^*_{A,1} = \mathbf{0}_\kappa$. Define $\mathbf{B}_{1,1}, \mathbf{B}_{1,2}, \mathbf{B}_{B,1}, \mathbf{B}_{B,2} \in \mathbb{F}^{\kappa\times 2\kappa}$ and $\mathbf{B}^*_{1,1}, \mathbf{B}^*_{1,2}, \mathbf{B}^*_{B,1}, \mathbf{B}^*_{B,2} \in \mathbb{F}^{2\kappa\times\kappa}$ analogously.
– Sample $\mathbf{S}_{1,1}, \mathbf{S}_{A,1}, \mathbf{S}_{B,1}, \mathbf{S}_{1,2}, \mathbf{S}_{A,2}, \mathbf{S}_{B,2} \leftarrow\$ \mathbb{F}^{\kappa\times k\ell_{\mathcal{C}}}$. Let $\mathbf{P}_{I_1}, \mathbf{P}_{I_2}$ be the projection matrices for the sets I_1, I_2, respectively.

$$\begin{aligned}\mathbf{T}_1 &\leftarrow \mathbf{B}^*_{1,1}\cdot\mathbf{S}_{1,1} + \mathbf{B}^*_{1,2}\cdot\mathbf{S}_{1,2}\cdot\mathbf{P}_{I_1},\\ \mathbf{T}_A &\leftarrow \mathbf{B}^*_{A,1}\cdot\mathbf{S}_{A,1} + \mathbf{B}^*_{A,2}\cdot\mathbf{S}_{A,2}\cdot\mathbf{P}_{I_2},\\ \mathbf{T}_B &\leftarrow \mathbf{B}^*_{B,1}\cdot\mathbf{S}_{B,1} + \mathbf{B}^*_{B,2}\cdot\mathbf{S}_{B,2}\cdot\mathbf{P}_{I_2}.\end{aligned}$$

– Finally, let $\mathbf{T}_* \leftarrow \mathbf{T}_A \otimes \mathbf{T}_B \in \mathbb{F}^{4\kappa^2\times\ell_{\mathcal{C}}^2}$.

The remaining terms are computed from ck_0 and the above terms exactly as in Setup. We also define, as construction-specific trapdoors, $\mathsf{td}_1 \leftarrow \mathbf{B}_{1,2}$ and $\mathsf{td}_2 \leftarrow (\mathbf{B}_{A,2}, \mathbf{B}_{B,2})$. These trapdoors are linked to I_1, I_2 respectively and we make them explicit as they play a role in the security proof. However, they not used for extractability or projection when considering the scheme as a whole. Output the commitment key ck and trapdoors $(\mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2)$.

$\mathsf{FuncProve}(\mathsf{ck}, \boldsymbol{x}, f \in \mathcal{F}_{k,\ell_{\mathcal{C}}})$**:**

– Parse $f = (\mathcal{C}_1, \ldots, \mathcal{C}_k)$ and split $\boldsymbol{x} = (\boldsymbol{x}_i)_{i\in[k]}$. Evaluate $\boldsymbol{y}_i \leftarrow \mathcal{C}_i(\boldsymbol{x}_i)$ and let $\boldsymbol{z}_i \in \mathbb{F}^{\ell}_{\mathcal{C}}$ be the trace of the i-th computation, i.e., the values of all wires when $\mathcal{C}_i(\boldsymbol{x}_i)$ is evaluated. Note that $\boldsymbol{z}_i = \boldsymbol{x}_i|\boldsymbol{z}_{i,\mathcal{C}}|\boldsymbol{y}_i$ where $\boldsymbol{z}_{i,\mathcal{C}} \in \mathbb{F}^{\ell_{\mathcal{C}}-2\ell}$.
– Commit to the computation trace of $\boldsymbol{z} \leftarrow \boldsymbol{z}_1|\cdots|\boldsymbol{z}_k$ in the type-I and type-II keys,

$$\sigma_1 \leftarrow [\mathbf{T}_1]_2\,\boldsymbol{z}, \quad \sigma_2 \leftarrow ([\mathbf{T}_A]_1\,\boldsymbol{z}, [\mathbf{T}_B]_2\,\boldsymbol{z})\,.$$

– Let $s_i : \mathbb{F}^{\ell}_{\mathcal{C}} \to \mathbb{F}^{\ell}_{\mathcal{C}}$ be given by $s(\boldsymbol{z}) = (z_1, \ldots, z_\ell, \mathbf{0}_{\ell_{\mathcal{C}}-\ell})$ be a function that projects the first ℓ coordinates of $\boldsymbol{z}$ into the first ℓ coordinates. Define $s : \mathbb{F}^{k\ell_{\mathcal{C}}} \to \mathbb{F}^{k\ell_{\mathcal{C}}}$ to be the parallel evaluation of s_i, which is a $\mathcal{S}_{\mathsf{in}}$-separable linear function. Let now $\boldsymbol{z}_{\mathsf{in}} = \boldsymbol{x}_1|\mathbf{0}_{\ell_{\mathcal{C}}-\ell_{\mathtt{x},\mathtt{w}}}|\cdots|\boldsymbol{x}_k|\mathbf{0}_{\ell_{\mathcal{C}}-\ell_{\mathtt{x},\mathtt{w}}}$ be a concatenation of input vectors padded appropriately. Compute an input projection proof, given by $\pi_{\mathsf{in}} \leftarrow \mathsf{OpenLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \boldsymbol{z}_{\mathsf{in}}, s)$.
– Compute a linear proof for consistency between σ_1 and σ_2 given by $\pi_{\mathsf{Lin}} \leftarrow \mathsf{OpenLin}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Lin}}), \boldsymbol{z}, id)$ where $id : \mathbb{F}^{k\ell_{\mathcal{C}}} \to \mathbb{F}^{k\ell_{\mathcal{C}}}$ is the identity function. Note that id is $\mathcal{S}_{\mathsf{Lin}}$-separable.
– Let $w_i : \mathbb{F}^{\ell}_{\mathcal{C}} \to \mathbb{F}^{\ell}_{\mathcal{C}}$ be the "next wire function" of circuit $\mathcal{C}_i$, such that the j-th output of $w_i(\boldsymbol{z}_i)$ is the result of evaluating the j-th gate of $\mathcal{C}_i$ on the $j-1$ first inputs. Define $w : \mathbb{F}^{k\ell_{\mathcal{C}}} \to \mathbb{F}^{k\ell_{\mathcal{C}}}$ as $w(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k) = (w_1(\boldsymbol{z}_1), \ldots, w_k(\boldsymbol{z}_k))$. Note that w is a $\mathcal{S}_{\mathsf{Quad}}$-separable quadratic function. Compute a quadratic gate proof $\pi_{\mathsf{Quad}} \leftarrow \mathsf{OpenQuad}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Quad}}), \boldsymbol{z}, w)$.
– Let $p_i : \mathbb{F}^{\ell}_{\mathcal{C}} \to \mathbb{F}^{\ell}_{\mathcal{C}}$ be given by $p(\boldsymbol{z}) = (z_{\ell_{\mathcal{C}}-\ell+1}, \ldots, z_{\ell_{\mathcal{C}}}, \mathbf{0}_{\ell_{\mathcal{C}}-\ell})$ be a function that projects the last ℓ coordinates of $\boldsymbol{z}$ into the first ℓ coordinates. Define $p : \mathbb{F}^{k\ell_{\mathcal{C}}} \to \mathbb{F}^{k\ell_{\mathcal{C}}}$ to be the parallel evaluation of p_i. Note that p is a $\mathcal{S}_{\mathsf{out}}$-separable linear (and therefore also quadratic) function. Compute an output projection proof from the type, given by $\pi_{\mathsf{out}} \leftarrow \mathsf{OpenQuad}((\mathsf{ck}', \mathsf{ck}_{\mathsf{out}}), \boldsymbol{z}, p)$.

Output $\pi \leftarrow (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$.

$\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_y, f, \pi)$**:**

Parse $\pi = (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$ and output 1 if all the following checks pass:
– $\mathsf{VerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \mathsf{com}, \sigma_2, s, \pi_{\mathsf{in}})$.

– $\mathsf{VerLin}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Lin}}), \sigma_1, \sigma_2, id, \pi_{\mathsf{Lin}})$.
– $\mathsf{VerQuad}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Quad}}), \sigma_2, \sigma_1, w, \pi_{\mathsf{Quad}})$.
– $\mathsf{VerQuad}((\mathsf{ck}', \mathsf{ck}_{\mathsf{out}}), \sigma_2, \mathsf{com}_y, p, \pi_{\mathsf{out}})$.

$\mathsf{PreFuncVer}(\mathsf{ck}, f)$**:**

Run the preprocessing algorithms for s, id, w and p, respectively:
– $\mathsf{ck}_{f,\mathsf{in}} \leftarrow \mathsf{PreVerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), s)$.
– $\mathsf{ck}_{f,\mathsf{Lin}} \leftarrow \mathsf{PreVerLin}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Lin}}), id)$.
– $\mathsf{ck}_{f,\mathsf{Quad}} \leftarrow \mathsf{PreVerQuad}((\mathsf{ck}_{\mathsf{base}}, \mathsf{ck}_{\mathsf{Quad}}), w)$.
– $\mathsf{ck}_{f,\mathsf{out}} \leftarrow \mathsf{PreVerQuad}((\mathsf{ck}', \mathsf{ck}_{\mathsf{out}}), p)$.
Output $\mathsf{ck}_f \leftarrow (\mathsf{ck}_{f,\mathsf{in}}, \mathsf{ck}_{f,\mathsf{Lin}}, \mathsf{ck}_{f,\mathsf{Quad}}, \mathsf{ck}_{f,\mathsf{out}})$

$\mathsf{EffFuncVer}(\mathsf{ck}_f, \mathsf{com}, \mathsf{com}_y, \pi)$**:**

Parse $\pi = (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$ and output 1 if all the following checks pass:
– $\mathsf{EffVerLin}(\mathsf{ck}_{f,\mathsf{in}}, \mathsf{com}, \sigma_2, \pi_{\mathsf{in}})$.
– $\mathsf{EffVerLin}(\mathsf{ck}_{f,\mathsf{Lin}}, \sigma_1, \sigma_2, \pi_{\mathsf{Lin}})$.
– $\mathsf{EffVerQuad}(\mathsf{ck}_{f,\mathsf{Quad}}, \sigma_2, \sigma_1, \pi_{\mathsf{Quad}})$.
– $\mathsf{EffVerQuad}(\mathsf{ck}_{f,\mathsf{out}}, \sigma_2, \mathsf{com}_y, \pi_{\mathsf{out}})$.

Lemma 14. *Assume that the* $\mathsf{MDDH}_{\kappa,k\ell_{\mathcal{C}},2\kappa}$ *holds over* bgp*. Then,* PCFC *satisfies setup indistinguishability.*

Proof. First, note that $\mathsf{ck}_{\mathsf{base}}$ satisfies setup indistinguishability as proven in Lemma 8 and [WW24, Theorem 4.9]. Second, note that the extensions to the commitment key by the algorithms Lin and Quad can be sampled in a black-box way from $\mathsf{ck}_{\mathsf{base}}$ and do not vary in projective mode. Therefore, ck satisfies setup indistinguishability.

Lemma 15. PCFC *satisfies CFC correctness (Definition 7).*

Proof. The proof follows from [WW24] as we rely on the correctness of Lin and Quad, and our modifications on the projective matrices (Definitions 21 and 22) do not affect any of their correctness properties. We omit the details.

Lemma 16. PCFC *satisfies CFC efficient verification (Definition 9) where* EffFuncVer *runs in time* $\mathcal{O}(\lambda \cdot \kappa^2)$.

Proof. The property follows directly from the descriptions of $\mathsf{PreFuncVer}(\mathsf{ck}, J)$ and $\mathsf{EffFuncVer}(\mathsf{ck}_J, \mathsf{com}, \pi)$, as well as from the efficient verification properties of the Lin and Quad proof systems. □

Before proving projective chain binding, we state a short lemma about the individual proof systems.

Lemma 17. *Let* Lin *and* Quad*, be as defined in [WW24] and modified in our* PCFC *construction. Then,* Lin *satisfies generalized linear chain binding (Definition 19) and* Quad *satisfies generalized quadratic chain binding (Definition 20).*

Proof. The proof follows from [WW24] and from our definitions of projective matrices. The main difference with respect to the original notions is that we allow the adversary to have the trapdoor of the commitment keys. We argue that this does not invalidate the security of the proof systems, as every computational step in the original proofs of linear chain binding ([WW24, Theorem 4.25]) and quadratic chain binding ([WW24, Theorem 4.39]) relies on a MDDH-type assumption whose challenge is on a matrix that is crafted in a black-box way from the original commitment key ck_0. More in detail, we have that for the proof of linear chain binding:

– The first computational step (Lemma 4.28) uses $\mathbf{A}$ as a Ker-DH challenge, which is sampled uniformly as $\mathbf{A} \leftarrow \mathbb{F}^{\kappa \times (\kappa+1)}$.
– The second computational step (Lemma 4.31) uses $(\hat{\mathbf{S}}_2, \mathbf{Z}_\alpha)$ as a MDDH challenge. $\hat{\mathbf{S}}_2$ is not included in td, and $\mathbf{Z}_\alpha$ is sampled black-box from ck_0.

For the proof of quadratic chain binding:

– The first computational step (Lemma 4.43) uses $\mathbf{A}$ as a Ker-DH challenge, exactly as before.
– The second computational step (Lemma 4.46) uses $(\mathbf{S}_{B,2}, \mathbf{Z})$ as a MDDH challenge. As before, $\mathbf{S}_{2,2}$ is not included in td, and $\mathbf{Z}$ is sampled black-box from ck_0.

Theorem 3. *Assume that the* $\mathsf{KerDH}_{\kappa,k\ell_\mathcal{C}}$ *and the* $\mathsf{MDDH}_{\kappa,k\ell_\mathcal{C},2\kappa}$ *assumptions hold over* bgp. *Then, our construction* PCFC *satisfies projective chain binding for the class* $f \in \mathcal{F}_{k,\ell_\mathcal{C}}$ *(c.f. Theorem 2).*

Proof. We start by defining the projection sets that we will use in the proof

$$I_j = \{(i^* - 1)\ell_\mathcal{C} + 1, \ldots, (i^* - 1)\ell_\mathcal{C} + j\}.$$

Note that $|I_j| = j$. We also recall that $\pi = (\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}})$. We define a sequence of hybrid games in Figure 4, Figure 5. The initial hybrid game, $\mathsf{Hyb}^{\mathsf{pcb}}_\mathcal{A}(\lambda)$, is the projective chain binding game from Definition 18. Throughout the hybrid games, the adversary obtains the primary trapdoor td_0, but not the internal trapdoors $\mathsf{td}_1, \mathsf{td}_2$, which are only available to the challenger. This is crucial in the proof as if the adversary had $\mathsf{td}_1, \mathsf{td}_2$, setup indistinguishability for the corresponding subsets I_1, I_2 would not hold, and several game transitions rely on this property. On the other hand, the projection set I_0 is fixed throughout the entire sequence of games. We progress through the hybrids as follows.

$\mathsf{Hyb}^{\mathsf{pcb}}_\mathcal{A}(\lambda)$:	$\mathsf{Hyb}^{\mathsf{in}}_\mathcal{A}(\lambda)$:
$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_\mathcal{C}}, (I_\ell, I_\ell, I_\ell))$	// as $\mathsf{Hyb}^{\mathsf{pcb}}_\mathcal{A}(\lambda)$ until "checkpoint line"
$(f, \mathsf{com}, \mathsf{com}', \mathsf{com}_y, \mathsf{com}'_y, \pi, \pi') \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{td}_0)$	**assert** $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma'_2)$
$(\sigma_1, \sigma_2, \pi_{\mathsf{in}}, \pi_{\mathsf{Lin}}, \pi_{\mathsf{Quad}}, \pi_{\mathsf{out}}) \leftarrow \pi$	**return** 1
$(\sigma'_1, \sigma'_2, \pi'_{\mathsf{in}}, \pi'_{\mathsf{Lin}}, \pi'_{\mathsf{Quad}}, \pi'_{\mathsf{out}}) \leftarrow \pi'$	
assert $\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_y, f, \pi) = 1$	
assert $\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}', \mathsf{com}'_y, f, \pi') = 1$	
assert $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}) = \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}')$	
assert $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}'_y)$	
// Checkpoint line	
return 1	

Fig. 4: First set of games for the proof of Theorem 3. We highlight changes between games.

$\mathsf{Hyb}^{\mathsf{pcb}}_\mathcal{A}(\lambda) \to \mathsf{Hyb}^{\mathsf{in}}_\mathcal{A}(\lambda)$**:** This step follows by the linear binding property of Lin (Lemma 17) for the relation in, i.e., the identity map between the base key (ck_0) and the type-II key commitments to the internal circuit wires. To see this, note that $\mathsf{ck}_{\mathsf{in}}$ is sampled on $\mathcal{S}_{\mathsf{in}}$ and $(I_\ell, I_\ell) \in \mathcal{S}_{\mathsf{in}}$. Next, we

$\mathsf{Hyb}^{j,0}_{\mathcal{A}}(\lambda), \ell \le j \le \ell_{\mathcal{C}}$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_j, I_j))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma_2')$

return 1

$\mathsf{Hyb}^{j,1}_{\mathcal{A}}(\lambda), \ell \le j \le \ell_{\mathcal{C}} - 1$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_{j+1}, I_j))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma_2')$

return 1

$\mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda), \ell \le j \le \ell_{\mathcal{C}} - 1$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_{j+1}, I_j))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma_2')$

assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma_1')$

return 1

$\mathsf{Hyb}^{j,3}_{\mathcal{A}}(\lambda), \ell \le j \le \ell_{\mathcal{C}} - 1$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_{j+1}, I_j))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma_1')$

return 1

$\mathsf{Hyb}^{j,4}_{\mathcal{A}}(\lambda), \ell \le j \le \ell_{\mathcal{C}} - 1$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_{j+1}, I_{j+1}))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma_1')$

return 1

$\mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda), \ell \le j \le \ell_{\mathcal{C}} - 1$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_{j+1}, I_{j+1}))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma_2')$

assert $\mathsf{Proj}(\mathsf{td}_1, \sigma_1) = \mathsf{Proj}(\mathsf{td}_1, \sigma_1')$

return 1

$\mathsf{Hyb}^{\mathsf{out}0}_{\mathcal{A}}(\lambda)$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_\ell, I_{\ell_{\mathcal{C}}}))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma_2')$

return 1

$\mathsf{Hyb}^{\mathsf{out}1}_{\mathcal{A}}(\lambda)$:

$(\mathsf{ck}, \mathsf{td}_0, \mathsf{td}_1, \mathsf{td}_2) \leftarrow \mathsf{ProjSetup}(1^\lambda, 1^{k\ell_{\mathcal{C}}}, (I_\ell, I_\ell, I_{\ell_{\mathcal{C}}}))$

// then, as $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ until "checkpoint line"

assert $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) = \mathsf{Proj}(\mathsf{td}_2, \sigma_2')$

assert $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}_y) = \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}_y')$

return 1

Fig. 5: Second set of games for the proof of Theorem 3. We highlight changes between games.

provide the details of the reduction for this game hop — the remaining steps follow an identical pattern and we will only give a proof sketch.

Let $\mathcal{A}$ be a PPT adversary that wins $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ but does not win $\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$. Then, we can construct a PPT adversary $\mathcal{B}_{\mathsf{in}}$ that wins $\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$ as follows:

- $\mathcal{B}_{\mathsf{Lin}}$ receives the key $(\mathsf{ck}', \mathsf{ck}_{\mathsf{Lin}})$ and the trapdoor td_0.
- $\mathcal{B}_{\mathsf{Lin}}$ simulates the remaining part of the commitment key given $\mathsf{ck}', \mathsf{ck}_{\mathsf{Lin}}$ and generates a complete ck. Then, it calls $\mathcal{A}(\mathsf{ck}, \mathsf{td}_0)$ and receives the proofs π, π'. It parses the commitments $\mathsf{com}, \mathsf{com}', \sigma_2, \sigma_2'$ and the Lin proofs $\pi_{\mathsf{in}}, \pi_{\mathsf{in}}'$ from their respective proofs.
- $\mathcal{B}_{\mathsf{Lin}}$ simply outputs $(s, \mathsf{com}, \mathsf{com}', \sigma_2, \sigma_2', \pi_{\mathsf{in}}, \pi_{\mathsf{in}}')$ to its challenger, where s is the identity function for the first ℓ coordinates (clearly $s \in \mathcal{F}^{\mathcal{S}}_{\mathsf{in}}$).

It is evident that $\mathcal{B}_{\mathsf{Lin}}$ runs in polynomial time. If $\mathcal{A}$ wins in $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$ but not in $\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda)$, the commitments $\mathsf{com}, \mathsf{com}'$ and proofs $\pi_{\mathsf{in}}, \pi_{\mathsf{in}}'$ output by $\mathcal{A}$ must satisfy:

- $\mathsf{Proj}(\mathsf{td}_0, \mathsf{com}) = \mathsf{Proj}(\mathsf{td}_0, \mathsf{com}')$,
- $\mathsf{Proj}(\mathsf{td}_2, \sigma_2) \neq \mathsf{Proj}(\mathsf{td}_2, \sigma_2')$,
- $\mathsf{VerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \mathsf{com}, \sigma_2, s, \pi_{\mathsf{in}})$,
- $\mathsf{VerLin}((\mathsf{ck}', \mathsf{ck}_{\mathsf{in}}), \mathsf{com}', \sigma_2', s, \pi_{\mathsf{in}}')$.

Hence, $\mathcal{B}_{\mathsf{Lin}}$ wins the generalized linear binding game (Definition 19) if $\mathcal{A}$ is successful. We conclude:

$$\mathsf{Adv}^{0}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{in}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{lbind}}_{\mathsf{Lin},\mathcal{B}_{\mathsf{Lin}}}(\lambda).$$

$\mathsf{Hyb}^{\mathsf{in}}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{\ell,0}_{\mathcal{A}}(\lambda)$**:** This is only a renaming step; the games are identical for the case $j = \ell$ that we consider. Therefore,

$$\mathsf{Adv}^{\mathsf{in}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) = \mathsf{Adv}^{\ell,0}_{\mathsf{PCFC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^{j,0}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,1}_{\mathcal{A}}(\lambda)$**:** For any $j \in \{\ell, \ldots, \ell_{\mathcal{C}} - 1\}$, this step readily follows by the setup indistinguishability property of PCFC (Lemma 14). We have that, for some PPT adversary $\mathcal{B}_{\mathsf{sind}}$,

$$\mathsf{Adv}^{\mathrm{j},0}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{j},1}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{sbind}}_{\mathsf{PCFC},\mathcal{B}_{\mathsf{sind}}}(\lambda).$$

$\mathsf{Hyb}^{j,1}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda)$**:** For any $j \in \{\ell, \ldots, \ell_{\mathcal{C}} - 1\}$, this step follows by the generalized quadratic chain binding of Quad (Lemma 17), as $\mathsf{ck}_{\mathsf{Quad}}$ is sampled on $\mathcal{S}_{\mathsf{Quad}}$ and $(I_j, I_{j+1}) \in \mathcal{S}_{\mathsf{Quad}}$. We have that, for some PPT adversary $\mathcal{B}_{\mathsf{Quad}}$,

$$\mathsf{Adv}^{\mathrm{j},1}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{j},2}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{qbind}}_{\mathsf{Quad},\mathcal{B}_{\mathsf{Quad}}}(\lambda).$$

$\mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,3}_{\mathcal{A}}(\lambda)$**:** This step is a simplification of $\mathsf{Hyb}^{j,2}_{\mathcal{A}}(\lambda)$ where we drop a verification condition on σ_2. Therefore,

$$\mathsf{Adv}^{\mathrm{j},2}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{j},3}_{\mathsf{PCFC},\mathcal{A}}(\lambda).$$

$\mathsf{Hyb}^{j,3}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,4}_{\mathcal{A}}(\lambda)$**:** For any $j \in \{\ell, \ldots, \ell_{\mathcal{C}} - 1\}$, this step again follows by the setup indistinguishability property of PCFC (Lemma 14). We have that, for some PPT adversary $\mathcal{B}_{\mathsf{sind}}$,

$$\mathsf{Adv}^{\mathrm{j},3}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{j},4}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{sbind}}_{\mathsf{PCFC},\mathcal{B}_{\mathsf{sind}}}(\lambda).$$

$\mathsf{Hyb}^{j,4}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda)$**:** For any $j \in \{\ell, \ldots, \ell_{\mathcal{C}} - 1\}$, this step follows by the generalized linear chain binding of Lin (Lemma 17), as $\mathsf{ck}_{\mathsf{Lin}}$ is sampled on $\mathcal{S}_{\mathsf{Lin}}$ and $(I_j, I_j) \in \mathcal{S}_{\mathsf{Lin}}$. We have that, for some PPT adversary $\mathcal{B}_{\mathsf{Lin}}$,

$$\mathsf{Adv}^{\mathrm{j},4}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathrm{j},5}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{lbind}}_{\mathsf{Lin},\mathcal{B}_{\mathsf{Lin}}}(\lambda).$$

$\mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{j+1,0}_{\mathcal{A}}(\lambda)$**:** This step is a simplification of $\mathsf{Hyb}^{j,5}_{\mathcal{A}}(\lambda)$ where we drop a verification condition on σ_1. Therefore,

$$\mathsf{Adv}^{\mathsf{j},5}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{j}+1,0}_{\mathsf{PCFC},\mathcal{A}}(\lambda).$$

Next, we bound the final sequence of games, starting from $\mathsf{Hyb}^{\ell_{\mathcal{C}},0}_{\mathcal{A}}(\lambda)$.

$\mathsf{Hyb}^{\ell_{\mathcal{C}},0}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{\mathsf{out0}}_{\mathcal{A}}(\lambda)$**:** In this game, we change the extraction sets from $(I_\ell, I_{\ell_{\mathcal{C}}}, I_{\ell_{\mathcal{C}}})$ to $(I_\ell, I_\ell, I_{\ell_{\mathcal{C}}})$. Again, the step follows by the setup indistinguishability property of PCFC (Lemma 14). We have that, for some PPT adversary $\mathcal{B}_{\mathsf{sind}}$,

$$\mathsf{Adv}^{\ell_\rfloor,0}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{out0}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{sind}}_{\mathsf{PCFC},\mathcal{B}_{\mathsf{sind}}}(\lambda).$$

$\mathsf{Hyb}^{\mathsf{out0}}_{\mathcal{A}}(\lambda) \to \mathsf{Hyb}^{\mathsf{out1}}_{\mathcal{A}}(\lambda)$**:** Finally, this step follows by the generalized quadratic chain binding of Quad (Lemma 17) for the output vector, as $\mathsf{ck}_{\mathsf{out}}$ is sampled on $\mathcal{S}_{\mathsf{out}}$ and $(I_\ell, I_{\ell_{\mathcal{C}}}) \in \mathcal{S}_{\mathsf{out}}$. We have that, for some PPT adversary $\mathcal{B}_{\mathsf{Quad}}$,

$$\mathsf{Adv}^{\mathsf{out0}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{out1}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathsf{qbind}}_{\mathsf{Quad},\mathcal{B}_{\mathsf{Quad}}}(\lambda).$$

$\mathsf{Hyb}^{\mathsf{out1}}_{\mathcal{A}}(\lambda)$**:** In this game, we have the following contradicting conditions:
- $\mathsf{Proj}(\mathsf{td}_1, \mathsf{com}_y) \neq \mathsf{Proj}(\mathsf{td}_1, \mathsf{com}'_y)$ (present since $\mathsf{Hyb}^{\mathsf{pcb}}_{\mathcal{A}}(\lambda)$), and
- $\mathsf{Proj}(\mathsf{td}_1, \mathsf{com}_y) = \mathsf{Proj}(\mathsf{td}_1, \mathsf{com}'_y)$.

Hence, the advantage of any adversary in this game is $\mathsf{Adv}^{\mathsf{out1}}_{\mathsf{PCFC},\mathcal{A}}(\lambda) = 0$.

The theorem follows by collecting all the bounds and noting the relevant computational assumptions that are used at each step. □

Acknowledgements

This work was carried out while David Balbás was at IMDEA Software Institute.

We are thankful to Hoeteck Wee and David J. Wu for useful discussions and insights, and to anonymous reviewers for identifying a flaw in an earlier version of the paper.

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), partially supported by projects PRODIGY (TED2021-132464B-I00), ESPADA (PID2022-142290OB-I00), and CEX2024-001471-M funded by MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU / PRTR.

References

ABF24. Gaspard Anthoine, David Balbás, and Dario Fiore. Fully-succinct multi-key homomorphic signatures from standard assumptions. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 317–351. Springer, Cham, August 2024. doi:10.1007/978-3-031-68382-4_10.

ACG24. Abtin Afshar, Jiaqi Cheng, and Rishab Goyal. Leveled fully-homomorphic signatures from batch arguments. Cryptology ePrint Archive, Report 2024/931, 2024. URL: https://eprint.iacr.org/2024/931.

ACL+22. Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Lattice-based SNARKs: Publicly verifiable, preprocessing, and recursively composable - (extended abstract). In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 102–132. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_4.

BCFL23. David Balbás, Dario Catalano, Dario Fiore, and Russell W. F. Lai. Chainable functional commitments for unbounded-depth circuits. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023: 21st Theory of Cryptography Conference, Part III*, volume 14371 of *Lecture Notes in Computer Science*, pages 363–393. Springer, Cham, November / December 2023. doi:10.1007/978-3-031-48621-0_13.

BCJP24. Maya Farber Brodsky, Arka Rai Choudhuri, Abhishek Jain, and Omer Paneth. Monotone-policy aggregate signatures. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part IV*, volume 14654 of *Lecture Notes in Computer Science*, pages 168–195. Springer, Cham, May 2024. doi:10.1007/978-3-031-58737-5_7.

BHK17. Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th Annual ACM Symposium on Theory of Computing*, pages 474–482. ACM Press, June 2017. doi:10.1145/3055399.3055497.

CCH+19. Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1082–1090. ACM Press, June 2019. doi:10.1145/3313276.3316380.

CEW25. Binyi Chen, Noel Elias, and David J. Wu. Pairing-based batch arguments for NP with a linear-size CRS. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025, Part V*, volume 16249 of *Lecture Notes in Computer Science*, pages 554–577. Springer, Singapore, December 2025. doi:10.1007/978-981-95-5116-3_18.

CGH98. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218. ACM Press, May 1998. doi:10.1145/276698.276741.

CGJ+23. Arka Rai Choudhuri, Sanjam Garg, Abhishek Jain, Zhengzhong Jin, and Jiaheng Zhang. Correlation intractability and SNARGs from sub-exponential DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 635–668. Springer, Cham, August 2023. doi:10.1007/978-3-031-38551-3_20.

CJJ21. Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. Non-interactive batch arguments for NP from standard assumptions. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 394–423, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84259-8_14.

CJJ22. Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd Annual Symposium on Foundations of Computer Science*, pages 68–79. IEEE Computer Society Press, February 2022. doi:10.1109/FOCS52979.2021.00016.

dCP23. Leo de Castro and Chris Peikert. Functional commitments for all functions, with transparent setup and from SIS. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 287–320. Springer, Cham, April 2023. doi:10.1007/978-3-031-30620-4_10.

DGKV22. Lalita Devadas, Rishab Goyal, Yael Kalai, and Vinod Vaikuntanathan. Rate-1 non-interactive arguments for batch-NP and applications. In *63rd Annual Symposium on Foundations of Computer Science*, pages 1057–1068. IEEE Computer Society Press, October / November 2022. doi:10.1109/FOCS54457.2022.00103.

DJWW25. Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025, Part VIII*, volume 16252 of *Lecture Notes in Computer Science*, pages 130–162. Springer, Singapore, December 2025. doi:10.1007/978-981-95-5125-5_5.

EHK+13. Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 129–147. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_8.

GLWW24. Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 143–177. Springer, Cham, August 2024. doi:10.1007/978-3-031-68382-4_5.

Goy24. Rishab Goyal. Mutable batch arguments and applications. Cryptology ePrint Archive, Report 2024/737, 2024. URL: https://eprint.iacr.org/2024/737.

GSWW22. Rachit Garg, Kristin Sheridan, Brent Waters, and David J. Wu. Fully succinct batch arguments for np from indistinguishability obfuscation. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 526–555. Springer, Cham, November 2022. doi:10.1007/978-3-031-22318-1_19.

GW11. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 99–108. ACM Press, June 2011. doi:10.1145/1993636.1993651.

GZ21. Alonso González and Alexandros Zacharakis. Fully-succinct publicly verifiable delegation from constant-size assumptions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 529–557. Springer, Cham, November 2021. doi:10.1007/978-3-030-90459-3_18.

HW15. Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science*, pages 163–172. Association for Computing Machinery, January 2015. doi:10.1145/2688073.2688105.

JJ21. Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 3–32. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_1.

JKKZ21. Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Yun Zhang. SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 708–721. ACM Press, June 2021. doi:10.1145/3406325.3451055.

KLVW23. Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th Annual ACM Symposium on Theory of Computing*, pages 1545–1552. ACM Press, June 2023. doi:10.1145/3564246.3585200.

KPY19. Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1115–1124. ACM Press, June 2019. doi:10.1145/3313276.3316411.

KVZ21. Yael Tauman Kalai, Vinod Vaikuntanathan, and Rachel Yun Zhang. Somewhere statistical soundness, post-quantum security, and SNARGs. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 330–368. Springer, Cham, November 2021. doi:10.1007/978-3-030-90459-3_12.

LRY16. Benoît Libert, Somindu C. Ramanna, and Moti Yung. Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016: 43rd International Colloquium on Automata, Languages and Programming*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:14. Schloss Dagstuhl, July 2016. doi:10.4230/LIPIcs.ICALP.2016.30.

Mic94. Silvio Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453. IEEE Computer Society Press, November 1994. doi:10.1109/SFCS.1994.365746.

MRV16. Paz Morillo, Carla Ràfols, and Jorge Luis Villar. The kernel matrix Diffie-Hellman assumption. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 729–758. Springer, Berlin, Heidelberg, December 2016. doi:10.1007/978-3-662-53887-6_27.

PP22. Omer Paneth and Rafael Pass. Incrementally verifiable computation via rate-1 batch arguments. In *63rd Annual Symposium on Foundations of Computer Science*, pages 1045–1056. IEEE Computer Society Press, October / November 2022. doi:10.1109/FOCS54457.2022.00102.

PS19. Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 89–114. Springer, Cham, August 2019. doi:10.1007/978-3-030-26948-7_4.

WW22. Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 433–463. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_15.

WW23a. Hoeteck Wee and David J. Wu. Lattice-based functional commitments: Fast verification and cryptanalysis. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part V*, volume

14442 of *Lecture Notes in Computer Science*, pages 201–235. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8733-7_7.

WW23b. Hoeteck Wee and David J. Wu. Succinct vector, polynomial, and functional commitments from lattices. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 385–416. Springer, Cham, April 2023. doi:10.1007/978-3-031-30620-4_13.

WW24. Hoeteck Wee and David J. Wu. Succinct functional commitments for circuits from k-Lin. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part II*, volume 14652 of *Lecture Notes in Computer Science*, pages 280–310. Springer, Cham, May 2024. doi:10.1007/978-3-031-58723-8_10.

A Wire-by-Wire Compiler: From Functional Extractability to BARGs

We provide a generic compiler to build a BARG for NP from somewhere extractable projective chainable funcitonal commitments (se-PCFC), as defined in Section 4. Our compiler can be seen as an abstraction of the construction by Waters and Wu [WW22]. The resulting BARG can evaluate circuits $\mathcal{C} : \bar{\mathcal{M}}^{\ell} \to \bar{\mathcal{M}}$ with arbitrary gates, proving that $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i) = 1$ for every $i \in [k]$. Notably, the compiler only makes black-box use of the underlying PCFC. Therefore, if the PCFC is algebraic, so does the resulting BARG.

A.1 Compiler construction

Our compiler is described in Figure 6. Given a quadratic gate function $\chi : \bar{\mathcal{M}}^{m} \to \bar{\mathcal{M}}$ for (arithmetic or Boolean) circuits $\mathcal{C} : \bar{\mathcal{M}}^{\ell_{\mathtt{x}},\mathtt{w}} \to \bar{\mathcal{M}}$, we define $f_\chi : \bar{\mathcal{M}}^{k\cdot m} \to \bar{\mathcal{M}}^{k}$ as $f_\chi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = (\chi(\boldsymbol{x}_1), \ldots, \chi(\boldsymbol{x}_k))$. Namely, f_χ is a k-separable function that runs χ in parallel k times, and such that $m < \ell_{\mathcal{C}}$. The compiler turns a PCFC which 1) supports the functions f_χ, 2) is somewhere extractable and 3) is functional extractable into a BARG for NP. The formal requirements and implication are stated in the following Theorem 4. Note that in our construction the crs is independent of the circuit size $\ell_{\mathcal{C}}$.

Theorem 4 (Basic Compiler). *Let $\mathcal{I} := \{\{i\} : i \in [k]\}$ (i.e. singletons of indices in $[k]$), $\bar{\mathcal{M}} \supseteq \{0,1\}$ be some message space, PCFC be a projective chainable functional commitment scheme (Definition 14) for the $\mathcal{I}$-separable functions $\mathcal{F} = \{f_\chi : \bar{\mathcal{M}}^{k\cdot m} \to \bar{\mathcal{M}}^{k} : \chi$ is a gate function$\}$ which satisfies*

- *(Definition 12) somewhere extractability for $(\mathcal{I}, \bar{\mathcal{M}})$ and*
- *(Definition 17) functional extractability for $(\mathcal{F}, \mathcal{I})$.*

Then, the construction BARG in Figure 6 is a somewhere extractable batch argument (Definition 1) for NP whose proof size is $|\pi| = \ell_{\mathcal{C}} \cdot |\mathsf{com}| + \ell_{\mathtt{w}} \cdot |\pi^b| + (\ell_{\mathcal{C}} - \ell_{\mathtt{x},\mathtt{w}}) \cdot |\pi^\chi|$ where π^b is a PC subspace proof, and π^χ is a PCFC functional proof for $f \in \mathcal{F}$. Moreover, BARG admits efficient verification with preprocessing (Definition 2).

Proof. The completeness, succinctness and setup indistinguishability of BARG follow by inspection, given the correctness, succinctness and setup indistinguishability of PCFC. The somewhere extractability of BARG follows from Lemma 18. Its efficient verification property follows from Lemma 21. □

In the remainder of the section, we prove somewhere extractability in Appendix A.2 and efficient verification in Appendix A.3.

A.2 Proof of somewhere extractability

Lemma 18. *Let PCFC and BARG be as defined in Theorem 4. It holds that BARG has somewhere extractability.*

Proof. We proceed by a series of hybrid games $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,0}, \ldots, \mathsf{Hyb}^{1,\ell_{\mathtt{w}}}, \mathsf{Hyb}^{2,0}, \ldots, \mathsf{Hyb}^{2,\ell_{\mathsf{out}}}$ which progressively include additional winning conditions. The formal descriptions of the games are given in Figure 7 with explanations given below.

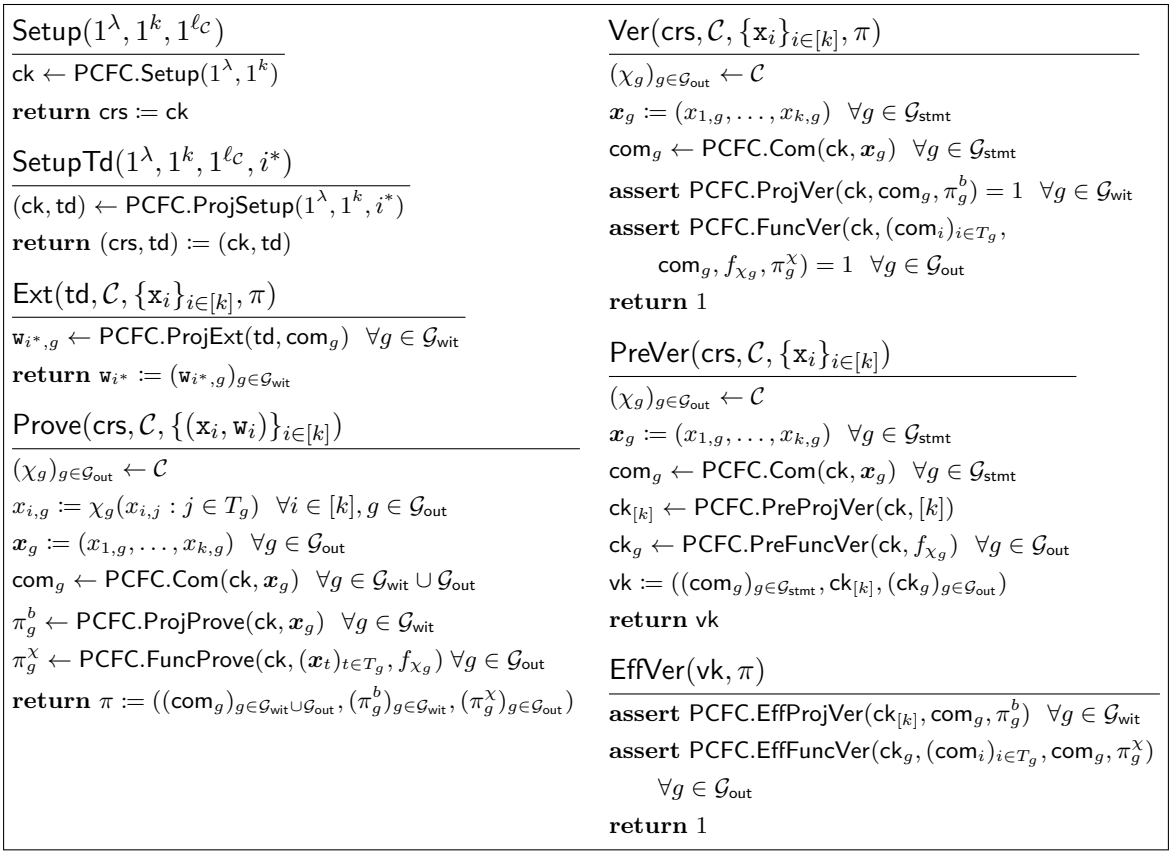


Fig. 6: Basic Compiler from PCFC to BARG.

Hyb^0**:** This is the standard somewhere extractability game (Definition 1) with the algorithms in BARG inlined. To recall, in this game the adversary outputs a proof π and the experiment runs $\mathtt{w}_{i^*} \leftarrow \mathsf{BARG.Ext}(\mathsf{td}, \mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi)$. The adversary wins if the proof verifies yet $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}_{i^*}) \neq 1$.

$\mathsf{Hyb}^{1,j}$**:** $\mathsf{Hyb}^{1,0}$ is identical to Hyb^0. In $\mathsf{Hyb}^{1,j}$, the game additionally checks that the first j entries of the extracted witness, i.e. those indexed by g belonging to the first j values in $\mathcal{G}_{\mathsf{wit}}$, resides in the message subspace $\bar{\mathcal{M}}$, i.e. $x^*_g \in \bar{\mathcal{M}}^{\ell_\mathtt{w}}$, and that they are consistent with the projected commitments, i.e. $\mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_g)$.

$\mathsf{Hyb}^{2,j}$**:** $\mathsf{Hyb}^{2,0}$ is identical to $\mathsf{Hyb}^{1,\ell_\mathtt{w}}$. To define the rest of the games, we let $\boldsymbol{x}^* \in \bar{\mathcal{M}}^{\ell_\mathcal{C}}$ be the (honestly computed) values of the circuit wires of instance i^* that result from the evaluation of $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}^*)$. Note that $\boldsymbol{x}^*$ is guaranteed to satisfy $\mathtt{w}^* \in \bar{\mathcal{M}}^{\ell_\mathtt{w}}$ by the winning condition. In $\mathsf{Hyb}^{2,j}$, the game additionally checks that $x^*_g = \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_g)$ for all g belonging to the first j values in $\mathcal{G}_{\mathsf{out}}$.

We now proceed to bound the advantage of any PPT adversary $\mathcal{A}$ against somewhere extractability, i.e. Hyb^0, by a series of lemmas. We remark that the proof also works for unbounded adversaries (proving statistical somewhere extractability of BARG) if PCFC satisfies (1) statistical functional somewhere extractability and (2) statistical projective somewhere extractability.

$\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$:

$(\mathsf{ck}, \mathsf{td}_{i^*}) \leftarrow \mathsf{PCFC.ProjSetup}(1^\lambda, 1^k, i^*)$

$(\mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \leftarrow \mathcal{A}(\mathsf{ck})$

$\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (x_{1,g}, \ldots, x_{k,g}))\ \ \forall g \in \mathcal{G}_{\mathsf{stmt}}$

$x^*_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}, \mathsf{com}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}}$

$\mathtt{w}^* := (x^*_g)_{g\in\mathcal{G}_{\mathsf{wit}}}$

assert $\mathsf{com}_{\ell_{\mathcal{C}}} = \mathsf{PCFC.Com}(\mathsf{ck}, \mathbf{1}_k)$

assert $\mathsf{PCFC.ProjVer}(\mathsf{ck}, \mathsf{com}_g, \pi^b_g) = 1\ \ \forall\, g \in \mathcal{G}_{\mathsf{wit}}$

assert $\mathsf{PCFC.FuncVer}(\mathsf{ck}, (\mathsf{com}_i)_{i\in T_g}, \mathsf{com}_g, f_{\chi_g}, \pi^\chi_g) = 1\ \ \forall\, g \in \mathcal{G}_{\mathsf{out}}$

assert $\mathcal{C}(\mathtt{x}_{i^*}, \mathtt{w}^*) \neq 1$

return 1

$\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathtt{w}}$:

// identical to $\mathsf{Hyb}^0_{\mathcal{A}}(\lambda)$ until before "**return** 1"

assert $x^*_g \in \bar{\mathcal{M}}\ \ \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$

assert $\mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_g) = \mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{wit}} \cap [\ell_{\mathtt{x}} + j]$

return 1

$\mathsf{Hyb}^{2,j}_{\mathcal{A}}(\lambda), 0 \le j \le \ell_{\mathsf{out}}$:

// identical to $\mathsf{Hyb}^{1,\ell_{\mathtt{w}}}_{\mathcal{A}}(\lambda)$ until before "**return** 1"

$\boldsymbol{x}^* \leftarrow \mathsf{Wires}(\mathcal{C}, \mathtt{x}_{i^*}, \mathtt{w}^*)$

assert $x^*_g = \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_g)\ \ \forall g \in \mathcal{G}_{\mathsf{out}} \cap [\ell_{\mathtt{x},\mathtt{w}} + j]$

return 1

Fig. 7: Games $\mathsf{Hyb}^0, \mathsf{Hyb}^{1,j}, \mathsf{Hyb}^{2,j}$ for the proof of BARG somewhere extractability. We highlight changes between games. We define the function $\boldsymbol{x} \leftarrow \mathsf{Wires}(\mathcal{C}, \mathtt{x}_i, \mathtt{w}_i)$ to output the value of all the internal values of a circuit computation $\mathcal{C}(\mathtt{x}_i, \mathtt{w}_i)$. Namely, $x_j \leftarrow \chi_j((x_t)_{t\in T_j})$ where the initial values are given by $\boldsymbol{x} = \mathtt{x}_i|\mathtt{w}_i|\mathbf{0}$ and $\boldsymbol{x}$ is recomputed iteratively $\forall j \in [N]$.

Lemma 19. *For any $j \in [\ell_{\mathtt{w}}]$, there exists a PPT adversary $\mathcal{B}$ against the somewhere extractability for $(\mathcal{I}, \bar{\mathcal{M}})$ of* PCFC *such that*

$$\left|\Pr[\mathsf{Hyb}^{1,j-1}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}^{1,j}_{\mathcal{A}}(\lambda) = 1]\right| \le \mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. Fix $j \in [\ell_{\mathtt{w}}]$. We build an adversary $\mathcal{B}$ as follows. $\mathcal{B}$ receives input ck from its challenger and runs $\mathcal{A}(\mathsf{ck})$. Then, it parses the proof π of $\mathcal{A}$ and retrieves the tuple $(\mathsf{com}_{\ell_{\mathtt{x}}+j}, \pi^b_{\ell_{\mathtt{x}}+j})$. Finally, $\mathcal{B}$ simply forwards these to its challenger.

We argue that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j-1}$ but not in $\mathsf{Hyb}^{1,j}$, then $\mathcal{B}$ also wins the game. First, as $\mathcal{A}$ wins in $\mathsf{Hyb}^{1,j}$, it holds that $\mathsf{PCFC.ProjVer}(\mathsf{ck}, \mathsf{com}_{\ell_{\mathtt{x}}+j}, \pi^b_{\ell_{\mathtt{x}}+j}) = 1$. Also, note that $x^*_{\ell_{\mathtt{x}}+j} \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathtt{x}}+j})$. Therefore, all the preconditions for $\mathcal{B}$ in the PC somewhere extractability game are met. Finally, as $\mathcal{A}$ does not win in $\mathsf{Hyb}^{1,j}$, it must be that either $\mathsf{PCFC.Proj}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathtt{x}}+j}) \neq \mathsf{PCFC.ProjCom}(\mathsf{td}_{i^*}, x^*_{\ell_{\mathtt{x}}+j})$ or $x^*_{\ell_{\mathtt{x}}+j} \notin \bar{\mathcal{M}}$. □

Lemma 20. *For any $j \in [\ell_{\mathsf{out}}]$, there exists a PPT adversary $\mathcal{B}$ against the functional extractability for $(\mathcal{F}, \mathcal{I})$ of* PCFC *such that*

$$\left|\Pr[\mathsf{Hyb}^{2,j-1}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Hyb}^{2,j}_{\mathcal{A}}(\lambda) = 1]\right| \leq \mathsf{Adv}^{\mathrm{fe}}_{\mathsf{PCFC},\mathcal{B}}(\lambda).$$

Proof. Fix $j \in [\ell_{\mathsf{out}}]$. We build $\mathcal{B}$ from $\mathcal{A}$ as follows:

- Receives input ck from its challenger and runs $(\mathcal{C}, \{\mathtt{x}_i\}_{i\in[k]}, \pi) \leftarrow \mathcal{A}(\mathsf{ck})$.
- Parses $\mathcal{C}$ to obtain the gate function χ_j and the input wire set T_j.
- Parses the proof π and obtains $\{\mathsf{com}_g\}_{g\in T_j \setminus [\ell_{\mathtt{x}}]}, \pi^{\chi}_j, \mathsf{com}_j$.
- Outputs $(f_{\chi_j}, \{\mathsf{com}_g\}_{g\in T_j}, \mathsf{com}_j, \pi^{\chi}_j)$ where the com_g for $g \in \mathcal{G}_{\mathsf{stmt}} \cap T_g$ are computed honestly as $\mathsf{com}_g \leftarrow \mathsf{PCFC.Com}(\mathsf{ck}, (x_{1,g}, \ldots, x_{k,g}))$.

We argue that if $\mathcal{A}$ wins in $\mathsf{Hyb}^{2,j-1}$ but not in $\mathsf{Hyb}^{2,j}$, then $\mathcal{B}$ is a successful adversary against CFC functional extractability.

- By the winning condition (inherited from Hyb^0), it holds that the functional proof verifies, $\mathsf{FuncVer}(\mathsf{ck}, (\mathsf{com}_g)_{g\in T_j}, f_{\chi_j}, \mathsf{com}_j, \pi^{\chi}_j) = 1$.
- Let $\tilde{x}_g \leftarrow \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_g)$ be the value extracted from the commitments for any $g \in \mathcal{G}$.
- As $\mathcal{A}$ wins in $\mathsf{Hyb}^{2,j-1}$, we have that $\tilde{x}_g = x^*_g = \chi_j((x^*_g)_{g\in T_j})$ for every $g \in [\ell_{\mathtt{x},\mathtt{w}} + j - 1]$, and therefore for every $g \in T_j$. By the definition of $\boldsymbol{x}^*$, it also follows that $\tilde{x}_g \in \bar{\mathcal{M}}$ for every $g \in T_j$.
- As $\mathcal{A}$ does not win in $\mathsf{Hyb}^{2,j}$, then $\tilde{x}_j \neq x^*_j$.

Hence, $\chi_j((\tilde{x}_g)_{g\in T_j}) \neq \tilde{x}_j$ and the output of $\mathcal{B}$ is a successful break of CFC functional extractability. □

Note that the advantage of $\mathcal{A}$ in $\mathsf{Hyb}^{2,\ell_{\mathsf{out}}}$ is zero, as the condition $\mathcal{C}(\mathtt{x}_{i^*}, w^*) \neq 1$ is incompatible with $x^*_{\ell_{\mathcal{C}}} = \mathsf{PCFC.ProjExt}(\mathsf{td}_{i^*}, \mathsf{com}_{\ell_{\mathcal{C}}}) = 1$, which holds as $\mathsf{com}_{\ell_{\mathcal{C}}}$ is computed honestly by the verifier. We collect the bounds of Lemma 19 and Lemma 20, and conclude that the advantage of $\mathcal{A}$ in the original Hyb^0 is bounded by

$$\Pr[\mathsf{Hyb}^0_{\mathcal{A}}(\lambda) = 1] \leq \ell_{\mathtt{w}} \cdot \mathsf{Adv}^{\mathrm{se}}_{\mathsf{PCFC},\mathcal{B}}(\lambda) + \ell_{\mathsf{out}} \cdot \mathsf{Adv}^{\mathrm{fe}}_{\mathsf{PCFC},\mathcal{B}}(\lambda). \qquad \square$$

A.3 Proof of efficient verification

Lemma 21. *Let* PCFC *and* BARG *be as defined in Theorem 4. If further* PCFC *has efficient verification with preprocessing (Definition 15), then* BARG *has efficient verification with preprocessing (Definition 2).*

Proof. As per Definition 2, it suffices to show that $|\mathsf{vk}|$ and the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ are at most $\mathsf{poly}(\lambda, \log k, \ell_{\mathcal{C}})$. We actually prove below a stronger claim that these quantities are at most $\mathsf{poly}(\lambda, \log k, \log \ell_{\mathcal{C}}) \cdot \ell_{\mathcal{C}}$.

We first examine $|\mathsf{vk}|$. Recall that $\mathsf{vk} = ((\mathsf{com}_g)_{g\in\mathcal{G}_{\mathsf{stmt}}}, \mathsf{ck}_{[k]}, (\mathsf{ck}_g)_{g\in\mathcal{G}_{\mathsf{out}}})$. Note that by Definition 5 fresh commitments are of size at most $\mathsf{poly}(\lambda, \log k)$ and by Definition 6 aggregated commitments are identical to fresh ones. Therefore, $|\mathsf{com}_g| \leq \mathsf{poly}(\lambda, \log k)$. Next, by Definition 9 it holds that $|\mathsf{ck}_g| \leq \mathsf{poly}(\lambda, \log k, \log m, \log |\chi_g|)$ where $m = |T_g| \leq \ell_{\mathcal{C}}$ in the current context and $|\chi_g| \leq \ell_{\mathcal{C}}$. Thus, $|\mathsf{ck}_g| \leq \mathsf{poly}(\lambda, \log k, \log \ell_{\mathcal{C}})$. Finally, by Definition 13 it holds that $|\mathsf{ck}_{[k]}| \leq \mathsf{poly}(\lambda, \log k)$. Putting everything together, we conclude that $|\mathsf{vk}| \leq \mathsf{poly}(\lambda, \log k, \log \ell_{\mathcal{C}})$.

For the runtime of $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$, by Definition 6, each of the $\ell_{\mathtt{w}}$ calls to $\mathsf{PCFC.EffProjVer}$ runs in time at most $\mathsf{poly}(\lambda, \log k)$ according to Definition 13. Also, each of the $|\mathcal{G}_{\mathsf{out}}| \leq \ell_{\mathcal{C}}$ calls to $\mathsf{PCFC.EffFuncVer}$ runs in time at most $\mathsf{poly}(\lambda, \log k, \log m, \log |\chi_g|)$ which as analysed above is at most $\mathsf{poly}(\lambda, \log k, \log \ell_{\mathcal{C}})$. We thus conclude that $\mathsf{BARG.EffVer}(\lambda, \mathsf{vk})$ runs in time at most $\mathsf{poly}(\lambda, \log k, \log \ell_{\mathcal{C}}) \cdot \ell_{\mathcal{C}}$. □

Note: in precomputation we could fix all wires of the statements corresponding to gates (when precomputing the function) by modifying the gates directly. The circuit would simply become a single-input $\mathcal{C}(\mathtt{w})$.

B Algebraic Interpretation of the Circuit-Succinct Compiler

Let $\mathcal{C}$ be the boolean circuit of the NP relation proven in a BARG, let $\ell_{\mathcal{C}}$ be the circuit size, and let k be the number of instances. The algebraic BARG underlying Corollary 1 and the proof of security can be intuitively seen as follows.

1. At setup, generate a PC commitment key of size $\ell = \ell_{\mathcal{C}} \cdot k$. In trapdoor mode for instance i^*, set the extraction set $I = \{(i^* - 1) \cdot \ell_{\mathcal{C}} + 1, \ldots, i^* \cdot \ell_{\mathcal{C}}\}$.
2. Arrange the multiple BARG computation traces $\boldsymbol{x}_i \in \{0,1\}^{\ell_{\mathcal{C}}}$ in rows in a matrix $\mathbf{X} \in \{0,1\}^{k \times \ell_{\mathcal{C}}}$. Recall that the first $\ell_{\mathtt{x},\mathtt{w}}$ coordinates of the $\boldsymbol{x}_i$ correspond to $\mathtt{x}_i|\mathtt{w}_i$ and that the last coordinate corresponds to y_i (see Table 1 for a visualization of $\mathbf{X}$). Note that I corresponds exactly to the i^*-th row of $\mathbf{X}$.
3. Commit to each position of the inputs $\mathtt{x}_i|\mathtt{w}_i$ across all instances of the batch. Graphically, this is equivalent to committing to the columns $\boldsymbol{x}_j$ of $\mathbf{X}$ corresponding to witnesses (i.e., $j \in [\ell_{\mathtt{x},\mathtt{w}}]$). We obtain $\ell_{\mathtt{x},\mathtt{w}}$ commitments $[\hat{\boldsymbol{c}}_j]_2$. Note that the intersection between I and the j-th column of $\mathbf{X}$ corresponds exactly to the $x_{i^*,j}$ element.
4. Prove the binariness of the values committed in $[\hat{\boldsymbol{c}}_j]_2$ corresponding to witness wires, i.e., for $j \in \{\ell_{\mathtt{x}} + 1, \ldots, \ell_{\mathtt{x},\mathtt{w}}\}$. Due to the somewhere extractability of the projection proofs, we can extract $\mathtt{w}_{i^*,j}$ from each $[\hat{\boldsymbol{c}}_j]_2$. Moreover, we know that their projective subspace commitment $\left[\hat{\boldsymbol{d}}_j\right]_2$ commits exactly to $\mathtt{w}_{i^*,j}$. Proving binariness for statement wires is not needed as these commitments are recomputed by the verifier.
5. Aggregate every input commitment $[\hat{\boldsymbol{c}}_j]_2$ into a commitment $[\hat{\boldsymbol{c}}]_2 = \sum_{j \in [\ell_{\mathtt{x},\mathtt{w}}]} [\hat{\boldsymbol{c}}_j]_2$. The resulting commitment corresponds to the first $\ell_{\mathtt{x},\mathtt{w}}$ rows of $\mathbf{X}$. Moreover, due to the algebraic nature of the aggregation, its projective commitment $\left[\hat{\boldsymbol{d}}\right]_2$ commits exactly to $\mathtt{x}_{i^*}|\mathtt{w}_{i^*}$.
   Due to the structure of the commitment keys, $[\hat{\boldsymbol{c}}]_2$ can be seen as a commitment to the following vector of length ℓ, if the whole commitment key $\left[\hat{\mathbf{T}}\right]_2$ is used:
   $$[\hat{\boldsymbol{c}}]_2 = \left[\hat{\mathbf{T}}\right]_2 \cdot \left[\mathtt{x}_1|\mathtt{w}_1|\mathbf{0}_{\ell_{\mathcal{C}} - \ell_{\mathtt{x},\mathtt{w}}}| \cdots |\mathtt{x}_k|\mathtt{w}_k|\mathbf{0}_{\ell_{\mathcal{C}} - \ell_{\mathtt{x},\mathtt{w}}}\right]^T,$$
   where the i^*-th block corresponds to the extraction set I.
6. Prove the trace of the computation of $\mathcal{C}$ using the CFC proof, where the CFC function is defined as k parallel copies of $\mathcal{C}$. The input commitment is $[\hat{\boldsymbol{c}}]_2$, and the output commitment $[\hat{\boldsymbol{c}}_y]_2$ commits to the vector of ones $\mathbf{1}_k$. Internally, the CFC proof ensures that after evaluating each of the copies of $\mathcal{C}$, we obtain a commitment to
   $$\left[\mathtt{x}_1|\mathtt{w}_1|\boldsymbol{x}_1|1| \cdots |\mathtt{x}_k|\mathtt{w}_k|\boldsymbol{x}_k|1\right]^T.$$

Then, we project the last coordinate of each block to obtain the commitment $[\hat{\boldsymbol{c}}_y]_2$ to $\mathbf{1}_k$. Security follows from the projective chain binding property of the PCFC as detailed in Lemma 4.

C Additional descriptions of the WW24 CFC

WW24 functional openings. Given an arithmetic circuit $\mathcal{C} : \mathbb{F}^\ell \to \mathbb{F}^{\ell_y}$, the CFC algorithms $\mathsf{FuncProve}(\mathsf{ck}, \boldsymbol{x} \in \mathbb{F}^\ell, \mathcal{C})$ and $\mathsf{FuncVer}(\mathsf{ck}, \mathsf{com}, \mathsf{com}_y, \mathcal{C}, \pi)$ work as follows (note that $\mathsf{com}, \mathsf{com}_y$ are standard type-I commitments):

- Evaluate $\mathcal{C}(\boldsymbol{x})$ and commit to the entire computation trace $\boldsymbol{z} \in \mathbb{F}^\ell_{\mathcal{C}}$ on both keys, obtaining σ_1, σ_2. Note that $\boldsymbol{z} = \boldsymbol{x}|\boldsymbol{z}_{\mathcal{C}}|\boldsymbol{y}$ where $\boldsymbol{z}_{\mathcal{C}} \in \mathbb{F}^{\ell_{\mathcal{C}}-\ell-\ell_y}$.
- Compute a subvector proof for $\boldsymbol{x}$ on $\boldsymbol{z}$ as $\pi_{\mathsf{Pre}} \leftarrow \mathsf{OpenPre}(\mathsf{ck}, \boldsymbol{x}, \boldsymbol{z})$, where ck has been extended via $\mathsf{ck} \leftarrow \mathsf{SetupPre}(\mathsf{ck}_{\mathsf{base}}, [\ell])$. This proof is verified as $\mathsf{VerPre}(\mathsf{ck}, \mathsf{com}, \sigma_1, \pi_{\mathsf{Pre}})$.
- Compute a linear proof for consistency between σ_1, σ_2 as $\pi_{\mathsf{Lin}} \leftarrow \mathsf{OpenLin}(\mathsf{ck}, \boldsymbol{z}, id)$ where $id : \mathbb{F}^\ell_{\mathcal{C}} \to \mathbb{F}^\ell_{\mathcal{C}}$ is the identity function. This proof is verified as $\mathsf{VerLin}(\mathsf{ck}, \sigma_1, \sigma_2, id, \pi_{\mathsf{Lin}})$.
- Compute a quadratic gate proof as $\pi_{\mathsf{Quad}} \leftarrow \mathsf{OpenQuad}(\mathsf{ck}, \boldsymbol{z}, \chi_{\mathcal{C}})$ where $\chi_{\mathcal{C}} : \mathbb{F}^\ell_{\mathcal{C}} \to \mathbb{F}^\ell_{\mathcal{C}}$ is the so-called "next-wire" function. Following the notation of Remark 8, $\chi_{\mathcal{C}}$ is a $([j-1], [j])$-separable function for every $j \in [\ell_{\mathcal{C}}]$ where the j-th output is the result of evaluating the j-th gate function of $\mathcal{C}$, i.e., $y_j \leftarrow \chi_j(x_1, \ldots, x_{j-1})$. This proof is verified as $\mathsf{VerQuad}(\mathsf{ck}, \sigma_2, \sigma_1, \chi_{\mathcal{C}}, \pi_{\mathsf{Quad}})$.
- Compute an output consistency proof, given by $\pi_{\mathsf{out}} \leftarrow \mathsf{OpenQuad}(\mathsf{ck}, \boldsymbol{z}, p)$ where $p : \mathbb{F}^\ell_{\mathcal{C}} \to \mathbb{F}^\ell_{\mathcal{C}}$ is given by $p(\boldsymbol{z}) = (z_{\ell_y+1}, \ldots, z_{\ell_{\mathcal{C}}}, \mathbf{0}_{\ell_{\mathcal{C}}-\ell_y})$. This function projects the output of the circuit $\boldsymbol{y} = \mathcal{C}(\boldsymbol{x})$ into the first ℓ coordinates, and is a $([j + \ell_{\mathcal{C}} - \ell_y], [j])$-separable function for every $j \in [\ell_y]$.[20] This proof is verified as $\mathsf{VerQuad}(\mathsf{ck}, \sigma_2, \mathsf{com}_y, p, \pi_{\mathsf{out}})$.

CFC evaluation binding (Definition 8) follows by a complex sliding-window hybrid argument, which presents similarities with the proof of Theorem 3.

[20] In the original scheme, this consistency proof is simply a linear projection from σ_1 to a type-II commitment to $\boldsymbol{y}$ computed by the verifier. Here, we introduce the quadratic consistency proof for chainability, namely to ensure that the output commitment is a well-formed type-I commitment.