# SNARKs from LWE via Non-black-box Reductions

*(Or: How to rewind non-interactively)*

Zhengzhong Jin
Northeastern University

Mingqi Lu
Northeastern University

Bo Peng
Peking University

**Abstract**

We construct the first succinct non-interactive arguments of knowledge (SNARKs) from the polynomial-hardness of Learning with Errors (LWE) for a subclass of $\mathsf{UP}$ languages whose witness unambiguity has a polynomial-size Extended Frege ($\mathcal{EF}$) proof. Our construction achieves the following soundness guarantee:

For any fixed sequence of false instances $\{x_\lambda\}_{\lambda\in\mathbb{N}}$, there exists a (non-constructive) constant $c > 0$ such that, whenever the uniform random CRS length exceeds $\lambda^c$, the construction achieves infinitely-often soundness for this sequence $\{x_\lambda\}_{\lambda\in\mathbb{N}}$: for any polynomial-time cheating prover $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, the probability that $\mathcal{A}_\lambda$ outputs an accepting proof for $x_\lambda$ is negligible for infinitely many $\lambda$.

As intermediate results, we also obtain:

- SNARGs for any $\mathsf{NP}$ language that has polynomial-size $\mathcal{EF}$ proofs of witness unambiguity for all instances outside of the language, based on polynomial-hard LWE, achieving the same style of soundness guarantee.
- SNARKs for all true instances in any language $\mathcal{L} \in \mathsf{UP}$ where every instance has a polynomial-size $\mathcal{EF}$ proof of witness unambiguity, under polynomial hardness of LWE, without soundness guarantees for false instances.

To achieve our main result, we employ a non-black-box soundness reduction. Along the way, we introduce a new logical proof system, the *Cryptographic Extended Frege* ($\mathcal{CEF}$) system, which extends $\mathcal{EF}$ with rules for formalizing the indistinguishability in cryptographic security proofs. Building on the Encrypt-hash-and-BARG framework of [Jin–Kalai–Lombardi–Vaikuntanathan, STOC'24], we further obtain SNARGs for $\mathsf{NP}$ languages that have $\mathcal{CEF}$ proofs of non-membership, which may be of independent interest.

# Contents


**1 Introduction** **3**

**2 Our Results** **4**
- 2.1 Discussions on Non-black-box Techniques . . . 7
- 2.2 Example: An $\mathcal{EF}$-UP Language Outside of coNP . . . 8
- 2.3 Key Idea: Soundness via Non-Black-Box Reduction . . . 9

**3 Technical Overview** **11**
- 3.1 Background on Extended Frege Logic . . . 11
- 3.2 Recap of Encrypt-Hash-and-BARG Template . . . 12
- 3.3 Warm-Up: SNARKs for True $\mathcal{EF}$-UP Statements . . . 14
- 3.4 Cryptographic Extended Frege Logic $\mathcal{CEF}$ . . . 15
- 3.5 Handling CRS Size Blow-Up in Non-Black-Box Soundness Reduction . . . 17
- 3.6 SNARGs via $\mathcal{CEF}$ Proof of Non-membership . . . 18
- 3.7 Formalizing the Correctness of Knowledge Extractor in $\mathcal{CEF}$ . . . 21

**4 Preliminary** **23**
- 4.1 Boolean Circuits . . . 23
  - 4.1.1 Universal Circuits . . . 23
- 4.2 Probabilistically Checkable Proofs . . . 24
- 4.3 Correlation Intractable Hash Functions . . . 25
- 4.4 Fully Homomorphic Encryption . . . 26
- 4.5 Somewhere Statistically Binding Hash Functions(SSB Hash) . . . 28
- 4.6 Somewhere Extractable Batch Arguments (BARGs) . . . 29

**5 Cryptographic Extended Frege Proofs** **30**

**6 SNARGs via $\mathcal{CEF}$-Proof of Non-membership** **33**
- 6.1 SNARG Construction . . . 34
- 6.2 Main Theorems . . . 36

**7 Proof of Lemma 6.3** **41**
- 7.1 Formalizing Existing Proofs in $\mathcal{CEF}$ . . . 41
- 7.2 Proof of Core Lemma . . . 52

**A $\mathcal{EF}$ Proof of Extraction Correctness for SSB Hash** **66**

**B $\mathcal{CEF}$ Proof of Correlation Intractability** **68**

**C $\mathcal{CEF}$ Proof of Extraction Correctness for BARG** **72**

# 1 Introduction

Succinct non-interactive arguments (SNARGs) allow a prover to convince a verifier of the truth of a statement via a short, single-round message. The standard setting for SNARGs is the common reference string (CRS) model, where a trusted setup algorithm generates a CRS that is shared by the prover and the verifier. SNARGs have broad applications, including delegation of computation [GKR15], verifying distributed computation [CT10], zero-knowledge protocols [Kiy22], verifiable private information retrieval [BDKP22], and even applications beyond cryptography, such as showing the cryptographic hardness of PPAD [BPR15].

Since Micali's seminal work [Mic00], which constructed SNARGs for NP in the random oracle model, a long line of work has centered on the following question:

*Can we construct SNARGs for* NP *in the standard model? Under what assumptions?*

Among them, one line of work targets supporting the full class of NP in the standard model. However, prior constructions in this direction either rely on knowledge-type assumptions (e.g., [CL08, Mie08, Gro10, GGPR13]) or on standard assumptions coupled with heavy cryptographic machinery—most notably indistinguishability obfuscation (iO) [SW14, WW24, WZ24, WW25], and recently, witness encryption or laconic 2-round arguments with efficient propositional proof of correctness [JKLM25].

**SNARGs from Simple Standard Assumptions.** Another line of work relies only on simple standard assumptions such as learning with errors (LWE) and decisional Diffie–Hellman (DDH). This line currently supports restricted subclasses of NP, with the long-term goal of broadening coverage and ultimately obtaining SNARGs for all of NP. To date, this line has yielded SNARGs for P from various standard assumptions [CJJ22, KVZ21, HJKS22, WW22, KLVW23, CGJ+23]; SNARGs for batched verification of NP languages [CJJ21, WW22, CGJ+23]; and SNARGs for monotone-policy batched verification of NP languages [BBK+23, NWW24, JJ25, NWW25]. Recent work [JKLV24] constructed SNARGs for NP languages whose non-membership has efficient propositional proofs, which essentially subsumes all prior SNARG constructions for subclasses of NP from LWE and sub-exponential DDH. However, under the polynomial-hardness of LWE, [JKLV24] only yields SNARGs for a subclass of languages in NP∩coNP. In this work, we ask the following question:

**Question 1:** *Can we construct SNARGs from LWE for a class of languages broader than* NP ∩ coNP*?*

**Argument of Knowledge.** Proof of knowledge [GMR89, BG93] provide a natural strengthening of soundness for proof systems. They require that from any prover that convinces the verifier of the truth of a statement, one can efficiently extract a witness. In the setting of SNARGs, this strengthening is captured by the argument-of-knowledge property, and SNARGs satisfying this property are commonly referred to as SNARKs. This property arises naturally when SNARGs are integrated with other cryptographic building blocks, and enables applications such as recursive composition of SNARGs [Val08, BCCT13], complexity-preserving SNARGs [BCCT13], and proof-carrying data [CT10].

However, despite being fundamental in both theory and practice, our understanding of SNARKs in the standard model is still limited. Current construction of SNARKs for NP either requires

knowledge-type assumptions or the random oracle model. Indeed, there are strong barriers to achieving SNARKs for NP in the standard model [BCPR14, BP15].

Within the standard-assumption regime, known positive results for SNARKs are limited to restricted subclasses of NP. For UP, where each statement has a unique witness, non-adaptive SNARKs are known from iO [CGKS23], but existing iO constructions [JLS21, JLS22, RVV24] rely on multiple standard assumptions and heavy machinery. The recent work [MPV24] achieves adaptive soundness based on evasive LWE, a relatively less well-studied assumption in lattices. From simpler standard assumptions such as LWE, previously only SNARKs for batch NP [CJJ21] and monotone-policy batch-NP [BBK+23] are known. This motivates the following question:

**Question 2:** *Can we construct SNARKs from simple standard assumptions for a broader class of languages?*

**Main Challenge: How to rewind "non-interactively"?** Indeed, under the random oracle model, SNARKs for NP have been known since Micali's initial work [Mic00]. In that setting, the reduction is very intuitive: it repeatedly reprograms the random oracle, gradually extracting small pieces of the NP witness from short SNARG proofs and stitching them together into a consistent witness. In the standard model and based only on standard assumptions, however, it is unclear how to achieve a similar rewinding-based reduction strategy.

The key difficulty arises from the non-interactive nature of SNARGs. At first glance, a reduction algorithm of SNARGs can still generate CRS with special structure, run the cheating prover to obtain the corresponding cheating proofs, and then attempt to "stitch" these proofs together into a consistent NP witness. Indeed, this idea underlies knowledge extraction in batch arguments for NP and in monotone-policy batch arguments for NP. However, when considering general languages beyond batch-NP style languages, it remains unclear how to argue that the cheating proofs under different CRS are *consistent*[1]. Indeed, this difficulty can be formalized as black-box separations of SNARKs from falsifiable assumptions in the setting of black-box extraction [CGKS23, ADVW13].

## 2 Our Results

Our result leverages *logical proofs* of witness uniqueness for UP languages to ensure consistency in the rewinding-style reduction. We begin by recalling the Extended Frege ($\mathcal{EF}$) proof system.

**Background: Extended Frege Logic.** A logical proof in $\mathcal{EF}$ for a propositional formula $\phi$ is a sequence of *lines*, where each line is a propositional formula built from logical connections "$\wedge$, $\vee$, and $\rightarrow$" representing "AND", "OR", "imply", respectively. Each variable represents either "T (True)" or "F (False)", and every line is derived from a constant number of previous lines via certain deduction rules, such as *modus ponens*[2]. A more detailed overview of the $\mathcal{EF}$ system is deferred to Section 3.1.

Our result assumes the existence of an $\mathcal{EF}$ proof of *unambiguity*, which formalizes—within the $\mathcal{EF}$ system—the statement that the NP witness for a given instance is unique.

[1] Batch-NP style languages do not have such a consistency concern because the witnesses for different NP instances in a batch are essentially independent.

[2] Modus ponens states that if a formula $p$ and the implication $p \rightarrow q$ have both been derived, then $q$ may be derived in the next step.

**$\mathcal{EF}$-Proof of Unambiguity.** Let $\mathcal{L}$ be an $\mathsf{NP}$ language with relation $C$. For any instance $x \in \{0,1\}^*$, an $\mathcal{EF}$ proof of unambiguity for $x$ is an $\mathcal{EF}$ proof for the following statement:

"the $\mathsf{NP}$ witness of $x$ is unique".

Formally, this corresponds to the propositional formula: "$C(x, w_1) \wedge C(x, w_2) \to (w_1 = w_2)$", where $x$ is *hardcoded* as constants, and $w_1, w_2$ are sequences of propositional variables representing the bits of the two $\mathsf{NP}$ witnesses. The formula is open in $w_1, w_2$, thereby expressing the universal quantification "for all $w_1, w_2$" implicitly.

Now we can define a subclass of $\mathsf{UP}$ languages, where the uniqueness property of the witness can be proved by polynomial-size $\mathcal{EF}$ proofs.

**Complexity Class $\mathcal{EF}$-$\mathsf{UP}$.** We say an $\mathsf{NP}$ language $\mathcal{L}$ belongs to $\mathcal{EF}$-$\mathsf{UP}$ if every instance—whether or not it lies in $\mathcal{L}$—has a polynomial-size $\mathcal{EF}$ proof of unambiguity. Note that we do not require the $\mathcal{EF}$ proof of unambiguity to be uniform in $x$: for each instance $x$, there may exist an $\mathcal{EF}$ proof of the uniqueness of the witness that non-uniformly depends on $x$.

Intuitively, this class includes any $\mathsf{UP}$ languages for which we can explicitly explain *why they belong to* $\mathsf{UP}$, i.e., we can *feasibly prove* [Coo75] that the language has at most one valid witness per instance. We further present in Section 2.2 an example of a language in $\mathcal{EF}$-$\mathsf{UP}$ that cannot be captured by previous SNARG constructions from LWE.

**Theorem 2.1** (Main Theorem, Informal)**.** *Assuming polynomial-hardness of learning with errors (LWE), there exists a SNARK with uniformly random CRS for every language $\mathcal{L} \in \mathcal{EF}$-$\mathsf{UP}$, satisfying the following soundness guarantee:*

*For any sequence of "false" statements $\{x_\lambda\}_{\lambda\in\mathbb{N}}$ where $x_\lambda \notin \mathcal{L}$, there exists a constant $c > 0$ such that, for any instantiation of our construction with CRS length exceeding $\lambda^c$, the scheme achieves infinitely-often soundness for this sequence of statements.*

*That is, for any polynomial-size cheating prover $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, the probability that $\mathcal{A}_\lambda$ convinces the verifier to accept $x_\lambda$ is negligible on infinitely many $\lambda$.*

**On the Soundness Guarantee.** Our soundness notion in Theorem 2.1 stems from a non-black-box soundness reduction based on a *disjunction argument*, which informally states:

- either the SNARK construction with a CRS length $\lambda$ is already sound for $\{x_\lambda\}_{\lambda\in\mathbb{N}}$; or
- there exists an adversary $\mathcal{A}$ that breaks the soundness for CRS length $\lambda$, in which case we prove that the SNARK becomes sound for $\{x_\lambda\}_{\lambda\in\mathbb{N}}$, as long as the CRS length exceeds a fixed polynomial in $\mathcal{A}$'s circuit size.

This reasoning leads naturally to the *infinitely-often* security guarantee in Theorem 2.1, since the adversary $\mathcal{A}$ may succeed only for infinitely many security parameters. Similar infinity-often security definitions or constructions are very common in non-black-box techniques based on disjunction arguments for various problems in cryptography [RTV04, MPS10, Den17, HNO+18, KY18, PV20, CKU20, Zha21, RV24, CJJQ23]. In some of these works, the infinitely-often aspect is explicitly discussed, while in others it is treated as a minor technical subtlety. We defer a more comprehensive discussion about non-black-box reductions in cryptography to Section 2.1.

This work uses the disjunction argument to obtain a non-black-box reduction for SNARGs. We note, however, that our soundness guarantee is weaker than a standard infinitely-often soundness: the constant $c$ in Theorem 2.1 that lower bounds the CRS length is a non-constructive integer that may depend on the instance family $\{x_\lambda\}_{\lambda\in\mathbb{N}}$. Nevertheless, this soundness notion still captures the intuition of soundness: for any fixed sequence of instances $\{x_\lambda\}_{\lambda\in\mathbb{N}}$, once the CRS length is long enough, the construction is secure. [3]

**Towards SNARKs for NP.** Valiant–Vazirani [VV86] showed a randomized reduction from NP to UP, suggesting that UP may capture the computational hardness of NP. However, this reduction cannot be directly combined with our result to obtain SNARKs for NP, since it is unclear how to prove, in polynomial-size Extended Frege, that the resulting UP language is indeed unambiguous. Closing this gap remains an intriguing direction for future work toward obtaining the first SNARKs for NP from polynomial-hard standard assumptions.

Our main result Theorem 2.1 can be extended to obtain SNARGs for languages beyond UP, as long as every "false" instance has a polynomial-size $\mathcal{EF}$-proof of unambiguity.

**Theorem 2.2** (SNARGs via $\mathcal{EF}$-Proofs of Unambiguity, Informal Version of Theorem 6.8 )**.** *Assuming polynomial-hardness of learning with errors (LWE), for any language $\mathcal{L} \in \mathsf{NP}$, if there exists a polynomial $\mathsf{poly}(\cdot)$ such that every false instance $x \notin \mathcal{L}$ has a $\mathsf{poly}(|x|)$ $\mathcal{EF}$-proof of unambiguity, then there exists a SNARG for $\mathcal{L}$ with the same soundness guarantee as in Theorem 2.1.*

**What Languages Have $\mathcal{EF}$-Proofs of Unambiguity?** We first observe that any language $\mathcal{L}$ admitting polynomial-size $\mathcal{EF}$ proofs of non-membership—as introduced in the recent work [JKLV24]—also admits polynomial-size $\mathcal{EF}$ proofs of unambiguity for all false instances. Specifically, [JKLV24] constructed SNARGs for NP languages $\mathcal{L}$ in which every false instance $x \notin \mathcal{L}$ has a polynomial-size $\mathcal{EF}$ proof of the statement "for all $w$, $C(x, w) = 0$", where $C$ is the NP relation circuit of $\mathcal{L}$.

To see why $\mathcal{EF}$-proofs of unambiguity generalize this earlier notion, note that an $\mathcal{EF}$-proof of non-membership informally asserts:

"If $C(x, \cdot) = 1$ is satisfiable, then we can derive a contradiction",

while an $\mathcal{EF}$-proof of unambiguity asserts:

"If $C(x, \cdot) = 1$ has two distinct satisfying assignments, then we can derive a contradiction."

Clearly, the former implies the latter. However, the converse may not hold: a language may have a polynomial-size $\mathcal{EF}$ proof of unambiguity for all false instances even when no polynomial-size $\mathcal{EF}$ proof of non-membership is known. In Section 2.2, we present such a language in $\mathcal{EF}$-UP.

As an intermediate result that may be of independent interest, we also obtain SNARKs (with the argument of knowledge property) for $\mathcal{EF}$-UP *via black-box extractions*. However, the knowledge extraction is guaranteed only for instances *within* the language, and our construction does not provide soundness guarantees for instances outside the language.

[3] The reader may wonder what exactly qualifies as a "long enough CRS". We note that a similar uncertainty already exists in the standard definition of security: for every constant $a > 0$, there exists a threshold $\lambda_a$ such that for all $\lambda > \lambda_a$, the adversary's advantage is at most $1/\lambda^a$. However, the bound $\lambda_a$ is never explicitly constructed.

**Theorem 2.3** (SNARKs for True $\mathcal{EF}$-UP Statements, Informal Version of Theorem 6.5 )**.** *Assuming polynomial-hardness of LWE, for any language $\mathcal{L} \in \mathcal{EF}$-UP, there exists a SNARK that satisfies argument of knowledge property for instances within $\mathcal{L}$, but it does not provide any soundness guarantee for instances outside $\mathcal{L}$.*

*Formally, there exists a uniform extractor $E$ and a polynomial $\mathsf{poly}(\cdot)$ such that, for any sequence of instances $\{x_\lambda\}_{\lambda\in\mathbb{N}}$ with $x_\lambda \in \mathcal{L}$ and any prover $\{\mathcal{P}^*_\lambda\}_{\lambda\in\mathbb{N}}$ that convinces the verifier to accept $\{x_\lambda\}_{\lambda\in\mathbb{N}}$ with non-negligible probability $\epsilon = \epsilon(\lambda)$, there exists a negligible function $\mathsf{negl} = \mathsf{negl}(\lambda)$ such that the black-box extractor $E^{\mathcal{P}^*_\lambda}(x_\lambda)$ outputs a valid witness for $x_\lambda$ in expected time $\mathsf{poly}(\lambda, 1/(\epsilon(\lambda) - \mathsf{negl}(\lambda)))$, for any sufficiently large $\lambda$.*

In contrast to the non-black-box reductions in Theorem 2.1 and Theorem 2.2, the extractor $E$ in the above theorem only invokes the prover $\{\mathcal{P}^*_\lambda\}_{\lambda\in\mathbb{N}}$ in a black-box way. Its correctness relies on LWE, so we allow a negligible loss in the prover's success probability. The reduction proving the correctness of $E$ remains black-box with respect to the prover $\{\mathcal{P}^*_\lambda\}_{\lambda\in\mathbb{N}}$.

Indeed, our main result (SNARK for $\mathcal{EF}$-UP, Theorem 2.1) can be obtained immediately by combining the argument of knowledge for true $\mathcal{EF}$-UP instances in Theorem 2.3 and the soundness for false $\mathcal{EF}$-UP instances in Theorem 2.2.

**On SNARKs for True Statements.** We note that SNARKs restricted to true statements — without any soundness guarantee for false ones — remain a meaningful and nontrivial notion. For instance, TFNP consists of NP languages that are always true, making traditional soundness vacuous since no false instances exist. In such settings, the argument of knowledge property captures the right intuition of soundness: it ensures that any prover convincing the verifier must know why the statement holds. Our Theorem 2.3 trivially implies a SNARK for TFNP $\cap$ $\mathcal{EF}$-UP, which may be of independent interest.

## 2.1 Discussions on Non-black-box Techniques

**Non-black-box Reductions in Cryptography.** Most cryptographic security proofs are black-box: the reduction only queries the adversary as an oracle. A large body of work shows that this methodology has inherent limitations, including black-box separations between cryptographic primitives [IR89, RTV04], barriers to black-box simulation in zero knowledge [GK96, CKPR01], and, in the setting of SNARGs, impossibility results of SNARGs from falsifiable assumptions [GW11].

Non-black-box reductions, which are allowed to use the code of the adversary, provide a way to bypass some of these barriers. Previously, not many non-black-box reduction techniques were known. Perhaps one of the most influential examples is Barak's non-black-box simulation technique [Bar01], which has leads to many applications [Bar02, BL02, BP12, CPS13, BKP19](and references therein). Other works use "win-win" or disjunction arguments: either a given cryptographic construction is already secure, or any adversary that breaks it can be used to construct another cryptographic scheme [DNRS99, Den17] (and subsequent works).

To the best of our knowledge, our work is the first to apply this disjunctive-argument paradigm to obtain non-black-box reductions in the setting of SNARGs. Our techniques for implementing this disjunctive argument differ from prior non-black-box techniques in the literature.

**Concurrent Work [DHK+26].** Independently and concurrently, [DHK+26] constructed SNARGs for NP assuming LWE and the existence of a (super-polynomial-length) non-signaling PCP for NP.

Their reduction also makes non-black-box use of the adversary, but their techniques are very different from ours.

**Are Non-black-box Reductions Necessary?** Both Theorem 2.1 and Theorem 2.2 rely on non-black-box reductions. This raises a natural question: are non-black-box techniques *necessary* for our results, or could one obtain essentially the same results as Theorem 2.1 and Theorem 2.2 via a black-box reduction? To investigate this question, we examine known black-box barriers in SNARGs, such as the Gentry–Wichs impossibility [GW11, CJ25] and the black-box separation of SNARKs from falsifiable assumptions [CGKS23, ADVW13].

Regarding the Gentry–Wichs barrier, which assumes adaptively sound SNARGs, our result achieves a soundness notion weaker than non-adaptive soundness, so it remains unclear whether the non-black-box techniques are necessary. In light of the revisited Gentry–Wichs impossibility [CJ25], which shows that a non-rewinding (i.e., one-query), black-box reduction with a short CRS is impossible for languages beyond $\mathsf{NP} \cap \mathsf{coNP}$, it only suggests that either a long CRS, or the rewinding reduction, or the non-black-box aspect of the reduction is essential. For the black-box separation of SNARKs from falsifiable assumptions [CGKS23, ADVW13], those results rely crucially on $\mathsf{NP}$ languages that have multiple witnesses for the same instance. In contrast, our work focuses on subclasses of $\mathsf{UP}$ where each instance has a unique witness, and hence the impossibility no longer applies.

In summary, whether there exists a black-box reduction for SNARKs for $\mathcal{EF}$-$\mathsf{UP}$ from polynomial-time assumptions, or whether non-black-box techniques are indeed essential for our results, remains an intriguing open question.

## 2.2 Example: An $\mathcal{EF}$-UP Language Outside of coNP

We present an example of a language $\mathcal{L} \in \mathcal{EF}$-$\mathsf{UP}$ that is unlikely to have a polynomial-size $\mathcal{EF}$ proof of non-membership. Consequently, the existence of a SNARG for $\mathcal{L}$ is not implied by prior work [JKLV24].

Our example is based on any "natural" perfectly correct [4] public-key encryption scheme whose security does not imply a separation between $\mathsf{AM} \cap \mathsf{co\text{-}AM}$ and $\mathsf{BPP}$, together with an arbitrary hash function. For concreteness, we instantiate the example using McEliece's cryptosystem [McE78].

Let $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote the McEliece PKE scheme, and let $\mathsf{Hash}$ be a hash function. We define:

$$\mathcal{L} = \Big\{(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m; r)) \mid \text{There exists } m, r, r_{\mathsf{Gen}} \text{ such that } \mathsf{pk} = \mathsf{Gen}(1^{\lambda=|m|}; r_{\mathsf{Gen}}) \text{ and } (r, r_{\mathsf{Gen}}) = \mathsf{Hash}(m)\Big\}.$$

Namely, $\mathcal{L}$ consists of all ciphertexts $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, m; r)$ such that both the encryption randomness $r$ and key-generation randomness $r_{\mathsf{Gen}}$ for $\mathsf{pk}$ are deterministically derived from the message $m$ via $\mathsf{Hash}$. If $\mathsf{Hash}$ were modeled as a random oracle, the ciphertexts generated this way should still hide $m$ for a randomly sampled $m \leftarrow \{0,1\}^\lambda$. Hence, we can expect $\mathcal{L}$ to be a hard language.

**$\mathcal{EF}$ Proof of Unambiguity.** Before formalizing the $\mathcal{EF}$ proof, we first explain informally why $\mathcal{L} \in \mathsf{UP}$. By the perfect correctness of McEliece's scheme, each well-formed ciphertext $\mathsf{ct}$ decrypts

[4] By "natural", we require the correctness of the PKE to be provable in polynomial-size $\mathcal{EF}$ proofs.

to a unique message $m$. Moreover, since $(r, r_{\mathsf{Gen}})$ is deterministically computed as $\mathsf{Hash}(m)$, the rest of the witness $(r, r_{\mathsf{Gen}})$ is uniquely determined once $m$ is unique. Hence, the witness for any $\mathsf{ct}$ is unique.

To formalize this proof of uniqueness, we first formalize the correctness of $\mathsf{PKE}$, which is stated as:

"If $\mathsf{pk}, \mathsf{sk}$ are correctly generated by $\mathsf{Gen}(1^\lambda; r_{\mathsf{Gen}})$ and $\mathsf{ct}$ is correctly generated by $\mathsf{Enc}(\mathsf{pk}, m; r)$, then $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ recovers the original message $m$."

This can be formalized in $\mathcal{EF}$ as the following propositional formula:

$$\Big[(\mathsf{pk}, \mathsf{sk}) = \mathsf{Gen}(1^\lambda; r_{\mathsf{Gen}}) \wedge \mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, m; r)\Big] \to \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = m.$$

This formula has a polynomial-size $\mathcal{EF}$ proof because McEliece's decryption primarily relies on Patterson's decoding algorithm for Goppa codes. Other operations such as permutation and multiplication by a non-singular matrix are purely arithmetic and thus formalizable in polynomial-size $\mathcal{EF}$ proofs. The Patterson's decoding algorithm and its correctness are fully arithmetic operations, so they can also be formalized in Cook's theory PV. By Cook's translation from PV to polynomial-size $\mathcal{EF}$ [Coo75], we obtain a polynomial-size $\mathcal{EF}$ proof for the correctness of McEliece's PKE.

Next, we show that $\mathcal{L}$ has polynomial-size $\mathcal{EF}$ proofs of unambiguity, and hence $\mathcal{L} \in \mathcal{EF}\text{-}\mathsf{UP}$. For any ciphertext $\mathsf{ct}$—whether or not $\mathsf{ct} \in \mathcal{L}$—let $(m, r, r_{\mathsf{Gen}})$ and $(m', r', r'_{\mathsf{Gen}})$ be two potential witnesses. We aim to derive $(m, r, r_{\mathsf{Gen}}) = (m', r', r'_{\mathsf{Gen}})$ within a polynomial-size $\mathcal{EF}$ proof.

From the first witness, we have $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, m; r)$, and from the second witness, $(\mathsf{pk}, \mathsf{sk}') = \mathsf{Gen}(1^\lambda; r'_{\mathsf{Gen}})$. Applying the correctness of $\mathsf{PKE}$, we derive $\mathsf{Dec}(\mathsf{sk}', \mathsf{ct}) = m$. Then we can also derive $\mathsf{Dec}(\mathsf{sk}', \mathsf{ct}) = m'$ by using $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, m'; r')$ from the second witness. Combining the two, we obtain $m = m'$. Finally, by applying the deterministic hash, we can derive

$$(r, r_{\mathsf{Gen}}) = \mathsf{Hash}(m) = \mathsf{Hash}(m') = (r', r'_{\mathsf{Gen}}),$$

yielding a polynomial-size $\mathcal{EF}$ proof of unambiguity.

**Non-Existence of $\mathcal{EF}$-Proofs of Non-membership.** We are not aware of any method to give a polynomial-size $\mathcal{EF}$ proof of non-membership for $\mathcal{L}$. Indeed, since McEliece cryptosystem is not known to imply a separation between $\mathsf{AM} \cap \mathsf{co\text{-}AM}$ and $\mathsf{BPP}$, the language $\mathcal{L}$ is unlikely to be in $\mathsf{coNP}$. Hence, it is unlikely that $\mathcal{L}$ has a polynomial-size $\mathcal{EF}$ proof of non-membership, since such a proof corresponds to a $\mathsf{coNP}$ witness. Intuitively, to show $\mathcal{L} \in \mathsf{coNP}$, one would need an $\mathsf{NP}$ witness to certify that a random string is far from any codeword of a "randomly permuted Goppa code", which remains an open problem.

### 2.3 Key Idea: Soundness via Non-Black-Box Reduction

Our starting point is Theorem 2.3. Our SNARK for true $\mathcal{EF}\text{-}\mathsf{UP}$ statements builds on the same construction as the SNARG for $\mathcal{EF}$-proofs of non-membership from [JKLV24], but we give a new proof to show the argument of knowledge property. Our proof crucially relies on the instance being true and on the existence of an $\mathcal{EF}$ proof of unambiguity to guarantee consistency of the extractor across different indices of the witness. We defer a detailed overview of our proof strategy to Section 3.3.

**Observation: Adversary as a Proof of Non-membership.** To prove the soundness for false instances and obtain our main result (Theorem 2.1), our key observation is that if an adversary $\mathcal{A}$ breaks the soundness of this SNARK on a "false" instance $x$ of a language $\mathcal{L}$, then $\mathcal{A}$ itself can be viewed as an "NP-type proof" certifying $x \notin \mathcal{L}$. The reason is as follows. If we ignore the subscripts such as $\lambda$ and success probability, then our Theorem 2.3 can be simplified as the following formula:

$$x \in \mathcal{L} \implies \text{extractor } E^{\mathcal{A}}(x) \text{ outputs a witness for } x, \tag{1}$$

or formally, let $C$ be the NP relation of $\mathcal{L}$,

$$\exists w \text{ s.t. } C(x, w) = 1 \implies C(x, E^{\mathcal{A}}(x)) = 1.$$

Its contrapositive states that if $E^{\mathcal{A}}$ fails to output an NP witness of $x$, then $x \notin \mathcal{L}$. Hence, for any false instance $x \notin \mathcal{L}$, any adversary $\mathcal{A}$ that produces an accepting proof for the SNARK in Theorem 2.3 can be viewed as a "NP-type proof" for $x \notin \mathcal{L}$: on the input instance $x$, the NP verifier takes the circuit $\mathcal{A}$ as a non-deterministic witness, and runs $E^{\mathcal{A}}(x)$. If the output is *not* an NP witness of $x$, then we decide that $x \notin \mathcal{L}$.

**Attempt: Soundness via Proof of Non-membership.** The above is a key observation, since the recent work [JKLV24] already constructed SNARGs for NP languages that have polynomial-size Extended Frege ($\mathcal{EF}$) proofs of non-membership. Specifically, they require that every false instance $x \notin \mathcal{L}$ has a polynomial-size $\mathcal{EF}$ proof of the statement "the relation circuit $C(x, \cdot)$ always outputs 0".

If we can further show that the correctness of the extractor $E$ — i.e. Equation (1) — has a polynomial-size $\mathcal{EF}$ proof, then soundness follows immediately. This is because we can combine the $\mathcal{EF}$ proof of Equation (1), which also proves Equation (1)'s contrapositive:

$$E^{\mathcal{A}}(x) \text{ fails to output a witness of } x \implies C(x, \cdot) \equiv 0,$$

with a straightforward $\mathcal{EF}$ proof that the output of $E^{\mathcal{A}}(x)$ is not an NP witness of $x$, to derive an $\mathcal{EF}$ proof of $C(x, \cdot) \equiv 0$. The latter is a direct, step-by-step computation of $C(x, E^{\mathcal{A}}(x)) = 0$ since $E, \mathcal{A}, x$ are all fixed circuits and constants in $\mathcal{EF}$. Once we have a polynomial-size $\mathcal{EF}$ proof of non-membership, soundness follows from [JKLV24]. Note that this is a *non-black-box reduction*, since the reduction in [JKLV24] uses FHE to encrypt the $\mathcal{EF}$ proof in the CRS, and in our case, the $\mathcal{EF}$ proof contains the adversary $\mathcal{A}$. Hence, the above reduction strategy makes non-black-box use of $\mathcal{A}$.

However, Equation (1) likely cannot be proven in $\mathcal{EF}$ (let alone in polynomial size), since the correctness of the extractor $E$ relies on cryptographic assumptions, while all $\mathcal{EF}$-provable statements hold unconditionally.

**Our Proof Strategy.** To resolve the above issue, we proceed in three steps:

1. Introduce a new logic system, the *Cryptographic Extended Frege* system $\mathcal{CEF}$, which extends $\mathcal{EF}$ with *random variables* and a new rule — the *cryptographic gadget substitution* rule — to formalize the indistinguishability-based reasoning in cryptography. This extension equips $\mathcal{CEF}$ to reason not only about the *correctness* of cryptographic constructions (as $\mathcal{EF}$ already

does in recent works [JJ22, JKLV24, MDS25, JKLM25, JJMP25]) but also about their *security* properties, such as the semantic security of public-key encryption and the soundness of SNARGs. This added expressive power is both essential to our results and of independent interest. Indeed, similar ideas are implicit in the recent work [JKLM25] to build SNARGs and succinct obfuscation [JJMP25]. We defer a detailed overview of $\mathcal{CEF}$ to Section 3.4.

2. Formalize the correctness of the extractor (Equation (1)) in $\mathcal{CEF}$. Indeed, the correctness proof in Theorem 2.3 relies solely on indistinguishability-based hybrid arguments and simple logical deductions—the former can be expressed using the cryptographic gadget substitution rule (the new rule in $\mathcal{CEF}$), and the latter have polynomial-size $\mathcal{EF}$ proofs. We defer a detailed overview of this formalization to Section 3.7.

3. Extending existing SNARGs [JKLV24] to remain sound for languages that have poly-size $\mathcal{CEF}$ proof of non-membership. This step encountered a significant technical challenge, which we discuss later in the technical overview Section 3.6.

**Summary of our Proof of Soundness.** Putting the above pieces together, we now summarize our strategy for proving the soundness of the SNARK in Theorem 2.3 on false instances. Suppose there exists an adversary $\mathcal{A}$ that breaks the soundness of this SNARK. Then the cheating prover $\mathcal{A}$, together with the correctness of the extractor $E$ (Item 2) and the fact that $E^{\mathcal{A}}$ fails to output a valid witness, collectively form a $\mathcal{CEF}$ proof of non-membership. Since our SNARG remains sound for all languages that have $\mathcal{CEF}$ proofs of non-membership (Item 3), soundness of the SNARK follows.

The above plan again is a *non-black-box reduction*: similar to [JKLV24], the $\mathcal{CEF}$ proof is encrypted in CRS using FHE during our soundness reduction (Item 3), and in our case, the $\mathcal{CEF}$ itself crucially depends on the circuit of the adversary $\mathcal{A}$. This creates further technical challenges—for example, the CRS now grows with the adversary $\mathcal{A}$'s size, leading to a circular dependence between the two. We defer these details to the technical overview Section 3.5.

# 3 Technical Overview

**Organization.** We give a detailed background in Section 3.1 about Extended Frege logic; In Section 3.2, we give a self-contained recap of the previous work [JKLV24], which our techniques built on. In Section 3.3, we overview our techniques to achieve Theorem 2.3, a SNARK for true $\mathcal{EF}$-UP statements, but does not provide any soundness for false statements. Then we follow the blueprint outlined in Section 2.3: first introduce the *Cryptographic Extended Frege* logic system $\mathcal{CEF}$ in Section 3.4; then we extend SNARGs for $\mathcal{EF}$-proofs of non-membership to handle $\mathcal{CEF}$-proofs of non-membership in Section 3.6; and finally we formalize the correctness of the extractor in Theorem 2.3 in polynomial-size $\mathcal{CEF}$-proofs in Section 3.7.

## 3.1 Background on Extended Frege Logic

Extended Frege ($\mathcal{EF}$) formalizes reasoning about Boolean formulas as a sequence of derivation steps. Our presentation differs slightly from the standard textbook definition, since formalizing cryptographic reasoning requires tracking details such as whether a trapdoor or secret key is still present in the current set of derived formulas. To capture this, we follow the bounded-space formulation

of $\mathcal{EF}$ [JKLV24, ET01, ABSRW02], which explicitly maintains the evolving set of formulas at each deduction step.

An $\mathcal{EF}$ proof of a formula $\theta$ is a sequence of sets of formulas

$$\Phi_1, \Phi_2, \ldots, \Phi_u,$$

starting from the empty set $\Phi_0 = \emptyset$ and ending with a configuration $\Phi_u$ that contains $\theta$, which is the statement to be proven. Each step derives $\Phi_{i+1}$ from $\Phi_i$ by applying exactly one of the following operations:

- **Inference:** There exist formulas $\varphi_1, \ldots, \varphi_c \in \Phi_i$ and a formula $\varphi$ such that $(\varphi_1 \wedge \cdots \wedge \varphi_c) \to \varphi$ is a tautology. In this case, we derive $\Phi_{i+1}$ from $\Phi_i$ by adding $\varphi$ to the configuration $\Phi_i$.
- **Extension:** We may introduce a fresh variable $e$ and define it to stand for a Boolean formula $\psi$ built from variables already in $\Phi_i$. This lets us "name" complex subformulas and reason about them more easily later in the proof. The new configuration $\Phi_{i+1}$ is obtained by adding this definitional formula $e \leftrightarrow \psi$ to the set $\Phi_i$.
- **Erasure:** We may remove a formula $\varphi$ from the current configuration if we know we will not need it anymore. This keeps the proof space (configuration) small. The next configuration $\Phi_{i+1}$ is derived by dropping $\varphi$ in $\Phi_i$. In particular, we require that if $\varphi$ is a definitional formula of a extension variable, then $\Phi_{i+1}$ no longer uses the extension variable.

### 3.2 Recap of Encrypt-Hash-and-BARG Template

Our starting point is the *Encrypt-Hash-and-BARG* framework from recent work [JKLV24], which unifies several recent SNARG constructions from simple standard assumptions such as LWE [JKLV24, JKLM25, BBK+23]. We first recall the notion of batch arguments (BARGs), on which the framework is built.

**Batch Arguments (BARGs).** A batch argument (BARG) is a SNARG for verifying a batch of $\mathsf{NP}$ statements simultaneously. Given statements $x_1, \ldots, x_k$ and witnesses $w_1, \ldots, w_k$, the prover sends a short proof $\pi$ whose size is much smaller than the total witness length $k \cdot |w_i|$. BARGs satisfy an *semi-adaptive argument-of-knowledge* property: given a cheating proof under a CRS encoding an index $i^*$, an efficient extractor with a trapdoor can recover the witness $w_{i^*}$ for $x_{i^*}$.

*Index BARG.* An *index BARG* is a special case where instances are generated in a "uniform" way from their index, i.e., the $i$-th instance $x_i$ is efficiently computable from its index $i$. In this setting, we can view the instances as indices $1, \ldots, k$, and the SNARG proof size can be reduced to $\mathsf{poly}(\lambda, |w_i|, \log k)$ [CJJ21], which essentially only grows with one of the $\mathsf{NP}$ witnesses.

**Sketch of Encrypt-Hash-and-BARG Construction.** The construction proceeds as follows:

- **CRS Generation:** Create two FHE ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ encrypting 0's and a somewhere statistically binding (SSB) hash key with extractability. The CRS includes $\mathsf{ct}_1, \mathsf{ct}_2$ and the hash key.
- **Prover:** Given a statement $x$ and $\mathsf{NP}$ witness $w$, the prover algorithm proceeds as the following:

– *FHE Evaluation:* Compute the relation circuit $C(x, w)$ and all of its intermediate wire values $\tau_C$. Then treat the ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ as encrypted circuits under FHE and evaluate them gate-by-gate on input $\tau_C$, recording all intermediate FHE ciphertexts during the evaluation as $\tau_1$ and $\tau_2$, respectively.

– *Hash:* Apply SSB hash to all intermediate wire values and FHE ciphertexts $\tau_C, \tau_1, \tau_2$ to obtain hash commitments $v_C, v_1, v_2$ for them, respectively.

– *BARG:* Generate index BARGs proofs certifying the correctness of $\tau_C, \tau_1, \tau_2$. Specifically, for $\tau_C$, generate a proof $\pi_C$ showing that for every gate in $R(x, \cdot)$, there exists a local opening of that gate with respect to $v_C$, and that the gate computation is correct. Moreover, for the output wire $\mathsf{out}_C$ of $R(x, \cdot)$, the BARG also proves that $\mathsf{out}_C = 1$.

For $\tau_i (i = 1, 2)$, generate a proof $\pi_i$ showing that for every gate encrypted in $\mathsf{ct}_i$, there exists a local opening of the gate w.r.t. $v_i$, and that the gate-by-gate FHE evaluation is correct on that encrypted gate.

- **Verifier:** Verify the index BARG proof $\pi_C, \pi_1, \pi_2$ w.r.t. the SSB hash roots $v_C, v_1, v_2$, respectively.

[JKLV24] proves soundness of the above SNARG construction for any language $\mathcal{L}$ that has polynomial-size $\mathcal{EF}$ proof of non-membership. This requires that, for any $x \notin \mathcal{L}$, there exists a polynomial-size $\mathcal{EF}$ proof for the statement "$C(x, \cdot)$ always outputs 0". The original proof of [JKLV24] is slightly different from our exposition.

**Proof of Soundness.** The idea of using $\mathcal{EF}$ proof to aid security reduction originated from [JJ22]. Given a poly-size $\mathcal{EF}$ proof of non-membership $\Phi_1, \ldots, \Phi_u$ (See Definition in Section 3.1), we can construct a sequence of corresponding circuits $E_1, \ldots, E_u$, where the $i$-th circuit $E_i$ computes "the AND of all formulas in $\Phi_i$", i.e.,

$$E_i = \wedge_{\varphi \in \Phi_i} \varphi.$$

Then the first circuit $E_0$ computes a single "1" since $\Phi_1$ is empty, and the last circuit $E_u$ computes a sequence of AND that contains a formula "$\mathsf{out}_C = 0$", where $\mathsf{out}_C$ denotes the output wire of $C(x, \cdot)$. This follows the requirement that the last configuration $\Phi_u$ must contain the "statement to be proven" itself (See Section 3.1). In [JKLV24], the statement to be proven is the non-membership of $x$: "$C(x, \cdot) \equiv 0$", which is expressed as "$\mathsf{out}_C = 0$" in $\mathcal{EF}$.

JKLV then applied a hybrid argument over this sequence of circuits to prove soundness. Suppose there exists a cheating prover $\mathcal{A}$ that breaks the soundness of the above SNARG construction. We construct $u$ hybrids, where the $i$-th hybrid uses $E_i$ as follows:

1. During CRS generation, encrypt $E_i$ into the first FHE ciphertext $\mathsf{ct}_1$.
2. Run $\mathcal{A}$ to obtain the cheating proof.
3. Extract the output wire of $E_i$ by first extracting the FHE ciphertext of $E_i$ from SSB hash $v_1$, and then decrypting the FHE ciphertext to obtain the output wire value of $E_i$.

JKLV then proves the following claim about this sequence of hybrids. Let us denote the output wire of $E_i$ by $\mathsf{out}_i$.

**Claim 3.1.** *If* $\mathsf{out}_i = 1$ *in the* $i$*-th hybrid, then* $\mathsf{out}_{i+1} = 1$ *in the* $(i+1)$*-th hybrid, except with negligible probability.*

To see why this claim implies soundness, consider the final hybrid (i.e., the $u$-th hybrid), which encrypts $E_u$. Recall that $E_u$ computes the AND of a sequence of formulas that contain the formula $\mathsf{out}_C = 0$. We can use the SSB to extract the entire computation of this formula and use BARG soundness to ensure this formula is computed correctly. However, this contradicts $\mathsf{out}_u = 1$ because the BARG soundness of $\pi_C$ also guarantees that $\mathsf{out}_C = 1$, and hence truth value of the formula $\mathsf{out}_C = 0$ is evaluated to 0. Since $E_u$ computes a bunch of AND that contains a 0, $\mathsf{out}_u$ should also be 0. This completes the soundness proof.

There are more details on how to prove the above Claim 3.1. However, knowing the above claim holds is sufficient to introduce the necessary techniques to understand our SNARK for true $\mathcal{EF}$-$\mathsf{UP}$ statements (Theorem 2.3). We will provide a more detailed proof of a generalized version of the claim (and why it is formalizable in $\mathcal{CEF}$ in the later section of this overview Section 3.7).

### 3.3 Warm-Up: SNARKs for True $\mathcal{EF}$-$\mathsf{UP}$ Statements

In this section, we present our construction of a SNARK for any true $\mathcal{EF}$-$\mathsf{UP}$ statement (Theorem 2.3). To build intuition, let us temporarily set aside the formal statement of Theorem 2.3 and attempt to construct a SNARK for a general $\mathsf{NP}$ language $\mathcal{L}$ from first principles.

**Attempt: Knowledge Extraction via "Rewinding".** Our starting point is the Encrypt-Hash-and-BARG template for SNARG in [JKLV24], summarized in Section 3.2. To show the argument of knowledge property for this construction, our initial attempt is to consider the following natural extractor $E^{\mathcal{A}}(x)$.

Given any adversary $\mathcal{A}$, the extractor first sets the SSB hash key to be extractable for the $i$-th bit of the witness $w$ and runs $\mathcal{A}$ to obtain a cheating proof under this CRS. Using the SSB extraction, it then extracts the $i$-th bit $w_i$ from the cheating proof. Repeating this process for each bit $i$ allows $E^{\mathcal{A}}$ to extract all bits of the witness $w$, one at a time. This process resembles the "rewinding" reduction strategy used in interactive proofs, since here the extractor rewinds the adversary $\mathcal{A}$ by rerunning it under different CRSs for the same instance $x$.

However, this extractor does not suffice: we cannot argue that bits extracted under different CRS come from the same underlying witness. If the language $\mathcal{L}$ can have multiple witnesses for the same instance, even when $x \in \mathcal{L}$, a cheating prover $\mathcal{A}$ may choose different witnesses for different CRS, preventing the above extractor from finding a consistent witness.

**Consistency via $\mathcal{EF}$-Proof of Unambiguity.** The above observation leads us to consider $\mathsf{UP}$ languages, which is the complexity class that consists of all $\mathsf{NP}$ languages that have at most one witness. However, it remains unclear how to leverage the uniqueness of the witness for a $\mathsf{UP}$ language in general, since intuitively, the reduction should only work when the language is in $\mathsf{UP}$, and does not work when the language is not in $\mathsf{UP}$. The same intuition has led to the use of $\mathcal{EF}$ proofs in the previous work [JJ22] and [JKLV24] where polynomial-size $\mathcal{EF}$ proofs are used to overcome the non-falsifiability of the obfuscation and SNARG definition.

In this work, we also assume a polynomial-size $\mathcal{EF}$ proof, but for the uniqueness of the $\mathsf{NP}$ witness. And we formalize this as the $\mathcal{EF}$ proof of unambiguity, which requires that, for any instance $x$ (could

be in $\mathcal{L}$ or not in $\mathcal{L}$), there exists a poly-size $\mathcal{EF}$ proof for the uniqueness of the NP witness of $x$. Namely, the following formula needs to have a polynomial-size proof:

$$C(x, w_1) \wedge C(x, w_2) \to w_1 = w_2.$$

With this assumption, we can prove the consistency of the aforementioned extractor $E$ when extracting different bits, as follows. For any true instance $x \in \mathcal{L}$ that has a polynomial $u = u(|x|)$ length $\mathcal{EF}$ proof of unambiguity, since $x \in \mathcal{L}$, there always exists a $w^*$ such that $C(x, w^*) = 1$, and then the $\mathcal{EF}$ proof of unambiguity implies an $\mathcal{EF}$ proof for

$$C(x, w) \to w = w^*,$$

since now $C(x, w^*) = 1$ holds automatically. Now we can use the same proof strategy as described in Section 3.2 to show the following claim:

**Claim 3.2.** *For any index $i$, if $E^{\mathcal{A}}(x)$ extracts the $i$-bit of the witness $w_i$ from the adversary $\mathcal{A}$, it must hold that $w_i = w_i^*$, except with negligible probability.*

This claim gives us the consistency we need in the extractor $E$ since it ensures that the extracted bits between different indices is consistent with the unique witness $w^*$.

To see why Claim 3.2 holds, everything in the soundness proof sketch in Section 3.2 remains the same, with the only difference that the statement that the $\mathcal{EF}$ proof proves now becomes $w_i = w_i^*$. Hence, the final configuration $\Phi_u$ contains such a formula $w_i = w_i^*$ and the last circuit $E_u$ also computes an AND of many formulas, with $w_i = w_i^*$ being one of them. The Claim 3.1 still applies, and so we can still argue that the output wire of $E_u$ in the last hybrid (i.e. the $u$-th hybrid) is still 1 when it gets extracted. Then we can switch the extraction position of SSB hash to the entire computation of $w_i = w_i^*$, the BARG soundness can ensure that $w_i = w_i^*$ must hold. This finish the sketch of the proof for Claim 3.2.

However, the above proof strategy only works when extracting a witness for "true statements" — instances that is in the language. It remains unclear how to achieve soundness for "false" instances. The reader may wonder whether this is a trivial notion, since for every true instance $x$, its NP witness $w^*$ always exists. A closer look reveals that this is already non-trivial: the efficient extractor $E^{\mathcal{A}}$ do not need to know the witness $w^*$, instead, only the reduction needs to know this $w^*$, and the extractor $E$ itself is an uniform extractor. This concludes our overview for Theorem 2.3.

### 3.4 Cryptographic Extended Frege Logic $\mathcal{CEF}$

As outlined in Section 2.3, to prove the soundness of our SNARK for false instances, we first extend the Extended Frege ($\mathcal{EF}$, see Section 3.1) to support *indistinguishability*-based reasoning in cryptography. In this section, we introduce this extended system as the *Cryptographic Extended Frege* ($\mathcal{CEF}$) system. In the subsequent sections, we will then formalize the correctness of the extractor $E$ from Section 3.3 in $\mathcal{CEF}$ and extend SNARG for $\mathcal{EF}$ proof of non-membership to handle $\mathcal{CEF}$ proofs as well.

**Augmenting $\mathcal{EF}$ with Random Variables.** Cryptography fundamentally relies on randomness, whereas $\mathcal{EF}$ is limited to static variables and constants ("true" and "false"). To capture cryptographic arguments, $\mathcal{CEF}$ extends $\mathcal{EF}$ with a new type of variable called *random variables*. In $\mathcal{CEF}$, we therefore distinguish two types of variables:

- **Proper Variables:** They are identical to variables in $\mathcal{EF}$, representing either "true" or "false".
- **Random Variables:** They are treated as formal symbols representing *independent, uniformly random bits*. Although random variables behave syntactically like variables in $\mathcal{EF}$, semantically they are closer to constants ($\mathsf{T}$ and $\mathsf{F}$ in $\mathcal{EF}$) since their values cannot be assigned.

One might wonder how the *extension rule* of $\mathcal{EF}$ behaves in this enriched setting. Recall that the extension rule allows the introduction of new variables defined as Boolean formulas over existing variables. In $\mathcal{CEF}$, a variable $e$ may be defined as a Boolean formula $\phi$ involving both proper and random variables. Such variables are neither proper nor random, and we refer to them simply as *extension variables*.

**Towards a Rule for Cryptographic Indistinguishability.** With random variables in place, we can introduce a new deduction rule in $\mathcal{CEF}$ that captures the essence of hybrid arguments in cryptography. The motivation is as follows: suppose we are proving a statement in $\mathcal{EF}$. At each deduction step, we may derive new formulas, introduce new variables via the extension rule, or discard formulas no longer needed. At some point, our set of derived formulas $\Phi$ (called *configuration*) may include a term such as $\mathsf{Enc}(\mathsf{pk}, m; \mathbf{r})$, where $\mathsf{Enc}$ is a public-key encryption algorithm, $m$ is a variable representing a bit, and $\mathbf{r}$ is a vector of random variables. If neither $\mathbf{r}$ nor the secret key associated with $\mathsf{pk}$ appears elsewhere in $\Phi$, then, by semantic security, we should be able to replace $\mathsf{Enc}(\mathsf{pk}, m; \mathbf{r})$ with $\mathsf{Enc}(\mathsf{pk}, 0; \mathbf{r})$.

**Cryptographic Gadget Substitution Rule.** We formalize this reasoning principle as the *cryptographic gadget substitution rule*. Suppose the current set of derived propositional formulas is $\Phi$, and $\Phi$ contains a formula of the form

$$\mathbf{y} = f(\mathbf{z}, \mathbf{c}),$$

where $\mathbf{z}$ is a vector of *constant number* of variables, $\mathbf{c}$ is a vector of random variables, and $f$ is a "black-box" circuit that takes $\mathbf{z}$ as input and uses $\mathbf{c}$ as internal randomness and $\mathbf{c}$ does not appear elsewhere (This is why we view $f$ as a "black-box"). Then we are allowed to replace $f$ with another circuit $g$ provided that the distribution families

$$\{f(z, \mathbf{c})\} \text{ and } \{g(z, \mathbf{c}')\}$$

are *computationally indistinguishable* for every fixed $z \in \{0,1\}^{|\mathbf{z}|}$, where the randomness is over uniformly sampled $\mathbf{c}$ and $\mathbf{c}'$. We illustrate this rule in Figure 1.

This rule captures the earlier intuition about ciphertext indistinguishability. In the example with $\mathsf{Enc}(\mathsf{pk}, m; \mathbf{r})$, the encryption algorithm can be viewed as a gadget $f$ with input $m$ and internal randomness $\mathbf{r}$. Since $\mathbf{r}$ appears nowhere else, the substitution rule allows us to replace $f$ with a gadget $g$ that always encrypts 0, formalizing semantic security of public key encryption directly within the logic system.

We defer more details of $\mathcal{CEF}$ to Section 5. When defining the cryptographic gadget substitution rule formally, we explicitly allow a special case where the input $\mathbf{z}$ is "moderately small" and the output $\mathbf{y}$ is "very small", purely for technical convenience in our proof, as explained later in Section 3.6.

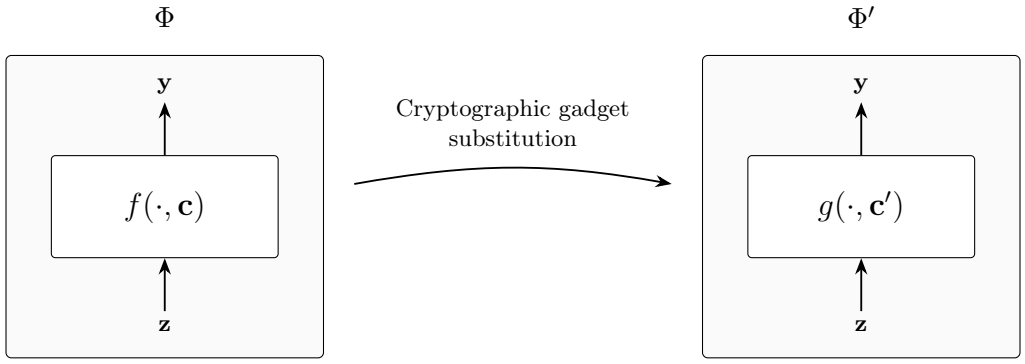


Figure 1: Illustration of the cryptographic gadget substitution rule. The original configuration (the set of derived formulas) $\Phi$ is shown on the left. After applying the rule, the "black-box" cryptographic gadget $f$ is replaced by $g$, resulting in a new configuration $\Phi'$ on the right.

**Relation to [JKLM25].** The recent work [JKLM25] also augments $\mathcal{EF}$ with cryptographic indistinguishability reasoning in its soundness proof. However, its techniques can not be extended directly to obtain SNARGs for $\mathcal{CEF}$ proofs of non-membership. The soundness proof of [JKLM25] can be viewed as using only cryptographic gadgets that take no proper variables or extension variables as input: these gadgets are "fixed" in the sense that they are efficiently samplable "constants", such as witness encryption ciphertexts of 0 or 1. In contrast, our soundness proof crucially needs gadgets whose input depends on the prover's witness $w$. Handling such gadgets is another key technical challenge of this work, which we explain in Section 3.6.

### 3.5 Handling CRS Size Blow-Up in Non-Black-Box Soundness Reduction

A subtle issue arises in our plan in Section 2.3 to prove soundness of the SNARK (presented in Theorem 2.3) for false instances. Since the CRS now FHE encrypts the adversary $\mathcal{A}$ during the reduction, the honest prover must read the entire CRS and hence runs longer than $\mathcal{A}$' size, while the adversary must run longer than the honest prover. This creates a circular dependence on the size of the CRS and the size of the adversary. At first glance, this seems to preclude any meaningful notion of soundness.

However, a close inspection reveals that our proof strategy guarantees the existence of a sufficiently large CRS that is sound for any sequence of instances $\{x_\lambda\}_{\lambda\in\mathbb{N}}$ that is fixed before choosing the CRS length. Specifically, if the SNARG is already sound for a moderately long CRS—say of length $\lambda^c$ for some constant $c$—then the claim already holds. Otherwise, suppose there exists an adversary that breaks soundness for this $\lambda^c$-length CRS on $\{x_\lambda\}_{\lambda\in\mathbb{N}}$. If the adversary's circuit size is $\mathsf{poly}(\lambda)$, then by choosing the CRS length to be sufficiently large—polynomially larger than this $\mathsf{poly}(\lambda)$—we can encode the adversary into the CRS and use the proof strategy outlined in Section 2.3 to obtain soundness again. In either case, whether the SNARK is already sound for a $\lambda^c$-length CRS or not, it becomes sound for some sufficiently large CRS. This yields the weaker form of non-adaptive soundness formalized in Theorem 2.1.

### 3.6 SNARGs via $\mathcal{CEF}$ Proof of Non-membership

Following the refined proof strategy in Section 3.5, we now handle the second case: when an "inner" adversary $\mathcal{A}$ breaks soundness for a moderately large CRS, we will show that the SNARG remains sound against any "outer" adversary $\mathcal{A}'$ provided the CRS is large enough to encrypt $\mathcal{A}$.

Recall that our proof strategy is to first show the following $\mathcal{CEF}$-proof of non-membership:

- Formalize the extractor's correctness in $\mathcal{CEF}$ i.e., prove by a polynomial-size $\mathcal{CEF}$ proof that "if $C(x, w) = 1$, then $E^{\mathcal{A}}(x)$ outputs $w$".
- Use a step-by-step evaluation of the circuit $E^{\mathcal{A}}(x)$ and $C(x, \cdot)$ to prove in $\mathcal{CEF}$ that "$C(x, E^{\mathcal{A}}(x)) = 0$" using a polynomial-size proof.
- Combine the two steps above to yield a contradiction and thus a $\mathcal{CEF}$ proof of *non-membership*: "for all $w$, $C(x, w) = 0$."

In this section, we will show how to use this $\mathcal{CEF}$ non-membership proof to prove the soundness of our SNARG.

**Challenge: Handling the Cryptographic Gadget Substitution Rule.** Our starting point is the soundness analysis of [JKLV24] (See Section 3.2). But here we must additionally account for the *cryptographic gadget substitution rule*, which we added as a new rule to the logic system. Our plan is to reprove Claim 3.1 for *$\mathcal{CEF}$-proofs* (rather than the $\mathcal{EF}$-proofs considered in Section 3.2):

**Claim 3.3** (Modified Claim 3.1 for $\mathcal{CEF}$-Proofs)**.** *Given configurations $\Phi_1, \ldots, \Phi_u$ of a $\mathcal{CEF}$ proof of non-membership, define circuit $E_i$ ($i \in [u]$) that computes the* AND *of all formulas in $\Phi_i$. Recall the $i$-th hybrid is constructed as follows: we encrypt $E_i$ in one of the FHE ciphertexts in the CRS, run the outer adversary $\mathcal{A}'$, and extract the output wire value of $E_i$ from the cheating proof using SSB extraction and FHE decryption. We have the following claim similar to Claim 3.1:*

*If the extracted output wire of $E_i$ in the $i$-hybrid is 1, then the extracted output wire of $E_{i+1}$ in the $(i+1)$-th hybrid is also 1, except with negligible probability.*

To prove this new claim, we note that when the step from $\Phi_i$ to $\Phi_{i+1}$ uses an $\mathcal{EF}$ rule, the same proof strategy in [JKLV24] applies. The difficulty is when the step uses a gadget substitution rule.

To see the issue, let us inspect what Claim 3.3 states when combined with our aforementioned $\mathcal{CEF}$ proof of non-membership. Our $\mathcal{CEF}$ proof of non-membership first needs to prove the correctness of the extractor $E^{\mathcal{A}}(x)$. This is where we crucially rely on the cryptographic substitution rule — the proof of correctness of the extractor first needs to switch the CRS for the inner adversary $\mathcal{A}$ to an FHE encryption of $w$, where $w$ is only determined when the adversary $\mathcal{A}'$ (or the honest prover) homomorphically evaluates the FHE ciphertexts in the CRS.

The corresponding hybrids are as follows. In the initial hybrid, the circuit $E_1$ always outputs 1. After a few $\mathcal{EF}$ extension steps, the $i$-th hybrid uses a circuit $E_i$ computing $E^{\mathcal{A}}(x)$ but hardwires its output to 1 (See Figure 2). In the $(i+1)$-th hybrid, we apply the cryptographic gadget substitution rule: the CRS still encrypts $E^{\mathcal{A}}$, but when $E^{\mathcal{A}}(x)$ extracts a witness from $\mathcal{A}$, we generate the CRS for the inner adversary $\mathcal{A}$ by homomorphically encrypting the witness $w$ chosen by the honest prover (or $\mathcal{A}'$). It is unclear how to prove the indistinguishability of the $i$-th hybrid and $(i+1)$-th hybrid using the semantic security of FHE here, since the plaintext $w$ does not even "exist" — we can never extract the entire $w$ to learn its value in the reduction. Instead, the reduction can only extract a few bits of $w$ each time it runs $\mathcal{A}'$.

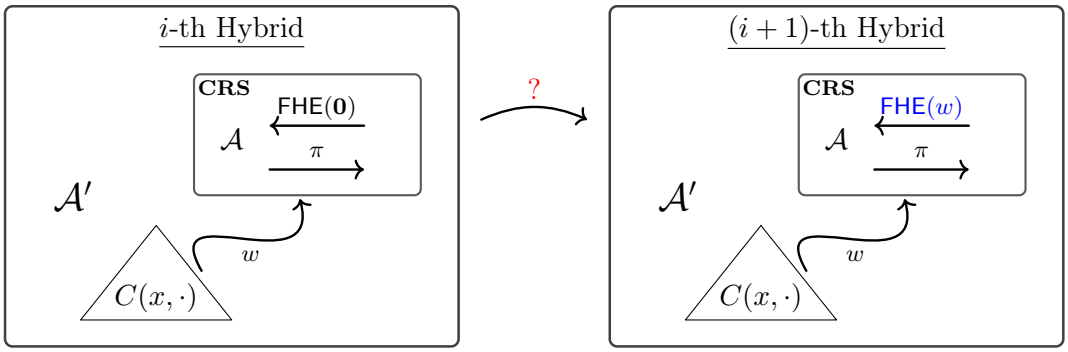


Figure 2: Illustration of the $i$-th hybrid and $(i+1)$-th hybrid. In both hybrids, the honest prover (or adversary $\mathcal{A}'$) chooses $w$ as input to $C(x,\cdot)$, and $w$ is treated as part of the intermediate wires of $C$. Then $w$ is fed as part of the input to the "extension circuit" $E_i$ encrypted in the CRS for $\mathcal{A}'$. $E_i$ ignores $w$ in the $i$-th hybrid, but in the next hybrid, $E_{i+1}$ FHE-encrypts $w$ using a fixed randomness and puts it in the CRS for the inner adversary $\mathcal{A}$ to extract a witness from $\mathcal{A}$ bit-by-bit. It is unclear how to argue the indistinguishability between the $i$-th hybrid and $(i+1)$-th hybrid using FHE security here since we can not extract the entire $w$ from $\mathcal{A}'$.

**Reducing to 1-bit $w$ via Bit-wise FHE Encryption.** A natural attempt to resolve this issue is to leverage the "bit-by-bit" nature of FHE encryption e.g., the GSW scheme [GSW13]. Namely, we first bit-decompose $w = (w_1, \ldots, w_m)$ with bits $w_j, j \in [m]$ and the FHE encrypts them bit-wise: $\mathsf{FHE}(w) = (\mathsf{FHE}(w_1), \ldots, \mathsf{FHE}(w_m))$. Then we build $m$ hybrids, switching from $\mathsf{FHE}(0^m)$ to $\mathsf{FHE}(w)$, one coordinate at a time. Now essentially the problem of proving indistinguishability reduces to the case of a single bit $w$, i.e., $|w| = 1$.

To prove the indistinguishability in the single-bit case, note that we only have two cases: $w = 0$ or $w = 1$. When $w = 1$, one might attempt to use the semantic security of FHE to replace $\mathsf{FHE}(1)$ with $\mathsf{FHE}(0)$. However, this does not suffice: after the replacement, the resulting hybrid generates one $\mathsf{FHE}(0; r)$ with randomness $r$ if $w = 0$ and generates another independent $\mathsf{FHE}(0; r')$ with randomness $r'$ if $w = 1$. We can argue the indistinguishability of this new hybrid and $(i+1)$-th hybrid using FHE semantic security, but it is unclear that this new hybrid is indistinguishable from the $i$-th hybrid. To finish the proof, we need to merge the two independent FHE ciphertexts of 0's into one.

**Reducing to Bit-to-Bit Gadget.** Our next observation is that we can further simplify the situation by bit-decomposing the random coins $r$ and $r'$, and therefore simplify the gadget to a "bit-to-bit gadget", which means it only has one variable in its input and only outputs one bit. Recall that our goal here is to show that if the output wire of $E_i$ is extracted as 1 in $i$-th hybrid, then the output of $E_{i+1}$ is also extracted as 1 except with negligible probability, where $E_i, E_{i+1}$ are the following circuits:

- $E_i$ contains a gadget which takes a single bit $w$ as input, but ignores it and always outputs $\mathsf{ct} = \mathsf{FHE}(0; r)$.
- $E_{i+1}$ is almost the same as $E_i$, except that we replace the gadget to a new gadget: it takes $w$ as input, but outputs $\mathsf{ct} = \mathsf{FHE}(0; r)$ if $w = 0$ and computes $\mathsf{ct} = \mathsf{FHE}(0; r')$ if $w = 1$. Both $r, r'$ here are the internal randomness of the gadget.

To show this, we first bit-decompose $r = (r_1, \ldots, r_s)$ and $r' = (r'_1, \ldots, r'_s)$. Then, merging the FHE ciphertexts $\mathsf{FHE}(0; r)$ and $\mathsf{FHE}(0; r')$ is the same as merging $r_j$ and $r'_j$'s. We can view this merging process as a sequence of cryptographic gadgets $f_j, g_j$:

$$f_j(w, r_j, r'_j) = r_j, \quad g_j(w, r_j, r'_j) = \begin{cases} r_j, \text{ if } w = 0 \\ r'_j, \text{ if } w = 1 \end{cases}.$$

$f_j, j \in [s]$ corresponds to the $i$-th hybrid, where we only use one FHE ciphertext $\mathsf{FHE}(0; r)$. $g_j, j \in [s]$ corresponds to the $(i+1)$-th hybrid, where we use $\mathsf{FHE}(0; r)$ if $w = 0$ and uses $\mathsf{FHE}(0; r')$ if $w = 1$. We use the two sequences of $s$ cryptographic gadgets to "select" the random coins $r$ or $r'$ bit-by-bit, depending on the value of $w$. Now, as long as we can prove that the switching from $f_j$ to $g_j$ is indistinguishable, we finish the proof. Note that this is a much simpler case than the general cryptographic gadget, since the output of $f_j, g_j$ is only one bit and they only use a one-bit variable $w$. Now we have reduced our problem to the bit-to-bit gadget case.

**Handling Bit-to-Bit Gadgets via a "Truth Table" Argument.** We now explain how we prove the indistinguishability for the bit-to-bit Gadgets. The claim we want to prove is essentially Claim 3.3 restricted to the bit-to-bit gadget:

**Claim 3.4.** *Suppose the circuit $E_{i+1}$ is obtained from $E_i$ by replacing a subcircuit $f$ with $g$. Here $f(z, \mathbf{c}), g(z, \mathbf{c}')$ are "bit-to-bit" gadgets, i.e., they each take a* one-bit *variable $z$ and arbitrary random variables $\mathbf{c}, \mathbf{c}'$ and output a single-bit variable.*

*Assume that for every fixed $z \in \{0, 1\}$, the distributions*

$$\left\{ f(z, c) : c \leftarrow \{0,1\}^{|\mathbf{c}|} \right\} \text{ and } \left\{ g(z, c') : c' \leftarrow \{0,1\}^{|\mathbf{c}'|} \right\}$$

*are computationally indistinguishable. Consider the $i$-th hybrid and $(i+1)$-th hybrid that respectively encrypt $E_i$ and $E_{i+1}$ in FHE ciphertext as part of the CRS for the outer adversary $\mathcal{A}'$, run $\mathcal{A}'$, and extract the output bit of $E_i, E_{i+1}$, respectively. Then the hybrids are indistinguishable. In particular, if the extraction in the $i$-th hybrid yields 1, then the extraction in the $(i+1)$-th hybrid also yields 1, except with negligible probability.*

The key idea in our proof is to tabulate the gadget's effect and defer its evaluation to the end. Using a polynomial-size $\mathcal{EF}$ proof, we show that the $i$-th hybrid is indistinguishable from one where the CRS encrypts a circuit $E'_i$ that outputs a two-bit truth table $(y_0, y_1) \in \{0,1\}^2$, where $y_b$ is the output of the circuit $E_i$ when the gadget $f$'s output is fixed to $b \in \{0, 1\}$. We then use SSB extraction and FHE decryption to extract the "selector bit" $z$ from the cheating proof, compute the gadget $b \coloneqq f(z, c), c \leftarrow \{0,1\}^{|\mathbf{c}|}$ *by ourselves (outside the game with the adversary)*, and finally read off $y_b$ from the truth table. In the next hybrid, we can replace $f$ by $g$ directly since the internal randomness $\mathbf{c}$ and $\mathbf{c}'$ are never encrypted in FHE ciphertexts during the hybrids. Then we can place the gadget $g$ back into the circuit and go through a polynomial-size $\mathcal{EF}$ proof again to reverse the hybrid back to the $(i+1)$-th hybrid.

**From Bit-to-Bit Gadgets to General Gadgets.** The argument above suffices for bit-to-bit gadgets, but we now generalize it to handle arbitrary gadgets. Consider a general substitution from $f(\mathbf{z}, \mathbf{c})$ to $g(\mathbf{z}, \mathbf{c})$, where $\mathbf{z}$ denotes a constant number of variables, and $\mathbf{c}, \mathbf{c}'$ are arbitrary *random*

*variables*, and the output length of $f, g$ are arbitrary polynomial length. To extend Claim 3.4 to this more general setting, we adopt an *input-by-input puncturing* proof strategy over the variables $\mathbf{z}$, similar to the puncturing technique used in the security proofs of indistinguishability obfuscation [SW14].

Namely, we enumerate all $z_0 \in \{0,1\}^{|\mathbf{z}|}$ and define a hybrid $\mathsf{Hybrid}(z_0)$: in this hybrid, we encrypt a circuit almost identical to $E_i$ except for the gadget, which behaves as follows. If $z < z_0$ it computes $g$; Otherwise, it computes $f$. Clearly, $\mathsf{Hybrid}(0)$ corresponds to the $i$-th hybrid, and the $\mathsf{Hybrid}(2^{|\mathbf{z}|})$ corresponds to the $(i+1)$-th hybrid. We now explain how to switch from $\mathsf{Hybrid}(i)$ to $\mathsf{Hybrid}(i+1)$ for any $i = 0, \ldots, 2^{|\mathbf{z}|} - 1$.

Our high-level idea is to first "puncture out" the point $\mathbf{z} = z_0$. Specifically, we define an intermediate hybrid $\mathsf{Hybrid}(i.1)$, which is almost identical to $\mathsf{Hybrid}(i)$ except that the gadget computes

$$\text{if } \mathbf{z} < z_0 : g(\mathbf{z}, \mathbf{c}'); \quad \text{if } \mathbf{z} = z_0 : f(z_0, \mathbf{c}); \quad \text{if } \mathbf{z} > z_0 : f(\mathbf{z}, \mathbf{c}).$$

This hybrid is indistinguishable from $\mathsf{Hybrid}(i)$ because the new gadget above is *functionality equivalent* to $f$, and this equivalence can be proven via a polynomial-size $\mathcal{EF}$ proof.

Our next observation is that, when $\mathbf{z} = z_0$, the output of $f(z_0, \mathbf{c})$ is fully determined — it follows the distribution $\{f(z_0, c) : c \leftarrow \{0,1\}^{|\mathbf{c}|}\}$, and hence is indistinguishable from $g(z_0, \mathbf{c}')$. We would like to rely on this to switch from $f$ to $g$ at $\mathbf{z} = z_0$. However, this does not work directly, since the same randomness $\mathbf{c}$ is reused across cases $z = z_0$ and $z > z_0$. Hence, we need first to introduce a fresh, independent random variable $\mathbf{c}''$ to isolate the punctured case $z = z_0$.

Thus, we move to another intermediate hybrid $\mathsf{Hybrid}(i.2)$, where for $z = z_0$, the gadget now computes $f(z_0, \mathbf{c}'')$ using an independent random vector $\mathbf{c}''$ of the same length as $\mathbf{c}$. The indistinguishability between $\mathsf{Hybrid}(i.1)$ and $\mathsf{Hybrid}(i.2)$ follows from a series of bit-to-bit gadgets substitutions, gradually replacing each bit of $\mathbf{c}$ with its fresh counterpart in $\mathbf{c}''$.

Then we replace $f$ by $g$ in the case $z = z_0$. Finally, we symmetrically switch the independent randomness $\mathbf{c}''$ back to $\mathbf{c}'$, yielding $\mathsf{Hybrid}(i+1)$. This completes the proof sketch of the general gadget substitution argument.

### 3.7 Formalizing the Correctness of Knowledge Extractor in $\mathcal{CEF}$

The final missing piece in our soundness proof (as outlined in Section 2.3) is to formalize the correctness of the extractor $E^{\mathcal{A}}$ in $\mathcal{CEF}$. Specifically, our goal is to show that, if there is a polynomial-size $\mathcal{EF}$ proof for the formula $C(x, w^*) \wedge C(x, w) \to w^* = w$, then there exists a polynomial-size $\mathcal{CEF}$ proof proving the correctness of $E^{\mathcal{A}}$, i.e., $C(x, E^{\mathcal{A}}(x)) = 1$.

**Generalized Statement.** Indeed, we prove a more general lemma, which shows that, not only does the above statement hold for proof of unambiguity, but it also holds for any property one can prove in $\mathcal{EF}$ given the premise $C(x, w) = 1$. Specifically, let $D$ be any polynomial-size predicate about $x$ and $w$, we will show that if there exists a polynomial-size $\mathcal{EF}$ (or even $\mathcal{CEF}$) proof for

$$C(x, w) \to D(x, w),$$

then if we extract the output bit of $D$ from running $\mathcal{A}$ with $D$ encrypted in the CRS, the output bit must also be 1. Moreover, the fact that the output bit is 1 itself has a polynomial-size $\mathcal{CEF}$ proof.

In our specific case, $D$ checks whether $w$ coincides with the "real" witness $w^*$. To incorporate the precondition $C(x, w^*) = 1$, we add an additional "global" predicate $D_{\mathsf{pre}}$ to appear both in the condition and the consequence in our lemma. An informal statement of our core lemma is as follows:

**Lemma 3.5** (Core Lemma, Informal Version of Lemma 6.3 )**.** *Assuming polynomial-hardness of learning with errors (LWE), for any language $\mathcal{L}$, any instance $x$, if there exists a polynomial-size $\mathcal{CEF}$ proof for*

$$\text{“Under a ‘global’ condition } D_{\mathsf{pre}}(x,y)=1,\ C(x,w)=1 \text{ implies } D(x,y,w)=1\text{”},$$

*where $y$ and $w$ are “proper variables” and $D_{\mathsf{pre}}, D$ are polynomial-size predicates, then there exists a polynomial-size $\mathcal{CEF}$ proof for the following statement:*

$$\text{“If } D_{\mathsf{pre}}(x,y)=1, \text{ then if we extract the output bit of } D(x,y,w) \text{ from } \mathcal{A} \text{ by encrypting } D \text{ in the CRS and running } \mathcal{A}, \text{ the output bit will be } 1\text{.”}$$

**Specialization to Extractor Correctness.** By instantiating $D_{\mathsf{pre}}(x,w^*)$ with $C(x,w^*)$ and $D(x,w^*,w)$ with the equality checking function $[w^*=w]$[5], we obtain a transformation from a $\mathcal{CEF}$ proof of

$$C(x,w^*)\wedge C(x,w)\to w^*=w$$

to a $\mathcal{CEF}$ proof of

$$\text{“If } C(x,w^*)=1\text{, then if we extract the output bit of } [w^*=w] \text{ from } \mathcal{A} \text{ by encrypting a circuit computing } [w^*=w] \text{ in the CRS and running } \mathcal{A}\text{, the output bit must be 1.”}$$

By invoking a “local consistency” of the SNARG extraction, we can move this equality check outside the extractor, recasting it as

$$\text{“If } C(x,w^*)=1\text{, then whenever we extract the } i\text{-bit } w_i \text{ of } w \text{ from } \mathcal{A}\text{, it must be that } w_i^*=w_i\text{.”}$$

Since we have $C(x,w^*)=1$ as a precondition, we can derive $C(x,E^{\mathcal{A}}(x))=C(x,w^*)=1$. This completes the proof of extractor correctness in $\mathcal{CEF}$.

**Proof of Core Lemma 3.5.** The proof of Lemma 3.5 proceeds similarly to that in Claim 3.3. Specifically, we encrypt the circuit $E_i$ corresponding to the $i$-th configuration in the $i$-th hybrid experiment. Then, in the final hybrid, the circuit $E_i$ should compute

$$D_{\mathsf{pre}}(x,y)\to(C(x,w)\to D(x,y,w)).$$

Since $D_{\mathsf{pre}}(x,y)=1$ holds by assumption and $C(x,w)=1$ is exactly what the adversary is trying to prove, we can perform one final simplification step to remove these conditions from the encrypted circuit, leaving only $D(x,y,w)$ whose output bit must still be 1. To show that the proof process itself can be formalized within $\mathcal{CEF}$, we observe that it relies only on simple logical deductions and on the semantic security of FHE—the former can be formalized in $\mathcal{EF}$ and the latter can be formalized using the cryptographic gadget substitution rule. We defer more details to Section 7.

[5]We use $[w^*=w]$ to denote a equality checking function which takes $w^*, w$ as input, and outputs 1 iff $w^*=w$.

# 4 Preliminary

## 4.1 Boolean Circuits

We describe a formal model for Boolean circuits $C : \{0,1\}^n \to \{0,1\}^m$ over the gate set of *all two-bit Boolean functions*.

The canonical description $\langle C \rangle$ of a size-$s$ circuit $C : \{0,1\}^n \to \{0,1\}^m$ is a sequence of gates $(g_1, g_2, \ldots, g_s)$, where each gate $g_i = (i, j, k, f)$ is a tuple consisting of:

- The output wire name $i$
- The two child wires $j < k < i$, and
- A Boolean function $f : \{0,1\} \times \{0,1\} \to \{0,1\}$.

The circuit evaluation $C(x)$ is executed by evaluating each gate $g_i(x)$ from 1 to $s$, where for $1 \leq i \leq n$ we define $g_i(x) = x_i$, while for any $i > n$ we have that $g_i(x) = f(g_j(x), g_k(x))$ for gate $g_i = (i, j, k, f)$. The *output wires* of $C$ are the wires $g_{s-m+1}, \ldots, g_s$, and so we have that $C(x) = (g_{s-m+1}(x), \ldots, g_s(x))$.

We sometimes allow gate names to come from an alphabet $\Sigma$ which is different from $[s]$ in order to encode extra information about the circuit topology. For example, in a *layered* circuit,[6] the name of a wire consists of its layer along with its position within the layer. Throughout this paper, we typically assume that our circuits are layered.

### 4.1.1 Universal Circuits

**Syntax.** A universal circuit construction consists of the following polynomial-time algorithms:

$\mathsf{UC.Gen}(1^s, 1^n) \to U$. Given as input a circuit size $s$ and an input length $n$, this algorithm outputs a universal circuit $U$. $U$ takes in as input a circuit description $C$ of size $s$ and an input $x \in \{0,1\}^n$, and outputs $C(x)$.

$\mathsf{UC.Map}(U, C, S) \to S'$. Given as input a universal circuit $U$, a circuit $C$, and a set of wire indices $S$ in $C$, this algorithm outputs a set of wire indices $S'$ in $U$.

$\mathsf{UC.Proc}(U, C, \mathbf{v}'_{S'}) \to \mathbf{v}_S$**.** Given as input a universal circuit $U$, a circuit $C$, and a set of wire values $\mathbf{v}'_{S'}$ in $U$, this algorithm outputs the corresponding wire values $\mathbf{v}_S$ in $C$.

**Construction.** Given $s, n$, we construct the universal circuit $U(\langle C \rangle, x)$ as follows. We will construct a sequence of sub-circuits $\{U_i\}_{1 \leq i \leq s}$, where $U_i$ evaluates the $i$-th gate of $C$. For $1 \leq j \leq s$, let $g^{(j)}_{\mathsf{out}}$ denote the output wire of $U_j$.

For the $i$-th gate $g_i$ in $C$:

- For every $1 \leq j < i$, the wire $g^{(j)}_{\mathsf{out}}$ is sent through an $O(\log s)$-size circuit that outputs $(1, b)$ if $j$ is equal to one of the children of $g_i$, where $b$ is the wire value of $g^{(j)}_{\mathsf{out}}$; otherwise, the output of this circuit is $(0, 0)$.

[6] A layered circuit is a circuit where each wire belongs to a layer, and every gate connects an output wire in layer $i$ to two input wires in layer $i - 1$.

- Then, these $i-1$ outputs are converted into *two* outputs via an $O(\log s)$-depth circuit that removes all of the computed $(0,0)$ values. When evaluated honestly, these two wires will be assigned the values $(g^{(j)}_{\mathsf{out}}, g^{(k)}_{\mathsf{out}})$ (in that order).
- Finally, a constant-size circuit is used to evaluate the function $f$ (in the description of $g_i$) on these two wires. The result is the output wire of $U_i$.

The output of $U$ is defined to be $(g^{(s-m+1)}_{\mathsf{out}}, \ldots, g^{(s)}_{\mathsf{out}})$. By construction, the output wire of each $U_i$ in an honest evaluation of $U(\langle C\rangle, x)$ will be exactly $g_i(x)$, and thus the output of $U$ will be $C(x)$.

Note that the universal circuit $U$ is a fixed circuit that does not depend on $C$ or $x$, with size $O(s^2 \log s)$. Moreover, without loss of generality, we can always write $U$ as a $\mathsf{NAND}$-circuit (every gate is a $\mathsf{NAND}$ gate).

For a circuit $C$ and a set of wire indices $S$ in $C$, we define $\mathsf{UC.Map}(U, C, S)$ to be the output wires of $\{U_i\}_{i\in S}$, plus the wires that correspond to the data flow from $g^{(j)}_{\mathsf{out}}, g^{(k)}_{\mathsf{out}}$ to $U_i$, where $j, k$ are the two children of gate $g_i$ in $C$. Finally, given a set of wire values $\mathbf{v}'_{S'}$ in $U$, $\mathsf{UC.Proc}(U, C, \mathbf{v}'_{S'})$ simply extracts the output wire values of $\{U_i\}_{i\in S}$ from $\mathbf{v}'_{S'}$.

**Properties.** It can be shown that the above construction satisfies the following properties. Furthermore, they can be formalized in the Extended Frege ($\mathcal{EF}$) system.

**Evaluation Correctness.** For any $s, n \in \mathbb{N}$, any circuit $C$ of size $s$ with $n$ input wires, and any input $x \in \{0,1\}^n$, let $U \leftarrow \mathsf{UC.Gen}(1^s, 1^n)$, then $U(C, x) = C(x)$.

**Efficiency.** For any $s, n \in \mathbb{N}, S \subseteq [s]$, let $U \leftarrow \mathsf{UC.Gen}(1^s, 1^n)$, and $S' \leftarrow \mathsf{UC.Map}(U, C, S)$. Then $\mathsf{UC.Gen}$ runs in time $\mathsf{poly}(s)$, $\mathsf{UC.Map}$ runs in time $\mathsf{poly}(|S|, \log s)$, and $\mathsf{UC.Proc}$ runs in time $\mathsf{poly}(|S'|)$. Moreover, $|S'|$ is at most $O(|S| \cdot \log^2 s)$.

**Mapping Correctness.** For any $s, n \in \mathbb{N}$, let $U \leftarrow \mathsf{UC.Gen}(1^s, 1^n)$. Then there exists an injective mapping $M : [s] \to [|U|]$ such that for any circuit $C$ of size $s$ with $n$ input wires, the following hold:

- For any $i \in [s]$, the $M(i)$-th wire value in $U(C, x)$ is equal to the $i$-th wire value in $C(x)$ for any input $x \in \{0,1\}^n$;
- For any set of wire indices $S$ in $C$, if $S' \leftarrow \mathsf{UC.Map}(U, C, S)$, then $M(S) \subseteq S'$;
- For any locally consistent assignment $\mathbf{v}'_{S'}$ to the wires in $S'$ in $U$, let $\mathbf{v}_S \leftarrow \mathsf{UC.Proc}(U, C, \mathbf{v}'_{S'})$, then $\mathbf{v}_S = \mathbf{v}'_{M(S)}$ and $\mathbf{v}_S$ is a locally consistent assignment to the wires in $S$ in $C$.

**Mapping Monotonicity.** For any $s, n \in \mathbb{N}$, any circuit $C$ of size $s$ with $n$ input wires, and any sets of wire indices $S_1 \subseteq S_2$ in $C$, if $U \leftarrow \mathsf{UC.Gen}(1^s, 1^n)$, $S'_1 \leftarrow \mathsf{UC.Map}(U, C, S_1)$, and $S'_2 \leftarrow \mathsf{UC.Map}(U, C, S_2)$, then $S'_1 \subseteq S'_2$.

## 4.2 Probabilistically Checkable Proofs

Let $C : \{0,1\}^{|x|} \times \{0,1\}^{|w|} \to \{0,1\}$ be a Boolean circuit. We define the following language

$$\mathsf{C\text{-}SAT} = \{x \mid \exists w : C(x, w) = 1\}.$$

**Syntax.** A PCP for $\mathsf{C\text{-}SAT}$ is a tuple of polynomial-time algorithms $(\mathsf{PCP.P}, \mathsf{PCP.Q}, \mathsf{PCP.V})$, with the following syntax.

- $\mathsf{PCP.P}(C, x, \omega)$ : The prover algorithm takes as input the circuit $C$, an instance $x$ and its witness $\omega$, and outputs a PCP proof $\pi \in \{0,1\}^*$.
- $\mathsf{PCP.Q}(C, r)$ : On input the circuit $C$ and the random coin $r$, the query algorithm generates a subset $Q \subseteq [|\pi|]$, and a state $\mathsf{st}$. The query set does not depend on the instance $x$.
- $\mathsf{PCP.V}(x, \mathsf{st}, \pi')$ : On input an instance $x$, a state $\mathsf{st}$, and a binary string $\pi' \in \{0,1\}^{|Q|}$, the online verification algorithm $\mathsf{PCP.V}$ *deterministically* decides to accept (output 1) or reject (output 0).

**Theorem 4.1** (PCP with Fast Online Verification [CJJ22, Lemma 1])**.** *There exist a PCP and a function $\rho(s) = 1 - 1/\mathrm{polylog}(s)$ with the following properties.*

- ***Completeness:*** *For any circuit $C$, any instance $x \in \mathsf{C\text{-}SAT}$, and any witness $\omega$ for $x$, we have*
$$\Pr_r\left[\pi \leftarrow \mathsf{PCP.P}(C, x, \omega), (Q, \mathsf{st}) \leftarrow \mathsf{PCP.Q}(C, r) : \mathsf{PCP.V}(x, \mathsf{st}, \pi|_Q) = 1\right] = 1.$$
- ***Knowledge Soundness:*** *There exists a deterministic polynomial-time extractor $\mathsf{PCP.E}$ such that, for any circuit $C$, any instance $x$, and any PCP proof $\pi^*$, if*
$$\Pr_r[(Q, \mathsf{st}) \leftarrow \mathsf{PCP.Q}(C, r) : \mathsf{PCP.V}(x, \mathsf{st}, \pi^*|_Q) = 1] > \rho(|C|),$$
*then $C(x, \mathsf{PCP.E}(C, x, \pi^*)) = 1$.*
- ***Fast Online Verification:*** *The query algorithm $\mathsf{PCP.Q}(C, r)$ runs in time $\mathsf{poly}(|C|)$, and outputs a set $Q$ of size $\mathrm{polylog}(|C|)$ and a state $\mathsf{st}$ of length $\mathsf{poly}(|x|, \log|C|)$. The online verification algorithm $\mathsf{PCP.V}$ runs in time $\mathsf{poly}(|x|, \log|C|)$.*
- ***Proof Length:*** *The randomness $r$ has length $O(\log|C|)$, and the PCP proof size is $|\pi| = \mathsf{poly}(|C|)$.*

The bound on the length of $r$ is not stated in [CJJ22], but their PCP implicitly satisfies it.

### 4.3 Correlation Intractable Hash Functions

**Definition 4.2** (Somewhere Perfect Correlation Intractable Hash for Searchable Relations)**.** *Let $\mathcal{C} = \{\mathcal{C}_\lambda(\cdot,\cdot)\}_{\lambda\in\mathbb{N}}$ be a circuit class. A hash function family $(\mathsf{Gen}, \mathsf{Hash})$ with a fake-key generation algorithm $\mathsf{PerfGen}$ is* somewhere perfectly correlation intractable *for $\mathcal{R}$ if*

1. *$\mathsf{PerfGen}(1^\lambda, C)$, where $C \in \mathcal{C}_\lambda$ is a circuit.*
2. *For any circuit sequence $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ with $C_\lambda \in \mathcal{C}_\lambda$, we have the following property:*
   - *the distribution ensembles $\{\mathsf{PerfGen}(1^\lambda, C_\lambda)\}_{\lambda\in\mathbb{N}}$ and $\{\mathsf{Gen}(1^\lambda)\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable, and*
   - *under the key $\mathsf{PerfGen}(1^\lambda, C_\lambda)$, the hash function is perfectly correlation intractable, namely,*
$$\Pr_{k\leftarrow\mathsf{PerfGen}(1^\lambda, C_\lambda)}\left[\exists x : \mathsf{Hash}(k, x) = C_\lambda(x)\right] = 0.$$

### 4.4 Fully Homomorphic Encryption

The following preliminary of gate-by-gate FHE is due to [JKLM25].

In this section, we define the type of fully homomorphic encryption used in this paper. We make use of a *leveled, gate-by-gate* FHE satisfying a *malicious* correctness property: for any pair of ciphertexts $\mathsf{ct}_0, \mathsf{ct}_1$ that decrypt to bits $x_0, x_1$, it should be the case that $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{ek}}(f, \mathsf{ct}_0, \mathsf{ct}_1)) = f(x_0, x_1)$. Moreover, our definition explicitly describes the evaluation key as a sequence of $d$ (small) keys, since this structure is relevant to our $\mathsf{SNARG}$ construction.

**Syntax.** A leveled, gate-by-gate fully homomorphic encryption scheme consists of a fixed key/ciphertext size $\ell = \ell(\lambda) = \mathsf{poly}(\lambda)$ and the following polynomial time algorithms:

$\mathsf{FHE.Setup}(1^\lambda, 1^d) \to (\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_d, \mathsf{sk}_0, \ldots, \mathsf{sk}_d)$. This is a probabilistic algorithm that takes as input a security parameter $1^\lambda$ and a circuit depth $1^d$. It outputs a public key $\mathsf{pk} \in \{0,1\}^\ell$, a sequence of $d$ evaluation keys $\mathsf{ek}_1, \ldots, \mathsf{ek}_d \in \{0,1\}^\ell$, and a sequence of $d+1$ secret keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_d \in \{0,1\}^\ell$.

$\mathsf{FHE.Enc}_{\mathsf{pk}}(b) \to c$. This is a probabilistic algorithm that takes as input a public key $\mathsf{pk}$ and a bit $b \in \{0,1\}$. It outputs a ciphertext $c \in \{0,1\}^\ell$.

$\mathsf{FHE.Dec}_{\mathsf{sk}}(c) \to b$. This is a deterministic algorithm that takes as input a secret key $\mathsf{sk}$ and a ciphertext $c \in \{0,1\}^\ell$. It outputs a bit $b \in \{0,1\}$.

$\mathsf{FHE.GateEval}_{\mathsf{ek}}(f, c_1, c_2) \to c^*$. This is a deterministic algorithm that takes as input an evaluation key $\mathsf{ek}$, the truth table of a *two input bit* function $f\colon \{0,1\}\times\{0,1\} \to \{0,1\}$ and two ciphertexts $c_1, c_2 \in \{0,1\}^\ell$. It outputs a ciphertext $c^* \in \{0,1\}^\ell$.

By iterating the $\mathsf{FHE.GateEval}_{\mathsf{pk}}$ algorithm many times, one can generically build a *circuit evaluation algorithm*:

$\mathsf{FHE.Eval}_{\mathsf{ek}_1,\ldots,\mathsf{ek}_d}(f, c_1, \ldots, c_n) \to c^*$. This is a deterministic algorithm that takes as input a public key $\mathsf{pk}$, a circuit representing a function $f\colon \{0,1\}^n \to \{0,1\}$ and $n$ ciphertexts $c_1, \ldots, c_n \in \{0,1\}^\ell$. It outputs a ciphertext $c^* \in \{0,1\}^\ell$.

**Definition 4.3** ($\mathsf{FHE}$)**.** *A leveled, gate-by-gate fully homomorphic encryption scheme*

$$\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.GateEval})$$

*is required to satisfy the following properties:*

**Encryption Correctness.** *For any choice of* $(\mathsf{pk}, \mathsf{sk}_0)$ *in the support of* $\mathsf{FHE.Setup}(1^\lambda, 1^d)$*, any* $b \in \{0,1\}$ *and any* $c \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(b)$ *we have* $\mathsf{FHE.Dec}_{\mathsf{sk}_0}(c) = b$*.*

**Honest Evaluation Correctness.** *For any choice of* $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_d, \mathsf{sk}_0, \ldots, \mathsf{sk}_d) \leftarrow \mathsf{FHE.Setup}(1^\lambda, 1^d)$*, any honestly generated ciphertexts* $c_1, \ldots, c_n \in \{0,1\}^\ell$ *such that* $c_i = \mathsf{FHE.Enc}_{\mathsf{pk}}(b_i)$*, and any* layered *circuit* $f\colon \{0,1\}^n \to \{0,1\}$ *of depth* $d$*, if we set* $c = \mathsf{FHE.Eval}_{\mathsf{ek}_1,\ldots,\mathsf{ek}_d}(f, c_1, \ldots, c_n)$ *then* $\mathsf{FHE.Dec}_{\mathsf{sk}_d}(c) = f(b_1, \ldots, b_n)$*.*

**Malicious Gate Correctness.** *For any choice of* $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_d, \mathsf{sk}_0, \ldots, \mathsf{sk}_d) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$*, any index* $0 \le i \le d-1$*, any* $f : \{0,1\}^2 \to \{0,1\}$*, and any ciphertexts* $c_1, c_2 \in \{0,1\}^\ell$*, if* $\mathsf{Dec}_{\mathsf{sk}_i}(c_1) = b_1$ *and* $\mathsf{Dec}_{\mathsf{sk}_i}(c_2) = b_2$*, then* $\mathsf{Dec}_{\mathsf{sk}_{i+1}}(\mathsf{FHE.GateEval}(f, \mathsf{ek}_{i+1}, c_1, c_2)) = f(b_1, b_2)$.

**Security.** *The encryption scheme is semantically secure in the presence of* $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_d)$.

**Theorem 4.4** ([BV11])**.** *Assuming the hardness of Learning with Errors [Reg05], there exists a leveled, gate-by-gate homomorphic encryption scheme.*

Specifically, the properties we require in Definition 4.3 all follow from the (leveled) *bootstrapping* construction of [Gen09], provided that the "base" somewhat homomorphic encryption scheme satisfies perfect correctness. Moreover, the public keys and ciphertexts in this scheme based on LWE-based scheme will be pseudo-random under the assumption LWE assumption.

**Remark 4.5.** *Given a* FHE *scheme, one can extend the definition of the encryption algorithm* FHE.Enc *to take as input a longer message* $m \in \{0,1\}^n$*, as opposed to a single bit.* $\mathsf{FHE.Enc}_{\mathsf{pk}}(m)$ *will simply run* $c_i = \mathsf{FHE.Enc}_{\mathsf{pk}}(m_i)$ *for every* $i \in [n]$*, and set* $c = (c_1, \ldots, c_n) \in \{0,1\}^{n \cdot \ell}$*. Similarly, we extend* FHE.Eval *to take as input a function* $f\colon \{0,1\}^n \to \{0,1\}^u$ *with multi-bit output.*

**Lemma 4.6** (Lemma B.1. in JKLM)**.** *Let* $e$ *be the noise term in LWE. Let's slightly modify the error such that: if* $\|e\|_\infty \ge \sigma\sqrt{(n+1)w}$*, then set* $e = 0$*. This slightly modified version of [GSW13] is a correct leveled fully homomorphic scheme with a* $\mathcal{EF}$*-proof of correctness, i.e., a* $\mathcal{EF}$*-proof of:*

$$\forall (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{FHE.Setup}(1^\lambda, 1^d),\ \forall m_1, \ldots, m_\ell \in \{0,1\},\ \forall C : \{0,1\}^\ell \to \{0,1\} \text{ of depth at most } d,$$

$$\forall \mathsf{ct}_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}}(m_i),\ \mathsf{ct}_{\mathsf{out}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}, C, (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)),$$

*then*

$$b \leftarrow \mathsf{FHE.Dec}(\mathsf{sk}, \mathsf{ct}_{\mathsf{out}}) \quad \text{satisfies} \quad b = C(m_1, \ldots, m_\ell).$$

**Lemma 4.7** ($\mathcal{EF}$ Proof of Malicious Gate Correctness)**.** *Let* FHE *be the gate-by-gate scheme obtained by bootstrapping the scheme of Lemma 4.6. For every* $0 \le i \le d-1$ *and every* $f\colon \{0,1\} \times \{0,1\} \to \{0,1\}$*, the formula*

$$\mathsf{FHE.Dec}_{\mathsf{sk}_{i+1}}\big(\mathsf{FHE.GateEval}_{\mathsf{ek}_{i+1}}(f, c_1, c_2)\big) = f\big(\mathsf{FHE.Dec}_{\mathsf{sk}_i}(c_1), \mathsf{FHE.Dec}_{\mathsf{sk}_i}(c_2)\big),$$

*where* $(\mathsf{pk}, \mathsf{ek}_1, \ldots, \mathsf{ek}_d, \mathsf{sk}_0, \ldots, \mathsf{sk}_d) := \mathsf{FHE.Setup}(1^\lambda, 1^d; r)$ *and* $c_1, c_2$ *are arbitrary ciphertexts, has an* $\mathcal{EF}$ *proof of length* $\mathsf{poly}(\lambda)$.

*Proof.* In the bootstrapping construction, each level $i$ has its own key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ of the scheme of Lemma 4.6, the evaluation key $\mathsf{ek}_{i+1}$ is $\mathsf{FHE.Enc}_{\mathsf{pk}_{i+1}}(\mathsf{sk}_i)$, and $\mathsf{FHE.GateEval}_{\mathsf{ek}_{i+1}}(f, c_1, c_2)$ homomorphically evaluates the circuit $g(\mathsf{sk}) = f(\mathsf{FHE.Dec}_{\mathsf{sk}}(c_1), \mathsf{FHE.Dec}_{\mathsf{sk}}(c_2))$ on $\mathsf{ek}_{i+1}$ under $\mathsf{pk}_{i+1}$. Hence the formula follows from Lemma 4.6 applied to $g$, whose depth is that of FHE.Dec plus one, as in the proof of Theorem A.1. This proof only involves the keys of levels $i$ and $i+1$, so its length is $\mathsf{poly}(\lambda)$. □

### 4.5 Somewhere Statistically Binding Hash Functions(SSB Hash)

**Syntax.** A SSB Hash scheme is a tuple of algorithms ($\mathsf{SSB.Gen}$, $\mathsf{SSB.TGen}$, $\mathsf{SSB.Hash}$, $\mathsf{SSB.Open}$, $\mathsf{SSB.Verify}$, $\mathsf{SSB.Ext}$) described below.

$\mathsf{SSB.Gen}(1^\lambda, 1^N, 1^{|S|}; r)$**:** On input a security parameter $\lambda$, the message length $N$, and the size of a subset $S \subseteq [N]$, the key generation algorithm outputs a uniformly random hash key $\mathsf{hk}$.

$\mathsf{SSB.TGen}(1^\lambda, 1^N, S; r)$**:** On input a security parameter $\lambda$, the message length $N$, the binding subset $S \subseteq [N]$, the trapdoor generation algorithm outputs a hash key $\mathsf{hk}$ and a trapdoor $\mathsf{td}$.

$\mathsf{SSB.Hash}(\mathsf{hk}, m)$**:** On input the hash key $\mathsf{hk}$, a message $m \in \{0,1\}^N$, the hashing algorithm outputs a hash value $c$.

$\mathsf{SSB.Open}(\mathsf{hk}, m, i)$**:** On input the hash key $\mathsf{hk}$, a message $m \in \{0,1\}^N$, an index $i \in [N]$, the opening algorithm outputs a local opening $\pi_i$ for position $i$.

$\mathsf{SSB.Verify}(\mathsf{hk}, c, v, i, \pi_i)$**:** On input the hash key $\mathsf{hk}$, a hash value $c$, a claimed bit $v \in \{0,1\}$, an index $i \in [N]$, and a local opening $\pi_i$, the verification algorithm decides to accept or reject (outputs 1 or 0, respectively) the local opening for message value $v$ at position $i$.

$\mathsf{SSB.Ext}(c, \mathsf{td})$**:** On input a hash value $c$ and the trapdoor $\mathsf{td}$ generated by $\mathsf{SSB.TGen}$ with respect to binding subset $S$, the extraction algorithm outputs an extracted string $v^*_S \in \{0,1\}^{|S|}$.

We require the SSB Hash scheme to satisfy the following properties.

**Key Indistinguishability.** For any n.u. PPT adversary $\mathcal{A}$ and any polynomial $N = N(\lambda)$, there exists a negligible function $\epsilon(\lambda)$ such that

$$\Big|\Pr\left[\ \mathcal{A}(\mathsf{hk}) = 1 \ : \ S \leftarrow \mathcal{A}(1^\lambda, 1^N), \mathsf{hk} \leftarrow \mathsf{SSB.Gen}(1^\lambda, 1^N, 1^{|S|}; r)\ \right] - \Pr\left[\ \mathcal{A}(\mathsf{hk}) = 1 \ : \ S \leftarrow \mathcal{A}(1^\lambda, 1^N), (\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SSB.TGen}(1^\lambda, 1^N, S; r)\ \right]\Big| \leq \epsilon(\lambda).$$

**Opening Completeness.** For any hash key $\mathsf{hk}$, any message $m = (m_1, \ldots, m_N) \in \{0,1\}^N$, and any index $i \in [N]$, we have

$$\Pr\left[\ \mathsf{SSB.Verify}(\mathsf{hk}, c, m_i, i, \pi_i) = 1 \ : \ c \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, m), \pi_i \leftarrow \mathsf{SSB.Open}(\mathsf{hk}, m, i)\ \right] = 1.$$

**Extraction Correctness.** For any subset $S \subseteq [N]$, any index $i \in S$, any trapdoor key $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{SSB.TGen}(1^\lambda, 1^N, S; r)$, any hash value $c$, any bit $v \in \{0,1\}$, and any proof $\pi_i$, we have

$$\Pr\left[(\mathsf{SSB.Verify}(\mathsf{hk}, c, v, i, \pi_i) = 1) \rightarrow (\mathsf{SSB.Ext}(c, \mathsf{td}) \mid_i = v)\right] = 1.$$

Since the extracted value $\mathsf{SSB.Ext}(c, \mathsf{td}) \mid_i$ is unique, extraction correctness implies statistical binding property.

In our paper, we assume that the $\mathsf{SSB}$ Hash scheme supports alphabet size larger than bits, e.g., $\Sigma = \{0,1\}^{\mathsf{poly}(\lambda)}$. The definitions and properties can be easily extended to support a larger alphabet size.

### 4.6 Somewhere Extractable Batch Arguments (BARGs)

The following preliminaries on BARGs are due to [BBK+23].

A batch argument system BARG for an NP language $\mathcal{L}$ enables proving that $k$ NP statements are true with communication cost that is polylogarithmic in $k$. There are many BARG variants which are known to be existentially equivalent under mild computational assumptions (see, e.g., [CJJ22, KVZ21, KLVW23]). In this work, for simplicity in our constructions, we make use of an argument system for what we call "batch index Turing machine SAT" (BatchTMSAT), defined below.

**Definition 4.8.** *The language* BatchIndexTMSAT *consists of instances of the form* $x = (M, z, k, T)$*, where:*

- $M$ *is the description of a Turing machine.*
- $z$ *is an input string (to* $M$*)*
- $k$ *is a batch size, and*
- $T$ *is a running time.*

*An instance* $x = (M, z, k, T)$ *is in* BatchIndexTMSAT *if for all* $i \leq i \leq k$*, there exists a string* $w_i$ *such that* $M(z, i, w_i)$ *accepts within* $T$ *steps.*

We sometimes use the notation $\mathcal{R}(x, i, w_i)$ to denote the relation with instance $(x, i)$ and corresponding witness $w_i$.

**Syntax.** A (publicly verifiable and non-interactive) somewhere extractable batch argument system BARG for BatchIndexTMSAT consists of the following polynomial time algorithms:

$\mathsf{Gen}(1^\lambda, 1^n, 1^m, k, i^*) \to (\mathsf{crs}, \mathsf{td})$. This is a probabilistic polynomial-time algorithm that takes as input a security parameter $1^\lambda$, input length $1^n$, witness length $1^m$, a parameter $k$, and an index $i^* \in [k]$. It outputs a common reference string crs along with a trapdoor td.

$\mathcal{P}(\mathsf{crs}, M, z, 1^T, w_1, \ldots, w_k) \to \pi$. This deterministic polynomial-time algorithm takes as input a crs, Turing machine $M$, input $z$, runtime $1^T$, and $k$ witnesses $w_1, \ldots, w_k$. It outputs a proof $\pi$.

$\mathcal{V}(\mathsf{crs}, x, \pi) \to 0/1$. This deterministic polynomial-time algorithm takes as input a crs, instance $x = (M, z, k, T)$, and a proof $\pi$. It outputs a bit (1 to accept, 0 to reject).

$\mathsf{Extract}(\mathsf{td}, \pi) \to w^*$. This deterministic polynomial-time algorithm takes as input a trapdoor td and a proof $\pi$. It outputs a single witness $w^*$.

**Definition 4.9** (BARG)**.** *A somewhere-extractable batch argument system* $\mathsf{BARG} = (\mathsf{Gen}, \mathcal{P}, \mathcal{V}, \mathsf{Extract})$ *for* BatchIndexTMSAT *is required to satisfy the following properties:*

**Completeness.** *For any* $n \in \mathbb{N}$*, any* $k(n), n(n), m(\lambda), T(\lambda) \leq 2^n$*, any instance* $x = (M, z, k, T) \in$ BatchIndexTMSAT *with* $|M| + |z| = n$*, any corresponding witnesses* $w_1, \ldots, w_k \in \{0,1\}^m$ *and any index* $i^* \in [k]$*,*

$$\Pr\left[ \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \;:\; \begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^m, i^*), \\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, M, z, 1^T, w_1, \ldots, w_k) \end{array} \right] = 1.$$

**Efficiency.** *In the completeness experiment above,* $|\mathsf{crs}|+|\pi| \leq m \cdot \mathsf{poly}(n, \log(knT))$. *The running time of the verifier is at most* $\mathsf{poly}(|\mathsf{crs}| + |\pi|) + \mathsf{poly}(n) \cdot |x|$.

**Index hiding.** *For any poly-size adversary* $\mathcal{A}$ *and any polynomials* $k(\lambda), n(\lambda), m(\lambda)$, *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $n \in \mathbb{N}$ *and every pair of indices* $i_0, i_1 \in [k]$,

$$\Pr\left[\mathcal{A}(\mathsf{crs}) = b \ : \ \begin{array}{c} b \leftarrow \{0,1\}, \\ (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^m, i_b) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(n).$$

**Somewhere argument of knowledge.** *For any poly-size adversary* $\mathcal{A}$*and any polynomials* $k(n)$, $n(\lambda), m(\lambda), T(\lambda)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any index* $i^* \in [k]$ *and for every* $n \in \mathbb{N}$,

$$\Pr\left[\begin{array}{c} \mathcal{V}(\mathsf{crs}, x, \pi) = 1 \\ \wedge\ (x, i^*, w^*) \notin \mathcal{R} \end{array} \ : \ \begin{array}{c} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda, 1^n, 1^m, i^*) \\ (M, z, \pi) = \mathcal{A}(\mathsf{crs}) \\ w^* \leftarrow \mathsf{Extract}\,(\mathsf{td}, \pi) \end{array}\right] \leq \mathsf{negl}(n).$$

**Remark 4.10.** *We say that a* BARG *scheme is* $T^*$*-secure, for* $T^* = T^*(n)$, *if the index hiding property and the somewhere argument of knowledge property hold w.r.t. a* $\mathsf{poly}(T^*)$*-size adversary (as opposed to a* $\mathsf{poly}(\lambda)$*-size), and the advantage probability is* $\mathsf{negl}(T^*)$ *(as opposed to* $\mathsf{negl}(n)$*). In addition, the index hiding property and the somewhere argument of knowledge property are required to hold even if the parameters* $k(\lambda), n(\lambda), m(\lambda), T(\lambda)$ *are polynomial in* $T^*(\lambda)$. *We refer to these properties as* $T^*$*-index-hiding and* $T^*$*-somewhere-argument-of-knowledge, respectively.*

Throughout this paper, when we refer to a BARG, we will implicitly mean a BARG for BatchIndexTMSAT.

**Remark 4.11.** *Given an* BARG, *one can naturally extend the definition of the key generation algorithm* Gen *to take as input an index* set $I \subset [k]$, *as opposed to a single index.* $\mathsf{Gen}(1^\lambda, 1^n, 1^m, I)$ *will simply run* $\mathsf{Gen}(1^\lambda, 1^n, 1^m, i)$ *for every* $i \in I$. *The prover algorithm* $\mathcal{P}$, *given a* crs *that encodes the* $|I|$ *indices, will simply generate* $|I|$ *proofs (one for each* crs*), and the verifier will check these* $|I|$ *proofs independently. In this situation, the* BARG *trapdoor* $\mathsf{td} = (\mathsf{td}_1, \ldots, \mathsf{td}_{|I|})$ *is written in* $|I|$ *components, and the scheme inherits a strong form of index hiding: given* BARG.crs, *the* $j$*th index* $i_j$ *remains hidden even given all* $|I| - 1$ *other trapdoors* $\mathsf{td}_{i_{j'}}$.

**Theorem 4.12** ([CJJ22, WW22, HJKS22, KLVW23])**.** *There exists an* BARG *for* BatchIndexTMSAT *assuming* LWE *or* DLIN *or (subexponential* DDH *and* QR*).*

# 5 Cryptographic Extended Frege Proofs

In this section, we define Cryptographic Extended Frege proofs ($\mathcal{CEF}$ proofs), which extend Extended Frege proofs by allowing a Cryptographic Gadget Substitution rule that captures computational indistinguishability. We will also prove some basic properties of $\mathcal{CEF}$ proofs.

**Definition 5.1** (Circuits as Propositional Formulas)**.** *For every circuit* $C$ *with input wires* **in** *and output wires* **out**, *we use the notation* $\mathbf{out} \coloneqq C(\mathbf{in})$ *to denote the set of formulas that represent gate computation in* $C$. *Namely, denoting by* $G$ *the set of all gates in* $C$,

$$\{\mathbf{out} \coloneqq C(\mathbf{in})\} \triangleq \{\varphi_g\}_{g \in G},$$

*where* $\varphi_g \coloneqq o \leftrightarrow g(l, r)$*, with* $l$ *and* $r$ *being the input wires to* $g$*, and* $o$ *being the output wire. If* **in** *and* **out** *are clear from the context, we suppress them in the notation.*

We first define Probabilistic Extended Frege proofs ($\mathcal{PEF}$ proofs), which can be seen as $\mathcal{CEF}$ proofs for a fixed security parameter $\lambda$.

**Definition 5.2** (Probabilistic Extended Frege Proofs ($\mathcal{PEF}$ Proofs))**.** *In probabilistic extended Frege logic, each variable is either a "proper variable", a "random variable", or an "extension variable". A proper variable represents a binary value that can be either true or false. A random variable is a formal symbol, whose intended meaning is a bit sampled uniformly and independently from* $\{0, 1\}$*. Each extension variable is defined by an extension rule in the following.*

*Let* $t, s, \ell$ *be integers, and let* $\epsilon \in [0, 1]$ *be a real number. We say that a formula* $\phi$ *admits a* $(t, s, \ell, \epsilon)$-Probabilistic Extended Frege proof *if there exists a sequence of configurations*

$$\Phi_1, \Phi_2, \ldots, \Phi_u,$$

*where the initial configuration is* $\Phi_1 = \varnothing$*, the final configuration is* $\Phi_u = \{\phi\}$*, and for every* $1 < i \leq u$*,* $\Phi_i$ *is a set of formulas that is derived from* $\Phi_{i-1}$ *via one of the following rules:*

- **Logical Inference.** *There exists a constant number of formulas* $\theta_1, \ldots, \theta_c \in \Phi_{i-1}$ *and a formula* $\theta \in \Phi_i$ *such that*
$$(\theta_1 \wedge \ldots \wedge \theta_c) \to \theta$$
*is a* tautology *and* $\Phi_i = \Phi_{i-1} \cup \{\theta\}$*.*
- **Extension.** *There exists a formula* $\phi$ *such that* $\Phi_i = \Phi_{i-1} \cup \{\phi\}$ *and* $\phi$ *is in the following form:*
$$e \leftrightarrow \psi,$$
*where* $e$ *is an extension variable and* $\psi$ *is a formula that only uses variables in* $\Phi_{i-1}$*.*
- **Erasure.** *There exists a formula* $\theta \in \Phi_{i-1}$ *such that* $\Phi_i = \Phi_{i-1} \setminus \{\theta\}$*, where* $\theta$ *is either:*
  - *an extension rule for a variable* $e$ *and* $e$ *does not appear in* $\Phi_i$*, or*
  - $\theta$ *is derived from a logical inference rule.*
- **Cryptographic Gadget Substitution.** *There exists a subset of formulas* $F$ *such that*
$$\Phi_{i-1} = F \cup \{\mathbf{y} \coloneqq f(\mathbf{z}, \mathbf{c})\}, \quad \Phi_i = F \cup \{\mathbf{y} \coloneqq g(\mathbf{z}, \mathbf{c}')\},$$
*where* $\mathbf{y}$ *is a vector of extension variables,* $\mathbf{z}$ *is a vector of arbitrary variables,* $\mathbf{c}, \mathbf{c}'$ *are vectors of random variables that do not appear in* $F$*;* $F$ *may involve* $\mathbf{y}$ *and* $\mathbf{z}$*, but* $\mathbf{c}, \mathbf{c}'$ *and all the intermediate variables for the intermediate wires in the computation of* $f, g$ *do not appear in* $F$*; and* $f, g$ *are two circuits that satisfy the following condition:*

  *For any circuit* $\mathcal{D}$ *of size at most* $1/\epsilon$*, for any fixed value of* $\mathbf{z}$*,*
$$\left| \Pr_{\mathbf{c}}[\mathcal{D}(f(\mathbf{z}, \mathbf{c})) = 1] - \Pr_{\mathbf{c}'}[\mathcal{D}(g(\mathbf{z}, \mathbf{c}')) = 1] \right| \leq \epsilon,$$
  *Furthermore, we require either* $|\mathbf{z}| \leq \log s$ *or* $|\mathbf{z}| \leq \ell \wedge |\mathbf{y}| \leq \log \ell$*.*

*In addition, the following global conditions must be satisfied:*

- **Length:**
$$\sum_{i=1}^{u} \sum_{\phi \in \Phi_i} |\phi| \leq t,$$
*where $|\phi|$ is the size of $\phi$.*

- **Space:**
$$\max_{i \in [u]} \sum_{\phi \in \Phi_i} |\phi| \leq s.$$

**Remark 5.3** (Derivation of a Set of Formulas)**.** *The definition above for a single formula $\phi$ can be extended naturally to a set of formulas $\Phi$. Namely, everything remains the same except that we require the final configuration $\Phi_u = \Phi$.*

**Remark 5.4** (Partial Derivation)**.** *The definition above can be extended naturally to start from one configuration $\Phi'$ and derive another configuration $\Phi$. Namely, everything remains the same except that we require the initial configuration $\Phi_1 = \Phi'$ and the final configuration $\Phi_u = \Phi$. In this case, we say there is a $(t, s, \ell, \epsilon)$-$\mathcal{PEF}$ proof* from $\Phi'$ to $\Phi$.

We demonstrate the power of the Cryptographic Gadget Substitution rule by showing how it can derive various weaker rules.

**Distribution Shift.** This is the special case where $\mathbf{z} = \varnothing$, so $\mathbf{y}$ is effectively a vector of efficiently sampleable random variables. This allows us to change the distribution and/or sampling algorithm of $\mathbf{y}$, as long as the two distributions are $\epsilon$-indistinguishable.

**Random Inference.** Let $\Phi$ be a configuration, and let $\phi$ be a formula that only involves random variables $\mathbf{c}$. If $\phi$ is satisfied with probability $1 - \epsilon$, then we can derive a new configuration $\Phi' = \Phi \cup \{\phi\}$. This is done by setting $(y, \mathbf{c}) = (\phi(\mathbf{c}'), \mathbf{c}')$ where $\mathbf{c}'$ is a fresh set of random variables not used anywhere else, adding a formula $y = \phi(\mathbf{c})$ via $\mathcal{EF}$ rules, and then switching to $(y, \mathbf{c}) = (1, \mathbf{c}')$.

Then, we define Cryptographic Extended Frege proofs ($\mathcal{CEF}$ proofs) as families of $\mathcal{PEF}$ proofs with negligible error.

**Definition 5.5** (Cryptographic Extended Frege Proofs ($\mathcal{CEF}$ Proofs))**.** *Let $t = t(\lambda), s = s(\lambda), \ell = \ell(\lambda)$ be three polynomials. We say a family of sets of propositional formulas $\{\Phi_\lambda\}_{\lambda \in \mathbb{N}}$ admits a* $(t, s, \ell)$-Cryptographic Extended Frege proof*, if there exists a polynomial $u = u(\lambda)$ and a family of sequence of configurations*
$$\left\{ \left( \Phi_\lambda^{(1)}, \ldots, \Phi_\lambda^{(u(\lambda))} \right) \right\}_{\lambda \in \mathbb{N}},$$
*such that for every constant $c > 0$, and for all sufficiently large $\lambda$, the sequence $\Phi_\lambda^{(1)}, \ldots, \Phi_\lambda^{(u(\lambda))}$ is a $(t(\lambda), s(\lambda), \ell(\lambda), \lambda^{-c})$-Probabilistic Extended Frege proof for $\Phi_\lambda$.*

*A $(t, s, \ell)$-$\mathcal{CEF}$ proof from $\{\Phi'_\lambda\}_{\lambda \in \mathbb{N}}$ to $\{\Phi_\lambda\}_{\lambda \in \mathbb{N}}$ can be defined similarly.*

**Remark 5.6.** *Using subexponential security, we can extend the first case of the Cryptographic Gadget Substitution rule to allow* $|\mathbf{z}| = \text{polylog}(t)$. *However, for simplicity, we will not use this generalization in this work.*

Intuitively, if we have a proof for a set of formulas $\Phi$ that involves a proper variable $z$, then we can substitute $z$ with anything and still obtain a corresponding proof, just like in most proof systems. This is formalized in the following lemma, with a caveat that the substituted term must be independent of all randomness in the original proof.

**Lemma 5.7** (Variable Substitution)**.** *Suppose a set of propositional formulas* $\Phi$ *admits a* $(t, s, \ell, \epsilon)$-$\mathcal{PEF}$ *proof. Let* $\mathbf{z}$ *be a set of proper variables in* $\Phi$*, i.e.,* $\mathbf{z}$ *do not come from extension rules. Let* $f(\mathbf{y}, \mathbf{c})$ *be a circuit that takes in proper variables* $\mathbf{y}$ *and random variables* $\mathbf{c}$*, and none of* $\mathbf{c}$ *appears in* $\Phi$*.*

*Then we have a* $(t + |f| \cdot t + |f|^2, s + |f|, \ell, \epsilon)$-$\mathcal{PEF}$ *proof for* $\Phi \cup \{\mathbf{z} \coloneqq f(\mathbf{y}, \mathbf{c})\}$.

*Proof.* We note two simple properties of our $\mathcal{PEF}$ proof definition:

1. A proper variable, once introduced, can never be removed or re-defined.
2. Whatever rules we apply to a proper variable, can be applied in the same way even if the variable is no longer proper, as long as it is still independent of all randomness in the original proof.

Let $\varnothing = \Phi^{(1)}, \ldots, \Phi^{(u)} = \Phi$ be the sequence of configurations that proves $\Phi$. Without loss of generality, we can assume that none of the configurations $\Phi^{(i)}$ contains $\mathbf{c}$. Then we can construct another sequence of configurations as follows:

- $\Phi'_0 = \varnothing$;
- $\Phi'_i = \Phi^{(i)} \cup \{\mathbf{z} \coloneqq f(\mathbf{y}, \mathbf{c})\}$.

It's easy to see that this is almost a valid sequence of configurations that proves $\Phi \cup \{\mathbf{z} \coloneqq f(\mathbf{y}, \mathbf{c})\}$, except for the gap between $\Phi'_0 = \varnothing$ and $\Phi'_1 = \{\mathbf{z} \coloneqq f(\mathbf{y}, \mathbf{c})\}$. This gap can be filled by multiple uses of the extension rule. □

**Remark 5.8.** *This lemma can be naturally generalized to partial derivations and* $\mathcal{CEF}$ *proofs.*

*Intuitively this lemma should appear as an axiom for Probabilistic Extended Frege proof. However, that would lead to more complex proofs in a later stage, similar to proving the equivalence between substitution Frege and Extended Frege. For simplicity, we leave it as future work.*

# 6 SNARGs via $\mathcal{CEF}$-Proof of Non-membership

In this section, we first present our SNARG construction, which is essentially the same as [JKLV24] with some minor modifications. Then, we state our main theorems that reduce the extractability and/or security of our SNARG construction to the existence of $\mathcal{CEF}$ proofs of witness uniqueness.

### 6.1 SNARG Construction

**Syntax.** Let $L = L(\lambda)$ be a polynomial. A SNARG with CRS length $L$ for a language

$$\mathcal{L} = \{\mathbf{x} \mid \text{there exists } \mathbf{w} : C(\mathbf{x}, \mathbf{w}) = 1\}$$

consists of the following polynomial-time algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. This is a probabilistic polynomial-time algorithm that takes as input a security parameter $1^\lambda$. It outputs a uniformly random CRS of length $L(\lambda)$: $\mathsf{crs} \leftarrow \{0,1\}^{L(\lambda)}$.
- $\mathcal{P}(\mathsf{crs}, C, \mathbf{x}, \mathbf{w}) \to \Pi$. This is a deterministic polynomial-time algorithm that takes as input a $\mathsf{crs}$, $\mathsf{NP}$ relation circuit $C$, input $\mathbf{x}$, and a witness $\mathbf{w}$. It outputs a proof $\Pi$.
- $\mathcal{V}(\mathsf{crs}, C, \mathbf{x}, \Pi) \to 0/1$. This is a deterministic polynomial-time algorithm that takes as input a $\mathsf{crs}$, $\mathsf{NP}$ relation circuit $C$, instance $\mathbf{x}$, and a proof $\Pi$. It outputs a bit (1 to accept, 0 to reject).

**Construction.** Let $L = L(\lambda), \ell = \ell(\lambda)$ be two polynomials. Our construction of SNARGs with CRS length $L \cdot \mathsf{poly}(\lambda)$ and proof size $\ell \cdot \mathsf{poly}(\lambda)$ is essentially the same as [JKLV24], with a slight simplification that we only use 2 FHE ciphertexts.

- **Setup:** The setup algorithm takes as input the security parameter $\lambda$ and does the following:
  1. Generate two $\mathsf{FHE}$ keys: $(\mathsf{pk}_i, \mathsf{ek}_i, \mathsf{sk}_i) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$ for $i \in \{1,2\}$.
  2. Generate two $\mathsf{FHE}$ ciphertexts of 0's of length $L(\lambda)$: $\mathsf{ct}_i \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_i}(0^{L(\lambda)})$, for $i \in \{1,2\}$.
  3. Let $U_\lambda \leftarrow \mathsf{UC.Gen}(1^{L(\lambda)}, 1^{L(\lambda)})$ be the universal circuit that takes in a circuit representation and an input, both of size at most $L(\lambda)$, and outputs the evaluation result.
  4. Generate an SSB hash key $\mathsf{hk}_{\mathsf{SSB}} = (\mathsf{hk}^{(0)}_{\mathsf{SSB}}, \mathsf{hk}^{(1)}_{\mathsf{SSB}}, \ldots, \mathsf{hk}^{(\ell)}_{\mathsf{SSB}})$ for vectors of at most $|U_\lambda|$ FHE ciphertexts, consisting of $\ell+1$ independent SSB keys, which we call its components. Components $1, \ldots, \ell$ support extraction of $\ell(\lambda)$ indices, one index each. Component 0 is a reference component: it is never used for extraction, and only serves the security proof.
  5. Generate three CRSs $\mathsf{crs}_{\mathsf{BARG}} = (\mathsf{crs}_{\mathsf{BARG},C}, \mathsf{crs}_{\mathsf{BARG},1}, \mathsf{crs}_{\mathsf{BARG},2})$ of $\mathsf{BARG}$ with index 1, for the Turing machines $M_C$, $M_1$, and $M_2$ defined below, each supporting $|U_\lambda|$ instances, each with witness length at most $\lambda^2$ FHE ciphertexts and SSB hash values.

  Finally, it outputs the $\mathsf{crs}$ as $(\mathsf{hk}_{\mathsf{SSB}}, \{\mathsf{ek}_i, \mathsf{ct}_i\}_{i\in\{1,2\}}, \mathsf{crs}_{\mathsf{BARG}})$.
- **Prover:** We will use the following convention: we use $h$ to denote hash values known to both parties, and use $v$ to denote hash values chosen by the prover. Each hash value $\mathsf{Hash}(\mathsf{hk}_{\mathsf{SSB}}, X)$ is the tuple of the hashes of $X$ under the $\ell+1$ components, and for such a tuple $h$ we write $h^{(\gamma)}$ for its entry under component $\gamma$. Given $(\mathsf{crs}, C, \mathbf{x}, \mathbf{w})$ where $C$ is the $\mathsf{NP}$ relation circuit, $\mathbf{x}$ is the statement, and $\mathbf{w}$ is the witness, the prover algorithm computes the proof $\Pi$ as follows:
  1. Let $h_{\mathbf{x}} = \mathsf{Hash}(\mathsf{hk}_{\mathsf{SSB}}, \mathbf{x})$, $h_C = \mathsf{Hash}(\mathsf{hk}_{\mathsf{SSB}}, \langle C \rangle)$ for the description $\langle C \rangle$ of $C$, and $h_i = \mathsf{Hash}(\mathsf{hk}_{\mathsf{SSB}}, (\mathsf{ct}_i, \mathsf{ek}_i))$ for $i \in \{1,2\}$, where $\mathsf{ek}_i$ is the sequence of evaluation keys of the $i$-th $\mathsf{FHE}$ key.

2. Compute $C(\mathbf{x}, \mathbf{w})$. The computation trace of $C$ over $(\mathbf{x}, \mathbf{w})$ is defined to be the concatenation of all wire values, denoted by $\tau_C$. Compute $v_C = \mathsf{Hash}(\mathsf{hk}_{\mathsf{SSB}}, \tau_C)$.
3. For $i \in \{1, 2\}$: Treat $\mathsf{ct}_i$ as an encryption of a circuit representation, and compute $\mathsf{FHE.Eval}_{\mathsf{ek}_i}(U_\lambda, (\mathsf{ct}_i, \tau_C))$, where $U_\lambda$ is the aforementioned universal circuit. Since $\mathsf{FHE.Eval}$ proceeds in a gate-by-gate manner, we can also define the (partially encrypted) computation trace of $U_\lambda$ over $(\mathsf{ct}_i, \tau_C)$, denoted by $\tilde{\tau}_i$, as the concatenation of all wire values (some of which may be encrypted).
4. Compute $v_i = \mathsf{Hash}(\mathsf{hk}_{\mathsf{SSB}}, \tilde{\tau}_i)$ for $i \in \{1, 2\}$.
5. Compute several BARG proofs for each component $\gamma \in [\ell]$. Each of them checks openings under component $\gamma$ and the reference component 0, so its instance contains only these two components of the keys and hash values:
   (a) $\pi_C^{(\gamma)}$ proves: $v_C$ can open to $\tau_C$ that is a valid computation trace of $C$, and there exists a valid openings of $h_{\mathbf{x}}$ to each input wire value of $\mathbf{x}$, and the output wire value is 1. More formally, let $M_C$ be the following Turing machine.
      - **Instance.** The instance $z_{C,\gamma}$, consisting of components 0 and $\gamma$ of $\mathsf{hk}_{\mathsf{SSB}}$, $h_{\mathbf{x}}$, $h_C$, and $v_C$, and an index $i$.
      - **Witness.** The description of the $i$-th gate of $C$ with its openings of $h_C$, and the openings of $v_C$ to the $i$-th wire value and its input wire values (or an opening of $h_{\mathbf{x}}$), all under components 0 and $\gamma$.
      - **Computation.** Decide whether the openings are valid and locally consistent, and, if $i$ is the output wire, whether its value is 1.

      The prover runs $\mathsf{BARG.P}$ on $M_C$ under $\mathsf{crs}_{\mathsf{BARG},C}$ with the instance $z_{C,\gamma}$ and the appropriate witnesses for each wire to obtain $\pi_C^{(\gamma)}$.
   (b) For each $i \in \{1, 2\}$, $\pi_i^{(\gamma)}$ proves: $v_i$ can open to $\tilde{\tau}_i$ that is a valid (partially encrypted) computation trace, and there exists a valid opening of $h_i$ or $v_C$ to each input wire value of $\mathsf{ct}_i$ or $\tau_C$. More formally, let $M_i$ be the following Turing machine.
      - **Instance.** The instance $z_{i,\gamma}$, consisting of components 0 and $\gamma$ of $\mathsf{hk}_{\mathsf{SSB}}$, $h_i$, $v_C$, and $v_i$, and an index $j$.
      - **Witness.** The openings of $v_i$ to the $j$-th wire value (may be encrypted) and its input wire values (or an opening of $h_i$ or $v_C$), and an opening of $h_i$ to the evaluation key used by the $j$-th gate, all under components 0 and $\gamma$.
      - **Computation.** Decide whether the openings are valid and locally consistent.

      The prover runs $\mathsf{BARG.P}$ on $M_i$ under $\mathsf{crs}_{\mathsf{BARG},i}$ with the instance $z_{i,\gamma}$ and the appropriate witnesses for each wire to obtain $\pi_i^{(\gamma)}$.
6. Output $\Pi = (v_C, \{\pi_C^{(\gamma)}\}_{\gamma\in[\ell]}, \{v_i, \{\pi_i^{(\gamma)}\}_{\gamma\in[\ell]}\}_{i\in\{1,2\}})$.

- **Verifier:** Given $(\mathsf{crs}, C, \mathbf{x}, \Pi)$ where $\Pi = (v_C, \{\pi_C^{(\gamma)}\}_{\gamma\in[\ell]}, \{v_i, \{\pi_i^{(\gamma)}\}_{\gamma\in[\ell]}\}_{i\in\{1,2\}})$:
  1. Compute $h_{\mathbf{x}}$, $h_C$, and $h_i$ for $i \in \{1, 2\}$;
  2. For every $\gamma \in [\ell]$, run $\mathsf{BARG.V}$ under $\mathsf{crs}_{\mathsf{BARG},C}$ on $M_C$, $z_{C,\gamma}$, and $\pi_C^{(\gamma)}$, and check that the verification passes;

3. For every $\gamma \in [\ell]$ and $i \in \{1, 2\}$, run $\mathsf{BARG.V}$ under $\mathsf{crs}_{\mathsf{BARG},i}$ on $M_i$, $z_{i,\gamma}$, and $\pi_i^{(\gamma)}$, and check that the verification passes;
4. Output 1 if all checks pass. Otherwise output 0.

**Extraction.** Given the SNARG construction above $\mathsf{SNARG} = (\mathsf{SNARG.Setup}, \mathsf{SNARG}.\mathcal{P}, \mathsf{SNARG}.\mathcal{V})$, we define two additional algorithms $\mathsf{SNARG.TGen}$ and $\mathsf{SNARG.Ext}$ to extract a set of intermediate wire values from any SNARG proof. We can use $\mathsf{SSB.TGen}$ to generate a trapdoor $\mathsf{td}_{\mathsf{SSB}}$, such that bits of the witness $\mathbf{w}$ can be extracted from $v_C$. Furthermore, we can switch $\mathsf{ct}_1$ to encrypt a circuit at our choice, and use the FHE secret key $\mathsf{sk}_1$ along with $\mathsf{td}_{\mathsf{SSB}}$ to extract even more complex functions of $\mathbf{w}$.

We define $\mathsf{SNARG.TGen}$ and $\mathsf{SNARG.Ext}$ as follows:

1. $\mathsf{SNARG.TGen}(1^\lambda, D, S) \to (\mathsf{crs}, \mathsf{td})$. It's required that $|D|$ and $|S|$ are sufficiently small compared with $L(\lambda)$ and $\ell(\lambda)$, respectively.
2. $\mathsf{SNARG.Ext}(\mathsf{td}, \Pi)$ extracts the $i$-th wire value in $D(\mathbf{w})$ for all $i \in S$.

For simplicity, we will not give their constructions or their correctness guarantees in this section. Their full constructions are given in Section 7.

**Remark 6.1** (Difference from [JKLV24])**.** *We only slightly modify the SNARG construction of [JKLV24]: each trace is certified by $\ell$ $\mathsf{BARG}$ proofs, one for each component $\gamma$ of $\mathsf{hk}_{\mathsf{SSB}}$, whose instances contain only components $\gamma$ and 0. The main reason for this modification is that the locality of the $\mathcal{CEF}$ proof of soundness of our construction then does not grow with the maximum locality $\ell$ when the locality actually needed is smaller. This is due to a technical point in our $\mathcal{CEF}$ formalization: the locality of the $\mathcal{CEF}$ proof of $\mathsf{BARG}$ extraction (Theorem C.1) grows with the size of the $\mathsf{BARG}$ instance, and the proof of soundness (Theorem 6.8) chooses $\ell$ larger than the locality of a $\mathcal{CEF}$ proof that it obtains, so this locality must not grow with $\ell$.*

## 6.2 Main Theorems

We now present our main results that reduce the extractability and/or security of our SNARG construction to the existence of $\mathcal{CEF}$ proofs of witness uniqueness.

Throughout this section, we will use the following variables and notations.

- Let $\{\mathbf{x}_\lambda\}_\lambda$ be any family of bit strings of size $\mathsf{poly}(\lambda)$.
- Let $\{C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda)\}_\lambda$ be any family of circuits of size $\mathsf{poly}(\lambda)$ that take in $\mathbf{x}_\lambda, \mathbf{w}_\lambda$ as input, and outputs a bit.
- Let $\{\mathcal{A}_\lambda\}_\lambda$ be any family of circuits of size $\mathsf{poly}(\lambda)$.
- Let $L = L(\lambda), \ell = \ell(\lambda)$ be the two polynomials that parametrize our $\mathsf{SNARG}$ construction. When appearing in the same context with a $\mathcal{PEF}$ or $\mathcal{CEF}$ proof, we use $\ell_{\mathsf{SNARG}}$ and $\ell_{\mathcal{PEF}}, \ell_{\mathcal{CEF}}$ to distinguish different parameters.

To avoid lengthy theorem statements, we first define a useful circuit $\mathsf{LAG}$ that captures the SNARG verification and extraction process. The circuit has two outputs: the first is a bit indicating whether it succeeds, and the second is the extraction result. When verification passes, $\mathsf{LAG}$ is supposed to extract a locally consistent assignment for any circuit $D_\lambda$ that takes in as input the witness $\mathbf{w}_\lambda$ claimed by the prover. However, we defer the proof of correctness of $\mathsf{LAG}$ to Section 7.

**Definition 6.2** (Local Assignment Generator)**.** *Let* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *be a polynomial that only depends on* $\mathsf{UC}, \mathsf{FHE}, \mathsf{SSB}, \mathsf{BARG}$ *constructions.*

*Let* $\{D_\lambda\}_\lambda$ *be a family of circuits that take in* $\mathbf{w}_\lambda$ *as input. Let* $\{S_\lambda\}_\lambda$ *be a family of sets of wire indices in* $D_\lambda$*. Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |C_\lambda| + |D_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_\lambda|)$*.*

*Let* $\mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_\lambda, S_\lambda)$ *denote the following circuit:*

$\underline{\mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_\lambda, S_\lambda)}$

- *Let* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{SNARG.TGen}(1^\lambda, D_\lambda, S_\lambda)$*.*
- *Let* $\Pi \leftarrow \mathcal{A}_\lambda(\mathsf{crs})$*.*
- *Let* $\mathsf{rel} \leftarrow \mathsf{SNARG}.\mathcal{V}(\mathsf{crs}, C_\lambda, \mathbf{x}_\lambda, \Pi)$*.*
- *Let* $\mathbf{e} \leftarrow \mathsf{SNARG.Ext}(\mathsf{td}, \Pi)$*.*
- *Output* $(\mathsf{rel}, \mathbf{e})$*.*

*For simplicity, we overload the notation, and use* $\mathsf{LAG}_\lambda(\mathcal{A}_\lambda, i_\lambda)$ *to denote* $\mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D'_\lambda, \{i'_\lambda\})$*, where* $D'_\lambda$ *is a dummy circuit that takes in* $\mathbf{w}_\lambda$ *but does nothing, and* $i'_\lambda$ *is the index of the input wire corresponding to the* $i_\lambda$*-th bit of* $\mathbf{w}_\lambda$*.*

The lemma below captures the main property we need from the $\mathsf{LAG}$: If we can prove a statement for all valid witnesses, then the statement still holds if we extract it from a adversary. In particular, this still holds even if no valid witness exists. We defer its proof to Section 7.

**Lemma 6.3** (Core Lemma)**.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{y}_\lambda, \mathbf{w}_\lambda\}_\lambda$ *be any family of propositional variables,* $\{\mathbf{x}_\lambda\}_\lambda$ *be any family of Boolean strings, and* $\{D_{\mathsf{pre},\lambda}, D_{\mathsf{res},\lambda}\}_\lambda$ *be any family of two circuits that take in* $(\mathbf{x}_\lambda, \mathbf{y}_\lambda), (\mathbf{x}_\lambda, \mathbf{y}_\lambda, \mathbf{w}_\lambda)$ *as input, respectively. Let* $i^*_\lambda$ *denote the index of the output wire of* $D_{\mathsf{res},\lambda}$*.*

*Suppose we have a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof for*

$$\begin{aligned} &\textit{Let } \mathsf{pre} \coloneqq D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda);\\ &\quad \mathsf{rel} \coloneqq C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda);\\ &\quad \mathsf{res} \coloneqq D_{\mathsf{res},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda, \mathbf{w}_\lambda).\\ &\textit{Then } \mathsf{pre} \to (\mathsf{rel} \to \mathsf{res}). \end{aligned}$$

*Suppose* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + s(\lambda) + |C_\lambda| + |D_{\mathsf{pre},\lambda}| + |D_{\mathsf{res},\lambda}|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + \ell_{\mathcal{CEF}}(\lambda))$*.*

*Assuming polynomial hardness of LWE, there exists a* $(\mathsf{poly}(t+L), \mathsf{poly}(L), \mathsf{poly}(\ell_{\mathcal{CEF}}))$*-*$\mathcal{CEF}$ *proof for*

$$\begin{aligned} &\textit{Let } \mathsf{pre} \coloneqq D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda);\\ &\quad (\mathsf{rel}, \mathsf{res}) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\mathsf{res},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda, \cdot), \{i^*_\lambda\});\\ &\textit{Then } \mathsf{pre} \to (\mathsf{rel} \to \mathsf{res}). \end{aligned}$$

*where all consumed randomness are random variables.*

Applying Lemma 6.3 to a $\mathcal{CEF}$ proof of witness uniqueness, we get the following corollary.

**Corollary 6.4.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Suppose that there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof for* $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda) \wedge C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda) \to \mathbf{w}_\lambda = \mathbf{w}^*_\lambda$*, where* $\mathbf{w}_\lambda, \mathbf{w}^*_\lambda$ *are two proper variables, and* $\mathbf{x}_\lambda$ *is hardwired.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + s(\lambda) + |C_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + \ell_{\mathcal{CEF}}(\lambda))$*.*

*Assuming polynomial hardness of LWE, for any family of indices* $\{i_\lambda\}_\lambda$*, there exists a* $(\mathsf{poly}(t + L), \mathsf{poly}(L), \mathsf{poly}(\ell_{\mathcal{CEF}}))$*-*$\mathcal{CEF}$ *proof for*

$$\begin{aligned} &\textit{Let } \mathsf{pre} \coloneqq C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda);\\ &\quad (\mathsf{rel}, \mathsf{res}) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, i_\lambda);\\ &\textit{Then } \mathsf{pre} \to (\mathsf{rel} \to (\mathsf{res} = \mathbf{w}^*_\lambda[i_\lambda])). \end{aligned}$$

*where all consumed randomness are random variables.*

*Proof.* Let $D_{\mathsf{res},\lambda}$ be the circuit that takes in $\mathbf{x}_\lambda, \mathbf{w}^*_\lambda, \mathbf{w}_\lambda$ and outputs $[\mathbf{w}_\lambda = \mathbf{w}^*_\lambda]$, and $i^*_\lambda$ denotes the index of the output wire of $D_{\mathsf{res},\lambda}$. By assumption, there is a $(t, s, \ell_{\mathcal{CEF}})$-$\mathcal{CEF}$ proof for

$$\begin{aligned} \text{Let } &\mathsf{pre} \coloneqq C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda);\\ &\mathsf{rel} \coloneqq C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda);\\ &\mathsf{res} \coloneqq D_{\mathsf{res},\lambda}(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda, \mathbf{w}_\lambda);\\ \text{Then } &\mathsf{pre} \to (\mathsf{rel} \to \mathsf{res}). \end{aligned}$$

Applying Lemma 6.3 to this formula yields a $\mathcal{CEF}$ proof for

$$\begin{aligned} \text{Let } &\mathsf{pre} \coloneqq C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda);\\ &(\mathsf{rel}, \mathsf{res}) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\mathsf{res},\lambda}(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda, \cdot), \{i^*_\lambda\});\\ \text{Then } &\mathsf{pre} \to (\mathsf{rel} \to \mathsf{res}). \end{aligned}$$

Since the output wire is the AND of checking each bit of $\mathbf{w}_\lambda$ equals the corresponding bit of $\mathbf{w}^*_\lambda$, by Remark 7.10 and Lemma 7.13, we can directly extract $\mathbf{w}_\lambda[i_\lambda]$, and prove that it's equal to $\mathbf{w}^*_\lambda[i_\lambda]$. □

Corollary 6.4 directly implies a SNARK for true $\mathcal{EF}$-UP statements, by extracting the witness bit by bit.

**Theorem 6.5** (SNARKs for True $\mathcal{EF}$-UP Statements, Formal Theorem Statement)**.** *Assuming polynomial-hardness of LWE, for any language* $\mathcal{L} \in \mathcal{EF}$-UP*, there exists a SNARK for instances that are in the language* $\mathcal{L}$*, but it does not guarantee any soundness for instances that are not in* $\mathcal{L}$*.*

*Namely, the SNARK has CRS size* $L(\lambda)$ *and proof size* $\ell(\lambda)$*, where* $L(\cdot)$ *and* $\ell(\cdot)$ *are two polynomials that only depend on* $\mathcal{L}$*, and there exists a uniform extraction algorithm* $E$ *and a polynomial* $\mathsf{poly}$ *such that, for any sequence of instances* $\{x_\lambda\}_{\lambda\in\mathbb{N}}$ *that are in* $\mathcal{L}$ *and for any prover* $\{\mathcal{P}^*_\lambda\}_{\lambda\in\mathbb{N}}$ *that makes the verifier accept for* $\{x_\lambda\}_{\lambda\in\mathbb{N}}$ *with non-negligible probability* $\epsilon = \epsilon(\lambda)$*, there exists a negligible function* $\mathsf{negl} = \mathsf{negl}(\lambda)$ *and the black-box extractor* $E^{\mathcal{P}^*_\lambda}(x_\lambda)$ *outputs a witness for* $x_\lambda$ *in expected time* $\mathsf{poly}(\lambda, 1/(\epsilon(\lambda) - \mathsf{negl}(\lambda)))$ *for any sufficiently large* $\lambda$*.*

*Proof.* This theorem follows directly from Corollary 6.4 and the definition of $\mathcal{EF}$-UP.

By Corollary 6.4, if we set $L = L(\lambda)$ and $\ell = \ell(\lambda)$ to be sufficiently large polynomials that only depend on the relation circuit size and the $\mathcal{EF}$-UP proof size of $\mathcal{L}$, then the SNARG parameterized by $L, \ell$ has CRS size and proof size being fixed polynomials of $L$ and $\ell$, and admits a polynomial-size $\mathcal{CEF}$ proof for

$$\begin{aligned}\text{Let } \mathsf{pre} &:= C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda);\\ (\mathsf{rel}, \mathsf{res}) &:= \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, i_\lambda);\\ \text{Then } \mathsf{pre} &\to (\mathsf{rel} \to (\mathsf{res} = \mathbf{w}^*_\lambda[i_\lambda])).\end{aligned}$$

If we set $\mathbf{w}^*_\lambda$ to be the unique valid witness for $\mathbf{x}_\lambda$, then the above formula implies

$$\begin{aligned}\text{Let } (\mathsf{rel}, \mathsf{res}) &:= \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, i_\lambda);\\ \text{Then } \mathsf{rel} &\to (\mathsf{res} = \mathbf{w}^*_\lambda[i_\lambda]).\end{aligned}$$

The existence of a polynomial-size $\mathcal{CEF}$ proof implies the existence of polynomially many hybrid worlds between the real world and the ideal world, where the failing probabilities of any two consecutive hybrids differ negligibly. Thus, by the definition of $\mathsf{LAG}_\lambda$, if the adversary $\mathcal{A}_\lambda$ makes the SNARG verifier accept with probability $\epsilon(\lambda)$, then with probability $\epsilon(\lambda) - \mathsf{negl}(\lambda)$ the extraction algorithm extracts the $i_\lambda$-th bit of the unique valid witness $\mathbf{w}^*_\lambda$. By repeating this for all indices of $\mathbf{w}^*_\lambda$, we can extract the full witness in expected time $\mathsf{poly}(\lambda, 1/(\epsilon(\lambda) - \mathsf{negl}(\lambda)))$. □

Next, we formalize the intuition that, if $\mathsf{LAG}$ cannot extract a valid witness from $\mathcal{A}$, then $\mathcal{A}$ serves as a proof of non-membership. Since it's impossible to find a witness for a false statement, any $\mathcal{A}$ with non-negligible probability implies a proof of non-membership.

**Theorem 6.6.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Suppose that* $\forall \lambda, \forall \mathbf{w}_\lambda, C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda) = 0$.

*Suppose that there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof for* $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda) \wedge C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda) \to \mathbf{w}_\lambda = \mathbf{w}^*_\lambda$*, where* $\mathbf{w}_\lambda, \mathbf{w}^*_\lambda$ *are the two proper variables, and* $\mathbf{x}_\lambda$ *is hardwired.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + s(\lambda) + |C_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + \ell_{\mathcal{CEF}}(\lambda))$.

*Suppose there exists a constant* $c > 0$ *such that for all sufficiently large* $\lambda$, $|\mathcal{A}_\lambda| \leq \lambda^c$*, and*

$$\Pr\left[\mathsf{SNARG}.\mathcal{V}(\mathsf{crs}, C_\lambda, \mathbf{x}_\lambda, \Pi) = 1 \;\middle|\; \begin{array}{l}\mathsf{crs} \leftarrow \mathsf{SNARG.Setup}(1^\lambda);\\ \Pi \leftarrow \mathcal{A}_\lambda(\mathsf{crs}, \mathbf{x}_\lambda);\end{array}\right] \geq \lambda^{-c}.$$

*Assuming polynomial hardness of LWE, there exists a* $(\mathsf{poly}(t + L + \lambda^c), \mathsf{poly}(L + \lambda^c), \mathsf{poly}(\ell_{\mathcal{CEF}}))$*-*$\mathcal{CEF}$ *proof for*

$$C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda) = 0$$

*Proof.* Fix $c$ as in the theorem statement.

By Corollary 6.4, we have a poly size $\mathcal{CEF}$ proof for

$$\begin{aligned}\text{Let } \mathsf{pre} &:= C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda);\\ \forall 1 \leq i \leq |\mathbf{w}^*_\lambda|, (\mathsf{rel}^{(i)}, \mathbf{w}'[i]) &:= \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, i);\\ \text{Then } \forall 1 \leq i \leq |\mathbf{w}^*_\lambda|, \mathsf{pre} &\to (\mathsf{rel}^{(i)} \to \mathbf{w}'[i] = \mathbf{w}^*_\lambda[i]).\end{aligned}$$

where each $\mathsf{LAG}_\lambda(\mathcal{A}_\lambda, i)$ uses a fresh set of random variables, so they are all independent.

We can repeat the $\mathcal{CEF}$ proof for $\lambda^{c+1}$ times, obtaining $\mathsf{rel}^{(i,1)}, \ldots, \mathsf{rel}^{(i,\lambda^{c+1})}$ and $\mathbf{w}'^{(1)}[i], \ldots, \mathbf{w}'^{(\lambda^{c+1})}[i]$, then with high probability at least one of $\mathsf{rel}^{(i,1)}, \ldots, \mathsf{rel}^{(i,\lambda^{c+1})}$ is 1. Note that $\mathsf{rel}^{(i,j)}$ do not depend on any proper variable ($\mathbf{w}^*_\lambda$ is the only proper variable here), so we can apply the Random Inference rule and obtain $\forall 1 \le i \le |\mathbf{w}^*_\lambda|, \bigvee_{j=1}^{\lambda^{c+1}} \mathsf{rel}^{(i,j)} = 1$.

Let $\mathsf{SelectFirst}$ be the circuit that takes in $\mathsf{rel}^{(i,1)}, \ldots, \mathsf{rel}^{(i,\lambda^{c+1})}$ and $\mathbf{w}'^{(1)}[i], \ldots, \mathbf{w}'^{(\lambda^{c+1})}[i]$, and outputs $\mathbf{w}'^{(k)}[i]$ where $k$ is the smallest index such that $\mathsf{rel}^{(i,k)} = 1$. Thus we obtain a poly size $\mathcal{CEF}$ proof for

$$
\begin{aligned}
&\text{Let } \mathsf{pre} \coloneqq C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda);\\
&\quad \forall 1 \le i \le |\mathbf{w}^*_\lambda|, \forall 1 \le j \le \lambda^{c+1}, (\mathsf{rel}^{(i,j)}, \mathbf{w}'^{(j)}[i]) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, i);\\
&\quad \forall 1 \le i \le |\mathbf{w}^*_\lambda|, \mathbf{w}[i] \coloneqq \mathsf{SelectFirst}(\{\mathsf{rel}^{(i,j)}\}_{1\le j\le\lambda^{c+1}}, \{\mathbf{w}'^{(j)}[i]\}_{1\le j\le\lambda^{c+1}});\\
&\text{Then } \mathsf{pre} \to (\forall 1 \le i \le |\mathbf{w}^*_\lambda|, \mathbf{w}[i] = \mathbf{w}^*_\lambda[i]).
\end{aligned}
$$

Now, note that $\mathbf{w}[i]$ does not depend on any proper variable, and we also know $C_\lambda(\mathbf{x}_\lambda, \cdot)$ is unsatisfiable (outside the logic), thus we can use the Random Inference rule and add $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}) = 0$. But since $\mathsf{pre} \to (\forall 1 \le i \le |\mathbf{w}^*_\lambda|, \mathbf{w}[i] = \mathbf{w}^*_\lambda[i])$, we get $\mathsf{pre} \to C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda) = 0$, which implies $\mathsf{pre} = 0$.

Finally, we remove all other unused lines using the Erasure rule. □

Similar to [JKLV24], we prove that if there exists a $\mathcal{CEF}$ proof of non-membership, then the SNARG is sound.

**Lemma 6.7.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Suppose that there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof for* $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda) = 0$*, where* $\mathbf{w}_\lambda$*, is the only proper variable, and* $\mathbf{x}_\lambda$ *is hardwired.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \ge \mathsf{poly}(\lambda + s(\lambda) + |C_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \ge \mathsf{poly}(\lambda + \ell_{\mathcal{CEF}}(\lambda))$.

*Assuming polynomial hardness of LWE, there exists a* $(\mathsf{poly}(t+L), \mathsf{poly}(L), \mathsf{poly}(\ell_{\mathcal{CEF}}))$*-*$\mathcal{CEF}$ *proof for*

$$
\begin{aligned}
&\textit{Let } \mathsf{crs} \coloneqq \mathsf{SNARG.Setup}(1^\lambda);\\
&\quad \Pi \coloneqq \mathcal{A}_\lambda(\mathsf{crs});\\
&\quad \mathsf{rel} \coloneqq \mathsf{SNARG}.\mathcal{V}(\mathsf{crs}, C_\lambda, \mathbf{x}_\lambda, \Pi);\\
&\textit{Then } \mathsf{rel} = 0.
\end{aligned}
$$

*where all consumed randomness are random variables.*

*Proof.* By Lemma 6.3, we have a $(\mathsf{poly}(t+L), \mathsf{poly}(L), \mathsf{poly}(\ell_{\mathcal{CEF}}))$-$\mathcal{CEF}$ proof for

$$
\begin{aligned}
&\text{Let } (\mathsf{rel}, \mathsf{res}) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, C_\lambda(\mathbf{x}_\lambda, \cdot), \{i^*_\lambda\});\\
&\text{Then } \mathsf{rel} \to (\mathsf{res} = 0).
\end{aligned}
$$

where $i^*_\lambda$ is the index of the output wire of $C_\lambda(\mathbf{x}_\lambda, \cdot)$.

However, by Lemma 7.18, we can prove the opposite direction, i.e., $\mathsf{rel} \to \mathsf{res}$. Combining the two proves $\mathsf{rel} = 0$. Finally, we switch $\mathsf{crs}$ back to plain $\mathsf{SNARG.Setup}(1^\lambda)$. □

Finally, combining Theorem 6.6 and Lemma 6.7, we prove the soundness of our SNARG construction.

**Theorem 6.8.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Suppose that* $\forall \lambda, \forall \mathbf{w}_\lambda, C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda) = 0$.

*Suppose that there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof for* $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda) \wedge C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda) \to \mathbf{w}_\lambda = \mathbf{w}^*_\lambda$*, where* $\mathbf{w}_\lambda, \mathbf{w}^*_\lambda$ *are the two proper variables, and* $\mathbf{x}_\lambda$ *is hardwired.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + s(\lambda) + |C_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + \ell_{\mathcal{CEF}}(\lambda))$.

*Assuming polynomial hardness of LWE, there exists a constant* $c' > 0$*, such that for all constant* $c > 0$*, if we set* $L(\lambda) \geq \mathsf{poly}(\lambda + s(\lambda) + |C_\lambda|)^{c'}$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + \ell_{\mathcal{CEF}}(\lambda))$*, then for infinitely many* $\lambda$*, for any polynomial-size adversary* $\{\mathcal{A}_\lambda\}_\lambda$*, we have*

$$\Pr\left[\mathsf{SNARG}.\mathcal{V}(\mathsf{crs}, C_\lambda, \mathbf{x}_\lambda, \Pi) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{SNARG}.\mathsf{Setup}(1^\lambda); \\ \Pi \leftarrow \mathcal{A}_\lambda(\mathsf{crs}, \mathbf{x}_\lambda); \end{array}\right] \leq \lambda^{-c}.$$

*Proof.* We will first set $c' = 1$ and see what happens. If the desired result does not hold, then there exists a constant $c > 0$ and a polynomial-size adversary $\{\mathcal{A}_\lambda\}_\lambda$ such that for all sufficiently large $\lambda$, the above probability is at least $\lambda^{-c}$.

By Theorem 6.6, we have a $(\mathsf{poly}(t+L+\lambda^c), \mathsf{poly}(L+\lambda^c), \mathsf{poly}(\ell_{\mathcal{CEF}}))$-$\mathcal{CEF}$ proof for $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}^*_\lambda) = 0$, where $L(\lambda) = \mathsf{poly}(\lambda + s(\lambda) + |C_\lambda|)$.

By Lemma 6.7, if we set $L'(\lambda) \geq \mathsf{poly}(L+\lambda^c)$ and $\ell'_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + \mathsf{poly}(\ell_{\mathcal{CEF}}(\lambda)))$, then there is a polynomial-size $\mathcal{CEF}$ proof for SNARG soundness for any polynomial-size adversary. Therefore, either the desired result holds for infinitely many $\lambda$ when $c' = 1$, or the desired result holds for all sufficiently large $\lambda$ when $c'$ is sufficiently large. □

# 7 Proof of Lemma 6.3

In this section, we prove useful properties of our SNARG construction and formalize the algorithms $\mathsf{SNARG.TGen}$ and $\mathsf{SNARG.Ext}$, culminating in a proof of Lemma 6.3.

## 7.1 Formalizing Existing Proofs in $\mathcal{CEF}$

The work of [JKLV24] has already proved many useful properties of the Encrypt-Hash-and-BARG framework, e.g., locally consistent extraction, no-signaling extraction, etc. However, we still need to go over them again due to two reasons: 1) we have a slightly different SNARG construction; and 2) we need to formalize these proofs in $\mathcal{CEF}$, which is less expressive than ZFC. Therefore, the proofs in this section look familiar to those of [JKLV24], yet are slightly different.

Throughout this section, we will use the following variables and notations.

- Let $\{\mathbf{x}_\lambda\}_\lambda$ be any family of bit strings of size $\mathsf{poly}(\lambda)$.
- Let $\{C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda)\}_\lambda$ be any family of circuits of size $\mathsf{poly}(\lambda)$ that take in $\mathbf{x}_\lambda, \mathbf{w}_\lambda$ as input, and outputs a bit. Let $\tau_{C_\lambda}$ denote a (supposely valid) computation trace of $C_\lambda$ on some $(\mathbf{x}_\lambda, \mathbf{w}_\lambda)$.
- Let $\{\mathcal{A}_\lambda\}_\lambda$ be any family of circuits of size $\mathsf{poly}(\lambda)$.
- Let $L(\lambda), \ell(\lambda)$ be the two polynomials that parametrize our $\mathsf{SNARG}$ construction.
- Let $U_\lambda$ be the universal circuit in the $\mathsf{SNARG}$ construction.

We start by extracting wire values from the computation $\mathsf{Eval}(\mathsf{ek}_i, U, (\mathsf{ct}_i, w))$.

**Definition 7.1** (Local Assignment Generator for $U$)**.** *Let $\{\mathbf{a}_\lambda\}_\lambda$ be a family of bit strings of length at most $L(\lambda)$, corresponding to the circuit representation input to $U_\lambda$; Let $\{S_\lambda\}_\lambda$ be a family of sets of wire indices in $U_\lambda$ of size at most $\ell_{\mathsf{SNARG}}(\lambda)$.*

*Let $\mathsf{LAG}^U_\lambda(\mathcal{A}_\lambda, \mathbf{a}_\lambda, S_\lambda)$ denote the circuit in Figure 3.*

$\underline{\mathsf{LAG}^U_\lambda(\mathcal{A}_\lambda, \mathbf{a}_\lambda, S_\lambda)}$

- Let $(\mathsf{pk}_1, \mathsf{ek}_1, \mathsf{sk}_1) \leftarrow \mathsf{FHE.Setup}(1^\lambda)$;
- Let $\mathsf{ct}_1 \leftarrow \mathsf{FHE.Enc}_{\mathsf{pk}_1}(\mathbf{a}_\lambda)$;
- Generate an SSB hash key $\mathsf{hk}_{\mathsf{SSB}}$ for a vector of $L(\lambda)$ FHE ciphertexts, supporting extraction at indices in $S_\lambda$: components $1, \ldots, |S_\lambda|$ extract these indices, one each, and the other components, including the reference component 0, are generated by $\mathsf{SSB.Gen}$.
- Let the rest of $\mathsf{crs}$ be generated in the same way as in $\mathsf{SNARG.Setup}(1^\lambda)$;
- Let $\Pi \leftarrow \mathcal{A}_\lambda(\mathsf{crs})$;
- Let $\mathsf{rel} \leftarrow \mathsf{SNARG}.\mathcal{V}(\mathsf{crs}, C_\lambda, \mathbf{x}_\lambda, \Pi)$;
- Let $\mathbf{u}' \leftarrow \mathsf{SSB.Ext}(\mathsf{td}_{\mathsf{SSB}}, \Pi.v_1)$;
- Let $\mathbf{u} \leftarrow \mathsf{FHE.Dec}_{\mathsf{sk}_1}(\mathbf{u}')$ (only decrypting the places that are supposed to be ciphertexts);
- Output $(\mathsf{rel}, \mathbf{u})$.

Figure 3: The local assignment generator $\mathsf{LAG}^U_\lambda(\mathcal{A}_\lambda, \mathbf{a}_\lambda, S_\lambda)$ for $U$.

In the security proof, we compare openings from different $\mathsf{BARG}$ proofs through the reference component $\mathsf{hk}^{(0)}_{\mathsf{SSB}}$, which we switch to binding mode temporarily.

**Lemma 7.2** (Computational Binding for $\mathsf{hk}^{(0)}_{\mathsf{SSB}}$)**.** *Let $\Phi$ be a configuration that defines $\mathsf{hk}^{(0)}_{\mathsf{SSB}} \coloneqq \mathsf{SSB.Gen}(1^\lambda, 1^N; \mathbf{r}^{(0)})$, where the random variables $\mathbf{r}^{(0)}$ appear nowhere else in $\Phi$. Let $p \in [N]$ be a fixed position, and let $h$, $(v, \sigma)$, and $(v', \sigma')$ be variables of $\Phi$. Assuming polynomial hardness of LWE, there exists a $\mathcal{CEF}$ proof of length $\mathsf{poly}(\lambda + |\Phi|)$ and locality $\mathsf{poly}(\lambda)$ from $\Phi$ to $\Phi \cup \{\phi\}$, where*

$$\phi \coloneqq \Big[\big(\mathsf{SSB.Verify}(\mathsf{hk}^{(0)}_{\mathsf{SSB}}, h, v, p, \sigma) = 1 \wedge \mathsf{SSB.Verify}(\mathsf{hk}^{(0)}_{\mathsf{SSB}}, h, v', p, \sigma') = 1\big) \to v = v'\Big].$$

*Moreover, if $\Phi$ defines $h \coloneqq \mathsf{SSB.Hash}(\mathsf{hk}^{(0)}_{\mathsf{SSB}}, X)$ for a vector $X$ whose entries are variables of $\Phi$ or constants, then the same holds for*

$$\phi_X \coloneqq \big[\mathsf{SSB.Verify}(\mathsf{hk}^{(0)}_{\mathsf{SSB}}, h, v, p, \sigma) = 1 \to v = X[p]\big].$$

*Proof.* By the Distribution Shift rule, we switch $\mathsf{hk}^{(0)}_{\mathsf{SSB}}$ to binding mode at $p$, since the two modes are indistinguishable under LWE. Now $\mathsf{hk}^{(0)}_{\mathsf{SSB}}$ is generated as follows:

$$(\mathsf{hk}^{(0)}_{\mathsf{SSB}}, \mathsf{td}^{(0)}) \coloneqq \mathsf{SSB.TGen}(1^\lambda, 1^N, p; \mathbf{r}').$$

The $\mathcal{EF}$ proof of extraction correctness of $\mathsf{SSB}$ (Theorem A.1), applied to both openings, shows that each of $v$ and $v'$ equals $\mathsf{SSB.Ext}(h, \mathsf{td}^{(0)})$ if its opening verifies, which gives $\phi$.

We then erase the formulas that involve $\mathsf{td}^{(0)}$, and switch the definition of $\mathsf{hk}_{\mathsf{SSB}}^{(0)}$ back by the Distribution Shift rule. The formula $\phi$ remains, since it does not involve $\mathsf{td}^{(0)}$.

For the second part, we add the honest opening $\sigma' \coloneqq \mathsf{SSB.Open}(\mathsf{hk}_{\mathsf{SSB}}^{(0)}, X, p)$ as an extension variable. Since $\mathsf{SSB.Verify}$ recomputes $h$ from $X[p]$ and $\sigma'$, and $\sigma'$ consists of intermediate values in the computation of $h$, the formula $\mathsf{SSB.Verify}(\mathsf{hk}_{\mathsf{SSB}}^{(0)}, h, X[p], p, \sigma') = 1$ has an $\mathcal{EF}$ proof by unfolding the definitions. The first part, applied with $(v', \sigma') = (X[p], \sigma')$, then gives $\phi_X$, and we erase $\sigma'$. $\square$

We first prove that the extraction result is locally consistent.

**Lemma 7.3.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{a}_\lambda\}_\lambda$ *be a family of vectors of proper variables. Let* $\{S_\lambda\}_\lambda$ *be a family of sets of wire indices in* $U_\lambda$.

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq |\mathbf{a}_\lambda|$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_\lambda|)$.

*Let* $t = s = \mathsf{poly}(\lambda + L + |C_\lambda| + |\mathcal{A}_\lambda|)$, *and* $\ell_{\mathcal{CEF}} = \mathsf{poly}(\lambda + |S_\lambda|)$. *Assuming polynomial hardness of LWE, there exists a* $(t, s, \ell_{\mathcal{CEF}})$-$\mathcal{CEF}$ *proof for*

$$\begin{gathered}\textit{Let } (\mathsf{rel}, \mathbf{u}) \coloneqq \mathsf{LAG}_\lambda^U(\mathcal{A}_\lambda, \mathbf{a}_\lambda, S_\lambda);\\ \textit{Then } \mathsf{rel} \to (\mathbf{u} \textit{ is consistent with } U_\lambda(\mathbf{a}_\lambda, \cdot)).\end{gathered}$$

*where all consumed randomness are random variables. Here* $\mathbf{x}_\lambda$, $C_\lambda$, $\mathcal{A}_\lambda$, *and* $S_\lambda$ *are fixed for each* $\lambda$ *and hardwired into the formula, and* $\mathbf{a}_\lambda$ *are its only proper variables.*

*"Consistent" means that:*

1. *For each gate, if its input and output wires are all extracted, then the gate relation holds.*
2. *If an input wire that reads from* $\mathsf{ct}_1$ *is extracted, then its value matches* $\mathbf{a}_\lambda$.

*We will use this definition of consistency in other lemmas as well.*

*Proof.* For any wire $i \in S_\lambda$, we will first prove the consistency condition that involves it, and then combine the results for all $i \in S_\lambda$. A consistency condition has two cases:

- Either the $i$-th wire is the output wire of a gate $f$ whose input wires $j$ and $j'$ are also in $S_\lambda$, and we need to prove $\mathbf{u}[i] = f(\mathbf{u}[j], \mathbf{u}[j'])$; or
- the $i$-th wire is an input wire that reads the $p$-th entry of $\mathsf{ct}_1$, and we need to prove $\mathbf{u}[i] = \mathbf{a}_\lambda[p]$.

We only prove the first case. The second case is similar and simpler. For each extracted wire $k$, let $\gamma_k$ be the component of $\mathsf{hk}_{\mathsf{SSB}}$ that extracts it, and let $\Gamma_i = \{\gamma_i, \gamma_j, \gamma_{j'}\}$.

The idea is to extract from the $\mathsf{BARG}$ proof $\Pi.\pi_1^{(\gamma_i)}$ its witness at index $i$ (Theorem C.1), which shows that the values it opens at the $i$-th wire and its input wires satisfy the gate. However, this witness is tied to the extraction result only at the wire that component $\gamma_i$ extracts. We therefore extract witnesses from the BARG proofs of all components in $\Gamma_i$, and use the "reference component" 0 to show that they open the same values.

**Extracting the BARG witnesses.** Theorem C.1 extracts a witness only when the BARG CRS is generated with the extraction index, while SNARG.Setup generates $\mathsf{crs}_{\mathsf{BARG},1}$ with index 1. By the index hiding property of BARG (Definition 4.9), we use the Distribution Shift rule to generate it instead as

$$(\mathsf{crs}_{\mathsf{BARG},1}, \mathsf{td}_{\mathsf{BARG}}) \coloneqq \mathsf{BARG.Gen}(1^\lambda, 1^n, 1^m, 1^T, |U_\lambda|, i; \mathbf{r}),$$

where $n$, $m$, and $T$ are the instance length, witness length, and running time of $M_1$, and $|U_\lambda|$ is the number of instances. For each $\gamma \in \Gamma_i$, the proof $\Pi.\pi_1^{(\gamma)}$ is for the instance $x_{1,\gamma} = (M_1, z_{1,\gamma}, |U_\lambda|, T)$, and we add the extension variable

$$\mathbf{s}^{(\gamma)} \coloneqq \mathsf{BARG.Ext}(\mathsf{td}_{\mathsf{BARG}}, x_{1,\gamma}, \Pi.\pi_1^{(\gamma)}),$$

the witness extracted at index $i$. Since the SNARG verifier checks every $\Pi.\pi_1^{(\gamma)}$, unfolding its definition gives $\mathsf{rel} \to \mathsf{BARG.V}(\mathsf{crs}_{\mathsf{BARG},1}, x_{1,\gamma}, \Pi.\pi_1^{(\gamma)}) = 1$. Hence Theorem C.1, applied once for each $\gamma \in \Gamma_i$, gives

$$\mathsf{rel} \to M_1(z_{1,\gamma}, i, \mathbf{s}^{(\gamma)}) = 1 \quad \text{for every } \gamma \in \Gamma_i.$$

We check at the end of the proof that Theorem C.1 applies here with locality $\mathsf{poly}(\lambda)$.

By the definition of $M_1$, the formula $M_1(z_{1,\gamma}, i, \mathbf{s}^{(\gamma)}) = 1$ says that $\mathbf{s}^{(\gamma)}$ contains values $c_i^{(\gamma)}$, $c_j^{(\gamma)}$, and $c_{j'}^{(\gamma)}$ of the $i$-th, $j$-th, and $j'$-th wires and an evaluation key $\mathsf{ek}^{(\gamma)}$ such that:

1. for each $k \in \{i, j, j'\}$, the value $c_k^{(\gamma)}$ has valid openings at position $k$ of both $\Pi.v_1^{(\gamma)}$ and $\Pi.v_1^{(0)}$;
2. the key $\mathsf{ek}^{(\gamma)}$ has valid openings of both $h_1^{(\gamma)}$ and $h_1^{(0)}$ at the position of the evaluation key used by the gate $f$;
3. $c_i^{(\gamma)} = \mathsf{FHE.GateEval}_{\mathsf{ek}^{(\gamma)}}(f, c_j^{(\gamma)}, c_{j'}^{(\gamma)})$.

**Identifying the opened values.** For each $k \in \{i, j, j'\}$, let $\mathbf{u}'[k]$ be the value of the $k$-th wire that component $\gamma_k$ extracts from $\Pi.v_1$, before decryption, as in Figure 3. We show that the witness $\mathbf{s}^{(\gamma_i)}$ contains these values and the true evaluation key, using three facts.

First, for each $k \in \{i, j, j'\}$, the value $c_k^{(\gamma_k)}$ has a valid opening of $\Pi.v_1^{(\gamma_k)}$ at position $k$, and component $\gamma_k$ is generated in extraction mode at position $k$. By the extraction correctness of SSB (Theorem A.1), $c_k^{(\gamma_k)} = \mathbf{u}'[k]$.

Second, for each $k \in \{i, j, j'\}$, the values $c_k^{(\gamma)}$ for $\gamma \in \Gamma_i$ all have valid openings of the same hash value $\Pi.v_1^{(0)}$ at position $k$. Component 0 is generated by SSB.Gen with randomness used nowhere else, so Lemma 7.2 applies at position $k$ and shows that these values are equal. In particular, $c_k^{(\gamma_i)} = c_k^{(\gamma_k)} = \mathbf{u}'[k]$.

Third, the verifier computes $h_1^{(0)} = \mathsf{SSB.Hash}(\mathsf{hk}_{\mathsf{SSB}}^{(0)}, (\mathsf{ct}_1, \mathsf{ek}_1))$ honestly from the CRS. Since $\mathsf{ek}^{(\gamma_i)}$ has a valid opening of $h_1^{(0)}$ at the position of the evaluation key used by $f$, the second part of Lemma 7.2 shows that $\mathsf{ek}^{(\gamma_i)}$ equals this entry of $\mathsf{ek}_1$, which we denote by $\mathsf{ek}$.

Substituting these equalities into the third condition for $\gamma = \gamma_i$, we get

$$\mathsf{rel} \to \mathbf{u}'[i] = \mathsf{FHE.GateEval}_{\mathsf{ek}}(f, \mathbf{u}'[j], \mathbf{u}'[j']).$$

**Decrypting.** The output $\mathbf{u}$ of $\mathsf{LAG}^U_\lambda$ decrypts each $\mathbf{u}'[k]$ under the secret key in $\mathsf{sk}_1$ for the level of the $k$-th wire. The malicious gate correctness of $\mathsf{FHE}$ (Lemma 4.7) holds for arbitrary ciphertexts, so decrypting both sides gives $\mathsf{rel} \to \mathbf{u}[i] = f(\mathbf{u}[j], \mathbf{u}[j'])$. If the wires of $f$ are not encrypted, then $M_1$ checks $f$ on their values directly, and this formula follows without decryption.

**Undoing the switch.** The formula $\mathsf{rel} \to \mathbf{u}[i] = f(\mathbf{u}[j], \mathbf{u}[j'])$ involves neither $\mathsf{td}_{\mathsf{BARG}}$ nor $\mathbf{s}^{(\gamma)}$. We erase all formulas that involve them, and then switch $\mathsf{crs}_{\mathsf{BARG},1}$ back to index 1 by the Distribution Shift rule. This gives a $\mathcal{CEF}$ proof of the condition at the $i$-th wire for $\mathsf{LAG}^U_\lambda$ itself. We do this for every $i \in S_\lambda$ in turn. Together, the resulting formulas say that $\mathbf{u}$ is consistent with $U_\lambda(\mathbf{a}_\lambda, \cdot)$, which proves the lemma. This multiplies the length of the proof by $|S_\lambda|$ and does not change its locality.

**Bounding the locality.** Only Lemma 7.2 and Theorem C.1 constrain the locality, since the other steps are $\mathcal{EF}$ steps, Distribution Shift steps, and erasures. Lemma 7.2 has locality $\mathsf{poly}(\lambda)$. We apply Theorem C.1 with the circuit $M_1$, the index $i$, and the parameters $n$, $m$, $T$, and $|U_\lambda|$, all of which are fixed for each $\lambda$. Its proper variables are the part of the $\mathsf{SNARG}$ CRS other than $\mathsf{crs}_{\mathsf{BARG},1}$, which includes $\mathsf{ct}_1$ and $\mathsf{hk}_{\mathsf{SSB}}$, and its adversary runs $\mathcal{A}_\lambda$ on the full CRS and outputs $(z_{1,\gamma}, \Pi.\pi_1^{(\gamma)})$. By Lemma 5.7, we then replace these proper variables with their definitions in $\mathsf{LAG}^U_\lambda$, which is allowed since their randomness is disjoint from that of $\mathsf{crs}_{\mathsf{BARG},1}$.

The locality given by Theorem C.1 is $\ell(|M_1|, n, m, \log|U_\lambda|, \lambda)$. The instance $z_{1,\gamma}$ consists of components 0 and $\gamma$ of $\mathsf{hk}_{\mathsf{SSB}}$, $h_1$, $v_C$, and $v_1$, and a witness consists of $O(1)$ values and keys with their openings, so $n$ and $m$ are $\mathsf{poly}(\lambda)$. Since $M_1$ only verifies $O(1)$ openings under two components and evaluates one $\mathsf{FHE}$ gate, $|M_1|$ is also $\mathsf{poly}(\lambda)$, even when $M_1$ is viewed as a circuit. Finally, $\log|U_\lambda| \le \lambda$. These polynomials do not depend on $L$, $\ell_{\mathsf{SNARG}}$, $C_\lambda$, $\mathbf{x}_\lambda$, or $\mathcal{A}_\lambda$, so the locality is $\mathsf{poly}(\lambda)$. □

We also prove that we can switch from one extraction set $S$ to another extraction set $S'$, and any property of $S \cap S'$ will be preserved.

**Lemma 7.4.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{a}_\lambda\}_\lambda$ *be a family of vectors of proper variables. Let* $\{S_{\lambda,1}, S_{\lambda,2}\}_\lambda$ *be two families of sets of wire indices in* $U_\lambda$*. Let* $\{P_\lambda\}_\lambda$ *be a family of predicates that takes in* $\mathbf{a}_\lambda, \mathsf{crs}, \Pi$ *and the wire values in* $S_{\lambda,1} \cap S_{\lambda,2}$*.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \ge |\mathbf{a}_\lambda|$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \ge \mathsf{poly}(\lambda + |S_{\lambda,1}| + |S_{\lambda,2}|)$*.*

*Let* $t = s = \mathsf{poly}(\lambda + L + |C_\lambda| + |\mathcal{A}_\lambda| + |P_\lambda|)$*, and* $\ell_{\mathcal{CEF}} = \mathsf{poly}(\lambda + |S_{\lambda,1}| + |S_{\lambda,2}|)$*. Assuming polynomial hardness of LWE, there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof from*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, \mathbf{u}) \coloneqq \mathsf{LAG}^U_\lambda(\mathcal{A}_\lambda, \mathbf{a}_\lambda, S_{\lambda,1});\\ &\textit{Then } \mathsf{rel} \to P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{u}[S_{\lambda,1} \cap S_{\lambda,2}]). \end{aligned}$$

*to*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, \mathbf{u}) \coloneqq \mathsf{LAG}^U_\lambda(\mathcal{A}_\lambda, \mathbf{a}_\lambda, S_{\lambda,2});\\ &\textit{Then } \mathsf{rel} \to P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{u}[S_{\lambda,1} \cap S_{\lambda,2}]). \end{aligned}$$

*where all consumed randomness are random variables, and* $\mathsf{crs}, \Pi$ *are generated inside* $\mathsf{LAG}^U_\lambda$*.*

*Proof.* Here we make non-black-box use of our $\mathsf{SSB}$ construction. Specifically, we use the fact that given a set of indices, the $\mathsf{SSB}$ construction actually generates an independent trapdoor for each index, and they are also independently used during extraction. Therefore, we can remove or add indices and trapdoors without affecting the rest of the construction.

The proof goes in three steps:

1. Use the Erasure rule to remove $\mathbf{u}[S_{\lambda,1} \setminus S_{\lambda,2}]$. Use the Distribution Shift rule to switch $\mathsf{SSB.TGen}$ to $\mathsf{SSB.Gen}$ for those indices.

2. For the remaining indices $S_{\lambda,1} \cap S_{\lambda,2}$, reorder and reposition them so that they are in the same order as in $S_{\lambda,2}$.

3. Use the Distribution Shift rule to switch $\mathsf{SSB.Gen}$ to $\mathsf{SSB.TGen}$ for the indices in $S_{\lambda,2} \setminus S_{\lambda,1}$. Use the Extension rule to add extraction results for those indices.

The first and third step are straightforward applications of the rules.

The second step is a bit more subtle. The $\mathsf{SSB}$ construction with extraction locality $\ell$ generates $\ell$ independent $\mathsf{crs}$, namely components $1, \ldots, \ell$ of $\mathsf{hk}_{\mathsf{SSB}}$, each with or without a trapdoor, and uses them independently during extraction. Say, for example, the $i$-th $\mathsf{crs}$ is equipped with a trapdoor that allows extraction of the $s$-th wire value, and the $j$-th $\mathsf{crs}$ does not have a trapdoor. Now we want $j$-th $\mathsf{crs}$ to extract the $s$-th wire value instead. We cannot directly switch their positions, since the $i$-th trapdoor is currently in use, breaking any indistinguishability guarantee of $\mathsf{SSB}$.

Instead, we will first make a copy in $j$, and then remove the $i$-th trapdoor. Specifically, we will create a new trapdoor for the $j$-th $\mathsf{crs}$ that allows it to extract the $s$-th wire value, prove that extracting from $j$ is the same as extracting from $i$, and then remove the $i$-th trapdoor.

The creating and removing of trapdoors can be done using the Distribution Shift rule. The consistency proof is argued as in the proof of Lemma 7.3: we apply Theorem C.1 to the $\mathsf{BARG}$ proofs $\Pi.\pi_1^{(i)}$ and $\Pi.\pi_1^{(j)}$ at index $s$, Lemma 7.2 shows that their witnesses open the same value of the $s$-th wire, and $\mathsf{SSB}$ binding identifies this value with the extraction results of the $i$-th and $j$-th components. □

Next, we encrypt a circuit $E$ inside $\mathsf{ct}_1$, and extract wire values of $E$.

**Definition 7.5** (Local Assignment Generator for $E$)**.** *Let* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *be a polynomial that only depends on* $\mathsf{UC}, \mathsf{FHE}, \mathsf{SSB}, \mathsf{BARG}$ *constructions.*

*Let* $\{E_\lambda\}_\lambda$ *be a family of circuits that take in* $\tau_{C_\lambda}$ *as input. Let* $\{S_\lambda\}_\lambda$ *be a family of sets of wire indices in* $E_\lambda$*. Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |C_\lambda| + |E_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_\lambda|)$.

*Let* $\mathsf{LAG}_\lambda^E(\mathcal{A}_\lambda, E_\lambda, S_\lambda)$ *denote the following circuit:*

- *Let* $(\mathsf{rel}, \mathbf{u}) \leftarrow \mathsf{LAG}_\lambda^U(\mathcal{A}_\lambda, E_\lambda, \mathsf{UC.Map}(U_\lambda, E_\lambda, S_\lambda))$*;*
- *Let* $\mathbf{e} \leftarrow \mathsf{UC.Proc}(U_\lambda, E_\lambda, \mathbf{u})$*;*
- *Output* $(\mathsf{rel}, \mathbf{e})$.

**Lemma 7.6.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{a}_\lambda\}_\lambda$ *be a family of vectors of proper variables. Let* $\{E_\lambda\}_\lambda$ *be a family of circuits that take in* $\mathbf{a}_\lambda, \tau_{C_\lambda}$ *as input. Let* $\{S_\lambda\}_\lambda$ *be a family of sets of wire indices in* $E_\lambda$.

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq |E_\lambda|$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_\lambda|)$.

*Let* $t = s = \mathsf{poly}(\lambda + L + |C_\lambda| + |\mathcal{A}_\lambda|)$, *and* $\ell_{\mathcal{CEF}} = \mathsf{poly}(\lambda + |S_\lambda|)$. *Assuming polynomial hardness of LWE, there exists a* $(t, s, \ell_{\mathcal{CEF}})$-$\mathcal{CEF}$ *proof for*

$$\begin{gathered}\textit{Let } (\mathsf{rel}, \mathbf{e}) \coloneqq \mathsf{LAG}^E_\lambda(\mathcal{A}_\lambda, E_\lambda(\mathbf{a}_\lambda, \cdot), S_\lambda);\\ \textit{Then } \mathsf{rel} \to (\mathbf{e} \textit{ is consistent with } E_\lambda(\mathbf{a}_\lambda, \cdot)).\end{gathered}$$

*where all consumed randomness are random variables.*

*Proof.* Using Lemma 7.3, and using Lemma 5.7 to replace $\mathbf{a}_\lambda$ with $E_\lambda(\mathbf{a}_\lambda, \cdot)$, we directly prove that $\mathbf{u}$ contains consistent wire values in $U(E_\lambda(\mathbf{a}_\lambda, \cdot), \cdot)$ if $\mathsf{rel} = 1$.

Then, we use the $\mathcal{EF}$ proof of $\mathsf{UC}$ mapping correctness to prove that $\mathbf{e}$ contains consistent wire values in $E_\lambda(\mathbf{a}_\lambda, \cdot)$. □

**Remark 7.7.** *Similar to Lemma 7.4, we can also switch extraction sets in* $\mathsf{LAG}^E_\lambda$ *while preserving properties on their intersection, using Lemma 7.4 and the monotonicity of* $\mathsf{UC.Map}$.

Next, we define the $\mathsf{SSB}$ *extension* of a circuit.

**Definition 7.8** ($\mathsf{SSB}$ Extension)**.** *For any* $\lambda$, *for any circuit* $D$, *let* $E_\lambda(D)$ *be a circuit that*

- *Takes in the same input as* $D$, *and computes and outputs* $D$;
- *Let* $\tau$ *be the concatenation of the value of each wire in* $D$;
- *Takes in* $\mathsf{hk}'_{\mathsf{SSB}}$ *as another input, and compute a Merkle tree on* $\tau$, *using* $\mathsf{SSB.Hash}$. *Note that this* $\mathsf{hk}'_{\mathsf{SSB}}$ *is different from the* $\mathsf{hk}_{\mathsf{SSB}}$ *in the SNARG construction, and only appears in the security proof; We make this distinction to avoid confusion and to avoid technical issues.*

*Note that the first* $|D|$ *wires of* $E_\lambda(D)$ *are exactly the same as* $D$.

Using $\mathsf{SSB}$ extension, we now define our final form of Local Assignment Generator.

**Definition 7.9** (Local Assignment Generator for $D$)**.** *Let* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *be a polynomial that only depends on* $\mathsf{UC}, \mathsf{FHE}, \mathsf{SSB}, \mathsf{BARG}$ *constructions.*

*Let* $\{D_\lambda\}_\lambda$ *be a family of circuits that take in* $\tau_{C_\lambda}$ *as input. Let* $\{S_\lambda\}_\lambda$ *be a family of sets of wire indices in* $D_\lambda$. *Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |C_\lambda| + |D_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_\lambda|)$.

*Let* $\mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_\lambda, S_\lambda)$ *denote the following circuit:*

- *Generate an SSB hash key* $\mathsf{hk}'_{\mathsf{SSB}}$ *for a vector of* $L(\lambda)$ *FHE ciphertexts, supporting extraction of* $\ell(\lambda)$ *indices;*
- *Let* $(\mathsf{rel}, \mathbf{e}) \leftarrow \mathsf{LAG}^E_\lambda(\mathcal{A}_\lambda, E_\lambda(D_\lambda)(\cdot, \mathsf{hk}'_{\mathsf{SSB}}), S_\lambda)$;
- *Output* $(\mathsf{rel}, \mathbf{e})$.

**Remark 7.10.** *Similar to Lemma 7.6, we can also prove extraction consistency for* $\mathsf{LAG}^D_\lambda$.

**Remark 7.11.** *Similar to Remark 7.7, we can also switch extraction sets in* $\mathsf{LAG}^D_\lambda$ *while preserving properties on their intersection.*

**Remark 7.12.** *The* $\mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_\lambda, S_\lambda)$ *in Definition 6.2 is almost the same as* $\mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_\lambda, S_\lambda)$ *defined here, except that* $D_\lambda$ *needs to take in extra bits in* $\tau_{C_\lambda}$ *and ignore them. For* $\mathsf{LAG}_\lambda(\mathcal{A}_\lambda, S_\lambda)$*, we let* $D'_\lambda$ *be the circuit that takes* $\tau_{C_\lambda}$ *as input but does nothing, and* $S'_\lambda$ *corresponds to the input wires of* $\mathbf{w}_\lambda[S_\lambda]$ *in* $\tau_{C_\lambda}$*, and run* $\mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D'_\lambda, S'_\lambda)$*. This also implicitly implies our construction for* $\mathsf{SNARG.TGen}$ *and* $\mathsf{SNARG.Ext}$*.*

We prove cross-circuit consistency of extraction using $\mathsf{LAG}^D_\lambda$.

**Lemma 7.13.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{a}_\lambda\}_\lambda$ *be a family of vectors of proper variables. Let* $\{D_{\lambda,1}, D_{\lambda,2}\}_\lambda$ *be two families of circuits that take in* $\mathbf{a}_\lambda, \tau_{C_\lambda}$ *as input. Let* $\{S_{\lambda,1}, S_{\lambda,2}\}_\lambda$ *be two families of sets of wire indices in* $D_{\lambda,1}, D_{\lambda,2}$ *respectively. Furthermore, suppose that the dependency circuit of* $S_{\lambda,1}$ *in* $D_{\lambda,1}$ *is "isomorphic" to that of* $S_{\lambda,2}$ *in* $D_{\lambda,2}$*, i.e., each wire has an exact counterpart in the other circuit. Let* $\{P_\lambda\}_\lambda$ *be a family of predicates that takes in* $\mathbf{a}_\lambda, \mathsf{crs}, \Pi$ *and the wire values in* $S_{\lambda,1}$ *or* $S_{\lambda,2}$*.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |D_{\lambda,1}| + |D_{\lambda,2}|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_{\lambda,1}| + |S_{\lambda,2}|)$*.*

*Let* $t = s = \mathsf{poly}(\lambda + L + |C_\lambda| + |\mathcal{A}_\lambda|)$*, and* $\ell_{\mathcal{CEF}} = \mathsf{poly}(\lambda + |S_{\lambda,1}| + |S_{\lambda,2}|)$*. Assuming polynomial hardness of LWE, there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof from*

$$\begin{aligned}&\textit{Let } (\mathsf{rel}, \mathbf{e}) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_{\lambda,1}(\mathbf{a}_\lambda, \cdot), S_{\lambda,1});\\ &\textit{Then } \mathsf{rel} \to P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{e}).\end{aligned}$$

*to*

$$\begin{aligned}&\textit{Let } (\mathsf{rel}, \mathbf{e}) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_{\lambda,2}(\mathbf{a}_\lambda, \cdot), S_{\lambda,2});\\ &\textit{Then } \mathsf{rel} \to P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{e}).\end{aligned}$$

*and vice versa, where all consumed randomness are random variables, and* $\mathsf{crs}, \Pi$ *are generated inside* $\mathsf{LAG}^D_\lambda$*.*

**Remark 7.14.** *The lemma statement only allows* $D_{\lambda,1}$ *and* $D_{\lambda,2}$ *to take the same* $\mathbf{a}_\lambda$*. However the proof can be easily extended to the case where* $D_{\lambda,1}$ *and* $D_{\lambda,2}$ *take in different inputs* $\mathbf{a}_{\lambda,1}, \mathbf{a}_{\lambda,2}$*, as long as the common dependency of* $S_{\lambda,1}$ *and* $S_{\lambda,2}$ *only takes in a subset of* $\mathbf{a}_{\lambda,1} \cap \mathbf{a}_{\lambda,2}$*.*

**Remark 7.15.** *Combining this lemma with Remark 7.10 and Remark 7.11, we can freely switch circuits and extraction sets, while preserving consistency and any other properties on their intersection.*

*Proof.* In the proof, we will make use of the lower level primitive $\mathsf{LAG}^E_\lambda$. We also overload the notation to $\mathsf{LAG}^E_\lambda(\mathcal{A}_\lambda, (E_{\lambda,1}, S_{\lambda,1}), (E_{\lambda,2}, S_{\lambda,2}))$: In this case, $E_{\lambda,1}, E_{\lambda,2}$ occupy $\mathsf{ct}_1, \mathsf{ct}_2$ respectively in the $\mathsf{SNARG}$ construction, and we extract $S_{\lambda,1}$ from $E_{\lambda,1}$, $S_{\lambda,2}$ from $E_{\lambda,2}$. However, we give no guarantees on cross-circuit consistency of extractions. For example, even if $E_{\lambda,1} = E_{\lambda,2}$ and $S_{\lambda,1} = S_{\lambda,2}$, it's still yet conceptually possible that $\mathsf{LAG}^E_\lambda(\mathcal{A}_\lambda, (E_{\lambda,1}, S_{\lambda,1}), (E_{\lambda,2}, S_{\lambda,2}))$ outputs different values for the two extractions.

Let $E_{\lambda,1} = E_\lambda(D_{\lambda,1}), E_{\lambda,2} = E_\lambda(D_{\lambda,2})$. Let $E_{\lambda,3}$ be a "merge" of $E_{\lambda,1}$ and $E_{\lambda,2}$, in the sense that it computes both $E_{\lambda,1}(\mathbf{a}_\lambda, \tau_{C_\lambda}, \mathsf{hk}'_{\mathsf{SSB}})$ and $E_{\lambda,2}(\mathbf{a}_\lambda, \tau_{C_\lambda}, \mathsf{hk}'_{\mathsf{SSB}})$, but the common part is only computed once. By definition, the common part includes the dependency circuits of $S_{\lambda,1}$ in $D_{\lambda,1}$ and $S_{\lambda,2}$ in $D_{\lambda,2}$. Let $S_{\lambda,3}$ denote the sets of wire indices in $E_{\lambda,3}$ that correspond to $S_{\lambda,1}$ and $S_{\lambda,2}$.

Our proof will proceed in five steps (this only proves one direction, the other is symmetric):

1. We switch $\mathsf{ct}_2$ to encrypt $E_{\lambda,3}(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}})$, and additionally extract $S_{\lambda,3}$ from $E_{\lambda,3}$;
2. We prove that the extraction from $E_{\lambda,1}$ is the same that from $E_{\lambda,3}$;
3. We switch $\mathsf{ct}_1$ to encrypt $E_{\lambda,2}(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}})$, and extract $S_{\lambda,2}$ from $E_{\lambda,2}$ instead of $S_{\lambda,1}$ from $E_{\lambda,1}$;
4. We prove that the extraction from $E_{\lambda,2}$ is the same that from $E_{\lambda,3}$;
5. We switch $\mathsf{ct}_2$ back to random, and remove the extraction from $E_{\lambda,3}$.

Combining the five steps, we get the desired result.

The steps 1,3,5 are almost a direct application of Distribution Shift rule and the security of $\mathsf{FHE}$. We do note that the Distribution Shift rule cannot be applied directly, since $\mathbf{a}_\lambda$ is a proper variable, and it involves switching encryption of zeros to encryption of a proper variable. However since $\mathsf{FHE.Enc}$ does the encryption in a bit-by-bit manner, we can break down the encryption process, and for each bit apply the Cryptographic Gadget Substitution rule. The details are omitted here.

Since step 2 and step 4 are symmetric, we will devote the rest of the proof to step 2. After applying step 1, we are currently at

$$\begin{aligned}&\text{Let } \mathsf{hk}'_{\mathsf{SSB}} \leftarrow \mathsf{SSB.Gen}\\ &\quad (\mathsf{rel}, \mathbf{e}_1, \mathbf{e}_3) \coloneqq \mathsf{LAG}^E_\lambda(\mathcal{A}_\lambda, (E_{\lambda,1}(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}}), S_{\lambda,1}), (E_{\lambda,3}(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}}), S_{\lambda,3}));\\ &\text{Then } \mathsf{rel} \to P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{e}_1[S_{\lambda,1}]).\end{aligned}$$

Our proof strategy is simple: Since both $E_{\lambda,1}$ and $E_{\lambda,3}$ contain $D_{\lambda,1}$ and its $\mathsf{SSB}$ extension, which is a $\mathsf{SSB}$ Merkle tree, we will go wire-by-wire to gradually prove the consistency of the Merkle tree root. Then we will switch $\mathsf{hk}'_{\mathsf{SSB}}$ to be perfectly binding at $S_1$, and the consistency of the root will imply the consistency of the these wire values.

For any $k$, the first $k$ wires in $D_{\lambda,1}$ forms a prefix in the Merkle tree in $E_{\lambda,1}$, so they will be exactly covered by $O(\log \lambda)$ nodes. Let $S_{1,k}$ denote these nodes in $E_{\lambda,1}$, and let $S_{3,k}$ be their counterparts in $E_{\lambda,3}$ (we omit the $\lambda$ subscript here).

Suppose that we are currently at

$$\begin{aligned}&\text{Let } \mathsf{hk}'_{\mathsf{SSB}} \leftarrow \mathsf{SSB.Gen}\\ &\quad (\mathsf{rel}, (\mathbf{e}_1, \mathbf{e}'_1), (\mathbf{e}_3, \mathbf{e}'_3)) \coloneqq \mathsf{LAG}^E_\lambda(\mathcal{A}_\lambda, (E_{\lambda,1}(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}}), S_{\lambda,1} \cup S_{1,k}), (E_{\lambda,3}(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}}), S_{\lambda,3} \cup S_{3,k}));\\ &\text{Then } \mathsf{rel} \to (P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{e}_1[S_{\lambda,1}]) \wedge \mathbf{e}'_1 = \mathbf{e}'_3).\end{aligned}$$

where $\mathbf{e}'_1, \mathbf{e}'_3$ are the extraction results for $S_{1,k}, S_{3,k}$ respectively. For $k = 0$, this is exactly the result after step 1.

We will prove that we can go from $k$ to $k+1$. To do so, we will 1) switch $\mathsf{hk}'_{\mathsf{SSB}}$ to be perfectly binding at $k+1$, and also its input wire indices (if any); and 2) switch $\mathsf{hk}_{\mathsf{SSB}}$ to extract those wires, and also their path to root in the Merkle tree.

Since the input wires of $k+1$ are already covered by $\mathbf{e}'_1, \mathbf{e}'_3$, we can use $\mathsf{BARG}$ extraction (Theorem C.1), Lemma 7.2, and $\mathsf{SSB}$ binding, as in the proof of Lemma 7.3, to prove that the input wire values are the same in both circuits. Then the $(k+1)$-th wire value must also be the same if the gate operation is computed correctly. Finally, combining the $(k+1)$-th wire values and $S_{1,k}, S_{3,k}$, we can extract the hash values in $S_{1,k+1}, S_{3,k+1}$, and they should be the same as well.

Therefore, we have proved that we can go from $k$ to $k+1$. By induction, we can go all the way to $k = |D_{\lambda,1}|$, which means the Merkle tree root is the same between the two circuits. Now we can switch $\mathsf{hk}'_{\mathsf{SSB}}$ to be perfectly binding at $S_1$, and the consistency of the root implies the consistency of all wire values in $S_1$, i.e., $\mathbf{e}_1 = \mathbf{e}_3$. □

Unfortunately, while strong, Lemma 7.13 alone is not enough for our purpose. We will prove another two lemmas that seems niche but cannot be directly derived from Lemma 7.13.

The first lemma states that, if the circuit $D_\lambda$ is bottlenecked through a small set of intermediate wires, then we can extract from these wires only, and do postprocessing on them outside the SNARG.

**Lemma 7.16.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{a}_\lambda\}_\lambda$ *be a family of vectors of proper variables. Let* $\{D_\lambda\}_\lambda$ *be a family of circuits that take in* $\mathbf{a}_\lambda, \tau_{C_\lambda}$ *as input. Let* $\{S_\lambda, S'_\lambda\}_\lambda$ *be two families of sets of wire indices in* $D_\lambda$*, such that the wire values of* $S'_\lambda$ *can be computed from those of* $S_\lambda$ *and* $\mathbf{a}_\lambda$ *through a circuit* $D'_\lambda$*, which is a subcircuit of* $D_\lambda$*.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |D_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_\lambda| + |S'_\lambda|)$*.*

*Let* $t = s = \mathsf{poly}(\lambda + L + |C_\lambda| + |\mathcal{A}_\lambda|)$*, and* $\ell_{\mathcal{CEF}} = \mathsf{poly}(\lambda + |S_\lambda| + |S'_\lambda|)$*. Assuming polynomial hardness of LWE, there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof for*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, (\mathbf{e}, \mathbf{e}')) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_\lambda(\mathbf{a}_\lambda, \cdot), S_\lambda \cup S'_\lambda);\\ &\textit{Then } \mathsf{rel} \to (\mathbf{e}' = D'_\lambda(\mathbf{a}_\lambda, \mathbf{e})). \end{aligned}$$

*where all consumed randomness are random variables.*

*Proof.* Let $E_\lambda = E_\lambda(D_\lambda), E'_\lambda = E_\lambda(D'_\lambda)$, and $\hat{E}_\lambda$ be a "merge" of $E_\lambda$ and $E'_\lambda$, but the common part (i.e., $D'_\lambda$) is only computed once.

The proof is similar to Lemma 7.13. It consists of four steps:

1. We switch $\mathsf{ct}_2$ to encrypt $\hat{E}_\lambda(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}})$, and additionally extract $S_\lambda \cup S'_\lambda$ from $\hat{E}_\lambda$;
2. We prove that the extraction from $E_\lambda$ is consistent with that from $\hat{E}_\lambda$;
3. We compute $E'_\lambda(\mathbf{a}_\lambda, \mathbf{e}, \mathsf{hk}'_{\mathsf{SSB}})$ outside the SNARG, and prove that the result is consistent with the extraction from $\hat{E}_\lambda$, and therefore equal to $\mathbf{e}'$;
4. We switch $\mathsf{ct}_2$ back to random.

The main work is in steps 2 and 3, which are similar to the proof of Lemma 7.13. □

The second lemma states that, if the circuit $D_\lambda$ does preprocessing on $\mathbf{a}$, then we can remove this preprocessing and directly supply the preprocessed values as part of $\mathbf{a}$.

**Lemma 7.17.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{a}_\lambda\}_\lambda$ *be a family of vectors of proper variables. Let* $\{D_\lambda\}_\lambda$ *be a family of circuits that take in* $\mathbf{a}_\lambda, \tau_{C_\lambda}$ *as input. Let* $\{D'_\lambda\}_\lambda$ *be another family of circuits that is a subcircuit of* $D_\lambda$*, such that* $D'_\lambda$ *takes in* $\mathbf{a}_\lambda$ *as input and outputs* $\mathbf{a}'_\lambda$*. Let* $D''_\lambda = D_\lambda \setminus D'_\lambda$ *denote the circuit that takes in* $\mathbf{a}'_\lambda, \tau_{C_\lambda}$ *as input and outputs the same as* $D_\lambda$*. Let* $\{S_\lambda\}_\lambda$ *be a family of sets of wire indices in* $D_\lambda$*. Let* $S''_\lambda$ *be the corresponding set of wire indices in* $D''_\lambda$*. Let* $\{P_\lambda\}_\lambda$ *be a family of predicates that takes in* $\mathbf{a}_\lambda, \mathsf{crs}, \Pi$ *and the wire values in* $S_\lambda$ *or* $S''_\lambda$*.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |D_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + |S_\lambda|)$*.*

*Let* $t = s = \mathsf{poly}(\lambda + L + |C_\lambda| + |\mathcal{A}_\lambda|)$*, and* $\ell_{\mathcal{CEF}} = \mathsf{poly}(\lambda + |S_\lambda|)$*. Assuming polynomial hardness of LWE, there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof from*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, \mathbf{e}) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_\lambda(\mathbf{a}_\lambda, \cdot), S_\lambda);\\ &\textit{Then } \mathsf{rel} \to P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{e}). \end{aligned}$$

*to*

$$\begin{aligned} &\textit{Let } \mathbf{a}'_\lambda \coloneqq D'_\lambda(\mathbf{a}_\lambda);\\ &\quad (\mathsf{rel}, \mathbf{e}) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D''_\lambda(\mathbf{a}'_\lambda, \cdot), S''_\lambda);\\ &\textit{Then } \mathsf{rel} \to P_\lambda(\mathbf{a}_\lambda, \mathsf{crs}, \Pi, \mathbf{e}). \end{aligned}$$

*and vice versa, where all consumed randomness are random variables, and* $\mathsf{crs}, \Pi$ *are generated inside* $\mathsf{LAG}^D_\lambda$*.*

*Proof.* Let $E_\lambda = E_\lambda(D_\lambda), E'_\lambda = E_\lambda(D'_\lambda), E''_\lambda = E_\lambda(D''_\lambda)$, and $\hat{E}_\lambda$ be a "merge" of $E_\lambda, E'_\lambda, E''_\lambda$, but the common part (i.e., the entire $D_\lambda$, including $D'_\lambda$ and $D''_\lambda$) is only computed once.

The proof is similar to Lemma 7.13. It consists of six steps:

1. We switch $\mathsf{ct}_2$ to encrypt $\hat{E}_\lambda(\mathbf{a}_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}})$, and additionally extract $S_\lambda$ from $\hat{E}_\lambda$;
2. We prove that the extraction from $E_\lambda$ is consistent with that from $\hat{E}_\lambda$;
3. We compute $\mathbf{a}'_\lambda \leftarrow E'_\lambda(\mathbf{a}_\lambda, \mathsf{hk}'_{\mathsf{SSB}})$ outside the SNARG, and prove that the Merkle tree root is consistent with the extraction from $\hat{E}_\lambda$.
4. We switch $\mathsf{ct}_1$ to encrypt $E''_\lambda(\mathbf{a}'_\lambda, \cdot, \mathsf{hk}'_{\mathsf{SSB}})$, and extract $S''_\lambda$ from $E''_\lambda$ instead of $S_\lambda$ from $E_\lambda$;
5. We prove that the extraction from $E''_\lambda$ is consistent with that from $\hat{E}_\lambda$;
6. We switch $\mathsf{ct}_2$ back to random.

The main work is in steps 2,3,5, which are similar to the proof of Lemma 7.13. □

So far, we have not yet used an important fact that the verifier checks validity of the proofs $\pi_C^{(\gamma)}$. In particular, they must prove that the value $v_C$ can be opened to a valid computation trace $\tau_C$. This trace $\tau_C$ is then provided as input to $D$. To add a finishing touch, we prove that if $D$ computes $C(\mathbf{x}, \mathbf{w})$ as a subcircuit, then the extraction of the subcircuit's output wire value must be 1.

**Lemma 7.18.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\{\mathbf{a}_\lambda\}_\lambda$ *be a family of vectors of proper variables; Let* $\{D_\lambda\}_\lambda$ *be a family of circuits that take in* $\mathbf{a}_\lambda, \tau_{C_\lambda}$ *as input. In particular, suppose that* $D_\lambda$ *contains a subcircuit that computes* $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda)$*, where* $\mathbf{w}_\lambda$ *is taken from the input wire values in* $\tau_{C_\lambda}$*, and let* $i^*_\lambda$ *be the output wire index of this subcircuit.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |D_\lambda|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda)$*.*

*Let* $t = s = \mathsf{poly}(\lambda + L + |C_\lambda| + |\mathcal{A}_\lambda|)$*, and* $\ell_{\mathcal{CEF}} = \mathsf{poly}(\lambda)$*. Assuming polynomial hardness of LWE, there exists a* $(t, s, \ell_{\mathcal{CEF}})$*-*$\mathcal{CEF}$ *proof for*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_\lambda(\mathbf{a}_\lambda, \cdot), \{i^*_\lambda\});\\ &\textit{Then } \mathsf{rel} \to o. \end{aligned}$$

*where all consumed randomness are random variables.*

*Proof.* Using Lemma 7.13, we can add or remove any dummy circuit in $D_\lambda$ without changing the extraction result at $i^*_\lambda$. Therefore, let's assume that inside $D_\lambda$, after computing $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda)$, we also check that each wire value is the same as the corresponding value in $\tau_{C_\lambda}$, and build a $\log|C_\lambda|$-depth binary AND tree to AND every check together.

We go wire-by-wire to prove consistency. For any $k$, the first $k$ wires in $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda)$ forms a prefix in the AND tree, so they will be exactly covered by $O(\log|C_\lambda|)$ nodes. Let $S_k$ denote these nodes. Suppose that we are currently at

$$\text{Let } (\mathsf{rel}, \mathbf{e}) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_\lambda(\mathbf{a}_\lambda, \cdot), S_k);$$
$$\text{Then } \mathsf{rel} \to \bigwedge_{i=1}^{k} \mathbf{e}[i].$$

For $k = 0$, this is an empty statement that always holds.

To go from $k$ to $k+1$, we use the extracted $S_k$, $\mathsf{SSB}$ binding and $\mathsf{BARG}$ extraction to prove the consistency of the input wires of the $(k+1)$-th wire, and then the consistency of the $(k+1)$-th wire as well. For the trace $\tau_{C_\lambda}$, $\mathsf{BARG}$ extraction applies Theorem C.1 to the proofs $\Pi.\pi^{(\gamma)}_C$ as in the proof of Lemma 7.3, and the second part of Lemma 7.2, applied to the honest hash $h_C$, shows that the gate descriptions in the extracted witnesses are those of $C_\lambda$. This argument is very similar to Lemma 7.13, so we omit the details here. In the end, we prove that the AND tree root is 1.

Finally, since the proofs $\pi^{(\gamma)}_C$ include a check that the output wire of $\tau_{C_\lambda}$ is 1, consistency implies that the output wire of $C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda)$ is also 1. □

## 7.2 Proof of Core Lemma

We formalize the connection between a $\mathcal{PEF}$ configuration and a circuit.

**Definition 7.19** ($\mathcal{PEF}$ Configuration as a Circuit)**.** *Given a configuration $\Phi$, we define its corresponding circuit $D_\Phi$, its advice $\mathbf{a}_\Phi$, and the wire index $i_\Phi$ as follows:*

- *$\mathbf{a}_\Phi$ is defined as the concatenation of all random variables and all proper variables in $\Phi$, except those in $\mathbf{w}$;*
- *$D_\Phi$ is constructed as follows:*
  - *Create an input wire for each bit in $\mathbf{a}_\Phi$;*
  - *Create an input wire for each bit in $\mathbf{w}$;*
  - *Go through the formulas one by one:*
    - *If it is a extension formula $x \coloneqq C(y_1, \ldots, y_k)$, create a subcircuit in $D_\Phi$ that takes in $y_1, \ldots, y_k$ as input, computes $C$, and outputs $x$;*
    - *If it is a normal formula $P(y_1, \ldots, y_k)$, create a subcircuit in $D_\Phi$ that takes in $y_1, \ldots, y_k$ as input, computes $P$, and outputs a single bit;*
  - *The output of $D_\Phi$ is the AND of all output bits from normal formulas. This is done by building a $\log|\Phi|$-depth binary tree of AND gates.*
- *$i_\Phi$ is the output wire index of $D_\Phi$.*

We will break down the proof of Lemma 6.3 into the smallest steps possible: going from one configuration to another using either an $\mathcal{EF}$ step or a Cryptographic Gadget Substitution step. We will prove the derivation for each step, and then combine them together to get the final result.

In this section, we will use $\ell_{\mathcal{PEF}}$ to denote the parameter $\ell$ in the definition of $\mathcal{PEF}$ proofs, and $\ell_{\mathsf{SNARG}}$ to denote the parameter $\ell$ in the SNARG construction.

First, we prove the derivation for a $\mathcal{EF}$ step, i.e., a step of Erasure, Extension or Logical Inference.

**Lemma 7.20.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\Phi, \Phi'$ *be any two configurations, such that there is a* $\mathcal{PEF}$ *proof from* $\Phi$ *to* $\Phi'$ *using a single step of Erasure, Extension or Logical Inference.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |\Phi| + |\Phi'|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda)$.

*Assuming polynomial hardness of LWE, for any inverse polynomial* $\epsilon(\lambda)$*, for any* $\lambda$*, there is a* $(\mathsf{poly}(L(\lambda)), \mathsf{poly}(L(\lambda)), \mathsf{poly}(\lambda), \epsilon(\lambda))$*-*$\mathcal{PEF}$ *proof from*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_\Phi(\mathbf{a}_\Phi, \cdot), \{i_\Phi\});\\ &\textit{Then } \mathsf{rel} \to o. \end{aligned}$$

*to*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\Phi'}(\mathbf{a}_{\Phi'}, \cdot), \{i_{\Phi'}\});\\ &\textit{Then } \mathsf{rel} \to o. \end{aligned}$$

*where all consumed randomness are random variables.*

*Proof.* We consider the three cases separately.

**Erasure.** If an extension formula is erased, then $D_{\Phi'}$ is a subcircuit of $D_\Phi$. We can derive the result using Lemma 7.13.

If a normal formula is erased, then $D_{\Phi'}$ is almost a subcircuit of $D_\Phi$, except that the binary AND tree is slightly reordered. We do the derivation in several steps:

- We rebuild a new AND tree that computes the AND of all normal formulas except the erased one. By Lemma 7.13, this does not affect the extraction of the old AND tree.
- One by one, we prove that extracting each node in the new AND tree gives 1, using the consistency proved in Remark 7.10. At any intermediate step we only need to extract $\log |\Phi'|$ nodes to "save progress". In the end we prove that the root of the new AND tree is 1.
- We remove the subcircuit that corresponds to the erased formula, and the old AND tree. By Lemma 7.13, this does not affect the extraction of the new AND tree.

**Extension.** $D_{\Phi'}$ is obtained from $D_\Phi$ by adding some dummy wires that do not affect the output. We can derive the result using Lemma 7.13.

**Logical Inference.** There exists formulas $\phi_1, \ldots, \phi_c$ in $\Phi$, such that $\phi_1, \ldots, \phi_c \vdash \phi$. $D_{\Phi'}$ is obtained from $D_\Phi$ by adding a subcircuit $\phi$ and putting it into the AND tree. We do the derivation in several steps:

- We add in a subcircuit that computes $\phi$. By Lemma 7.13, this does not affect the extraction of the old AND tree root.
- We extract the leaves corresponding to $\phi_1, \ldots, \phi_c$. By Remark 7.10, they are all 1.
- We extract the relevant variables in $\phi_1, \ldots, \phi_c$, and also the output of $\phi$. By Remark 7.10, they are consistent. Since $\phi_1, \ldots, \phi_c \vdash \phi$, we can use the same Logical Inference Rule to prove the output of $\phi$ is 1.
- We build a new AND tree that includes the output of $\phi$, prove bottom-up that the extraction of the new AND tree root is 1, and remove the old AND tree.

□

Next, we prove the derivation for a Cryptographic Gadget Substitution step. This requires several different techniques for different parameter regimes, so we will break it down into three lemmas.

First, we prove the case where $|\mathbf{z}| \leq \ell \wedge |\mathbf{y}| \leq \log \ell$.

**Lemma 7.21.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\Phi, \Phi'$ *be any two configurations, such that there is a* $\mathcal{PEF}$ *proof from* $\Phi$ *to* $\Phi'$ *using a single Cryptographic Gadget Substitution step with parameters* $(\ell_{\mathcal{PEF}}, \epsilon')$*, and* $|\mathbf{z}| \leq \ell_{\mathcal{PEF}} \wedge |\mathbf{y}| \leq \log \ell_{\mathcal{PEF}}$.

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |\Phi| + |\Phi'|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda + \ell_{\mathcal{PEF}})$.

*Assuming polynomial hardness of LWE, for any inverse polynomial* $\epsilon(\lambda)$*, for any* $\lambda$ *such that* $\epsilon' \leq \epsilon(\lambda)$*, there is a* $(\mathsf{poly}(L(\lambda)), \mathsf{poly}(L(\lambda)), \mathsf{poly}(\lambda + \ell_{\mathcal{PEF}}), \epsilon(\lambda))$*-*$\mathcal{PEF}$ *proof from*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_\Phi(\mathbf{a}_\Phi, \cdot), \{i_\Phi\}); \\ &\textit{Then } \mathsf{rel} \to o. \end{aligned}$$

*to*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\Phi'}(\mathbf{a}_{\Phi'}, \cdot), \{i_{\Phi'}\}); \\ &\textit{Then } \mathsf{rel} \to o. \end{aligned}$$

*where all consumed randomness are random variables.*

*Proof.* Let's fix a $\lambda$, and reason in terms of $\mathcal{PEF}$ proof instead of $\mathcal{CEF}$ proof.

Let $\Phi = F \cup \{\mathbf{y} \coloneqq f(\mathbf{z}, \mathbf{c})\}, \quad \Phi' = F \cup \{\mathbf{y} \coloneqq g(\mathbf{z}, \mathbf{c}')\}$, where $f, g$ are the two gadgets where we apply the substitution.

Since $|\mathbf{y}|$ is small, we can manipulate $\Phi$ into $\hat{\Phi}$ using only a polynomial number of $\mathcal{EF}$ steps, where $\hat{\Phi}$ is of the form $\{(T, \mathbf{z}) \coloneqq C(\ldots)\} \cup \{\mathbf{y} \coloneqq f(\mathbf{z}, \mathbf{c})\} \cup \{T[\mathbf{y}]\}$, and $C$ does not read from $\mathbf{y}$ or $\mathbf{c}$. Basically, we modify $F$ to compute the result for any $\mathbf{y}$, store the results into a truth table $T$ (which is of size $2^{\log \ell'}$), and assert that $T[\mathbf{y}]$ is true.

By Lemma 7.20, we can go from the beginning to

$$\begin{aligned}&\text{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\hat{\Phi}}(\mathbf{a}_{\hat{\Phi}}, \cdot), \{i_{\hat{\Phi}}\});\\ &\text{Then } \mathsf{rel} \to o.\end{aligned}$$

Next, let $S_T$ be the set of wire indices in $D_{\hat{\Phi}}$ that correspond to the truth table $T$, and $S_{\mathbf{z}}$ be the set of wire indices that correspond to $\mathbf{z}$. Since $o = T[f(\mathbf{z}, \mathbf{c})]$ and $\mathbf{c}$ is included in $\mathbf{a}_{\hat{\Phi}}$, by Lemma 7.16, we can instead extract $S_T \cup S_{\mathbf{z}}$, and do postprocessing outside the SNARG to compute $o$. Therefore, we can arrive at

$$\begin{aligned}&\text{Let } (\mathsf{rel}, (T, \mathbf{z})) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\hat{\Phi}}(\mathbf{a}_{\hat{\Phi}}, \cdot), S_T \cup S_{\mathbf{z}});\\ &\text{Then } \mathsf{rel} \to T[f(\mathbf{z}, \mathbf{c})].\end{aligned}$$

Next, let $\tilde{\Phi} = \{(T, \mathbf{z}) \coloneqq C(\ldots)\}$. That is, $\tilde{\Phi}$ only computes the truth table $T$, but does not use it. By Lemma 7.13, we can arrive at

$$\begin{aligned}&\text{Let } (\mathsf{rel}, (T, \mathbf{z})) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\tilde{\Phi}}(\mathbf{a}_{\tilde{\Phi}}, \cdot), S_T \cup S_{\mathbf{z}});\\ &\text{Then } \mathsf{rel} \to T[f(\mathbf{z}, \mathbf{c})].\end{aligned}$$

where $\mathbf{c}$ is no longer included in $\mathbf{a}_{\tilde{\Phi}}$, and is only used once in computing $f$.

Finally, using the Cryptographic Gadget Substitution rule, we replace the formula $\mathbf{y} \coloneqq f(\mathbf{z}, \mathbf{c})$ by $\mathbf{y} \coloneqq g(\mathbf{z}, \mathbf{c}')$. This finishes the transform from $f$ to $g$. To go back from truth table computation to the original configuration, we repeat the previous steps, but in the reverse order. □

Next, we prove the case where $|\mathbf{z}| = 0$.

**Lemma 7.22.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\Phi, \Phi'$ *be any two configurations, such that there is a* $\mathcal{PEF}$ *proof from* $\Phi$ *to* $\Phi'$ *using a single Cryptographic Gadget Substitution step with parameter* $\epsilon'$*, and* $|\mathbf{z}| = 0$*.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \geq \mathsf{poly}(\lambda + |\Phi| + |\Phi'|)$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \geq \mathsf{poly}(\lambda)$*.*

*Assuming polynomial hardness of LWE, for any inverse polynomial* $\epsilon(\lambda)$*, for any* $\lambda$ *such that* $\epsilon' \leq \epsilon(\lambda)$*, there is a* $(\mathsf{poly}(L(\lambda)), \mathsf{poly}(L(\lambda)), \mathsf{poly}(\lambda), \epsilon(\lambda))$*-*$\mathcal{PEF}$ *proof from*

$$\begin{aligned}&\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_\Phi(\mathbf{a}_\Phi, \cdot), \{i_\Phi\});\\ &\textit{Then } \mathsf{rel} \to o.\end{aligned}$$

*to*

$$\begin{aligned}&\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\Phi'}(\mathbf{a}_{\Phi'}, \cdot), \{i_{\Phi'}\});\\ &\textit{Then } \mathsf{rel} \to o.\end{aligned}$$

*where all consumed randomness are random variables.*

*Proof.* This is almost a direct application of Lemma 7.17.

Let $\Phi = F \cup \{\mathbf{y} \coloneqq f(\mathbf{c})\}$, $\Phi' = F \cup \{\mathbf{y} \coloneqq g(\mathbf{c}')\}$, where $f, g$ are the two gadgets where we apply the substitution. Starting from $D_\Phi$, we use Lemma 7.17 to remove the preprocessing that computes $f(\mathbf{c})$ from $\mathbf{c}$, and directly supply $f(\mathbf{c})$ as part of $\mathbf{a}$. Then we use the Cryptographic Gadget Substitution rule to replace $f(\mathbf{c})$ by $g(\mathbf{c}')$. Finally, we use Lemma 7.17 in reverse to add back the preprocessing that computes $g(\mathbf{c}')$ from $\mathbf{c}'$, arriving at $D_{\Phi'}$. □

Finally, we prove the case where $|\mathbf{z}| \le \log s$.

**Lemma 7.23.** *There exists a polynomial* $\mathsf{poly} = \mathsf{poly}(\cdot)$ *such that the following holds.*

*Let* $\Phi, \Phi'$ *be any two configurations, such that there is a* $\mathcal{PEF}$ *proof from* $\Phi$ *to* $\Phi'$ *using a single Cryptographic Gadget Substitution step with parameters* $(s', \epsilon')$*, and* $|\mathbf{z}| \le \log s'$*.*

*Suppose that* $L = L(\lambda)$ *and* $\ell_{\mathsf{SNARG}} = \ell_{\mathsf{SNARG}}(\lambda)$ *satisfy* $L(\lambda) \ge \mathsf{poly}(\lambda + |\Phi| + |\Phi'| + s')$ *and* $\ell_{\mathsf{SNARG}}(\lambda) \ge \mathsf{poly}(\lambda)$*.*

*Assuming polynomial hardness of LWE, for any inverse polynomial* $\epsilon(\lambda)$*, for any* $\lambda$ *such that* $\epsilon' \le \epsilon(\lambda)$*, there is a* $(\mathsf{poly}(L(\lambda)), \mathsf{poly}(L(\lambda)), \mathsf{poly}(\lambda), \epsilon(\lambda))$*-*$\mathcal{PEF}$ *proof from*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_\Phi(\mathbf{a}_\Phi, \cdot), \{i_\Phi\});\\ &\textit{Then } \mathsf{rel} \to o. \end{aligned}$$

*to*

$$\begin{aligned} &\textit{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\Phi'}(\mathbf{a}_{\Phi'}, \cdot), \{i_{\Phi'}\});\\ &\textit{Then } \mathsf{rel} \to o. \end{aligned}$$

*where all consumed randomness are random variables.*

*Proof.* Let's fix a $\lambda$, and reason in terms of $\mathcal{PEF}$ proof instead of $\mathcal{CEF}$ proof.

Let $\Phi = F \cup \{\mathbf{y} \coloneqq f(\mathbf{z}, \mathbf{c})\}$, $\quad \Phi' = F \cup \{\mathbf{y} \coloneqq g(\mathbf{z}, \mathbf{c}')\}$, where $f, g$ are the two gadgets where we apply the substitution.

Let's split $\mathbf{c}$ into $c_1 c_2 \dots c_{|\mathbf{c}|}$, and we shall focus on $c_1$. Our goal is to change $c_1$ into an extension variable: $c_1 \coloneqq c_1^{(\mathbf{z})}$, where $c_1^{(00\dots0)}, \dots, c_1^{(11\dots1)}$ are $2^{|\mathbf{z}|}$ new random variables. Since $|\mathbf{z}| \le \log s'$, this only introduces at most $s'$ new variables and formulas.

Using Lemma 7.21 once, we can change $c_1$ into $c_1 \coloneqq c_1^{(\mathbf{z}[1])}$, where $c_1^{(0)}, c_1^{(1)}$ are two new random variables, and $\mathbf{z}[1]$ is the first bit of $\mathbf{z}$. Then use Lemma 7.21 again to change $c_1^{(0)}$ into $c_1^{(0,\mathbf{z}[2])}$, and $c_1^{(1)}$ into $c_1^{(1,\mathbf{z}[2])}$. Continuing this way, after $2^{|\mathbf{z}|}$ times we can change $c_1$ into $c_1^{(\mathbf{z})}$.

Repeating the same process for each bit in $\mathbf{c}$, we can change $\mathbf{c}$ into $\mathbf{c}^{(\mathbf{z})}$, where $\mathbf{c}^{(00\dots0)}, \dots, \mathbf{c}^{(11\dots1)}$ are $2^{|\mathbf{z}|}$ new strings of random variables. This takes $|\mathbf{c}| \cdot 2^{|\mathbf{z}|}$ times of Lemma 7.21. Let $\hat{\Phi} = F \cup \{\mathbf{y} \coloneqq f(\mathbf{z}, \mathbf{c})\} \cup \{\mathbf{c} \coloneqq \mathbf{c}^{(\mathbf{z})}\}$. We now arrive at

$$\begin{aligned} &\text{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\hat{\Phi}}(\mathbf{a}_{\hat{\Phi}}, \cdot), \{i_{\hat{\Phi}}\});\\ &\text{Then } \mathsf{rel} \to o. \end{aligned}$$

Now, for each $\mathbf{x} \in \{0,1\}^{|\mathbf{z}|}$, we can let $\mathbf{y}^{(\mathbf{x})} \coloneqq f(\mathbf{x}, \mathbf{c}^{(\mathbf{x})})$, then set $\mathbf{y} = \mathbf{y}^{(\mathbf{z})}$. Let $\tilde{\Phi} = F \cup \{\mathbf{y}^{(\mathbf{x})} \coloneqq f(\mathbf{x}, \mathbf{c}^{(\mathbf{x})})\}_{\mathbf{x} \in \{0,1\}^{|\mathbf{z}|}} \cup \{\mathbf{y} = \mathbf{y}^{(\mathbf{z})}\}$. Apparently $\tilde{\Phi}$ is equivalent to $\hat{\Phi}$. By Lemma 7.20, we now arrive at

$$\begin{aligned} &\text{Let } (\mathsf{rel}, o) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\tilde{\Phi}}(\mathbf{a}_{\tilde{\Phi}}, \cdot), \{i_{\tilde{\Phi}}\});\\ &\text{Then } \mathsf{rel} \to o. \end{aligned}$$

Finally, for each $\mathbf{x} \in \{0,1\}^{|\mathbf{z}|}$, $\mathbf{y}^{(\mathbf{x})} \coloneqq f(\mathbf{x}, \mathbf{c}^{(\mathbf{x})})$ concerns only random variables, so we can use Lemma 7.22 to change it into $\mathbf{y}^{(\mathbf{x})} \coloneqq g(\mathbf{x}, \mathbf{c}'^{(\mathbf{x})})$. This takes $2^{|\mathbf{z}|}$ times of Lemma 7.22. After that, we reverse the arguments to condense $2^{|\mathbf{z}|}$ random variables back into one. □

Now we are ready to prove the core lemma.

*Proof.* By assumption, we have a poly size $\mathcal{CEF}$ proof for

$$\begin{aligned}\text{Let } \mathsf{pre} &\coloneqq D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda);\\ \mathsf{rel} &\coloneqq C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda);\\ \mathsf{res} &\coloneqq D_{\mathsf{res},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda, \mathbf{w}_\lambda),\\ \text{Then } \mathsf{pre} &\to (\mathsf{rel} \to \mathsf{res}).\end{aligned}$$

where $\mathbf{y}_\lambda, \mathbf{w}_\lambda$ are proper variables, and $\mathbf{x}_\lambda$ is hardwired. Let $\Phi_\lambda$ be the configuration that corresponds to the above statement.

Combining Lemma 7.20, Lemma 7.21, and Lemma 7.23, for each $\lambda$, one step at a time, we can transform the existing $\mathcal{CEF}$ proof into a poly size $\mathcal{CEF}$ proof for

$$\begin{aligned}\text{Let } (\mathsf{rel}, o) &\coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\Phi_\lambda}(\mathbf{y}_\lambda, \cdot), \{i_{\Phi_\lambda}\});\\ \text{Then } \mathsf{rel} &\to o.\end{aligned}$$

where all consumed randomness are random variables.

Next, we need to take the computation of $\mathsf{pre}$ to the outside, and remove the computation of $\mathsf{rel}$ from $\Phi_\lambda$. To this end, we revert to the lower level primitive $\mathsf{LAG}^D_\lambda$, and explicitly write out the circuit $D_{\Phi_\lambda}$. To avoid confusion, we use $\mathsf{rel}_{\mathsf{SNARG}}$ to denote the verification result inside $\mathsf{LAG}^D_\lambda$, and $\mathsf{rel}_D$ to denote the $\mathsf{rel}$ computed in $D_{\Phi_\lambda}$.

Let $D_\lambda$ denote the following circuit (which is almost the same as $D_{\Phi_\lambda}$, except that it takes in $\tau_{C_\lambda}$ instead of $\mathbf{w}_\lambda$):

- $\mathbf{x}_\lambda$ is hardwired; Take in $\mathbf{y}_\lambda$ and $\tau_{C_\lambda}$ as input;
- Let $\mathbf{w}_\lambda$ be the input wire values in $\tau_{C_\lambda}$;
- Let $\mathsf{pre} \leftarrow D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda)$;
- Let $\mathsf{rel}_D \leftarrow C_\lambda(\mathbf{x}_\lambda, \mathbf{w}_\lambda)$;
- Let $\mathsf{res} \leftarrow D_{\mathsf{res},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda, \mathbf{w}_\lambda)$;
- Output $\mathsf{pre} \to (\mathsf{rel}_D \to \mathsf{res})$.

Let $i_\lambda$ be the output wire index of $D_\lambda$. We are currently at

$$\begin{aligned}\text{Let } (\mathsf{rel}_{\mathsf{SNARG}}, o) &\coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D_\lambda(\mathbf{y}_\lambda, \cdot), \{i_\lambda\});\\ \text{Then } \mathsf{rel}_{\mathsf{SNARG}} &\to o.\end{aligned}$$

It's easy to take the computation of $\mathsf{pre}$ outside the SNARG using Lemma 7.17, since $\mathbf{x}_\lambda$ is hardwired and $\mathbf{y}_\lambda$ is part of the advice. Let $D'_\lambda(\mathsf{pre}, \mathbf{y}_\lambda, \tau_{C_\lambda})$ be the same as $D_\lambda$, except that $\mathsf{pre}$ is supplied as part of the advice instead of being computed inside. Let $i'_\lambda$ be the output wire index of $D'_\lambda$. We can arrive at

$$\begin{aligned}\text{Let } \mathsf{pre} &\coloneqq D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda);\\ (\mathsf{rel}_{\mathsf{SNARG}}, o) &\coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D'_\lambda(\mathsf{pre}, \mathbf{y}_\lambda, \cdot), \{i'_\lambda\});\\ \text{Then } \mathsf{rel}_{\mathsf{SNARG}} &\to o.\end{aligned}$$

Using Lemma 7.16, we can extract $\mathsf{rel}_D$ and $\mathsf{res}$ from $D'_\lambda$, and compute $o = (\mathsf{pre} \to (\mathsf{rel}_D \to \mathsf{res}))$ outside the SNARG. Using Lemma 7.18, the extracted $\mathsf{rel}_D$ is 1. Let $i_{\lambda,\mathsf{rel}}, i_{\lambda,\mathsf{res}}$ be the wire indices of $\mathsf{rel}_D$ and $\mathsf{res}$ in $D'_\lambda$, respectively. We can arrive at

$$\begin{aligned}
&\text{Let } \mathsf{pre} \coloneqq D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda);\\
&\quad (\mathsf{rel}_{\mathsf{SNARG}}, (\mathsf{rel}_D, \mathsf{res})) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D'_\lambda(\mathsf{pre}, \mathbf{y}_\lambda, \cdot), \{i_{\lambda,\mathsf{rel}}, i_{\lambda,\mathsf{res}}\});\\
&\text{Then } (\mathsf{rel}_{\mathsf{SNARG}} \to (\mathsf{pre} \to (\mathsf{rel}_D \to \mathsf{res}))) \wedge \mathsf{rel}_D.
\end{aligned}$$

which is equivalent to

$$\begin{aligned}
&\text{Let } \mathsf{pre} \coloneqq D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda);\\
&\quad (\mathsf{rel}_{\mathsf{SNARG}}, (\mathsf{rel}_D, \mathsf{res})) \coloneqq \mathsf{LAG}^D_\lambda(\mathcal{A}_\lambda, D'_\lambda(\mathsf{pre}, \mathbf{y}_\lambda, \cdot), \{i_{\lambda,\mathsf{rel}}, i_{\lambda,\mathsf{res}}\});\\
&\text{Then } \mathsf{pre} \to (\mathsf{rel}_{\mathsf{SNARG}} \to \mathsf{res}).
\end{aligned}$$

Finally, since $\mathsf{res}$ is now the only thing we need to extract from $D'_\lambda$, we use Lemma 7.13 to do the extraction from $D_{\mathsf{res},\lambda}$ instead of $D'_\lambda$. This gives us the final result:

$$\begin{aligned}
&\text{Let } \mathsf{pre} \coloneqq D_{\mathsf{pre},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda);\\
&\quad (\mathsf{rel}_{\mathsf{SNARG}}, \mathsf{res}) \coloneqq \mathsf{LAG}_\lambda(\mathcal{A}_\lambda, D_{\mathsf{res},\lambda}(\mathbf{x}_\lambda, \mathbf{y}_\lambda, \cdot), \{i^*_\lambda\});\\
&\text{Then } \mathsf{pre} \to (\mathsf{rel}_{\mathsf{SNARG}} \to \mathsf{res}).
\end{aligned}$$

where $i^*_\lambda$ is the output wire index of $D_{\mathsf{res},\lambda}$. □

## References


[ABSRW02] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002. 12

[ADVW13] Shweta Agrawal, Yevgeniy Dodis, Vinod Vaikuntanathan, and Daniel Wichs. On continual leakage of discrete log representations. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 401–420. Springer, Berlin, Heidelberg, December 2013. 4, 8

[Bar01] Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd FOCS*, pages 106–115. IEEE Computer Society Press, October 2001. 7

[Bar02] Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *43rd FOCS*, pages 345–355. IEEE Computer Society Press, November 2002. 7

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 252–283. Springer, Cham, August 2023. 3, 4, 12, 29

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120. ACM Press, June 2013. 3

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th ACM STOC*, pages 505–514. ACM Press, May / June 2014. 4

[BDKP22] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. Verifiable private information retrieval. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part III*, volume 13749 of *LNCS*, pages 3–32. Springer, Cham, November 2022. 3

[BG93] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 390–420. Springer, Berlin, Heidelberg, August 1993. 3

[BKP19] Nir Bitansky, Dakshita Khurana, and Omer Paneth. Weak zero-knowledge beyond the black-box barrier. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1091–1102. ACM Press, June 2019. 7

[BL02] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In *34th ACM STOC*, pages 484–493. ACM Press, May 2002. 7

[BP12] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *53rd FOCS*, pages 223–232. IEEE Computer Society Press, October 2012. 7

[BP15] Elette Boyle and Rafael Pass. Limits of extractability assumptions with distributional auxiliary input. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 236–261. Springer, Berlin, Heidelberg, November / December 2015. 4

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In Venkatesan Guruswami, editor, *56th FOCS*, pages 1480–1498. IEEE Computer Society Press, October 2015. 3

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011. 27

[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, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 635–668. Springer, Cham, August 2023. 3

[CGKS23] Matteo Campanelli, Chaya Ganesh, Hamidreza Khoshakhlagh, and Janno Siim. Impossibilities in succinct arguments: Black-box extraction and more. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 465–489. Springer, Cham, July 2023. 4, 8

[CJ25] Liyan Chen and Zhengzhong Jin. On the impossibility of snargs with short crs (or: Revisiting the gentry–wichs barrier in the non-adaptive setting). In *Proceedings of the 66th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2025)*, 2025. 8

[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, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 394–423, Virtual Event, August 2021. Springer, Cham. 3, 4, 12

[CJJ22] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *62nd FOCS*, pages 68–79. IEEE Computer Society Press, February 2022. 3, 25, 29, 30, 72, 73

[CJJQ23] Geoffroy Couteau, Abhishek Jain, Zhengzhong Jin, and Willy Quach. A note on non-interactive zero-knowledge from CDH. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 731–764. Springer, Cham, August 2023. 5

[CKPR01] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires omega (log n) rounds. In *33rd ACM STOC*, pages 570–579. ACM Press, July 2001. 7

[CKU20] Geoffroy Couteau, Shuichi Katsumata, and Bogdan Ursu. Non-interactive zero-knowledge in pairing-free groups from weaker assumptions. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 442–471. Springer, Cham, May 2020. 5

[CL08] Giovanni Crescenzo and Helger Lipmaa. Succinct np proofs from an extractability assumption. In *Proceedings of the 4th Conference on Computability in Europe: Logic and Theory of Algorithms*, CiE '08, page 175–185, Berlin, Heidelberg, 2008. Springer-Verlag. 3

[CLW18] Ran Canetti, Alex Lombardi, and Daniel Wichs. Fiat-shamir: From practice to theory, part II (NIZK and correlation intractability from circular-secure FHE). Cryptology ePrint Archive, Paper 2018/1248, 2018. 69

[Coo75] Stephen A. Cook. Feasibly constructive proofs and the propositional calculus (preliminary version). In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, page 83–97, New York, NY, USA, 1975. Association for Computing Machinery. 5, 9

[CPS13] Kai-Min Chung, Rafael Pass, and Karn Seth. Non-black-box simulation from one-way functions and applications to resettable security. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 231–240. ACM Press, June 2013. 7

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS 2010*, pages 310–331. Tsinghua University Press, January 2010. 3

[Den17] Yi Deng. Magic adversaries versus individual reduction: Science wins either way. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 351–377. Springer, Cham, April / May 2017. 5, 7

[DHK+26] Lalita Devadas, Samuel B. Hopkins, Yael Tauman Kalai, Pravesh K. Kothari, Alex Lombardi, and Surya Mathialagan. SNARGs for NP and non-signaling PCPs, revisited. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing (STOC '26)*, 2026. To appear. Cryptology ePrint Archive, Paper 2026/006, https://eprint.iacr.org/2026/006. 7

[DNRS99] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry J. Stockmeyer. Magic functions. In *40th FOCS*, pages 523–534. IEEE Computer Society Press, October 1999. 7

[ET01] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Inf. Comput.*, 171(1):84–97, nov 2001. 12

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009. 27

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013. 3

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25(1):169–192, 1996. 7

[GKR15] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *Journal of the ACM*, 62(4):27:1–27:64, 2015. 3

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. 3

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Berlin, Heidelberg, December 2010. 3

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013. 19, 27

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011. 7, 8

[HJKS22] James Hulett, Ruta Jawale, Dakshita Khurana, and Akshayaram Srinivasan. SNARGs for P from sub-exponential DDH and QR. In Orr Dunkelman and Stefan Dziembowski,

editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 520–549. Springer, Cham, May / June 2022. 3, 30

[HLR21] Justin Holmgren, Alex Lombardi, and Ron D. Rothblum. Fiat-Shamir via list-recoverable codes (or: parallel repetition of GMW is not zero-knowledge). In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 750–760. ACM Press, June 2021. 68, 70, 71

[HNO+18] Iftach Haitner, Kobbi Nissim, Eran Omri, Ronen Shaltiel, and Jad Silbak. Computational two-party correlation: A dichotomy for key-agreement protocols. In Mikkel Thorup, editor, *59th FOCS*, pages 136–147. IEEE Computer Society Press, October 2018. 5

[HW14] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. Cryptology ePrint Archive, Paper 2014/669, 2014. 66

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st ACM STOC*, pages 44–61. ACM Press, May 1989. 7

[JJ22] Abhishek Jain and Zhengzhong Jin. Indistinguishability obfuscation via mathematical proofs of equivalence. In *63rd FOCS*, pages 1023–1034. IEEE Computer Society Press, October / November 2022. 11, 13, 14

[JJ25] Abhishek Jain and Zhengzhong Jin. Sometimes-decryptable homomorphic encryption from sub-exponential DDH. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 406–439. Springer, Cham, August 2025. 3

[JJMP25] Abhishek Jain, Zhengzhong Jin, Surya Mathialagan, and Omer Paneth. On succinct obfuscation via propositional logic. In *Proceedings of the 66th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2025)*, 2025. 11

[JKLM25] Zhengzhong Jin, Yael Tauman Kalai, Alex Lombardi, and Surya Mathialagan. Universal SNARGs for NP from proofs of correctness. In Michal Koucký and Nikhil Bansal, editors, *57th ACM STOC*, pages 933–943. ACM Press, June 2025. 3, 11, 12, 17, 26

[JKLV24] Zhengzhong Jin, Yael Kalai, Alex Lombardi, and Vinod Vaikuntanathan. SNARGs under LWE via propositional proofs. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 1750–1757. ACM Press, June 2024. 3, 6, 8, 9, 10, 11, 12, 13, 14, 18, 33, 34, 36, 40, 41

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021. 4

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in $NC^0$. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Cham, May / June 2022. 4

[Kiy22] Susumu Kiyoshima. Public-coin 3-round zero-knowledge from learning with errors and keyless multi-collision-resistant hash. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 444–473. Springer, Cham, August 2022. 3

[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 ACM STOC*, pages 1545–1552. ACM Press, June 2023. 3, 29, 30

[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, Part I*, volume 13042 of *LNCS*, pages 330–368. Springer, Cham, November 2021. 3, 29

[KY18] Ilan Komargodski and Eylon Yogev. On distributional collision resistant hashing. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 303–327. Springer, Cham, August 2018. 5

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology, January/February 1978. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF. 8

[MDS25] Yaohua Ma, Chenxin Dai, and Elaine Shi. Quasi-linear indistinguishability obfuscation via mathematical proofs of equivalence and applications. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part III*, volume 15603 of *LNCS*, pages 157–186. Springer, Cham, May 2025. 11

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. 3, 4

[Mie08] Thilo Mie. Polylogarithmic two-round argument systems. *Journal of Mathematical Cryptology*, 2(4):343–363, 2008. 3

[MPS10] Hemanta K. Maji, Manoj Prabhakaran, and Amit Sahai. On the computational complexity of coin flipping. In *51st FOCS*, pages 613–622. IEEE Computer Society Press, October 2010. 5

[MPV24] Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Adaptively sound zero-knowledge SNARKs for UP. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 38–71. Springer, Cham, August 2024. 4

[NWW24] Shafik Nassar, Brent Waters, and David J. Wu. Monotone policy BARGs from BARGs and additively homomorphic encryption. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part II*, volume 15365 of *LNCS*, pages 399–430. Springer, Cham, December 2024. 3

[NWW25] Shafik Nassar, Brent Waters, and David J. Wu. Monotone-policy BARGs and more from BARGs and quadratic residuosity. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part IV*, volume 15677 of *LNCS*, pages 283–313. Springer, Cham, May 2025. 3

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Cham, August 2019. 68

[PV20] Rafael Pass and Muthuramakrishnan Venkitasubramaniam. Is it easier to prove theorems that are guaranteed to be true? In *61st FOCS*, pages 1255–1267. IEEE Computer Society Press, November 2020. 5

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. 27

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 1–20. Springer, Berlin, Heidelberg, February 2004. 5, 7

[RV24] Ron D. Rothblum and Prashant Nalini Vasudevan. Collision resistance from multicollision resistance. *Journal of Cryptology*, 37(2):14, April 2024. 5

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 3–36. Springer, Cham, December 2024. 4

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. 3, 21

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, March 2008. 3

[VV86] L.G. Valiant and V.V. Vazirani. Np is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986. 6

[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, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 433–463. Springer, Cham, August 2022. 3, 30

[WW24] Brent Waters and David J. Wu. Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 387–398. ACM Press, June 2024. 3

[WW25] Brent Waters and David J. Wu. A pure indistinguishability obfuscation approach to adaptively-sound SNARGs for NP. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VII*, volume 16006 of *LNCS*, pages 292–326. Springer, Cham, August 2025. 3

[WZ24] Brent Waters and Mark Zhandry. Adaptive security in SNARGs via iO and lossy functions. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 72–104. Springer, Cham, August 2024. 3

[Zha21] Mark Zhandry. Quantum lightning never strikes the same state twice. or: Quantum money from cryptographic assumptions. *Journal of Cryptology*, 34(1):6, January 2021. 5

# A $\mathcal{EF}$ Proof of Extraction Correctness for SSB Hash

In this section, we recall the construction of *somewhere statistically binding (SSB) hash* from [HW14], and then prove its extraction correctness in $\mathcal{EF}$.

**Ingredient.** Let $\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$ be the leveled fully homomorphic encryption scheme of Lemma 4.6, where $\mathsf{FHE.Setup}(1^\lambda, 1^{\mathsf{dep}})$ generates keys that support evaluating circuits of depth at most $\mathsf{dep}$. We set $\mathsf{dep}$ to be the depth of the circuit $f_{\mathsf{ct}_0,\mathsf{ct}_1}$ defined in $\mathsf{SSB.Hash}$ below, which is the depth of $\mathsf{FHE.Dec}$ plus a constant.

**Construction.** Without loss of generality, we assume that $N = 2^d$ for an integer $d$, and that the binding set is a single index, $S = \{i\}$. For a general set $S$, we repeat the construction for each index in $S$. We construct the algorithms of the SSB hash family $\mathsf{SSB} = (\mathsf{SSB.Gen}, \mathsf{SSB.TGen}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify}, \mathsf{SSB.Ext})$ recursively as follows.

- $\mathsf{SSB.Gen}(1^\lambda, 1^N, 1^1; r)$: Output $\mathsf{hk} = (\mathsf{pk}_0, \ldots, \mathsf{pk}_d, c_1, \ldots, c_d)$, where all $\mathsf{pk}_j$ and $c_j$ are uniformly random strings.
- $\mathsf{SSB.TGen}(1^\lambda, 1^N, i; r)$:
  - If $d = 0$ (i.e., $N = 1$): Run $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{FHE.Setup}(1^\lambda, 1^{\mathsf{dep}}; r)$. Output the hash key $\mathsf{hk} = \mathsf{pk}_0$ and the trapdoor $\mathsf{td} = \mathsf{sk}_0$.
  - If $d \geq 1$: Parse $r = (r_d, \bar{r})$, and write the index in binary as $i = b \| \bar{i}$ with $b \in \{0,1\}$ and $\bar{i} \in \{0,1\}^{d-1}$. Then:
    1. Run $(\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{FHE.Setup}(1^\lambda, 1^{\mathsf{dep}}; r_d)$.
    2. Run $(\bar{\mathsf{hk}}, \mathsf{td}') \leftarrow \mathsf{SSB.TGen}(1^\lambda, 1^{N/2}, \bar{i}; \bar{r})$. By construction, $\mathsf{td}' = \mathsf{sk}_{d-1}$ is the secret key generated at level $d-1$.
    3. Compute $c_d \leftarrow (\mathsf{FHE.Enc}_{\mathsf{pk}_d}(\mathsf{sk}_{d-1}), \mathsf{FHE.Enc}_{\mathsf{pk}_d}(b))$, which encrypts $\mathsf{sk}_{d-1}$ and the first bit $b$ of $i$ under $\mathsf{pk}_d$.
    4. Output the hash key $\mathsf{hk} = (\bar{\mathsf{hk}}, \mathsf{pk}_d, c_d)$ and the trapdoor $\mathsf{td} = \mathsf{sk}_d$.
- $\mathsf{SSB.Hash}(\mathsf{hk}, m)$:
  - If $d = 0$: Output $\mathsf{SSB.Hash}(\mathsf{hk}, m) = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(m; \bar{0})$.
  - If $d \geq 1$: Parse $\mathsf{hk} = (\bar{\mathsf{hk}}, \mathsf{pk}_d, c_d)$, and write $m = m_0 \| m_1$ with $m_0, m_1 \in \{0,1\}^{2^{d-1}}$. Then:
    1. Compute $\mathsf{ct}_0 = \mathsf{SSB.Hash}(\bar{\mathsf{hk}}, m_0)$ and $\mathsf{ct}_1 = \mathsf{SSB.Hash}(\bar{\mathsf{hk}}, m_1)$.
    2. Define the function $f_{\mathsf{ct}_0,\mathsf{ct}_1}(\mathsf{sk}, b) = \mathsf{FHE.Dec}_{\mathsf{sk}}(\mathsf{ct}_b)$, with $\mathsf{ct}_0, \mathsf{ct}_1$ hardwired.
    3. Output $\mathsf{SSB.Hash}(\mathsf{hk}, m_0 \| m_1) = \mathsf{FHE.Eval}_{\mathsf{pk}_d}(f_{\mathsf{ct}_0,\mathsf{ct}_1}, c_d)$.
- $\mathsf{SSB.Open}(\mathsf{hk}, m, i)$:
  - If $d = 0$: Output $\mathsf{SSB.Open}(\mathsf{hk}, m, i) = \perp$.
  - If $d \geq 1$: Parse $\mathsf{hk} = (\bar{\mathsf{hk}}, \mathsf{pk}_d, c_d)$, write $m = m_0 \| m_1$ with $m_0, m_1 \in \{0,1\}^{2^{d-1}}$, and write $i = b \| \bar{i}$ with $b \in \{0,1\}$ and $\bar{i} \in \{0,1\}^{d-1}$. Then:

1. Compute $\mathsf{ct}_0 = \mathsf{SSB.Hash}(\bar{\mathsf{hk}}, m_0)$ and $\mathsf{ct}_1 = \mathsf{SSB.Hash}(\bar{\mathsf{hk}}, m_1)$.
2. Output $\mathsf{SSB.Open}(\mathsf{hk}, m_0\|m_1, b\|\bar{i}) = \big(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{SSB.Open}(\bar{\mathsf{hk}}, m_b, \bar{i})\big)$.

- $\mathsf{SSB.Verify}(\mathsf{hk}, c, v, i, \pi_i)$: Write $i = b\|\bar{i}$ with $b \in \{0,1\}$ and $\bar{i} \in \{0,1\}^{d-1}$. The opening $\pi_i$ consists of $2d$ FHE ciphertexts, two for each level.

  If $d = 0$, check whether $c = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(v; \bar{0})$.

  Otherwise, parse $\pi_i = (\mathsf{ct}_0, \mathsf{ct}_1, \pi_i')$ and $\mathsf{hk} = (\bar{\mathsf{hk}}, \mathsf{pk}_d, c_d)$, and check whether

$$\big(c = \mathsf{FHE.Eval}_{\mathsf{pk}_d}(f_{\mathsf{ct}_0,\mathsf{ct}_1}, c_d)\big) \ \wedge \ \big(\mathsf{SSB.Verify}(\bar{\mathsf{hk}}, \mathsf{ct}_b, v, \bar{i}, \pi_i') = 1\big)\,.$$

- $\mathsf{SSB.Ext}(c, \mathsf{td})$: Output $v = \mathsf{FHE.Dec}_{\mathsf{td}}(c)$.

**Theorem A.1** ($\mathcal{EF}$-Proofs of Extraction Correctness)**.** *There exists a polynomial $L = L(\lambda)$ such that for all integers $d, \lambda \in \mathbb{N}$ with $N = 2^d \le 2^\lambda$, the following formula has an $\mathcal{EF}$ proof of length $L(\lambda)$:*

$$\Big((\mathsf{hk}, \mathsf{td}) = \mathsf{SSB.TGen}(1^\lambda, 1^N, i; r) \ \wedge \ \mathsf{SSB.Verify}(\mathsf{hk}, c, v, i, \pi_i) = 1\Big) \to \mathsf{SSB.Ext}(c, \mathsf{td}) = v,$$

*where* SSB.TGen, SSB.Verify, *and* SSB.Ext *are the circuits constructed above.*

*Proof.* We prove the theorem by induction on $d$.

**Base case.** If $d = 0$, then $N = 1$, and by the construction it suffices to prove

$$\Big((\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{FHE.Setup}(1^\lambda, 1^{\mathsf{dep}}; r) \ \wedge \ c = \mathsf{FHE.Enc}_{\mathsf{pk}_0}(v; \bar{0})\Big) \to \mathsf{FHE.Dec}_{\mathsf{sk}_0}(c) = v.$$

This follows from the $\mathcal{EF}$ proof of FHE correctness in Lemma 4.6.

**Induction step.** Let $d \ge 1$, and write $i = b\|\bar{i}$, where $b$ is a variable and $\bar{i}$ is a vector of $d-1$ variables. Since $\mathsf{SSB.Ext}(c, \mathsf{td}) = \mathsf{FHE.Dec}_{\mathsf{td}}(c)$ has an $\mathcal{EF}$ proof by construction, it suffices to prove

$$\Big((\mathsf{hk}, \mathsf{td}) = \mathsf{SSB.TGen}(1^\lambda, 1^N, i; r) \ \wedge \ \mathsf{SSB.Verify}(\mathsf{hk}, c, v, b\|\bar{i}, \pi_i) = 1\Big) \to \mathsf{FHE.Dec}_{\mathsf{td}}(c) = v. \quad (*)$$

Since $\mathcal{EF}$ proves that SSB.TGen and SSB.Verify unfold according to their recursive definitions, it suffices to prove

$$\begin{aligned}
&\Big((\bar{\mathsf{hk}}, \mathsf{td}') = \mathsf{SSB.TGen}(1^\lambda, 1^{N/2}, \bar{i}; \bar{r}) && (2)\\
&\wedge \ (\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{FHE.Setup}(1^\lambda, 1^{\mathsf{dep}}; r_d) && (3)\\
&\wedge \ c_d = (\mathsf{FHE.Enc}_{\mathsf{pk}_d}(\mathsf{sk}_{d-1}), \mathsf{FHE.Enc}_{\mathsf{pk}_d}(b)) && (4)\\
&\wedge \ c = \mathsf{FHE.Eval}_{\mathsf{pk}_d}(f_{\mathsf{ct}_0,\mathsf{ct}_1}, c_d) && (5)\\
&\wedge \mathsf{SSB.Verify}(\bar{\mathsf{hk}}, \mathsf{ct}_b, v, \bar{i}, \pi_i') = 1\Big) && (6)\\
&\to \mathsf{FHE.Dec}_{\mathsf{td}}(c) = v, && (7)
\end{aligned}$$

where Eqs. (2) to (4) come from the definition of SSB.TGen, and Eqs. (5) and (6) come from the definition of SSB.Verify.

By the induction hypothesis, namely the $\mathcal{EF}$ proof of (*) for $d-1$, the formula

$$\Big((\bar{\mathsf{hk}}, \mathsf{td}') = \mathsf{SSB.TGen}(1^\lambda, 1^{N/2}, \bar{i}; \bar{r}) \ \wedge \ \mathsf{SSB.Verify}(\bar{\mathsf{hk}}, \mathsf{ct}_b, v, \bar{i}, \pi_i') = 1\Big) \to \mathsf{FHE.Dec}_{\mathsf{td}'}(\mathsf{ct}_b) = v$$

has an $\mathcal{EF}$ proof. Since $\mathsf{td}' = \mathsf{sk}_{d-1}$ and $\mathsf{td} = \mathsf{sk}_d$, it therefore suffices to prove

$$\Big((\mathsf{pk}_d, \mathsf{sk}_d) \leftarrow \mathsf{FHE.Setup}(1^\lambda, 1^{\mathsf{dep}}; r_d) \ \wedge \ c_d = (\mathsf{FHE.Enc}_{\mathsf{pk}_d}(\mathsf{sk}_{d-1}), \mathsf{FHE.Enc}_{\mathsf{pk}_d}(b)) \\ \wedge \ \mathsf{FHE.Dec}_{\mathsf{sk}_{d-1}}(\mathsf{ct}_b) = v\Big) \to \mathsf{FHE.Dec}_{\mathsf{sk}_d}(\mathsf{FHE.Eval}_{\mathsf{pk}_d}(f_{\mathsf{ct}_0,\mathsf{ct}_1}, c_d)) = v.$$

This again follows from the $\mathcal{EF}$ proof of FHE correctness in Lemma 4.6, which applies because $f_{\mathsf{ct}_0,\mathsf{ct}_1}$ has depth at most dep.

**Bounding the proof length.** Let $L_d$ denote the length of the $\mathcal{EF}$ proof constructed above for level $d$. The proof for level $d$ consists of the proof for level $d-1$, with its variables renamed, together with the steps of the induction step: unfolding SSB.TGen and SSB.Verify by one level, and applying Lemma 4.6. These steps only involve the keys, ciphertexts, and the circuit $f_{\mathsf{ct}_0,\mathsf{ct}_1}$ introduced at level $d$, so their length is at most $p(\lambda)$ for a polynomial $p$ independent of $d$; the same bound holds for the base case. Hence $L_d \le L_{d-1} + p(\lambda)$, and therefore $L_d \le (d+1)\cdot p(\lambda) \le (\lambda+1)\cdot p(\lambda)$, since $d \le \lambda$. Setting $L(\lambda) = (\lambda+1)\cdot p(\lambda)$ proves the theorem. □

# B $\mathcal{CEF}$ Proof of Correlation Intractability

In this section, we prove the correlation intractability of CIH in $\mathcal{CEF}$. We first recall in Lemma B.3 the hash function of [PS19], which is somewhere perfectly correlation intractable for efficiently searchable relations. We then prove in Lemma B.5 that it is also correlation intractable for efficiently enumerable relations, with a $\mathcal{CEF}$ proof. Finally, we prove in Lemma B.7 that the construction of [HLR21] is correlation intractable for efficiently verifiable product relations, again with a $\mathcal{CEF}$ proof.

**Definition B.1** (Efficiently Searchable Relation)**.** *A relation class $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is* efficiently searchable *if for every integer $\lambda$ and every $R \in \mathcal{R}_\lambda$ over $X_\lambda \times Y_\lambda$, there exists a polynomial-size circuit $C : X_\lambda \to Y_\lambda$ such that $R = \{(x, C(x)) : x \in X_\lambda\}$.*

**Definition B.2** (Somewhere Perfectly Correlation Intractability [PS19])**.** *Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be an efficiently searchable relation class. A hash family* (Gen, Hash)*, together with a fake key generation algorithm* PerfGen*, is* somewhere perfectly correlation intractable *for $\mathcal{R}$ if for every sequence of polynomial-size circuits $\{C_\lambda\}_{\lambda\in\mathbb{N}}$ with $\{(x, C_\lambda(x)) \mid x \in X_\lambda\} \in \mathcal{R}_\lambda$ for every $\lambda$, the following hold:*

- ***Key indistinguishability:*** *The ensembles $\{\mathsf{PerfGen}(1^\lambda, C_\lambda)\}_{\lambda\in\mathbb{N}}$ and $\{\mathsf{Gen}(1^\lambda)\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable.*
- ***Perfect correlation intractability:*** *For every $\lambda \in \mathbb{N}$,*

$$\Pr_{\mathsf{hk}\leftarrow\mathsf{PerfGen}(1^\lambda, C_\lambda)}\big[\exists x \in X_\lambda : \mathsf{Hash}(\mathsf{hk}, x) = C_\lambda(x)\big] = 0.$$

**Lemma B.3** (Somewhere Perfect Correlation Intractability [PS19])**.** *Assuming the hardness of* LWE*, for every efficiently searchable relation class $\mathcal{R}$, there exists a hash family* (Gen, Hash) *with a fake key generation algorithm* PerfGen *that is somewhere perfectly correlation intractable for $\mathcal{R}$.*

**Extending to Enumerable Relations.** Next, we show that the $\mathsf{CIH}$ of Lemma B.3 is automatically correlation intractable for efficiently enumerable relations (Definition B.4). Moreover, this property has a $\mathcal{CEF}$ proof with locality $\ell = \mathsf{poly}(|C_\lambda|, \lambda)$, where $|C_\lambda|$ is the size of the enumeration algorithm, and size $\mathsf{poly}(\log |X_\lambda|, \log |Y_\lambda|, \lambda)$.

**Definition B.4** (Efficiently Enumerable Relation)**.** *A relation class $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ is* efficiently enumerable *if for every integer $\lambda$ and every $R \in \mathcal{R}_\lambda$ over $X_\lambda \times Y_\lambda$, there exists a polynomial-time algorithm $C$ that, on input $x \in X_\lambda$, outputs a list $y_1, \ldots, y_m$ such that, for every $y \in Y_\lambda$, $R(x, y) = 1$ if and only if $y \in \{y_1, \ldots, y_m\}$.*

**Lemma B.5** ($\mathcal{CEF}$ Version of Lemma 3.8 in [CLW18])**.** *Let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda\in\mathbb{N}}$ be an efficiently enumerable relation class over $X_\lambda \times Y_\lambda$, and let $(\mathsf{Gen}, \mathsf{Hash})$, together with $\mathsf{PerfGen}$, be somewhere perfectly correlation intractable for efficiently searchable relations over $X_\lambda \times Y_\lambda$ (Definition B.2). Then $\mathsf{Hash}$ is correlation intractable for $\mathcal{R}$, and this can be formalized in $\mathcal{CEF}$.*

*Formally, there exists a polynomial $\ell$, depending only on the randomness length of $\mathsf{PerfGen}$, such that the following holds. Let $\{R_\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of efficiently verifiable relations with $R_\lambda \in \mathcal{R}_\lambda$, and let $C_\lambda$ be a circuit that efficiently enumerates $R_\lambda$. For every sequence of polynomial-size adversary circuits $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ with outputs in $X_\lambda$, and every sequence of polynomial-length proper variables $\{\mathbf{a}_\lambda\}_{\lambda\in\mathbb{N}}$, the following formula has a $\mathcal{CEF}$ proof of length $\mathsf{poly}(|R_\lambda|, \lambda)$ and locality $\ell(|R_\lambda|, |C_\lambda|, \lambda)$:*

$$(\mathbf{x} \coloneqq \mathcal{A}(\mathbf{a}_\lambda, \mathsf{hk})) \to (R_\lambda(\mathbf{x}, \mathsf{Hash}(\mathsf{hk}, \mathbf{x})) = 0),$$

*where the hash key variable $\mathsf{hk} \coloneqq \mathsf{Gen}(1^\lambda; \mathbf{r})$ is formalized as a vector of extension variables computed from the random variables $\mathbf{r}$.*

*Proof.* For any $\lambda \in \mathbb{N}$, let $R \in \mathcal{R}_\lambda$ be an efficiently enumerable relation, and let $C_\lambda$ be the algorithm that enumerates it. Since $C_\lambda$ runs in polynomial time, it has at most polynomially many outputs, say $m(\lambda)$. For each $i \in [m(\lambda)]$, let $C_i$ be the circuit that runs $C_\lambda$ and outputs its $i$-th output. Then $R_i = \{(x, C_i(x)) \mid x \in X_\lambda\}$ is a searchable relation.

We derive a sequence of intermediate configurations $\Phi_0, \Phi_1, \ldots, \Phi_{m(\lambda)}$, where $\Phi_i$ consists of the definitions $\mathsf{hk} \coloneqq \mathsf{Gen}(1^\lambda; \mathbf{r})$ and $\mathbf{x} \coloneqq \mathcal{A}(\mathbf{a}_\lambda, \mathsf{hk})$, together with the formula

$$\bigwedge_{j\le i} \big(\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \neq C_j(\mathbf{x})\big).$$

The configuration $\Phi_0$ only contains the two definitions, which contains a set of Extension rules. To go from $\Phi_{i-1}$ to $\Phi_i$, we use the following three *Cryptographic Gadget Substitution* steps.

1. Replace $\mathsf{hk} \coloneqq \mathsf{Gen}(1^\lambda; \mathbf{r})$ by $\mathsf{hk} \coloneqq \mathsf{PerfGen}(1^\lambda, C_i; \mathbf{r}')$, where $\mathbf{r}'$ are fresh random variables. This gadget has output $\mathsf{hk}$, no input, and internal randomness $\mathbf{r}$ (respectively $\mathbf{r}'$), so this is the Distribution Shift rule. This replacement is allowed by the key indistinguishability in Definition B.2.

2. Introduce an extension variable $b_i \coloneqq f_i(\mathbf{r}', \mathbf{x})$, where $f_i$ is the following gadget

$$f_i(\mathbf{r}', \mathbf{x}) = \begin{cases} 1 & \text{if } \mathsf{Hash}(\mathsf{PerfGen}(1^\lambda, C_i; \mathbf{r}'), \mathbf{x}) = C_i(\mathbf{x}), \\ 0 & \text{otherwise.} \end{cases}$$

Derive $b_i \leftrightarrow \big(\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) = C_i(\mathbf{x})\big)$ by $\mathcal{EF}$ rules, and erase the intermediate formulas involving the wires of $f_i$. This gadget has input $(\mathbf{r}', \mathbf{x})$, output $b_i$, and no internal randomness. By the perfect correlation intractability in Definition B.2, $f_i(\mathbf{r}', \mathbf{x}) = 0$ for every value of $(\mathbf{r}', \mathbf{x})$. Hence, the Cryptographic Gadget Substitution rule, with $\epsilon = 0$, replaces $b_i := f_i(\mathbf{r}', \mathbf{x})$ by $b_i := 0$. We then derive $\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \neq C_i(\mathbf{x})$ and erase $b_i$.

3. Replace $\mathsf{hk} := \mathsf{PerfGen}(1^\lambda, C_i; \mathbf{r}')$ back by $\mathsf{hk} := \mathsf{Gen}(1^\lambda; \mathbf{r})$, again by the Distribution Shift rule. The formula $\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \neq C_i(\mathbf{x})$ stays in the configuration, and together with $\Phi_{i-1}$ it gives $\Phi_i$.

Finally, we derive $R_\lambda(\mathbf{x}, \mathsf{Hash}(\mathsf{hk}, \mathbf{x})) = 0$ from $\Phi_{m(\lambda)}$. This step uses the completeness of the enumeration in Definition B.4. Let $\mathbf{y}$ be the extension variables that compute $\mathsf{Hash}(\mathsf{hk}, \mathbf{x})$, and introduce an extension variable $b := f(\mathbf{x}, \mathbf{y})$, where the gadget $f$ outputs 1 if and only if

$$R_\lambda(\mathbf{x}, \mathbf{y}) \ \wedge \bigwedge_{i \le m(\lambda)} \big(\mathbf{y} \neq C_i(\mathbf{x})\big).$$

This gadget has input $(\mathbf{x}, \mathbf{y})$ of length $\log |X_\lambda| + \log |Y_\lambda| \le |R_\lambda|$, output $b$, and no internal randomness. By Definition B.4, $f(\mathbf{x}, \mathbf{y}) = 0$ for every value of $(\mathbf{x}, \mathbf{y})$, so the Cryptographic Gadget Substitution rule replaces $b := f(\mathbf{x}, \mathbf{y})$ by $b := 0$. Together with $\Phi_{m(\lambda)}$, $\mathcal{EF}$ rules then give $R_\lambda(\mathbf{x}, \mathsf{Hash}(\mathsf{hk}, \mathbf{x})) = 0$.

The gadget substitutions in the second step have input length $|\mathbf{r}'| + \log |X_\lambda|$, where $|\mathbf{r}'|$ grows with $|C_i| = \mathsf{poly}(|C_\lambda|)$, and the last one has input length $\log |X_\lambda| + \log |Y_\lambda|$. Both are $\mathsf{poly}(|C_\lambda|, \lambda)$, since $C_\lambda$ reads $x \in X_\lambda$ and outputs elements of $Y_\lambda$. All of them have output length 1, which determines the locality $\ell(|R_\lambda|, |C_\lambda|, \lambda)$; this polynomial depends only on the randomness length of $\mathsf{PerfGen}$. The circuits $f_i$ and $f$ have size $\mathsf{poly}(|R_\lambda|, |C_\lambda|, \lambda)$, and there are $O(m(\lambda))$ steps in total. □

**Definition B.6** (Efficiently Verifiable Product Relation [HLR21])**.** *Let $X, Y$ be two sets and $t$ be an integer. A relation $R \subseteq X \times Y^t$ is a* product relation *if for every $x$, the set $R_x = \{y : (x, y) \in R\} \subseteq Y^t$ can be written as*

$$R_x = S_1 \times S_2 \times \cdots \times S_t.$$

*A product relation $R$ is* efficiently verifiable *if there is a polynomial-size circuit $V$ such that for every input $x$, with the decomposition $R_x = S_1 \times S_2 \times \cdots \times S_t$, we have*

$$S_i = \{y : V(x, y, i) = 1\}.$$

**Lemma B.7** ($\mathcal{CEF}$-Proof of Correlation Intractability of [HLR21])**.** *There exists a polynomial $\ell$ such that the following holds. Let $\{\rho_\lambda\}_{\lambda \in \mathbb{N}}$ be a sequence of reals in $(0, 1)$, let $\{t_\lambda\}_{\lambda \in \mathbb{N}}$ be a sequence of integers with $\lambda / \log(1/\rho_\lambda) \le t_\lambda \le \mathsf{poly}(\lambda)$, and let $\mathcal{R} = \{\mathcal{R}_\lambda\}_{\lambda \in \mathbb{N}}$ be a class of efficiently verifiable product relations over $X_\lambda \times Y_\lambda^{t_\lambda}$ (Definition B.6) with $|Y_\lambda| \le \mathsf{poly}(\lambda)$, such that for every $R \in \mathcal{R}_\lambda$ and every $x \in X_\lambda$, the sets $S_1, \ldots, S_{t_\lambda}$ in the decomposition of $R_x$ have size at most $\rho_\lambda |Y_\lambda|$.*

*Assuming the hardness of* LWE*, there exists a hash family $(\mathsf{Gen}, \mathsf{Hash})$ with outputs in $Y_\lambda^{t_\lambda}$ whose correlation intractability for $\mathcal{R}$ has a polynomial-size $\mathcal{CEF}$ proof, in the following sense. For every sequence of relations $\{R_\lambda\}_{\lambda \in \mathbb{N}}$ with $R_\lambda \in \mathcal{R}_\lambda$, every sequence of polynomial-size adversary circuits $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ with outputs in $X_\lambda$, and every sequence of polynomial-length proper variables $\{\mathbf{a}_\lambda\}_{\lambda \in \mathbb{N}}$, the following formula has a $\mathcal{CEF}$ proof of length $\mathsf{poly}(|R_\lambda|, \lambda)$ and locality $\ell(|R_\lambda|, |Y_\lambda|, \lambda)$:*

$$(\mathbf{x} := \mathcal{A}(\mathbf{a}_\lambda, \mathsf{hk})) \to \big(R_\lambda(\mathbf{x}, \mathsf{Hash}(\mathsf{hk}, \mathbf{x})) = 0\big),$$

*where the hash key variable* $\mathsf{hk} := \mathsf{Gen}(1^\lambda; \mathbf{r})$ *is formalized as a vector of extension variables computed from the random variables* $\mathbf{r}$.

*Proof.* For any fixed $\lambda \in \mathbb{N}$, write $t = t_\lambda$ and $R = R_\lambda$, and let $V$ be the circuit that verifies $R$ (Definition B.6). For $x \in X_\lambda$, write $R_x = S_1(x) \times \cdots \times S_t(x)$, where $S_i(x) = \{y \in Y_\lambda : V(x, y, i) = 1\}$.

**Constructing the hash family.** Following [HLR21], the hash function first hashes $x$ with the hash family of Lemma B.3 and then encodes the hash value with a list-recoverable code. Recall that a code $C : Z \to Y^t$ is *list-recoverable from lists of size* $\rho_\lambda|Y|$ if there is an algorithm that, given sets $T_1, \ldots, T_t \subseteq Y$ with $|T_i| \le \rho_\lambda |Y|$ for every $i \in [t]$, runs in time $\mathsf{poly}(t, |Y|, \log |Z|)$ and outputs the list of all $z \in Z$ such that $C(z) \in T_1 \times \cdots \times T_t$. The bounds $|Y_\lambda| \le \mathsf{poly}(\lambda)$ and $\lambda / \log(1/\rho_\lambda) \le t_\lambda \le \mathsf{poly}(\lambda)$ are those of [HLR21, Theorem 5.1]. Under these bounds, [HLR21, Lemma 5.4] gives an ensemble of codes $C_\lambda : Z_\lambda \to Y_\lambda^t$, where $Z_\lambda = \{0,1\}^{k_\lambda}$ for some $k_\lambda = \lambda^{\Omega(1)}$, such that a random code in the ensemble is list-recoverable from lists of size $\rho_\lambda |Y_\lambda|$, except with probability $2^{-\Omega(k_\lambda)}$. The key generation samples a code $C_\lambda$ from this ensemble and includes it in the hash key. The proof below uses the list recoverability of $C_\lambda$ only through the following formula on lists $T_1, \ldots, T_t$ and $z \in Z_\lambda$: if $|T_i| \le \rho_\lambda |Y_\lambda|$ for every $i \in [t]$ and $C_\lambda(z) \in T_1 \times \cdots \times T_t$, then the list-recovery algorithm outputs $z$ on input $(T_1, \ldots, T_t)$. Except with probability $2^{-\Omega(k_\lambda)}$ over the choice of $C_\lambda$, this formula holds for all values of the lists and $z$, which consist of $\mathsf{poly}(\lambda)$ bits, so it can be derived in $\mathcal{CEF}$ with locality $\mathsf{poly}(\lambda)$.

Let $(\mathsf{Gen}, \mathsf{Hash}')$, together with $\mathsf{PerfGen}$, be the hash family of Lemma B.3 with outputs in $Z_\lambda$, for the efficiently searchable relations over $X_\lambda \times Z_\lambda$ that the proof of Lemma B.5 uses for the enumerator $E$ below. Our hash family is $(\mathsf{Gen}, \mathsf{Hash})$ with

$$\mathsf{Hash}(\mathsf{hk}, x) = C_\lambda(\mathsf{Hash}'(\mathsf{hk}, x)).$$

**Reducing to an enumerable relation.** We reduce to Lemma B.5 via the relation

$$R' = \{(x, z) \in X_\lambda \times Z_\lambda : R(x, C_\lambda(z)) = 1\},$$

which is computed by the circuit $R'(x, z) = R(x, C_\lambda(z))$ of size $|R| + \mathsf{poly}(\lambda)$. The following algorithm $E$ enumerates $R'$ in the sense of Definition B.4. On input $x \in X_\lambda$:

1. For each $i \in [t]$, compute $S_i(x)$ by evaluating $V(x, y, i)$ on every $y \in Y_\lambda$.
2. Run the list-recovery algorithm of $C_\lambda$ on $(S_1(x), \ldots, S_t(x))$, and output the resulting list $z_1, \ldots, z_m$.

Since $|S_i(x)| \le \rho_\lambda |Y_\lambda|$ for every $i$, the list-recovery algorithm outputs exactly the $z \in Z_\lambda$ with $C_\lambda(z) \in S_1(x) \times \cdots \times S_t(x) = R_x$, that is, exactly the $z$ with $R'(x, z) = 1$. The first step takes time $t \cdot |Y_\lambda| \cdot |V|$, and the second takes time $\mathsf{poly}(t, |Y_\lambda|, k_\lambda)$. Since $t$, $|Y_\lambda|$, and $k_\lambda$ are at most $\mathsf{poly}(\lambda)$, $E$ has size $\mathsf{poly}(|R|, \lambda)$.

**Applying Lemma B.5.** The relations $R'_\lambda$ form an efficiently enumerable relation class over $X_\lambda \times Z_\lambda$, and $(\mathsf{Gen}, \mathsf{Hash}')$, together with $\mathsf{PerfGen}$, is somewhere perfectly correlation intractable for the efficiently searchable relations that the proof of Lemma B.5 uses. Applying Lemma B.5 to $\{R'_\lambda\}_{\lambda \in \mathbb{N}}$

with the enumerators $E = E_\lambda$, the adversaries $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, and the proper variables $\{\mathbf{a}_\lambda\}_{\lambda\in\mathbb{N}}$, we obtain a $\mathcal{CEF}$ proof of

$$(\mathbf{x} \coloneqq \mathcal{A}(\mathbf{a}_\lambda, \mathsf{hk})) \to \big(R'_\lambda(\mathbf{x}, \mathsf{Hash}'(\mathsf{hk}, \mathbf{x})) = 0\big)$$

with length and locality $\mathsf{poly}(|R'_\lambda|, |E_\lambda|, \lambda)$. Since $t \le |R_\lambda|$, both $|R'_\lambda|$ and $|E_\lambda|$ are at most $\mathsf{poly}(|R_\lambda|, |Y_\lambda|)$ for a polynomial that does not depend on the relations, so the locality is at most $\ell(|R_\lambda|, |Y_\lambda|, \lambda)$ for such a polynomial $\ell$, and the length is $\mathsf{poly}(|R_\lambda|, \lambda)$. By the definitions of $R'$ and $\mathsf{Hash}$, the formula $R'_\lambda(\mathbf{x}, \mathsf{Hash}'(\mathsf{hk}, \mathbf{x})) = 0$ unfolds to $R_\lambda(\mathbf{x}, \mathsf{Hash}(\mathsf{hk}, \mathbf{x})) = 0$ by $\mathcal{EF}$ rules, which proves the lemma.

□

**Remark B.8** (Relations with an Auxiliary Input)**.** *Lemmas B.5 and B.7 extend to relations that also read an auxiliary input sampled independently of the hash key, such as a trapdoor. Let* $\mathbf{w} \coloneqq W_\lambda(\mathbf{r_w})$ *be a vector of extension variables computed by a polynomial-size circuit* $W_\lambda$ *from random variables* $\mathbf{r_w}$ *disjoint from* $\mathbf{r}$*, and let* $R_\lambda(w, x, y)$ *be a relation such that* $R_\lambda(w, \cdot, \cdot) \in \mathcal{R}_\lambda$ *for every value of* $w$*. Then, for adversaries that also read* $\mathbf{w}$*, the formula*

$$(\mathbf{x} \coloneqq \mathcal{A}(\mathbf{a}_\lambda, \mathbf{w}, \mathsf{hk})) \to \big(R_\lambda(\mathbf{w}, \mathbf{x}, \mathsf{Hash}(\mathsf{hk}, \mathbf{x})) = 0\big)$$

*has a* $\mathcal{CEF}$ *proof with the same length and locality bounds. The proofs are the same, except for two changes. First, each Distribution Shift step in the proof of Lemma B.5 replaces the definitions* $(\mathbf{w}, \mathsf{hk}) \coloneqq (W_\lambda(\mathbf{r_w}), \mathsf{Gen}(1^\lambda; \mathbf{r}))$ *by* $(\mathbf{w}, \mathsf{hk}) \coloneqq (W_\lambda(\mathbf{r_w}), \mathsf{PerfGen}(1^\lambda, C_i(\mathbf{w}, \cdot); \mathbf{r'}))$ *together. These two distributions are indistinguishable: a distinguisher would, for some fixed value of* $w$*, distinguish* $\mathsf{Gen}(1^\lambda)$ *from* $\mathsf{PerfGen}(1^\lambda, C_i(w, \cdot))$*, contradicting the key indistinguishability in Definition B.2. Second, the gadgets* $f_i$ *and* $f$ *also take* $\mathbf{w}$ *as input, which adds* $|\mathbf{w}| \le |R_\lambda|$ *to their input length.*

# C $\mathcal{CEF}$ Proof of Extraction Correctness for BARG

In this section, we first recall the BARG construction of [CJJ22] for BatchIndexTMSAT (Definition 4.8) and prove its somewhere extraction correctness in $\mathcal{CEF}$ in Theorem C.1.

**Ingredients.** The construction uses the following ingredients.

- **SSB hash.** The SSB hash family SSB of Theorem A.1, with algorithms SSB.TGen, SSB.Hash, SSB.Open, SSB.Verify, and SSB.Ext, which hashes a string in $\{0,1\}^N$ and is binding at one index. Its extraction correctness has an $\mathcal{EF}$ proof (Theorem A.1).
- **PCP.** The PCP (PCP.P, PCP.Q, PCP.V, PCP.E) of Theorem 4.1 for circuit satisfiability. For a circuit $C$, we use the following properties.
  - *Randomness.* The query algorithm $\mathsf{PCP.Q}(C, y)$ uses randomness $y \in Y_C = \{0,1\}^{O(\log|C|)}$, so $|Y_C| \le \mathsf{poly}(|C|)$.
  - *Knowledge soundness.* For every string $\pi^*$, if PCP.V accepts $\pi^*$ with probability more than $\rho(|C|) = 1 - 1/\mathrm{polylog}(|C|)$ over $y$, then $C(x, \mathsf{PCP.E}(C, x, \pi^*)) = 1$.
  - *Fast online verification.* The query set has size $\mathrm{polylog}(|C|)$, the state st has length $\mathsf{poly}(|x|, \log|C|)$, and $\mathsf{PCP.V}(x, \mathsf{st}, \pi')$ runs in time $\mathsf{poly}(|x|, \log|C|)$.

- **CI hash.** The hash family $(\mathsf{CIH.Gen}, \mathsf{CIH.Hash})$ of Lemma B.7, together with Remark B.8, for the product relations defined in the proof of Theorem C.1, with $t = \lceil \lambda/(1-\rho(|C|)) \rceil$ coordinates, alphabet $Y_C$, and sparsity $\rho(|C|)$. The hypotheses of Lemma B.7 hold since $|Y_C| \le \mathsf{poly}(\lambda)$, $\log(1/\rho(|C|)) \ge 1 - \rho(|C|)$, and $t \le \lambda \cdot \mathrm{polylog}(|C|)$.

**Construction.** We recall the BARG of [CJJ22], which is recursive in $d = \lceil \log k \rceil$; if $k$ is not a power of two, we pad the batch to $2^d$ instances, letting every index $i > k$ check the $k$-th instance. For an instance $x = (M, z, k, T)$, let $C = C_{M,z,T}$ be the circuit that, on instance $i$ and witness $w$, checks whether $M(z, i, w)$ accepts within $T$ steps; we then write $M(z, i, w) = 1$. When $M$ is a circuit, as in Theorem C.1, $C$ is simply $M$ with its first input fixed to $z$, so $|C| \le |M|$. As in [CJJ22], the shared input $z$ is part of the circuit, so the PCP instance is only the index $i$ and the PCP states are short. If $d = 0$, the proof is the witness itself, the verifier runs $M$ on it, and the extractor outputs the proof. For $d \ge 1$, the algorithms are as follows.

- $\mathsf{BARG.Gen}(1^\lambda, 1^n, 1^m, 1^T, k, i^*)$:
  - Sample an SSB key $\mathsf{hk_{SSB}}$ binding at index $i^*$ with trapdoor $\mathsf{td_{SSB}}$, and a CI hash key $\mathsf{hk_{CIH}}$.
  - Recursively sample a key $(\mathsf{crs}', \mathsf{td}')$ for $k/2$ instances binding at index $j^* = \lceil i^*/2 \rceil$.
  - Output $\mathsf{crs} = (\mathsf{hk_{SSB}}, \mathsf{hk_{CIH}}, \mathsf{crs}')$ and $\mathsf{td} = (i^*, \mathsf{td_{SSB}}, \mathsf{td}')$.
- BARG.P:
  - Compute PCPs for the $k$ instances, write them as the rows of a matrix, and SSB hash each column $q$ to $c_q$ with $\mathsf{hk_{SSB}}$.
  - Apply the CI hash to $x$ and $c = (c_q)_q$ to get $t$ PCP randomness strings $(y_1, \ldots, y_t)$, and hence query sets and states $(Q_u, \mathsf{st}_u) = \mathsf{PCP.Q}(C, y_u)$ shared by all $k$ instances.
  - Prove, with the BARG for $k/2$ instances under $\mathsf{crs}'$, that every pair of rows $2j-1$ and $2j$ passes the PCP verification and the SSB local opening verification.
    - **Instance.** The instance $z'$ of this BARG consists of $\mathsf{hk_{SSB}}$, the SSB hash value of the queried columns in $Q_u$, and the states $(Q_u, \mathsf{st}_u)_{u \in [t]}$.
    - **Witness.** The witness for index $j$ consists of the queried bits of rows $2j-1$ and $2j$ with their SSB openings, and the machine $M'$ accepts if the openings verify and PCP.V accepts the queried bits of both rows for every $u$.

    Let $\Pi'$ be the resulting proof.
  - Output $\Pi = (c, \Pi')$.
- $\mathsf{BARG.V}(\mathsf{crs}, x, \Pi)$:
  - Parse $\Pi = (c, \Pi')$, and recompute $(Q_u, \mathsf{st}_u)_{u \in [t]}$ and $z'$ from $(x, c)$.
  - Run the verifier for $k/2$ instances on $\mathsf{crs}'$, the instance $x' = (M', z', k/2, T')$, and $\Pi'$, where $T'$ is the running time of $M'$.
- $\mathsf{BARG.Ext}(\mathsf{td}, x, \Pi)$:
  - Extract the row $\pi^*$ at index $i^*$, whose $q$-th bit is $\mathsf{SSB.Ext}(c_q, \mathsf{td_{SSB}})$.

– Output $\mathsf{PCP.E}(C, i^*, \pi^*)$.

**Theorem C.1** ($\mathcal{CEF}$ Proof of Extraction Correctness)**.** *There exists a polynomial $\ell$ such that the following holds. For any sequence of polynomial-size circuits $\{M_\lambda\}_{\lambda\in\mathbb{N}}$, any polynomials $n = n(\lambda)$, $m = m(\lambda), T = T(\lambda), k = k(\lambda)$, every sequence of indices $i^*_\lambda \in [k]$, every sequence of polynomial-size adversary circuits $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$, and every sequence of polynomial-length proper variables $\{\mathbf{a}_\lambda\}_{\lambda\in\mathbb{N}}$, the following formula has a $\mathcal{CEF}$ proof of polynomial length and locality $\ell(|M_\lambda|, n, m, \log k, \lambda)$:*

$$\Big((z, \Pi) \coloneqq \mathcal{A}_\lambda(\mathbf{a}_\lambda, \mathsf{crs})\Big) \to \Big(\mathsf{BARG.V}(\mathsf{crs}, x, \Pi) = 1 \to M_\lambda\big(z, i^*, \mathsf{BARG.Ext}(\mathsf{td}, x, \Pi)\big) = 1\Big),$$

*where $x = (M_\lambda, z, k, T)$, and $(\mathsf{crs}, \mathsf{td}) \coloneqq \mathsf{BARG.Gen}(1^\lambda, 1^n, 1^m, 1^T, k, i^*; \mathbf{r})$ is a vector of extension variables computed from the random variables $\mathbf{r}$.*

*In this formula, the circuit $M_\lambda$, the numbers $n$, $m$, $T$, $k$, and $i^*$, and the circuit $\mathcal{A}_\lambda$ are fixed for each $\lambda$ and hardwired into the formula. The only proper variables are $\mathbf{a}_\lambda$, the only random variables are $\mathbf{r}$, and $\mathsf{crs}$, $\mathsf{td}$, $z$, and $\Pi$ are extension variables. In particular, the only variable part of the instance $x$ is $z$, which the adversary outputs.*

*Proof Sketch.* We construct the $\mathcal{CEF}$ proof by induction on $d = \lceil \log k \rceil$, for all sequences of circuits, lengths, running times, and batch sizes at once, since the induction step applies the induction hypothesis to the circuit $M'$ of the construction. We bound its locality and length at the end. Throughout, we write $M = M_\lambda$, $x = (M, z, k, T)$, and $C = C_{M,z,T}$, which is $M$ with its first input fixed to $z$.

**Base case.** If $d = 0$, the verifier checks that $M(z, 1, \Pi) = 1$ and the extractor outputs $\Pi$, so the formula has an $\mathcal{EF}$ proof that unfolds these definitions.

**Applying the induction hypothesis.** Let $d \geq 1$, and parse $\mathsf{crs} = (\mathsf{hk}_{\mathsf{SSB}}, \mathsf{hk}_{\mathsf{CIH}}, \mathsf{crs}')$ and $\Pi = (c, \Pi')$. Unfolding the verifier, $\mathsf{BARG.V}(\mathsf{crs}, x, \Pi) = 1$ says that the verifier for $k/2$ instances accepts $\Pi'$ for the instance $x' = (M', z', k/2, T')$ under $\mathsf{crs}'$. We apply the induction hypothesis to this $\mathsf{BARG}$, with the index $j^* = \lceil i^*/2 \rceil$, the proper variables $(\mathbf{a}_\lambda, \mathsf{hk}_{\mathsf{SSB}}, \mathsf{hk}_{\mathsf{CIH}})$, and the adversary that runs $\mathcal{A}_\lambda$ and outputs $(z', \Pi')$. Then we obtain the following formula:

$$\Big((z', \Pi') \coloneqq \mathcal{A}'(\mathbf{a}', \mathsf{crs}')\Big) \to \Big(\mathsf{BARG.V}(\mathsf{crs}', x', \Pi') = 1 \to M'\big(z', j^*, \mathsf{BARG.Ext}(\mathsf{td}', x', \Pi')\big) = 1\Big),$$

where $\mathcal{A}'$ is this adversary, $\mathbf{a}' = (\mathbf{a}_\lambda, \mathsf{hk}_{\mathsf{SSB}}, \mathsf{hk}_{\mathsf{CIH}})$, $n'$ and $m'$ are the instance and witness lengths of $M'$, and $(\mathsf{crs}', \mathsf{td}') \coloneqq \mathsf{BARG.Gen}(1^\lambda, 1^{n'}, 1^{m'}, 1^{T'}, k/2, j^*; \mathbf{r}')$.

By Variable Substitution (Lemma 5.7), we then replace $\mathsf{hk}_{\mathsf{SSB}}$ and $\mathsf{hk}_{\mathsf{CIH}}$ with their definitions, which is allowed since their randomness is disjoint from that of $\mathsf{crs}'$. We obtain that $M'$ accepts the witness extracted from $\Pi'$ at index $j^*$.

**Extracting the queried entries.** Since $i^* \in \{2j^* - 1, 2j^*\}$, the witness extracted from $\Pi'$ contains the queried bits of row $i^*$ with their SSB openings, and $M'$ checks that the openings verify and that $\mathsf{PCP.V}$ accepts these bits on every query set $Q_u$. By the extraction correctness of $\mathsf{SSB}$ (Theorem A.1), each opened bit equals the corresponding bit of the row $\pi^*$ extracted from $c$ with $\mathsf{td}_{\mathsf{SSB}}$. Hence $\mathsf{PCP.V}$ accepts $\pi^*$ on all $t$ query sets.

**Applying correlation intractability.** It remains to show that $\mathsf{PCP.E}$ extracts a valid witness from $\pi^*$, which is the conclusion of the theorem by the definition of $\mathsf{BARG.Ext}$. Consider the following relation $R$:

$$R\big(\mathsf{td}_{\mathsf{SSB}}, (x, c), (y_1, \ldots, y_t)\big) \coloneqq \left[ C\big(i^*, \mathsf{PCP.E}(C, i^*, \pi^*)\big) = 0 \ \wedge \bigwedge_{u \in [t]} \mathsf{PCP.V}\big(i^*, \mathsf{st}_u, \pi^*|_{Q_u}\big) = 1 \right],$$

where $C = C_{M,z,T}$ is computed from $x$, $\pi^*$ is extracted from $c = (c_q)_q$ via $\mathsf{SSB.Ext}$, and $(Q_u, \mathsf{st}_u) = \mathsf{PCP.Q}(C, y_u)$.

For fixed $\mathsf{td}_{\mathsf{SSB}}$ and $(x, c)$, the outputs $(y_1, \ldots, y_t)$ satisfying $R$ form a product set $S^t$, where $S$ is the set of PCP randomness on which $\mathsf{PCP.V}$ accepts $\pi^*$ if $\mathsf{PCP.E}$ fails, and $S = \varnothing$ otherwise. By knowledge soundness, $|S| \leq \rho(|C|) \cdot |Y_C|$, so $R$ is a product relation of the kind handled by Lemma B.7. We apply Lemma B.7 together with Remark B.8, where the auxiliary input consists of the SSB key and trapdoor and the key and trapdoor for $k/2$ instances, whose randomness is disjoint from that of $\mathsf{hk}_{\mathsf{CIH}}$. This gives a $\mathcal{CEF}$ proof that $R$ does not hold on the CI hash output $(y_1, \ldots, y_t)$, which together with the previous paragraph shows that $\mathsf{PCP.E}$ succeeds.

**Bounding the locality, length, and error.** Only the applications of Lemma B.7, one per level, constrain the locality, which is polynomial in $\lambda$, $|Y_C| \leq \mathsf{poly}(|C|)$, and the size of the relation $R$ at that level, for a polynomial that does not depend on the level. At the top level, $R$ has size $\mathsf{poly}(\lambda, |M|, \log k)$, since $|C| \leq |M|$ and $R$ extracts each bit of $\pi^*$ from an SSB hash of $k$ bits and then runs $\mathsf{PCP.E}$, $C$, and $\mathsf{PCP.V}$. By the fast online verification of the PCP, the circuit at the next level has size $\mathsf{poly}(\lambda, \log k, \log |C|)$, so the circuits at all lower levels have size $\mathsf{poly}(\lambda, \log k, \log |M|)$. Hence the locality at every level is at most $\ell(|M|, n, m, \log k, \lambda)$ for a polynomial $\ell$ that does not depend on $M$, $n$, $m$, $T$, or $k$. Each induction step adds a polynomial in $\lambda$ to the size of the adversary and to the length of the proof, and there are $d = \lceil \log k \rceil$ levels, so the total length is polynomial. The error comes only from the $d$ applications of Lemma B.7, so it is negligible.

□