SNARGs under LWE via Propositional Proofs

Zhengzhong Jin          Yael Tauman Kalai          Alex Lombardi
Northeastern and MIT     Microsoft Research and MIT    Princeton
Vinod Vaikuntanathan    MIT

June 14, 2024

Abstract

We construct a succinct non-interactive argument (SNARG) system for every NP language \( \mathcal{L} \) that has a propositional proof of non-membership for each \( x \notin \mathcal{L} \). The soundness of our SNARG system relies on the hardness of the learning with errors (LWE) problem. The common reference string (CRS) in our construction grows with the space required to verify the propositional proof, and the size of the proof grows poly-logarithmically in the length of the propositional proof.

Unlike most of the literature on SNARGs, our result implies SNARGs for languages \( \mathcal{L} \) with proof length shorter than logarithmic in the deterministic time complexity of \( \mathcal{L} \). Our SNARG improves over prior SNARGs for such “hard” NP languages (Sahai and Waters, STOC 2014, Jain and Jin, FOCS 2022) in several ways:

- For languages with polynomial-length propositional proofs of non-membership, our SNARGs are based on a single, polynomial-time falsifiable assumption, namely LWE.
- Our construction handles propositional proofs of super-polynomial length, as long as they have bounded space, under the subexponential LWE assumption.
- Our SNARGs have a transparent setup, meaning that no private randomness is required to generate the CRS.

Moreover, our approach departs dramatically from these prior works: we show how to design SNARGs for hard languages without publishing a program (in the CRS) that has the power to verify NP witnesses.

The key new idea in our cryptographic construction is what we call a “locally unsatisfiable extension” of the NP verification circuit \( \{C_x\}_x \). We say that an NP verifier has a locally unsatisfiable extension if for every \( x \notin \mathcal{L} \), there exists an extension \( E_x \) of \( C_x \) that is not even locally satisfiable in the sense of a local assignment generator [Paneth-Rothblum, TCC 2017]. Crucially, we allow \( E_x \) to be depend arbitrarily on \( x \) rather than being efficiently constructible.

In this work, we show – via a “hash-and-BARG” for a hidden, encrypted computation – how to build SNARGs for all languages with locally unsatisfiable extensions. We additionally show that propositional proofs of unsatisfiability generically imply the existence of locally unsatisfiable extensions, which allows us to deduce our main results.

As an illustrative example, our results imply a SNARG for the decisional Diffie-Hellman (DDH) language under the LWE assumption.
# Contents

1 Introduction
   1.1 Our Results ................................................................. 2
   1.2 Related Work .............................................................. 3
   1.3 Subsequent Work ......................................................... 5
   1.4 Paper Organization ....................................................... 5

2 Technical Overview .......................................................... 5
   2.1 Handling extensions of super-polynomial size ....................... 7
   2.2 Extended Frege proofs and local unsatisfiability .................. 9

3 Preliminaries ........................................................................ 14
   3.1 Boolean Circuits .......................................................... 15
   3.2 Hash Family with Local Opening ...................................... 16
   3.3 Fully Homomorphic Encryption ....................................... 17
   3.4 Succinct Non-Interactive Arguments ................................. 18
   3.5 Somewhere Extractable Batch Arguments (seBARGs) ............. 19
   3.6 Propositional Logic Systems ........................................... 21
   3.7 Cook’s Theory $PV$ ....................................................... 22
   3.8 Theory $PV_1$ ............................................................... 24

4 SNARGs from Locally Unsatisfiable Circuit Extensions ............... 25
   4.1 Local Unsatisfiability ..................................................... 25
   4.2 Encoded Computation .................................................... 27
   4.3 Main Theorem Statement ............................................... 30
   4.4 SNARG Construction: Encrypted Hash-and-BARG ................. 31
   4.5 Analysis ................................................................. 33
   4.6 Subsequent work: a strengthening of Theorem 4.8 ................. 39

5 SNARGs from Locally Unsatisfiable Extensions of Super-polynomial Size 40
   5.1 Theorem Statement ....................................................... 40
   5.2 SNARG Construction .................................................... 41
   5.3 Analysis ................................................................. 45
   5.4 Subsequent work: a strengthening of Theorem 5.2 ................. 48
   5.5 Limitations of locally unsatisfiable extensions .................... 49

6 Locally Unsatisfiable Extensions from Propositional Proofs of Unsatisfiability 50
   6.1 Binary AND-Tree Circuits ............................................. 51
   6.2 Construction of the Extension Circuit ............................... 54
   6.3 Proof of Local Unsatisfiability ....................................... 56

7 Locally Unsatisfiable Extensions from Bounded Space Propositional Proofs of Unsatisfiability 58
   7.1 Space Complexity of Propositional Proofs .......................... 58
   7.2 Locally Unsatisfiable Extensions from Bounded Space Proofs of Unsatisfiability ... 59
8 Applications
  8.1 SNARGs for the DDH Language

9 Acknowledgements
1 Introduction

Succinct non-interactive arguments (SNARGs) are a powerful cryptographic primitive whose feasibility is still poorly understood. Informally, a SNARG for an NP language L is a computationally sound non-interactive argument system for L whose proofs are short, much shorter than the length of an NP witness, and easy to verify. To achieve this, the prover and the verifier are given access to a common reference string crs (also called a structured random string), and this string is used both to produce the SNARG proofs and to verify them.

In the random oracle model, it is known that there are SNARGs for every NP language [Kil92, Mic94]. However, constructing SNARGs for NP in the “plain model” under falsifiable and preferably standard cryptographic assumptions remains a widely open problem. Indeed, Gentry and Wichs [GW11] formalized a serious barrier to constructing SNARGs in the case where the cheating prover is allowed to choose the statement x adaptively based on the crs; such a soundness guarantee is known as adaptive soundness.

Faced with the Gentry-Wichs barrier, we focus on the problem of constructing SNARGs with non-adaptive soundness, where cheating provers are restricted to choose the statement x independently of the crs. Non-adaptive soundness is interesting for two reasons: First, in many settings (1) the adversary indeed does not get to choose the statement depending on the crs, or (2) one can informally argue that statements that do depend on the crs are likely to be contrived. Moreover, by using standard complexity-leveraging techniques, one can obtain adaptive soundness from non-adaptive soundness, at the price of making the SNARG proof grow with the input size, which is often much smaller than the witness size.

SNARGs vs. Proofs of non-Membership. Let us first describe why even constructing such non-adaptive SNARGs for all of NP seems to be challenging, by giving an intuitive barrier to constructing computationally sound argument systems that applies to most known SNARG constructions [KRR14, BHK17, BKK+18, CJJ22, KVZ21, BBK+23].

Soundness of a SNARG is typically proved by exhibiting a polynomial (or sub-exponential) time reduction R that uses any cheating prover P*, in a black-box way, to break the underlying (falsifiable) cryptographic assumption. In the case of non-adaptive soundness, the analysis fixes any x /∈ L and presents a reduction Rx that uses any successful cheating prover P*x to break the assumption. Intuitively, such an Rx together with the mathematical proof πx, that indeed Rx can use any successful P*x to break the underlying assumption, constitutes a “witness” for x /∈ L. Note that for x ∈ L, such a pair (Rx, πx) does not exist (unless the underlying assumption is false), since for x ∈ L there does exist an efficient convincing prover Px (who is given a corresponding witness w), and thus there does not exist a poly-time reduction Rx that uses Px to break the assumption (unless, of course, the assumption is false). We believe that there are NP languages for which membership for x ∈ L requires a “witness” of size 2|x|, and thus we believe that any such reduction Rx “must be” of exponential size, which in turn implies that we need to rely on a cryptographic assumption that is 2|x|-secure. Indeed, the only known SNARG construction for all of NP which is based on falsifiable assumptions, due to Sahai and Waters [SW14], uses a reduction of size 2|x| and thus relies on a 2|x|-hard cryptographic assumption.

We note that the above intuitive argument is not a proof.1 Nevertheless, this thought experiment

---

1A proof of soundness is a single proof π for the statement that for every x /∈ L, the reduction Rx can be used to break the assumption, whereas the above argument assumes that for every x /∈ L there is a pair (Rx, πx) where πx is
suggests a connection between constructing SNARGs for an NP language \( \mathcal{L} \) and the problem of mathematically proving that an instance \( x \) is in the language \( \overline{\mathcal{L}} \). We thus focus on the following informal question:

Are there non-adaptive SNARGs for every language \( \mathcal{L} \in \text{NP} \) with a security reduction of size \( \text{poly}(T) \), where \( T \) is the length of a proof for \( x \notin \mathcal{L} \)?

1.1 Our Results

In this work, we provide a positive answer for the case where the size-\( T \) proof for \( x \notin \mathcal{L} \) is a (extended Frege) propositional proof [Kra95]. Our SNARG proof length grows poly-logarithmically with \( T \), while the \( \text{crs} \) grows linearly with the space required to verify this propositional proof, and non-adaptive soundness relies on the \( \text{poly}(T) \)-hardness of LWE. In the case that the propositional proof is of polynomial size (and the instance size is polynomially related to the security parameter) the assumption becomes the polynomial hardness of LWE.

**Theorem 1.1** (Informal). Given functions \( T = T(n) \) and \( S = S(n) \), there exists a SNARG for any NP language \( \mathcal{L} \) such that every \( x \notin \mathcal{L} \) has a propositional proof for \( x \notin \mathcal{L} \) of length \( T = T(|x|) \), that can be verified in space \( S = S(|x|) \). The size of the \( \text{crs} \) is \( S \cdot \text{poly}(\lambda) \) and the size of the SNARG is \( \text{poly}(\lambda, \log T) \). The SNARG is non-adaptively sound based on the \( \text{poly}(T) \)-hardness of LWE.

Two remarks about Theorem 1.1 are in order. First, the SNARG construction does not depend on the propositional proof, rather only on the parameters \( T \) and \( S \). The actual propositional proof appears only in the analysis. Thus, one does not need to “know” such a propositional proof in order to construct the SNARG, nor does such a proof need to be efficiently computable.

Second, we consider the implications of Theorem 1.1 when \( T(n) \) is polynomial in \( n \). In this setting, we get a SNARG under the polynomial hardness of LWE for a subclass of NP \( \cap \text{coNP} \), namely ones that have a poly-size propositional proof for \( x \notin \mathcal{L} \). While it is tempting to think that languages in NP \( \cap \text{coNP} \) should have propositional proofs of non-membership, it is not necessarily the case.

Nevertheless, important languages in NP \( \cap \text{coNP} \) do have such proofs. As an example, we observe that the DDH language does have a poly-size propositional proof of non-membership. We thus obtain the following corollary.

**Corollary 1.2.** There is a SNARG for the DDH language that has non-adaptive soundness under the polynomial hardness of LWE. The proof length is \( \text{poly}(\lambda) \) and the \( \text{crs} \) length is \( \text{poly}(n, \lambda) \), where \( n \) is the instance size.

One interesting fact about Corollary 1.2 is that the DDH language does not appear to have a non-signaling PCP (with small locality). Indeed, it is believed that the DDH language, on groups of size roughly \( 2^n \), are not decidable in DTIME(\( 2^{n \epsilon} \)) for some constant \( \epsilon > 0 \). As a result, our SNARG has proof length which is less than logarithmic in the deterministic time complexity of the language! This stands in contrast to (1) most prior works on SNARGs from standard assumptions, which were

---

*Note: The text above is a natural representation of the document content as per the instructions provided.*
only for languages that have non-signaling PCPs,\(^2\) and (2) the Gentry-Wichs barrier, which says that exactly this kind of SNARG with adaptive soundness is impossible to achieve. We believe that the techniques we use in this work to circumvent these barriers are of independent interest and will lead to further follow-up work.

**Encrypt-Hash-and-BARG**  We propose a new method for constructing SNARGs. Specifically, we extend the Hash-and-BARG method \([\text{CJJ22, KVZ21}]\) to an “encrypt-hash-and-BARG” method. We elaborate on this method in Section 2, and in what follows we state its implications.

For a given \(\text{NP}\) language \(L\) and for any instance \(x \in \{0, 1\}^*\), we denote by \(C_x\) the verification circuit that given an input \(w\) outputs 1 if and only if \(w\) is a valid witness for \(x \in L\).

**Theorem 1.3** (informal, see Theorem 4.8). For every parameter \(\ell = \ell(n)\), we construct a SNARG for any \(\text{NP}\) language \(L\) that has the following property: For every \(x \not\in L\) there exists an extension \(E_x\) of the verification circuit \(C_x\) such that \(E_x\) has the property that local consistency with locality \(\ell\) implies global correctness. The length of the SNARG is \(\text{poly}(\lambda, \ell, \log |E_x|)\), and the length of the crs is \(|E_x| \cdot \text{poly}(\lambda)\).\(^3\) The SNARG has non-adaptive soundness assuming the \(\text{poly}(|E_x|)\)-hardness of LWE.

We call such a family of circuits \(\{E_x\}\) a locally unsatisfiable extension of the circuit \(C(x, w)\) (Definition 4.4). We emphasize that in Theorem 1.3, the extended circuit \(E_x\) need not be efficiently computable from \(x\), which is how we circumvent prior barriers. Our SNARG construction does not use this extension and it is only used in the analysis.

Moreover, we show an improved version of Theorem 1.3 that holds even when the extended circuit \(E_x\) has super-polynomial size, provided that every gate \(g\) in \(E_x\) the sub-circuit of \(E_x\) computing \(g\) is of size at most \(\gamma\).

**Theorem 1.4** (informal, see Theorem 5.2). For parameters \(\ell = \ell(n)\) and \(\gamma = \gamma(n)\), there exists a SNARG for any \(\text{NP}\) language \(L\) that has the following two properties: (1) For every \(x \not\in L\) there exists an extension \(E_x\) of the verification circuit \(C_x\) such that \(E_x\) has the property that local consistency with locality \(\ell\) implies global correctness. (2) For every gate \(g\) in \(E_x\) the sub-circuit of \(E_x\) computing \(g\) is of size at most \(\gamma\).

The length of the SNARG is \(\text{poly}(\lambda, \ell, \log |E_x|)\), the length of the crs is \(\gamma \cdot \text{poly}(\lambda)\),\(^4\) and non-adaptive soundness holds assuming the \(\text{poly}(|E_x|)\)-hardness of LWE.

We deduce Theorem 1.1 by combining Theorem 1.4 with a construction of such locally unsatisfiable extensions for any \(\text{NP}\) language \(L\) with bounded space propositional proofs of unsatisfiability (Theorem 7.3).

**1.2 Related Work**

There is a large body of work, especially in recent years, that construct SNARGs under standard hardness assumptions for more and more expressive languages.

\(^2\)The only other work that break this non-signaling barrier are the works of [SW14, JJ22] which rely on indistinguishability obfuscation, and the recent work [BBK* 23]. We elaborate on all these three works, and compare them to our work, in Section 1.2.

\(^3\)In fact, we can even compress the crs length to polynomial in a succinct description \(\text{aux}_x\) of the circuit \(E_x\), which may be much smaller.

\(^4\)One could strengthen this result to make the crs grow only with the (possibly succinct) description of the circuit computing each gate \(g\). For the sake of simplicity we chose to avoid this extra layer of complication in the technical section.
In what follows, we restrict ourselves to discussing the most closely related works, the ones that break the non-signaling barrier, and construct SNARGs for languages that we believe do not have non-signaling PCPs. Similarly to our work, all these works provide non-adaptive soundness.

**Comparison with [SW14].** Sahai and Waters [SW14] constructed the first SNARG system for all of NP, without relying on random oracles or knowledge assumptions. Their SNARG system has proofs of length $O(\lambda)$, independent of the statement or witness length, and a CRS of length $O(|w|)$, where $|w|$ is the length of the NP witness. In contrast to [SW14], in our SNARG system the proof size also grows poly-logarithmically with the length of the propositional proof of unsatisfiability. On the other hand, our CRS has length proportional to the space required to verify the propositional proof, which in general is shorter than the witness length, perhaps even exponentially so.

Importantly, [SW14] requires the full power of secure indistinguishability obfuscation, which when converted into a falsifiable assumption, becomes a $2^{|x|}$-hardness assumption (where $x$ is the instance). In contrast, our constructions are based on a single, simple, falsifiable, post-quantum assumption, namely the poly($T$)-hardness of LWE, where $T$ is the length of the propositional proof.

Finally, our construction permits a transparent setup, meaning that our CRS can be generated using public coins, whereas the CRS in the [SW14] construction consists of an obfuscated circuit.

**Comparison with [JJ22].** Our work is inspired by the work of Jain and Jin [JJ22], which pioneered the idea of using co-non-deterministic proofs for a language $L$ to improve the assumption in [SW14] from a $2^{|x|}$-hardness assumption to an assumption falsifiable in time less than $2^{|x|}$. Specifically, in [JJ22], they are able to rely on the $2^{|x|^\epsilon}$-hardness of falsifiable assumptions (used to build indistinguishability obfuscation for circuits of input length $|x|^\epsilon$) to instantiate a new variant of the [SW14] SNARG.

We improve on [JJ22] in several ways: they construct a SNARG for all NP languages $L$ that have polynomial-size propositional proofs for $x \notin L$, and their soundness relies on indistinguishability obfuscation for circuits with security parameter input length. We construct a SNARG for the same NP languages assuming the polynomial hardness of LWE. In addition, [JJ22] inherits the trusted setup of [SW14] while our scheme has transparent setup. Finally, we construct SNARGs for a richer class of languages: as described in Theorem 1.1, our construction can work with a propositional proof for non-membership which may be of super-polynomial size $T$, as long as it can be verified in polynomial space.

**Comparison with [BBK+23].** Our work also draws inspiration from [BBK+23], which constructs a SNARG with non-adaptive soundness from the polynomial hardness of LWE for any monotone batch NP language. Their construction relies on the hash-and-BARG blueprint, but they use a special hash family, which they call predicate extractable. Looking into the details of this predicate-extractable hash function, one can think of it as a commitment to, and quasi-argument about, the output of an encrypted, instance-dependent extended circuit. Indeed, there is a specific, simple locally unsatisfiable extension implicit in [BBK+23], and as a result our main theorem implies the result of [BBK+23] as a corollary (albeit with a slightly different construction and analysis).

Importantly, such languages are not known to have non-signaling PCPs, and thus [BBK+23] was the first to break the “non-signaling PCP barrier”. However, the monotone policy batch languages considered in [BBK+23] were still easy in that they could be decided in time $2^\ell$, where $\ell$ denotes
the SNARG proof length. In contrast, in this work we construct SNARGs for hard languages such as DDH, which is a much more powerful result.

1.3 Subsequent Work

A subsequent work [JKLM24] makes non-black-box use of the proofs of two of our main theorems (Theorems 5.2 and 6.2) in order to give new constructions of SNARGs for all NP languages (under stronger assumptions than what was needed in this paper). For ease of understanding, we have added formulations of the properties used by [JKLM24] as additional theorem statements. The proofs of these new theorem statements are identical to the proofs of our original theorems, but the theorem formulations are due to [JKLM24]. See Sections 4.6 and 5.4 and Remark 6.1 for details.

1.4 Paper Organization

We first present our general method for constructing SNARGs. Specifically, Theorem 1.3 is formalized and proved in Section 4, and Theorem 1.4 is formalized and proved in Section 5. Then we focus on propositional proofs. Theorem 1.1 is formalized and proved in Sections 6 and 7 and in Section 8 we derive the corollary for DDH (Corollary 1.2).

2 Technical Overview

Our starting point is the blueprint of constructing a SNARG via the hash-and-BARG paradigm [CJJ22, KVZ21]. In one variant of this construction, to prove that $x \in L$, the prover takes the following actions:

- First, the prover uses its witness to compute a “transcript” $\tau$ containing the values of all the wires of the verification circuit $C_x$ when evaluated on $w$.
- Then, the prover hashes $\tau$ using a hash family with local opening, obtaining a hash value $v$. The prover sends $v$ to the verifier along with an opening of the output wire of $C_x$ to 1.
- Finally, the prover sends $\ell$ proofs that these hashed wire values satisfy all the gates of the circuit, by using a non-interactive batch argument (BARG) [CJJ22]. That is, the prover sends $\ell$ BARGs certifying that for every gate $g$ of $C_x$ there is an opening to the input and output wires of $g$ such that these openings are valid w.r.t. $v$ and they respect the gate.

The parameter $\ell$ in this construction is called its “locality.” Assuming that the BARG satisfies a security notion called “somewhere extractability” (Definition 3.7), the above construction is known to satisfy the following weak soundness guarantee: one can efficiently convert any successful cheating prover $P^*$ into a local assignment generator for the verification circuit $C_x$, denoted by LocalGen.

A local assignment generator (with locality parameter $\ell$) is a randomized algorithm that takes as input $\ell$ wires, and outputs an assignment to these wires, with the following guarantees:

1. **Local consistency**: The assignment is locally consistent (i.e., it respects all the gates).

2. **Non-Signaling**: for any two possible wire sets $T_0, T_1$ of size at most $\ell$, the distribution of the assignments for the wires in $T_0 \cap T_1$ in LocalGen($T_0$) and in LocalGen($T_1$) are computationally indistinguishable.
A local assignment generator satisfies the circuit $C_x$ if it has one more property:

3. **Satisfiability**: The output wire of $C_x$ is assigned a value of 1 with high probability.

Unfortunately, it is well known that even if the circuit $C_x$ is unsatisfiable, it may still have a local assignment generator; in other words, “local consistency” need not imply “global correctness” for $C_x$. To remedy this, starting from the work of [KRR14], there have been several attempts to construct an extension of $C$, denoted $E(C)$, such that local consistency of the extended circuit $E(C)_x$ does imply global correctness. However, these attempts are inherently limited: such an extension can exist only for languages that have a PCP with non-signaling soundness, which in turn is known to limit the underlying language $L$ to the complexity class $\text{DTIME}(n^{O(1)})$ [KRR14].

**New Insight.** Our new idea is to consider inefficient, instance-dependent encodings of the circuit $C_x$. Namely, suppose for any fixed $x \notin L$ there exists a way to extend the verification circuit $C_x$ into a new circuit $E_x$ such that local consistency of $E_x$ implies that the specific wire of $E_x$ corresponding to the output wire of $C_x$ must be 0. We call such a family $\{E_x\}$ a locally unsatisfiable extension of the circuit $C$ (Definition 4.4) and remark that in principle, $E_x$ need not even have polynomial size.

A naive proposal would be to ask the prover to hash-and-BARG the extended circuit $E_x$ and thus derive soundness from the local unsatisfiability of $E_x$. Unfortunately, it seems like we are in a lose-lose situation:

- If $E_x$ can be computed efficiently from $C_x$, then we can essentially think of the extension as being independent of $x$, by thinking of the new extended circuit as being $E(C)_x$, similar to prior works.
- On the other hand, if $E_x$ cannot be computed efficiently from $C_x$, then the prover does not know what computation to run, so we do not have completeness.

Additionally, even if the prover magically knew a description of the circuit $E_x$, if $E_x$ has super-polynomial size, the honest prover cannot even evaluate it.

Nevertheless, we give a construction of a SNARG assuming the existence of a locally unsatisfiable extension $\{E_x\}$ of the NP verification circuit $C$, despite being unable to compute what it is. Our construction can handle any polynomial-size such extension, and moreover can handle extensions where the size of $E_x$ is super-polynomial, as long as each individual wire value of $E_x(w)$ is efficiently computable from the input $w$.

**The encrypt-hash-and-BARG paradigm.** How do we build a SNARG that makes use of the existence of $E_x$ without being able to compute it? We ask the prover to give a hash-and-BARG for a hidden, encrypted computation that is not checked in the clear! Namely, we include in the crs a ciphertext (using fully homomorphic encryption); in the actual scheme, this ciphertext can be an encryption of the all-zero string, but in the analysis we will think of this ciphertext as encrypting the description of the extension $E_x$. Since the FHE secret key is not provided to any party, the success probability of a cheating prover will not change when the encryption of the all zero string is replaced with an encryption of the description of $E_x$ in the analysis.

---

5Namely, $E_x$ is obtained by adding wires and gates to $C_x$.

6For technical reasons, our SNARG proof length will grow with $\log|E_x|$, so we can handle $|E_x| = 2^4$ but not $|E_x| = 2^n$. 
In our construction, the prover hash-and-BARGs the original circuit $C_x$ and additionally computes the values of all the wires of the (homomorphically evaluated) encrypted extended computation, and adds a hash-and-BARG for this encrypted computation. In more detail, we think of the extended circuit $E_x$ as first computing $C_x$ and then computing a circuit $D_x$ that takes as input all the wires of $C_x$. The prover now takes the following actions:

- As before, compute the transcript $\tau$ for $C_x(w)$.
- (New) Consider the encrypted computation $\text{Eval}(U_{D_x}, \cdot)$, that takes as input all the wires of $C_x$ and a circuit description $\langle D_x \rangle$, and evaluates the circuit $D_x$ on these wires “underneath” of the encryption. Compute the transcript $\overline{\tau}$ for $\text{Eval}(U_{D_x}, \cdot)$ on $\tau$.
- Apply Hash-and-BARG to the joint computation of $\tau$ and $\overline{\tau}$. As before, the prover provides an opening of the hash value $v$ to demonstrate that the output wire of $C_x$ is assigned 1.

In the actual scheme, the behavior of the prover (and verifier) is completely independent of $E_x$. However, in the soundness analysis, we replace the all zero encryption with an encryption of $\langle D_x \rangle$. The punchline of this approach is that we can show (Section 4) that given a cheating prover for this “encrypted-hash-and-BARG” construction on a non-adaptively chosen input $x$, one can obtain a local assignment generator for the circuit $E_x$ with a slightly worse locality parameter than before.\(^7\) This requires handling two technical issues:

1. The computation of $D_x$ is encrypted rather than in the clear. Essentially, one needs to show that a local assignment generator for the encrypted computation implies a local assignment generator for the computation “underneath” the encryption. This is proven (Lemma 4.12) assuming that the FHE scheme has gate-by-gate evaluation and satisfies a standard notion of malicious correctness.

2. Even “underneath the encryption,” what is evaluated (in the honest case) is a universal circuit $U$ on input $\langle (D_x), C_x(w) \rangle$ rather than the circuit $D_x$ on $C_x(w)$. Thus, one needs to show that local assignment generators are roughly preserved under a standard universal circuit transformation (Lemma 4.13).

We handle these technical issues and show that a local assignment generator for $E_x$ can in fact be constructed. Thus, soundness of our SNARG holds assuming that $E_x$ is locally unsatisfiable. We refer the reader to Section 4 for further details.

### 2.1 Handling extensions of super-polynomial size

Thus far, we have described a SNARG construction in which the crs length and prover runtime grow linearly with the size of a family of extended circuits $\{E_x\}$. This suffices to prove Theorem 1.3 but not Theorem 1.4. We next show how to extend this paradigm to handle locally unsatisfiable extensions of super-polynomial size, as long as each gate in the extension can be computed in polynomial time.

\(^7\)In fact, we implicitly show something stronger: a “universal local assignment generator” $\text{LocalGen}(\langle E_x \rangle, S)$ that takes $\langle E_x \rangle$ as additional input, such that for every fixed $E_x$, $\text{LocalGen}(\langle E_x \rangle, \cdot)$ is a local assignment generator for $E_x$. We do not use this fact in this paper, but a subsequent work [JKLM24] defines and makes use of this stronger property.
Suppose that \( \{ E_x \} \) is a family of extended circuits such that every wire in \( E_x \) is computed by a circuit of size at most \( \gamma = \gamma(n) = \text{poly}(n) \). Here is a naive idea for reducing the prover complexity to be linear in \( \gamma \) rather than \( |E_x| \):

- Instead of including an encryption of \( \langle D_x \rangle \) in the crs, include an encryption of the description of a size-\( (\ell \cdot \gamma) \) sub-circuit of \( D_x \), denoted \( D_x[S] \), corresponding to the sub-circuit necessary to compute all wires in a set \( S \) of size \( \ell \).

  Again, this step only occurs in the analysis; in the real construction, an all-zero circuit of size \( \ell \cdot \gamma \) will be used.

- As before, the prover homomorphically evaluates the sub-circuit on \( \tau = C_x(w) \). Since this sub-circuit has \( \ell \) specially designated “output wires,” the prover can send the corresponding encrypted outputs.

- Finally, similarly to before, give a Hash-and-BARG proof of correctness for its claimed computations (both \( C_x(w) \) and the encrypted computation).

Given a cheating prover for this new candidate SNARG, one might hope to obtain a very simple local assignment generator for \( E_x \): on a set \( S \), the local assignment generator can simulate the prover on an encryption of \( \langle D_x[S] \rangle \) repeatedly until it produces a valid BARG proof; then, decrypt (under the FHE) the “evaluated ciphertext” sent by the prover, resulting in an \( \ell \)-bit assignment to \( S \). Indeed, one can argue that these assignments must always satisfy the gates of \( E_x \) contained in \( S \).

Unfortunately, this idea does not work. Briefly, this is because the candidate local assignment generator for \( E_x \) is evaluated using the FHE secret key. Since the circuit \( D_x[S] \) is encrypted under the FHE, this means that all \( \ell \) bits of the output of the local assignment generator may depend on \( S \), and thus they may not satisfy non-signaling.

To ensure non-signaling, we change the construction to independently encrypt \( \ell \) different size \( \gamma \) sub-circuits \( D_1, \ldots, D_{\ell} \) (one for each wire in \( S \)) under different FHE keys. Then, the analogous “local assignment generator” would satisfy a form of non-signaling: by semantic security, the value decrypted under FHE secret key \( \text{sk} \) would not signal information about which wires were encrypted under the other FHE keys. However, this is also unlikely to result in a local assignment generator, due to the following two (related) issues:

- **Wire self-consistency.** It is not clear that if the same wire is encrypted under two different FHE keys, the resulting evaluated ciphertexts will decrypt to the same output.

- **Gate consistency.** It is not clear that the gates of \( E_x \) are respected by this wire assignment distribution, since the three wires forming a gate are encrypted under different FHE keys.

**Why doesn’t the BARG save us?** So far, we have not actually invoked security of the hash-and-BARG in our new analysis. Perhaps hash-and-BARG security can help argue the two consistency properties above!

Repeating our analysis from the proof of Theorem 1.3, we see that hash-and-BARG security (along with FHE correctness) has a very specific implication: the existence of a “second-level” local assignment generator for the following computation:

- Input: a candidate witness \( w \).
• First, evaluate $\tau = C_x(w)$
• Then, for a fixed tuple $i_1, \ldots, i_\ell$, evaluate the sub-circuit $D_{ij}$ of $E_x(\tau)$ that outputs the $ij$th wire of $E_x(\tau)$, for all $1 \leq j \leq \ell$.

The $\ell$ output wires of this new circuit correspond to (one sample from) the original local assignment generator applied to $(i_1, \ldots, i_\ell)$.

Now, we can ask: is this enough to imply self-consistency? If $i_1 = i_2$, must the first two output wires of the second-level local assignment generator be assigned the same value? Unfortunately, each $D_{ij}$ can be an arbitrary polynomial-size circuit, and local assignment generators on arbitrary circuits may not satisfy this form of “global consistency.”

Solution: extend the sub-circuits, again. We solve this issue by turning to one of the original tools from [KRR14]: we will efficiently extend the $\ell$ sub-circuits to enforce a form of global consistency on the second-level local assignment generator. Originally, [KRR14] showed that an appropriate extension $C_{\text{Ext}}$ of any deterministic circuit $C$ has the property that a local assignment generator for $C_{\text{Ext}}(x)$ must assign $C(x)$ to the output wire. Our idea is to have the prover homomorphically evaluate (and certify having evaluated) $(D_{i_1})_{\text{Ext}}, \ldots, (D_{i_\ell})_{\text{Ext}}$ rather than $D_{i_1}, \ldots, D_{i_\ell}$. Then, making use of a new variant of the [KRR14] analysis of their circuit extension (Theorem 4.5), we are able to deduce the desired wire self-consistency property for the (first-level) local assignment generator.

We remark that rather than appealing to [KRR14], one could also cryptographically extend the $D_{ij}$ using a collision-resistant hash function (as in [KP16, CJJ22]). We use the information-theoretic [KRR14] extension for technical reasons, in order to simplify some of our intermediate abstractions and security reductions.

Finally, in order to handle gate consistency, we combine the two approaches described so far. In the final construction, we use $\ell + 1$ independent FHE keys, where the last key is used to encrypt a description of the joint computation $D_x[S]$. We defer the reader to Section 5 for how to prove gate consistency, but the ideas are similar to how we handled wire self-consistency above.

2.2 Extended Frege proofs and local unsatisfiability

Armed with the new paradigm for constructing SNARGs we now ask: which NP languages have locally unsatisfiable extensions with the appropriate complexity?

For simplicity, let us consider the case where $|E_x| = \text{poly}(n)$. In this case, it is easy to see that any language $\mathcal{L}$ with a $\ell$-locally unsatisfiable extension lies in the complexity class $\text{NP} \cap \text{coNTIME}(n^{O(\ell)})$ (Theorem 5.8); to prove that a statement $x$ is not in $\mathcal{L}$, one can non-deterministically guess the description of $E_x$ and then compute the optimal (perfectly) non-signaling strategy in time $n^{O(\ell)}$.

Thus, we restrict (essentially without loss of generality) to considering languages in $\text{NP} \cap \text{coNTIME}(T)$ for some parameter $T$. However, even in this restricted setting, it is highly unclear whether any given family of circuit extensions $\{E_x\}_x$ satisfies the “local consistency implies global correctness” property that we want.

In this work, we show a connection between propositional proofs (specifically, Extended Frege proofs) and local unsatisfiability; this corresponds to the sub-class of $\text{NP} \cap \text{coNTIME}(T)$ where

---

8One can hope to handle super-polynomial $T$ by allowing $E_x$ to have super-polynomial size. In the end, our SNARG proof length will grow poly-logarithmically with $T$, which prevents extending our results to NP-complete languages.
non-membership in $L$ has a length $T \mathcal{EF}$ proof. This begs the question: why propositional proofs? Extended Frege proofs were utilized to great effect in the related work [JJ22], to reduce the length of a hybrid argument in proving the security of iO (and therefore of the [SW14] SNARG). But what do they have to do with local assignment generators?

**Special case: deterministic computations.** To understand this question, let us consider the very simple setting of deterministic computations, which were studied in [KRR14]. Here, it is not obvious (at least from first principles) whether extending the circuit is even necessary. Namely, one can ask: does a local assignment generator for a deterministic computation $C(x)$ have the property that the output wire $\tau_{out}$ is assigned the value $C(x)$ with high probability?

In general, the answer is no (as previously discussed). However, the answer is actually yes if we restrict to an even simpler and unrealistic special case: if the local assignment generator has *perfect* (zero error) non-signaling and local correctness, then the output wire is assigned $C(x)$ with probability 1. To see this, simply observe that in a deterministic computation, every wire $\tau_i$ has an associated “correct value” $\tau_i(x)$ that the local assignment generator should provide. One can argue, by induction on a topological ordering of $C$, that the $i$th wire must be assigned the value $\tau_i(x)$ by making use of the non-signaling and local correctness properties of a local assignment generator.

In [KRR14,BHK17], it is shown how to handle the case where the local assignment generator *has errors*. In this case, extending the circuit is actually necessary, because the analysis above requires *exponentially many invocations* of non-signaling and local correctness. This seems strange – the circuit has polynomial size, after all – but the issue is the *locality* $\ell$. It turns out that the number of steps in the above analysis grows with the number of steps required to *pebble* the circuit $C$ in the reversible pebbling game [Ben89], when restricted to $\ell$ pebbles. In general, a circuit may require exponentially many steps to pebble for small choices of $\ell$, which would lead to an intolerable error blowup in the analysis. To get around this, [KRR14] extend the circuit $C$ in a way that reduces the complexity of its pebbling game.

**Enter propositional proofs.** Now, we return to the general case of non-deterministic computations $C_x(w)$. Suppose that $C_x(\cdot)$ is unsatisfiable. Here, we know that unlike the deterministic case, *even a perfect local assignment generator* for $C_x(\cdot)$ need not satisfy the property that $\tau_{out}$ is assigned 0 (because this can be tested in sub-exponential time). Intuitively, the operational difficulty is that individual wires of $C_x$ are no longer supposed to take specific (deterministic) values; each candidate witness gives a different wire assignment to $C_x$, so it is not clear what should be argued inductively about these wire values.

This leads to the key insight of this section:

Propositional proofs give a **deterministic handle** on a non-deterministic computation.

That is, since a propositional proof is a sequence of tautologies – formulas that must be satisfied by any assignment to variables satisfying a collection of premises – one can hope to generalize from the deterministic case by arguing about the *sequence of formulas* instead of the *individual variables*.

Bringing this idea to fruition requires significant care, especially in order to handle imperfect local assignment generators; we outline our approach below.
Background on Logic. We recall propositional logic systems with extension rules (i.e. Extended Frege systems) [Kra95]. Such a logic system is defined as a tuple of variables and connectives such as “∧”, “∨”, “→”, “¬”, “↔”, which represent “and”, “or”, “imply”, “negation”, and “equivalent”, respectively. A proof in such a logic system is a sequence of “lines”, where each line is a formula derived via one of the following rules.

- **Axioms.** The formula is a constant size tautology, which means for every truthful assignment, the formula is evaluated to 1.

- **Modus Ponens.** There exist two formulas $f, g$ such that the formulas $f, g, f \rightarrow g$ are the lines somewhere before the current line, and the current line is the formula $g$.

- **Extension Rule.** The line is in the form $v \leftrightarrow \phi$, where $v$ is a variable that has not appeared before the current line, and also does not appear in $\phi$.

The size of a formula is the length of the binary string that is used to write down such a formula, including all the punctuations. The size of a proof is the total size of the lines in the proof.

The extension rule can be used to shorten the size of each line in the proof, by introducing new propositional variables to the subformulas used in the proof. Indeed, one can assume without loss of generality that each line of the proofs in the Extended Frege system is a constant size formula, without significantly blowing up the size of the proofs.

We first describe how to build a polynomial-size extended circuit, if the propositional logic proof is polynomial size. Indeed, our construction of the extension circuit for polynomial-size propositional proofs is almost identical to the padded circuit used in [JJ22]. We describe the construction again in the context of the extended circuits.

**Extension from Polynomial Size Propositional Proofs.** The key structural property of propositional logic proofs is that each line of the proof is a formula, and hence is naturally a circuit. Hence, we can use them in our construction of $E_x$. Moreover, the proof is local, which means that the truthfulness only depends on a constant number of previous lines. Recall that, we assume without loss of generality that each line of the proof has a constant size.

Bearing these structural properties in mind, given a propositional logic proof $\theta_1, \ldots, \theta_T$ for the unsatisfiability of $C_x$, the initial attempt to build $E_x$ is as follows. For each $\theta_i$, we build a circuit $C_i$ that computes $\theta_i$, and add $C_i$ in $E_x$. To prove the local-to-global property, our intuition is that, since each line must be true, the circuits $C_i$’s must output 1’s. Moreover, we hope to use an inductive argument to show that this is indeed true if we extract the output of $C_i$ using the local assignment generator. For example, suppose $\theta_i$ is derived from $\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_c}$, then we can extract the outputs of $C_{i_1}, \ldots, C_{i_c}, C_i$ to argue that, if all the circuits $C_{i_1}, \ldots, C_{i_c}$ outputs 1, then $C_i$ must also output 1. Finally, the last circuit $C_T$ must compute a function checking whether $out$ is 0 or not, where $out$ is the output gate of $C_x$. This is because a propositional proof must end with the statement, and in our case, the statement is “$out = 0$”. Hence, if we can use the aforementioned inductive argument to show that $C_T$ outputs 1, then it implies $out = 0$. In this way, we can argue that local consistency implies global correctness.

However, the above initial attempt does not work directly, because a closer examination will reveal that the security loss in such an inductive argument is exponential in the length of the proof. The reason is that, for each $\theta_i$ that follows from an inference rule, the indices $i_1, \ldots, i_c$ may be different for each $i$, and they may not follow any pattern. Recall that in the inductive argument
above, before arguing that $C_i$ must output 1, we need to extract $C_{i_1}, \ldots, C_{i_c}$ and prove that they all output 1 first. However, when we transit from the index $(i - 1)$ to $i$ in the inductive argument, it is possible that the outputs of $C_{i_1}, \ldots, C_{i_c}$ have not been extracted. A potential solution to this issue is to extract all the outputs of $C_1, \ldots, C_{i-1}$, but we cannot afford such locality in the local assignment generator.

To resolve this issue, we need to have a mechanism that allows us to “succinctly” memorize the fact that we have proven that the outputs of $C_1, \ldots, C_{i-1}$ are all 1’s, by only extracting a small number of wires and gates in $E_x$. We achieve this by building a binary tree of AND gates, where the leaf nodes of the tree are the outputs of $C_1, \ldots, C_T$. In the inductive argument, at the $i$-th step, we extract $O(\log T)$ roots of some sub-trees in the binary tree. Those roots are chosen such that the leaves of those roots consist of the first $i$ leave nodes. Now to ensure the first $i$ circuits output 1, we only need to examine whether those roots output 1 or not. For more details, see Section 6.

Note that in this construction of $E_x$, we can only handle polynomial-size propositional logic proofs. If the propositional logic proof is of super-polynomial size, the number of leaves in the AND tree is super-polynomial. Then it is not clear how to compute the root node of the AND in polynomial time, since the root of the tree depends on the values of all the leaves.

Towards Super-polynomial Size Propositional Proofs. To extend the idea to super-polynomial size propositional proofs, our key observation is that the procedure of adding $C_i$’s to the leaves of the AND tree is similar to writing down a mathematical proof on a piece of paper, where the space of the paper is limited. Note that, sometimes we can erase some formulas on the paper and rewrite new formulas there, as long as the formulas can be derived only using the remaining formulas on the paper. Then we can reuse the space of the paper and write long proofs. This motivates us to define a space notion of propositional proofs, and only keep the formulas in the space on the leaves of the AND tree.

Indeed, space complexity has been defined for propositional logic proofs [ET01, ABSRW02]. However, to the best of our knowledge, those definitions only define space complexity for propositional proof systems without the extension rule. In our setting, we can handle the extension rule by adding new gates in $E_x$. Specifically, we can have each extended variable be associated to a new gate in $E_x$, and use the defining formula of the variable to specify how the gate should be computed from existing gates.

Defining Space of Extended Frege Proofs. In this work, we introduce a new definition of the space complexity for propositional proofs with extension rules (i.e., for extended Frege systems). In our definition, at any point in the derivation of the proof, we maintain a set of formulas as the “memory”, and we require that new formulas (resp. new variables) must be derived from existing formulas (resp. variables) in the memory. We also allow the erasure of the formulas in the memory. However, if the erased formula is the defining formula of some variable $v$, then we further require that the variable is not used later in the proof. The space complexity of the proof is then defined as the maximum total size of the formulas in the memory throughout the derivation of the propositional proof. For more details, see Section 7.1.

We will show that if a circuit $C_x$ has propositional proofs of unsatisfiability, where the proof has polynomial space, then we can build an extended circuit $E_x$, where the size of the circuit grows with the size of the propositional proof, which we allow it to be super-polynomial. However, each gate in the extended circuit can be computed using a polynomial-size sub-circuit. From our aforementioned
construction of SNARGs via Encrypt-Hash-and-BARGs, this would give us SNARGs with CRS size growing with the space of the propositional proof, and SNARG size grows only logarithmically in the size of the propositional proofs.

**Extended Circuits from Bounded-Space Proofs.** The construction of the circuit extension \( E_x \) is as follows. Our starting point is our aforementioned AND tree construction in the case of polynomial-size propositional proofs. To ensure the AND tree is polynomial size, our idea is to build the AND with only \( s \) leaves, where \( s \) is a polynomial upper bound on the space. However, it remains to handle the erasure operation in propositional proofs. A natural idea is to build a “dynamic” AND binary tree, with insertions and deletions of the leaves. However, it is not clear if we can delete the gates in \( E_x \) during the construction procedure of \( E_x \). Because if we delete the gates, what is the point of adding them in the first place?

\[
\begin{array}{c}
\wedge \\
/ \\
/ \\
/ \\
1&2&3&4
\end{array}
\quad
\begin{array}{c}
\wedge \\
/ \\
/ \\
/ \\
1&2&3&4
\end{array}
\]

Figure 1: An example of inserting a new leaf-to-root path for the second leaf node. Left: original AND tree circuit. Right: circuit after adding a new leaf-to-root path for the second leaf node. The new gates are marked in circles.

To deal with deletions, our idea is to use insertions to “emulate” deletions, as follows. To delete a leaf node, instead of really deleting it, we build an entire new root-to-leaf path for that leaf node on top of the original tree. Namely, the children of the gates on the new root-to-leaf path are set to be the nodes in the original tree, if the child is not included in the root-to-leaf path. We show an illustrative example in Figure 1. Then we set the new leaf node \((2')\) on the new root-to-leaf path as a constant 1. Now if we only look at the sub-circuit that the new root in the root-to-leaf path depends on, it is still a binary tree, but now the leaf (2) can be regarded as “deleted”, because its value is set to be a constant 1, which does not affect the output of the AND tree.

In summary, to build \( E_x \), we start with a binary tree of \( s \) leaves, where \( s \) is the space of the propositional proof. Initially, all the leaf nodes are set to output the constant 1. Then we iterate through the lines in the proof. If the new line is derived via an inference rule, then we add a sub-circuit \( C_i \) computing the truth value of that line. Then we choose a leaf node that has not been assigned before. We create a new root-to-leaf path as above for that leaf node, and assign the output of \( C_i \) to the new leaf node. For the erasures, we find the leaf node that was assigned to the sub-circuit computing the formula being erased, and create an entirely new root-to-leaf path to it, and set the new leaf node to be the constant 1. Finally, we take the resulting circuit as \( E_x \).

The proof that local consistency implies global correctness follows from the same high-level idea as the polynomial-size propositional proof case, but there are more technical issues in the detail. For example, we need a new mechanism to succinctly memorize that all the inputs to the AND tree are all 1’s, because now the order of assigning \( C_i \)'s to the leaves is no longer monotonous and
consecutive, since the position of the newly assigned leaf may be arbitrary among all the leaves. For more details, see Section 7.

On the Provability of Propositional Logic. So far, we have shown how to build SNARGs from LWE, if the language has a polynomial-size or bounded-space proportional proof of non-membership. It remains to ask, for what languages do we have a propositional proof of non-membership? More broadly, one may ask what mathematical theorems can be formalized in polynomial-size or bounded-space propositional logic proofs?

Indeed, the latter question is one of the central questions in proof complexity. On the positive side, the bounded arithmetic developed by Cook [Coo75], Buss [Bus86], and many others shows that if a proof can be formalized in certain sub-theories of Peano arithmetic, then the proof can be translated into propositional proofs of certain lengths in Extended Frege systems. Moreover, any language in NP has a propositional proof of non-membership of size exponential in the length of the NP witness. However, it is unlikely that such a proof can always be polynomial size. Indeed, it has been shown [CR79] that if every tautology has a polynomial-size propositional proof, then NP = coNP.

In this work, we rely on the propositional translation [Coo75, Bus86] to show that the Diffie-Hellman language (DDH) has a polynomial-size propositional proof of non-membership. As an application, we show how to build SNARGs for DDH.

Applications to Diffie-Hellman Language. Recall that, Diffie-Hellman is the following language.

\[ L = \{(g, h, g^w, h^w) \mid g, h \in \mathbb{G}, w \in \mathbb{Z}_q\}, \]

where \( \mathbb{G} \) is a cyclic group of order \( q \). Let \( x = (g, h, g', h') \) be a fixed instance with \( x \notin L \), we want to show that the verification circuit \( C_x(w) \) for \( L \) has polynomial-size propositional proof of unsatisfiability. Note that for any such fixed \( x \), there exists a fixed \( s \in \mathbb{Z}_q \) such that \( h = g^s \) and \( h' \neq g'^s \). Then we can write down a mathematical proof that \( C_x(w) \) is not satisfiable, as follows.

- Suppose \( C_x(w) = 1 \) for some \( w \), then \( g' = g^w \) and \( h' = h^w \).
- From \( h = g^s \), this means that \( h' = h^w = (g^s)^w \).
- \( (g^s)^w = (g^w)^s = g'^s \). We reach a constriction with \( h' \neq g'^s \). Hence, \( C_x(w) \) is unsatisfiable.

We observe that each step of the above mathematical proof can be converted to polynomial-size propositional proof using Cook’s propositional translation [Coo75]. Putting them together, we obtain a polynomial-size propositional proof of unsatisfiability of \( C_x \). For more details, see Section 8.

3 Preliminaries

Notation. We will let \( \lambda \) denote the security parameter throughout the paper. We use \( \text{PPT} \) to denote probabilistic polynomial-time, and denote the set of all positive integers up to \( n \) as \( [n] := \{1, \ldots, n\} \). For any \( x \in \{0, 1\}^n \) and any subset \( J \subset [n] \) we denote by \( x_J = (x_j)_{j \in J} \). For any finite set \( S \), \( x \leftarrow S \) denotes a uniformly random element \( x \) from the set \( S \). Similarly, for any distribution \( D \), \( x \leftarrow D \) denotes an element \( x \) drawn from the distribution \( D \).
3.1 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, 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.

3.1.1 Universal Circuits

Let $\mathcal{C}$ denote a family of circuits, and let $\langle C \rangle$ denote the canonical representation of a circuit $C \in \mathcal{C}$. We define the explicit universal circuit $U(x, \langle C \rangle)$ to consist of a sequence of sub-circuits $\{U_i\}_{1 \leq i \leq s}$, where $U_i$ evaluates the $i$th gate of $C$ as follows:

- For every $1 \leq j \leq i$, the output wire $g^{(j)}_{\text{out}}$ of $U_j$ is connected to $g_i$ via a $O(\log s)$-size circuit that outputs $(1, b)$ if $j$ is equal to one of the children of $i$ in $g_i$, where $b$ is the wire value of $g^{(i)}_{\text{out}}$, otherwise, the output of this circuit is $(0, 0)$.
- Then, these $i \leq s$ outputs are converted into two outputs via a $O(\log s)$-depth circuit that removes all of the computed $(0, 0)$ values. When evaluated honestly, these two wires will be assigned the values $(g^{(j)}_{\text{out}}, g^{(k)}_{\text{out}})$ (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 wire of $U$ is defined to be the output wire of $U_s$. By construction, the output wire of each $U_i$ in an honest evaluation of $U(x, \langle C \rangle)$ will be exactly $g_i(x)$, and thus the output wire of $U$ will be $C(x)$.

Note that if $\mathcal{C}$ is a class of size $s$, depth $d$ (layered) circuits, the universal circuit $U(x, \langle C \rangle)$ is a (layered) circuit of size $O(s^2 \log s)$ and depth $O(d \log s)$.

Moreover, without loss of generality, we can always write $U$ as a NAND-circuit (every gate is a NAND gate).

---

9 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 $i - 1$.

10 There are more efficient known constructions of universal circuits, but this simple construction suffices for our purposes.
3.2 Hash Family with Local Opening

In this section we recall the definition of a hash family with local opening [Mer88]. These preliminaries are due to [BBK+23].

Syntax. A hash family (HT) with succinct local opening consists of the following algorithms:

Gen$\left(1^\lambda\right) \rightarrow \text{hk}$. This is a PPT algorithm that takes as input the security parameter $1^\lambda$ in unary and outputs a hash key $\text{hk}$.

Hash$(\text{hk}, x) \rightarrow \text{rt}$. This is a deterministic poly-time algorithm that takes as input a hash key $\text{hk}$ and an input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$, and outputs a hash value $\text{rt}$.

Open$(\text{hk}, x, j) \rightarrow \rho$. This is a deterministic poly-time algorithm that takes as input a hash key $\text{hk}$, an input $x \in \{0,1\}^N$ for $N \leq 2^\lambda$, and an index $j \in [N]$, and outputs an opening $\rho$.

Verify$(\text{hk}, \text{rt}, j, b, \rho) \rightarrow 0/1$. This is a deterministic poly-time algorithm that takes as input a hash key $\text{hk}$, a hash value $\text{rt}$, an index $j \in [N]$, a bit $b \in \{0,1\}$ and an opening $\rho$. It outputs 1 (accept) or 0 (reject).

Definition 3.1. (Properties of HT) A HT family $(\text{Gen}, \text{Hash}, \text{Open}, \text{Verify})$ is required to satisfy the following properties.

Opening completeness. For any $\lambda \in \mathbb{N}$, any $N \leq 2^\lambda$, any $x \in \{0,1\}^N$, and any index $j \in [N]$,

$$\Pr \left[ \text{Verify}(\text{hk}, \text{rt}, j, x_j, \rho) = 1 : \begin{align*} \text{hk} &\leftarrow \text{Gen}(1^\lambda), \\ \text{rt} &\leftarrow \text{Hash}(\text{hk}, x), \\ \rho &\leftarrow \text{Open}(\text{hk}, x, j) \end{align*} \right] = 1 - \text{negl}(\lambda).$$

Succinctness. In the completeness experiment above, we have that $|\text{hk}|+|\text{rt}|+|\rho| = \text{poly}(\lambda)$.

Collision resistance w.r.t. opening. For any poly-size adversary $A$ there exists a negligible function $\text{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr \left[ \begin{align*} \text{Verify}(\text{hk}, \text{rt}, j, 0, \rho_0) = 1 &: \text{hk} \leftarrow \text{Gen}(1^\lambda), \\ \land \text{Verify}(\text{hk}, \text{rt}, j, 1, \rho_1) = 1 &: \text{(rt, j, \rho_0, \rho_1) } \leftarrow A(\text{hk}) \end{align*} \right] = \text{negl}(\lambda).$$

Remark 3.1. We say that a hash family with local opening is $T$-secure, for $T = T(\lambda)$, if the collision resistance w.r.t. opening property holds against any poly($T$)-size adversary (as opposed to poly($\lambda$)-size) and the probability that the adversary finds a collision is $\text{negl}(T)$ (as opposed to $\text{negl}(\lambda)$). We refer to this property as $T$-collision-resistance w.r.t. opening.

Remark 3.2. One can naturally extend the definition of a hash family with local opening to allow the Open algorithm to take as input $(\text{hk}, x, J)$ where $J \subseteq [N]$ consists of a set of indices, as opposed to a single index. Open$(\text{hk}, x, J)$ will simply run Open$(\text{hk}, x, j)$ for every $j \in J$. Verify can be extended in a similar way to take as input $(\text{hk}, \text{rt}, J, b_j, \rho_j)$, and accept if and only if Verify$(\text{hk}, \text{rt}, j, b_j, \rho_j) = 1$ for every $j \in J$.

Theorem 3.2 ([Mer88]). Assuming the existence of a collision resistant hash family there exists a hash family with local opening (according to Definition 3.1).
3.3 Fully Homomorphic Encryption

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 $ct_0, ct_1$ that decrypt to bits $x_0, x_1$, it should be the case that $\text{Dec}_{sk}(\text{Eval}_{ek}(f, ct_0, ct_1)) = f(x_0, x_1)$.

Moreover, our definition explicitly describes the evaluation key as a sequence of $d$ (small) keys, since this structure is relevant to our SNARG construction.

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

- $\text{FHE.Setup}(1^\lambda, 1^d) \to (\text{pk}, \vec{ek}, \vec{sk})$. 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 $\text{pk} \in \{0, 1\}^{\ell}$, a tuple of $d$ evaluation keys $\vec{ek} = (\vec{ek}_1, \ldots, \vec{ek}_d) \in ((0, 1)^{\ell})^d$, and a tuple of $d + 1$ secret keys $\vec{sk} = (\vec{sk}_0, \ldots, \vec{sk}_d) \in ((0, 1)^{\ell})^{d+1}$.

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

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

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

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

- $\text{FHE.Eval}_{\text{ek}}(f, c_1, \ldots, c_n) \to c^*$. This is a deterministic algorithm that takes as input an evaluation key tuple $\vec{ek}$, a circuit representing a function $f: \{0, 1\}^n \to \{0, 1\}$ and $n$ ciphertexts $c_1, \ldots, c_n \in \{0, 1\}^\ell$. It outputs a ciphertext $c^* \in \{0, 1\}^\ell$.

**Definition 3.3 (FHE).** A leveled, gate-by-gate fully homomorphic encryption scheme $\text{FHE} = (\text{FHE.Setup}, \text{FHE.Enc}, \text{FHE.Dec}, \text{FHE.GateEval})$ is required to satisfy the following properties:

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

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

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

**Security.** The encryption scheme is semantically secure in the presence of $(\text{pk}, \vec{ek})$. 

17
**Theorem 3.4** ([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 3.3 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 LWE-based scheme will be pseudorandom under the LWE assumption.

**Remark 3.3.** 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. FHE.Enc\(_{pk}(m)\) will simply run \( c_i = \text{FHE.Enc}_{pk}(m_i) \) for every \( i \in [n] \), and set \( c = (c_1, \ldots, c_n) \in \{0,1\}^{\ell \cdot \text{poly}(\lambda)} \). Similarly, we extend FHE.Eval to take as input a function \( f: \{0,1\}^n \to \{0,1\}^u \) with multi-bit output.

### 3.4 Succinct Non-Interactive Arguments

We define succinct non-interactive argument systems (SNARG) [Mic94, GW11, BCCT13] for the computation of a non-deterministic polynomial-time Turing machine \( M \). Our definition allows for a long common reference string, maintaining verifier efficiency by requiring that the \( \text{crs} = (\text{crs}_P, \text{crs}_V) \) be separated into prover and verifier components (where the verifier component is required to be short).

A SNARG system for \( M \) consists of polynomial-time algorithms (SNARG.Gen, SNARG.P, SNARG.V) with the following syntax:

- The randomized setup algorithm SNARG.Gen takes as input a security parameter \( \lambda \in \mathbb{N} \) and an input length \( n \), both in unary, and outputs a pair of common reference strings \( \text{crs} = (\text{crs}_P, \text{crs}_V) \).
- The prover algorithm SNARG.P takes as input the prover reference string \( \text{crs}_P \), an input \( x \in \{0,1\}^n \) and its associated witness \( w \in \{0,1\}^m \), and outputs a proof \( \pi \).
- The verifier algorithm SNARG.V takes as input the verifier reference string \( \text{crs}_V \), an input \( x \in \{0,1\}^n \) and a proof \( \pi \). It outputs a bit indicating if it accepts or rejects.

**Definition 3.5.** A triple of algorithms \((\text{SNARG.Gen}, \text{SNARG.P}, \text{SNARG.V})\) is a SNARG system for a Turing machine \( M \) if the following hold:

**Completeness.** For every \( \lambda, n \in \mathbb{N} \) and every \( x \in \{0,1\}^n \) and \( w \in \{0,1\}^m \) such that \( M(x, w) = 1 \),

\[
\Pr \left[ \text{SNARG.V}(\text{crs}_V, x, \pi) = 1 : \begin{cases} (\text{crs}_P, \text{crs}_V) \leftarrow \text{SNARG.Gen}(1^\lambda, 1^n), \\ \pi \leftarrow \text{SNARG.P}(\text{crs}_P, x, w) \end{cases} \right] = 1.
\]

**Efficiency.** The length of \( \text{crs}_V \) is \( \text{poly}(\lambda, \log n, \log m) \). The length of a proof \( \pi \) is also \( \text{poly}(\lambda, \log n, \log m) \).

The runtime of \( \text{SNARG.V} \) is \( \text{poly}(|\text{crs}_V|, |\pi|) + \text{poly}(\lambda, \log n, \log m) \cdot n \).

**Non-adaptive Soundness.** For every \( x \in \{0,1\}^n \) where \( x \notin \mathcal{L}_M \) and every poly-size adversary \( \text{Adv} \), there is a negligible function \( \mu \) such that

\[
\Pr \left[ \text{SNARG.V}(\text{crs}_V, x, \pi^*) = 1 : \begin{cases} (\text{crs}_P, \text{crs}_V) \leftarrow \text{SNARG.Gen}(1^\lambda, 1^n), \\ \pi^* \leftarrow \text{Adv}(\text{crs}_P, \text{crs}_V, x) \end{cases} \right] \leq \mu(\lambda).
\]

18
An additional efficiency parameter of interest is the length of \( \text{crs}_P \); we do not impose any explicit requirements on it, but seek to minimize the length of \( \text{crs}_P \) when possible.

**Remark 3.4 (Transparent Setup).** The SNARGs constructed in this paper can be made to satisfy an additional property called transparent setup: the non-adaptive soundness notion above holds even if Adv is given the random coins used by SNARG.Gen. All of our SNARGs can be made to satisfy this property because in all of our constructions, the algorithm SNARG.Gen\((1^\lambda, 1^n)\) generates a polynomial number of ciphertexts for an LWE-based encryption scheme along with a hash key \( h_k \), and outputs these strings (for the prover) along with a tree hash of the ciphertexts (for the verifier). Since the encryption scheme we use has pseudorandom ciphertexts and the hash family may be chosen to have uniformly random keys, we may modify the SNARG.Gen algorithm to simply output a long, uniformly random string along with a tree hash of this string (this modification preserves non-adaptive soundness). This modified SNARG.Gen algorithm has no private randomness and thus these SNARGs have transparent setup.

3.5 Somewhere Extractable Batch Arguments (seBARGs)

The following preliminaries on seBARGs 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 3.6.** 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 seBARG for BatchIndexTMSAT consists of the following polynomial time algorithms:

\[ \text{Gen}(1^\lambda, 1^n, 1^m, k, i^*) \rightarrow (\text{crs}, \text{td}) \].

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

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

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

**Definition 3.7 (seBARG).** A somewhere-extractable batch argument system \text{seBARG} = (Gen, \mathcal{P}, \mathcal{V}, \text{Extract}) for \text{BatchIndexTMSAT} is required to satisfy the following properties:

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

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

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

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

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

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

\[
\Pr \left[ \mathcal{V}(\text{crs}, x, \pi) = 1 \ \land \ (x, i^*, w^*) \not\in \mathcal{R} \ : \ (\text{crs}, \text{td}) \leftarrow \text{Gen}(1^\lambda, 1^n, 1^m, i^*), \ (M, z, \pi) = A(\text{crs}), \ w^* \leftarrow \text{Extract}(\text{td}, \pi) \right] \leq \text{negl}(\lambda).
\]

**Remark 3.5.** We say that a seBARG scheme is \( T^* \)-secure, for \( T^* = T^*(\lambda) \), if the index hiding property and the somewhere argument of knowledge property hold w.r.t. a \text{poly}(T^*)-size adversary (as opposed to a \text{poly}(\lambda)-size), and the advantage probability is \( \text{negl}(T^*) \) (as opposed to \( \text{negl}(\lambda) \)). 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 or seBARG, we will implicitly mean a seBARG for \text{BatchIndexTMSAT}.

**Remark 3.6.** Given an seBARG, one can naturally extend the definition of the key generation algorithm \text{Gen} to take as input an index set \( I \subset [k] \), as opposed to a single index. \text{Gen}(1^\lambda, 1^n, 1^m, I) will simply run \text{Gen}(1^\lambda, 1^n, 1^m, i) for every \( i \in I \). The prover algorithm \( \mathcal{P} \), given a \text{crs} that encodes
the \(|I|\) indices, will simply generate \(|I|\) proofs (one for each \(\text{crs}\)), and the verifier will check these \(|I|\) proofs independently. In this situation, the \(\text{seBARG}\) trapdoor \(\text{td} = (\text{td}_1, \ldots, \text{td}_{|I|})\) is written in \(|I|\) components, and the scheme inherits a strong form of index hiding: given \(\text{seBARG} \cdot \text{crs}\), the \(j\)th index \(i_j\) remains hidden even given all \(|I|−1\) other trapdoors \(\text{td}_{j'}\).

**Theorem 3.8** ([CJJ22, WW22, HJKS22, KLVW23]). There exists an \(\text{seBARG}\) for \(\text{BatchIndexTMSAT}\) assuming \(\text{LWE}\) or \(\text{DLIN}\) or (subexponential \(\text{DDH}\) and \(\text{QR}\)).\(^{11}\)

The remaining preliminaries in Section 3 are due to [JJ22].

### 3.6 Propositional Logic Systems

We use extended Frege systems (denote as \(\mathcal{EF}\)) for propositional logic. Such a system is described by a set of \(\text{variables}\), a set of \(\text{connectives}\), and a set of \(\text{inference rules}\). \(\text{Variables}\) are the most basic elements, usually represented by letters such as \(x, y, z\). \(\text{Connectives}\) are used to connect variables. We only use two connectives \(\rightarrow\) and \(\neg\) for “imply” and “negation”, respectively. Other connectives such as \(\land, \lor, \oplus\) for “and”, “or”, “xor”, can be defined using \(\rightarrow\) and \(\neg\). We use \(\leftrightarrow\) to denote “if and only if”, and formally, \(a \leftrightarrow b\) is the abbreviation of \((a \rightarrow b) \land (b \rightarrow a)\). We use \(\top\) to represent “false”, and define “true” \(\bot\) as \(\neg\top\).

\(\text{Formulas}\) are defined inductively: \(\top\) (“false”) is a formula; any variable is a formula; if \(u, v\) are formulas, then \(u \rightarrow v, \neg u\) are formulas. A formula can be treated as a labeled tree, where leaves are labeled with variables, and internal nodes are labeled with connectives. A \(\text{subformula}\) of \(A\) is defined as a subtree of \(A\). We define the following complexity measures of formulas. For each formula \(A\), we define the \(\text{size}\) of \(A\) as the number of nodes in the tree. We denote \(\text{ldp}(A)\) as the \(\text{logical depth}\) of \(A\), which represents the depth of the tree. Formally, it can be inductively defined as follows: \(\text{ldp}(\top) = 0\); for any variable \(a\), \(\text{ldp}(a) = 0\); for any two formulas \(u, v\), \(\text{ldp}(u \rightarrow v) = 1 + \max(\text{ldp}(u), \text{ldp}(v))\), and \(\text{ldp}(\neg u) = 1 + \text{ldp}(u)\).

A \(\text{substitution}\) \(\sigma\) is a map from the set of variables to the set of formulas. If \(A\) is a formula, then the result of \(\text{applying}\) \(\sigma\) to \(A\) is denoted as \(A\sigma\), which is a formula obtained by replacing each occurrence of the variables in \(A\) by its image under \(\sigma\). For example, let \(A = p \rightarrow (q \rightarrow p)\) and let a substitution \(\sigma\) be \(p \mapsto a \land b, q \mapsto a \lor b\), then \(A\sigma = (a \land b) \rightarrow ((a \lor b) \rightarrow (a \land b))\).

A \(\text{Frege system}\) is specified by a set of \(\text{inference rules}\). Each inference rule is defined as \(A_1, A_2, \ldots, A_k \vdash A_0\), where \(A_0, A_1, \ldots, A_k\) are formulas. Intuitively, it means that “if \(A_1, A_2, \ldots, A_k\) are valid, then \(A_0\) is also valid”. If \(k = 0\), then we say such an inference rule is an \(\text{axiom}\).

In this work, we use the following set of axioms and modus ponens as inference rules for propositional logic.

- **Axiom 1**: \(p \rightarrow (q \rightarrow p)\)
- **Axiom 2**: \((p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))\)
- **Axiom 3**: \(\neg\neg p \rightarrow p\)
- **Modus Ponens**: \(p, p \rightarrow q \vdash q\)

\(^{11}\)The work [CGJ⁺23] also constructs \(\text{seBARGs}\) based on subexponential DDH, but not with efficiency parameters matching our definition. Relatedly, the \(\text{seBARGs}\) of [CGJ⁺23] cannot be sub-exponentially secure and therefore cannot be used in our Section 5 as written.
In a proposition $p_1, p_2, \ldots, p_n \vdash q$, we refer to $p_1, p_2, \ldots, p_n$ as the premise and refer to $q$ as the conclusion. A derivation for the proposition $p_1, p_2, \ldots, p_n \vdash q$ is a series of formulas $\theta_1, \theta_2, \ldots, \theta_\ell$ with $\theta_\ell = q$, and for each $i \in [\ell]$, $\theta_i$ is either

- A premise $p_j$ with $j \in [k]$, or
- $A_0\sigma$, where $A_1, A_2, \ldots, A_k \vdash A_0$ is an inference rule, $\sigma$ is a substitution, and $\{A_1\sigma, A_2\sigma, \ldots, A_k\sigma\}$ are a subset of the formulas $\{\theta_1, \theta_2, \ldots, \theta_{i-1}\}$.

A proof is a derivation with no premise.

The extended Frege system denoted as $\mathcal{EF}$ is a logic system that additionally has the following extension axioms. Namely, for a derivation $(\theta_1, \theta_2, \ldots, \theta_\ell)$ in $\mathcal{EF}$, for each $i \in [\ell]$, $\theta_i$ needs to satisfy the aforementioned constraint or $\theta_i$ is of the form $t \leftrightarrow A$, where $A$ is a formula, and $t$ is a new variable that has not occurred in $\theta_1, \theta_2, \ldots, \theta_{i-1}$, and also does not occur in $A$. We define the size of a derivation $(\theta_1, \theta_2, \ldots, \theta_\ell)$ as the summation of the sizes of the formulas $\theta_1, \theta_2, \ldots, \theta_\ell$.

Remark 3.7. First note that since the axioms and Modus Ponens are of constant size, each line in the proof (i.e., each formula $\theta_i$) depends on only a constant number of previous lines. Second, we can assume without loss of generality that each line of the proof is of constant size by using the extension axiom.

3.7 Cook’s Theory $PV$

Cook introduced a theory $PV$ [Coo75] to capture the intuition of feasibly constructive proofs (i.e. polynomial-time reasoning). $PV$ is an equational theory, i.e. each statement in $PV$ asserts that two terms are equal. Moreover, it allows the introduction of new function symbols by recursive definition (i.e. Cobham’s definition of polynomial-time functions [Cob65]). Hence, any polynomial-time function is definable in $PV$ [Coo75]. Moreover, common used arithmetical operations such as addition, multiplication, and modulus functions can also be defined in $PV$. Their related properties such as commutative law, associative law etc. can be proven in $PV$ [Bus86].

Formally, Cook’s theory $PV$ [Coo75] is defined as follows. $PV$ works on the natural numbers that are represented in the dyadic notation, where any natural number $x$ is uniquely represented as a finite string of integers in $\{1, 2\}^\ast$. Specifically, we represent $x$ as the string $x_nx_{n-1}x_{n-2}\ldots x_1 \in \{1, 2\}^\ast$, if $\sum_{i=1}^n x_i 2^i = x$, and use an empty string to represent 0. It’s easy to see that such presentation is unique for any natural number. The function $s_i(x) = 2x + i$, $i = 1, 2$ appends $i$ to the string $x$. Hence, we also denote $s_i(x)$ as $x \upharpoonright i$.

We introduce the following terminologies. Terms are defined inductively as follows: any variable is a term; any function symbol of arity 0 is a term; if $t_1, t_2, \ldots, t_k$ are terms, and $f$ is a function symbol, then $f(t_1, t_2, \ldots, t_k)$ is a term. Equations are of the form $t = u$, where both $t$ and $u$ are terms. A derivation for the statement $E_1, E_2, \ldots, E_n \vdash_{PV} E$ in $PV$ is a series of equations $D_1, D_2, \ldots, D_\ell$ such that $D_\ell = E$ and for any $i \in [\ell]$, the equation $D_i$ is either a premise $E_j (j \in [n])$, or a defining equation for some function symbol that we will introduce later, or follows from some inference rule that we will introduce later. A proof in $PV$ is a derivation with no premise ($n = 0$).

Introducing Function Symbols. A new function symbol $f$ can be introduced in $PV$ in the following two ways. The first way is to define

$$f(x_1, x_2, \ldots, x_k) = t,$$
where $t$ is a term with variables $x_1, x_2, \ldots, x_k$.

The second way is to recursively define the function on the dyadic notion (i.e. Cobham’s characterization of polynomial-time functions [Cob65]). Specifically, for existing function symbols $g, h_1, h_2, k_1, k_2$ in $PV$, define the following equations as defining equations

$$f(0, y) = g(y), \quad f(x||i, y) = h_i(x, y, f(x, y)), i = 1, 2,$$

where $y = (y_1, \ldots, y_k)$ is a series of $k$ variables. Then how $f$ is computed for any $x, y$ is fully specified. To avoid any undecidable issue, $PV$ further requires that the output length of $f$ is bounded by a polynomial. To ensure this, Cook requires that “$\lvert h_i(x, y, z) \rvert \leq \lvert z \rvert + \lvert k_i(x, y) \rvert$” is provable in $PV$, where $\lvert \cdot \rvert$ is the length of the dyadic presentation. To achieve this, Cook introduced the LESS function, and it is defined with other initial functions as follows. $s_i, i = 1, 2$ has no defining functions. 0 is also function symbol with arity 0, and has no defining function.

- TR: $TR(0) = 0, TR(x||i) = x, i = 1, 2$. It cuts off the least significant digit in the dyadic notion.
- $\times$: $\times(x, 0) = x, \times(x, y||i) = s_i(x, y), i = 1, 2$. It concatenates the string $x$ and $y$.
- $\oplus$: $\oplus(x, 0) = x, \oplus(x, y||i) = \times(x, \oplus(x, y)), i = 1, 2$. It concatenates $\lvert y \rvert$ copies of $x$.
- LESS : LESS$(x, 0) = x, LESS(x, y||i) = TR(LESS(x, y)), i = 1, 2$. It cuts off the $\lvert y \rvert$ right most digits of $x$ in the dyadic notion. Then we can use LESS$(x, y) = 0$ to express $\lvert x \rvert \leq \lvert y \rvert$.

To complete the definition of function $f$, $PV$ requires two proofs $\pi_1, \pi_2$ in $PV$ for LESS$(h_i(x, y, z), z \star k_i(x, y)) = 0, i = 1, 2$. Then a function symbol $f$ is defined as the tuple $(g, h_1, h_2, k_1, k_2, \pi_1, \pi_2)$.

The inference rules are in the following. Here, $t, u, v$ are any terms, $x$ is any variable, and $y = (y_1, y_2, \ldots, y_k)$ is any tuple of $k \geq 0$ variables. $f$ is any function symbol (we will define later).

- $R_1$: $t = u \vdash u = t$.
- $R_2$: $t = u, u = v \vdash t = u$
- $R_3$: $t_1 = u_1, t_2 = u_2, \ldots, t_k = u_k \vdash f(t_1, t_2, \ldots, t_k) = f(u_1, u_2, \ldots, u_k)$.
- $R_4$: $t = u \vdash t(v/x) = u(v/x)$. Here, the notation “$t(v/x)$” means replacing each occurrence of the variable $x$ with the term $v$. “$u(v/x)$” is defined in the same way.
- $R_5$: $E_1, E_2, \ldots, E_6 \vdash f_1(x, y) = f_2(x, y)$, where $E_1, E_2, \ldots, E_6$ are the defining equations 1 for $f_1, f_2$, with the same function symbols $g, h_1, h_2$.

**Propositional Translation.** In the same work [Coo75], Cook showed that any proofs in $PV$ can be translated to polynomial size propositional logic proofs. The original theorem statement uses extended resolutions logic. Later [CR79] showed that extended resolution and extended Frege system are essentially equivalent in terms of proof size. For simplicity, we use extended Frege system in this work, and state Cook’s result in extended Frege system.

Before we formally state the theorem, we first describe how to transform a theorem statement in $PV$ to proposition logic. The idea is to use variables in $\mathcal{EF}$ to present each digit in the dyadic notation. Specifically, let $m$ be an integer. For each term $t$ in $PV$, let $P_0[t], P_1[t], \ldots, P_m[t]$ and
For the easy of representation, in this work we use the following notation $V_m[t]$ to denote the variables \( \{P_i[t], Q_i[t]\}_{i=1}^m \) corresponds to \( t \). For each term \( t \), one can associate it with a proposition formula \( \text{prop}_m[t] \), asserting $V_m[t]$ is computed correctly from the variables $V_m[p_1], \ldots, V_m[p_k]$, where $p_1, p_2, \ldots, p_k$ are all variables appear in $t$. For any variable $x$, $\text{prop}_m[x]$ is the formula asserting $V_m[x]$ is well-formed, i.e. $\neg Q_i[x]$ implies $\neg Q_{i+1}[x]$ for $i \in [m-1]$. The definition of $\text{prop}_m[t]$ can be inductively defined for any term $t$. For more details, see [Coo75].

For any integer $n$, if the computation of all terms in the proof only needs $m$ dyadic digits, then $m$ is called a bounding value. For any equation $t = u$, where $t$ and $u$ are both terms. \([t = u]_m^n\) is defined as the propositional formula asserting that if the variables in $t$ are all less than $n$ digit, then the value of $t$ and $u$ are equal. For its formal definition, see [Coo75].

Next, we present the theorem statement for Cook’s propositional translation.

**Theorem 3.9 (Corollary of ER Simulation Theorem in [Coo75]).** For any two terms $t$ and $u$, and any $n$ and any polynomial bounding value $m = m(n)$, if $\vdash_{PV} t = u$, then $[t = u]_m^n$ has polynomial size logic proofs in extended Frege logic.

The idea of Theorem 3.9 is to do an induction on the length of the proofs in $PV$, and translate each step of the proof in $PV$ to a polynomial size proof in the extended Frege.

### 3.8 Theory $PV_1$

In the same work [Coo75], Cook also introduced a theory $PV_1$ in which formalizing proofs is easier than $PV$. [Coo75] showed that the theory $PV_1$ is a conservative extension of $PV$, which means that any theorem proven in $PV_1$ can also be proven in $PV$. Hence, in this work, we do not distinguish $PV$ and $PV_1$.

The theory $PV_1$ contains all variables, function symbols, and terms in $PV$. Furthermore, it contains formulas, which is either equations, or truth-functional combinations of equations, using \(\land, \lor, \neg, \rightarrow, \leftrightarrow\), which express “and”, “or”, “negation”, “imply”, and “equivalent”.

The axioms of $PV_1$ are defined as follows. Here, $x$ is a variable, $t, u, t_i, u_i$ are terms.

- **$E_1$:** $t = t$
- **$E_2$:** $t = u \rightarrow u = t$
- **$E_3$:** $t = u \land u = v \rightarrow t = v$
- **$E_4$:** $\left( t_1 = u_1 \land \ldots \land t_k = u_k \right) \rightarrow f(t_1, \ldots, t_k) = f(u_1, \ldots, u_k)$, where $f$ is a function symbol in $PV$.
- **$E_5$:** $x = y \leftrightarrow x \| i = y \| i$, $i = 1, 2$
- **$E_6$:** $\neg(x \| 1 = x \| 2)$
$E_7$: $\neg(0 = x||i), \ i = 1, 2$

- **Defining Functions**: Defining equations for any function symbol in $PV$ is axioms in $PV_1$. Moreover, $PV_1$ allows defining multi-variable functions via recursion as follows. Let $g_{00}, g_{01}, g_{10}, \{h_{ij}, k_{ij}\}_{i,j \in \{1,2\}}$ be function symbols. Then a new function symbol $f$ can be defined by the following defining equations.

$$
\begin{align*}
    f(0,0,z) &= g_{00}(z) \\
    f(0,y||j,z) &= g_{01}(z), \ j = 1, 2 \\
    f(x||i,0,z) &= g_{10}(z), \ i = 1, 2 \\
    f(x||i,y||j,z) &= h_{ij}(x,y,z, f(x,y,z)), \ i, j \in \{1, 2\}
\end{align*}
$$

Moreover, $\text{LESS}(h_{ij}(x,y,z,u), u \ast k_{ij}(x,y,z)) = 0, \ i,j \in \{1,2\}$ needs to be provable in $PV$. Finally, the defining of the initial functions $\text{TR}, \ast, \odot, \text{LESS}$ are also axioms of $PV_1$.

- **Tautology**: The truth-functionally valid formulas of $PV_1$ are axioms.

The inference rules of $PV_1$ are as follows.

- **Substitution**: $A \vdash A(t/x)$, where $A$ is a formula in $PV_1$, $t$ is a term and $x$ is a variable. Here, we use $A(t/x)$ to denote the formula $A\sigma$, where $\sigma$ is the substitution $x \mapsto t$.

- **Implication**: $A_1, A_2, \ldots, A_k \vdash B$, where the formula $B$ is a truth-functional implication of formulas $A_1, \ldots A_k$.

- **$k$-Induction**: $\{A(0/x_i)\}_{i \in [k]}, \{A \rightarrow A(x_1||j_1/x_1, \ldots, x_k||j_k/x_k)\}_{j_1, j_2, \ldots, j_k \in \{1, 2\}} \vdash A$, where $A$ is a formula of the variables $x_1, \ldots, x_k$.

## 4 SNARGs from Locally Unsatisfiable Circuit Extensions

In this section, we define the new notion of locally unsatisfiable extension (Definition 4.4) and construct a SNARG for any $NP$ language with a locally unsatisfiable extension of polynomial size (Theorem 4.8). In Section 5, we extend this result to handle certain types of superpolynomial-size extension circuits.

### 4.1 Local Unsatisfiability

We begin by recalling the definition of a local assignment generator [PR17].

**Definition 4.1 (Local Assignment Generator)**. Let $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$ denote a Boolean circuit of size $s$ where $n = n(\lambda), m = m(\lambda)$ and $s = s(\lambda)$. For $\ell = \ell(\lambda)$ and $\varepsilon = \varepsilon(\lambda)$, we say that $C$ has an $(\ell, \varepsilon)$-local assignment generator if there is a (not necessarily efficient) algorithm $\text{LocalGen}$ with the following properties.

- **Syntax**: The input to $\text{LocalGen}$ is the description of a subset of wires $T \subseteq [s]$ of size at most $\ell$ and its output is an assignment $(\sigma_i)_{i \in T} \in \{0, 1\}^T$ to the corresponding wires of $C$.  

25
• **Local Consistency:** for any gate $g_i = (i,j,k,f)$ of $C$ connecting input wires $j,k$ to an output wire $i$, and any set $T \supseteq \{i,j,k\}$ of size at most $\ell$, we have
\[
\Pr \left[ \sigma_i \neq f(\sigma_j, \sigma_k) : (\sigma_\alpha)_{\alpha \in T} \leftarrow \text{LocalGen}(T) \right] \leq \varepsilon.
\]
where the probability is over the randomness of LocalGen.

• **Computational Non-Signaling:** for any sets $T_0, T_1 \subseteq [s]$ of size at most $\ell$, the following distributions are $\epsilon$-computationally indistinguishable:
\[
\left(\{\sigma_i\}_{i \in T_0 \cap T_1} : \{\sigma_i\}_{i \in T_0} \leftarrow \text{LocalGen}(T_0) \right) \approx_{\epsilon, \ell} \left(\{\sigma_i\}_{i \in T_0 \cap T_1} : \{\sigma_i\}_{i \in T_1} \leftarrow \text{LocalGen}(T_1) \right)
\]

**An Equivalent Formulation.** Definition 4.1 defines local assignment generators to take as input a set $T$ of wire values. We briefly discuss an equivalent view of local assignment generators with slightly different syntax, which we make use of in Section 4.5.

**Definition 4.2** (Local Tuple Assignment Generator). In the same setting as Definition 4.1, a $(\ell, \epsilon)$-local tuple assignment generator LocalGen for a circuit $C$ has the following properties.

• **Syntax:** The input to LocalGen is the description of an $\ell$-tuple of wires $(i_1,\ldots,i_\ell) \in [s]^{\ell}$. Its output is an assignment $(\sigma_{i_1},\ldots,\sigma_{i_\ell}) \in \{0,1\}^{\ell}$ to the corresponding wires of $C$.

• **Wire Consistency:** for any $\ell$-tuple $(i_1,\ldots,i_\ell)$ and any pair $j_1,j_2$ such that $i_{j_1} = i_{j_2}$, we have
\[
\Pr \left[ \sigma_{i_{j_1}} \neq \sigma_{i_{j_2}} : (\sigma_{i_j})_{j=1}^\ell \leftarrow \text{LocalGen}(i_1,\ldots,i_\ell) \right] \leq \varepsilon.
\]

• **Gate Consistency:** for any tuple $(i_1, \ldots, i_\ell)$ and triple $(j_1, j_2, j_3)$, if $(i_{j_1}, i_{j_2}, i_{j_3})$ form a gate of $C$ with corresponding function $f$, we have
\[
\Pr \left[ \sigma_{i_{j_1}} \neq f(\sigma_{i_{j_2}}, \sigma_{i_{j_3}}) : (\sigma_{i_j})_{j=1}^{\ell} \leftarrow \text{LocalGen}(i_1,\ldots,i_\ell) \right] \leq \varepsilon.
\]

• **Computational Non-Signaling:** for any pair of $\ell$-tuples $(i_1, \ldots, i_\ell), (i_1', \ldots, i_\ell')$, let $T = \{j \in [\ell] : i_j = i_j'\}$. Then, the following distributions are $\epsilon$-computationally indistinguishable:
\[
\left(\{\sigma_{i_j}\}_{j \in T} : \{\sigma_{i_j}\}_{j \in [\ell]} \leftarrow \text{LocalGen}(i_1,\ldots,i_\ell) \right) \approx_{\epsilon, \ell} \left(\{\sigma_{i_j'}\}_{j \in T} : \{\sigma_{i_j'}\}_{j \in [\ell]} \leftarrow \text{LocalGen}(i_1',\ldots,i_\ell') \right)
\]

The key difference between Definition 4.2 and Definition 4.1 is that the non-signaling property in Definition 4.2 is defined with respect to tuple indices rather than sets of wires (and is therefore weaker). To compensate for this, the wire consistency property implies that a tuple assignment is roughly order-invariant.

**Lemma 4.3.** If a circuit $C$ has a $(\ell, \epsilon)$-local tuple assignment generator, then it also has a $(\ell, O(\ell \cdot \epsilon))$-local assignment generator.

**Proof.** Given a tuple local assignment generator LocalGen$(i_1,\ldots,i_\ell)$, we can define a (standard) local assignment generator LocalGen'$(T)$ that converts a set $T$ into an $\ell$-tuple $(i_1,\ldots,i_\ell)$ listing the elements of $T$ in lexicographic order, including “dummy” copies $i_{j+1},\ldots,i_\ell = s$ if $|T| = j < \ell$. Local consistency follows immediately from the gate consistency property of LocalGen.
Computational non-signaling of $\text{LocalGen}'$ follows from a hybrid argument. Given sets $T, T'$, let $(i_1, \ldots, i_\ell)$ and $(i'_1, \ldots, i'_\ell)$ denote their corresponding $\ell$-tuples, and let $\pi$ denote a permutation on $[\ell]$ that matches the indices of $T \cap T'$ between these two $\ell$-tuples. Then $\pi$ can be written as a composition of at most $\ell$ transpositions. For every $0 \leq r \leq \ell$, let $\pi_r$ denote the composition of the first $j$ such transpositions, so that $\pi_\ell = \pi$. Then, one can show inductively that

$$(\{\sigma_{i_j}\}_{i_j\in T \cap T'} : \{\sigma_{i_j}\}_{j\in[\ell]} \leftarrow \text{LocalGen}(i_1, \ldots, i_\ell)) \approx_{\ell, O(\varepsilon)}$$

$$(\{\sigma_{i_{\pi_r(j)}}\}_{i_{\pi_r(j)}\in T \cap T'} : \{\sigma_{i_{\pi_r(j)}}\}_{j\in[\ell]} \leftarrow \text{LocalGen}(\pi_r(i_1), \ldots, \pi_r(i_\ell))),$$

where the inductive step requires a constant number of invocations of the gate consistency and computational non-signaling of $\text{LocalGen}$.

We now proceed to define a new object related to local assignment generators: family of locally unsatisfiable circuit extensions.

**Definition 4.4 (Locally Unsatisfiable Extension).** A circuit family $\{C_x\}_{x \in \{0,1\}^n}$ is said to have an $\ell$-locally unsatisfiable extension of size $s = s(\lambda)$ if there exists a family of circuits $\{E_x\}_{x \in \{0,1\}^n}$ satisfying the following properties:

- For every $x \in \{0,1\}^n$, $E_x$ is a size-$s$ extension of $C_x$, meaning that the gates of $E_x$ consist of (a) the gates of $C_x$; and (b) the gates corresponding to the evaluation of some circuit $D_x(\tau)$, where $\tau$ denotes the values of all the wires of $C_x(w)$.
  
  In particular, the output of $E_x(w)$ is identical to that of $C_x(w)$; there are simply additional (unnecessary) “intermediate” gates.

- For every $x \in \{0,1\}^{n(\lambda)}$ and every $\varepsilon = \varepsilon(\lambda)$, if $C_x$ is unsatisfiable, then for any $(\ell, \varepsilon)$-local assignment generator $\text{LocalGen}$ for $E_x$ such that

$$\Pr[\sigma_{\text{out}} = 1 : \sigma_{\text{out}} \leftarrow \text{LocalGen}(\{\text{out}\})] \leq \text{poly}(\lambda, s) \cdot \varepsilon$$

where $\text{out}$ denotes the output wire of $E_x$ (which is also the output wire of $C_x$) and where the probability is over the randomness of $\text{LocalGen}$.

### 4.2 Encoded Computation

In this section, we describe a variant of encoded computation that captures the “augmented circuit” construction of [KRR14]. We state two local unsatisfiability properties that the encoded computation satisfies; the first of these properties is proved directly in [KRR14], while the second has not been previously stated but follows fairly straightforwardly from [KRR14].

**Theorem 4.5.** There exists a polynomial $p$ and an efficient (extension) function $\text{Ext}$ that takes any circuit $C : \{0,1\}^n \rightarrow \{0,1\}$ and converts it into a new circuit $C_{\text{Ext}} : \{0,1\}^n \rightarrow \{0,1\}$ such that for every circuit $C : \{0,1\}^n \rightarrow \{0,1\}$ and every $\lambda \in \mathbb{N}$ the following holds:

1. $|C_{\text{Ext}}| \leq p(|C|)$.  

27
2. Let $\ell = p(\log |C|)$, and suppose that $\text{LocalGen}$ is a $(\ell, \epsilon)$ local assignment generator for $C_{\text{Ext}}$ whose input wires are deterministic; that is, there exists a string $x \in \{0, 1\}^n$ such that for any set of input wires $T \subseteq [n]$ of size at most $\ell$ it holds that

$$\Pr[\text{LocalGen}(T) \neq x_T] \leq \epsilon.$$ 

Then,

$$\Pr[\text{LocalGen}(\text{out}) \neq C(x)] \leq \text{poly}(|C|) \cdot \epsilon.$$ 

3. Define the “double circuit” $C_{\text{Ext}}^{(2)}$ to consist of two parallel copies of $C_{\text{Ext}}$ that share their input wires. We define $C_{\text{Ext}}^{(2)}$ to have two output wires, $\text{out}_0$ and $\text{out}_1$. For $\ell = p(\log |C|)$ as above, suppose that $\text{LocalGen}$ is an $(2\ell, \epsilon)$ local assignment generator for $C_{\text{Ext}}^{(2)}$. Then, for any set $T \supset \{\text{out}_0, \text{out}_1\}$, of size at most $2\ell$,

$$\Pr[\sigma_{\text{out}_0} \neq \sigma_{\text{out}_1} : \{\sigma_i\}_{i \in T} \leftarrow \text{LocalGen}(T)] \leq \text{poly}(|C|, \lambda) \cdot \epsilon.$$ 

Property (3) above is (up to a factor of 2 in the locality) a strict generalization of property (2), which is only stated for convenience. Unlike property (2), it gives a meaningful guarantee for certain kinds of non-deterministic computations.

**Proof.** Let $\text{Ext}$ denote the augmented circuit construction from [KRR14]. Properties (1) and (2) of this construction are proved directly in [KRR14].

To establish property (3), we make use of two key properties of $\text{Ext}$. The first of these properties is a strengthening of property (2). For any Boolean circuit $E : \{0, 1\}^n \rightarrow \{0, 1\}$ and any input $x \in \{0, 1\}^n$, we define a weak $(\ell, \epsilon)$-local assignment generator for $E$ with respect to $x$ to be a PPT algorithm $\text{LocalGen}$ with the same syntax as in Definition 4.1, satisfying the same computational non-signaling property as in Definition 4.1, but satisfying a relaxed local consistency property:

- **Relaxed Local Consistency w.r.t. $x$:** for any gate $g_i = (i, j, k, f)$ of $E$ connecting input wires $j, k$ to an output wire $i$, and any set $T \supset \{i, j, k\}$ of size at most $\ell$, we have

$$\Pr[\sigma_j = g_j(x) \land \sigma_k = g_k(x) \land \sigma_i \neq g_i(x) : (\sigma_\alpha)_{\alpha \in T} \leftarrow \text{LocalGen}(T)] \leq \epsilon.$$ 

where the probability is over the randomness of $\text{LocalGen}$. Moreover, we require that for any input wire $i \in [n]$ and any set $T \supset \{i\}$ of size at most $\ell$, we have

$$\Pr[\sigma_i \neq x_i : (\sigma_\alpha)_{\alpha \in T} \leftarrow \text{LocalGen}(T)] \leq \epsilon.$$ 

We then make use of the following fact about the [KRR14] augmented circuit (for appropriately chosen $\ell = \text{poly}(\log |C|)$):

**Fact 4.6.** Suppose that $\text{LocalGen}$ is a weak $(\ell, \epsilon)$ local assignment generator for $C_{\text{Ext}}$ with respect to an input $x \in \{0, 1\}^n$. Then,

$$\Pr[\text{LocalGen}(\text{out}) \neq C(x)] \leq \text{poly}(|C|) \cdot \epsilon.$$ 

28
The second fact about Ext that we make use of is structural:

**Fact 4.7** ([KRR14], Sections 9 and A). For any layered Boolean circuit \( C \), the circuit \( C_{\text{Ext}} \) consists of \( C \) along with a Boolean sub-circuit for each layer \( i \) of \( C \), whose input wires are the wires of the \( i \)th layer of \( C \) and whose gates all compute linear functions on their two input wires. Moreover, these linear functions depend only on the topology of \( C \) and do not depend on the gate description of \( C \).

To conclude property (3), fix any circuit \( C \) and consider any \((2\ell, \varepsilon)\) local assignment generator \( \text{LocalGen} \) for \( C^{(2)} \). We define a new circuit \( C^\perp \) as follows:

- \( C^\perp \) has the same topology as \( C \).
- For every gate \( g_i \) of \( C^\perp \), the Boolean function \( f : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\} \) is the OR function.

In total, this means that if the \( i \)th gate \( g_i \) of \( C \) is \((i, j, k, f)\), then the \( i \)th gate of \( C^\perp \) is \((i, j, k, \text{OR})\).

We claim that there is a weak \((\ell, 2\varepsilon)\) local assignment generator \( \text{LocalGen}^\perp \) for \((C^\perp)_{\text{Ext}}\) with respect to the input \( 0^n \). Indeed, for any subset \( T \) of size at most \( \ell \), we define \( \text{LocalGen}^\perp(T) \) as follows:

- Call \( \text{LocalGen}(T \times \{0, 1\}) \) to obtain values \( \{\sigma_{i,b}\}_{i \in T, b \in \{0, 1\}} \), where \((i, b)\) denotes the \( i \)th wire in the \( b \)th copy of \( C_{\text{Ext}} \) (note that if \( i \in [n] \), then the \((i, 0)\) and \((i, 1)\) wires refer to the same wire).
- Define \( \sigma_{i,1}^\perp = \sigma_{i,0} \oplus \sigma_{i,1} \).

By construction, we have that \( \sigma_{i}^\perp = 0 \) (with probability 1) for all \( i \in [n] \), because the two copies of \( C_{\text{Ext}} \) in \( C^{(2)} \) share their input wires. Next, we show that for all gates \( g_i = (i, j, k, f) \) in the circuit \((C^\perp)_{\text{Ext}}\) and all \( T \supseteq \{i, j, k\} \), we have that

\[
\Pr \left[ \sigma_{j}^\perp = g_j^\perp(0^n) \land \sigma_{k}^\perp = g_k^\perp(0^n) \land \sigma_{i}^\perp \neq g_i^\perp(0^n) : (\sigma_{\alpha}^\perp)_{\alpha \in T} \leftarrow \text{LocalGen}^\perp(T) \right] \leq 2\varepsilon.
\]

To show this, we make use of the fact that \( g_i^\perp(0^n) = 0 \) for every \( i \), which is true because:

- For every gate \( i \) of the original circuit \( C^\perp \), it holds that \( g_i^\perp(0^n) = 0 \) (because \( g_i^\perp \) is computed by repeatedly applying an OR to two previously computed values).
- For every gate in the "extended" part of \((C^\perp)_{\text{Ext}}\), we know that the output wire of this gate is a linear (note: not affine) function of the wires of \( C^\perp \), which is therefore equal to zero.

To prove the weak local consistency property of \( \text{LocalGen}^\perp \), we consider two cases:

- If \( g_i^\perp \) is a gate of \( C^\perp \), then the event that \( \sigma_{i}^\perp = 0 \) is equivalent to the statement that \( \sigma_{i,0} = \sigma_{i,1} \) for \( \{\sigma_{\alpha,b}\} \leftarrow \text{LocalGen}(T \times \{0, 1\}) \). The same is true for \( \sigma_{j}^\perp \) and \( \sigma_{k}^\perp \). Therefore, our probability is upper bounded by \( 2\varepsilon \) by the local correctness of \( \text{LocalGen} \) w.r.t. the gates \( g_{i,0} \) and \( g_{i,1} \).
- If \( g_i^\perp \) is a gate of the extended part of \((C^\perp)_{\text{Ext}}\), then we know that \( g_i^\perp \) is supposed to compute a linear function \( \phi_i \) of the \( j \)th and \( k \)th wire values of \((C^\perp)_{\text{Ext}}\). Moreover, by Fact 4.7, the corresponding gate \( g_i \) of \( C_{\text{Ext}} \) is supposed to compute the same linear function \( \phi_i \) on the \( j \)th and \( k \)th wire values of \( C_{\text{Ext}} \). Thus, our probability is again upper bounded by \( 2\varepsilon \) by the local correctness of \( \text{LocalGen} \) w.r.t. the gates \( g_{i,0} \) and \( g_{i,1} \).
We have therefore established that LocalGen\textsuperscript{⊥} is a weak local assignment generator for (C\textsuperscript{⊥})\textsubscript{Ext} with respect to the input 0\textsuperscript{n}. Thus, by Fact 4.6, we conclude that
\[ \Pr[\text{LocalGen}^{\bot}(\text{out}) \neq 0] \leq \poly(|C|) \cdot \epsilon, \]
which is the desired conclusion because LocalGen\textsuperscript{⊥}(out) = σ\textsubscript{out}_0 \oplus σ\textsubscript{out}_1 except with probability at most 2ε. This completes the proof of Theorem 4.5.

4.3 Main Theorem Statement

Our main theorem holds under the following setup.

Fix any NP language \( \mathcal{L} \subseteq \{0,1\}^* \) with a corresponding NP relation \( \mathcal{R}_\mathcal{L} \). Moreover, let \( C = \{C^{(n)}(x, w)\}_{n \in \mathbb{N}} \) be a circuit family deciding \( \mathcal{R}_\mathcal{L} \) and \( M_\mathcal{L} \) denote a polynomial-time Turing machine that on input \((1^n, i)\) outputs the \( i \)th gate of \( C^{(n)} \).

Suppose that the circuit family \( \mathcal{C} = \{C_x\}_{x \in \{0,1\}^*} \) has an \( \ell \)-locally unsatisfiable extension \( \{E_x\}_{x \in \{0,1\}^*} \) of polynomial size, where \( E_x \) is the composition of \( C_x \) with a circuit \( D_x \). Finally, let \( \text{aux}_x \) denote a polynomial-size description of \( D_x \), let \( R = R^{(n)} \) denote a circuit such that \( R(\text{aux}_x) \) outputs the canonical description \( (D_x) \), and let \( M_\mathcal{L} \) denote a polynomial \( \poly(n, s) \)-time Turing machine that on input \((n, i)\) outputs the \( i \)th gate of \( R \).\textsuperscript{12}

**Theorem 4.8.** Assume the existence of a leveled, gate-by-gate FHE scheme (Definition 3.3) as well as a seBARG scheme (Definition 3.7). Then, \( \mathcal{L} \) as above has a SNARG with the following properties:

- The proof length is \( \poly(\lambda, \ell, \log|C_x|) \).
- The crs length is \( \poly(\lambda) \cdot (|\text{aux}_x| + \text{Depth}(R) + \text{Depth}(E_x)) \).
- The scheme is non-adaptively sound.

**Remark 4.1.** Under the stronger assumption that a (gate-by-gate) unleveled FHE scheme exists, the crs length above can be improved to \( \poly(\lambda) \cdot |\text{aux}_x| \).

We prove Theorem 4.8 in several steps.

- We first describe our SNARG construction, and show that a (non-adaptive) adversarial prover convincing the verifier of a false statement \( x \notin L \) implies the existence of a local assignment generator for an encrypted universal computation \( \widehat{U}_{E_x} \), defined as follows:

**Definition 4.9** (Encrypted Universal Computation). \( \widehat{U}_{E_x} \) is the circuit that on input \( w \) computes the values all the wires of the circuit \( C_x(w) \), denoted by \( \tau = \tau(x, w) \), and then computes \( \text{FHE.Eval}(U_\tau, \widehat{D_x}) \), where

- \( U_\tau \) is the (explicit) universal circuit that on input \( \langle D_x \rangle \) computes \( U(\tau, \langle D_x \rangle) = D_x(\tau) \).
- \( \langle D_x \rangle \) is an encryption of \( \langle D_x \rangle \) that is hard-coded in the description of \( \widehat{U}_{E_x} \).

- We show that such a local assignment generator generically implies a local assignment generator for the unencrypted universal computation \( U_{E_x} \), which on input \( w \) computes the values all the wires of the circuit \( C_x(w) \), denoted by \( \tau \), and then evaluates \( U(\tau, \langle D_x \rangle) \).

\textsuperscript{12}For example, one can always set \( d_x = \langle D_x \rangle \) and define \( R \) to be the trivial “do nothing” circuit. However, the additional parameters allow use of a compressed representation of \( D_x \), which decreases the crs size in our SNARG.
We then show that this generically implies a local assignment generator for $E_x$.

The definition of a locally unsatisfiable extension then guarantees the soundness of our protocol.

### 4.4 SNARG Construction: Encrypted Hash-and-BARG

Let $L, C, M_L, \{E_x, D_x, aux_x\}_{x \in \{0,1\}^*}$, $R, M_C$ be as in the Theorem statement. For every $n \in \mathbb{N}$ and every $x \in \{0,1\}^n$, we denote by $s = s(n)$ the number of gates in the circuit $U_{D_x}$, and we denote by $S = S(n, \lambda)$ the number of gates in the circuit $\widehat{U_{D_x}}$.

We define a candidate SNARG for $L$ using the following building blocks:

- A hash family with local opening $HT = (HT.Gen, HT.Hash, HT.Open, HT.Verify)$, defined in Section 3.2.
- A leveled fully homomorphic encryption scheme $FHE = (FHE.Gen, FHE.Enc, FHE.Dec, FHE.GateEval, FHE.Eval)$ with gate-by-gate evaluation, defined in Section 3.3.
- A somewhere extractable batch argument system $seBARG$, defined in Section 3.5.

Our SNARG construction works as follows:

**The Setup Algorithm.** SNARG.Gen$(1^\lambda, 1^n)$ does the following:

1. Set the FHE depth parameter $d_{FHE} = \text{poly}(\lambda, \log s) \cdot (\text{Depth}(R) + \text{Depth}(E_x))$ for a polynomial $\text{poly}$ specified in the description of the Prover algorithm.
2. Sample FHE keys $(pk, \vec{ek}, \vec{sk}) \leftarrow FHE.Gen(1^\lambda, 1^{d_{FHE}})$.
3. Sample a ciphertext $ct \leftarrow FHE.Enc(pk, 0 | aux_0 | n |)$.
4. Sample a hash key $hk \leftarrow HT.Gen(1^\lambda)$ and compute $v = \text{Hash}(hk, (\vec{ek}, ct))$.
5. Sample $(\text{crs}_{seBARG}, \text{td}_{seBARG}) \leftarrow seBARG.Gen(1^\lambda, 1^{n^*}, 1^{m^*}, k, (i^*_1, \ldots, i^*_L))$, where
   - The parameters $n^* = \text{poly}(\lambda), m^* = \text{poly}(\lambda)$ and $k = \text{poly}(s)$ are all defined below (when we describe the prover algorithm).
   - $L = \ell \cdot \text{poly}(\lambda)$ is roughly the length of $\ell$ FHE ciphertexts.
   - $i^*_1, \ldots, i^*_L$ are arbitrary indices in $[k]$. We arbitrarily set $(i^*_1, \ldots, i^*_L) = (1, \ldots, L)$.

The common reference string is set to be $\text{crs} = (\text{crs}_P, \text{crs}_V)$, where

$$\text{crs}_P = (hk, \vec{ek}, ct, \text{crs}_{seBARG})$$

and

$$\text{crs}_V = (hk, v, \text{crs}_{seBARG}).$$
The Prover Algorithm. \textsc{SNARG}_P(crs_P, x, w) operates as follows.

1. Parse \(\text{crs}_P = (\text{hk}, \vec{e}k, \text{ct}, \text{crs}_{\text{seBARG}})\).

2. Run the Turing machine \(M_C\) repeatedly to generate the full gate-by-gate description \(\langle R \rangle\) of the circuit \(R\).

3. Given \(w\), compute \(C_x(w)\) and let \(\tau = \tau_x(w)\) denote the wire assignment for this computation.

4. Run the (uniformly described) circuit \(\left( \text{FHE.Eval}_{\vec{e}k}(U_{\langle R \rangle}, \cdot) \right)_{\text{Ext}}\) on \(\text{ct}\), gate-by-gate, where \(U_{\langle R \rangle}\) expands an input \(\text{aux}\) into a circuit description \(\langle D \rangle\) and \(\text{Ext}\) denotes the extended circuit from Theorem 4.5. Note that this can be described as the execution of a sequence of (extended) circuits that each evaluate a single gate homomorphically. This results in an intermediate \(\text{FHE} \) ciphertext \(\text{ct}'\) that the prover continues to operate on in the next step.

5. Run the (uniformly described) circuit \(\text{FHE.Eval}_{\vec{e}k}(U_{\tau}, \cdot)\) on \(\text{ct}'\), gate-by-gate, where \(U_{\tau}(\langle D \rangle) = D(\tau)\) is another universal circuit. The depth parameter \(d_{\text{FHE}}\) is chosen to be the depth of the computation homomorphically evaluated in Steps 4-5.

6. Let \(\vec{\tau}\) denote the wire assignment for the gate-by-gate homomorphic computation in Steps 4-5.

7. Compute \(v_{\text{inp}} = \text{Hash(hk, } x\text{)}\) and \(v_{\text{P}} = \text{Hash(hk, (} \langle R \rangle, v_{\text{inp}}, \tau, \vec{\tau}\text{)})\). In the following description, we define an \emph{opening} of \(v_P\) to a bit \(b\) in location \(i\) of the string \((\langle R \rangle, x, \tau, \vec{\tau})\) as follows:
   - If \(i\) corresponds to a wire of \(x\), an opening in location \(i\) consists of an opening of \(v_P\) to \(v_{\text{inp}}\) along with an opening of \(v_{\text{inp}}\) to \(b\) in location \(i\).
   - Otherwise, we use the usual notion of an opening of \(v_P\) w.r.t. \text{Hash(hk, \cdot)}.

8. Compute an \text{seBARG} proof for the (informal) claims that (1) every bit of \(\langle R \rangle\) was computed correctly, and that (2) every gate of \(\tau, \vec{\tau}\) was computed correctly. Formally, let \(x^* = (M, z, k, T)\), where
   
   - \(k = |\tau| + |\vec{\tau}| + |\langle R \rangle|\)
   - \(z = (\text{hk}, v, v_P)\).
   - \(M\) is the Turing machine that on input \((z, i, w_i)\) does the following:
     - If \(1 \leq i \leq |\langle R \rangle|\), \(M\) checks that \(w_i\) is a valid local opening of \(v_P\) (in the \(i\)th location) to the \(i\)th bit of \(\langle R \rangle\) (as computed by \(M_C\)).
     - If \(i > |\langle R \rangle|\), \(M\) checks that \(w_i = (b_1, \rho_1, b_2, \rho_2, b_3, \rho_3)\) is a valid local opening of \((v, v_P)\) to three bits \(b_1, b_2, b_3\) in the locations \(i_1, i_2, i_3\), where \((i_1, i_2, i_3)\) are the \((i - |\langle R \rangle|)\)th gate\(^{13}\) of the circuit describing Steps 3-5.\(^{14}\) Moreover, \(M\) checks that \(b_1 = f_i(b_2, b_3)\) satisfies the \(i\)th gate equation.
   - \(T\) is the run-time of \(M\), \(n^* = |(M, z, k, T)|\) and \(m^* = 3(|\rho| + 1)\), where \(|\rho|\) is the size of one opening.

\(^{13}\)By construction, the description of this gate is computable by a fixed Turing Machine.

\(^{14}\)Note that this includes “input” wire values corresponding to the bits of \(e_k\) and \(c_t\).
Let
\[ \pi_{\text{seBARG}} \leftarrow \text{seBARG}.P(\text{crs}_{\text{seBARG}}, M, z, 1^T, (w_1, \ldots, w_k)) \]
where \( w_1, \ldots, w_k \) are honestly generated witnesses. Note that \( n^* \) and \( m^* \) are polynomial in \((\lambda, \log s)\), while \( T = \text{poly}(\lambda, s) \).

9. Send \( \pi = (v_P, \pi_{\text{seBARG}}, \rho_{\text{inp}}, \rho_{\text{out}}) \) to the verifier, where \( \rho_{\text{inp}} \) is an opening of \( v_P \) to \( v_{\text{inp}} \) and \( \rho_{\text{out}} \) is an opening of \( v_P \) to \( \tau_{\text{out}} \), the output bit of \( \tau \).

The Verifier Algorithm. \( \text{SNARG}.V(\text{crs}_V, x, \pi) \) does the following:

1. Parse \( \text{crs}_V = (hk, v, \text{crs}_{\text{seBARG}}) \) and \( \pi = (v_P, \pi_{\text{seBARG}}, \rho_{\text{inp}}, \rho_{\text{out}}) \).
2. Let \( z = (hk, v, v_P) \) and let \( M \) be the Turing machine defined in the Prover algorithm.
3. Check that \( \text{seBARG}.V(\text{crs}_{\text{seBARG}}, x^*, \pi_{\text{seBARG}}) = 1 \), where \( x^* = (M, z, k, T) \).
4. Compute \( v_{\text{inp}} = \text{HT.Hash}(hk, x) \) and check that \( \text{HT.Verify}(hk, v_P, T_{\text{inp}}, v_{\text{inp}}, \rho_{\text{inp}}) = 1 \), where \( T_{\text{inp}} \) is the subset of \([k]\) corresponding to \( v_{\text{inp}} \).
5. Check that \( \text{HT.Verify}(hk, v_P, \text{out}, \tau_{\text{out}}, \rho_{\text{out}}) = 1 \), for \( \tau_{\text{out}} = 1 \).

4.5 Analysis

We prove that the \( \text{SNARG} \) construction presented in Section 4.4 satisfies Theorem 4.8. To this end, let \( L, C, M_L, \{E_x, D_x, \text{aux}_x\}_{x \in \{0,1\}^*}, R, M_C \) be as in the Theorem statement.

The completeness guarantee follows immediately from the completeness guarantees of the underlying primitives. The efficiency guarantees can also be easily observed from the construction:

- The length of \( \text{crs}_P \) is \( \text{poly}(\lambda) \cdot (|\text{aux}_0| + d_{\text{FHE}}) + |\text{crs}_{\text{seBARG}}| \), where \( d_{\text{FHE}} \) is nearly linear\(^{15} \) in \( \text{Depth}(R) + \text{Depth}(D_x) \) and \( |\text{crs}_{\text{seBARG}}| = \text{poly}(\lambda, m^*) = \text{poly}(\lambda, \log s) \).
- The length of \( \text{crs}_V \) is \( \text{poly}(\lambda, \log s) \).
- The length of \( \pi \) is \( \text{poly}(\lambda, \log s) + |\pi_{\text{seBARG}}| = \text{poly}(\lambda, \log s, m^*) = \text{poly}(\lambda, \log s) \).
- The runtime of \( \text{SNARG}.V \) is \( \text{poly}(\lambda) \cdot n + \text{poly}(\lambda, |\pi|) \) by the efficiency property of \( \text{seBARG} \).

We thus focus on proving the non-adaptive soundness, which follows from the following four lemmas. Let \( C_P[x, \langle R \rangle, ek, ct] \) denote the following circuit:

- Input: the input wires of \( C_P \) correspond to a witness \( w \) along with hard-coded inputs \((x, ek, ct)\).
- Run Steps 3-5 of \( \text{SNARG}.P \).
- The output gate is the output gate of \( C(x, w) \).

\(^{15}\)Recall that our universal circuits preserve circuit depth up to a logarithmic factor.
Lemma 4.10. Let \( \ell \) be larger than some absolute constant. Suppose that there exists a \( \text{poly}(\lambda) \)-size prover \( \mathcal{P}^* \) and a non-negligible function \( \epsilon = \epsilon(\lambda) \), and for every \( \lambda \in \mathbb{N} \) there exists \( x^* = x^*_\lambda \notin \mathcal{L} \) of size \( n(\lambda) \) such that

\[
\Pr[\text{SNARG}.\mathcal{V}^{\text{crs}_{\mathcal{V}}, x^*, \pi^*} = 1 : (\text{crs}_{\mathcal{P}}, \text{crs}_{\mathcal{V}}) \leftarrow \text{SNARG}.\text{Gen}(1^\lambda, 1^n); \mathcal{P}^*(\text{crs}_{\mathcal{P}}, \text{crs}_{\mathcal{V}}) = \pi^*] \geq \epsilon \quad (2)
\]

Then, assuming the underlying FHE, HT, and seBARG schemes are \( \text{poly}(\lambda, 1/\epsilon) \)-secure, there exists a choice of inputs \((pk, ek, ct)\) such that:

- \((pk, ek)\) are valid outputs of \( \text{FHE}.\text{Gen}(1^\lambda, 1^{d\text{FHE}}) \), and
- \(ct\) is a valid encryption of \( aux_x \) under \( pk \),

as well as a \((L, \delta)\)-local assignment generator for \( C_\mathcal{P} \), such that \( \delta = \text{negl}(\lambda) \) and the output gate is assigned 1 with probability \( 1 - \text{negl}(\lambda) \).

Lemma 4.11. Let \((pk, ek, ct)\) be as in the conclusion of Lemma 4.10. Given a \((L, \delta)\)-local assignment generator for \( C_\mathcal{P} \), there exists a ciphertext \( ct' \) that decrypts to \( (D_x) \) and a \((L, \delta_2 = \text{poly}(\log(\lambda s)) \cdot \delta)\)-local assignment generator for the circuit \( \widehat{U}_{E_x} \) (using these hard-coded keys and ciphertexts) with the same marginal distribution on its output gate.

Lemma 4.12. Let \((pk, ek, ct')\) be as in the conclusion of Lemma 4.11. Given an \((L, \delta_2)\)-local assignment generator for \( \widehat{U}_{E_x} \), one can construct an \((\ell, \delta_3 = \text{poly}(\lambda) \cdot \delta_2)\)-local assignment generator for \( U_{E_x} \), with the same marginal distribution on its output gate.

Lemma 4.13. For all \( \ell \geq 3 \cdot \log(s) + O(1) \), given an \((\ell, \delta_3)\)-local assignment generator for \( U_{E_x} \), one can construct a \((\ell, \delta_4 = \text{poly}(s) \cdot \delta_3)\) local assignment generator for \( E_x \), with the same marginal distribution on its output gate.

Combining these claims, we conclude that the candidate SNARG is sound for all languages \( \mathcal{L} \) that have locally unsatisfiable extensions (Definition 4.4).

Proof of Lemma 4.10  Fix \( \mathcal{P}^* \) and \( \{x = x^{(n(\lambda))}\}_{\lambda \in \mathbb{N}} \) as in the lemma statement. First, we consider an alternative algorithm \( \text{SNARG}.\text{Gen}' \) that differs from \( \text{SNARG}.\text{Gen} \) only on \( \lambda, n(\lambda) \). It is identical to \( \text{SNARG}.\text{Gen}(1^\lambda, 1^n) \) except for the following differences:

- It samples \( ct \) differently: Rather than encrypting \( 0^{|aux_x|} \), it encrypts the string \( aux_x \). Namely, \( ct \leftarrow \text{FHE}.\text{Enc}(pk, aux_x) \).

Equation (2), together with the security of the underlying FHE scheme, implies that there is a negligible function \( \mu \) such that for every \( \lambda \in \mathbb{N} \),

\[
\Pr[\text{SNARG}.\mathcal{V}^{\text{crs}_{\mathcal{V}}, x, \pi^*} = 1 : \mathcal{P}^*(\text{crs}_{\mathcal{P}}) = \pi^*] \geq \epsilon - \mu \quad (3)
\]

where \((\text{crs}_{\mathcal{P}}, \text{crs}_{\mathcal{V}}) \leftarrow \text{SNARG}.\text{Gen}' \). At this point, we non-uniformly fix the best choice of randomness for \((pk, ek, ct)\) (for each security parameter) for the rest of the analysis, so that Eq. (3) holds for these fixed strings.

Next, we will construct a \((L, \text{negl}(\lambda))\)-local tuple assignment generator (Definition 4.2) for \( C_\mathcal{P} \); by Lemma 4.3, this implies a \((L, \text{negl}(\lambda))\)-local assignment generator for \( C_\mathcal{P} \).

For any \( i_1, \ldots, i_L \in [k] \), consider another alternative algorithm \( \text{SNARG}.\text{Gen}'_{i_1, \ldots, i_L} \) that differs from \( \text{SNARG}.\text{Gen}' \) only on \( \lambda, n(\lambda) \). For these parameters, \( \text{SNARG}.\text{Gen}'_{i_1, \ldots, i_L}(1^\lambda, 1^n) \) has hardwired \( aux_x \) as well as \( i_1, \ldots, i_L \). It is identical to \( \text{SNARG}.\text{Gen}'(1^\lambda, 1^n) \) except for the following difference:
Rather than setting \((i_1^*, \ldots, i_L^*) = (1, \ldots, L)\), it sets \((i_1^*, \ldots, i_L^*) = (i_1, \ldots, i_L)\).

Equation (3), together with the index-hiding property of the underlying `seBARG` scheme, implies that there exists a negligible function \(\mu\) such that for every \(\lambda \in \mathbb{N}\), and every \(i_1, \ldots, i_L \in [k]\),

\[
\Pr[\text{SNARG}.\mathcal{V}(\text{crs}_P', x, \pi^*) = 1 : \mathcal{P}^*(\text{crs}_P') = \pi^*] \geq \epsilon - \mu
\]

where \((\text{crs}_P', \text{crs}_P) \leftarrow \text{SNARG}.\text{Gen}'(1^\lambda, 1^n).

Recall that \(\pi^* = (v_P, \pi_{\text{seBARG}}, \rho_{\text{inp}}, \rho_{\text{out}})\), where \(\pi_{\text{seBARG}}\) is a `seBARG` proof for the claim \(x^* = (M, z, k, T) \in \text{BatchIndexTMSAT}\), where \((M, z, k, T)\) is defined as in Item 8 of the definition of `SNARG`. As before, we define \(n^* = |x^*|\) and \(m^*\) to be the length of a witness corresponding to \(x^*\), and let \(\mathcal{R}\) denote the \(\mathcal{NP}\)-relation for `BatchIndexTMSAT`. By the somewhere argument of knowledge property of the underlying `seBARG` scheme (Definition 3.7), it holds that there exists a negligible function \(\nu\) such that for every \(\lambda \in \mathbb{N}\), every \(i_1, \ldots, i_L \in [k]\),

\[
\Pr[\text{seBARG}.\mathcal{V}(\text{crs}_{\text{seBARG}}, x^*, \pi_{\text{seBARG}}) = 1 \wedge \exists j \in [L] \text{ s.t. } (x^*, i_j, w^*_j) \notin \mathcal{R} : (\text{crs}_{\text{seBARG}}, \text{td}_{\text{seBARG}}) \leftarrow \text{seBARG}.\text{Gen}(1^\lambda, 1^{n^*}, 1^{m^*}, k, (i_1, \ldots, i_L)) \quad \text{hk} \leftarrow \text{HT}.\text{Gen}(1^\lambda) \quad \text{crs}_P' = (\text{hk}, \text{pk}, \text{ek}, \text{ct}, \text{crs}_{\text{seBARG}}) \quad (v_P, \pi_{\text{seBARG}}, \rho_{\text{inp}}, \rho_{\text{out}}) = \mathcal{P}^*(\text{crs}_P') \quad (w^*_{i_1}, \ldots, w^*_{i_L}) \leftarrow \text{Extract} (\text{td}_{\text{seBARG}}, \pi_{\text{seBARG}})] \leq \nu(\lambda).
\]

We next use the above equations to construct an \((L, \delta)\)-local tuple assignment generator `LocalGen_x` for \(C_P\), as follows:

1. Re-define \(i_1 = i_1 + |\langle R \rangle|, \ldots, i_L = i_L + |\langle R \rangle|\).\(^{16}\)
2. Set \(c = 1\) and \(K = \frac{\lambda}{\epsilon(\lambda)}\).
3. Sample \((\text{crs}_{\text{seBARG}}, \text{td}_{\text{seBARG}}) \leftarrow \text{seBARG}.\text{Gen}(1^\lambda, 1^{n^*}, 1^{m^*}, k, (i_1, \ldots, i_L))\).
4. Sample \(\text{hk} \leftarrow \text{HT}.\text{Gen}(1^\lambda)\).
5. Let \(\text{crs}_P' = (\text{hk}, \text{ek}, \text{ct}, \text{crs}_{\text{seBARG}})\) and \(\text{crs}_P' = (\text{hk}, \text{HT}.\text{Hash}(\text{hk}, (\text{ek}, \text{ct})), \text{crs}_{\text{seBARG}})\).
6. Compute \(\pi = (v_P, \pi_{\text{seBARG}}, \rho_{\text{inp}}, \rho_{\text{out}}) = \mathcal{P}^*(\text{crs}_P')\).
7. If \(\text{SNARG}.\mathcal{V}(\text{crs}_P', x, \pi) = 0:\)
   (a) If \(c < K\) then set \(c = c + 1\) and go to Item 3.
   (b) Otherwise output \(\bot\).
8. Compute \((w^*_{i_1}, \ldots, w^*_{i_L}) \leftarrow \text{Extract} (\text{td}_{\text{seBARG}}, \pi_{\text{seBARG}})\).
9. For every \(j \in [L]\) parse \(w_{ij} = (b_{j,1}, b_{j,2}, b_{j,3}, \rho_{j,1}, \rho_{j,2}, \rho_{j,3})\).
10. Output \((b_{j,1})_{j \in [L]}\).

\(^{16}\)This step occurs because we are building a local assignment generator for the computation that already has \(\langle R \rangle\) hard-wired.
Equation (4) implies that there exists a negligible function $\negl$ such that for every $i_1, \ldots, i_L \in [k]$
\[
\Pr[\text{LocalGen}_x(i_1, \ldots, i_L) = \bot] = \negl(\lambda).
\]
Moreover, Eq. (5) directly implies that the probability that $\text{LocalGen}_x(i_1, \ldots, i_L) \neq \bot$ but some triple $(b_{j,1}, b_{j,2}, b_{j,3})$ fails to satisfy the gate $i_j$ is also $\negl(\lambda)$! In order to conclude that $\text{LocalGen}_x$ is a valid local tuple assignment generator, we verify the necessary properties:

- **Computational Non-Signaling.** By a hybrid argument, it suffices to prove the computational non-signaling property on tuples $(i_1, \ldots, i_L)$, $(i'_1, \ldots, i'_L)$ that differ on a single coordinate $i_j \neq i'_j$.

  We further do a hybrid argument over the $K$ steps of the main loop defining $\text{LocalGen}$. We claim that in each step of the loop, we may replace an invocation of $\text{seBARG.Gen}(1^\lambda, 1^{n^*}, 1^{m^*}, k, i_1, \ldots, i_L)$ with an invocation of $\text{seBARG.Gen}(1^\lambda, 1^{n^*}, 1^{m^*}, k, i'_1, \ldots, i'_L)$. This follows immediately from the index hiding property of $\text{seBARG}$ (as in Remark 3.6), as an adversary distinguishing these two hybrids (given only the output of $\text{LocalGen}$ restricted to $[L]\{j\}$) immediately gives an algorithm violating the index hiding property of $\text{seBARG}$ in the presence of the $L-1$ $\text{seBARG}$ trapdoor components (outside of index $j$).

- **Wire consistency.** There are two consistency properties that we have to verify:
  
  - If a wire is opened in two different indices $i_{j_1}, i_{j_2}$, the bit assigned to the wire is consistent except with $\negl(\lambda)$ probability.
  
  - For any wire corresponding to an input bit (of $x$, $(R)$, $\vec{ek}$, or $ct$), the bit assigned to this gate by $\text{LocalGen}$ matches the input except with $\negl(\lambda)$ probability.

  These consistency properties (almost) all follow from the computational binding property of $\text{HT}$. To prove this, we union bound over the $K$ steps of the main loop in $\text{LocalGen}$, so it suffices to show that in each step of the main loop, the probability that $\text{LocalGen}$ assigns inconsistent wire values across indices (or compared to the inputs $x, \vec{ek}, ct$) is negligible. This follows immediately from computational binding, because $\text{LocalGen}$ only assigns a wire value when it has a local opening of $(v, v_P)$ to this wire value in the correct location. Moreover, $v$ is computed honestly by the $\text{Gen}$ algorithm and the verifier explicitly checks an opening of $v_P$ to $v_{\text{inp}} = \text{Hash}(hk, x)$. Thus, any inconsistency directly implies a violation of the binding property of $\text{HT}$ using the hash key $hk$ generated in the current execution of $\text{LocalGen}$.

  Finally, we must prove the consistency claim about the bits of $(R)$. Rather than following from computational binding, this follows from the somewhere argument of knowledge property of the $\text{seBARG}$ (Eq. (5)), as the $\text{seBARG}$ directly claims that for $1 \leq i \leq |\langle R \rangle|$, the bit $b_{i_1}$ is equal to the $i$th bit of $(R)$.

- **Gate Consistency.** The wire consistency analysis above implies that all $3L$ wire values that are revealed in the execution of $\text{LocalGen}$ (rather than just the $L$ wire values that are output) must be internally consistent (assuming the computational binding property of $\text{HT}$), except with negligible probability. Gate consistency follows from this, combined with the fact that any extracted triple $(b_{j_1}, b_{j_2}, b_{j_3})$ fails to satisfy the gate $i_j$ with negligible probability.

Having verified all of the claimed properties of $\text{LocalGen}$, we have completed the proof of Lemma 4.10. \[\square\]
Theorem 4.5, because \( L, \delta \) the output of a \((\sigma)\) that denote the wire values output by \( \text{LocalGen} \) suppose that the set \( T \) of the layered circuit corresponding to the encrypted 17 \( \ell \) at most \( \text{output by} \ FHE \ Let \((\text{Proof of Lemma 4.12.}) \). Finally, the correctness property of the \( \text{LocalGen} \) that every output wire of this sub-computation is correct except with probability \( \lambda, |\langle R \rangle| \cdot \delta \). Finally, the correctness property of the FHE implies that \( \text{ct}' \) does indeed decrypt to \( \langle D_x \rangle \). This completes the proof of Lemma 4.11.

Proof of Lemma 4.11. Let \((pk, \overline{ek}, \text{ct})\) be hard-coded strings such that \((pk, \overline{ek})\) are valid keys output by \( \text{FHE.Gen} \) and \( \text{ct} \) is a valid encryption of aux.

Let \( \text{LocalGen} \) denote a \((L, \delta)\) local assignment generator \( C_P \) with the above hard-coded inputs along with hard-coded (\((R), x)\). We use essentially the same algorithm and claim that it is also a \((L, \delta_2)\)-local assignment generator for \( \overrightarrow{U_{E_x}} \) with respect to the hard-coded input
\[
\text{ct}' = \text{FHE.Eval}_{\overline{ek}}(U_{\langle R \rangle}, \text{ct}).
\]

More formally, our new local assignment generator \( \text{LocalGen}' \), for any set \( T \) of wires of \( \overrightarrow{U_{E_x}} \), calls \( \text{LocalGen}(T') \) for the following set \( T' \):

- For any input wire in \( T \) corresponding to a bit of the hard-coded ciphertext \( \text{ct}' \), the set \( T' \) contains the corresponding output wire of the sub-circuit \( \left( \text{FHE.Eval}_{\overline{ek}}(U_{\langle R \rangle}, \text{ct}) \right)_{\text{Ext}} \) inside \( C_P \).

- For any non-input wire of \( \overrightarrow{U_{E_x}} \) in \( T \), the set \( T' \) contains the corresponding wire of \( C_P \), which contains \( \overrightarrow{U_{E_x}} \) as a sub-circuit.

By construction, non-signaling and gate correctness of \( \text{LocalGen}' \) follow immediately from the corresponding properties of \( \text{LocalGen} \). The only property that needs to be verified is that \( \text{LocalGen}' \) assigns input wires consistently with \( \text{ct}' \), with probability \( 1 - \delta_2 \). This holds by an invocation of Theorem 4.5, because \( \text{LocalGen also} \) definitionally contains a local assignment generator for the sub-circuit \( \left( \text{FHE.Eval}_{\overline{ek}}(U_{\langle R \rangle}, \cdot) \right)_{\text{Ext}} \) evaluated on deterministic input \( \text{ct} \), which by Theorem 4.5 implies that every output wire of this sub-computation is correct except with probability \( \text{poly}(\lambda, |\langle R \rangle|) \cdot \delta \).

Finally, the correctness property of the FHE implies that \( \text{ct}' \) does indeed decrypt to \( \langle D_x \rangle \). This completes the proof of Lemma 4.11.

Proof of Lemma 4.12. Let \((pk, \overline{ek}, \text{ct}')\) be hard-coded strings such that \((pk, \overline{ek})\) are valid keys output by \( \text{FHE.Gen} \) (with corresponding secret keys \( \overline{sk} \)) and \( \text{ct}' \) decrypts (under the appropriate \( \text{sk}_i \)) to \( \langle D_x \rangle \).

Finally, let \( \text{LocalGen} \) denote a \((L, \delta_3)\)-local assignment generator for \( \overrightarrow{U_{E_x}} \). We now describe a \((L, \delta_3)\)-local assignment generator \( \text{LocalGen}' \) for \( U_{E_x} \); the algorithm \( \text{LocalGen}'(T) \), on a set \( T \) at most \( \ell \), constructs the set \( T' \) containing all \( \leq L \) wires of \( \overrightarrow{U_{E_x}} \) that compute the \( \ell \) ciphertexts corresponding to the encrypted17 wire values of \( T \). Finally, for each index \( j \in T \), if the \( j \)th wire of the layered circuit \( U_{E_x} \) is in layer \( i \), the local assignment generator sets the \( j \)th wire value to \( \text{Dec}_{\overline{sk}_i}(\text{ct}_j) \), where \( \text{ct}_j \) is the string of wire values associated to \( j \) from the output of \( \text{LocalGen} \).

The computational non-signaling property of \( \text{LocalGen}' \) immediately follows from that of \( \text{LocalGen} \) (recall that the \( \text{sk}_i \) are fixed and, in particular, not secret).

To prove local correctness of \( \text{LocalGen}' \), we first invoke the local correctness of \( \text{LocalGen} \). That is, suppose that the set \( T \) contains a gate \( g_i = (i, j, k, f) \) that occurs in layer \( \alpha \) of \( U_{E_x} \), and let \( (\sigma_i, \sigma_j, \sigma_k) \) denote the wire values output by \( \text{LocalGen}' \). Then, we know there are ciphertexts \( \text{ct}_i, \text{ct}_j, \text{ct}_k \) such that \( \sigma_i = \text{Dec}_{\overline{sk}_i}(\text{ct}_i), \sigma_j = \text{Dec}_{\overline{sk}_i}(\text{ct}_j), \sigma_k = \text{Dec}_{\overline{sk}_i}(\text{ct}_k) \), such that the output of \( \text{LocalGen}'(T) \) contains the output of a \((L, \delta_2)\)-local assignment generator for the evaluation of the computation
\[
\text{FHE.GateEval}_{\overline{ek}}(f, \text{ct}_j, \text{ct}_k).
\]

17Wires in the unencrypted evaluation of \( C(x, w) \) are left unchanged.
Since the locality $L$ is larger than the entire homomorphic gate evaluation circuit, we can union bound over the errors from each of its $\text{poly}(\lambda)$ gates and conclude that

$$\text{ct}_i = \text{FHE.GateEval}_{e_k}(f, \text{ct}_j, \text{ct}_k)$$

with probability $1 - \text{poly}(\lambda) \cdot \delta_2$.

Finally, by the malicious gate correctness of the FHE, this implies that

$$\sigma_i = \text{FHE.Dec}_{\vec{x}}(\text{FHE.GateEval}_{e_k}(f, \text{ct}_j, \text{ct}_k)) = f(\sigma_j, \sigma_k)$$

with probability $1 - \text{poly}(\lambda) \cdot \delta_2$. This completes the proof of Lemma 4.12.

**Proof of Lemma 4.13.** Let LocalGen denote an $(\ell, \delta_3)$-local assignment generator for the universal circuit $U_{E_x}$ with the canonical description $(D_x)$ of $D_x$ hard-coded. We then give a simple definition of a local assignment generator for $E_x$: for any subset $T \subset [s]$, LocalGen$(T)$ calls LocalGen$(\bar{T})$, where $\bar{T}$ denotes the set containing the output wire of each sub-circuit $U_i$ for $i \in T$ (see Section 3.1.1). The computational non-signaling property of LocalGen then follows immediately from the analogous property of LocalGen.

What remains is to prove is the *local correctness* property of LocalGen. At a high level, local correctness holds because while LocalGen does not compute the gates of $E_x$ directly, a “gate” of $E_x$ is computed via a simple (low-depth) NAND-circuit *deterministically* with respect to a small number of wire values, and so the correctness error only “blows up” by a polynomial factor. This argument necessarily involves the details of our universal circuit implementation (Section 3.1.1).

Formally, we wish to show that for every gate $(i, j, k, f)$ of $E_x$, the probability that $\sigma_i \neq f(\sigma_j, \sigma_k)$ is at most $\text{poly}(s) \cdot \delta_3$. To see this, let $i^*, j^*, k^*$ denote the indices corresponding to the output wires of $U_i, U_j, U_k$ in $U_{E_x}$. In $U_{E_x}$, the wire $i^*$ is computed by a constant-size NAND-circuit on input wires $(j', k', \text{aux}_1, \text{aux}_2)$, where $\text{aux}_1, \text{aux}_2$ are copies of the two wires containing $f$ in the (hard-coded) description of gate $g_i$. Finally, let $T_i$ denote the set of $3 \log s + 2$ indices corresponding to the hard-coded input $g_i$.

By the computational non-signaling of LocalGen, it suffices to show that for every set $\bar{T}$ containing $\{i^*, j^*, k^*, j', k', \text{aux}_1, \text{aux}_2\} \cup T_i$, we have that $\sigma_i^* = f(\sigma_{j^*}, \sigma_{k^*})$ with all but $\text{poly}(s) \cdot \delta_3$ probability. To prove this, we observe that the desired claim follows from three properties of $(\sigma_{i^*}, \sigma_{j^*}, \sigma_{k^*}, \sigma_{j'}, \sigma_{k'}, \sigma_{\text{aux}_1}, \sigma_{\text{aux}_2})$:

1. $\sigma_{j^*} = \sigma_{j'}$,
2. $\sigma_{k^*} = \sigma_{k'}$, and
3. $\sigma_{i^*} = f(\sigma_{j'}, \sigma_{k'})$, where $f$ is the function described by $(\sigma_{\text{aux}_1}, \sigma_{\text{aux}_2})$.

The probability that property (3) fails is at most $O(\delta_3)$ by the local correctness of LocalGen. Thus, it suffices to show that properties (1) and (2) hold except with probability $\text{poly}(s) \cdot \delta_3$. This holds because the computation of $\sigma_{j^*}, \sigma_{k^*}$ is deterministic as a function of $\sigma_{j'}, \sigma_{k'},$ and $\sigma_{g_i}$, and is computed via a depth $O(\log s)$ circuit. Specifically, for all indices $\alpha$ in the sub-circuit $U_i$, there is a deterministic polynomial-time function $f_\alpha(\sigma_{j^*}, \sigma_{k^*})$ (corresponding to the honest evaluation of $U_i$) such that we expect $\sigma_\alpha = f_\alpha(\sigma_{j^*}, \sigma_{k^*})$ on any set $\bar{S}$ containing $(j^*, k^*, \alpha)$. We claim inductively with respect to $\alpha$ that

$$\Pr[\sigma_\alpha \neq f_\alpha(\sigma_{j^*}, \sigma_{k^*})] \leq 2(d_\alpha + 1)\delta_3 \cdot 2^{d_\alpha},$$

38
where $d_\alpha$ denotes the depth of $\alpha$ within the circuit $U_i$. This holds immediately for the base cases of $\alpha = j^*, k^*$. For the inductive step, let $\alpha$ have children $\beta, \gamma$ in $U_i$, whose corresponding depths satisfy $d_\beta, d_\gamma \leq d_\alpha - 1$. Then, by the non-signaling property of $\text{LocalGen}$ (up to error $\delta_3$), it suffices to consider sets $S$ that contain $(i^*, j^*, \alpha, \beta, \gamma)$; by the inductive hypothesis, we have that
\[
\Pr\left[\sigma_\beta \neq f_\beta(\sigma_{j^*}, \sigma_{k^*})\right] \leq 2(d_\beta + 1)\delta_3 \cdot 2^{d_\beta}
\]
and
\[
\Pr\left[\sigma_\gamma \neq f_\gamma(\sigma_{j^*}, \sigma_{k^*})\right] \leq 2(d_\gamma + 1)\delta_3 \cdot 2^{d_\gamma}.
\]
Finally, by the local correctness of $\text{LocalGen}$, we have that
\[
\Pr\left[\sigma_\alpha \neq \text{NAND}(\sigma_\beta, \sigma_\gamma)\right] \leq \delta_3.
\]
Since $f_\alpha(\sigma_{j^*}, \sigma_{k^*}) = \text{NAND}(f_\beta(\sigma_{j^*}, \sigma_{k^*}), f_\gamma(\sigma_{j^*}, \sigma_{k^*}))$ by definition, we conclude that
\[
\Pr\left[\sigma_\alpha \neq f_\alpha(\sigma_{j^*}, \sigma_{k^*})\right] \leq 2\delta_3 + 2(d_\beta + 1)\delta_3 \cdot 2^{d_\beta} + 2(d_\gamma + 1)\delta_3 \cdot 2^{d_\gamma} \leq 2(d_\alpha + 1)\delta_3 \cdot 2^{d_\alpha},
\]
completing the induction. Substituting $\alpha = j'$ and $\alpha = k'$ respectively, and observing that $f_{j'}(\sigma_{j^*}, \sigma_{k^*}) = \sigma_{j^*}$ and $f_{k'}(\sigma_{j^*}, \sigma_{k^*}) = \sigma_{k^*}$, we conclude the correctness of properties (1) and (2) up to error $\text{poly}(s) \cdot \delta_3$. This proves the local correctness property of $\text{LocalGen}'$ and thus completes the proof of Lemma 4.13.

### 4.6 Subsequent work: a strengthening of Theorem 4.8

Subsequent to the original results of this paper, it has proved useful to view the soundness analysis used to prove Theorem 4.8 as an efficient (black-box) reduction that outputs an (efficient!) local assignment generator $\text{LocalGen}$ given an efficient prover $P^*$ breaking soundness of the SNARG construction. Moreover, one can view this reduction as taking the description $<E_x>$ of the extended circuit as input; that is, it is universal with respect to $E_x$.

For ease of understanding, we formally state a strengthening of Theorem 4.8 that follows immediately from our proof of Theorem 4.8.

**Theorem 4.14 ([JKLM24]).** Assume the existence of a leveled, gate-by-gate FHE scheme (Definition 3.3) as well as a seBARG scheme (Definition 3.7). Then, the SNARG construction of Section 4.4 has the following soundness property: there exists a polynomial-time universal local assignment generator
\[
\text{LocalGen}(1^\lambda, T, \langle E_x \rangle, \epsilon)^{P^*}
\]
with the following guarantees.

- $\text{LocalGen}(1^\lambda, T, \langle E_x \rangle, \epsilon)^{P^*}$ is an oracle algorithm running in time $\text{poly}(s, \lambda, 1/\epsilon)$ for any circuit $E_x$ of size $s$.
- For any $\text{poly}(\lambda)$-size prover $P^*$, any non-negligible function $\epsilon = \epsilon(\lambda)$, and for every $\lambda \in \mathbb{N}$ and $x^* = x^*_\lambda$ of size $n(\lambda)$, if
  \[
  \Pr[\text{SNARG}.\forall(crs_\forall, x^*, \pi^*) = 1 : (\text{crs}_P, \text{crs}_\forall) \leftarrow \text{SNARG.Gen}(1^\lambda, 1^n); P^*(\text{crs}_P, \text{crs}_\forall) = \pi^*] \geq \epsilon
  \]

39
then for any extension $E_x^*$ of $C_x^*$ of size $s$, that can be described by a string $\text{aux}_x$ of (fixed) polynomial length, it holds that LocalGen($1^n, T, (E_x), \epsilon$)$^{\mathcal{P}^*'}(\cdot)$ is a ($\ell, \text{negl}(\lambda)$)-local assignment generator for the circuit $E_x^*$.

The statement of Theorem 4.14 is due to [JKLM24] and is not required for the results of this work. In Section 5.4, we state a similar rephrasing of Theorem 5.2 that is also due to [JKLM24].

5 SNARGs from Locally Unsatisfiable Extensions of Super-polynomial Size

In Section 4, we constructed a SNARG for any non-deterministic computation that has a $\ell$-locally unsatisfiable extension. However, the runtime of the honest prover in this SNARG grows with the size of the extension $E_x$ (while the size of the proof only grows with the locality $\ell$). As a result, we could only handle extensions of polynomial size.

In this section, we show how to construct SNARGs even for languages that have locally unsatisfiable extensions of super-polynomial size, as long as each gate in the extension can be computed in polynomial time. We formalize this via the notion of “dependent subcircuit” below.

**Dependent Subcircuit.** For any circuit $C$, we define the “dependent subcircuit” $\text{Dep}[g]$ for every gate $g$ in $C$. Informally, $\text{Dep}[g]$ contains all the wires (and gates) in $C$ that are used to compute the output of $g$.

Formally, we define a set of gates $\text{Dep}[g]$ that $g$ depends on. $\text{Dep}[g]$ is defined recursively as follows. Suppose that $g$ is computed using the input wires $l, r$. Let $g_l, g_r$ be the gates that compute $l, r$, respectively. If $l$ (resp. $r$) is an input wire to $C$, then we denote $g_l$ (resp. $g_r$) as an empty gate. Then we define $\text{Dep}[g] = \{g\} \cup \text{Dep}[g_l] \cup \text{Dep}[g_r]$ inductively. If $g$ is an empty gate, then we define $\text{Dep}[g]$ as the empty set.

With this definition, we may view $\text{Dep}[g]$ as a subcircuit of $C$ whose output wire is the output of $g$. Similarly, for any set $T$ of gates, one can define the dependent subcircuit $\text{Dep}[T] = \bigcup_{g \in T} \text{Dep}[g]$. Finally, these definitions allow us to define what it means for a circuit extension $\{E_x\}$ to be locally computable.

**Definition 5.1.** We say that a locally unsatisfiable extension $\{E_x\}_{x \in \{0,1\}^*}$ for the circuit family $\{C_x\}_{x \in \{0,1\}^*}$ is $\gamma(n)$-locally computable for a function $\gamma = \gamma(n)$, if for any $x \in \{0,1\}^n$, every gate $g$ of $D_x$ satisfies $|\text{Dep}[g]| \leq \gamma(n)$.

In Section 7.2, we show that if a circuit family has a bounded space propositional proof of unsatisfiability, then it has a locally computable locally unsatisfiable extension.

5.1 Theorem Statement

Fix any NP language $\mathcal{L} \subseteq \{0,1\}^*$ with a corresponding NP relation $\mathcal{R}_\mathcal{L}$. Let $C = \{C^{(n)}(x, w)\}_{n \in \mathbb{N}}$ be a circuit family deciding $\mathcal{R}_\mathcal{L}$, and let $M_\mathcal{L}$ denote a polynomial-time Turing machine that on input $(1^n, i)$ outputs the $i$th gate of $C^{(n)}$. Finally, suppose that the circuit family $\mathcal{C} = \{C_x\}_{x \in \{0,1\}^*}$ has an $\ell$-locally unsatisfiable extension $\{E_x\}_{x \in \{0,1\}^*}$ of size $s(n)$ that is $\gamma(n)$-locally computable (Definition 5.1).
Theorem 5.2. Assume the existence of a leveled, gate-by-gate FHE scheme (Definition 3.3) as well as a seBARG scheme (Definition 3.7) with subexponential security. Then, \( L \) as above has a SNARG with the following properties:

- The proof length is \( \text{poly}(\lambda, \ell, \log s) \).
- The crs length and prover complexity is \( \text{poly}(\lambda, \ell, \gamma, \log s) \).
- The scheme is non-adaptively sound.

The proof of Theorem 5.2 is similar to the proof of Theorem 4.8, with the following high-level differences:

- Most importantly, the honest prover cannot afford to homomorphically evaluate all of an extended circuit \( E \) of size \( s \). Instead, we have the prover homomorphically evaluate an arbitrary set of \( \ell \) (intermediate) wires of this circuit. Since each wire can be computed in time \( \gamma(n) \), a polynomial-time prover can carry out this operation.

- We would like to say that the resulting ciphertexts encrypt wire values forming a local assignment generator for a variant of the circuit \( E_x \). In order to ensure both non-signaling and local correctness of these decrypted wire values, we have the prover compute these \( \ell \) wire values in two different ways: (1) \textit{jointly}, to ensure local correctness, and (2) \textit{separately} (under \( \ell \) independent keys), to ensure non-signaling.

- The technical crux of the argument is showing that these two different computations of the wire values (under different keys) are consistent; this follows by applying arguments made in the proof of Theorem 4.8 to a second, nested local assignment generator.

### 5.2 SNARG Construction

Let \( L, C, M_L, \{E_x, D_x\}_{x \in \{0,1\}^n} \) be as in the Theorem statement. For every \( n \in \mathbb{N} \) and every \( x \in \{0, 1\}^n \), we denote by \( S(n) = \tilde{O}(s(n)^2) \) the number of gates in the circuit \( U_{D_x} \), noting that \( S(n) \) is not required to be a polynomial function.

We define a candidate SNARG for \( L \) using the following building blocks:

- A hash family with local opening \( HT = (HT.\text{Gen}, HT.\text{Hash}, HT.\text{Open}, HT.\text{Verify}) \), defined in Section 3.2.

- A leveled fully homomorphic encryption scheme

\[
\text{FHE} = (\text{FHE.\text{Gen}}, \text{FHE.\text{Enc}}, \text{FHE.\text{Dec}}, \text{FHE.\text{GateEval}}, \text{FHE.\text{Eval}})
\]

with gate-by-gate evaluation, defined in Section 3.3

- A somewhere extractable batch argument system \text{seBARG}, defined in Section 3.5.

Our construction makes use of the following definitions:

- For any gate \( g \) of \( D_x \), we define \( \text{Dep}[g] \) to be the sub-circuit of \( D_x \) required to compute the gate \( g \), and \( \text{Dep}[T] \) similarly for a set \( T \) of gates.
We define \( \text{Dep}[g]_{\text{Ext}} \) to be the [KRR14] extension of the circuit \( \text{Dep}[g] \), which consists \( \text{Dep}[g] \) along with an encoding of each layer of \( \text{Dep}[g] \).

For a set \( T \) of gates, we define a circuit \( \text{Dep}^*[T] \), which will be an extension of \( \text{Dep}[T] \) that is different from \( \text{Dep}[T]_{\text{Ext}} \). We define \( \text{Dep}^*[T] \) to be the circuit containing \( \text{Dep}[T] \) along with a copy of \( \text{Dep}[g]_{\text{Ext}} \) for every \( g \in T \). These copies of \( \text{Dep}[g]_{\text{Ext}} \) are all defined to contain the sub-circuit \( \text{Dep}[g] \) inside \( \text{Dep}[T] \).

We define the parameter \( \Gamma(n) = \text{poly}(\gamma(n)) \) to be the size of the canonical description of the circuit \( \text{Dep}[g]_{\text{Ext}} \) for a gate \( g \) of \( D_x \).

We define \( \Delta(n, \ell) = \text{poly}(\ell, \gamma) \) to be the size of the canonical description of the circuit \( \text{Dep}^*[T] \) for a set \( T \) of \( \ell \) gates of \( E_x \).

Our SNARG construction works as follows:

**The Setup Algorithm.** \( \text{SNARG.Gen}(1^{\lambda}, 1^n) \) does the following:

1. Set the FHE depth parameter \( d_{\text{FHE}} = \text{poly}(\lambda, \log s, \gamma, \ell) \) for a polynomial \( \text{poly} \) specified in the description of the Prover algorithm.

2. Set the security parameter \( \kappa = \lambda \cdot \text{poly}(\log s) \) so that the primitives \( \text{FHE}, \text{HT}, \) and \( \text{seBARG} \) are \( \text{negl}(\lambda) \cdot s^{-\alpha} \)-secure when using \( \kappa \) as a security parameter for a sufficiently large constant \( \alpha \) (for simplicity of notation below, we will assume \( \text{negl}(\lambda, s) \)-security, but this is unnecessary).

3. Sample \( \ell \) FHE key tuples \( (pk_i, ek^{(i)}, sk^{(i)})_{1 \leq i \leq \ell} \) each by calling \( \text{FHE.Gen}(1^k, 1^{d_{\text{FHE}}}) \).

4. Sample FHE key tuple \( (pk_{\text{joint}}, ek_{\text{joint}}, sk_{\text{joint}}) \leftarrow \text{FHE.Gen}(1^k, 1^{d_{\text{FHE}}}) \).

5. Sample \( \ell \) ciphertexts \( ct_i \leftarrow \text{FHE.Enc}(pk_i, 0^\Gamma) \).

6. Sample a ciphertext \( ct_{\text{joint}} \leftarrow \text{FHE.Enc}(pk_{\text{joint}}, 0^\Gamma) \).

7. Sample a hash key \( hk \leftarrow \text{HT.Gen}(1^k) \) and compute \( v = \text{Hash}(hk, (ek_{\text{joint}}, ek_1, \ldots, ek_{\ell}, ct_{\text{joint}}, ct_1, \ldots, ct_\ell)) \).

8. Sample \( (\text{crs}_{\text{seBARG}}^{\text{joint}}, \text{td}_{\text{seBARG}}^{\text{joint}}) \leftarrow \text{seBARG.Gen}(1^k, 1^{n^*}, 1^{m^*}, (i_1^*, \ldots, i_L^*)) \), where
   
   - The parameters \( n^* = \text{poly}(\lambda, \log s) \), \( m^* = \text{poly}(\lambda, \log s) \) and \( k = \text{poly}(\lambda, \gamma, \log s) \) are all defined below (when we describe the prover algorithm).
   
   - \( L = \ell + \text{poly}(\lambda) \) is chosen large enough for the analysis below.
   
   - \( i_1^*, \ldots, i_L^* \) are arbitrary indices in \([k]\). We arbitrarily set \((i_1^*, \ldots, i_L^*) = (1, \ldots, L)\).

9. For \( 1 \leq i \leq \ell \), sample \( (\text{crs}_{\text{seBARG}}^{(i)}, \text{td}_{\text{seBARG}}^{(i)}) \leftarrow \text{seBARG.Gen}(1^k, 1^{n^*}, 1^{m^*}, k, (i_1^*, \ldots, i_L^*)) \) for the same parameters above.

10. Sample \( (\text{crs}_{\text{seBARG}}, \text{td}_{\text{seBARG}}) \leftarrow \text{seBARG.Gen}(1^k, 1^{n^*}, 1^{m^*}, k, (i_1^*, \ldots, i_L^*)) \) for the same parameters above.
The common reference string is set to be \( \text{crs} = (\text{crs}_P, \text{crs}_V) \), where

\[
\text{crs}_P = (\text{hk}, ek^{(1)}, \ldots, ek^{(\ell)}, ct_{\text{joint}}, ct_1, \ldots, ct_\ell, \text{crs}_{\text{seBARG}}^{(1)}, \ldots, \text{crs}_{\text{seBARG}}^{(\ell)}, \text{crs}_{\text{seBARG}})
\]

and

\[
\text{crs}_V = (\text{hk}, v, \text{crs}_{\text{seBARG}}^{(1)}, \ldots, \text{crs}_{\text{seBARG}}^{(\ell)}, \text{crs}_{\text{seBARG}}).
\]

The Prover Algorithm. \( \text{SNARG} \cdot \mathcal{P}(\text{crs}_P, x, w) \) operates as follows.

1. Parse \( \text{crs}_P = (\text{hk}, ek^{(1)}, \ldots, ek^{(\ell)}, ct_{\text{joint}}, ct_1, \ldots, ct_\ell, \text{crs}_{\text{seBARG}}^{(1)}, \ldots, \text{crs}_{\text{seBARG}}^{(\ell)}, \text{crs}_{\text{seBARG}}) \).

2. Given \( w \), compute \( C_x(w) \) and let \( \tau = \tau_x(w) \) denote the wire assignment for this computation.

3. Run the (uniformly described) circuit that computes \( \text{FHE}.\text{Eval}_e(\tau, \cdot) \) on \( \text{ct} \), gate-by-gate, where \( U_{\tau}((A)) = A(\tau) \) is a universal circuit. The depth parameter \( d_{\text{FHE}} \) is chosen to be the depth of the computation homomorphically evaluated in this step.

4. Let \( \tau^{\text{joint}} \) denote the wire assignment for the above gate-by-gate homomorphic computation.

5. For \( 1 \leq i \leq \ell \), run the (uniformly described) circuit that computes \( \text{FHE}.\text{Eval}_e^{(i)}(\tau, \cdot) \) on \( \text{ct}_i \), gate-by-gate.

6. Let \( \tau^{(i)} \) denote the wire assignment for this gate-by-gate homomorphic computation.

7. Compute \( v_{\text{inp}} = \text{Hash}(\text{hk}, x) \) and \( v_P = \text{Hash}(\text{hk}, (v_{\text{inp}}, \tau, \tau^{\text{joint}}, \tau^{(1)}, \ldots, \tau^{(\ell)})) \). In the following description, we define an opening of \( v_P \) to a bit \( b \) in location \( i \) as follows:
   - If \( i \) corresponds to a wire of \( x \), an opening in location \( i \) consists of an opening of \( v_P \) to \( v_{\text{inp}} \) along with an opening of \( v_{\text{inp}} \) to \( b \) in location \( i \).
   - Otherwise, we use the usual notion of an opening of \( v_P \) w.r.t. \( \text{Hash}(\text{hk}, \cdot) \).

8. Compute an \text{seBARG} proof for the (informal) claims that every gate of \( \tau \) was computed correctly.

   Formally, let \( x^* = (M'_{C}, z, k, T_0) \), where
   - \( k = |\tau| + |U_{\tau}| \), noting that \( k \geq |\tau| = |C_x| \).
   - \( z = (\text{hk}, v, v_P) \).
   - \( M'_{C} \) is the Turing machine that on input \((z, i, w_i)\) does the following:
     - If \( 1 \leq i \leq |C| \), \( M \) checks that \( w_i = (b_1, \rho_1, b_2, \rho_2, b_3, \rho_3) \) is a valid local opening of \( v_P \) to bits \((b_1, b_2, b_3)\) in the locations corresponding to the \( i \)th gate of \( C \), and that \( b_1 = f(b_2, b_3) \) respects the gate computation.
     - Note that \( C \) has a uniform description \( M_C \), so there is a Turing machine \( M'_{C} \) with the above behavior.
   - \( T_0 \) is the run-time of \( M \), \( n^* = |(M, z, k, T)| \) and \( m^* = |w_i| = \text{poly}(\lambda, \log s) \).

Let

\[
\pi_{\text{seBARG}} \leftarrow \text{seBARG} \cdot \mathcal{P}(\text{crs}_{\text{seBARG}}, M'_{C}, z, 1^{T_0}, (w_1^{(0)}, \ldots, w_k^{(0)})),
\]

where \( w_1^{(0)}, \ldots, w_k^{(0)} \) are honestly generated witnesses.
9. Compute an seBARG proof for the claim that every homomorphic gate of \( \tau^{\text{joint}} \) was computed correctly.

Formally, let \( x^* = (M, z, k, T) \), where

- \( k = |\tau| + |U_\tau| \) as above.
- \( z = (hk, v, v_P) \) as above.
- \( M \) is the Turing machine that on input \((z, i, w_i)\) does the following:
  - \( M \) checks that \( w_i = (ek_i^{\text{joint}}, \rho, \hat{b}_1, \rho_1, \hat{b}_2, \rho_2, \hat{b}_3, \rho_3) \) is a valid local opening of \( (v, v_P) \) to:
    - * Three strings \( \hat{b}_1, \hat{b}_2, \hat{b}_3 \) in the (block) locations \( i_1, i_2, i_3 \), where \( (i_1, i_2, i_3) \) are the \( i \)th gate of \( U(\tau, \cdot) \), and
    - * An evaluation key \( ek_i^{\text{joint}} \) in the \( d \)th block location of \( v \), where \( d \) is the layer index of the \( i \)th gate of \( U(\tau, \cdot) \).
  - \( M \) checks that \( \hat{b}_1 = \text{FHE.GateEval}_{ek_i^{\text{joint}}}(f, \hat{b}_2, \hat{b}_3) \) respects the \( i \)th homomorphically evaluated gate of \( U(\tau, \cdot) \) on \( ct_{\text{joint}} \).

Note that the universal circuit \( U \) has a uniform description, so there is a polynomial-time Turing machine \( M \) with the above behavior.

- \( T \) is the run-time of \( M \), \( n^* = |(M, z, k, T)| \) and \( m^* = |w_i| = \text{poly}(\lambda, \log s) \).

Let

\[
\pi^{\text{joint}}_{\text{seBARG}} \leftarrow \text{seBARG} \cdot \mathcal{P}(\text{crs}^{\text{joint}}_{\text{seBARG}}, M, (z, i), 1^T, (w_1^{\text{joint}}, \ldots, w_k^{\text{joint}})),
\]

where \( w_1^{\text{joint}}, \ldots, w_k^{\text{joint}} \) are honestly generated witnesses.

10. Similarly, generate seBARG proofs \( \pi^{(1)}_{\text{seBARG}}, \ldots, \pi^{(\ell)}_{\text{seBARG}} \) for the same computation as above w.r.t. the ciphertexts \( ct_1, \ldots, ct_\ell \) (and using evaluation keys \( ek^{(1)}, \ldots, ek^{(\ell)} \)), to verify the computations of each \( \tau^{(i)} \).

11. Send \( \pi = (v_P, \pi_{\text{seBARG}}, \pi^{(1)}_{\text{seBARG}}, \ldots, \pi^{(\ell)}_{\text{seBARG}}, \rho_{\text{inp}}, \rho_{\text{out}}, \tau^{\text{joint}}_{\text{out}}, \rho^{(1)}_{\text{out}}, \ldots, \rho^{(\ell)}_{\text{out}}, \rho^{(\ell)}_{\text{out}}) \) to the verifier, where:

- \( \rho_{\text{inp}} \) is an opening of \( v_P \) to \( v_{\text{inp}} \) in the appropriate locations.
- \( \rho_{\text{out}} \) is an opening of \( \tau_{\text{out}} \), the output bit of \( \tau \),
- \( \rho^{\text{joint}} \) is an opening of the output wires of \( \tau^{\text{joint}} \), and
- For each \( 1 \leq i \leq \ell \), \( \rho^{(i)} \) is an opening of the output wires of \( \tau_i^{(i)} \).

The Verifier Algorithm. SNARG.\( \mathcal{V}(\text{crs}_\mathcal{V}, x, \pi) \) does the following:

1. Parse \( (\text{crs}_\mathcal{V}, \pi) \) as

\[
\text{crs}_\mathcal{V} = (hk, v, \text{crs}^{\text{joint}}_{\text{seBARG}}, \text{crs}^{(1)}_{\text{seBARG}}, \ldots, \text{crs}^{(\ell)}_{\text{seBARG}}, \text{crs}_{\text{seBARG}}),
\]

\[
\pi = (v_P, \pi_{\text{seBARG}}, \pi^{\text{joint}}_{\text{seBARG}}, \pi^{(1)}_{\text{seBARG}}, \ldots, \pi^{(\ell)}_{\text{seBARG}}, \rho_{\text{inp}}, \rho_{\text{out}}, \tau^{\text{joint}}_{\text{out}}, \rho^{(1)}_{\text{out}}, \ldots, \rho^{(\ell)}_{\text{out}}, \rho^{(\ell)}_{\text{out}}).
\]
2. Let \( z = (hk, v, \nu_P) \) and let \( M \) be the Turing machine defined in the Prover algorithm.

3. Check that \( \text{seBARG}.V(\text{crs}_{\text{seBARG}}, (M'_L, z, k, T_0), \pi_{\text{seBARG}}) = 1 \).

4. Check that \( \text{seBARG}.V(\text{crs}_{\text{joint}}(M, z, k, T), \pi_{\text{crs}}) = 1 \).

5. For \( 1 \leq i \leq \ell \), check that \( \text{seBARG}.V(\text{crs}_{\text{joint}}(i), (M, (z, i), k, T), \pi_{\text{crs}}) = 1 \).

6. Compute \( \nu_{\text{ inp}} = \text{HT}.\text{Hash}(hk, x) \) and check that \( \text{HT}.\text{Verify}(hk, \nu_P, T_{\text{ inp}}, x, \rho_{\text{ inp}}) = 1 \), where \( T_{\text{ inp}} \) is the set corresponding to the input hash.

7. Check that \( \text{HT}.\text{Verify}(hk, \nu_P, \text{ out}, \tau_{\text{ out}}, \rho_{\text{ out}}) = 1 \), for \( \tau_{\text{ out}} = 1 \).

8. Check that \( \rho_{\text{ joint}} \) is a valid opening to \( \tilde{\tau}_{\text{ joint}} \) and that each \( \rho^{(i)} \) is a valid opening to \( \tilde{\tau}^{(i)} \).

### 5.3 Analysis

We prove that the \( \text{SNARG} \) construction presented above satisfies \textbf{Theorem 5.2}. To this end, let \( L, C, M_L, \{E_x, D_x\}_{x \in \{0,1\}^*} \) be as in the Theorem statement.

The completeness guarantee follows immediately from the completeness guarantees of the underlying primitives. The efficiency guarantees can also be easily observed from the construction:

- The length of \( \text{crs}_P \) is
  \[
  \text{poly}(\kappa) \cdot (\Delta + \ell \cdot d_{\text{FHE}} + \ell \cdot |\text{crs}_{\text{seBARG}}|) = \text{poly}(\lambda, \log s, \gamma, \ell),
  \]
  where \( d_{\text{FHE}} = \tilde{O}(\Delta) \) and \( |\text{crs}_{\text{seBARG}}| = \text{poly}(\kappa, \log k, m^*, L) = \text{poly}(\lambda, \log s) \).

- The length of \( \text{crs}_V \) is \( \text{poly}(\lambda, \log s) \cdot \ell + \ell \cdot |\text{crs}_{\text{seBARG}}| = \text{poly}(\lambda, \log s, \ell) \).

- The length of \( \pi \) is similarly \( \text{poly}(\lambda, \log s, \ell) \).

- The runtime of \( \text{SNARG}.V \) is \( \text{poly}(\kappa) \cdot n + \text{poly}(\kappa, |\pi|) \) by the efficiency property of \( \text{seBARG} \).

We thus focus on proving non-adaptive soundness, which directly follows from the main claim below.

**Lemma 5.3.** Suppose that there exists a \( \text{poly}(\lambda) \)-size prover \( P^* \) and a non-negligible function \( \epsilon = \epsilon(\lambda) \), and for every \( \lambda \in \mathbb{N} \) there exists \( x^* = x^* \not\in L \) of size \( n(\lambda) \) such that
\[
\Pr[\text{SNARG}.V(\text{crs}_V, x^*, \pi^*) = 1 : (\text{crs}_P, \text{crs}_V) \leftarrow \text{SNARG}.\text{Gen}(1^\lambda, 1^n); P^*(\text{crs}_P, \text{crs}_V) = \pi^*] \geq \epsilon \quad (6)
\]

Then, assuming the underlying FHE scheme is \( \text{negl}(\lambda, s) \)-secure against \( \text{poly}(\lambda) \)-time adversaries, there is a function \( \delta = \text{negl}(\lambda, s) \) and a \( (\ell, \delta) \) local assignment generator \( \text{LocalGen}_x \) for the circuit \( E_x \) such that the output wire \( \sigma_{\text{ out}} = 1 \) with probability \( 1 - \delta \).

As before, to construct a local assignment generator, we first construct a local tuple assignment generator (\textbf{Definition 4.2}). The local tuple assignment generator \( \text{LocalGen}_x \) that we work with is constructed slightly differently from what was done in \textbf{Section 4.5}; we give a description of \( \text{LocalGen}_x(i_1, \ldots, i_\ell) \).
1. (Preprocessing phase): Compute the extended dependent circuits $\text{Dep}[g_{i_1}]_{\text{Ext}}, \ldots, \text{Dep}[g_{i_\ell}]_{\text{Ext}}$, $\text{Dep}^*[[g_{i_1}, \ldots, g_{i_\ell}]]$ w.r.t. the circuit $D_x$. The result is a collection of $\ell + 1$ polynomial-size circuit descriptions $(D_{i_1}), \ldots, (D_{i_\ell}), (D^*)$.

2. Auxiliary inputs: we think of $\text{LocalGen}_x$ receiving additional inputs $(i^*_1, \ldots, i^*_L)$. For every index $j$ such that $i^*_j$ corresponds to a wire of $(x, \tau)$, we define $i^*_j = i_j$. For all other $j$, we define $i^*_j = 1$.

3. Set $c = 1$ and $K = \frac{\lambda \log(s)}{\epsilon(\lambda)}$.

4. Sample $(\text{crs}_{\text{seBARG}}, \text{td}_{\text{seBARG}}) \leftarrow \text{seBARG.Gen}(1^\kappa, 1^{n^*}, 1^{m^*}, (i^*_1, \ldots, i^*_L))$.

5. Sample $(\text{crs}_{\text{joint}}^{\text{seBARG}}, \text{td}_{\text{joint}}^{\text{seBARG}}) \leftarrow \text{seBARG.Gen}(1^\kappa, 1^{n^*}, 1^{m^*}, (i^*_1, \ldots, i^*_L))$.

6. For $1 \leq j \leq \ell$, sample $(\text{crs}_{\text{seBARG}}^{(j)}, \text{td}_{\text{seBARG}}^{(j)}) \leftarrow \text{seBARG.Gen}(1^\kappa, 1^{n^*}, 1^{m^*}, (i^*_1, \ldots, i^*_L))$.

7. Sample $(\text{pk}_{\text{joint}}, \vec{e}_k^{\text{joint}}, \vec{s}^{\text{joint}}) \leftarrow \text{FHE.Gen}(1^\kappa, 1^{d_{\text{FHE}}})$.

8. For $1 \leq j \leq \ell$, sample $(\text{pk}_j, \vec{e}_k^{(j)}, \vec{s}^{(j)}) \leftarrow \text{FHE.Gen}(1^\kappa, 1^{d_{\text{FHE}}})$.

9. Sample $hk \leftarrow \text{HT.Gen}(1^\kappa)$.

10. Compute $\text{ct}_{\text{joint}} \leftarrow \text{FHE.Enc}(\text{pk}_{\text{joint}}, (D^*))$.

11. For $1 \leq j \leq \ell$, compute $\text{ct}_j \leftarrow \text{FHE.Enc}(\text{pk}_j, (D_{i_j}))$.

12. Let $\text{crs}'_\ell = (hk, \vec{e}_k^{\text{joint}}, \vec{r}^{(1)}, \ldots, \vec{r}^{(\ell)}, \text{ct}_{\text{joint}}, c_{t_1}, \ldots, c_{t_\ell}, \text{crs}_{\text{seBARG}}^{(1)}, \text{crs}_{\text{seBARG}}^{(2)}, \ldots, \text{crs}_{\text{seBARG}}^{(\ell)}, \text{crs}_{\text{seBARG}}, \text{crs}_{\text{BARG}})$.

13. Let $\text{crs}'_v = (hk, v_{\text{seBARG}}, \text{crs}_{\text{seBARG}}^{(1)}, \ldots, \text{crs}_{\text{seBARG}}^{(\ell)}, \text{crs}_{\text{seBARG}}, \text{crs}_{\text{BARG}})$ for $v = \text{HT.Hash}(hk, (\vec{e}_k, \vec{r}^{(1)}, \ldots, \vec{r}^{(\ell)}, c_{t_1}, \ldots, c_{t_\ell}))$.

14. Compute $\pi = (v_{\text{seBARG}}, \pi_{\text{seBARG}}^{(1)}, \pi_{\text{seBARG}}^{(2)}, \ldots, \pi_{\text{seBARG}}^{(\ell)}, \text{c}_{\text{seBARG}, \text{inp}}, \text{c}_{\text{seBARG}, \text{out}}, \text{c}_{\text{out}}, \text{c}_{\text{out}}, \text{c}_{\text{out}}, \ldots, c_{\text{out}}, c_{\text{out}}, c_{\text{out}}, c_{\text{out}}, c_{\text{out}}) = \text{P}^*(\text{crs}'_v)$.

15. If $\text{SNARG.\emph{V}}(\text{crs}'_v, x, \pi) = 0$:

   a. If $c < K$ then set $c = c + 1$ and go to Item 4.

   b. Otherwise output $\bot$.

16. Compute $(w^*_1, \ldots, w^*_L) \leftarrow \text{Extract}(\text{td}_{\text{seBARG}}, \pi_{\text{seBARG}})$.

17. For all $j$ such that $i^*_j$ corresponds to a wire of $(x, \tau)$, set $b_{i_j}$ to the “output wire bit” of $w^*_{i_j}$.

18. For all other $j$, compute $b_{i_j} = \text{FHE.Dec}(sk^{(i_j)}_{\text{FHE}}, \vec{r}^{(j)}_{\text{out}})$.

19. Compute $(b_{i_j}^*) = \text{FHE.Dec}(sk^{\text{joint}}_{\text{FHE}}, \vec{r}^{\text{joint}})$.

20. Output $(\sigma_{i_j} = b_{i_j})_{j \in [\ell]}$.
Just as in the proof of Lemma 4.10, Eq. (6) along with the $\text{negl}(\lambda, s)$-security of the FHE implies that the probability that $\text{LocalGen}_x(i_1, \ldots, i_\ell)$ outputs $\bot$ is $\text{negl}(\lambda, s)$. Moreover, the $\text{negl}(\lambda, s)$-computational non-signaling of $\text{LocalGen}_x$ follows immediately from the security of the FHE, because in each loop iteration, each $b_{i_j}$ depends only on $\text{sk}^{(j)}$ and no other secret keys.

Thus, to prove Lemma 5.3, it suffices to show that $\text{LocalGen}_x$ satisfies wire consistency and gate consistency. This relies crucially on the seBARG to “tie together” the $\ell$ plaintext values encrypted under unrelated FHE keys. On the other hand, we will not rely on FHE security at all, but rather use the FHE secret keys freely in our analysis.

We establish the wire consistency and gate consistency properties by considering a simple extension $\text{LocalGen}^*_x(i_1, \ldots, i_\ell)$ of $\text{LocalGen}_x(i_1, \ldots, i_\ell)$ that outputs two tuple assignments $(b_{i_j}, b^*_j)_{j=1}^\ell$ instead of just one. We then make the following claim.

**Claim 5.4.** The probability that $b_{i_j} \neq b^*_j$ for any $1 \leq j \leq \ell$ is $\text{negl}(\lambda, s)$.

Claim 5.4 implies both wire consistency and gate consistency because, just as in the proof of Lemma 4.10, there is a local assignment generator for the circuit $(x, w) \mapsto D^*(C(x, w))$ whose $\ell$ output value wires match $(b^*_j)_{j=1}^\ell$. If $i_1 = i_2$, then the two output wires are actually identical wires in $D^*$, and thus they trivially have the same assignment. Additionally, for any gate $g = (i, j, k, f)$ of $E_x$, the corresponding wires of $D^*$ are connected via an $f$-gate, and thus we can conclude that $b^*_i = f(b^*_j, b^*_k)$ with all but $\text{negl}(\lambda, s)$ probability.

Thus, it suffices to establish Claim 5.4, which we can prove via a union bound over all $1 \leq j \leq \ell$. Fix a tuple $(i_1, \ldots, i_\ell)$ and an index $j \in [\ell]$. We define the following two circuits.

- Let $E_{i_1, \ldots, i_\ell, j}$ be the circuit that on input $w \in \{0, 1\}^m$:
  - computes $C_x(w)$, resulting in wire assignment $\tau$.
  - computes $D^*(\tau)$.
  - computes $D_{i_j}(\tau)$.
- Let $U_{i_1, \ldots, i_\ell, j}$ be the universal circuit that on input $(w, \langle D^* \rangle)$ computes $E_{i_1, \ldots, i_\ell, j}(w)$. Note that this “universal” circuit still contains a copy of $C_x$ in it; only the $D^*, D_{i_j}$ computations are made universal.

We prove Claim 5.4 by proving the following two lemmas.

**Lemma 5.5.** There is a negligible function $\delta'(\lambda, s)$ such that for every $(i_1, \ldots, i_\ell, j)$ as above, there is a $(L, \delta')$-local assignment generator $\text{LocalGen}'_{x, i_1, \ldots, i_\ell, j}$ for $U_{i_1, \ldots, i_\ell, j}$ such that for any index $j$ and any set $T$ containing $\{\text{out}_{D_{i_j}}, \langle \text{out}_{D^*} \rangle\}$, the marginal distribution of $\sigma'_T$ on $\{\text{out}(D_{i_j}), \text{out}(D^*)\}$ is $\delta(\lambda, s)$-close to that of $(b_{i_j}, b^*_{i_j})$ from $\text{LocalGen}^*_x(i_1, \ldots, i_\ell)$.

**Proof.** We describe the algorithm $\text{LocalGen}'_{x, i_1, \ldots, i_\ell, j}(i_1^*, \ldots, i_\ell^*)$ for a local tuple assignment generator:

- Hard-coded inputs: the circuit descriptions $\langle D_{i_j} \rangle$ for $j \in [\ell]$ and the circuit $D^*$.
- Run the algorithm $\text{LocalGen}^*_x(i_1, \ldots, i_\ell)$ using auxiliary inputs $i_1^*, \ldots, i_\ell^*$.
- Compute $(w_{i_1}^*, \ldots, w_{i_\ell}^*) \leftarrow \text{Extract}(\text{tdseBARG}, \pi_{\text{seBARG}})$.
• Compute \((w_{i_1}^{\text{joint}}, \ldots, w_{i_L}^{\text{joint}}) \leftarrow \text{Extract}(\mathsf{td}_{\text{seBARG}}^{\text{joint}}, \pi_{\text{seBARG}}^{\text{joint}})\)

• For \(1 \leq \alpha \leq L\), compute \((w_{i_1}^{(\alpha)}, \ldots, w_{i_L}^{(\alpha)}) \leftarrow \text{Extract}(\mathsf{td}_{\text{seBARG}}^{(\alpha)}, \pi_{\text{seBARG}}^{(\alpha)}).\)

• For \(1 \leq \alpha \leq L\), output the following value of \(\sigma_{i_\alpha}^{*}\):
  - If \(i_\alpha^*\) is a wire in \(C\), output the wire value \(b_{i_\alpha^*}\) contained in \(w_{i_\alpha}^*\).
  - If \(i_\alpha^*\) is a wire in \(D^*\), output the wire value \(\mathsf{FHE}.\mathsf{Dec}_{sk}^{\text{joint}}(\hat{b}_{i_\alpha^*})\) computed from the \(\hat{b}_{i_\alpha^*}\) contained in \(w_{i_\alpha}^{\text{joint}}\).
  - If \(i_\alpha^*\) is a wire in \(D_i\), output the wire value \(\mathsf{FHE}.\mathsf{Dec}_{sk}^{(i)}(\hat{b}_{i_\alpha^*})\) computed from the \(\hat{b}_{i_\alpha^*}\) contained in \(w_{i_\alpha}^{(i)}\).

The non-signaling property of \text{LocalGen}' holds by the same argument for non-signaling in Lemma 4.10; namely, by a hybrid argument over the loop in the algorithm \text{LocalGen}, invoking the index hiding property of all of the \text{seBARGs}. The wire and gate consistency of \text{LocalGen}' follows by combining the somewhere argument-of-knowledge property of the \text{seBARGs} with the computational binding property of \text{HT} (as in the proof of Lemma 4.10) and the malicious gate correctness property of the \text{FHE} schemes (as in Lemma 4.12). Crucially, the non-signaling and local correctness errors are now \text{negl}(\lambda, s) because the index hiding, computational binding and somewhere argument-of-knowledge properties all have security error \text{negl}(\lambda, s) under our new parameter settings and assumptions.

Lemma 5.6. For all \(L \geq \ell + 3 \log(\Delta) + O(1)\), given a \((L, \delta')\)-local assignment generator for \(U_{i_1,\ldots,i_L}^*\), one can construct a \((L, \text{poly}(\Delta) \cdot \delta')\)-local assignment generator \text{LocalGen}'' for \(E_{i_1,\ldots,i_L}^*\) preserving (exactly) the marginal distribution on \(\{\text{out}(D_{i_j}), \text{out}(D^*)_{i_j}\}\).

The proof of Lemma 5.6 is almost identical to that of Lemma 4.13, so we omit the proof.

Finally, Claim 5.4 follows from Lemmas 5.5 and 5.6, and Theorem 4.5, because within the circuit \(E_{i_1,\ldots,i_L}^*\), the wires \(\text{out}(D_{i_j})\) and \(\text{out}(D^*)_{i_j}\) are jointly computed by a “double encoded circuit” of \(D_{i_j}\). This completes the proof of Claim 5.4 and therefore the proof of Theorem 5.2.

5.4 Subsequent work: a strengthening of Theorem 5.2

Just as in Section 4.6, one can view Theorem 5.2 as an efficient reduction that outputs a local assignment generator \text{LocalGen} given an efficient prover \(P^*\) breaking soundness of the \text{SNARG} construction, along with a description \(\langle E_x \rangle\) of an extended circuit.

While one benefit of doing this over Theorem 4.14 is that it applies to super-polynomial size circuit extensions, another benefit is that evaluating the local assignment generator \text{LocalGen} here does not even require knowing the full description of \(E_x\). Indeed, inspecting the proof of Theorem 5.2, it is clear that \text{LocalGen}(T) only requires knowing the dependent sub-circuit \text{Dep}[T] rather than \(\langle E_x \rangle\). We give one formalization of this fact below.

Theorem 5.7 ([JKLM24]). Fix any \text{NP} language \(L \subseteq \{0,1\}^*\) with a corresponding \text{NP} relation \(R_L\). Let \(C = \{C^{(n)}(x,w)\}_{n \in \mathbb{N}}\) be a circuit family deciding \(R_L\), and let \(M_L\) denote a polynomial-time Turing machine that on input \((1^n, i)\) outputs the \(i\)th gate of \(C^{(n)}\).
Let $\ell(\cdot), \gamma(\cdot), s(\cdot)$ denote efficiently computable functions of $n$. Assume the existence of a leveled, gate-by-gate FHE scheme (Definition 3.3) as well as a seBARG scheme (Definition 3.7) with $\text{negl}(\lambda, s)$ security. Then, there exists a SNARG for $\mathcal{L}$ with the following properties.

- **The proof length is** $\text{poly}(\lambda, \ell, \log s)$.
- **The crs length and prover complexity** is $\text{poly}(\lambda, \ell, \gamma, \log s)$.
- **Soundness:** there exists a polynomial-time universal local assignment generator $\text{LocalGen}(1^\lambda, T, \langle E_x \rangle, \epsilon)^{\mathcal{P}^*}$ with the following guarantees for any family of extensions $\{E_x\}_x$ of size $s$ that is $\gamma$-locally computable.
  
  - $\text{LocalGen}(1^\lambda, T, \langle E_x \rangle, \epsilon)^{\mathcal{P}^*}$ is an oracle algorithm running in time $\text{poly}(s, \lambda, 1/\epsilon)$.
  
  - For any $\text{poly}(\lambda)$-size prover $\mathcal{P}^*$, any non-negligible function $\epsilon = \epsilon(\lambda)$, and for every $\lambda \in \mathbb{N}$ and $x^* = x^*_\lambda$ of size $n(\lambda)$, if $\Pr[\text{SNARG}, V^{\text{crs}_V, x^*, \pi^*} = 1 : (\text{crs}_V, \text{crs}_V) \leftarrow \text{SNARG.Gen}(1^\lambda, 1^{\mathcal{P}^*}); \mathcal{P}^*(\text{crs}_V, \text{crs}_V) = \pi^*] \geq \epsilon$

  then $\text{LocalGen}(1^\lambda, T, \langle E_x \rangle, \epsilon)^{\mathcal{P}^*}$ is a $(\ell, \text{negl}(\lambda, s))$-local assignment generator for $E_x$.

- **Extension hiding:** For $\mathcal{P}^*$ as above, any set $T$ of size at most $\ell$, and any two extensions $E_x, E'_x$ such that the dependent sub-circuits $\text{Dep}[T], \text{Dep'}[T]$ are identical, we have that

  \[
  \text{LocalGen}(1^\lambda, T, \langle E_x \rangle, \epsilon)^{\mathcal{P}^*} \approx_{c, \text{negl}(\lambda, s)} \text{LocalGen}(1^\lambda, T, \langle E'_x \rangle, \epsilon)^{\mathcal{P}^*}.
  \]

The statement of Theorem 5.7 is due to [JKLM24] and is not required for the main results of this work. Its proof follows immediately from that of Theorem 5.2.

### 5.5 Limitations of locally unsatisfiable extensions

At the end of this section, we briefly remark some complexity-theoretic limitations of our approach based on locally satisfiable extensions. Specifically, we claim:

**Theorem 5.8.** Any NP language with a $\ell$-locally unsatisfiable extension of size $s$ is contained in coNTIME($s^{O(\ell)}$).

Indeed, one co-nondeterministic algorithm for deciding such a language is, given $x$, to guess the description of $E_x$ and then (approximately) compute the optimal probability that $g_{\text{out}} = 1$ over all perfectly non-signaling local assignment generators for $E_x$. This can be computed in time $s^{O(\ell)}$ by linear programming [KRR14].

To see that this algorithm correctly decides $\mathcal{L}$, we observe that when $x \in \mathcal{L}$, this maximum probability is equal to 1 for any choice of $E_x$, while when $x \notin \mathcal{L}$, by assumption there exists some $E_x$ such that this probability is bounded away from 1.

As a result, we conclude that under standard complexity-theoretic conjectures, there exist NP complete languages that do not have $\text{poly}(\lambda)$-locally unsatisfiable extensions of size $s = 2^\lambda$,\(^\text{18}\) for appropriate parameter settings such as $\lambda = n^\epsilon$. On the other hand, it remains plausible that every NP language has a non-trivial locally-unsatisfiable extension.

\(^{18}\)We restrict to $s = 2^\lambda \ll 2^n$ because in Theorem 5.2, SNARGs based on locally unsatisfiable extensions have proof length that grows logarithmically in $s$. 

49
Question 5.9. Does every NP language with witness length $m$ have a $\sqrt{m}$-locally unsatisfiable extension of size $2^{O(\sqrt{m})}$?

6 Locally Unsatisfiable Extensions from Propositional Proofs of Unsatisfiability

In this section, we prove that if a circuit has a polynomial-size propositional logic proof of unsatisfiability, then it has a locally unsatisfiable extension. This is stated formally (in Theorem 6.2) below.

Circuit as a Set of Propositions. For any circuit $C$, we define a set of propositional formulas $\text{Prop}[C]$, which represents that each gate computation in $C$ is correct. More specifically, we assign each wire in $C$ a propositional variable. Then for each gate $g$ in $C$ with input wires $l, r$ and output wire $o$, we can use the propositional formula $o \leftrightarrow g(l, r)$ to represent that $g$ is computed correctly. We put all such formulas together as a set, and define it as $\text{Prop}[C]$.

Definition 6.1 (Propositional Proofs of Unsatisfiability). For any set $S \subseteq \{0, 1\}^*$, for any circuit family $\{C_x\}_{x \in S}$ over $S$, we say $\{C_x\}_{x \in S}$ has an $\ell$-propositional proof of unsatisfiability for a function $\ell = \ell(n)$, if there exists a family of propositional logic proof $\{\pi_x\}_{x \in S}$ in the Extended Frege system, where $\pi_x$ is the derivation of

$$\text{Prop}[C_x] \vdash \text{out} \leftrightarrow \text{F},$$

where $\text{out}$ is the output wire of $C_x$. Moreover, the size of $\pi_x$ is bounded by $\ell(n)$ for any $x \in S \cap \{0, 1\}^n$.

In this section, we will prove the following main theorem.

Theorem 6.2. For any set $S \subseteq \{0, 1\}^*$ and any circuit family $\{C_x\}_{x \in S}$, if $\{C_x\}_{x \in S}$ has an $\ell$-propositional proof of unsatisfiability, where $\ell = \ell(n)$ is a polynomial, then there exists a family of polynomial-size circuits $\{E_x\}_{x \in S}$ and a constant $A$, such that $\{E_x\}_{x \in S}$ is an $A \log \ell$-locally unsatisfiable extension.

We prove this Theorem in three steps: We first build a helper circuit in Section 6.1; then we construct the extension circuit in Section 6.2; finally, we prove its local unsatisfiability in Section 6.3.

Remark 6.1. We extend Theorem 6.2 to the following Theorem 6.3, which will be used in the subsequent work [JKLM24]. The main difference is that we bound the size of the Turing machine used to describe the extension circuit in Theorem 6.2, when $C_x$ and its $\mathcal{EF}$ proof of unsatisfiability can be described by uniform Turing machines with input $\text{aux, aux}_{\mathcal{EF}}$, respectively. We can extend the proof of Theorem 6.2 to this uniform setting, because our extension circuit in the proof of Theorem 6.2 is highly uniform from the circuit $C_x$ and the $\mathcal{EF}$ proof.

Theorem 6.3. There exists a polynomial poly such that the following holds. Fix parameters $T = T(n)$, $d = d(n)$, $D = D(n)$, and $L = L(n)$. Fix any circuit family $\{C_x\}_{x \in \{0, 1\}^*}$ such that each $C_x$ is of size $T = T(|x|)$ and depth $d = d(|x|)$, and can be generated by a $(T + L)$-time uniform Turing machine with advice $\text{aux} = \text{aux}_x$.

If $S \subseteq \{0, 1\}^*$ is a subset such that for every $x \in S$, $C_x$ is unsatisfiable and has an $\mathcal{EF}$ proof for the statement “for all $w$, $C_x(w) = 0$” of length $L(|x|)$, and this $\mathcal{EF}$ proof can be generated by a $(T + L)$-time uniform Turing machine with advice $\text{aux}_{\mathcal{EF}} = \text{aux}_{\mathcal{EF}, x}$ of length $D$, then there exists a
circuit family \(\{E_x\}_{x \in S}\) where each \(E_x\) is of size \(T' \leq \poly(T + L)\), depth \(D' \leq (d + L) \cdot \poly(n)\), and it can be generated by a \(T'\)-time Turing machine with advice \((\text{aux}, \text{aux}_{\epsilon})\). Moreover, there exists an \(\epsilon = \polylog(T', L')\), and \(\{E_x\}_{x \in S}\) is an \(\epsilon\)-locally unsatisfiable extension of \(\{C_x\}_{x \in S}\).

6.1 Binary AND-Tree Circuits

Before we prove Theorem 6.2, we first build a family of helper circuits \(\{\Delta_n\}_{n \in \mathbb{N}}\) in Figure 2. The circuit \(\Delta_n\) is simply the binary AND tree on \(n\) inputs when \(n\) is a power of 2. For general \(n\), we first find the largest \(n' \leq n\) that is a power of 2, and build a binary tree on the first \(n'\) input wires, and then we recursively apply the same procedure to the remaining \((n - n')\) input wires.

The circuit \(\Delta_n\) is a sub-circuit in the extended circuit construction (in Section 6.2). We will prove that the circuit \(\Delta_n\) achieves the following property, which will be useful in arguing the local unsatisfiability of our extended circuit construction. For any \(n\), suppose we want to argue that for some \((\Theta(\log n), \epsilon)\)-local assignment generator \(\text{LocalGen}\), all the input wires \(w_1, w_2, \ldots, w_n\) are all 1’s with overwhelming probability. However, we do not wish to extract all the input wires \(w_1, w_2, \ldots, w_n\), because we can only extract a small number of wires. To resolve this issue, we only extract the output of the root nodes of the binary trees in \(\Delta_n\).

Let \(\text{Root}_n\) be the set of the root nodes in \(\Delta_n\). Clearly, \(w_1, \ldots, w_n\) are all 1’s if and only if the outputs of \(\text{Root}_n\) are all 1’s. Moreover, we will further use the following properties of \(\Delta_n\) and \(\text{Root}_n\), which we formally prove in Lemma 6.4.

- **Read.** We can “read” any input wire \(w_i\) from \(\text{Root}_n\) if all gates in \(\text{Root}_n\) output 1. Namely, if we extract \(\text{Root}_n\) and an input wire \(w_i\), and all the gates in \(\text{Root}_n\) output 1’s, then the input wire \(w_i\) must also be 1 except for a small probability. This property can be generalized from a single index \(i\) to any constant-size subset of indices \(S \subseteq [n]\).

- **Increment.** For any \(k < n\), \(\Delta_k\) is a sub-circuit of \(\Delta_n\), where the input to \(\Delta_k\) are the first \(k\) input wires \(w_1, w_2, \ldots, w_k\). Hence, a local assignment generator for \(\Delta_n\) is also a local assignment generator for the sub-circuit \(\Delta_k\).

- **Append.** We can “append” \(\text{Root}_{n-1}\) to \(\text{Root}_n\) if the \(n\)-th input wire is always 1. Namely, if we extract all the wires in \(\text{Root}_{n-1} \cup \text{Root}_n\) and also the \(n\)-th input wire \(w_n\), and all the gates in \(\text{Root}_{n-1}\) output 1’s and \(w_n = 1\), then all the gates in \(\text{Root}_n\) also output 1’s except for a small probability.

**Lemma 6.4.** For the circuit \(\Delta_n\) in Figure 2, for any constant \(c \geq 1\), any function \(\epsilon = \epsilon(\lambda)\), and any \((4c[\log n], \epsilon)\)-local assignment generator \(\text{LocalGen}\), \(\Delta_n\) satisfies the following properties. Recall that \(w_1, w_2, \ldots, w_n\) are the input wires of \(\Delta_n\) and \(\text{Root}_n\) is the set of the output gates of \(\Delta_n\).

- **Read.** For any subset of input wires \(S \subseteq [n]\) with \(|S| \leq c\), we have
  \[
  \Pr[\forall i \in \text{Root}_n, w_i = 1 \text{ and } \exists j \in S, w_j \neq 1] \leq \poly(n) \cdot \epsilon(\lambda),
  \]
  where \(\{w_i\}_{i \in S \cup \text{Root}_n} \leftarrow \text{LocalGen}(S \cup \text{Root}_n)\).

- **Increment.** For any integer \(k < n\), there is a sub-circuit in \(\Delta_n\) that is identical to \(\Delta_k\). Moreover, the input wires to the sub-circuit are the first \(k\) input wires \(w_1, w_2, \ldots, w_k\) of \(\Delta_n\).

---

\(^{19}\)Note that if \(n\) is a power of 2 then there is a single root note, however for general \(n\) there may be as many as \(\log n\) root nodes.
Circuit $\Delta_n$

Input: $n$ wires $w_1, w_2, \ldots, w_n$.
Output: $O(\log n)$ values of the gates in a set $\text{Root}_n$.

1. If $n = 1$, we build a gate computing the identity function, set its input wire as $w_1$, and let $\text{Root}$ be the set only contains that gate. Otherwise, $n > 1$, then we continue to do the following.

2. Find the largest $n'$ that is a power of 2, and build a perfect binary tree of $n'$ leaves, where each internal node of the tree is an AND gate, and each leaf node of the tree computes the identity function. For every $i \in [n']$, the $i$-th leaf node uses the wire $w_i$ as its input wire. Let $\text{Root}_{n'}$ be the single element set that only contains the root gate of the perfect binary AND tree.

3. If $n > n'$, we build a circuit $\Delta_{n-n'}$ recursively on the remaining $n-n'$ input wires $w_{n'+1}, \ldots, w_n$, and let $\text{Root}'$ be the set of the output gates of it. Update $\text{Root}_n := \text{Root}_n \cup \text{Root}'$.

4. Set the output as the outputs of $\text{Root}_n$.

Figure 2: Construction of the Binary AND-Tree circuit $\Delta_n$.

- **Append.** Define $\text{Root}_0 = \emptyset$ as the empty set and $\Delta_0$ as the empty circuit. For any integer $n \geq 1$, since $\Delta_{n-1}$ is a sub-circuit of $\Delta_n$, $\text{Root}_{n-1}$ is a subset of gates in $\Delta_n$. We have

$$\Pr[\forall i \in \text{Root}_{n-1}, w_i = 1, w_n = 1, \text{ and } \exists j \in \text{Root}_n, w_j \neq 1] \leq \text{poly}(n) \cdot \epsilon(\lambda),$$

where $\{w_i\}_{i \in \text{Root}_{n-1} \cup \text{Root}_n \cup \{n\}} \leftarrow \text{LocalGen}(\text{Root}_{n-1} \cup \text{Root}_n \cup \{n\})$.

**Proof.** Before we prove the properties, we first introduce the following notations.

- For any input wire $w_i$, we define $\text{PATH}_i$ as the path containing all the gates from the $i$-th leaf to one of the roots in $\text{Root}_n$. Note that $\Delta_n$ consists of several disjoint perfect binary AND trees, and hence there exists a unique binary tree that contains the $i$-th leaf.

- For any input wire $w_i$, we define $\text{OPEN}_i$ as the set that contains $\text{PATH}_i$ and also the left child and right child nodes of every non-leaf node in $\text{PATH}_i$. More generally, for any subset of input wires $S \subseteq [n]$, we denote $\text{OPEN}_S = \bigcup_{i \in S} \text{OPEN}_i$.

**Proof of “Read” Property.** Suppose $\text{LocalGen}$ is a $(4c[\log n], \epsilon)$-local assignment generator, for any subset of input wires $S \subseteq [n]$, we have $\text{LocalGen}$ extract all the gates in $\text{Root}_n \cup \text{OPEN}_S \cup S$. Then except with $\text{poly}(n) \cdot \epsilon(\lambda)$ probability over the randomness of $\text{LocalGen}$, the extracted gates and wires in $\text{Root}_n \cup \text{OPEN}_S \cup S$ are locally consistent. Namely, for each gate $x \in \text{Root}_n \cup \text{OPEN}_S$, then $w_x = w_l \land w_r$, where $w_l, w_r$ are the left and right input wires of $x$ and $w_x$ is the output of the gate $x$. 52
Hence, the gates in \( \text{Root}_n \cup \text{OPEN}_S \cup S \) form a subcircuit that computes an AND of its inputs, where the output gates are a subset of \( \text{Root}_n \). Moreover, if the extracted gates and wires are locally consistent, then the sub-circuit is computed correctly. Hence, if all the gates in \( \text{Root}_n \) output 1, then all the wires in \( S \) must also be 1’s. Hence, we have proven

\[
\Pr \left[ \forall i \in \text{Root}_n, w_i = 1 \land \exists i \in S, w_i \neq 1 \right] \leq \text{poly}(n) \cdot \epsilon(\lambda),
\]

where \( \{w_i\}_{i \in \text{Root}_n \cup \text{OPEN}_S \cup S} \leftarrow \text{LocalGen}(\text{Root}_n \cup \text{OPEN}_S \cup S) \).

Applying the no-signaling property again to have \( \text{LocalGen} \) only extract \( \text{Root}_n \cup S \), we finish the proof of the “read” property.

**Proof of “Increment” Property.** We prove the “increment” property inductively on the input length \( n \). For the base case \( n = 1 \), the property clearly holds. For the inductive step, assume that the property holds for any input length less than \( n \), now we prove the property holds for \( n \), where \( n \geq 2 \). To prove this, it suffices to show that \( \Delta_{n-1} \) is a sub-circuit of \( \Delta_n \) on the first \( (n - 1) \) input wires \( w_1, w_2, \ldots, w_{n-1} \). This is because for any \( k < n \), from the induction hypothesis, \( \Delta_k \) is a sub-circuit of \( \Delta_{n-1} \) on the first \( k \) input wires. Once we prove that \( \Delta_{n-1} \) is a sub-circuit of \( \Delta_n \), \( \Delta_k \) is also a sub-circuit of \( \Delta_n \) on the first \( k \) input wires, and hence we finish the proof.

Next, we show that \( \Delta_{n-1} \) is a sub-circuit of \( \Delta_n \) on the first \((n - 1)\) input wires. We have two cases depending on whether \( n \) is a power of 2.

If \( n \) is not a power of 2, then in Step 2 of the construction (Figure 2) of \( \Delta_{n-1} \) and \( \Delta_n \), we choose the same largest \( n' \) that is a power of 2 and \( n' \leq n - 1 \), and build the same perfect binary AND tree circuit over the first \( n' \) input wires. The remaining parts (Step 3) in the constructions of \( \Delta_{n-1} \) and \( \Delta_n \) are \( \Delta_{n-1-n'} \) and \( \Delta_{n-n'} \) on the remaining input wires, respectively. Now we can use the induction hypothesis to argue that \( \Delta_{n-1-n'} \) is also a sub-circuit of \( \Delta_{n-n'} \) on the first \((n - 1 - n')\) input wires. Hence, we have proven the property for \( n \) when \( n \) is not a power of 2.

In the case that \( n \) is a power of 2, then \( n = 2^m \) for some integer \( m \). In this case, \( \Delta_n \) is a perfect binary tree, where each gate is an AND gate. \( \Delta_{n-1} \) consists of \( m \) perfect binary AND trees, each of input length \( 2^i, i = m - 1, m - 2, \ldots, 0 \). Clearly, \( \Delta_{n-1} \) is a sub-circuit of \( \Delta_n \) on the first \((n - 1)\) input wires due to the structure of the perfect binary trees.

We have shown that \( \Delta_{n-1} \) is a sub-circuit of \( \Delta_n \) in both cases, and hence the “increment” property holds for any \( n \) from the inductive argument.

**Proof of “Append” Property.** We claim that for any \( n \), if all the gates in \( \text{OPEN}_n \) are locally consistent, then \( w_i = 1 \ \forall i \in \text{Root}_{n-1} \cup \{n\} \) implies that \( w_i = 1 \ \forall i \in \text{Root}_n \). Then we have \( \text{LocalGen} \) extract the wires in \( \text{Root}_{n-1} \cup \text{Root}_n \cup \text{OPEN}_n \). Except with \( \text{poly}(n) \cdot \epsilon(\lambda) \) probability, the wires in \( \text{OPEN}_n \) are locally consistent and then we can apply the claim. Finally, we use the no-signaling property to have \( \text{LocalGen} \) only extract the wires in \( \text{Root}_{n-1} \cup \text{Root}_n \cup \{n\} \) and finish the proof of the “append” property. Hence, it remains to prove the claim.

We will prove the claim inductively on the input length \( n \). For the base case \( n = 1 \), the claim clearly holds. We only need to prove the inductive step. Assume that the claim holds for any input length less than \( n \), and now we prove that the claim holds for the input length \( n \). We again have two cases depending on whether \( n \) is a power of 2.

If \( n \) is not a power of 2, then from the same argument in the proof of “increment” property, in Step 2 of the constructions of \( \Delta_{n-1} \) and \( \Delta_n \), they use the same \( n' \) that is a power of 2, and build the same binary AND trees over the first \( n' \) input wires. Since \( \text{OPEN}_n \) is fully contained in
the remaining part $\Delta_{n-n'}$, we can invoke the induction hypothesis, and use the fact that the claim holds for the circuits $\Delta_{n-n'}$. Moreover, Root$_{n-1}$ and Root$_n$ has the following structure.

$$\text{Root}_{n-1} = \{r\} \cup \text{Root}_{n-1-n'}, \quad \text{Root}_n = \{r\} \cup \text{Root}_{n-n'},$$

where $r$ is the root gate of the perfect binary AND tree of the first $n'$ input wires, and $\text{Root}_{n-1-n'}, \text{Root}_{n-n'}$ are the output gates of $\Delta_{n-1-n'}, \Delta_{n-n'}$ on the remaining input wires, respectively. Hence, $\forall i \in \text{Root}_{n-1} \cup \{n\}, w_i = 1$ now translated to $w_r = 1, \forall i \in \text{Root}_{n-1-n'} \cup \{n\}, w_i = 1$. $\forall i \in \text{Root}_{n-n'} \cup \{n\}, w_i = 1$ can be combined with the induction hypothesis to argue that $\forall i \in \text{Root}_{n-n'}, w_i = 1$. Hence, the claim holds for the input length $n$ when $n$ is not a power of 2.

If $n = 2^m$ is a power of 2 for some integer $m$, then $\Delta_n$ is a perfect binary AND tree with $n$ input wires, and $\Delta_{n-1}$ consists of $m$ perfect binary AND trees of input lengths $2^{m-1}, 2^{m-2}, \ldots, 2^0$. In this case, $\text{PATH}_n$ is the root-to-leaf path for the rightmost $n$-th input wire. Hence, the claim holds due to the perfect binary tree structure of $\Delta_n$.

From the induction argument, we prove the claim for any input length $n$. □

Given the helper circuit in Figure 2, we are ready to construct the extension circuit.

### 6.2 Construction of the Extension Circuit

In this section, for any set $S \subseteq \{0, 1\}^*$, we construct the extension circuit for any circuit family $\{C_x\}_{x \in S}$ with propositional proofs of unsatisfiability.

We first describe an initial attempt to construct the extension circuit $E_x$. We build the circuit $C_x$ as part of $E_x$, then we iterate through the lines of the propositional proof to process all the extension rules by adding new gates in $E_x$. Each new gate represents a new propositional variable. Next, we build a helper circuit $\Delta_\ell$ from Figure 2, where the input length $\ell$ is set to be the same as the number of lines in the propositional proof. Then we iterate through the propositional proof again, and add all the lines as sub-circuits in $E_x$, and set their outputs as the input wires of $\Delta_\ell$.

However, there is an issue in the above construction. Note that each line of the propositional proof might be of polynomial size in $n$, whereas in the proof of local unsatisfiability, we can only extract a small number of wires from the local assignment generator. This is an issue because we will need to extract the entire line later in our proof of unsatisfiability. To resolve this issue, we break down the lines by introducing new variables to the subformulas in the lines, and hence the size of each line can be a constant.

The following Lemma 6.5 shows that we can convert any extended Frege proof to a new extended Frege proof where each line is constant size, and the number of lines only blows up a polynomial.

**Lemma 6.5** (Implicit in [Sta77]). Let $T$ be a set of formulas, and $\theta$ be a formula. For any extended Frege proof $(\theta_1, \ldots, \theta_\ell)$ of $T \vdash \theta$, there exists another extended Frege proof $(\theta'_1, \ldots, \theta'_{\ell'})$ of $T \vdash \theta$, where $\ell' = \text{poly}(\ell)$ and the size of each line is bounded by $\text{poly}(|\theta|)$.

**Proof Sketch.** The proof of the lemma is also implicit in [Kra95], Lemma 4.5.7. We provide a sketch here to be self-contained. Let $Q$ be a set of formulas which contains all the subformulas in the lines of the propositional logic proof $(\theta_1, \ldots, \theta_\ell)$. Next, we assign each formula $\phi$ in $Q$ a new propositional variable $V[\phi]$ if $\phi$ itself is not already a propositional variable nor $F$, otherwise, we let $V[\phi]$ be $\phi$ itself.
Then for each formula $\phi \in Q$, we add an extension rule for $V[\phi]$, as follows. If $\phi$ is in the form $\alpha \rightarrow \beta$ from some formulas $\alpha, \beta$, then we add the extension $V[\phi] \leftrightarrow (V[\alpha] \rightarrow V[\beta])$. Otherwise, if $\phi$ is in the form $\neg \alpha$, then we add the extension $V[\phi] \leftrightarrow \neg V[\alpha]$.

For each line $\theta_i, i \in [\ell]$ in the original proof, we will derive the $V[\theta_i]$ as a line in the new proof of constant size. For each $\theta_i$, if $\theta_i$ is a promise or an extension, then we can derive $V[\theta_i]$ via $\text{poly}(|\theta_i|)$ lines of constant size proofs. Otherwise, if $\theta_i$ is derived from the previous lines $\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_c}$ using an inference rule, then we can also derive $V[\theta_i]$ from $V[\theta_{i_1}], \ldots, V[\theta_{i_c}]$ via a constant size of lines.

Applying Lemma 6.5 to the propositional proof of unsatisfiability, we can always assume without loss of generality that every line in the propositional proof of unsatisfiability is a constant size formula. We formally state this in the following corollary.

**Corollary 6.6.** There exists a constant $C > 0$ such that, for any set $S \subseteq \{0, 1\}^*$, if a circuit family $\{C_x\}_{x \in S}$ has $\ell$-propositional proof of unsatisfiability for a polynomial function $\ell = \ell(n)$, then $\{C_x\}_{x \in S}$ has a $\text{poly}(\ell)$-propositional proof of unsatisfiability, where the size of each line in the proof is at most $C$.

**Construction of the Extension Circuit** $E_x$. For any integer $n$ and $x \in \{0, 1\}^n \cap S$, suppose that $\langle \theta_1, \theta_2, \ldots, \theta_\ell \rangle$ is the propositional logic proof in the Extended Frege system for $\text{Prop}[C_x] \vdash \text{out} \leftrightarrow F$,

where out is the output wire of $C_x$, and each line $\theta_i$ is a constant size formula. We construct the extension circuit $E_x$ for $C_x$ as follows.

**Input:** wires $w_1, w_2, \ldots, w_n$.

**Output:** single bit.

1. Compute $C_x(w_1, \ldots, w_n)$. Each gate in $C_x$ is treated as a variable in $\text{Prop}[C_x]$. Hence, each propositional variable in $\text{Prop}[C_x]$ is associated to a gate in the circuit. For each input wire $w_i$, we also build a gate which takes $w_i$ as input, and computes an identity function. Then we treat the gate as the propositional variable $w_i$.

2. **Adding Extensions.** We iterate through the propositional proof, and process each extension rule in the proof by specifying how each new variable is computed in the circuit $E_x$.

More specifically, for each $i = 1, 2, \ldots, \ell$, if $\theta_i$ is an extension, then $\theta_i$ is in the form $v_i \leftrightarrow \phi_i$, where $v_i$ is the new variable and $\phi_i$ is a propositional formula of the existing variables. Then we build a circuit $E_i$ which takes the variables that appear in $\phi_i$ as its input and computes $\phi_i$. Finally, we set the output gate of $E_i$ as $v_i$.

3. **Adding Proofs.** We build a circuit $\Delta_\ell$ from Lemma 6.4, where the input length is set to be the same as the number of lines. Then we iterate through the lines in the propositional proof and add each line to the input wires to $\Delta_\ell$.

Specifically, for each $\theta_i, i \in [\ell]$, we build a circuit $C_i$ computing $\theta_i$. $C_i$ takes the wires corresponding to the propositional variables that appear in $\theta_i$ as input. Then we connect the output gate of $C_i$ to the $i$-th input of $\Delta_\ell$.

4. Set the output of $E_x$ as the output of $C_x$. 55
We will prove that the above construction of $E_x$ satisfies Definition 4.4. The first property in Definition 4.4 follows directly from the construction, since we compute $C_x$ in Step 1. We now proceed to prove the second property in Definition 4.4, which is the local unsatisfiability.

### 6.3 Proof of Local Unsatisfiability

If $C_x$ has a propositional proof of unsatisfiability, then $C_x$ is certainly unsatisfiable, and we will prove that there exists a constant $A$ such that, for any function $\epsilon = \epsilon(\lambda)$, and any $(A \log \ell, \epsilon)$-local assignment generator $\text{LocalGen}$ for $E_x$, $\Pr[w_{\text{out}} = 1]$ can be bounded. To show this, we first prove the following lemma.

**Lemma 6.7.** There exists a constant $A > 0$ such that for any function $\epsilon = \epsilon(\lambda)$, any $(A \log \ell, \epsilon)$-local assignment generator $\text{LocalGen}$ of $E_x$, we have

\[
\Pr[\exists i \in \text{Root}_\ell, w_i \neq 1] \leq \text{poly}(n) \cdot \epsilon(\lambda).
\]

Lemma 6.7 follows from the following Lemma 6.8 by a direct inductive argument on the index $k \in [\ell]$. In each induction step, we switch from $\text{LocalGen}(\text{Root}_{k-1})$ to $\text{LocalGen}(\text{Root}_k)$, and use the no-signaling property of the local assignment generator to ensure that the gate values in $\text{Root}_k$ are all 1’s, except with a small probability. Next, we state and prove Lemma 6.8.

**Lemma 6.8.** There exists a constant $A > 0$ and a polynomial $q = q(n)$, such that for any function $\epsilon = \epsilon(\lambda)$, for any $(A \log \ell, \epsilon)$-local assignment generator $\text{LocalGen}$ of $E_x$, for every $k \in [\ell]$,

\[
\Pr[\forall i \in \text{Root}_{k-1}, w_i = 1, \exists i \in \text{Root}_k w_i \neq 1] \leq q(n) \cdot \epsilon(\lambda),
\]

(7)

where $\{w_i\}_{i \in \text{Root}_{k-1} \cup \text{Root}_{k-1}} \leftarrow \text{LocalGen}(\text{Root}_{k-1} \cup \text{Root}_k)$, and $\text{Root}_k$ is the set specified in the Lemma 6.4.

**Proof.** We will use the “append” property of $\Delta_\ell$ to prove the lemma. Namely, we will extract $\text{Root}_{k-1} \cup \text{Root}_k \cup \{y_k\}$ from $\text{LocalGen}$, where $y_k$ is the gate connecting to the $k$-th input of $\Delta_\ell$. We will leverage the “read” property from Lemma 6.4 to argue that $y_k$ is also 1 if all the gates in $\text{Root}_{k-1}$ output 1’s. Then we can apply the “append” property.

Specifically, we rely on the following information theoretical property about $E_x$ to argue that $y_k$ must also be 1 if all gates in $\text{Root}_{k-1}$ output 1’s. We defer the proof of the claim to the end of the proof.

**Claim 6.9.** For each index $k \in [\ell]$, there exists a constant number of indices $i_1, \ldots, i_c < k$ and a constant-size set of gates $S$ in $E_x$ such that, if the gates in $S$ are locally consistent, and $y_{i_1}, \ldots, y_{i_c}$ all output 1’s, then $y_k$ must also output 1. Here, $y_{i_1}, \ldots, y_{i_c}$ are the gates connecting to the $i_1$-th, ..., $i_c$-th input wires of $\Delta_\ell$, respectively.

From the claim, there exists a set $S$ of constant size such that if the gates in $S$ are locally consistent, and $y_{i_1}, \ldots, y_{i_c}$ are all 1’s, then $y_k$ must also be 1. Hence, we further have $\text{LocalGen}$ extract $S \cup \{y_{i_1}, \ldots, y_{i_c}\}$. Then we use the “read” property from Lemma 6.4 to argue that if all the gates in $\text{Root}_{k-1}$ output 1’s, then $\{y_{i_1}, \ldots, y_{i_c}\} \cup \text{Root}_{k-1}$ are all 1’s, except with probability $\text{poly}(n) \cdot \epsilon(\lambda)$. Moreover, except with an additional $\text{poly}(n) \cdot \epsilon(\lambda)$ probability, the gates in $S$ are locally consistent, then $y_k$ must also output 1 from Claim 6.9. Relying on the no-signaling property, we can argue that if we only extract $\text{Root}_{k-1} \cup \text{Root}_k \cup \{y_k\}$ and all the gates in $\text{Root}_{k-1}$ output 1’s, then $y_k$ must also outputs 1, except with probability $\text{poly}(n) \cdot \epsilon(\lambda)$. 56
Now we are ready to apply the “append” property from Lemma 6.4, and argue that the extracted wires in Root\(_k\) are all 1’s, if all gates in Root\(_{k-1}\) ∪ \(y_k\) output 1’s, except with probability \(\text{poly}(n) \cdot \epsilon(\lambda)\). We finish the proof.

Finally, we prove Claim 6.9.

**Proof of Claim 6.9.** Depending on how \(\theta_k\) is derived, we have the following cases.

**Premises.** If \(\theta_k\) is a premise, then we let \(S\) be the following set of gates in \(E_x\). \(S\) contains

- All the gates that correspond to the variables appeared in the premise.
- All the gates in the sub-circuit \(C_k\) in Step 3.

For example, if the premise is a gate computing \(a : (b \odot c)\) for a function “\(\odot\)”, then we extract the wires \(a, b, c\), and also the gates in \(C_k\), which takes \(a, b, c\) as input, and computes the truth value of the formula \(a : (b \odot c)\) as its output. The case when the premise is a negation gate in \(C_x\) is similar.

If the gates in \(S\) are locally consistent, i.e. \(a = b \odot c\), then \(C_k\) will output 1, i.e., the gate \(y_k\) outputs 1.

**Extensions.** If \(\theta_k\) is an extension \(v : \phi\), where \(v\) is the new variable and \(\phi\) is a formula. Recall that, in Step 2, we add a sub-circuit \(E_k\) which takes all the variables in \(\phi\) as input and computes the truth value of \(\phi\), and in Step 3, we add a sub-circuit \(C_k\) which computes the truth value of \(v : \phi\) from the gate \(v\) and all the gate representing variables in \(\phi\). Hence, we let \(S\) be the set of gates in \(E_k\) and \(C_k\). If the gates in \(S\) are locally consistent, then the truth value of \(v\) equals the truth value of \(\phi\), and hence \(C_k\) outputs 1, which means the gate \(y_k\) outputs 1.

**Inferences.** In this case, there exist integers \(i_1, i_2, \ldots, i_c < k\) for a constant \(c\) such that

\[
\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_c} \vdash \theta_k
\]

is an application of an inference rule (See Section 3.6). Then the formula \(\theta_{i_1} \land \theta_{i_2} \land \ldots \land \theta_{i_c} \rightarrow \theta_k\) must be a tautology. We let \(\{i_1, \ldots, i_c\}\) be the set of indices required in the statement of Claim 6.9, and let \(S\) be the set of gates in \(C_{i_1}, C_{i_2}, \ldots, C_{i_c}, C_k\). Recall that, the sub-circuit \(C_i\) is defined in Step 3 for every \(i \in [\ell]\). If the gates in \(S\) are locally consistent, then \(C_{i_1}, \ldots, C_{i_c}, C_k\) compute the truth values of \(\theta_{i_1}, \ldots, \theta_{i_c}, \theta_k\), respectively. Moreover, if \(y_{i_1}, \ldots y_{i_c}\) are all 1’s, then all of \(C_{i_1}, \ldots, C_{i_c}\) output 1’s, and hence \(C_k\) must also output 1, which follows from the property of tautology.

This finishes the proof of the claim.

**Lemma 6.10.** There exists a constant \(A > 0\) such that, the construction of \(\{E_x\}_{x \in S}\) in Section 6.2 is an \(A \log n\)-locally unsatisfiable extension of the circuit family \(\{C_x\}_{x \in S}\).

**Proof.** The first property of Definition 4.4 is satisfied by the construction of \(E_x\) in Section 6.2, since \(C_x\) is part of the circuit \(E_x\).

Now we only need to prove that the construction of \(E_x\) satisfies the second property in Definition 4.4. Let \(\epsilon = \epsilon(\lambda)\) be a function and let \(\text{LocalGen}\) be an \((A \log n, \epsilon)\)-local assignment generator for \(E_x\). From Lemma 6.7, if we extract \(\{w_i\}_{i \in \text{Root}_\ell} \leftarrow \text{LocalGen}(\text{Root}_\ell)\), then \(\forall i \in \text{Root}_\ell, w_i = 1\) with probability \(1 - \text{poly}(\ell(n)) \cdot \epsilon(\lambda)\). Then we apply the “read” property of \(\Delta_\ell\) to show that, if we
extract \( y_\ell \) from \( \text{LocalGen}(\{\ell\}) \), where \( y_\ell \) is the \( \ell \)-th input wire of \( \Delta_\ell \) and \( \ell \) is the index for \( y_\ell \), we have \( y_\ell = 1 \) with probability \( 1 - \text{poly}(\ell(n)) \cdot \epsilon(\lambda) \).

Moreover, note that the last line of the propositional logic proof \( \theta_\ell \) must be the statement \( w_{\text{out}} \leftrightarrow F \), where \( w_{\text{out}} \) is the output wire of \( C_x \). If we extract \( \text{LocalGen}(\{y_\ell\} \cup \{\text{out}\}) \), then from the no-signaling property of \( \text{LocalGen} \), \( y_\ell = 1 \) except with probability \( 1 - \text{poly}(\ell(n)) \cdot \epsilon(\lambda) \).

Furthermore, except with probability \( O(\epsilon(\lambda)) \), the computation of the wire \( y_\ell := w_{\text{out}} \leftrightarrow F \) is correct. Then \( w_{\text{out}} \leftrightarrow F \) happens with probability \( 1 - \text{poly}(\ell(n)) \cdot \epsilon(\lambda) \). Finally, applying the no-signaling property again to only extract the output wire \( \text{out} \), we have that \( w_{\text{out}} = 1 \) with probability at most \( \text{poly}(\ell(n)) \cdot \epsilon(\lambda) \). This finishes the proof.

Theorem 6.2 follows directly from Lemma 6.10.

7 Locally Unsatisfiable Extensions from Bounded Space Propositional Proofs of Unsatisfiability

In this section, we show that if a circuit family has a polynomially bounded space propositional proof of unsatisfiability, then we can build a locally unsatisfiable extension with locally computable property (Definition 5.1). We will first define the space complexity of propositional proofs (extended Frege system) in Section 7.1.

7.1 Space Complexity of Propositional Proofs

We define the propositional proofs of bounded space in the Extended Frege system as follows. The space complexity of propositional proofs has been defined [ET01,ABSRW02] in logic proof systems without the “extension rule”. In this work, we extend those definitions to Frege systems with extension rules. Specifically, we allow the introduction of new propositional variables in the proof, and we count the size of the defining formula of the new variable into the space measure. When we erase an existing formula in the configuration, if the formula is a definitional formula for some variable, then we require that the variable no longer appear later in the proof. Formally, we define the space complexity as follows in Definition 7.1.

**Definition 7.1.** Let \( T \) be a set of propositional formulas, and \( \theta \) be a propositional formula, we say \( T \vdash \theta \) has a \((t, s)\)-propositional proof, if there exists a sequence of “configurations” \( M_1, M_2, \ldots, M_u \), which are sets of formulas, with the following property. For any \( i \in [u] \), define \( M_0 \) as the empty set, then \( M_i \) is derived from \( M_{i-1} \) via one of the following rules.

- **Premise Download.** There exists a formula \( \phi \in T \) such that
  \[
  M_i = M_{i-1} \cup \{\phi\}.
  \]

- **Extension.** There exists a variable \( v \) and a formula \( \phi \) such that \( v \) does not appear in \( M_1, \ldots, M_{i-1}, \phi \), and the premise \( T \), and
  \[
  M_i = M_{i-1} \cup \{v \leftrightarrow \phi\}.
  \]

- **Inference.** There exists a formula \( \phi \) and \( \phi_1, \phi_2, \ldots, \phi_c \in M_{i-1} \) such that
  \[
  M_i = M_{i-1} \cup \{\phi\}.
  \]

Moreover, \( \phi_1, \phi_2, \ldots, \phi_c \vdash \phi \) follows from an inference rule.
• **Erasure.** There exists a formula \( \phi \in M_{i-1} \) such that

\[
M_i = M_{i-1} \setminus \{ \phi \},
\]

where \( \phi \) is either

– an extension of a variable \( v \) and \( v \) does not appear in \( M_i \), or

– \( \phi \) is derived via an inference rule, or

– \( \phi \) is a premise in \( T \).

Moreover, we require that the last configuration \( M_u \) must contain \( \theta \). The total size of the formulas in each \( M_i \) is at most \( s \) for every \( i \in [u] \), and \( \sum_{i=1}^{u} \sum_{\phi \in M_i} |\phi| \) is at most \( t \).

We say \( T \vdash \theta \) has a propositional proof of length \( t \) and space \( s \), if \( T \vdash \theta \) has a \((t, s)\)-propositional proof.

Similar to Definition 6.1, we define bounded space propositional proofs of unsatisfiability as follows.

**Remark 7.1.** We remark that there is an alternative way to define the space complexity of extended Frege systems. Namely, when we erase a defining formula for some variable, instead of requiring the variable no long to appear in the proof later, the alternative definition allows the variable to appear later in the proof, and only prohibits the use of its defining formula. In this work, we choose Definition 7.1 as our space complexity definition.

**Definition 7.2 (Bounded Space Propositional Proofs of Unsatisfiability).** For any set \( S \subseteq \{0, 1\}^* \) and any circuit family \( \{C_x\}_{x \in S} \) over \( S \), let \( t = t(n) \), \( s = s(n) \) be two functions in \( n \), we say \( \{C_x\}_{x \in S} \) has a \((t, s)\)-propositional proof of unsatisfiability, if there exists a \((t(n), s(n))\)-propositional proof of

\[
\text{Prop}[C_x] \vdash \text{out} \leftrightarrow \text{F},
\]

for every \( x \in S \cap \{0, 1\}^n \), where \( \text{out} \) is the output wire of \( C_x \), and \( \text{Prop}[C_x] \) is the set of formulas defined in Definition 6.1.

### 7.2 Locally Unsatisfiable Extensions from Bounded Space Proofs of Unsatisfiability

In this section, we prove the following theorem, which shows that if a circuit family has bounded space propositional proofs of unsatisfiability, then it has a locally unsatisfiable extension that is locally computable.

**Theorem 7.3.** For any set \( T \subseteq \{0, 1\}^* \) and any family of circuits \( \{C_x\}_{x \in T} \), let \( t = t(n) \) and \( s = s(n) \) be two functions, if \( \{C_x\}_{x \in T} \) has a \((t, s)\)-propositional proof of unsatisfiability, then there exists a circuit family \( \{E_x\}_{x \in T} \) such that,

- There exists a constant \( A > 0 \), \( \{E_x\}_{x \in T} \) is an \( A \log n \)-locally unsatisfiable extension of \( \{C_x\}_{x \in T} \).

- \( \{E_x\}_{x \in T} \) is \( \text{poly}(s) \)-locally computable.

- The size of \( E_x \) is bounded by \( \text{poly}(t(n), s(n)) \) for any \( x \in T \cap \{0, 1\}^n \).
We will prove that the helper circuits \( \{ \Delta_n \}_{n \in \mathbb{N}} \) in Section 6.1 satisfy some additional properties. We first introduce a new subset of gates \( \text{Root}_k \).

Recall that, for each \( k \in [n] \), we denote \( \text{PATH}_k \) as the set of gates in the root-to-leaf path of the \( k \)-th input wire in the circuit \( \Delta_n \).

**Definition 7.4 (Set \( \text{Root}_k \)).** For every \( k \in [n] \), we define \( \text{Root}_k \) be the set that contains the root nodes of \( \Delta_n \) after deleting the nodes in \( \text{PATH}_k \).

Clearly, \( |\text{Root}_k| = O(\log n) \). \( \text{Root}_k \) contains the roots of some subtrees, whose leaf nodes contain all the leaves in \( \Delta_n \) except the \( k \)-th leaf. Then \( \text{Root}_k \) satisfies that, all the gates in \( \text{Root}_k \) output 1’s if and only if all the input wires in \( [n] \setminus \{k\} \) are all 1’s. Formally, we prove the following properties about \( \text{Root}_k \).

**Lemma 7.5.** For any integer \( n \), the helper circuit \( \Delta_n \) constructed in Figure 2 satisfies the following property. For any index \( k \in [n] \) of the input wires, there exists a subset of gates \( \text{Root}_k \) of size \( |\text{Root}_k| = O(\log n) \).

Moreover, there exists a constant \( A > 0 \) such that, for any function \( \epsilon = \epsilon(\lambda) \), for any \((A \log n, \epsilon)\)-local assignment generator \( \text{LocalGen} \) for \( \Delta_n \), we have the following properties.

- **Split.** For any input wire \( k \in [n] \), if all the gates in \( \text{Root}_k \) output 1’s, then all the gates in \( \text{Root}_k \) must also output 1’s. Formally,

\[
\Pr \left[ \forall i \in \text{Root}_k, w_i = 1 \land \exists i \in \text{Root}_k, w_i \neq 1 \right] \leq \text{poly}(n) \cdot \epsilon(n),
\]

where \( \{w_i\}_{i \in \text{Root}_k} \leftarrow \text{LocalGen}(\text{Root}_k \cup \text{Root}_k) \).

- **Merge.** If all the gates in \( \text{Root}_k \) output 1’s and the \( k \)-th input wire \( w_k \) is also 1, then all the gates in \( \text{Root}_k \) must also output 1’s. Formally,

\[
\Pr \left[ \forall i \in \text{Root}_k, w_i = 1 \land w_k = 1 \land \exists i \in \text{Root}_n, w_i \neq 1 \right] \leq \text{poly}(n) \cdot \epsilon(n),
\]

where \( \{w_i\}_{i \in \text{Root}_k \cup \text{Root}_n \cup \{k\}} \leftarrow \text{LocalGen}(\text{Root}_k \cup \text{Root}_n \cup \{k\}) \).

**Proof.** Recall that we denote \( \text{OPEN}_k \) as the set of gates that contains \( \text{PATH}_k \), as well as the gates that produce the input wires to the gates in \( \text{PATH}_k \) and \( \Delta_n \) consists of a series of perfect binary trees.

We will use the following information-theoretic property of \( \Delta_n \) to prove both properties. We defer the proof of the claim to the end of the proof.

**Claim 7.6.** If the gates and wires in \( \text{OPEN}_k \cup \text{Root}_n \cup \{k\} \) are locally consistent, then all the gates in \( \text{Root}_n \) output 1’s, if and only if all the gates in \( \text{Root}_k \) outputs 1’s and the \( k \)-th input wire is 1.

**Proof of “Split” Property.** For any local assignment generator \( \text{LocalGen} \), if we extract \( \text{OPEN}_k \cup \text{Root}_n \cup \{k\} \), then except with \( O(\log n) \cdot \epsilon \) probability, all the gates in \( \text{OPEN}_k \cup \text{Root}_n \) are locally consistent. From the “only if” part of **Claim 7.6**, if all the gates in \( \text{Root}_n \) output 1, then all the gates in \( \text{Root}_k \) also output 1. Using the no-signaling property to only extract the gates in \( \text{Root}_n \cup \text{Root}_k \), we prove the split property.
Proof of “Merge” Property. We use LocalGen to extract $\text{OPEN}_k \cup \text{Root}_n \cup \{k\}$. Except with probability $O(\log n) \cdot \epsilon$, the extracted gates are locally consistent. From the “if” part of Claim 7.6, if $\tilde{\text{Root}}_k$ output 1’s and $w_k = 1$, then $\text{Root}_n$ output 1’s. Applying the no-signaling property to only extract $\text{Root}_n \cup \tilde{\text{Root}}_k \cup \{k\}$, we finish the proof.

Proof of Claim 7.6. Note that $\tilde{\text{Root}}_k$ almost contains all the gates in $\text{Root}_n$, except the root node of the perfect binary tree that contains the $k$-th input wire. Namely, let $\text{rt}$ denote the root node of the trees that contains the $k$-th input wire in $\Delta_n$. Then $\text{Root}_n \setminus \tilde{\text{Root}}_k = \{\text{rt}\}$. Moreover, the gate $\text{rt}$ can be computed via a subcircuit formed by the gates in $\text{OPEN}_k$. The subcircuit simply computes an AND of gates in $\tilde{\text{Root}}_k \setminus \text{Root}_n$ and $w_k$. Hence, if the gates in $\text{OPEN}_k$ are locally consistent, then $w_{\text{rt}} = 1$ if and only if $w_k = 1$ and all gates in $\tilde{\text{Root}}_k \setminus \text{Root}_n$ output 1. This finishes the proof of the claim, because $\{w_{\text{rt}}\} \cup (\tilde{\text{Root}}_k \cap \text{Root}_n) = \text{Root}_n$ and $(\tilde{\text{Root}}_k \setminus \text{Root}_n) \cup (\tilde{\text{Root}}_k \cap \text{Root}_n) = \tilde{\text{Root}}_k$. □

Similar to Corollary 6.6, we have the following lemma for bounded space propositional proofs. Intuitively, it states that we can always break the lines in a bounded space propositional proof to constant-size lines while preserving the space.

Lemma 7.7. For any set $T \subseteq \{0,1\}^*$ and functions $t = t(n), s = s(n)$, if there exists a $(t,s)$-propositional proof of unsatisfiability for a circuit family $\{C_x\}_{x \in T}$, then there exists a $(\text{poly}(s,t), \text{poly}(s))$-propositional proof of unsatisfiability, where each line in the proof is a constant-size formula.

The proof of the lemma follows the same strategy as Lemma 6.5.

Construction of Locally Computable Extension $E_x$. Our construction will use the helper circuit $\{\Delta_n\}_{n \in \mathbb{N}}$ in Figure 2 as an ingredient.

Let $T \subseteq \{0,1\}^*$ be a set, and let $M_1, M_2, \ldots, M_t$ be a $(t(n), s(n))$-propositional proof of

$$\text{Prop}[C_x] \vdash \text{out} \leftrightarrow \text{F},$$

where out is the output wire of $C_x$. From Lemma 7.7, we assume without loss of generality that the formulas in $M_t$ are of constant sizes.

Input: wires $w_1, \ldots, w_n$

Output: single bit.

1. Compute $C_x(w_1, \ldots, w_n)$.

2. Adding Extensions. This part is exactly the same as the Step 2 in the construction of $E_x$ in Section 6.2. Namely, if $\theta_i$ is an extension $v \leftrightarrow \phi$, we add a new sub-circuit $E_i$ computing $\phi$, and set the output gate of $E_i$ as the gate representing $v$.

3. Adding Lines. We build a helper circuit $\Delta_s^0$ of input length $s = s(n)$. $\Delta_s^0$ is identical to the circuit $\Delta_s$ in Section 6.2. We set all the input wires to $\Delta_s^0$ as 1’s, and let $\text{UNUSED} = [s]$ be the set of indices of all the “unused” input wires.

We iterate through the sets $M_1, M_2, \ldots, M_t$ one-by-one. At the $i$-th iteration $(i \in [t])$, we will find a subcircuit $\Delta_s^i$ of $E_x$, which is identical to $\Delta_s$. We will build a new subcircuit $C_i$, and select an input wire $j_i$ to $\Delta_s^{i-1}$.

Denote $M_0$ as the empty set. Depending on how $M_i$ is derived from $M_{i-1}$, we do the following.
- **Premise, Extension, or Inference.** Then $M_i = M_{i-1} \cup \{\phi\}$ for some formula $\phi$. We build a subcircuit $C_i$ computing $\phi$. The input to $C_i$ are the gates representing the proportional variables in $\phi$. Take an arbitrary index $j_i \in \text{UNUSED}$. We update $\text{UNUSED} := \text{UNUSED} \setminus \{j_i\}$.

- **Erasure.** In this case, $M_i = M_{i-1} \setminus \{\phi\}$ for some formula $\phi \in M_{i-1}$.
  Let $j_i$ be the index in $[s] \setminus \text{UNUSED}$ such that the $j_i$-th input to $\Delta_s^{i-1}$ is a subcircuit computing $\phi$.
  Build $C_i$ as a new subcircuit that always outputs 1.
  We update $\text{UNUSED} := \text{UNUSED} \cup \{j_i\}$.

4. Generate a new root-to-leaf path for the $j_i$-input wire of $\Delta_s^{i-1}$. Then we set the new $j_i$-th input wire as the output of $C_i$.
Specifically, recall that, $\Delta_s^{i-1}$ consists of a series of perfect binary trees, and we use $\text{PATH}_{j_i}$ to denote the set containing all the gates from the $j_i$-th input wire to the root of the tree that contains the $j_i$-th input wire.

(a) For each gate $g$ in $\text{PATH}_{j_i}$, we add a new gate $g'$ computing the same function as $g$. If $g$ is a non-leaf node, then we set the input wires to $g$ as follows.

(b) Let $g_l, g_r$ be the gates producing the input wires to $g$. If $g_l \in \text{PATH}_{j_i}$ (resp. $g_r \in \text{PATH}_{j_i}$), then $g_l$ (resp. $g_r$) is on the root-to-leaf path of the $j_i$-th input wire, then we define $u$ as the new gate $g'_l$ (resp. $v = g'_r$). Otherwise, we define $u$ as the old gate $g_l$ (resp. $v = g_r$).

(c) We set the input wires of $g'$ as the outputs of $u$ and $v$.

We define $\Delta_s^i$ as the updated version of $\Delta_s^{i-1}$ containing the new root-to-leaf path. We illustrate in Figure 3 for an example.

5. We set the $j_i$-th input wire to $\Delta_s^i$ as the output of $C_i$.

6. Set the output gate as the same output gate of $C_s(w_1, \ldots, w_n)$.

![Figure 3: An example of adding a new leaf-to-root path for $j_k = 2$. Left: original $\Delta_{j_k}^{i-1}$ circuit. Right: $\Delta_{j_k}^{i-1}$ and $\Delta_s^i$ after adding a new leaf-to-root path for the second leaf node. The new gates in $\Delta_{j_k}^{i-1}$ are marked in circles.](image)

**Lemma 7.8.** There exists a constant $A > 0$ such that, the above construction of $\{E_x\}_{x \in T}$ is an $A \log n$-locally unsatisfiable extension of $\{C_x\}_{x \in T}$.
We prove the lemma following the same strategy as Lemma 6.7. We first prove the following Lemma 7.9. Then we use an inductive argument on the index of the configurations \( k \in [\ell(n)] \). In each step of the inductive argument, we argue that the gates in \( \text{Root}(\Delta^k_s) \) output 1’s, except with a small probability \( \text{poly}(s, t) \cdot \epsilon \). We use the no-signaling property and Lemma 7.9 to transit from \((k-1)\) to \(k\). Finally, we derive that all gates in \( \text{Root}(\Delta^{[\ell(n)]}_s) \) output 1’s except with probability \( \text{poly}(s, t) \cdot \epsilon \). The rest of the proof follows the same strategy as Lemma 6.10.

**Lemma 7.9.** There exists a constant \( A > 0 \) and a polynomial \( q = q(s) \) such that, for any \( A \log s \)-local assignment generator \( \text{LocalGen} \), the following holds for every \( k \in [\ell(n)] \),

\[
\Pr \left[ \forall i \in \text{Root}(\Delta^k_s), w_i = 1 \land \exists i \in \text{Root}(\Delta^k_s), w_i \neq 1 \right] \leq q(s) \cdot \epsilon.
\]

**Proof Sketch.** The proof follows the same strategy as Lemma 6.8. The only difference is that, here, we need to deal with dynamic addition and deletion of the formulas on the leaves of \( \Delta_s \).

For every fixed \( k \), depending on how \( M_k \) is derived from \( M_{k-1} \), we have the following cases.

**Premise, Extension, or Inference.** In these cases, \( M_k = M_{k-1} \cup \{\phi\} \) for some formula \( \phi \). Let \( j_k \) be the output gate of the subcircuit \( C_k \). We will use the following claim modified from Claim 6.9 to fit in our bounded space setting. The proof of Claim 7.10 follows the exact same strategy as Claim 6.9.

**Claim 7.10.** There exists a constant-size subset \( S \) of gates in \( E_x \), and a constant number of input wires \( i_1, i_2, \ldots, i_c \) of \( \Delta^k_s \) which are not the output wire of \( j_k \), such that if the gates in \( S \) are locally consistent, and all the \( i_1 \)-th, \( i_2 \)-th, \ldots, \( i_c \)-th input wire of \( \Delta^k_s \) are 1’s, then \( j_k \) must also output 1.

Given Claim 7.10, we prove the lemma as follows. We first use \( \text{LocalGen} \) to extract \( \text{Root}(\Delta^k_s-1) \cup \{i_1, i_2, \ldots, i_c\} \) in the subcircuit \( \Delta^k_s-1 \). We can do this, because \( i_1, \ldots, i_c \)-th input wires of \( \Delta^k_s \) are also input wires of \( \Delta^k_s-1 \). Applying the “read” property in Lemma 6.4 to \( \Delta^k_s-1 \), we have that if \( \text{Root}(\Delta^k_s-1) \) output 1’s, then the input wires \( i_1, \ldots, i_c \) are all 1’s, except with probability \( \text{poly}(s(n)) \cdot \epsilon(\lambda) \).

Now we extract \( \text{Root}(\Delta^k_s-1) \cup \{i_1, i_2, \ldots, i_c\} \cup S \cup \{j_k\} \) from \( \text{LocalGen} \). From the no-signaling property, except with probability \( \text{poly}(s(n)) \cdot \epsilon(\lambda) \), we still have that if \( \text{Root}(\Delta^k_s-1) \) are all 1’s, then the wire values of \( \{i_1, \ldots, i_c\} \) are all 1’s. Moreover, except for an additional \( |S| \cdot \epsilon(\lambda) = \text{poly}(s(n)) \cdot \epsilon(\lambda) \) probability, the gates in \( S \) are locally consistent. From Claim 7.10, this implies that the wire value of \( j_k \) must also be 1 if \( \text{Root}(\Delta^k_s-1) \) are all 1’s, except for some small probability.

Next, using the “split” property from Lemma 7.5, we know that if we extract \( \text{Root}(\Delta^k_s-1) \cup \text{Root}_{j_k} \) in \( \Delta^k_s-1 \) and \( \text{Root}(\Delta^k_s-1) \) are all 1’s, then \( \text{Root}_{j_k} \) are all 1’s. Note that from our construction of \( \text{Root}_{j_k} \), the set \( \text{Root}_{j_k} \) in \( \Delta^k_s-1 \) is also the set \( \text{Root}_{j_k} \) in \( \Delta^k_s \). Hence, we can use the no-signaling property to combine this with the conclusion in the last paragraph. That is, if we extract \( \text{Root}(\Delta^k_s-1) \cup \text{Root}_{j_k} \cup \{j_k\} \), then if \( \text{Root}(\Delta^k_s-1) \) are all 1’s, then \( \text{Root}_{j_k} \) are all 1’s, and \( j_k \) also outputs 1, except with \( \text{poly}(s) \cdot \epsilon \) probability. Now we can apply the “merge” property from Lemma 7.5 to obtain that if \( \text{Root}(\Delta^k_s-1) \) are all 1’s, then \( \text{Root}(\Delta^k_s) \) are all 1’s, except for some small probability. We finish the proof in this case.
Erasure.  In this case, $M_k = M_{k-1} \setminus \{\phi\}$ for some formula $\phi \in M_{k-1}$. Let $j_k$ be the output gate of $C_t$.

We first apply the “split” property from Lemma 7.5 and no-signaling property to show that, if we extract $\text{Root}(\Delta_n^{k-1}) \cup \text{Root}(\Delta_n^k) \cup \text{Root}_k \cup \{j_k\}$ from $\text{LocalGen}$ and $\text{Root}(\Delta_n^{k-1})$ are all 1’s, then $\text{Root}_k$ are all 1’s, except for $\text{poly}(s) \cdot \epsilon$ probability. Since we set $C_k$ to be a circuit that allows outputs 1, except with an additional $O(\epsilon)$ probability, the wire value of $j_k$ is 1. Finally, applying the “merge” property from Lemma 7.5 to $\text{Root}_k \cup \{j_k\}$ and the no-signaling property, we derive that $\text{Root}(\Delta_n^k)$ must also be 1’s, except for $\text{poly}(s) \cdot \epsilon$ probability. Hence, we finish the proof. □

**Lemma 7.11.** The above construction of the extension circuit $\{E_x\}_{x \in T}$ is $O(s)$-locally computable extension of $\{C_x\}_{x \in T}$. Moreover, the size of $E_x$ is $\text{poly}(t(n), s(n))$ for any $x \in T \cap \{0,1\}^n$.

*Proof.* For each gate $g$ added in $E_x$, the subcircuit of gates that $g$ depends on is included in the configuration that contains $g$ as a propositional variable. Hence, the size of the subcircuit is bounded by the total size of formulas in the configuration, which is at most $O(s)$. The fact that the size of $E_x$ is $\text{poly}(t, s)$ follows directly from the construction. □

**Layered Extension Circuit.** In the following lemma, we prove that any extension circuit can be converted into a layered circuit.

**Lemma 7.12** (Layered Extension Circuit). Let $S \subseteq \{0,1\}^*$ be a set, and $\{C_x\}_{x \in S}$ be a family of circuits. Let $s = s(n), t = t(n), \text{size} = \text{size}(n)$ be functions. If $\{C_x\}_{x \in S}$ has an $\ell$-locally unsatisfiable extension of size $\text{size}$ that is $s$-locally computable, then there exists a circuit family $\{E_x\}_{x \in S}$ such that,

- $E_x$ is a layered circuit for every $x \in S$.
- $\{E_x\}_{x \in S}$ is an $O(\ell)$-locally unsatisfiable extension of $\{C_x\}_{x \in S}$.
- $E_x$ is $\text{poly}(s)$-locally computable.
- The size of $E_x$ is bounded by $\text{poly}(\text{size}(n))$ for every $x \in T \cap \{0,1\}^n$.

*Proof Sketch.* Let $\{E'_x\}_{x \in S}$ be an $\ell$-locally unsatisfiable extension of size $\text{size}$ for $\{C_x\}_{x \in S}$, and $\{E'_x\}_{x \in S}$ is $s$-locally computable. We create a layered circuit family $\{E_x\}_{x \in S}$ from $\{E'_x\}_{x \in S}$, as follows.

We iterate the layers of $E'_x$ one by one, from the bottom layer (input wires) to the top layer (output gates). At each layer, $E_x$ contains all the gates of $E'_x$ in the same layer, and also includes some additional auxiliary gates that save the values from the previous layers. Namely, at the $i$-th layer of $E_x$, we add the auxiliary gates that copy the outputs of gates of $E'_x$ in the $(i-1)$-th layer, and also add the auxiliary gates that copy the outputs of the auxiliary gates in the $(i-1)$-th layer of $E_x$. Finally, for each gate $g'$ in $i$-th layer of $E'_x$, we add a gate $g$ in $E_x$ computing the same function. If the inputs to $g'$ are the gates in the $(i-1)$-th layer of $E'_x$, then we connect the same gates in the $(i-1)$-th layer of $E_x$ to $g$. Otherwise, we use the auxiliary gates in $(i-1)$-th layer to compute $g$.

The size of $E_x$ is bounded by $O((\text{size}(n))^3)$. From the construction, if $E'_x$ is $s$-locally computable, then $E_x$ is $O(s)$-locally computable, because the depth of $E'_x$ is at most $s$. If $\text{LocalGen}$ is an $(\ell, \epsilon)$-local assignment generator for $\{E_x\}_{x \in S}$, then we can build a new $(O(\ell), O(s) \cdot \epsilon)$-local assignment generator $\text{LocalGen}'$ for $\{E'_x\}_{x \in S}$, as follows. To extract a set $S'$ of gates from $\text{LocalGen}'$, we
extract the corresponding set $S$ of gates using LocalGen. To show that local consistency holds with probability $1 - s \cdot \varepsilon$, we use the no-signaling property of LocalGen' to trace through the auxiliary gates. Hence, $\{E_x\}_{x \in S}$ is $O(\ell)$-locally unsatisfiable, if $\{E_x\}_{x \in S}$ is $\ell$-locally unsatisfiable.

8 Applications

In this section we show how to build SNARGs for Diffie-Hellman languages. We choose the Diffie-Hellman language as an illustration of Theorem 1.1.

8.1 SNARGs for the DDH Language

Let $p$ be a prime, and let $G$ be a cyclic subgroup of $\mathbb{Z}_p^*$, where $\mathbb{Z}_p^*$ is the multiplicative group modulo $p$. Let $q$ be the order of the group $G$. The Diffie-Hellman language is defined as follows.

$$L_{DH} = \{(g, h, g', h') \in G^4 | \exists w \in [0, q) : g' = g^w \land h' = h^w\}.$$

We prove the following theorem.

**Theorem 8.1.** Assuming the existence of an FHE scheme, for any $n$-bit prime integer $p$, there exists a non-adaptively sound SNARG for $L_{DH}$ with the following parameters.

- The proof length is $\text{poly}(\lambda, \log n) = \text{poly}(\lambda, \log \log p)$.
- The crs length is $\text{poly}(\lambda, n) = \text{poly}(\lambda, \log p)$.

We prove Theorem 8.1 as follows. For $x = (g, h, g', h') \in G^4$, let $C_x(w)$ be the circuit that takes an $n$-bit string $w$ as input, and output a boolean value of

$$g' = g^w \land h' = h^w.$$

We will prove in Lemma 8.2 that there exists a polynomial-size propositional proof of unsatisfiability for $\{C_x\}_{x \notin L_{DH}}$. Once we have Lemma 8.2, we can apply Theorem 6.2 to obtain a locally unsatisfiable extension of $\{C_x\}_{x \notin L_{DH}}$. Finally, from Theorem 4.8, we obtain a SNARG for $L_{DH}$ with the claimed proof length and crs length.

**Lemma 8.2.** Let $C_x$ be as defined above. There exists a polynomial $\ell = \ell(n)$ such that $\{C_x\}_{x \notin L_{DH}}$ has $\ell$-propositional proof of unsatisfiability.

**Proof.** Instead of directly proving the unsatisfiability of $C_x$ for any $x \notin L_{DH}$ in the propositional logic proof system, we first prove that $C_x(w)$ is unsatisfiable in Cook’s theory $PV$ [Coo75], which is more convenient because $PV$ supports basic arithmetic operations and their related theorems can also be formalized in $PV$. Then we use Cook’s propositional translation [Coo75] to translate this proof in $PV$ to a polynomial-size proof of unsatisfiability in the Extended Frege system (See Section 3.7).

For any fixed $x = (g, h, g', h') \in G^4$, since $G$ is a cyclic group, if $g$ is not the identity element, then there exists an $s$ such that $h = g^s$. For any $x \notin L_{DH}$, we have $h' \neq g'^s$. Note that the value $s$ can be viewed as a fixed constant once we choose a fixed $x = (g, h, g', h') \in G^4$.

In Cook’s theory $PV$, we formalize a function $F$ as

$$F(g, h, g', h', w) = \text{Eq}(g', g^w) \land \text{Eq}(h', h^w)$$

(8)
where $\text{Eq}(x, y)$ is a function symbol that outputs 1 if $x$ and $y$ are the same, and outputs 0 otherwise. Note that we treat $g, h, g', h'$ as variables in $PV$ throughout the proof of unsatisfiability in $PV$. We will replace the variables $g, h, g', h'$ with their concrete fixed values in the final step. Now we can prove in $PV$ that

$$h = g^s \land h' \neq g'^s \vdash_{PV} F(g, h, g', h', w) = 0,$$

with the following derivation. Here we only give a sketch of the derivation.

1. $F(g, h, g', h', w) = \text{Eq}(g', g^w) \land \text{Eq}(h', (g^s)^w)$. (This follows from Equation 8 and $h = g^s$.)
2. $(g^s)^w = (g^w)^s$. (This follows from basic properties about arithmetic [Bus86].)
3. $F(g, h, g', h', w) = \text{Eq}(g', g^w) \land \text{Eq}(h', (g^w)^s)$. (This follows from Line 1 and Line 2.)
4. $\text{Eq}(g', g^w) = 1 \rightarrow g' = g^w$. (This line is an application of the tautology $\text{Eq}(x, y) = 1 \rightarrow x = y$.)
5. $\text{Eq}(g', g^w) = 1 \rightarrow \text{Eq}(h', (g^w)^s) = \text{Eq}(h', g'^s)$. (Replace $g^w$ with $g'$ via Line 4.)
6. $\text{Eq}(h', g'^s) = 0$. (This line can be derived from the premise $h' \neq g'^s$.)
7. $\text{Eq}(g', g^w) = 1 \rightarrow \text{Eq}(h', (g^w)^s) = 0$. (This line is obtained from Line 5 and Line 6.)
8. $F(g, h, g', h', w) = 0$. (This line follows from Line 3 and Line 7.)

By Cook’s propositional translation [Coo75] (Theorem 3.9), there exists a propositional logic proof for

$$h = g^s \land h' \neq g'^s \rightarrow F(g, h, g', h', w) = 0,$$

where the variables $h, g, h', g', s, w$ are decomposed as a series of propositional variables, and each propositional variable represents one bit. Crucially, the translated propositional proof in Extended Frege system has size polynomial in the bit-length of $p$. Our final propositional proof of unsatisfiability of $C_x$ is the concatenation of the following two parts.

- A propositional proof of $h = g^s \land h' \neq g'^s$, where $h, g, h', g'$ are represented as series of propositional variables. Note that since $h, g, h', g'$ are fixed values here, the propositional proof consists of all the wire values of computing $h = g^s \land h' \neq g'^s$. Hence, the size of this part is polynomial in $\log p$.

- A translated proof of $h = g^s \land h' \neq g'^s \rightarrow F(g, h, g', h', w) = 0$ from Cook’s propositional translation, where the propositional variables used in $g, h, g', h', s$ are replaced with their fixed constant values, respectively. Since the translated proof is of polynomial size, the size of this part is also polynomial in $\log p$.

Now we have shown a polynomial-size Extended Frege proof for $C_x(w) \leftrightarrow F$ which finishes the proof of Theorem 8.1. □
9 Acknowledgements

Yael Tauman Kalai and Vinod Vaikuntanathan are supported by DARPA under Agreement No. HR00112020023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA. Part of this research was completed while Zhengzhong Jin was a postdoctoral associate at MIT CSAIL, where he was supported by the above grant. Part of this research was completed while Alex Lombardi was a postdoctoral fellow at the Simons Institute, where he was supported by a Simons-Berkeley postdoctoral fellowship.

References


[JKLM24] Zhengzhong Jin, Yael Kalai, Alex Lombardi, and Surya Mathialagan. Universal snargs for up from proofs of completeness, 2024. unpublished manuscript. 5, 7, 39, 40, 48, 49, 50


