Incrementally Verifiable Computation via Incremental PCPs

Moni Naor * Omer Paneth† Guy N. Rothblum ‡

December 4, 2019

Abstract

If I commission a long computation, how can I check that the result is correct without re-doing the computation myself? This is the question that efficient verifiable computation deals with. In this work, we address the issue of verifying the computation as it unfolds. That is, at any intermediate point in the computation, I would like to see a proof that the current state is correct. Ideally, these proofs should be short, non-interactive, and easy to verify. In addition, the proof at each step should be generated efficiently by updating the previous proof, without recomputing the entire proof from scratch. This notion, known as incrementally verifiable computation, was introduced by Valiant [TCC 08] about a decade ago. Existing solutions follow the approach of recursive proof composition and can be based on strong and non-falsifiable cryptographic assumptions (so-called "knowledge assumptions").

In this work, we present a new framework for constructing incrementally verifiable computation schemes in both the publicly verifiable and designated-verifier settings. Our designated-verifier scheme is based on somewhat homomorphic encryption (which can be based on Learning with Errors) and our publicly verifiable scheme is based on the notion of zero-testable homomorphic encryption, which can be constructed from ideal multi-linear maps [Paneth and Rothblum, TCC 17].

Our framework is anchored around the new notion of a probabilistically checkable proof (PCP) with incremental local updates. An incrementally updatable PCP proves the correctness of an ongoing computation, where after each computation step, the value of every symbol can be updated locally without reading any other symbol. This update results in a new PCP for the correctness of the next step in the computation. Our primary technical contribution is constructing such an incrementally updatable PCP. We show how to combine updatable PCPs with recently suggested (ordinary) verifiable computation to obtain our results.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Isael. Email: moni.naor@weizmann.ac.il. Supported in part by grant from the Israel Science Foundation (no. 950/16). Incumbent of the Judith Kleeman Professorial Chair.

†MIT and Northeastern University. Supported by NSF Grants CNS-1413964, CNS-1350619 and CNS-1414119, and the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226 and W911NF-15-C-0236.

‡Weizmann Institute of Science. Email: rothblum@alum.mit.edu. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702).

Contents

1 Introduction

Efficient verification of complex computations is a foundational question in the theory of computation. Recent years have seen exciting progress in the study of this problem, from a rich theory of efficient protocols to concrete implementations and new application domains. In the verifiable computation paradigm, the output of a computation is accompanied by a proof of the result's correctness. The proof should be efficient to construct (not much more expensive than simply computing the output), and super-efficient to verify (e.g. verification in nearly-linear time).

Incrementally verifiable computation. In this work we revisit the question of *incrementally* verifiable computation, introduced by Valiant [Val08] about a decade ago. To motivate this question, consider the following scenarios:

Intermediate outputs: Consider a server that executes a long computation for a client. Even before the entire computation terminates, the client may want to obtain intermediate outputs or to audit the server's progress throughout the computation. This is especially significant in the presence of transient faults that are hard to detect: suppose that the computation is so long that faults are likely to occur eventually. Without a methodology for detecting these faults, then the final output is likely to be wrong.

Transferable computation: We would like to split a long sequential computation between different parties such that every party performs a small part of the computation and passes it on to the next party. Together with the current state of their computation, parties should include a proof that the computation was performed correctly, not only in the last step, but in its entirety. As a compelling example, consider an extremely long computation that would require all of humanity many generations to complete. We would like every generation to perform its part, and pass the state of the computation along to the next generation together with a proof of correctness.

In both examples above we need a correctness proof that can be constructed *incrementally*, so that at any intermediate point in the computation, the current state can be verified. The process of updating the proof must be *fast* and *stateless*, meaning that, first, the time to update the proof is independent of the running time of the computation so far and, second, to update the proof we only need to know the most recent version of the proof and the current state of the computation.

We restrict our attention to non-interactive protocols for deterministic computations, where both the prover and verifier have access to an honestly generated common reference string, and where soundness is only required to hold against computationally bounded adversarial provers. Even without the issue of incremental updates, both of these relaxations are known to be necessary under standard complexity theoretic assumptions (see Goldreich and Håstad [GH98]).

In a verifiable computation protocol an honest prover executes a program M on input y. For every timestep t, let c_t denote the state of the program (including the program's entire memory) after the first t steps. Given the common reference string (CRS) the prover constructs a proof Π_t for the correctness of the state c_t. For security parameter κ, the verifier takes the CRS, the input y, the state c_t and the proof Π_t and decides if to accept the proof in time $(|y| + |c_t|) \cdot \mathrm{poly}(\kappa)$, independently of t. Soundness asserts that, given an honestly generated CRS, no efficient adversarial prover can find an input y, a time t and an accepting proof for any state other then c_t (except with negligible probability).

A verifiable computation protocol is incrementally updatable if there is an update procedure that, given the CRS, the state c_t and the proof Π_t, computes the proof Π_{t+1} for the next state in time $(|y| + |c_t|) \cdot \mathrm{poly}(\kappa)$.

The state of the art. Valiant presented an approach for constructing incrementally verifiable computation based on the idea of recursive proof composition. Very roughly, given a proof Π_t for state c_t the updated proof Π_{t+1} for the next state c_{t+1} asserts that: (1) there exists a state c_t and a proof Π_t for

timestep t that are accepted by the verifier, and (2) the computation starting from state c_t transitions to state c_{t+1}. Constructing the proof Π_{t+1} given c_t and Π_t may potentially be fast since Π_{t+1} only argues about the fast verification algorithm and one step of the computation.

The challenge in implementing this idea is maintaining soundness. Existing solutions are based on the strong notion of succinct non-interactive arguments of knowledge for non-deterministic computations also known as SNARKs [Val08, BCC$^+$17, BCCT13]. Currently such SNARKs are known based on non-standard non-falsifiable assumptions (so-called "knowledge assumptions"). We therefore ask:

Is incrementally verifiable computation possible under standard assumptions?

1.1 This Work

In this work we give a new framework for constructing incrementally verifiable computation. Based on this framework we give new protocols in both the publicly verifiable and designated-verifier settings.

Designated verifier. In the designated-verifier setting the common reference string (CRS) is generated together with a secret key. Only a verifier that holds this secret key can check the proof, and soundness is *not* guaranteed against parties who know the secret key. In this setting we prove the following:

Theorem 1.1 (informal)**.** *Assuming a somewhat-homomorphic encryption scheme for computations of poly-logarithmic degree, there exists a designated-verifier incrementally verifiable computation protocol.*

The protocol is based on the (non-incremental) verifiable computation protocol of Kalai et al. [KRR14] with the improvements of Brakerski et al. [BHK17]. Their construction can use any computational private information retrieval (PIR) scheme. To get incremental updates, we rely on the stronger notion of somewhat-homomorphic encryption. Such encryption schemes are known under the Learning with Errors assumption (see Brakerski and Vaikuntanathan and Gentry et al. [BV11, GSW13]).

Public verification. In a publicly verifiable protocol, the proof can be verified by anyone who knows the CRS, and there is no secret key. In this setting we prove the following:

Theorem 1.2 (informal)**.** *Assuming a 3-key zero-testable somewhat homomorphic encryption scheme with correctness for adversarially-generated ciphertexts, there exists a publicly verifiable incrementally verifiable computation protocol.*

The protocol is based on the (non-incremental) verifiable computation protocol of Paneth and Rothblum [PR17] and is proven secure under the same assumption as their work. We refer the reader to [PR17] for the definition of the required notion of zero-testable homomorphic encryption. We note, however, that currently, candidates for such homomorphic encryption are only known based on (efficiently falsifiable) assumptions about ideal multilinear maps.

Our framework deviates from the recursive proof composition approach. Instead, our constructions are based on a new type of probabilistically checkable proof (PCP) with *incremental local updates*.

Incrementally updatable PCP. In contrast to the setting of verifiable computation, known constructions in the PCP model have proofs that are longer than the computation whose correctness is being proved. Verification, on the other hand, is performed by querying only a small number of locations in the proof, and in running time that is nearly-linear in the input length. Moreover, in the PCP model positive results are known even for non-deterministic computations with unconditional soundness. PCPs allow us to prove that for a non-deterministic program M and input y there exists a witness w that will make M reach state c_t after t steps. The proof Π_t is a string of size $\mathrm{poly}(t)$ over some alphabet Σ of

size $\mathrm{polylog}(t)$ (our setting requires a non-binary alphabet) and verification queries $\mathrm{polylog}(t)$ symbols of the proof achieving negligible soundness error.

In this setting, the question of incremental updates is as follows: given the proof Π_t for state c_t, and given a state c_{t+1} that follows c_t (for non-deterministic computations, there may be more than one state that follows c_t), we would like to update Π_t and obtain a new proof Π_{t+1} for c_{t+1}. We cannot hope for the update time to be independent of t since, given the error-correcting nature of the proof, every proof symbol must change. Instead we require that every symbol of the proof can "self-update" quickly. That is, given the i-th symbol of Π_t and the states c_t and c_{t+1} we can compute the i-th symbol of the new proof Π_{t+1} in time $(|y| + |c_t|) \cdot \mathrm{polylog}(t)$.

The main technical contribution of this work is a construction of an incrementally updatable PCP. Our construction is based on the classic PCP of Babai, Fortnow, Levin and Szegedy (BFLS) [BFLS91]. We modify their PCP by considering a larger alphabet Σ and augmenting every symbol of the original proof with supplemental values that allow the augmented symbol to self-update.

From PCP to verifiable computation, heuristically. Biehl, Meyer and Wetzel [BMW98] suggested a heuristic transformation from PCPs to verifiable computation protocols. We refer to their technique as the *hidden query heuristic*. Roughly speaking, the idea is to perform the required PCP queries in a manner that does not allow the prover to figure out the query locations. This idea can be implemented by placing random PCP queries in the CRS, encoded using a private information retrieval (PIR) scheme, or, alternatively, encrypted with a homomorphic encryption scheme (where every query is encrypted under a different key). The prover homomorphically evaluates the PCP answers and sends the encrypted results as the proof. The (designated) verifier decrypts the results and checks that the underlying PCP accepts.

We observe that instantiating the hidden query heuristic with a PCP that can be incrementally updated gives a heuristic incrementally verifiable computation protocol. To see this, recall that following the hidden query heuristic, the proof consists of a few PCP symbols encrypted under homomorphic encryption. Since every one of these symbols can self-update, we can homomorphically evaluate the PCP update procedure under the encryption and obtain encryptions of the updated PCP symbols. We note that, while the hidden query heuristic can be implemented with PIR, getting incrementally verifiable computation requires the stronger notion of homomorphic encryption which supports "multi-hop" evaluation. This is because we update the proof by homomorphically evaluating the PCP update procedure over the encrypted PCP answers.

Secure instantiations. For many years it was not known whether the hidden query technique can be shown to be sound (see Dwork et al. [DLN+00] for the obstacles in proving its soundness, as well as [DNR16] and [DHRW16]). However, recent works give secure instantiations of this heuristic in both the designated-verifier and the publicly verifiable settings. Next, we discuss these instantiations and explain how we turn them into incrementally verifiable computation protocols based on our incrementally updatable PCP.

Starting from the designated-verifiable setting, the works of [KRR13, KRR14, BHK17] prove that the hidden query heuristic is secure, assuming the underlying PCP satisfies a strong form of soundness called *no-signaling* soundness. Our designated-verifier protocol is based on the no-signaling PCP construction of Brakerski, Holmgren and Kalai (BHK) [BHK17], which in turn is based on the PCP of BFLS with several changes that facilitate the proof of no-signaling soundness. Very roughly, their construction has the following structure:

1. Given a program M, define an augmented program $\tilde{M}$ that emulates M while encoding each of its states c_t with a particular error correcting code.

2. The honest prover computes the PCP proof for the augmented program $\tilde{M}$. This proof is essentially the same as in the PCP of BFLS.

3. The verifier locally tests the PCP proof. These tests differ significantly from the tests performed by the original BFLS verifier.

To turn this PCP into a verifiable computation protocol, BHK apply the hidden query technique using any PIR scheme.

To achieve incremental updates, we make the following two changes to the BHK protocol: first, we modify the prover to compute the PCP proof for $\tilde{M}$ using our incrementally updatable PCP instead of the PCP of BFLS. Recall that our PCP augments every symbol of the original BFLS proof with supplemental values. Since these supplemental values are only needed to update the proof, the verifier can simply ignore them. Other than that, our verifier is the same as that of BHK. Second, as discussed above, to turn the PCP into an incrementally verifiable computation protocol we use homomorphic encryption instead of PIR. We note that in our PCP the answers can be computed by a polynomial of poly-logarithmic degree and, therefore, somewhat homomorphic encryption is sufficient [Gen09].

We emphasize that while our honest prover is defined differently, the verification procedure of our incrementally verifiable computation is essentially the same as the one in BHK. Therefore, the soundness of our protocol follows directly from the analysis in BHK. Indeed, the focus of this work is on showing that the *honest proof* can be constructed incrementally. We note that there some minor differences between the BFLS construction that we use and the one used in BHK. However, a careful inspection shows that the analysis in BHK can be easily modified to fit our PCP (see Section 2.4 for more detail).

In the publicly verifiable setting, the work of [PR17] gives a verifiable computation protocol based on the hidden query heuristic. While they do not require that the PCP satisfies no-signaling soundness, they need a stronger notion of homomorphic encryption that supports a weak zero-test operation as well as some additional properties. They show that such encryption can be based on ideal multi-linear maps. Similarly to the BHK protocol, in [PR17], the honest prover simply constructs the PCP proof for an augmented program $\tilde{M}$ using the PCP of BFLS. We modify their protocol to use our incrementally updatable PCP instead and use the same verification procedure (ignoring any supplemental values added to the BFLS proof). Therefore, designated-verifiable setting, the soundness of our protocol follows immediately from the proof analysis of [PR17].

On the locality of updates. A natural relaxation of incrementally updatable PCP would allow for updating of every proof symbol given the values of a small number of other symbols. PCPs with such local updates may be easier to construct than PCPs with strictly self-updating symbols. Note, however, that in order to go from incrementally updatable PCPs to incrementally verifiable computation following our framework, it is crucial that PCP symbols can self-update. If computing one symbol of the new proof requires the values of even two old symbols, then the number of symbols we need to maintain under every encryption may grow exponentially with the number of updates.

On strong soundness. The focus of this work is on constructing PCPs and verifiable computation protocols where the *honest* proof can be computed incrementally. An intriguing question for future research is to design PCPs and verifiable computation protocols where even an adversarially generated proof can be updated. That is, if an adversary produces an accepting proof for timestep t, we can continue updating this proof to get accepting proof for subsequent steps. This strong soundness guarantee is motivated, for example, by the transferable computation scenario described above where multiple mutually distrustful parties incrementally construct the correctness proof.

Our PCP construction does not satisfy this stronger guarantee. Very roughly, the reason is that we augment the standard PCP of BFLS by adding supplemental values encoded into every symbol. These supplemental values are crucial for implementing self-updates, but play no role in the verification of

the PCP. In particular, an adversarially generated proof may consist of a good "core PCP" that verifies correctly, together with corrupted supplemental values that would prevent this PCP from updating.

Related work. In a recent work, Holmgren and Rothblum [HR18] construct designated-verifier argument systems where the prover's space and time complexity are very close to the time and space needed to perform the computation. While their work does not consider or achieve the notion of incrementally updatable PCPs, there are technical similarities in the way the two PCP systems are constructed. Indeed, they consider a related notion where the prover is given streaming access to the computation's tableau. In this related model, they can process additions to the tableau in small *amortized* time. On a technical level, we note that they do not limit the space used by the machine, which leads to significant complications. Further connections between incrementally verifiable computation and argument systems with very efficient provers were explored in [Val08, BCCT13].

In a very recent work (subsequent to ours), Kalai, Paneth and Yang [KPY19] construct a verifiable computation protocol with public verification based on a falsifiable assumption on bilinear groups. While their protocol also relies on the hidden query technique, we do not know how to make it incremental based on our PCP. This is because their protocol also uses a bootstrapping technique (to go from a long CRS to a short CRS) that significantly complicates the prover's strategy.

Future directions. We leave open the question of constructing incrementally verifiable computation protocols with strong soundness, where even adversarially generated proofs can be updated as discussed above. Another interesting direction is to explore alternative approaches to incrementally verifiable computation based on standard assumptions. One potential path towards this goal is to implement Valiant's idea of recursive proof composition, replacing knowledge assumptions with the recent bootstrapping technique of [KPY19]. We emphasize that the approach proposed in this work is not based on recursive proof composition. In particular, our solution can also be applied in the designated-verifier setting, based on the Learning with Errors assumption.

2 Technical Overview

Next we describe our construction of an incrementally updatable PCP. We start by recalling the PCP of BFLS. In Section 2.2 we describe our PCP proof string and in Section 2.3 we explain how to update it.

2.1 The BFLS Construction

Our construction builds on the PCP of BFLS [BFLS91]. We recall some of the details of that construction.

Setup. For a non-deterministic polynomial-time Turing machine M and input $y \in \{0,1\}^n$ we construct a proof for the fact that there exists a witness that makes M accepts y. As we know from the Cook-Levin Theorem, it is possible to represent M's computation on an input y by a Boolean 3CNF formula ϕ_y over $N = \mathrm{poly}(n)$ variables such that ϕ_y is satisfiable if and only if there exists a witness that makes M accept y. Let $\mathbb{F}$ be a field of size $\Theta(\log^2 N)$ and let $\mathbb{H} \subset \mathbb{F}$ be a subset of size $\log(N)$. We set $u \in \mathbb{N}$ such that $|\mathbb{H}|^u = N$ and index the variables of ϕ_y by vectors in $\mathbb{H}^u$. Given a witness that makes M accept y we can compute an assignment $X \colon \mathbb{H}^u \to \{0,1\}$ that satisfies ϕ_y.

Arithmetization. The first part of the PCP proof contains the assignment X represented as a multivariate polynomial $\tilde{X} \colon \mathbb{F}^u \to \mathbb{F}$ of degree at most $(|\mathbb{H}|-1)$ in each variable, that identifies with X on $\mathbb{H}^u$. We also describe the formula ϕ_y algebraically as polynomial $\varphi_y \colon \mathbb{F}^\ell \to \mathbb{F}$ over $\ell = 3(u+1)$ variables,

with individual degree $\mathrm{polylog}(N)$. For every 3 variables $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3 \in \mathbb{H}^u$ and 3 bits $b_1, b_2, b_3 \in \{0,1\}$, if the formula ϕ_y contains the clause:

$$(X(\mathbf{h}_1) = b_1) \vee (X(\mathbf{h}_2) = b_2) \vee (X(\mathbf{h}_3) = b_3) \ ,$$

then the polynomial φ_y evaluates to 1 on $(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, b_1, b_2, b_3)$. Otherwise, φ_y evaluates to 0. The polynomial φ_y can be computed by an arithmetic circuit of size $\mathrm{polylog}(N) + O(|y|)$.

The consistency check polynomial. The proof contains a consistency check polynomial $Q^0 \colon \mathbb{F}^\ell \to \mathbb{F}$. For every 3 variables $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3 \in \mathbb{H}^u$ and 3 bits $b_1, b_2, b_3 \in \{0,1\}$ the polynomial Q^0 evaluates to a non-zero value on $(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, b_1, b_2, b_3)$ if only if the formula ϕ_y contains the clause defined by $(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, b_1, b_2, b_3)$ and this clause is not satisfied by the assigned values $X(\mathbf{h}_1), X(\mathbf{h}_3), X(\mathbf{h}_3)$. It follows that Q^0 vanishes on $\mathbb{H}^\ell$ if and only if the assignment X satisfies ϕ_y (which implies that there exists a witness that makes M accept y). The polynomial Q^0 is defined as follows:

$$Q^0\left(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, b_1, b_2, b_3\right) = \varphi_y\left(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, b_1, b_2, b_3\right) \cdot \prod_{i \in [3]} \left(\tilde{X}\left(\mathbf{h}_i\right) - b_i\right) \ .$$

The sum-check polynomials. To allow the verifier to check that Q^0 vanishes on $\mathbb{H}^\ell$ (and, therefore, M accepts y), the proof contains "sum-check polynomials" $Q^1, \ldots, Q^\ell \colon \mathbb{F}^\ell \to \mathbb{F}$. The j-th polynomial in this sequence is a low-degree extension of Q^0 in its first j variables. In particular, for $0 < j \leq \ell$, the polynomial Q^j is defined as:

$$Q^j\left(y_1, \ldots, y_\ell\right) = \sum_{h_1, \ldots, h_j \in \mathbb{H}} \mathsf{ID}_j\left((h_1, \ldots, h_j), (y_1, \ldots, y_j)\right) \cdot Q^0\left(h_1, \ldots, h_j, y_{j+1}, \ldots, y_\ell\right) \ .$$

Where $\mathsf{ID}_j \colon \mathbb{F}^{2j} \to \mathbb{F}$ is the (unique) polynomial with individual degree $(\mathbb{H} - 1)$ such that for every $\mathbf{h}, \mathbf{h}' \in \mathbb{H}^j$, $\mathsf{ID}_j(\mathbf{h}, \mathbf{h}') = 1$ if $\mathbf{h} = \mathbf{h}'$ and $\mathsf{ID}_j(\mathbf{h}, \mathbf{h}') = 0$ otherwise.

The proof string. The PCP proof string contains, for every $\mathbf{u} \in \mathbb{F}^u$ the value $\tilde{X}(\mathbf{u})$ and for every $\mathbf{v} \in \mathbb{F}^\ell$ the values $Q^0(\mathbf{v}), \ldots, Q^\ell(\mathbf{v})$.

On verifying the PCP. For the sake of this technical overview, for the most part we ignore the tests run by the verifier (which include various low-degree tests and consistency checks). This is because our focus is on the structure of the proof itself and the procedure that updates it.

2.2 The Incremental PCP Construction

We start by describing the content of the proof at any intermediate timestep and then explain how to update. Our construction relies on the leveled structure of the formula ϕ_y representing the computation. Specifically, if the computation $M(y)$ requires time T and space S, we can view the variables of ϕ_y as organized in a table with $\mathsf{T}' = \mathsf{T} \cdot \beta$ rows and $\mathsf{S}' = \mathsf{S} \cdot \beta$ columns for some constant β. Any assignment $X \colon [\mathsf{T}'] \times [\mathsf{S}'] \to \{0,1\}$ that satisfies ϕ_y corresponds to an execution of M on input y with some witness as follows: for every timestep $t \in [\mathsf{T}]$ the assignment to the $(t \cdot \beta)$-th row corresponds to the configuration c_t of M after t steps, and rows $(t \cdot \beta) + 1$ through $((t+1) \cdot \beta) - 1$ contain auxiliary variables used to verify the consistency of the configurations c_t and c_{t+1}.[1] A crucial fact that we will use is that ϕ_y is *leveled*. That is, every clause in ϕ_y only involves variables from two consecutive rows.

[1] These auxiliary rows can be avoided if ϕ_y is a k-CNF formula for some constant $k > 3$. However, for our purpose, it is important that ϕ_y is a 3CNF formula.

Partial assignments. We set $m, k \in \mathbb{N}$ such that $|\mathbb{H}|^m = \mathsf{T}'$ and $|\mathbb{H}|^k = \mathsf{S}'$ and we index every variable by a pair in $\mathbb{H}^m \times \mathbb{H}^k$. As before, given a witness that makes M accept y we can compute an assignment $X \colon \mathbb{H}^{m+k} \to \{0,1\}$ that satisfies ϕ_y. For $\tau \in [\mathsf{T}']$ we define the assignment $X_\tau \colon \mathbb{H}^{m+k} \to \{0,1\}$ that agrees with X on the first τ rows and assigns 0 to all variables in rows larger than τ. As before, we consider a polynomial $\tilde{X}_\tau \colon \mathbb{F}^{m+k} \to \mathbb{F}$ of individual degree at most $(|\mathbb{H}|-1)$, that identifies with X_τ on $\mathbb{H}^{m+k}$. As discussed above, every step of M's computation determines an assignment for β consecutive rows. After completing only the first t steps of the computation and reaching configuration c_t, we can already compute the assignment X_τ for $\tau = t \cdot \beta$. Moreover, the assignment to the variables in row τ is only a function of the configuration c_t.

The new formula. Now, to prove that M's computation on input y can indeed reach a configuration c_t after t steps, it is sufficient to prove that both:

1. The assignment X_τ satisfies all of ϕ_y's clauses involving variables of the first $\tau = t \cdot \beta$ rows.
2. The assignment to row τ matches the assignment defined by the configuration c_t.

For a fixed configuration c_t, we therefore define another 3CNF formula ϕ_τ that is satisfied if and only if the assignment of row τ matches c_t.[2] As before, we consider a polynomial $\varphi_\tau \colon \mathbb{F}^\ell \to \mathbb{F}$ describing the clauses of the formula ϕ_τ. We let $\varphi_{y,\tau}$ denote the polynomial $\varphi_y + \varphi_\tau$ describing the clauses of the combined formula $\phi_{y,\tau} = \phi_y \wedge \phi_\tau$.

The consistency check polynomial. Our new consistency check polynomial $Q^0_\tau \colon \mathbb{F}^\ell \to \mathbb{F}$ is defined similarly to Q^0 except that it "ignores" clauses on variables beyond row τ. Recall that every clause in ϕ_y only involves variables from two consecutive rows. We assume WLOG that if $\varphi_{y,\tau}$ contains a clause on the variables $(\mathbf{t}_1, \mathbf{s}_1), (\mathbf{t}_2, \mathbf{s}_2), (\mathbf{t}_3, \mathbf{s}_3) \in \mathbb{H}^{m+k}$ then $\mathbf{t}_2 = \mathbf{t}_3$ is the index of the row immediately before $\mathbf{t}_1$. Therefore, the polynomial Q^0_τ is defined as follows:

$$\begin{aligned} &Q^0_\tau\left(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, b_1, b_2, b_3\right) \\ &\qquad = \mathsf{LE}_m(\mathbf{t}_1, \tau) \cdot \varphi_{y,\tau}\left(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, b_1, b_2, b_3\right) \cdot \prod_{i \in [3]} \left(\tilde{X}_\tau\left(\mathbf{t}_i, \mathbf{s}_i\right) - b_i\right) \ . \end{aligned}$$

Where $\mathsf{LE}_j \colon \mathbb{F}^{2j} \to \mathbb{F}$ is the (unique) polynomial of individual degree $(|\mathbb{H}| - 1)$ such that for every $\mathbf{h}, \mathbf{h}' \in \mathbb{H}^j$, $\mathsf{LE}_j(\mathbf{h}, \mathbf{h}') = 1$ if the row indexed by $\mathbf{h}$ is smaller than or equal to the one indexed by $\mathbf{h}'$, and $\mathsf{LE}_j(\mathbf{h}, \mathbf{h}') = 0$ otherwise. We purposefully order the input variables to Q^0_τ leading with the row indices. As discussed later in this overview, this simplifies the update procedure of the sum-check polynomials. The sum-check polynomials $Q^1_\tau, \ldots, Q^\ell_\tau$ are defined by Q^0_τ as before.

The proof string. In our new proof we group together $O(\ell)$ symbols of the original proof into one symbol (over a larger alphabet). This grouping is crucial for allowing this larger symbol to self-update. The PCP proof for the computation up to timestep $t \in [\mathsf{T}]$ is given by Π_τ for $\tau = t \cdot \beta$. The string Π_τ contains one symbol $\sigma^{\mathbf{z}}_\tau$ for every vector $\mathbf{z} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, b_1, b_2, b_3) \in \mathbb{F}^\ell$. The symbol $\sigma^{\mathbf{z}}_\tau$ contain the values $\tilde{X}_\tau(\mathbf{t}_1, \mathbf{s}_1), \tilde{X}_\tau(\mathbf{t}_2, \mathbf{s}_2), \tilde{X}_\tau(\mathbf{t}_3, \mathbf{s}_3)$ and the values $Q^0_\tau(\mathbf{z}), \ldots, Q^\ell_\tau(\mathbf{z})$. Further, every symbol contains additional supplemental values that are needed for self-updating. The supplemental values are discussed below, when we detail the update procedure.

[2] While ϕ_τ can be described by simple conjunction, in our construction it will convenient to view it as a 3CNF formula.

On verifying the PCP. The new PCP can be verified via the same tests performed by the original BFLS verifier. The grouping of values into symbols and the supplemental values in every symbol are needed only for updates and are ignored by the verifier. Note that in our new construction every value $\tilde{X}_\tau(\mathbf{u})$ is contained in multiple symbols. When the BFLS verifier queries the value $\tilde{X}_\tau(\mathbf{u})$, it is crucial for soundness that the symbol we read in order to answer this query is chosen as a function of $\mathbf{u}$ alone, independently of the other verifier queries.

2.3 Updating the PCP.

We start with the $(t-1)$-th configuration c_{t-1} and one symbol $\sigma^{\mathbf{z}}_{(t-1)\cdot\beta}$ of the proof $\Pi_{(t-1)\cdot\beta}$. Given the next configuration c_t our goal is to compute the symbol $\sigma^{\mathbf{z}}_{t\cdot\beta}$ of the new proof $\Pi_{t\cdot\beta}$. Starting from $\tau = (t-1)\cdot\beta + 1$ we show how to update $\sigma^{\mathbf{z}}_{\tau-1}$ to $\sigma^{\mathbf{z}}_{\tau}$ and we repeat this update β times. Recall that X_τ is our partial assignment to the first τ rows. We first use the new configuration c_t to obtain row τ of X_τ. We denote this assignment by $\gamma_\tau \colon \mathbb{H}^k \to \{0,1\}$. We proceed to update every value in the symbol $\sigma^{\mathbf{z}}_{\tau-1}$. In what follows we denote $\mathbf{z} = (\mathbf{t}, \mathbf{s}, \mathbf{b})$ for $\mathbf{t} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3) \in \mathbb{F}^{3m}$, $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \in \mathbb{F}^{3k}$, and $\mathbf{b} = (b_1, b_2, b_3) \in \mathbb{F}^3$.

Updating $\tilde{X}$. The symbol $\sigma^{\mathbf{z}}_{\tau-1}$ contains the evaluations of the assignment polynomial $\tilde{X}_{\tau-1}$ at locations $(\mathbf{t}_i, \mathbf{s}_i)$. We show how to update these evaluations and compute $\tilde{X}_\tau(\mathbf{t}_i, \mathbf{s}_i)$. Recall that $\tilde{X}_\tau$ is a polynomial of individual degree at most $(|\mathbb{H}| - 1)$, that identifies with X_τ on $\mathbb{H}^{m+k}$. Equivalently, $\tilde{X}_\tau$ is the unique low-degree extension of X_τ given by the sum:

$$\tilde{X}_\tau(\mathbf{v}) = \sum_{\mathbf{h}\in\mathbb{H}^{m+k}} \mathsf{ID}_{m+k}(\mathbf{h}, \mathbf{v}) \cdot X_\tau(\mathbf{h}) \ . \tag{1}$$

Since the assignment $X_{\tau-1}$ and X_τ only differ on the τ-th row where $X_{\tau-1}(\tau, \cdot)$ is identically zero and $X_\tau(\tau, \cdot) = \gamma_\tau$ we have that:

$$X_{\tau+1}(\mathbf{v}) - X_\tau(\mathbf{v}) = \sum_{\mathbf{h}\in\{0,1\}^k} \mathsf{ID}_{m+k}((\tau, \mathbf{h}), \mathbf{v}) \cdot \gamma_\tau(\mathbf{h}) \ .$$

Therefore, given the old value $X_{\tau-1}(\mathbf{t}_i, \mathbf{s}_i)$ and γ_τ we can efficiently compute the new value $X_\tau(\mathbf{t}_i, \mathbf{s}_i)$ by summing the $O(\mathsf{S}')$ summands above.

Updating Q. The symbol $\sigma^{\mathbf{z}}_{t-1}$ also contains the evaluations of the consistency check and sum-check polynomial $Q^j_{\tau-1}(\mathbf{z})$ for every $0 \le j \le \ell$. We show how to update these evaluations and compute $Q^j_\tau(\mathbf{z})$. The update procedure for Q^j_τ is more involved than the update of $\tilde{X}_\tau$, since the polynomial Q^j_τ is not just linear combination of the values $X_\tau(\cdot)$. For different values of j, we give a different procedures updating Q^j_τ. In this overview we demonstrate the main technical ideas by focusing on some of these cases.

Updating Q^0. For $j = 0$ we can efficiently evaluate the consistency check polynomial $Q^0_\tau(\mathbf{z})$ since the values $X_\tau(\mathbf{t}_i, \mathbf{s}_i)$ have already been computed, the circuit LE can be efficiently evaluated, and the circuit $\varphi_{y,\tau}$ can be efficiently evaluated given the input y and the assignment γ_τ.

Updating Q^m. For $j = m$ we want to compute:

$$Q^m_\tau(\mathbf{z}) = \sum_{\mathbf{h}_1\in\mathbb{H}^m} \mathsf{ID}\left(\mathbf{h}_1, \mathbf{t}_1\right) \cdot Q^0_\tau\left(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) \ .$$

In computing this sum, we exploit the fact that the first m inputs to Q^0_τ are always in $\mathbb{H}$. First, for $\mathbf{h}_1 \in \mathbb{H}^m$ we have $\mathsf{LE}(\mathbf{h}_1, \tau) = 1$ when $\mathbf{h}_1 \leq \tau$ and $\mathsf{LE}(\mathbf{h}_1, \tau) = 0$ when $\mathbf{h}_1 > \tau$. (In contrast for an arbitrary $\mathbf{u} \in \mathbb{F}^m$, $\mathsf{LE}(\mathbf{u}, \tau)$ may not be in $\{0, 1\}$.) Therefore, by the definition of Q^0_τ, we can write the sum above as:

$$\sum_{\mathbf{h}_1 \leq \tau} \mathsf{ID}\left(\mathbf{h}_1, \mathbf{t}_1\right) \cdot \varphi_{y,\tau}\left(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) \cdot \left(\tilde{X}_\tau\left(\mathbf{h}_1, \mathbf{s}_1\right) - b_1\right) \cdot \prod_{i \in [2,3]} \left(\tilde{X}_\tau\left(\mathbf{t}_i, \mathbf{s}_i\right) - b_i\right) \ .$$

Since we have already computed the values $\tilde{X}_\tau(\mathbf{t}_2, \mathbf{s}_2)$ and $\tilde{X}_\tau(\mathbf{t}_3, \mathbf{s}_3)$ it is sufficient to compute the following sum denoted by $A^m_\tau(\mathbf{z}, \mathbf{b})$:

$$A^m_\tau(\mathbf{z}) = \sum_{\mathbf{h}_1 \leq \tau} \mathsf{ID}\left(\mathbf{h}_1, \mathbf{t}_1\right) \cdot \varphi_{y,\tau}\left(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) \cdot \left(\tilde{X}_\tau\left(\mathbf{h}_1, \mathbf{s}_1\right) - b_1\right) \ .$$

Computing the τ summands above from scratch requires time proportional to the running time of the computation so far. Therefore, we instead maintain $A^m_\tau(\mathbf{z})$ as a supplemental value contained in the symbol $\sigma^{\mathbf{z}}_\tau$. Thus, it is sufficient to compute $A^m_\tau(\mathbf{z})$ from the old value $A^m_{\tau-1}(\mathbf{z})$ given in the symbol $\sigma^{\mathbf{z}}_{\tau-1}$. Specifically, we show how to efficiently compute the difference $A^m_\tau(\mathbf{z}) - A^m_{\tau-1}(\mathbf{z})$. We observe that most of the summands are equal in $A^m_\tau(\mathbf{z})$ and in $A^m_{\tau-1}(\mathbf{z})$ and, therefore, the difference contains a constant number of summands that we can compute. Specifically we show that for every $\mathbf{h}_1 < \tau - 1$:

$$\varphi_{y,\tau-1}\left(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) = \varphi_{y,\tau}\left(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) \ . \tag{2}$$

$$\tilde{X}_{\tau-1}\left(\mathbf{h}_1, \mathbf{s}_1\right) = \tilde{X}_\tau\left(\mathbf{h}_1, \mathbf{s}_1\right) \ . \tag{3}$$

We first use (2) and (3) to show how to efficiently compute $A^m_\tau(\mathbf{z}) - A^m_{\tau-1}(\mathbf{z})$, and then explain why these equalities hold. Given that (2) and (3) hold for every $\mathbf{h}_1 < \tau - 1$ we can write the difference $A^m_\tau(\mathbf{z}) - A^m_{\tau-1}(\mathbf{z})$ as:

$$\begin{aligned} A^m_\tau(\mathbf{z}) - A^m_{\tau-1}(\mathbf{z}) =& \mathsf{ID}\left(\tau, \mathbf{t}_1\right) \cdot \varphi_{y,\tau}\left(\tau, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) \cdot \left(\tilde{X}_\tau\left(\tau, \mathbf{s}_1\right) - b_1\right) \\ &+ \mathsf{ID}\left(\tau - 1, \mathbf{t}_1\right) \cdot \varphi_{y,\tau}\left(\tau - 1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) \cdot \left(\tilde{X}_\tau\left(\tau - 1, \mathbf{s}_1\right) - b_1\right) \\ &- \mathsf{ID}\left(\tau - 1, \mathbf{t}_1\right) \cdot \varphi_{y,\tau-1}\left(\tau - 1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}\right) \cdot \left(\tilde{X}_{\tau-1}\left(\tau - 1, \mathbf{s}_1\right) - b_1\right) \end{aligned}$$

Recall that the circuits $\varphi_{y,\tau}$ and $\varphi_{y,\tau-1}$ can be efficiently evaluated given the input y and the assignments γ_τ and $\gamma_{\tau-1}$. Therefore, it remains to compute the values:

$$\tilde{X}_\tau\left(\tau, \mathbf{s}_1\right) \quad , \quad \tilde{X}_\tau\left(\tau - 1, \mathbf{s}_1\right) \quad , \quad \tilde{X}_{\tau-1}\left(\tau - 1, \mathbf{s}_1\right) \ .$$

By (1), for any $\mathbf{h} \leq \tau$ the value $\tilde{X}_\tau\left(\mathbf{h}, \mathbf{s}_1\right)$ is just a linear combination of the values assigned to the $\mathbf{h}$-th row:

$$\tilde{X}_\tau\left(\mathbf{h}, \mathbf{s}_1\right) = \sum_{\mathbf{h}' \in \mathbb{H}^k} \mathsf{ID}_k(\mathbf{h}', \mathbf{s}_1) \cdot X_\tau(\mathbf{h}, \mathbf{h}') = \sum_{\mathbf{h}' \in \mathbb{H}^k} \mathsf{ID}_k(\mathbf{h}', \mathbf{s}_1) \cdot \gamma_{\mathbf{h}}(\mathbf{h}') \ . \tag{4}$$

Therefore, we can compute the required evaluations of $\tilde{X}_\tau$ and $\tilde{X}_{\tau-1}$ given the assignments γ_τ and $\gamma_{\tau-1}$. To complete the description of the update procedure for Q^m, we argue that (2) and (3) hold. For (2) we first observe that formulas $\phi_{y,\tau-1} = \phi_y \wedge \phi_{\tau-1}$ and $\phi_{y,\tau} = \phi_y \wedge \phi_\tau$ only differ on clauses over variables in rows $\tau - 1$ and τ. Therefore, if it was the case that $\mathbf{z} \in \mathbb{H}^\ell$ and $\mathbf{h}_1 < \tau - 1$ then (2) would hold. We show how to appropriately modify the definition of the polynomial $\varphi_{y,\tau}$ so that (2) holds for all $\mathbf{z} \in \mathbb{F}^\ell$ as long as $\mathbf{h}_1 \in \mathbb{H}^m$. Recall that the polynomial $\varphi_{y,\tau} = \varphi_y + \varphi_\tau$ describes the formula $\phi_{y,\tau} = \phi_y \wedge \phi_\tau$.

We can assume WLOG that every clause in ϕ_τ on variables $(\mathbf{t}'_1, \mathbf{s}'_1), (\mathbf{t}'_2, \mathbf{s}'_2), (\mathbf{t}'_3, \mathbf{s}'_3) \in \mathbb{H}^{m+k}$ satisfies $\mathbf{t}'_1 = \tau$. Therefore, we can redefine $\varphi_{y,\tau}$ as:

$$\varphi_{y,\tau}(\mathbf{z}) = \varphi_y(\mathbf{z}) + \mathsf{ID}_m(\mathbf{t}_1, \tau) \cdot \varphi_\tau(\mathbf{z}) \ .$$

The new polynomial $\varphi_{y,\tau}(\mathbf{z})$ still represents the same formula $\phi_{y,\tau}$ and (2) holds since for $\mathbf{h}_1 < \tau - 1$ we have:

$$\varphi_{y,\tau-1}(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) = \varphi_y(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) = \varphi_{y,\tau}(\mathbf{h}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) \ .$$

To see why (3) holds, recall that by (4), since $\mathbf{h}_1 \leq \tau - 1$ the value $\tilde{X}_\tau(\mathbf{h}_1, \mathbf{s}_1)$ is just a linear combination of the values assigned to the $\mathbf{h}_1$-th row and therefore:

$$\tilde{X}_{\tau-1}(\mathbf{h}_1, \mathbf{s}_1) = \sum_{\mathbf{h} \in \mathbb{H}^k} \mathsf{ID}_k(\mathbf{h}, \mathbf{s}_1) \cdot \gamma_{\mathbf{h}_1}(\mathbf{h}) = \tilde{X}_\tau(\mathbf{h}_1, \mathbf{s}_1) \ .$$

Updating Q^j for $j < m$. The final case we consider in this overview is $0 < j < m$. Here we want to compute:

$$Q^j_\tau(\mathbf{z}) = \sum_{\mathbf{h} \in \mathbb{H}^j} \mathsf{ID}(\mathbf{h}, \mathbf{t}_1[:j]) \cdot Q^0_\tau(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) \ ,$$

where $\mathbf{t}_1[:j]$ and $\mathbf{t}_1[j+1:]$ denote the j-bit prefix and the $(m-j)$-bit suffix of $\mathbf{t}_1$ respectively. This case is very similar to the case $j = m$ with the added difficulty that now only the first j inputs to Q^0_τ are in $\mathbb{H}$ (as opposed to the previous case, where the entire first index was in $\mathbb{H}$). In the case where $j = m$ we argued that when $\mathbf{u} \in \mathbb{H}^m$, either $\mathbf{u} \leq \tau$ and $\mathsf{LE}(\mathbf{u}, \tau) = 1$, or $\mathbf{u} > \tau$ and $\mathsf{LE}(\mathbf{u}, \tau) = 0$. Now, however, only the first j bits of $\mathbf{u}$ are in $\mathbb{H}$ and the rest may be in $\mathbb{F}$. Thus, it is not immediately clear whether we can say anything about the output of $\mathsf{LE}(\mathbf{u}, \tau)$. We show that in some cases the outcome of $\mathsf{LE}(\mathbf{u}, \tau)$ can be determined given only the prefix of the inputs that is in $\mathbb{H}$. Specifically, using the fact that LE has individual degree $(|\mathbb{H}| - 1)$, we show that for $\mathbf{u} \in \mathbb{H}^j \times \mathbb{F}^{m-j}$ if $\mathbf{u}[:j] > \tau[:j]$ then $\mathsf{LE}(\mathbf{u}, \tau) = 0$. Therefore, we can write $Q^j_\tau(\mathbf{z})$ as:

$$Q^j_\tau(\mathbf{z}) = \sum_{\mathbf{h} \leq \tau[:j]} \mathsf{ID}(\mathbf{h}, \mathbf{t}_1[:j]) \cdot Q^0_\tau(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) \, .$$

As before, since we have already computed the values $X_\tau(\mathbf{t}_2, \mathbf{s}_2)$ and $X_\tau(\mathbf{t}_3, \mathbf{s}_3)$, it is sufficient to show how to maintain the sum $A^j_\tau(\mathbf{z}, \mathbf{b})$ as a supplemental value in $\sigma^{\mathbf{z}}_\tau$:

$$A^j_\tau(\mathbf{z}) = \sum_{\mathbf{h} \leq \tau[:j]} \mathsf{ID}(\mathbf{h}, \mathbf{t}_1[:j]) \cdot \varphi_{y,\tau}(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) \cdot \left(\tilde{X}_\tau(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{s}_1) - b_1 \right) \ .$$

As before, in order to compute the difference $A^j_\tau(\mathbf{z}) - A^j_{\tau-1}(\mathbf{z})$ we first need to show that, analogously to (2) and (3) above, for every $\mathbf{h} < (\tau - 1)[:j]$:

$$\varphi_{y,\tau-1}(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) = \varphi_{y,\tau}(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}, \mathbf{b}) \ . \tag{5}$$

$$\tilde{X}_{\tau-1}(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{s}_1) = \tilde{X}_\tau(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{s}_1) \ . \tag{6}$$

The proof of (5) follows the same argument as (2) except that now we also use the fact that for $\mathbf{h} < \tau[:j]$ it holds that $\mathsf{ID}_m((\mathbf{h}, \mathbf{t}_1[j+1:]), \tau) = 0$ even when $\mathbf{t}_1$ is not in $\mathbb{H}^m$. To see why (6) holds recall that by (1) for $\mathbf{h} < \tau[:j]$ and any $\mathbf{u} \in \mathbb{F}^{m-j}$ the value $\tilde{X}_\tau((\mathbf{h}, \mathbf{u}), \mathbf{s}_1)$ is just a linear combination of the values assigned to rows whose indices (in $\mathbb{H}^m$) start with the prefix $\mathbf{h}$ where $\tilde{X}_\tau$ and $\tilde{X}_{\tau-1}$ identify on these rows:

$$\tilde{X}_\tau((\mathbf{h}, \mathbf{u}), \mathbf{s}_1) = \sum_{(\mathbf{h}', \mathbf{h}'') \in \mathbb{H}^{m-j} \times \mathbb{H}^k} \mathsf{ID}((\mathbf{h}', \mathbf{h}''), (\mathbf{u}, \mathbf{s})) \cdot \gamma_{(\mathbf{h}, \mathbf{h}')}(\mathbf{h}'') = \tilde{X}_{\tau-1}((\mathbf{h}, \mathbf{u}), \mathbf{s}_1) \ .$$

Let $\mathbf{u}$ denote the vector $(\mathbf{t}_1[j+1:], \mathbf{s}_1) \in \mathbb{F}^{m-j+k}$. Similarly to the previous case, it remains to compute the values:

$$\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right) \quad , \quad \tilde{X}_\tau\left((\tau-1)[:j], \mathbf{u}\right) \quad , \quad \tilde{X}_{\tau-1}\left((\tau-1)[:j], \mathbf{u}\right) \ .$$

As explained above, the value $\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right)$ is a linear combination of the values assigned to rows in whose indices (in $\mathbb{H}^m$) start with the prefix $\tau[:j]$. The number of such rows can be proportional to τ, so this values cannot be efficiently computed from scratch. Instead, we update these values using additional supplemental values, which we place in $\sigma_\tau^{\mathbf{z}}$ and maintain:

$$\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right) \quad , \quad \tilde{X}_\tau\left((\tau-1)[:j], \mathbf{u}\right) \ .$$

We explain how to compute $\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right)$ given $\tilde{X}_{\tau-1}\left((\tau-1)[:j], \mathbf{u}\right)$ (updating $\tilde{X}_\tau\left((\tau-1)[:j], \mathbf{u}\right)$ is done similarly). First, recall that the value $\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right)$ is a linear combinations of the values assigned to rows whose indices (in $\mathbb{H}^m$) start with the prefix $\tau[:j]$:

$$\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right) = \sum_{\mathbf{h} \in \mathbb{H}^{m-j+k}} \mathsf{ID}(\mathbf{h}, \mathbf{u}) \cdot X_\tau(\tau[:j], \mathbf{h}) \ .$$

When updating $\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right)$, we distinguish between two cases. First we consider the case where $\tau[:j] = (\tau-1)[:j]$. In this case, both values $\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right)$ and $\tilde{X}_{\tau-1}\left((\tau-1)[:j], \mathbf{u}\right)$ are computed from the values assigned to the same set of rows whose indices start with the prefix $\tau[:j] = (\tau-1)[:j]$. Since the assignment $X_{\tau-1}$ and X_τ only differ on the τ-th row where $X_{\tau-1}(\tau, \cdot)$ is identically zero and $X_\tau(\tau, \cdot) = \gamma_\tau$ we have that:

$$\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right) - \tilde{X}_{\tau-1}\left((\tau-1)[:j], \mathbf{u}\right) = \sum_{\mathbf{h}' \in \mathbb{H}^k} \mathsf{ID}((\tau[j+1:], \mathbf{h}'), \mathbf{u}) \cdot \gamma_\tau(\mathbf{h}') \ .$$

Therefore, in this case we can compute the value $\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right)$ given $\tilde{X}_{\tau-1}\left((\tau-1)[:j], \mathbf{u}\right)$ by summing the $O(\mathsf{S}')$ summands above. Next, we consider the case where $\tau[:j] \neq (\tau-1)[:j]$. In this case, the τ-th row is the only row that starts with the prefix $\tau[:j]$ and assigned a non-zero value by X_τ. Therefore, in this case we can directly compute the value $\tilde{X}_\tau\left(\tau[:j], \mathbf{u}\right)$

Updating Q^j for $j > m$. Updating $Q^j_\tau(\mathbf{z})$ for $j > m$ involves many of the ideas described above. The main difference is that in this case we do not only sum over the first row index. To update the sum we rely on the fact that the polynomial $\varphi_{y,\tau}$ evaluates to 0 whenever the indices $\mathbf{t}_2$ and $\mathbf{t}_3$ are different than $\mathbf{t}_1 - 1$. As in the case where $j < m$, here we also need to deal with the cases where only a prefix of the row indices is in $\mathbb{H}$.

2.4 From PCP to Verifiable Computation.

As discussed in Section 1.1, our designated-verifier incrementally verifiable computation protocol is basically the protocol of BHK [BHK16, Appendix A], where the PCP is replaced by our incrementally updatable PCP. In particular, our verification procedure is essentially identical to that of BHK (ignoring the supplemental values in every symbol that are not part of the original PCP of BFLS). There is, however, a minor differences between our PCP and the PCP in BHK which affects the verification procedure: in our PCP, the sum-check polynomial Q^j is the low-degree extension of Q^0 in its first j variables, while in BHK, Q^j and Q^0 satisfy a different relation. However, the analysis in BHK [BHK16, Appendix B] with only minor changes fits our construction as well.

3 Definitions

In this section we define incrementally updatable PCPs and verifiable computation.

3.1 Incrementally Updatable PCP

We start by recalling the standard notion of a probabilistically checkable proof (PCP) and then define incremental updates. Fix a non-deterministic Turing machine M with running time $\mathsf{T} = \mathsf{T}(n)$. For an input $y \in \{0,1\}^n$ and a witness string $w \in \{0,1\}^t$ where $t \in [\mathsf{T}]$, let $M(y;w)$ denote the configuration of M when executing on input y after t steps using w as a witness. The configuration includes the machine's work tapes, state, and the machine's locations on all tapes. Let $\mathcal{L}_M$ be the language that contains a tuple (y,t,c) if there exists $w \in \{0,1\}^t$ such that $c = M(y;w)$. Let $\mathcal{R}_M$ be the corresponding witness relation. A PCP system for M with alphabet $\Sigma = \{\Sigma_n\}_{n\in\mathbb{N}}$, query complexity $q = q(n)$, and proof length $\ell = \ell(n)$ consists of a deterministic polynomial-time algorithm P and a randomized oracle machine V with the following syntax:

P : given $(x = (y,t,c), w) \in \mathcal{R}_M$ outputs a proof $\Pi \in \Sigma^\ell$.

V : given x and oracle access to the proof Π makes q oracle queries and outputs a bit.

Definition 3.1. *A PCP system (P,V) satisfies the following requirements*

Completeness: *For every $(x,w) \in \mathcal{R}_M$, let $\Pi = \mathsf{P}(x,w)$. It holds that:*

$$\Pr\left[\mathsf{V}^{\Pi}(x) = 1\right] = 1 \ .$$

Soundness: *For every $x \notin \mathcal{L}_M$ and for every $\Pi \in \Sigma^\ell$:*

$$\Pr\left[\mathsf{V}^{\Pi}(x) = 1\right] \leq \frac{1}{2} \ .$$

Incremental updates. In an incrementally updatable PCP, each location in the proof string can be maintained and updated in a step-by-step fashion: given the machine's configuration and the value of the PCP at a certain location z after t steps of the computation, the *updated* value of the PCP at location z after $(t+1)$ steps can be computed *locally*, without looking at any PCP symbols except the symbol in location z. Note that the "updated PCP" proves an "updated claim" about the $(t+1)$-th configuration. Note also that, while this local update does require knowledge of the entire current configuration (whose size is dominated by the machine's *space* complexity), this can be much smaller than the length of the PCP (which is larger than the machine's *time* complexity). Formally, an incrementally updatable PCP comes with a deterministic polynomial-time algorithm Update with the following syntax: given an instance $(y, t-1, c_{t-1})$, a witness bit w_t, a position $z \in [\ell]$, and symbol $\sigma^z_{t-1} \in \Sigma$ outputs a new symbol $\sigma^z_t \in \Sigma$. For every $(x = (y,t,c_t), w) \in \mathcal{R}_M$, the PCP proof $\Pi_t = \mathsf{P}(x,w)$ can be constructed by running Update as follows:

1. Let c_0 be the initial configuration of $M(y)$ and let $\sigma^z_0 = \perp$.
2. For every $\tau \in [t]$, $z \in [\ell]$, update M's configuration from $c_{\tau-1}$ to c_τ using witness bit w_τ and let:
$$\sigma^z_\tau \leftarrow \mathsf{Update}\left((y, \tau-1, c_{\tau-1}), w_\tau, z, \sigma^z_{\tau-1}\right) \ .$$
3. Output the proof $\Pi_t = \left(\sigma^1_t, \ldots, \sigma^\ell_t\right)$.

3.2 Incrementally Verifiable Computation

We start with the definition of verifiable computation and then define incremental updates. Fix a deterministic Turing machine M with running time $\mathsf{T} = \mathsf{T}(n)$. For an input $y \in \{0,1\}^n$ and $t \in [\mathsf{T}]$, let $M(y;1^t)$ be the configuration of M when executing on input y after t steps (a configuration includes

the machine's work tapes, state, and the machine's locations on all tapes). Let $\mathcal{L}_M$ be the language that contains a tuple (y, t, c) if $c = M(y; 1^t)$. A verifiable computation scheme consists of a randomized polynomial-time algorithm G and deterministic polynomial time algorithms P, V with the following syntax:

G**:** given the security parameter 1^κ, outputs a pair of keys: a prover key pk and a verifier key vk.

P**:** given the prover key pk, a time bound 1^t and an instance $x = (y, t, c)$, outputs a proof Π.

V**:** given the verifier key vk, an instance x and a proof Π, outputs a bit.

We say that the proof is publicly verifiable if the algorithm G always outputs identical prover and verifier keys $\mathsf{vk} = \mathsf{pk}$. Otherwise the proof is designated verifier.

Definition 3.2. *A verifiable computation scheme* $(\mathsf{G}, \mathsf{P}, \mathsf{V})$ *for* $\mathcal{L}_M$ *satisfies the following requirements:*

Completeness: *For every* $\kappa \in \mathbb{N}$ *and for every* $x = (y, t, c) \in \mathcal{L}_M$*:*

$$\Pr\left[\mathsf{V}(\mathsf{vk}, x, \Pi) = 1 \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{G}(1^\kappa) \\ \Pi \leftarrow \mathsf{P}(\mathsf{pk}, 1^t, x) \end{array}\right] = 1 \ .$$

Efficiency: *In the above honest experiment the length of the proof* Π *is* $\mathrm{poly}(\kappa, \log(t))$*. The verifier's running time is* $|x| \cdot \mathrm{poly}(\kappa, |\Pi|)$*.*

Soundness: *For every polynomial* $\mathsf{T} = \mathsf{T}(\kappa)$ *and for every polynomial size cheating prover* P^* *there exists a negligible function* μ *such that for every* $\kappa \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{l} x = (y, \mathsf{T}, c) \notin \mathcal{L}_M \\ \mathsf{V}(\mathsf{vk}, x, \Pi) = 1 \end{array} \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{G}(1^\kappa) \\ (x, \Pi) \leftarrow \mathsf{P}^*(\mathsf{pk}) \end{array}\right] \leq \mu(\kappa) \ .$$

Incremental updates. A verifiable computation scheme (with either public or designated verifier) satisfying Definition 3.2 is incrementally verifiable if given the honest proof Π_t for a statement (y, t, c_t) and the configuration c_t we can obtain the next proof Π_{t+1} for the statement $(y, t+1, c_{t+1})$ without repeating the entire computation. Formally, an incrementally verifiable computation scheme also includes a deterministic polynomial-time algorithm Update with the following syntax: given the prover key pk, a statement $(y, t-1, c_{t-1}) \in \mathcal{L}_M$ and a proof Π_{t-1}, Update outputs a new proof Π_t. For every statement $x = (y, t, c) \in \mathcal{L}_M$, the proof $\Pi_t = \mathsf{P}(\mathsf{pk}, 1^t, x)$ can be constructed by running Update as follows:

1. Let c_0 be the initial configuration of $M(y)$ and let $\Pi_0 = \perp$.
2. For every $\tau \in [t]$, update M's configuration from $c_{\tau-1}$ to c_τ and let:
$$\Pi_\tau \leftarrow \mathsf{Update}(\mathsf{pk}, (y, \tau - 1, c_{\tau-1}), \Pi_{\tau-1}) \ .$$
3. Output Π_t.

The completeness, efficiency, and soundness requirements of

4 PCP Construction

In this section we introduce notation and describe the PCP system. The update procedure for this PCP is described in the next section. Before reading the full details, we recommend the reader familiarize themselves with the overview in Section 2.

4.1 Preliminaries

We start by introducing notations and simple clams that are used throughout the following sections.

Operations on strings. For an alphabet Σ, a string $\mathbf{v} = v_1, \ldots, v_n \in \Sigma^n$ and $1 \leq i \leq j \leq n$ we denote by $\mathbf{v}[i{:}j]$ the substring $v_i, \ldots, v_j$. We shorthand $\mathbf{v}[1{:}i]$ by $\mathbf{v}[{:}i]$ and $\mathbf{v}[j{:}n]$ by $\mathbf{v}[j{:}]$. We also define $\mathbf{v}[{:}0]$ and $\mathbf{v}[n+1{:}]$ to be the empty string. For a pair of strings $\mathbf{u}, \mathbf{v} \in \Sigma^n$ and $i \in [0, n]$ let $(\mathbf{u}|\mathbf{v})_i$ denote the string $(\mathbf{u}[{:}i], \mathbf{v}[i+1{:}])$.

The field $\mathbb{F}$. Fix any field $\mathbb{F}$ and a subset $\mathbb{H} \subseteq \mathbb{F}$. (The sizes of $\mathbb{F}$ and $\mathbb{H}$ will be set later in this section.) We also fix a linear order on $\mathbb{H}$ and use the lexicographical order on strings in $\mathbb{H}^m$ for any $m \in \mathbb{N}$. We denote the minimal and maximal element in $\mathbb{H}$ by 0 and $|\mathbb{H}| - 1$ respectively. For $\mathbf{t} \in \mathbb{H}^m$ such that $\mathbf{t} > 0^m$ we denote the predecessor of $\mathbf{t}$ by $\mathbf{t} - 1$.

Arithmetic circuits. We denote by $C : \mathbb{F}^i \to \mathbb{F}^j$ an arithmetic circuit over a field $\mathbb{F}$ with i input wires and j output wires. The circuit is constructed from addition, subtraction and multiplication gates of arity 2 as well as constants from $\mathbb{F}$. The size of C is the number of gates in C. We say that C is of degree d if the polynomial computed by C over $\mathbb{F}$ is of degree d or less in every one of its input variables.

Useful predicates. We make use of arithmetic circuits computing simple predicates.

Claim 4.1. *For every $i \in \mathbb{N}$ there exist arithmetic circuits $\mathsf{ID}_i, \mathsf{LE}_i, \mathsf{PR}_i : \mathbb{F}^{2i} \to \mathbb{F}$ of size $O(i)$ and degree $|\mathbb{H}| - 1$ such that for every input $\mathbf{u}, \mathbf{v} \in \mathbb{H}^i$, the circuits' output is in $\{0, 1\}$ and:*

$$\begin{aligned}
\mathsf{ID}_i(\mathbf{u}, \mathbf{v}) = 1 \quad &\Leftrightarrow \quad \mathbf{u} = \mathbf{v} && \textit{(Identity)} \\
\mathsf{LE}_i(\mathbf{u}, \mathbf{v}) = 1 \quad &\Leftrightarrow \quad \mathbf{u} \leq \mathbf{v} && \textit{(Lesser-or-equal)} \\
\mathsf{PR}_i(\mathbf{u}, \mathbf{v}) = 1 \quad &\Leftrightarrow \quad \mathbf{u} - 1 = \mathbf{v} && \textit{(Predecessor)}
\end{aligned}$$

Proof. For $i = 1$, circuits $\mathsf{ID}_1, \mathsf{LE}_1, \mathsf{PR}_1$ exist by straightforward interpolation. For $i > 1$, $u, v \in \mathbb{F}$ and $\mathbf{u}, \mathbf{v} \in \mathbb{F}^{i-1}$, let $\mathbf{u}' = (u, \mathbf{u})$ and $\mathbf{v}' = (v, \mathbf{v})$. The circuit ID_i is given by:

$$\mathsf{ID}_i(\mathbf{u}', \mathbf{v}') = \mathsf{ID}_1(u, v) \cdot \mathsf{ID}_{i-1}(\mathbf{u}, \mathbf{v}) \ .$$

The circuit LE_i is given by:

$$\mathsf{LE}_i(\mathbf{u}', \mathbf{v}') = [\mathsf{ID}_1(u, v) \cdot \mathsf{LE}_{i-1}(\mathbf{u}, \mathbf{v})] + \mathsf{LE}_1(u, v) - \mathsf{ID}_1(u, v) \ .$$

The circuit PR_i is given by:

$$\begin{aligned}
\mathsf{PR}_i(\mathbf{u}', \mathbf{v}') = \ &\mathsf{ID}_1(u, v) \cdot \mathsf{PR}_{i-1}(\mathbf{u}, \mathbf{v}) \\
&+ \mathsf{PR}_1(u, v) \cdot \mathsf{ID}_{i-1}(\mathbf{u}, 0^{i-1}) \cdot \mathsf{ID}_{i-1}(\mathbf{v}, (|\mathbb{H}| - 1)^{i-1})
\end{aligned}$$

□

We also rely on the following useful property of the circuits $\mathsf{ID}, \mathsf{LE}, \mathsf{PR}$. Intuitively, it says that in some cases, the output of the predicate can be determine from the prefix for the input (even if the rest of the input is not in $\mathbb{H}$).

Claim 4.2. *For every $i \in \mathbb{N}$, $j \in [i]$, $\mathbf{t}_1 = (\mathbf{h}_1, \mathbf{f}_1), \mathbf{t}_2 = (\mathbf{h}_2, \mathbf{f}_2) \in \mathbb{H}^j \times \mathbb{F}^{i-j}$ and $\mathbf{h} \in \mathbb{H}^i$:*

- $\mathbf{h}_1 \neq \mathbf{h}_2 \Rightarrow \mathsf{ID}_i(\mathbf{t}_1, \mathbf{t}_2) = 0$.
- $\mathbf{h}_1 > \mathbf{h}_2 \Rightarrow \mathsf{LE}_i(\mathbf{t}_1, \mathbf{t}_2) = 0$.
- $\mathbf{h}_1 < \mathbf{h}_2 \Rightarrow \mathsf{LE}_i(\mathbf{t}_1, \mathbf{t}_2) = 1$.
- $(\mathbf{h} - 1)[:j] \neq \mathbf{h}_1 \Rightarrow \mathsf{PR}_i(\mathbf{h}, \mathbf{t}_1) = 0$.

The proof of Claim 4.2 follows from the next fact.

Fact 4.3. *Let $\varphi\colon \mathbb{F}^i \to \mathbb{F}$ be an arithmetic circuit of degree $(|\mathbb{H}| - 1)$. For every $j \in [i]$, $\mathbf{t} \in \mathbb{H}^j$ and $b \in \{0,1\}$:*

$$\forall \mathbf{h} \in \mathbb{H}^{i-j}\colon \varphi(\mathbf{t}, \mathbf{h}) = b \quad \Rightarrow \quad \forall \mathbf{f} \in \mathbb{F}^{i-j}\colon \varphi(\mathbf{t}, \mathbf{f}) = b \ .$$

4.2 The Constraints

In this section we give an algebraic representation of the constraints of a computation through the notion of a constraint circuit.

The tableau formula. Let M be a non-deterministic Turing machine with running time $\mathsf{T} = \mathsf{T}(n)$ and space complexity $\mathsf{S} = \mathsf{S}(n)$. By the Cook-Levin theorem the computation of M on some input can be described by $\mathsf{T}' \cdot \mathsf{S}'$ Boolean variables where $\mathsf{T}' = \mathsf{T} \cdot \beta$ and $\mathsf{S}' = \mathsf{S} \cdot \beta$ for some constant $\beta \in \mathbb{N}$. Intuitively, we think the variables as organized in a table with T' rows S' columns. The variables in row $\beta \cdot t$ correspond to the configuration of M after t steps. All other rows, whose indices are not a multiple of β, contain auxiliary variables used to verify the consistency of adjacent configurations. An assignment to the variables describes a valid computation of M (with any witness) if and only if it satisfies a 3CNF "tableau formula" ϕ_y.

Claim 4.4 (Cook-Levin-Karp)**.** *There exists a constant β such that for every input $y \in \{0,1\}^n$ there exists a 3CNF formula ϕ_y over the variables $\{x_{t,s}\}_{t \in [\mathsf{T}'], s \in [\mathsf{S}']}$ where $\mathsf{T}' = \beta \cdot \mathsf{T}$ and $\mathsf{S}' = \beta \cdot \mathsf{S}$ such that the following holds:*

Completeness: *For every witness $w \in \{0,1\}^{\mathsf{T}}$ there exists an assignment $X^{y,w}\colon [\mathsf{T}'] \times [\mathsf{S}'] \to \{0,1\}$ that satisfies ϕ_y. Moreover, for any $t \in [\mathsf{T}]$, given only the configuration $c_t = M(y; w[:t])$ we can compute a row assignment $\gamma^{c_t}\colon [\mathsf{S}'] \to \{0,1\}$ such that $X^{y,w}(t \cdot \beta, \cdot) = \gamma^{c_t}$. Similarly, for any $(t-1) \cdot \beta < \tau \leq t \cdot \beta$, given only the configuration $c_{t-1} = M(y; w[:t-1])$ and the witness bit w_t we can compute a row assignment $\gamma_\tau^{c_{t-1}, w_t}\colon [\mathsf{S}'] \to \{0,1\}$ such that $X^{y,w}(\tau, \cdot) = \gamma_\tau^{c_{t-1}, w_t}$.*

Soundness: *For any assignment $X\colon [\mathsf{T}'] \times [\mathsf{S}'] \to \{0,1\}$ that satisfies ϕ_y there exists a witness w such that $X = X^{y,w}$. Moreover, for every $t \in [\mathsf{T}]$ and configuration c, if $X(t \cdot \beta, \cdot) = \gamma^c$ then $c = M(y; w[:t])$.*

Leveled structure: *Every constraint in ϕ_y is of the form $(x_{t_1,s_1} = b_1) \vee (x_{t_2,s_2} = b_2) \vee (x_{t_3,s_3} = b_3)$ where $t_1 - 1 = t_2 = t_3$.*

The configuration formula. For every $\tau \in [\mathsf{T}']$ and a row assignment $\gamma\colon [\mathsf{S}'] \to \{0,1\}$ we define a a 3CNF "configuration formula" $\phi_{\tau,\gamma}$ over the same variables as the tableau formula ϕ_y checking that the τ-th row assignment is equal to γ. That is:

- If $\tau = t \cdot \beta$ for some $t \in [\mathsf{T}]$ then $\phi_{\tau,\gamma}$ is satisfied by an assignments $X\colon [\mathsf{T}'] \times [\mathsf{S}'] \to \{0,1\}$ if and only if $X(\tau, \cdot) = \gamma$.
- Otherwise, $\phi_{\tau,\gamma} = 1$ is the empty formula.

For technical reasons, we assume WLOG that all the constraints in $\phi_{\tau,\gamma}$ are of the form $(x_{t_1,s_1} = b_1) \vee (x_{t_2,s_2} = b_2) \vee (x_{t_3,s_3} = b_3)$, where $\tau = t_1$. Additionally we assume that $\phi_{\tau,\gamma}$ has the same leveled structure as the tableau formula. That is, $t_1 - 1 = t_2 = t_3$. For τ that is not a multiple of β, $\phi_{\tau,\gamma}$ is the empty formula so our assumptions on the structure of $\phi_{\tau,\gamma}$ hold vacuously.

Arithmetizing the constraint formula. Let $\mathbb{F}$ be a field of size $\Theta(\log^2 \mathsf{T}')$, let $\mathbb{H} \subset \mathbb{F}$ be a subset of size $\lceil \log \mathsf{T}' \rceil$ such that $\{0,1\} \subseteq \mathbb{H}$ and let

$$m = \frac{\log \mathsf{T}'}{\log\log \mathsf{T}'} \quad , k = \frac{\log \mathsf{S}'}{\log\log \mathsf{T}'} \ .$$

We assume WLOG that m, k and $\log \mathsf{T}'$ are all integers and, therefore, $|\mathbb{H}^m| = \mathsf{T}'$ and $|\mathbb{H}^k| = \mathsf{S}'$. We identify elements of $\mathbb{H}^m$ (with lexicographic order) with indices in $[\mathsf{T}]$, and elements in $\mathbb{H}^k$ with indices in $[\mathsf{S}]$. We view an assignment X for the variables $\{x_{t,s}\}$ as a function $X \colon \mathbb{H}^{m+k} \to \{0,1\}$.

Constraint circuits. We can implicitly represent a 3CNF formula ϕ over the variables $\{x_{t,s}\}$ using a multivariate polynomial. Intuitively, this polynomial represents the indicator function indicating whether a given 3-disjunction is in the formula. We represent this polynomial via an arithmetic circuit over $\mathbb{F}$.

Definition 4.5 (Constraint circuit)**.** *An arithmetic circuit $\varphi \colon \mathbb{F}^{3(m+k+1)} \to \mathbb{F}$ is a constraint circuit representing a 3CNF formula ϕ over the variables $\{x_{\mathbf{t},\mathbf{s}}\}_{\mathbf{t}\in\mathbb{H}^m,\mathbf{s}\in\mathbb{H}^k}$ if for every $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3 \in \mathbb{H}^m$, $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3 \in \mathbb{H}^k$ and $b_1, b_2, b_3 \in \{0,1\}$ if ϕ contains the constraint:*

$$(x_{t_1,s_1} = b_1) \vee (x_{t_2,s_2} = b_2) \vee (x_{t_3,s_3} = b_3) \ ,$$

then φ evaluates to 1 on $(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, b_1, b_2, b_3)$. Otherwise φ evaluates to 0.

Next, we claim that the tableau formula and configuration formula can be efficiently represented as constraint circuits. The proof is standard and it is omitted.

Claim 4.6. *For every input $y \in \{0,1\}^n$, $\tau \in [\mathsf{T}']$ and assignment $\gamma \colon \mathbb{H}^k \to \{0,1\}$ let ϕ_y and $\phi_{\tau,\gamma}$ be the tableau formula and the configuration formula defined above.*

- *Given y we can efficiently compute a tableau constraint circuit φ_y of size $O(n) + \mathrm{poly}(m)$ and degree $\mathrm{poly}(m,k)$ describing ϕ_y.*
- *Given τ and γ we can efficiently compute a configuration constraint circuit $\varphi_{\tau,\gamma}$ of size $\mathsf{S}\cdot\mathrm{poly}(m)$ and degree $O(1)$ describing $\phi_{\tau,\gamma}$.*

The constraint circuit $\tilde{\varphi}$. The constraint circuit $\tilde{\varphi}$ used in our PCP construction is a combination of the constraint circuit and predicates above. Let ϕ_y and $\phi_{\tau,\gamma}$ be the tableau and configuration formulas defined above and let φ_y and $\varphi_{\tau,\gamma}$ be the constraint circuits that describe them, given by Claim 4.6. For $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3 \in \mathbb{F}^m$ and $\mathbf{f} \in \mathbb{F}^{3k+3}$, let $\tilde{\varphi}_{y,\tau,\gamma}$ be the circuit give by:

$$\tilde{\varphi}_{y,\tau,\gamma}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{f}) = \mathsf{PR}_m(\mathbf{t}_1, \mathbf{t}_2) \cdot \mathsf{PR}_m(\mathbf{t}_1, \mathbf{t}_3) \cdot (\varphi_y(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{f}) + \mathsf{ID}_m(\tau, \mathbf{t}_1) \cdot \varphi_{\tau,\gamma}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{f})) \ .$$

Claim 4.7. *For every $y \in \{0,1\}^n$, $\tau \in \mathbb{H}^m$, $\gamma \colon \mathbb{H}^k \to \{0,1\}$, $\tilde{\varphi}_{y,\tau,\gamma}$ describes the 3CNF formula $\phi_y \wedge \phi_{\tau,\gamma}$. Moreover, for every $\tau' \in \mathbb{H}^m$, $\gamma' \colon \mathbb{H}^k \to \{0,1\}$, $i < m$, $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3 \in \mathbb{H}^i \times \mathbb{F}^{m-i}$, $\mathbf{h} \in \mathbb{H}^m$ and $\mathbf{f} \in \mathbb{F}^{3k+3}$:*

$$\begin{aligned} \{\mathbf{t}_2[:i], \mathbf{t}_3[:i]\} \neq \{(\mathbf{h}-1)[:i]\} \quad &\Rightarrow \quad \tilde{\varphi}_{y,\tau,\gamma}(\mathbf{h}, \mathbf{t}_2, \mathbf{t}_3, \mathbf{f}) = 0 \ , \\ \mathbf{t}_1[:i] \notin \{\tau[:i], \tau'[:i]\} \quad &\Rightarrow \quad \tilde{\varphi}_{y,\tau,\gamma}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{f}) = \tilde{\varphi}_{y,\tau',\gamma'}(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{f}) \ . \end{aligned}$$

Proof sketch. Since the tableau constraints and the configuration constraints are disjoint, the circuit $\varphi_y + \varphi_{\tau,\gamma}$ describes the 3CNF formula $\phi_y \wedge \phi_{\tau,\gamma}$. By the leveled structure of the formulas ϕ_y and $\phi_{\tau,\gamma}$ the circuit $\varphi_y + \varphi_{\tau,\gamma}$ identifies with $\tilde{\varphi}_{y,\tau,\gamma}$ on $\mathbb{H}^{3(m+k+1)}$. The rest of the claim follows from the leveled structure of the formulas ϕ_y and $\phi_{\tau,\gamma}$ and by Claim 4.2. □

4.3 The Proof String

Recall that for every $t \in [\mathsf{T}]$ and $z \in [\ell]$, $\sigma_t^z \in \Sigma$ denotes the z-th symbol of the proof after t updates. We start by specifying the value of σ_t^z and in the next section we describe the procedure Update maintaining it. Next we introduce some notation and describe the different components of the PCP. See Section 2 for a high level overview of the construction.

The first part of our construction closely follows the PCP of BFLS. Fix $y \in \{0,1\}^n$ and $w \in \{0,1\}^{\mathsf{T}}$ and let $X^{y,w}$ be the assignment given by Claim 4.4. For $\tau \in \mathbb{H}^m$ we define:

- Let $\gamma_\tau \colon \mathbb{H}^k \to \{0,1\}$ be the row assignment $\gamma_\tau = X^{y,w}(\tau, \cdot)$.
- Let $X_\tau \colon \mathbb{H}^{m+k} \to \{0,1\}$ be the assignment such that $X_\tau(\mathbf{t}, \cdot) = \gamma_{\mathbf{t}}$ for all $\mathbf{t} \leq \tau$ and for all $\mathbf{t} > \tau$, $X_\tau(\mathbf{t}, \cdot)$ is identically zero.
- Let $\tilde{X}_\tau \colon \mathbb{F}^{m+k} \to \mathbb{F}$ be the polynomial of degree $|\mathbb{H}| - 1$ that identifies with X_τ on $\mathbb{H}^{m+k}$:

$$\tilde{X}_\tau(\mathbf{f}) = \sum_{\mathbf{h} \in \mathbb{H}^{m+k}} \mathsf{ID}(\mathbf{h}, \mathbf{f}) \cdot X_\tau(\mathbf{h})$$

- Let $Q_\tau^0 \colon \mathbb{F}^{3(m+k+1)} \to \mathbb{F}$ be the following polynomial. For $\mathbf{z} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \in \mathbb{F}^{3(m+k)}$, $\mathbf{b} = (b_1, b_2, b_3) \in \mathbb{F}^3$ and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$:

$$Q_\tau^0(\bar{\mathbf{z}}) = \mathsf{LE}(\mathbf{t}_1, \tau) \cdot \tilde{\varphi}_{y,\tau,\gamma_\tau}(\bar{\mathbf{z}}) \cdot \prod_{i \in [3]} \left(\tilde{X}_\tau(\mathbf{t}_i, \mathbf{s}_i) - b_i \right) .$$

- For $j \in [3(m+k)]$ let $Q_\tau^j \colon \mathbb{F}^{3(m+k+1)} \to \mathbb{F}$ be the polynomial:

$$Q_\tau^j(\mathbf{f}) = \sum_{\mathbf{h} \in \mathbb{H}^j} \mathsf{ID}(\mathbf{h}, \mathbf{f}[:j]) \cdot Q_\tau^0(\mathbf{h}, \mathbf{f}[j+1:]) .$$

Next we introduce additional polynomials that are not a part of the BFLS construction). These polynomials define the supplemental values added to the PCP to support updates.

First we define polynomials $A_\tau^0, B_\tau^0, C_\tau^0$ by multiplying together subsets of the factors of Q_τ^0. Let $A_\tau^0, B_\tau^0, C_\tau^0 \colon \mathbb{F}^{3(m+k+1)} \to \mathbb{F}$ be the following polynomials. For $\mathbf{z} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \in \mathbb{F}^{3(m+k)}$, $\mathbf{b} = (b_1, b_2, b_3) \in \mathbb{F}^3$, and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$:

$$\begin{aligned}
A_\tau^0(\bar{\mathbf{z}}) &= \tilde{\varphi}_{y,\tau,\gamma_\tau}(\bar{\mathbf{z}}) \cdot \left(\tilde{X}_\tau(\mathbf{t}_1, \mathbf{s}_1) - b_1 \right) , \\
B_\tau^0(\bar{\mathbf{z}}) &= \tilde{\varphi}_{y,\tau,\gamma_\tau}(\bar{\mathbf{z}}) \cdot \left(\tilde{X}_\tau(\mathbf{t}_1, \mathbf{s}_1) - b_1 \right) \cdot \left(\tilde{X}_\tau(\mathbf{t}_2, \mathbf{s}_2) - b_2 \right) , \\
C_\tau^0(\bar{\mathbf{z}}) &= \tilde{\varphi}_{y,\tau,\gamma_\tau}(\bar{\mathbf{z}}) \cdot \left(\tilde{X}_\tau(\mathbf{t}_1, \mathbf{s}_1) - b_1 \right) \cdot \left(\tilde{X}_\tau(\mathbf{t}_2, \mathbf{s}_2) - b_2 \right) \cdot \left(\tilde{X}_\tau(\mathbf{t}_3, \mathbf{s}_3) - b_3 \right) .
\end{aligned}$$

Next we define the polynomials $A_\tau^j, B_\tau^j, C_\tau^j$. Similar to the definition of Q_τ^j via Q_τ^0, the evaluations of A_τ^j, B_τ^j and C_τ^j on input $\bar{\mathbf{z}} \in \mathbb{F}^{3(m+k+1)}$ are given by a weighted sum of evaluations of A_τ^0, B_τ^0 and C_τ^0 respectively, over inputs $\bar{\mathbf{z}}'$ whose prefix is in $\mathbb{H}$ and suffix in equal to that of $\bar{\mathbf{z}}$. However, unlike the definition of Q_τ^j, we do not sum over all possible prefixes in $\mathbb{H}$, but only over prefixes with a certain structure. Specifically:

- A_τ^j sums over prefixes $\mathbf{h} \in \mathbb{H}^j$ such that $\mathbf{h} < \tau[:j]$.
- B_τ^j sums over prefixes $(\mathbf{h}, (\mathbf{h}-1)[:j]) \in \mathbb{H}^{m+j}$ such that $0^m < \mathbf{h} < \tau$.
- C_τ^j sums over prefixes $(\mathbf{h}, \mathbf{h}-1, (\mathbf{h}-1)[:j]) \in \mathbb{H}^{2m+j}$ such that $0^m < \mathbf{h} < \tau$.

Formally, for $j \in [m]$ let $A^j_\tau, B^j_\tau, C^j_\tau \colon \mathbb{F}^{3(m+k+1)} \to \mathbb{F}$ be the following polynomials. For $\mathbf{z} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \in \mathbb{F}^{3(m+k)}$, $\mathbf{b} = (b_1, b_2, b_3) \in \mathbb{F}^3$ and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$:

$$A^j_\tau(\bar{\mathbf{z}}) = \sum_{\mathbf{h} < \tau[:j]} \mathsf{ID}\left(\mathbf{h}, \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \ ,$$

$$B^j_\tau(\bar{\mathbf{z}}) = \sum_{0^m < \mathbf{h} < \tau} \mathsf{ID}\left((\mathbf{h}, (\mathbf{h}-1)[:j]), \bar{\mathbf{z}}[:m+j]\right) \cdot B^0_\tau\left(\left((\mathbf{h}, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_{m+j}\right) \ ,$$

$$C^j_\tau(\bar{\mathbf{z}}) = \sum_{0^m < \mathbf{h} < \tau} \mathsf{ID}\left((\mathbf{h}, \mathbf{h}-1, (\mathbf{h}-1)[:j]), \bar{\mathbf{z}}[:2m+j]\right) \cdot C^0_\tau\left(\left((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_{2m+j}\right) \ .$$

Finally, we define polynomials $\bar{A}^j_\tau, \bar{B}^j_\tau, \bar{C}^j_\tau$. These are defined similarly to $A^j_\tau, B^j_\tau, C^j_\tau$ except that we sum over different prefixes:

- $\bar{A}^j_\tau$ sums over prefixes $(\mathbf{h}, (\mathbf{h}-1)[:j]) \in \mathbb{H}^{m+j}$ such that $0^m < \mathbf{h} < \tau$ and $(\mathbf{h}-1)[:j] = \tau[:j]$.
- $\bar{B}^j_\tau$ sums over prefixes $(\mathbf{h}, \mathbf{h}-1, (\mathbf{h}-1)[:j]) \in \mathbb{H}^{2m+j}$ such that $0^m < \mathbf{h} < \tau$ and $(\mathbf{h}-1)[:j] = \tau[:j]$.
- $\bar{C}^j_\tau$ sums over prefixes $(\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1, \mathbf{h}') \in \mathbb{H}^{3m+j}$ such that $0^m < \mathbf{h} < \tau$ and $\mathbf{h}' \in \mathbb{H}^j$.

Formally, for $j \in [m]$ let $\bar{A}^j_\tau, \bar{B}^j_\tau \colon \mathbb{F}^{3(m+k+1)} \to \mathbb{F}$ be the following polynomials. For $\mathbf{z} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \in \mathbb{F}^{3(m+k)}$, $\mathbf{b} = (b_1, b_2, b_3) \in \mathbb{F}^3$ and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$:

$$\bar{A}^j_\tau(\bar{\mathbf{z}}) = \sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j] = \tau[:j]}} \mathsf{ID}\left((\mathbf{h}, \tau[:j]), \bar{\mathbf{z}}[:m+j]\right) \cdot A^0_\tau\left(\left((\mathbf{h}, \tau)\,|\bar{\mathbf{z}}\right)_{m+j}\right) \ ,$$

$$\bar{B}^j_\tau(\bar{\mathbf{z}}) = \sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j] = \tau[:j]}} \mathsf{ID}\left((\mathbf{h}, \mathbf{h}-1, \tau[:j]), \bar{\mathbf{z}}[:2m+j]\right) \cdot B^0_\tau\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_{2m+j}\right) \ ,$$

For $j \in [3k]$ let $\bar{C}^j_\tau \colon \mathbb{F}^{3(m+k+1)} \to \mathbb{F}$ be the following polynomial:

$$\bar{C}^j_\tau(\bar{\mathbf{z}}) = \sum_{\substack{0^m < \mathbf{h} < \tau \\ \mathbf{h}' \in \mathbb{H}^j}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1, \mathbf{h}'\right), \bar{\mathbf{z}}[:3m+j]\right) \cdot C^0_\tau\left(\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1, \mathbf{h}', \bar{\mathbf{z}}[3m+j+1:]\right) \ .$$

We are now ready to define the PCP proof string. We set $\ell = |\mathbb{F}|^{3(m+k)}$ and identify elements of $\mathbb{F}^{3(m+k)}$ with indices in $[\ell]$.

We first define for every $\tau \in \mathbb{H}^m$ and $\mathbf{z} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \in \mathbb{F}^{3(m+k)}$ an auxiliary symbol $\bar{\sigma}^{\mathbf{z}}_\tau$. These auxiliary symbols will be useful in defining the update procedure. Then, for every $t \in [\mathsf{T}]$, we set the proof symbol $\sigma^{\mathbf{z}}_t$ to be the symbol $\bar{\sigma}^{\mathbf{z}}_{t \cdot \beta}$. The auxiliary symbol $\bar{\sigma}^{\mathbf{z}}_\tau$ contains the values:

1. $\tilde{X}_\tau((\tau|\mathbf{t}_i)_j, \mathbf{s}_i)$ for every $i \in [3]$ and $j \in [0, m]$.
2. $\tilde{X}_\tau((\tau - 1|\mathbf{t}_i)_j, \mathbf{s}_i)$ for every $i \in [3]$ and $j \in [0, m]$. (Only if $\tau > 0^m$.)
3. $Q^j_\tau(\mathbf{z}, \mathbf{b})$ for every $j \in [0, 3(m+k)]$ and $\mathbf{b} \in \{0,1\}^3$.
4. $A^j_\tau(\mathbf{z}, \mathbf{b}), \bar{A}^j_\tau(\mathbf{z}, \mathbf{b}), B^j_\tau(\mathbf{z}, \mathbf{b}), \bar{B}^j_\tau(\mathbf{z}, \mathbf{b}), C^j_\tau(\mathbf{z}, \mathbf{b})$ for every $j \in [m]$ and $\mathbf{b} \in \{0,1\}^3$.
5. $\bar{C}^j_\tau(\mathbf{z}, \mathbf{b})$ for every $j \in [3k]$ and $\mathbf{b} \in \{0,1\}^3$.

The following theorem (that follows from the proof of [BFLS91]) states that the construction above is indeed a PCP proof. In the following sections we prove that this PCP is incrementally updatable.

Theorem 4.8 (Follows from [BFLS91])**.** *There exists a PCP system* (P, V) *for* M *with alphabet* $\Sigma = \mathbb{F}^{O(m+k)}$*, query complexity* $q = \mathrm{poly}(m+k)$*, and proof length* $\ell = |\mathbb{F}|^{3(m+k)} = \mathrm{poly}(\mathsf{T} \cdot \mathsf{S})$ *such that given* $((y, t, c), w) \in \mathcal{R}_M$*,* P *outputs the proof* $\Pi = \left(\sigma^1_t, \ldots, \sigma^\ell_t\right)$*.*

5 Useful Claims About $\tilde{X}$

In this section we prove some useful claims about the polynomial $\tilde{X}_\tau$ defined in Section 4. We start by introducing some notation. Let $\tilde{X}'_\tau \colon \mathbb{H}^m \times \mathbb{F}^k \to \mathbb{F}$ be the function computing the low-degree extension of every individual row of $\tilde{X}_\tau$:

$$\tilde{X}'_\tau(\mathbf{t},\mathbf{s}) = \sum_{\mathbf{h}\in\mathbb{H}^k} \mathsf{ID}_k(\mathbf{h},\mathbf{s}) \cdot X_\tau(\mathbf{t},\mathbf{h}) \ .$$

By the definition of X_τ:

$$\text{For } \mathbf{t} \le \tau\colon \quad \tilde{X}'_\tau(\mathbf{t},\mathbf{s}) = \sum_{\mathbf{h}\in\mathbb{H}^k} \mathsf{ID}_k(\mathbf{h},\mathbf{s}) \cdot \gamma_{\mathbf{t}}(\mathbf{h}) \tag{7}$$

$$\text{For } \mathbf{t} > \tau\colon \quad \tilde{X}'_\tau(\mathbf{t},\mathbf{s}) = 0 \tag{8}$$

Now, since $\tilde{X}_\tau$ is defined as the low-degree extension of X_τ of we can write $\tilde{X}_\tau$ as follows:

$$\begin{aligned}\tilde{X}_\tau(\mathbf{t},\mathbf{s}) &= \sum_{(\mathbf{h}_1,\mathbf{h}_2)\in\mathbb{H}^{m+k}} \mathsf{ID}_{m+k}((\mathbf{h}_1,\mathbf{h}_2),(\mathbf{t},\mathbf{s})) \cdot X_\tau(\mathbf{h}_1,\mathbf{h}_2) \\ &= \sum_{\mathbf{h}_1\in\mathbb{H}^m} \mathsf{ID}_m(\mathbf{h}_1,\mathbf{t}) \cdot \tilde{X}'_\tau(\mathbf{h}_1,\mathbf{s}) \ . \end{aligned} \tag{9}$$

Claim 5.1. *For every $\mathbf{s} \in \mathbb{F}^k$, $\tau \in \mathbb{H}^m$ and $\tau' \le \tau$ we can efficiently compute $\tilde{X}_\tau(\tau',\mathbf{s})$ given $\gamma_{\tau'}$.*

Proof. By (9):

$$\tilde{X}_\tau(\tau',\mathbf{s}) = \sum_{\mathbf{h}\in\mathbb{H}^m} \mathsf{ID}_m(\mathbf{h},\tau') \cdot \tilde{X}'_\tau(\mathbf{h},\mathbf{s}) = \tilde{X}'_\tau(\tau',\mathbf{s}) \ .$$

By (7), $\tilde{X}'_\tau(\tau',\mathbf{s})$ can be computed efficiently from $\gamma_{\tau'}$. The claim follows. □

Claim 5.2. *For every $j \in [0,m]$, $\mathbf{h} \in \mathbb{H}^j$, $\mathbf{t} \in \mathbb{F}^{m-j}$, $\mathbf{s} \in \mathbb{F}^k$ and $\tau > 0^m$, if $\mathbf{h} < \tau[:j]$ then $\tilde{X}_\tau(\mathbf{h},\mathbf{t},\mathbf{s}) = \tilde{X}_{\tau-1}(\mathbf{h},\mathbf{t},\mathbf{s})$.*

Proof. For $\mathbf{h} < \tau[:j]$, and for all $\mathbf{h}' \in \mathbb{H}^{m-j}$ it follows from (7) that $\tilde{X}'_\tau(\mathbf{h},\mathbf{h}',\mathbf{s}) = \tilde{X}'_{\tau-1}(\mathbf{h},\mathbf{h}',\mathbf{s})$. Therefore, by Claim 4.2:

$$\begin{aligned}\tilde{X}_\tau(\mathbf{h},\mathbf{t},\mathbf{s}) &= \sum_{\mathbf{h}'\in\mathbb{H}^m} \mathsf{ID}_m\left(\mathbf{h}',(\mathbf{h},\mathbf{t})\right) \cdot \tilde{X}'_\tau\left(\mathbf{h}',\mathbf{s}\right) \\ &= \sum_{\mathbf{h}'\in\mathbb{H}^{m-j}} \mathsf{ID}_{m-j}\left(\mathbf{h}',\mathbf{t}\right) \cdot \tilde{X}'_\tau\left(\mathbf{h},\mathbf{h}',\mathbf{s}\right) \\ &= \sum_{\mathbf{h}'\in\mathbb{H}^{m-j}} \mathsf{ID}_{m-j}\left(\mathbf{h}',\mathbf{t}\right) \cdot \tilde{X}'_{\tau-1}\left(\mathbf{h},\mathbf{h}',\mathbf{s}\right) \\ &= \sum_{\mathbf{h}'\in\mathbb{H}^m} \mathsf{ID}_m\left(\mathbf{h}',(\mathbf{h},\mathbf{t})\right) \cdot \tilde{X}'_{\tau-1}\left(\mathbf{h}',\mathbf{s}\right) = \tilde{X}_{\tau-1}\left(\mathbf{h},\mathbf{t},\mathbf{s}\right) \ . \end{aligned}$$

□

Claim 5.3. *For every $(\mathbf{t},\mathbf{s}) \in \mathbb{F}^{m+k}$ and $\tau = 0^m$ we can efficiently compute $\tilde{X}_\tau(\mathbf{t},\mathbf{s})$ given γ_τ.*

Proof. By (8) and (9):

$$\begin{aligned}\tilde{X}_\tau(\mathbf{t},\mathbf{s}) &= \sum_{\mathbf{h}\in\mathbb{H}^m} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_\tau(\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}(\tau,\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) + \sum_{\mathbf{h}>\tau} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_\tau(\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}(\tau,\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) \ .\end{aligned}$$

By (7), $\tilde{X}'_\tau(\tau,\mathbf{s})$ can be computed efficiently from γ_τ. The claim follows. □

Claim 5.4. *For every $(\mathbf{t},\mathbf{s})\in\mathbb{F}^{m+k}$ and $\tau>0^m$ we can efficiently compute $\tilde{X}_\tau(\mathbf{t},\mathbf{s})$ given $\tilde{X}_{\tau-1}(\mathbf{t},\mathbf{s})$ and γ_τ.*

Proof. By (7), (8) and (9):

$$\begin{aligned}\tilde{X}_\tau(\mathbf{t},\mathbf{s}) &= \sum_{\mathbf{h}\in\mathbb{H}^m} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_\tau(\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}(\tau,\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) + \sum_{\mathbf{h}<\tau} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_\tau(\mathbf{h},\mathbf{s}) + \sum_{\mathbf{h}>\tau} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_\tau(\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}(\tau,\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) + \sum_{\mathbf{h}<\tau} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_\tau(\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}(\tau,\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) + \sum_{\mathbf{h}<\tau} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_{\tau-1}(\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}(\tau,\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) + \sum_{\mathbf{h}\in\mathbb{H}^m} \mathsf{ID}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_{\tau-1}(\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}(\tau,\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) + \tilde{X}_{\tau-1}(\mathbf{t},\mathbf{s}) \ .\end{aligned}$$

By (7), $\tilde{X}'_\tau(\tau,\mathbf{s})$ can be computed efficiently from γ_τ. The claim follows. □

Claim 5.5. *For every $j\in[0,m]$, $\mathbf{t}\in\mathbb{F}^{m-j}$, $\mathbf{s}\in\mathbb{F}^k$ and $\tau>0^m$ we can efficiently compute $\tilde{X}_\tau(\tau[:j],\mathbf{t},\mathbf{s})$ given $\tilde{X}_{\tau-1}((\tau-1)[:j],\mathbf{t},\mathbf{s})$ and γ_τ.*

Proof. If $\tau[:j]=(\tau-1)[:j]$ the claim follows from Claim 5.4. Otherwise, $\tau=(\tau[:j],0^{m-j})$. Therefore, by Claim 4.2 and by (8):

$$\begin{aligned}\tilde{X}_\tau(\tau[:j],\mathbf{t},\mathbf{s}) &= \sum_{\mathbf{h}\in\mathbb{H}^m} \mathsf{ID}_m(\mathbf{h},(\tau[:j],\mathbf{t}))\cdot\tilde{X}'_\tau(\mathbf{h},\mathbf{s}) \\ &= \sum_{\mathbf{h}\in\mathbb{H}^{m-j}} \mathsf{ID}_{m-j}(\mathbf{h},\mathbf{t})\cdot\tilde{X}'_\tau(\tau[:j],\mathbf{h},\mathbf{s}) \\ &= \mathsf{ID}_{m-j}(0^{m-j},\mathbf{t})\cdot\tilde{X}'_\tau(\tau,\mathbf{s}) \ .\end{aligned}$$

By (7), $\tilde{X}'_\tau(\tau,\mathbf{s})$ can be computed efficiently from γ_τ and the claim follows. □

6 The Update Procedure

In this section we show how to update the the PCP described in the Section 4.

Using the last two configuration. The procedure Update is given as input an instance $(y, t-1, c_{t-1})$, witness bit w_t, position $\mathbf{z}$ and symbol $\sigma^{\mathbf{z}}_{t-1}$ (if $t = 1$ then $\sigma^{\mathbf{z}}_{t-1} = \perp$). In this section we assume that Update is also given the configuration c_{t-2} that precedes c_{t-1} and the witness bit w_{t-1} (if $t = 1$ then $c_{t-2} = \perp$ and $w_{t-1} = \perp$). We explain why this is without loss of generality.

Given a procedure Update that needs the last two configurations c_{t-2} and c_{t-1}, we explain how to modify the PCP and construct a procedure Update$'$ that only needs the last configuration c_{t-1} (as defined in Section 3). The high level idea is that given a configuration c_t we can compute the two configuration that can follow it, and add to each proof symbol also the symbols of the proofs of the two future statements.

Formally, in the new PCP for the instance (y, t, c_t) the symbol σ'_t in position $\mathbf{z}$ contains the symbol σ_t of the original proof, as well as two symbols $\sigma_{t+1,0}$ and $\sigma_{t+1,1}$ for the proof of the next possible statements.[3] That is, for $b \in \{0,1\}$, the symbol $\sigma_{t+1,b}$ is the symbol in position $\mathbf{z}$ of the original proof for the instance $(y, t+1, c_{t+1,b})$ where $c_{t+1,b}$ is the configuration that follows c_t after reading the witness bit b.

The procedure Update$'$ is given as input the instance $(y, t-1, c_{t-1})$, witness bit w_t, position $\mathbf{z}$ and symbol $\sigma'_{t-1} = (\sigma_{t-1}, \sigma_{t,0}, \sigma_{t,1})$ and it outputs the next symbol $\sigma'_t = (\sigma_t, \sigma_{t+1,0}, \sigma_{t+1,1})$. The procedure sets $\sigma_t = \sigma_{t,w_t}$. To obtain the symbol $\sigma_{t+1,b}$ for $b \in \{0,1\}$, the procedure first computes the configuration c_t that follows c_{t-1} after reading the witness bit w_t. Then it executes the procedure Update with the instance (y, t, c_t), witness bit b, position $\mathbf{z}$ and the symbol σ_t as well as the configuration c_{t-1} that precedes c_t and the witness bit w_t.

Notation. We proceed to the describe the input and output of the procedure Update. The procedure is given an instance $(y, t-1, c_{t-1})$, witness bit w_t, position $\mathbf{z} = (\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, \mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) \in \mathbb{F}^{3(m+k)}$ and symbol $\sigma^{\mathbf{z}}_{t-1} = \bar{\sigma}^{\mathbf{z}}_{(t-1)\cdot\beta}$ (if $t = 1$ then $\sigma^{\mathbf{z}}_{t-1} = \perp$). As explained above, we assume that the procedure is also given the configuration c_{t-2} that precedes c_{t-1} and the witness bit w_{t-1} (if $t = 1$ then $c_{t-2} = \perp$ and $w_{t-1} = \perp$).

The procedure compute the output symbol $\sigma^{\mathbf{z}}_t = \bar{\sigma}^{\mathbf{z}}_{t\cdot\beta}$ as follows. For every $(t-1)\cdot\beta < \tau \leq t\cdot\beta$ we use the symbol $\bar{\sigma}^{\mathbf{z}}_{\tau-1}$ the compute the next symbol $\bar{\sigma}^{\mathbf{z}}_{\tau}$. For $\tau > 0^m$ the symbol $\bar{\sigma}^{\mathbf{z}}_{\tau-1}$ contains the values:

$$\left\{\tilde{X}_{\tau-1}((\tau-1|\mathbf{t}_i)_j, \mathbf{s}_i)\right\}_{i\in[3],\, j\in[0,m]} ,$$
$$\left\{\tilde{X}_{\tau-1}((\tau-2|\mathbf{t}_i)_j, \mathbf{s}_i)\right\}_{i\in[3],\, j\in[0,m]} \quad \text{(only if } \tau-1 > 0^m\text{)} ,$$
$$\left\{Q^j_{\tau-1}(\mathbf{z}, \mathbf{b})\right\}_{j\in[0,3(m+k)],\, \mathbf{b}\in\{0,1\}^3} ,$$
$$\left\{A^j_{\tau-1}(\mathbf{z}, \mathbf{b}), \bar{A}^j_{\tau-1}(\mathbf{z}, \mathbf{b}), B^j_{\tau-1}(\mathbf{z}, \mathbf{b}), \bar{B}^j_{\tau-1}(\mathbf{z}, \mathbf{b}), C^j_{\tau-1}(\mathbf{z}, \mathbf{b})\right\}_{j\in[m],\, \mathbf{b}\in\{0,1\}^3} ,$$
$$\left\{\bar{C}^j_{\tau-1}(\mathbf{z}, \mathbf{b})\right\}_{j\in[3k],\, \mathbf{b}\in\{0,1\}^3} .$$

In the rest of this section we show how to compute the symbol $\bar{\sigma}^{\mathbf{z}}_{\tau}$. Our first step is computing the row assignments γ_τ, $\gamma_{\tau-1}$ (if $\tau > 0^m$) and $\gamma_{\tau-2}$ (if $\tau - 1 > 0^m$) . Following Claim 4.4 we set $\gamma_\tau = \gamma_\tau^{c_{t-1}, w_t}$. If $\tau - 1 = (t-1)\cdot\beta$ we set $\gamma_{\tau-1} = \gamma^{c_{t-1}}$, otherwise we set $\gamma_{\tau-1} = \gamma_{\tau-1}^{c_{t-1}, w_t}$. If $\tau - 2 = (t-1)\cdot\beta - 1$ we set $\gamma_{\tau-2} = \gamma_{\tau-2}^{c_{t-2}, w_{t-1}}$. If $\tau - 2 = (t-1)\cdot\beta$ we set $\gamma_{\tau-2} = \gamma^{c_{t-1}}$. Otherwise we set $\gamma_{\tau-2} = \gamma_{\tau-2}^{c_{t-1}, w_t}$.

[3] For simplicity of notation we omit the script $\mathbf{z}$ from the symbols.

6.1 Updating $\tilde{X}$

For $i \in [3]$ and $j \in [0, m]$, we compute the values:

$$\tilde{X}_\tau\left((\tau - 1|\mathbf{t}_i)_j, \mathbf{s}_i\right) \quad , \quad \tilde{X}_\tau\left((\tau|\mathbf{t}_i)_j, \mathbf{s}_i\right) \quad . \tag{10}$$

If $\tau = 0^m$ the values in (10) can be efficiently computed by Claim 5.3. If $\tau > 0^m$, by Claim 5.4, given the values of $\tilde{X}_{\tau-1}$ contained in the previous symbol $\bar{\sigma}_{\tau-1}$ we can efficiently compute the value $\tilde{X}_\tau\left((\tau - 1|\mathbf{t}_i)_j, \mathbf{s}_i\right)$, and by Claim 5.5 we can efficiently compute the value $\tilde{X}_\tau\left((\tau|\mathbf{t}_i)_j, \mathbf{s}_i\right)$. By Claim 5.4 we can also efficiently compute the value:

$$\tilde{X}_\tau\left((\tau - 2|\mathbf{t}_i)_j, \mathbf{s}_i\right) \quad . \tag{11}$$

This value is not a part of the symbol $\bar{\sigma}_\tau$, but we use it in the update procedure.

6.2 Updating A

For $j \in [m]$ and $\mathbf{b} = (b_1, b_2, b_3) \in \{0,1\}^3$ let $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$. We compute the value:

$$A^j_\tau(\bar{\mathbf{z}}) = \sum_{\mathbf{h} < \tau[:j]} \mathsf{ID}\left(\mathbf{h}, \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \quad . \tag{12}$$

For $\tau = 0^m$ the sum is empty. For $\tau > 0^m$ we compute this sum in parts:

1. $\mathbf{h} < (\tau - 1)[:j]$.
2. $\mathbf{h} = (\tau - 1)[:j]$ and $(\tau - 1)[:j] < \tau[:j]$.

Part 1. We compute the part of the sum in (12) where $\mathbf{h} < (\tau - 1)[:j]$:

$$\sum_{\mathbf{h} < (\tau-1)[:j]} \mathsf{ID}\left(\mathbf{h}, \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \quad .$$

By Claim 4.7, for $\mathbf{h} < (\tau - 1)[:j]$:

$$\tilde{\varphi}_{y,\tau-1,\gamma_{\tau-1}}\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) = \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \quad .$$

By Claim 5.2, for $\mathbf{h} < (\tau - 1)[:j]$:

$$\tilde{X}_\tau\left(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{s}_1\right) = \tilde{X}_{\tau-1}\left(\mathbf{h}, \mathbf{t}_1[j+1:], \mathbf{s}_1\right) \quad .$$

Therefore, by the definition of A^0_τ for $\mathbf{h} < (\tau - 1)[:j]$:

$$A^0_\tau\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) = A^0_{\tau-1}\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \quad . \tag{13}$$

Therefore we can write the sum in this part as:

$$\begin{aligned}\sum_{\mathbf{h} < (\tau-1)[:j]} \mathsf{ID}\left(\mathbf{h}, \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) &= \sum_{\mathbf{h} < (\tau-1)[:j]} \mathsf{ID}\left(\mathbf{h}, \bar{\mathbf{z}}[:j]\right) \cdot A^0_{\tau-1}\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \\ &= A^j_{\tau-1}(\bar{\mathbf{z}}) \quad .\end{aligned}$$

The value $A^j_{\tau-1}\left(\bar{\mathbf{z}}\right)$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. Therefore, we can efficiently compute the sum in this part.

Part 2. We compute the part of the sum in (12) where $\mathbf{h} = (\tau-1)[:j]$:

$$\mathsf{ID}\left((\tau-1)[:j], \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left((\tau-1|\bar{\mathbf{z}})_j\right) \ .$$

By the definition of A^0_τ:

$$A^0_\tau\left((\tau-1|\bar{\mathbf{z}})_j\right) = \tilde{\varphi}_{y,\tau,\gamma_\tau}\left((\tau-1|\bar{\mathbf{z}})_j\right) \cdot \left(\tilde{X}_\tau\left((\tau-1|\mathbf{t}_1)_j, \mathbf{s}_1\right) - b_1\right) \ .$$

The value $\tilde{X}_\tau\left((\tau-1|\mathbf{t}_1)_j, \mathbf{s}_1\right)$ is computed in (10). The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

6.3 Updating $\bar{A}$

For $j' \in [m]$ and $\mathbf{b} = (b_1, b_2, b_3) \in \{0,1\}^3$ let $j = m + j'$ and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$. We compute the value:

$$\bar{A}^{j'}_\tau(\bar{\mathbf{z}}) = \sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\left((\mathbf{h}, \tau)|\bar{\mathbf{z}}\right)_j\right) \ . \tag{14}$$

For $\tau = 0^m$ the sum is empty. For $\tau > 0^m$ we compute this sum in parts:

1. $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$.
2. $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$.

Part 1. We compute the part of the sum in (14) where $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$:

$$\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\left((\mathbf{h}, \tau)|\bar{\mathbf{z}}\right)_j\right) \ .$$

First note that if $(\tau-1)[:j'] \neq \tau[:j']$ the sum is empty. Next we compute the sum in the case that $(\tau-1)[:j'] = \tau[:j']$. It follows from (13) (with $j' = m$) that for $\mathbf{h} < \tau - 1$:

$$A^0_\tau\left(\left((\mathbf{h}, \tau)|\bar{\mathbf{z}}\right)_j\right) = A^0_{\tau-1}\left(\left((\mathbf{h}, \tau)|\bar{\mathbf{z}}\right)_j\right) \ .$$

Therefore, we can write the sum in this part as:

$$\begin{aligned} &\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\left((\mathbf{h}, \tau)|\bar{\mathbf{z}}\right)_j\right) \\ = &\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] = (\tau-1)[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, (\tau-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot A^0_{\tau-1}\left(\left((\mathbf{h}, \tau-1)|\bar{\mathbf{z}}\right)_j\right) = \bar{A}^{j'}_{\tau-1}(\bar{\mathbf{z}}) \ . \end{aligned}$$

The value $\bar{A}^{j'}_{\tau-1}(\bar{\mathbf{z}})$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. Therefore, we can efficiently compute the sum in this part.

Part 2. We compute the part of the sum in (14) where $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$:

$$\mathsf{ID}\left(\left(\tau-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot A^0_\tau\left(\left((\tau-1, \tau)|\bar{\mathbf{z}}\right)_j\right) \ .$$

By the definition of A^0_τ:

$$A^0_\tau\left(\left((\tau-1, \tau)|\bar{\mathbf{z}}\right)_j\right) = \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left((\tau-1, \tau)|\bar{\mathbf{z}}\right)_j\right) \cdot \left(\tilde{X}_\tau\left(\tau-1, \mathbf{s}_1\right) - b_1\right) \ .$$

The value $\tilde{X}_\tau\left(\tau-1, \mathbf{s}_1\right)$ is computed in (10). The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

6.4 Updating B

For $j' \in [m]$ and $\mathbf{b} = (b_1, b_2, b_3) \in \{0,1\}^3$ let let $j = m + j'$ and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$. We compute the value:

$$B_\tau^{j'}(\bar{\mathbf{z}}) = \sum_{0^m < \mathbf{h} < \tau} \mathsf{ID}\left(\left(\mathbf{h}, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) \ . \tag{15}$$

For $\tau = 0^m$ the sum is empty. For $\tau > 0^m$ we compute this sum in parts:

1. $0^m < \mathbf{h} < \tau$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$.
2. $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$.
3. $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$.

Part 1. We compute the part of the sum in (15) where $0^m < \mathbf{h} < \tau$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$:

$$\sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \tau)\,|\bar{\mathbf{z}}\right)_j\right)$$

By the definition of A_τ^0 and B_τ^0:

$$B_\tau^0\left(\left((\mathbf{h}, \tau)\,|\bar{\mathbf{z}}\right)_j\right) = A_\tau^0\left(\left((\mathbf{h}, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \cdot \left(\tilde{X}_\tau\left((\tau|\mathbf{t}_2)_{j'}, \mathbf{s}_2\right) - b_2\right)$$

Therefore, by the definition of $\bar{A}_\tau^{j'}$ we can write the sum in this part as:

$$\begin{aligned}
&\sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \\
&= \left(\tilde{X}_\tau\left((\tau|\mathbf{t}_2)_{j'}, \mathbf{s}_2\right) - b_2\right) \cdot \sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot A_\tau^0\left(\left((\mathbf{h}, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \\
&= \left(\tilde{X}_\tau\left((\tau|\mathbf{t}_2)_{j'}, \mathbf{s}_2\right) - b_2\right) \cdot \bar{A}_\tau^{j'}(\bar{\mathbf{z}})
\end{aligned}$$

The value $\bar{A}_\tau^{j'}(\bar{\mathbf{z}})$ is computed in (14). The value $\tilde{X}_\tau\left((\tau|\mathbf{t}_2)_{j'}, \mathbf{s}_2\right)$ is computed in (10). Therefore, we can efficiently compute the sum in this part.

Part 2. We compute the part of the sum in (15) where $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$\sum_{\substack{0^m < \mathbf{h} < \tau - 1 \\ (\mathbf{h}-1)[:j'] < \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) \ .$$

By Claim 4.7, for $\mathbf{h} < \tau - 1$:

$$\tilde{\varphi}_{y,\tau-1,\gamma_{\tau-1}}\left(\left((\mathbf{h}, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) = \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left((\mathbf{h}, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) \ .$$

By Claim 5.2, for $\mathbf{h} < \tau - 1$ such that $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$\tilde{X}_{\tau-1}\left(\mathbf{h}, \mathbf{s}_1\right) = \tilde{X}_{\tau}\left(\mathbf{h}, \mathbf{s}_1\right) \ ,$$
$$\tilde{X}_{\tau-1}\left(\left(\mathbf{h}-1|\mathbf{t}_2\right)_{j'}, \mathbf{s}_2\right) = \tilde{X}_{\tau}\left(\left(\mathbf{h}-1|\mathbf{t}_2\right)_{j'}, \mathbf{s}_2\right) \ .$$

Therefore, by the definition of B^0_τ for $\mathbf{h} < \tau - 1$ such that $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$B^0_{\tau}\left(\left(\left(\mathbf{h}, \mathbf{h}-1\right)|\bar{\mathbf{z}}\right)_j\right) = B^0_{\tau-1}\left(\left(\left(\mathbf{h}, \mathbf{h}-1\right)|\bar{\mathbf{z}}\right)_j\right) \ . \tag{16}$$

Therefore, by the definition of $B^{j'}_\tau$ we can write the sum in this part as:

$$\begin{aligned}
&\sum_{\substack{0^m<\mathbf{h}<\tau-1 \\ (\mathbf{h}-1)[:j']<\tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B^0_{\tau}\left(\left(\left(\mathbf{h}, \mathbf{h}-1\right)|\bar{\mathbf{z}}\right)_j\right) \\
=&\sum_{\substack{0^m<\mathbf{h}<\tau-1 \\ (\mathbf{h}-1)[:j']<\tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B^0_{\tau-1}\left(\left(\left(\mathbf{h}, \mathbf{h}-1\right)|\bar{\mathbf{z}}\right)_j\right) \\
=&B^{j'}_{\tau-1}(\bar{\mathbf{z}}) - \sum_{\substack{0^m<\mathbf{h}<\tau-1 \\ (\mathbf{h}-1)[:j']=\tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B^0_{\tau-1}\left(\left(\left(\mathbf{h}, \tau\right)|\bar{\mathbf{z}}\right)_j\right) \ .
\end{aligned}$$

The value $B^{j'}_{\tau-1}\left(\bar{\mathbf{z}}\right)$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. Therefore, to compute the sum in this part it is sufficient to compute the sum:

$$\sum_{\substack{0^m<\mathbf{h}<\tau-1 \\ (\mathbf{h}-1)[:j']=\tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B^0_{\tau-1}\left(\left(\left(\mathbf{h}, \tau\right)|\bar{\mathbf{z}}\right)_j\right) \ .$$

If $(\tau-1)[:j'] \neq \tau[:j']$ the sum is empty. Otherwise, if $(\tau-1)[:j'] = \tau[:j']$, By the Definition of A^0_τ, B^0_τ and $\bar{A}^{j'}_\tau(\bar{\mathbf{z}})$ we can write the sum as:

$$\begin{aligned}
&\sum_{\substack{0^m<\mathbf{h}<\tau-1 \\ (\mathbf{h}-1)[:j']=\tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B^0_{\tau-1}\left(\left(\left(\mathbf{h}, \tau\right)|\bar{\mathbf{z}}\right)_j\right) \\
=&\sum_{\substack{0^m<\mathbf{h}<\tau-1 \\ (\mathbf{h}-1)[:j']=(\tau-1)[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, (\tau-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B^0_{\tau-1}\left(\left(\left(\mathbf{h}, \tau-1\right)|\bar{\mathbf{z}}\right)_j\right) \\
=&\bar{A}^{j'}_{\tau-1}(\bar{\mathbf{z}}) \cdot \left(\tilde{X}_{\tau}\left(\left(\tau-1|\mathbf{t}_2\right)_{j'}, \mathbf{s}_2\right) - b_2\right) \ .
\end{aligned}$$

The value $\bar{A}^{j'}_{\tau-1}(\bar{\mathbf{z}})$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. The value $\tilde{X}_{\tau}\left(\left(\tau-1|\mathbf{t}_2\right)_{j'}, \mathbf{s}_2\right)$ is computed in (10). Therefore, we can efficiently compute the sum in this part.

Part 3. We compute the part of the sum in (15) where $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$\mathsf{ID}\left(\left(\tau-1, (\tau-2)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B^0_{\tau}\left(\left(\left(\tau-1, \tau-2\right)|\bar{\mathbf{z}}\right)_j\right) \ .$$

By the definition of B^0_τ:

$$\begin{aligned}
B^0_{\tau}\left(\left(\left(\tau-1, \tau-2\right)|\bar{\mathbf{z}}\right)_j\right) =&\tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left(\left(\tau-1, \tau-2\right)|\bar{\mathbf{z}}\right)_j\right) \\
&\cdot \left(\tilde{X}_{\tau}\left(\tau-1, \mathbf{s}_1\right) - b_1\right) \cdot \left(\tilde{X}_{\tau}\left(\left(\tau-2|\mathbf{t}_2\right)_{j'}, \mathbf{s}_2\right) - b_2\right) \ .
\end{aligned}$$

The value $\tilde{X}_{\tau}\left(\tau-1, \mathbf{s}_1\right)$ is computed in (10). The value $\tilde{X}_{\tau}\left(\left(\tau-2|\mathbf{t}_2\right)_{j'}, \mathbf{s}_2\right)$ is computed in (11). The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

6.5 Updating $\bar{B}$

For $j' \in [m]$ and $\mathbf{b} = (b_1, b_2, b_3) \in \{0,1\}^3$ let $j = 2m + j'$ and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$. We compute the value:

$$\bar{B}_\tau^{j'}(\bar{\mathbf{z}}) = \sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \ . \tag{17}$$

For $\tau = 0^m$ the sum is empty. For $\tau > 0^m$ we compute this sum in parts:

1. $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$.
2. $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$.

Part 1. We compute the part of the sum in (17) where $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$:

$$\sum_{\substack{0^m < \mathbf{h} < \tau - 1 \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \ .$$

First note that if $(\tau-1)[:j'] \neq \tau[:j']$ the sum is empty. Next we compute the sum in the case that $(\tau-1)[:j'] = \tau[:j']$. It follows from (16) (with $j' = m$) that for $\mathbf{h} < \tau - 1$:

$$B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) = B_{\tau-1}^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \ .$$

Therefore, we can write the sum in this part as:

$$\begin{aligned}
&\sum_{\substack{0^m < \mathbf{h} < \tau - 1 \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j']\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \\
=&\sum_{\substack{0^m < \mathbf{h} < \tau - 1 \\ (\mathbf{h}-1)[:j'] = (\tau-1)[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, (\tau-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_{\tau-1}^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau-1)\,|\bar{\mathbf{z}}\right)_j\right) \\
=&\bar{B}_{\tau-1}^{j'}(\bar{\mathbf{z}}) \ .
\end{aligned}$$

The value $\bar{B}_{\tau-1}^{j'}(\bar{\mathbf{z}})$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. Therefore, we can efficiently compute the sum in this part.

Part 2. We compute the part of the sum in (17) where $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$:

$$\mathsf{ID}\left(\left(\tau-1, \tau-2, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\tau-1, \tau-2, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \ .$$

By the definition of B_τ^0:

$$\begin{aligned}
B_\tau^0\left(\left((\tau-1, \tau-2, \tau)\,|\bar{\mathbf{z}}\right)_j\right) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left((\tau-1, \tau-2, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \\
&\cdot \left(\tilde{X}_\tau\left(\tau-1, \mathbf{s}_1\right) - b_1\right) \cdot \left(\tilde{X}_\tau\left(\tau-2, \mathbf{s}_2\right) - b_2\right) \ .
\end{aligned}$$

The value $\tilde{X}_\tau(\tau-1, \mathbf{s}_1)$ is computed in (10). The value $\tilde{X}_\tau(\tau-2, \mathbf{s}_2)$ is computed in (11). The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

6.6 Updating C

For $j' \in [m]$ and $\mathbf{b} = (b_1, b_2, b_3) \in \{0,1\}^3$ let $j = 2m + j'$ and $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$. We compute the value:

$$C_\tau^{j'}(\bar{\mathbf{z}}) = \sum_{0^m < \mathbf{h} < \tau} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) \quad . \tag{18}$$

For $\tau = 0^m$ the sum is empty. For $\tau > 0^m$ we compute this sum in parts:

1. $0^m < \mathbf{h} < \tau$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$.
2. $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$.
3. $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$.

Part 1. We compute the part of the sum in (18) where $0^m < \mathbf{h} < \tau$ and $(\mathbf{h}-1)[:j'] = \tau[:j']$:

$$\sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right)$$

By the definition of B_τ^0 and C_τ^0:

$$C_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) = B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \cdot \left(\tilde{X}_\tau\left(\left(\tau|\mathbf{t}_3\right)_{j'}, \mathbf{s}_3\right) - b_3\right)$$

Therefore, by the definition of $\bar{B}_\tau^{j'}$ we can write the sum in this part as:

$$\begin{aligned}
&\sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \\
&= \left(\tilde{X}_\tau\left(\left(\tau|\mathbf{t}_3\right)_{j'}, \mathbf{s}_3\right) - b_3\right) \cdot \sum_{\substack{0^m < \mathbf{h} < \tau \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot B_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \tau)\,|\bar{\mathbf{z}}\right)_j\right) \\
&= \left(\tilde{X}_\tau\left(\left(\tau|\mathbf{t}_3\right)_{j'}, \mathbf{s}_3\right) - b_3\right) \cdot \bar{B}_\tau^{j'}(\bar{\mathbf{z}})
\end{aligned}$$

The value $\bar{B}_\tau^{j'}(\bar{\mathbf{z}})$ is computed in (17). The value $\tilde{X}_\tau\left(\left(\tau|\mathbf{t}_3\right)_{j'}, \mathbf{s}_3\right)$ is computed in (10). Therefore, we can efficiently compute the sum in this part.

Part 2. We compute the part of the sum in (18) where $0^m < \mathbf{h} < \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$\sum_{\substack{0^m < \mathbf{h} < \tau - 1 \\ (\mathbf{h}-1)[:j'] < \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C_\tau^0\left(\left((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) \quad .$$

By Claim 4.7, for $\mathbf{h} < \tau - 1$:

$$\tilde{\varphi}_{y,\tau-1,\gamma_{\tau-1}}\left(\left((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) = \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)\,|\bar{\mathbf{z}}\right)_j\right) \quad .$$

By Claim 5.2, for $\mathbf{h} < \tau - 1$ such that $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$\begin{aligned}\tilde{X}_{\tau-1}(\mathbf{h}, \mathbf{s}_1) &= \tilde{X}_{\tau}(\mathbf{h}, \mathbf{s}_1) \quad , \\ \tilde{X}_{\tau-1}(\mathbf{h}-1, \mathbf{s}_2) &= \tilde{X}_{\tau}(\mathbf{h}-1, \mathbf{s}_2) \quad , \\ \tilde{X}_{\tau-1}\left((\mathbf{h}-1|\mathbf{t}_3)_{j'}, \mathbf{s}_3\right) &= \tilde{X}_{\tau}\left((\mathbf{h}-1|\mathbf{t}_3)_{j'}, \mathbf{s}_3\right) \quad .\end{aligned}$$

Therefore, by the definition of C^0_τ for $\mathbf{h} < \tau - 1$ such that $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$C^0_{\tau}\left(((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)|\bar{\mathbf{z}})_j\right) = C^0_{\tau-1}\left(((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)|\bar{\mathbf{z}})_j\right) \quad . \tag{19}$$

Therefore, by the definition of $C^{j'}_\tau$ we can write the sum in this part as:

$$\begin{aligned}&\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] < \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_{\tau}\left(((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)|\bar{\mathbf{z}})_j\right) \\ =&\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] < \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, (\mathbf{h}-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_{\tau-1}\left(((\mathbf{h}, \mathbf{h}-1, \mathbf{h}-1)|\bar{\mathbf{z}})_j\right) \\ =&C^{j'}_{\tau-1}(\bar{\mathbf{z}}) - \sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_{\tau-1}\left(((\mathbf{h}, \mathbf{h}-1, \tau)|\bar{\mathbf{z}})_j\right) \quad .\end{aligned}$$

The value $C^{j'}_{\tau-1}(\bar{\mathbf{z}})$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. Therefore, to compute the sum in this part it is sufficient to compute the sum:

$$\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_{\tau-1}\left(((\mathbf{h}, \mathbf{h}-1, \tau)|\bar{\mathbf{z}})_j\right) \quad .$$

If $(\tau-1)[:j'] \neq \tau[:j']$ the sum is empty. Otherwise, if $(\tau-1)[:j'] = \tau[:j']$, By the Definition of B^0_τ, C^0_τ and $\bar{B}^{j'}_\tau(\bar{\mathbf{z}})$ we can write the sum as:

$$\begin{aligned}&\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] = \tau[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, \tau[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_{\tau-1}\left(((\mathbf{h}, \mathbf{h}-1, \tau)|\bar{\mathbf{z}})_j\right) \\ =&\sum_{\substack{0^m < \mathbf{h} < \tau-1 \\ (\mathbf{h}-1)[:j'] = (\tau-1)[:j']}} \mathsf{ID}\left(\left(\mathbf{h}, \mathbf{h}-1, (\tau-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_{\tau-1}\left(((\mathbf{h}, \mathbf{h}-1, \tau-1)|\bar{\mathbf{z}})_j\right) \\ =&\bar{B}^{j'}_{\tau-1}(\bar{\mathbf{z}}) \cdot \left(\tilde{X}_{\tau}\left((\tau-1|\mathbf{t}_3)_{j'}, \mathbf{s}_3\right) - b_3\right) \quad .\end{aligned}$$

The value $\bar{B}^{j'}_{\tau-1}(\bar{\mathbf{z}})$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. The value $\tilde{X}_{\tau}\left((\tau-1|\mathbf{t}_3)_{j'}, \mathbf{s}_3\right)$ is computed in (10). Therefore, we can efficiently compute the sum in this part.

Part 3. We compute the part of the sum in (18) where $0^m < \mathbf{h} = \tau - 1$ and $(\mathbf{h}-1)[:j'] < \tau[:j']$:

$$\mathsf{ID}\left(\left(\tau-1, \tau-2, (\tau-2)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_{\tau}\left(((\tau-1, \tau-2, \tau-2)|\bar{\mathbf{z}})_j\right) \quad .$$

By the definition of C^0_τ:

$$\begin{aligned} C^0_\tau\left(\left((\tau-1,\tau-2,\tau-2)\,|\bar{\mathbf{z}}\right)_j\right) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left((\tau-1,\tau-2,\tau-2)\,|\bar{\mathbf{z}}\right)_j\right) \\ & \cdot\left(\tilde{X}_\tau\left(\tau-1,\mathbf{s}_1\right)-b_1\right)\cdot\left(\tilde{X}_\tau\left(\tau-2,\mathbf{s}_2\right)-b_2\right) \\ & \cdot\left(\tilde{X}_\tau\left(\left(\tau-2|\mathbf{t}_3\right)_{j'},\mathbf{s}_3\right)-b_3\right) \ . \end{aligned}$$

The value $\tilde{X}_\tau\left(\tau-1,\mathbf{s}_1\right)$ is computed in (10). The values $\tilde{X}_\tau\left(\tau-2,\mathbf{s}_2\right)$ and $\tilde{X}_\tau\left(\left(\tau-2|\mathbf{t}_3\right)_{j'},\mathbf{s}_3\right)$ are computed in (11) The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

6.7 Updating $\bar{C}$

For $j'\in[3k]$ and $\mathbf{b}=(b_1,b_2,b_3)\in\{0,1\}^3$ let $j=3m+j'$ and $\bar{\mathbf{z}}=(\mathbf{z},\mathbf{b})$. We compute the value:

$$\bar{C}^{j'}_\tau(\bar{\mathbf{z}})=\sum_{\substack{0^m<\mathbf{h}<\tau\\ \mathbf{h}'\in\mathbb{H}^{j'}}}\mathsf{ID}\left(\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}'\right),\bar{\mathbf{z}}[:j]\right)\cdot C^0_\tau\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}',\bar{\mathbf{z}}[j+1:]\right)\ . \tag{20}$$

For $\tau=0^m$ the sum is empty. For $\tau>0^m$ we compute this sum in parts:

1. $0^m<\mathbf{h}<\tau-1$.
2. $0^m<\mathbf{h}=\tau-1$.

Part 1. We compute the part of the sum in (20) where $0^m<\mathbf{h}<\tau-1$:

$$\sum_{\substack{0^m<\mathbf{h}<\tau-1\\ \mathbf{h}'\in\mathbb{H}^{j'}}}\mathsf{ID}\left(\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}'\right),\bar{\mathbf{z}}[:j]\right)\cdot C^0_\tau\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}',\bar{\mathbf{z}}[j+1:]\right)\ .$$

It follows from (19) (with $j'=m$) that for $\mathbf{h}<\tau-1$:

$$C^0_\tau\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}',\bar{\mathbf{z}}[:j+1]\right)=C^0_{\tau-1}\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}',\bar{\mathbf{z}}[:j+1]\right)\ .$$

Therefore, we can write the sum in this part as:

$$\begin{aligned} &\sum_{\substack{0^m<\mathbf{h}<\tau-1\\ \mathbf{h}'\in\mathbb{H}^{j'}}}\mathsf{ID}\left(\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}'\right),\bar{\mathbf{z}}[:j]\right)\cdot C^0_\tau\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}',\bar{\mathbf{z}}[j+1:]\right)\\ =&\sum_{\substack{0^m<\mathbf{h}<\tau-1\\ \mathbf{h}'\in\mathbb{H}^{j'}}}\mathsf{ID}\left(\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}'\right),\bar{\mathbf{z}}[:j]\right)\cdot C^0_{\tau-1}\left(\mathbf{h},\mathbf{h}-1,\mathbf{h}-1,\mathbf{h}',\bar{\mathbf{z}}[j+1:]\right)\\ =&\bar{C}^{j'}_{\tau-1}(\bar{\mathbf{z}})\ . \end{aligned}$$

The value $\bar{C}^{j'}_{\tau-1}(\bar{\mathbf{z}})$ is contained in the previous symbol $\bar{\sigma}_{\tau-1}$. Therefore, we can efficiently compute the sum in this part.

Part 2. We compute the part of the sum in (20) where $0^m < \mathbf{h} = \tau - 1$:

$$\sum_{\mathbf{h}' \in \mathbb{H}^{j'}} \mathsf{ID}\left(\left(\tau-1, \tau-2, \tau-2, \mathbf{h}'\right), \bar{\mathbf{z}}[:j]\right) \cdot C^0_\tau(\tau-1, \tau-2, \tau-2, \mathbf{h}', \bar{\mathbf{z}}[j+1:]) \ .$$

Let $\mathbf{s}'_1, \mathbf{s}'_2, \mathbf{s}'_3 \in \mathbb{F}^k$ be such that $(\mathbf{s}'_1, \mathbf{s}'_2, \mathbf{s}'_3) = (\mathbf{h}', \bar{\mathbf{z}}[j+1:])$. By the definition of C^0_τ:

$$\begin{aligned} C^0_\tau\left(\tau-1, \tau-2, \tau-2, \mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\tau-1, \tau-2, \tau-2, \mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \\ & \cdot \left(\tilde{X}_\tau\left(\tau-1, \mathbf{s}'_1\right) - b_1\right) \cdot \left(\tilde{X}_\tau\left(\tau-2, \mathbf{s}'_2\right) - b_2\right) \\ & \cdot \left(\tilde{X}_\tau\left(\tau-2, \mathbf{s}'_3\right) - b_3\right) \ . \end{aligned}$$

By Claim 5.1, given $\gamma_{\tau-1}$ and $\gamma_{\tau-2}$ we can efficiently compute the values:

$$\tilde{X}_\tau\left(\tau-1, \mathbf{s}'_1\right) \quad , \quad \tilde{X}_\tau\left(\tau-2, \mathbf{s}'_2\right) \quad , \quad \tilde{X}_\tau\left(\tau-2, \mathbf{s}'_3\right) \ .$$

The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, since this part contains $|\mathbb{H}|^{j'} < \mathrm{poly}(\mathsf{S})$ summands, we can efficiently compute the sum in this part.

6.8 Updating Q

For $j \in [0, 3(m+k)]$ and $\mathbf{b} = (b_1, b_2, b_3) \in \{0,1\}^3$ let $\bar{\mathbf{z}} = (\mathbf{z}, \mathbf{b})$. We compute the value $Q^j_\tau(\bar{\mathbf{z}})$ in each of the following cases:

1. $j = 0$.
2. $j \in [m]$.
3. $j \in [m+1, 2m]$.
4. $j \in [2m+1, 3m]$.
5. $j \in [3m+1, 3(m+k)]$.

6.8.1 Case 1: $j = 0$.

By the definition of the polynomial Q^0_τ:

$$Q^0_\tau(\bar{\mathbf{z}}) = \mathsf{LE}(\mathbf{t}_1, \tau) \cdot \tilde{\varphi}_{y,\tau,\gamma_\tau}(\bar{\mathbf{z}}) \cdot \prod_{i \in [3]} \left(\tilde{X}_\tau(\mathbf{t}_i, \mathbf{s}_i) - b_i\right) \ .$$

The values $\tilde{X}_\tau(\mathbf{t}_i, \mathbf{s}_i)$ for $i \in [3]$ are computed in (10). The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute $Q^0_\tau(\bar{\mathbf{z}})$.

6.8.2 Case 2: $j \in [m]$.

In this case we can write Q^j_τ as:

$$Q^j_\tau(\bar{\mathbf{z}}) = \sum_{\mathbf{h} \in \mathbb{H}^j} \mathsf{ID}(\mathbf{h}, \mathbf{t}_1[:j]) \cdot Q^0_\tau(\mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ . \tag{21}$$

We compute this sum in parts:

1. $\mathbf{h} > \tau[:j]$.
2. $\mathbf{h} = \tau[:j]$.
3. $\mathbf{h} < \tau[:j]$.

Part 1. We compute the part of the sum in (21) where $\mathbf{h} > \tau[:j]$:

$$\sum_{\mathbf{h}>\tau[:j]} \mathsf{ID}\left(\mathbf{h}, \mathbf{t}_1[:j]\right) \cdot Q_\tau^0\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \ .$$

By Claim 4.2, for $\mathbf{h} > \tau[:j]$,

$$\mathsf{LE}_m\left(\left(\mathbf{h}, \mathbf{t}_1[j+1:]\right), \tau\right) = 0 \ .$$

Therefore, by the definition of Q_τ^0:

$$Q_\tau^0\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) = 0 \ ,$$

and the sum in this part is 0.

Part 2. We compute the part of the sum in (21) where $\mathbf{h} = \tau[:j]$:

$$\mathsf{ID}\left(\tau[:j], \mathbf{t}_1[:j]\right) \cdot Q_\tau^0\left(\left(\tau|\bar{\mathbf{z}}\right)_j\right) \ .$$

By the definition of Q_τ^0:

$$\begin{aligned} Q_\tau^0\left(\left(\tau|\bar{\mathbf{z}}\right)_j\right) =& \mathsf{LE}_m\left(\left(\tau|\mathbf{t}_1\right)_j, \tau\right) \cdot \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left(\tau|\bar{\mathbf{z}}\right)_j\right) \\ & \cdot \left(\tilde{X}_\tau\left(\left(\tau|\mathbf{t}_1\right)_j, \mathbf{s}_1\right) - b_1\right) \cdot \prod_{i\in[2,3]} \left(\tilde{X}_\tau\left(\mathbf{t}_i, \mathbf{s}_i\right) - b_i\right) \\ & + \left(1 - \mathsf{LE}_m\left(\left(\tau|\mathbf{t}_1\right)_j, \tau\right)\right) \cdot \tilde{X}_\tau\left(\left(\tau|\mathbf{t}_1\right)_j, \mathbf{s}_1\right) \ . \end{aligned}$$

The following values are computed in (10):

$$\tilde{X}_\tau\left(\left(\tau|\mathbf{t}_1\right)_j, \mathbf{s}_1\right) \quad , \quad \tilde{X}_\tau\left(\mathbf{t}_2, \mathbf{s}_2\right) \quad , \quad \tilde{X}_\tau\left(\mathbf{t}_3, \mathbf{s}_3\right) \ .$$

The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

Part 3. We compute the part of the sum in (21) where $\mathbf{h} < \tau[:j]$:

$$\sum_{\mathbf{h}<\tau[:j]} \mathsf{ID}(\mathbf{h}, \mathbf{t}_1[:j]) \cdot Q_\tau^0(\mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ .$$

By Claim 4.2, for $\mathbf{h} < \tau[:j]$ we have $\mathsf{LE}_m((\mathbf{h}|\mathbf{t}_1)_j, \tau) = 1$. Therefore, by the definition of Q_τ^0 and A_τ^0:

$$Q_\tau^0(\mathbf{h}, \bar{\mathbf{z}}[j+1:]) = A_\tau^0\left(\mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \cdot \prod_{i\in[2,3]} \left(\tilde{X}_\tau(\mathbf{t}_i, \mathbf{s}_i) - b_i\right) \ .$$

Therefore, by the definition of A_τ^j:

$$\sum_{\mathbf{h}<\tau[:j]} \mathsf{ID}(\mathbf{h}, \mathbf{t}_1[:j]) \cdot Q_\tau^0(\mathbf{h}, \bar{\mathbf{z}}[j+1:]) = A_\tau^j\left(\bar{\mathbf{z}}\right) \cdot \prod_{i\in[2,3]} \left(\tilde{X}_\tau(\mathbf{t}_i, \mathbf{s}_i) - b_i\right) \ .$$

The value $A_\tau^j\left(\bar{\mathbf{z}}\right)$ was computed in (12) and the values $\tilde{X}_\tau\left(\mathbf{t}_2, \mathbf{s}_2\right)$ and $\tilde{X}_\tau\left(\mathbf{t}_3, \mathbf{s}_3\right)$ are computed in (10). Therefore, we can efficiently compute the sum in this part.

6.8.3 Case 3: $j \in [m+1, 2m]$.

Let $j' = j - m \in [m]$. We can write Q_τ^j as

$$Q_\tau^j(\bar{\mathbf{z}}) = \sum_{\substack{\mathbf{h}_1 \in \mathbb{H}^m \\ \mathbf{h}_2 \in \mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]) \ . \tag{22}$$

We compute this sum in parts:

1. $\mathbf{h}_1 > \tau$.
2. $0^m < \mathbf{h}_1 = \tau$ and $\mathbf{h}_2 = (\mathbf{h}_1 - 1)[:j']$.
3. $0^m < \mathbf{h}_1 < \tau$ and $\mathbf{h}_2 = (\mathbf{h}_1 - 1)[:j']$.
4. $\mathbf{h}_1 = 0^m$ or ($0^m < \mathbf{h}_1 \leq \tau$ and $\mathbf{h}_2 \neq (\mathbf{h}_1 - 1)[:j']$).

Part 1. We compute the part of the sum in (22) where $\mathbf{h}_1 > \tau$:

$$\sum_{\substack{\mathbf{h}_1 > \tau \\ \mathbf{h}_2 \in \mathbb{H}^{j'}}} \mathsf{ID}((\mathbf{h}_1, \mathbf{h}_2), \bar{\mathbf{z}}[:j]) \cdot Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]) \ .$$

For $\mathbf{h}_1 > \tau$, by the definition of Q_τ^0:

$$Q_\tau^0\left(\mathbf{h}_1, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]\right) = 0 \ .$$

Therefore, the sum in this part is 0.

Part 2. We compute the part of the sum in (22) where $0^m < \mathbf{h}_1 = \tau$ and $\mathbf{h}_2 = (\mathbf{h}_1 - 1)[:j']$:

$$\mathsf{ID}\left(\left(\tau, (\tau-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0\left(\left((\tau, \tau-1)\,|\bar{\mathbf{z}}\right)_j\right) \ .$$

By the definition of Q_τ^0 for $\mathbf{h}_1 = \tau$:

$$\begin{aligned} Q_\tau^0\left(\left((\tau, \tau-1)\,|\bar{\mathbf{z}}\right)_j\right) = & \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\left((\tau, \tau-1)\,|\bar{\mathbf{z}}\right)_j\right) \\ & \cdot \left(\tilde{X}_\tau\left(\tau, \mathbf{s}_1\right) - b_1\right) \cdot \left(\tilde{X}_\tau\left((\tau-1|\mathbf{t}_2)_{j'}, \mathbf{s}_2\right) - b_1\right) \cdot \left(\tilde{X}_\tau\left(\mathbf{t}_3, \mathbf{s}_3\right) - b_3\right) \ . \end{aligned}$$

The following values are computed in (10):

$$\tilde{X}_\tau\left(\tau, \mathbf{s}_1\right) \quad , \quad \tilde{X}_\tau\left((\tau-1|\mathbf{t}_2)_{j'}, \mathbf{s}_2\right) \quad , \quad \tilde{X}_\tau\left(\mathbf{t}_3, \mathbf{s}_3\right) \ .$$

The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

Part 3. We compute the part of the sum in (22) where $0^m < \mathbf{h}_1 < \tau$ and $\mathbf{h}_2 = (\mathbf{h}_1 - 1)[:j']$:

$$\sum_{0^m < \mathbf{h}_1 < \tau} \mathsf{ID}\left(\left(\mathbf{h}_1, (\mathbf{h}_1 - 1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau\left(\left((\mathbf{h}_1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}}\right)_j\right) \ .$$

By the definition of Q^0_τ and B^0_τ, for $\mathbf{h}_1 < \tau$:

$$Q^0_\tau(((\mathbf{h}_1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}})_j) = B^0_\tau(((\mathbf{h}_1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}})_j) \cdot \left(\tilde{X}_\tau\left(\mathbf{t}_3, \mathbf{s}_3\right) - b_3\right) \ .$$

Therefore, by the definition of $B^{j'}_\tau$:

$$\sum_{0^m < \mathbf{h}_1 < \tau} \mathsf{ID}\left(\left(\mathbf{h}_1, (\mathbf{h}_1 - 1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau\left(\left((\mathbf{h}_1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}}\right)_j\right) = B^{j'}_\tau\left(\bar{\mathbf{z}}\right) \cdot \left(\tilde{X}_\tau\left(\mathbf{t}_3, \mathbf{s}_3\right) - b_3\right) \ .$$

The value $B^{j'}_\tau\left(\bar{\mathbf{z}}\right)$ was computed in (15) and the value $\tilde{X}_\tau\left(\mathbf{t}_3, \mathbf{s}_3\right)$ is computed in (10). Therefore, we can efficiently compute the sum in this part.

Part 4. We compute the part of the sum in (22) where $\mathbf{h}_1 = 0^m$:

$$\sum_{\mathbf{h}_2 \in \mathbb{H}^{j'}} \mathsf{ID}\left((0^m, \mathbf{h}_2), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(0^m, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]) \ .$$

Or, where $0^m < \mathbf{h}_1 \leq \tau$ and $\mathbf{h}_2 \neq (\mathbf{h}_1 - 1)[:j']$:

$$\sum_{\substack{0^m < \mathbf{h}_1 \leq \tau \\ \mathbf{h}_2 \neq (\mathbf{h}_1 - 1)[:j']}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(\mathbf{h}_1, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]) \ .$$

By Claim 4.7 in both cases $\tilde{\varphi}_{y,\tau,\gamma_\tau}(\mathbf{h}_1, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]) = 0$. Therefore, by the definition of Q^0_τ for $\mathbf{h}_1 \leq \tau$:

$$\begin{aligned} Q^0_\tau(\mathbf{h}_1, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}(\mathbf{h}_1, \mathbf{h}_2, \bar{\mathbf{z}}[j+1:]) \\ & \cdot \left(\tilde{X}_\tau(\mathbf{h}_1, \mathbf{s}_1) - b_1\right) \cdot \left(\tilde{X}_\tau((\mathbf{h}_2|\mathbf{t}_2)_{j'}, \mathbf{s}_2) - b_2\right) \cdot \left(\tilde{X}_\tau(\mathbf{t}_3, \mathbf{s}_3) - b_3\right) = 0 \ . \end{aligned}$$

Therefore, the sum in this part is 0.

6.8.4 Case 4: $j \in [2m+1, 3m]$.

Let $j' = j - 2m \in [m]$. We can write Q^j_τ as

$$Q^j_\tau(\bar{\mathbf{z}}) = \sum_{\substack{\mathbf{h}_1, \mathbf{h}_2 \in \mathbb{H}^m \\ \mathbf{h}_3 \in \mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) \ . \tag{23}$$

We compute this sum in parts:

1. $\mathbf{h}_1 > \tau$.
2. $0^m < \mathbf{h}_1 = \tau$ and $\mathbf{h}_2 = \mathbf{h}_1 - 1$ and $\mathbf{h}_3 = (\mathbf{h}_1 - 1)[:j']$.
3. $0^m < \mathbf{h}_1 < \tau$ and $\mathbf{h}_2 = \mathbf{h}_1 - 1$ and $\mathbf{h}_3 = (\mathbf{h}_1 - 1)[:j']$.
4. $\mathbf{h}_1 = 0^m$ or ($0^m < \mathbf{h}_1 \leq \tau$ and ($\mathbf{h}_2 \neq \mathbf{h}_1 - 1$ or $\mathbf{h}_3 \neq (\mathbf{h}_1 - 1)[:j']$)).

Part 1. We compute the part of the sum in (23) where $\mathbf{h}_1 > \tau$:

$$\sum_{\substack{\mathbf{h}_1 > \tau \\ \mathbf{h}_2 \in \mathbb{H}^m \\ \mathbf{h}_3 \in \mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) \ .$$

For $\mathbf{h}_1 > \tau$, by the definition of Q_τ^0:

$$Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) = 0 \ .$$

Therefore, the sum in this part is 0.

Part 2. We compute the part of the sum in (23) where $0^m < \mathbf{h}_1 = \tau$, $\mathbf{h}_2 = \mathbf{h}_1 - 1$ and $\mathbf{h}_3 = (\mathbf{h}_1 - 1)[:j']$:

$$\mathsf{ID}\left(\left(\tau, \tau-1, (\tau-1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(((\tau, \tau-1, \tau-1)\,|\bar{\mathbf{z}})_j) \ .$$

By the definition of Q_τ^0 for $\mathbf{h}_1 = \tau$:

$$\begin{aligned} Q_\tau^0\left(((\tau, \tau-1, \tau-1)\,|\bar{\mathbf{z}})_j\right) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(((\tau, \tau-1, \tau-1)\,|\bar{\mathbf{z}})_j\right) \\ & \cdot \left(\tilde{X}_\tau(\tau, \mathbf{s}_1) - b_1\right) \cdot \left(\tilde{X}_\tau(\tau-1, \mathbf{s}_2) - b_1\right) \\ & \cdot \left(\tilde{X}_\tau\left((\tau-1|\mathbf{t}_3)_{j'}, \mathbf{s}_3\right) - b_3\right) \ . \end{aligned}$$

The following values are computed in (10):

$$\tilde{X}_\tau(\tau, \mathbf{s}_1) \quad , \quad \tilde{X}_\tau(\tau-1, \mathbf{s}_2) \quad , \quad \tilde{X}_\tau\left((\tau-1|\mathbf{t}_3)_{j'}, \mathbf{s}_3\right) \ .$$

The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, we can efficiently compute the summand in this part.

Part 3. We compute the part of the sum in (23) where $0^m < \mathbf{h}_1 < \tau$, $\mathbf{h}_2 = \mathbf{h}_1 - 1$ and $\mathbf{h}_3 = (\mathbf{h}_1 - 1)[:j']$:

$$\sum_{0^m < \mathbf{h}_1 < \tau} \mathsf{ID}\left(\left(\mathbf{h}_1, \mathbf{h}_1 - 1, (\mathbf{h}_1 - 1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0\left(((\mathbf{h}_1, \mathbf{h}_1 - 1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}})_j\right) \ .$$

By the definition of Q_τ^0 and C_τ^0, for $\mathbf{h}_1 < \tau$:

$$Q_\tau^0(((\mathbf{h}_1, \mathbf{h}_1 - 1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}})_j) = C_\tau^0(((\mathbf{h}_1, \mathbf{h}_1 - 1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}})_j) \ .$$

Therefore, by the definition of $C_\tau^{j'}$:

$$\sum_{0^m < \mathbf{h}_1 < \tau} \mathsf{ID}\left(\left(\mathbf{h}_1, \mathbf{h}_1 - 1, (\mathbf{h}_1 - 1)[:j']\right), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0\left(((\mathbf{h}_1, \mathbf{h}_1 - 1, \mathbf{h}_1 - 1)\,|\bar{\mathbf{z}})_j\right) = C_\tau^{j'}(\bar{\mathbf{z}}) \ .$$

The value $C_\tau^{j'}(\bar{\mathbf{z}})$ was computed in (18). Therefore, we can efficiently compute the sum in this part.

Part 4. We compute the part of the sum in (23) where $\mathbf{h}_1 = 0^m$:

$$\sum_{\substack{\mathbf{h}_2 \in \mathbb{H}^m \\ \mathbf{h}_3 \in \mathbb{H}^{j'}}} \mathsf{ID}\left((0^m, \mathbf{h}_2, \mathbf{h}_3), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(0^m, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) \ .$$

Or, where $0^m < \mathbf{h}_1 \leq \tau$ and $\mathbf{h}_2 \neq \mathbf{h}_1 - 1$:

$$\sum_{\substack{0^m < \mathbf{h}_1 \leq \tau \\ \mathbf{h}_2 \neq \mathbf{h}_1 - 1 \\ \mathbf{h}_3 \in \mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) \ .$$

Or, where $0^m < \mathbf{h}_1 \leq \tau$, $\mathbf{h}_2 = \mathbf{h}_1 - 1$ and $\mathbf{h}_3 \neq (\mathbf{h}_1 - 1)[:j']$:

$$\sum_{\substack{0^m < \mathbf{h}_1 \leq \tau \\ \mathbf{h}_3 \neq (\mathbf{h}_1 - 1)[:j']}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_1 - 1, \mathbf{h}_3), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(\mathbf{h}_1, \mathbf{h}_1 - 1, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) \ .$$

By Claim 4.7 in all three cases $\tilde{\varphi}_{y,\tau,\gamma_\tau}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) = 0$. Therefore, by the definition of Q_τ^0 for $\mathbf{h}_1 \leq \tau$:

$$\begin{aligned} Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \bar{\mathbf{z}}[j+1:]) \\ & \cdot \left(\tilde{X}_\tau(\mathbf{h}_1, \mathbf{s}_1) - b_1\right) \cdot \left(\tilde{X}_\tau(\mathbf{h}_2, \mathbf{s}_2) - b_2\right) \cdot \left(\tilde{X}_\tau((\mathbf{h}_3|\mathbf{t}_3)_{j'}, \mathbf{s}_3) - b_3\right) = 0 \ . \end{aligned}$$

Therefore, the sum in this part is 0.

6.8.5 Case 5: $j \in [3m+1, 3(m+k)]$.

Let $j' = j - 3m \in [3k]$. We can write Q_τ^j as

$$Q_\tau^j(\bar{\mathbf{z}}) = \sum_{\substack{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3 \in \mathbb{H}^m \\ \mathbf{h} \in \mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ . \tag{24}$$

We compute this sum in parts:

1. $\mathbf{h}_1 > \tau$.
2. $0^m < \mathbf{h}_1 = \tau$ and $\mathbf{h}_2 = \mathbf{h}_1 - 1$ and $\mathbf{h}_3 = \mathbf{h}_1 - 1$.
3. $0^m < \mathbf{h}_1 < \tau$ and $\mathbf{h}_2 = \mathbf{h}_1 - 1$ and $\mathbf{h}_3 = \mathbf{h}_1 - 1$.
4. $\mathbf{h}_1 = 0^m$ or ($0^m < \mathbf{h}_1 \leq \tau$ and ($\mathbf{h}_2 \neq \mathbf{h}_1 - 1$ or $\mathbf{h}_3 \neq \mathbf{h}_1 - 1$)).

Part 1. We compute the part of the sum in (24) where $\mathbf{h}_1 > \tau$:

$$\sum_{\substack{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3 \in \mathbb{H}^m \\ \mathbf{h} \in \mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}), \bar{\mathbf{z}}[:j]\right) \cdot Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ .$$

For $\mathbf{h}_1 > \tau$, by the definition of Q_τ^0:

$$Q_\tau^0(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) = 0$$

Therefore, the sum in this part is 0.

Part 2. We compute the part of the sum in (24) where $0^m < \mathbf{h}_1 = \tau$, $\mathbf{h}_2 = \mathbf{h}_3 = \mathbf{h}_1 - 1$:

$$\sum_{\mathbf{h}\in\mathbb{H}^{j'}} \mathsf{ID}\left((\tau, \tau-1, \tau-1, \mathbf{h}), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(\tau, \tau-1, \tau-1, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ .$$

Let $\mathbf{s}'_1, \mathbf{s}'_2, \mathbf{s}'_3 \in \mathbb{F}^k$ be such that $(\mathbf{s}'_1, \mathbf{s}'_2, \mathbf{s}'_3) = (\mathbf{h}, \bar{\mathbf{z}}[j+1:])$. By the definition of Q^0_τ for $\mathbf{h}_1 = \tau$:

$$\begin{aligned} Q^0_\tau\left(\tau, \tau-1, \tau-1, \mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}\left(\tau, \tau-1, \tau-1, \mathbf{h}, \bar{\mathbf{z}}[j+1:]\right) \\ & \cdot \left(\tilde{X}_\tau\left(\tau, \mathbf{s}'_1\right) - b_1\right) \cdot \left(\tilde{X}_\tau\left(\tau-1, \mathbf{s}'_2\right) - b_2\right) \\ & \cdot \left(\tilde{X}_\tau\left(\tau-1, \mathbf{s}'_3\right) - b_3\right) \ . \end{aligned}$$

By Claim 5.1, given γ_τ and $\gamma_{\tau-1}$ we can efficiently compute the values:

$$\tilde{X}_\tau\left(\tau, \mathbf{s}'_1\right) \quad , \quad \tilde{X}_\tau\left(\tau-1, \mathbf{s}'_2\right) \quad , \quad \tilde{X}_\tau\left(\tau-1, \mathbf{s}'_3\right) \ .$$

The circuit $\tilde{\varphi}_{y,\tau,\gamma_\tau}$ can be efficiently evaluated given γ_τ. Therefore, since this part contains $|\mathbb{H}|^{j'} < \mathrm{poly}(\mathsf{S})$ summands, we can efficiently compute the sum in this part.

Part 3. We compute the part of the sum in (24) where $0^m < \mathbf{h}_1 < \tau$ and $\mathbf{h}_2 = \mathbf{h}_3 = \mathbf{h}_1 - 1$.

$$\sum_{\substack{0^m < \mathbf{h}_1 < \tau \\ \mathbf{h}\in\mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_1-1, \mathbf{h}_1-1, \mathbf{h}), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(\mathbf{h}_1, \mathbf{h}_1-1, \mathbf{h}_1-1, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ .$$

By the definition of Q^0_τ and C^0_τ, for $\mathbf{h}_1 < \tau$:

$$Q^0_\tau(((\mathbf{h}_1, \mathbf{h}_1-1, \mathbf{h}_1-1)\,|\bar{\mathbf{z}})_j) = C^0_\tau(((\mathbf{h}_1, \mathbf{h}_1-1, \mathbf{h}_1-1)\,|\bar{\mathbf{z}})_j) \ .$$

Therefore, by the definition of $\bar{C}^{j'}_\tau$:

$$\sum_{\substack{0^m < \mathbf{h}_1 < \tau \\ \mathbf{h}\in\mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_1-1, \mathbf{h}_1-1, \mathbf{h}), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(\mathbf{h}_1, \mathbf{h}_1-1, \mathbf{h}_1-1, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) = \bar{C}^{j'}_\tau\left(\bar{\mathbf{z}}\right) \ .$$

The value $\bar{C}^{j'}_\tau\left(\bar{\mathbf{z}}\right)$ was computed in (20). Therefore, we can efficiently compute the sum in this part.

Part 4. We compute the part of the sum in (24) where $\mathbf{h}_1 = 0^m$:

$$Q^j_\tau(\bar{\mathbf{z}}) = \sum_{\substack{\mathbf{h}_2, \mathbf{h}_3 \in \mathbb{H}^m \\ \mathbf{h}\in\mathbb{H}^{j'}}} \mathsf{ID}\left((0^m, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(0^m, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ .$$

Or, where $0^m < \mathbf{h}_1 \leq \tau$ and ($\mathbf{h}_2 \neq \mathbf{h}_1 - 1$ or $\mathbf{h}_3 \neq \mathbf{h}_1 - 1$):

$$\sum_{\substack{0^m < \mathbf{h}_1 \leq \tau \\ (\mathbf{h}_2 \neq \mathbf{h}_1-1) \vee (\mathbf{h}_3 \neq \mathbf{h}_1-1) \\ \mathbf{h}\in\mathbb{H}^{j'}}} \mathsf{ID}\left((\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}), \bar{\mathbf{z}}[:j]\right) \cdot Q^0_\tau(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) \ .$$

By Claim 4.7 in both cases $\tilde{\varphi}_{y,\tau,\gamma_\tau}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) = 0$. Therefore, by the definition of Q^0_τ for $\mathbf{h}_1 \leq \tau$ and for $(\mathbf{s}'_1, \mathbf{s}'_2, \mathbf{s}'_3) = (\mathbf{h}, \bar{\mathbf{z}}[j+1:])$:

$$\begin{aligned} Q^0_\tau(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) =& \tilde{\varphi}_{y,\tau,\gamma_\tau}(\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \mathbf{h}, \bar{\mathbf{z}}[j+1:]) \\ & \cdot \left(\tilde{X}_\tau(\mathbf{h}_1, \mathbf{s}'_1) - b_1\right) \cdot \left(\tilde{X}_\tau(\mathbf{h}_2, \mathbf{s}'_2) - b_2\right) \cdot \left(\tilde{X}_\tau((\mathbf{h}_3|\mathbf{t}_3)_{j'}, \mathbf{s}'_3) - b_3\right) = 0 \ . \end{aligned}$$

Therefore, the sum in this part is 0.

References

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptology*, 30(4):989–1066, 2017.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 21–31, 1991.

[BHK16] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive RAM and batch NP delegation from any PIR. *IACR Cryptology ePrint Archive*, 2016:459, 2016.

[BHK17] Zvika Brakerski, Justin Holmgren, and Yael Tauman Kalai. Non-interactive delegation and batch NP verification from standard computational assumptions. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 474–482, 2017.

[BMW98] Ingrid Biehl, Bernd Meyer, and Susanne Wetzel. Ensuring the integrity of agent-based computations by short proofs. In *Mobile Agents, Second International Workshop, MA'98, Stuttgart, Germany, September 1998, Proceedings*, pages 183–194, 1998.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.

[DHRW16] Yevgeniy Dodis, Shai Halevi, Ron D. Rothblum, and Daniel Wichs. Spooky encryption and its applications. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 93–122, 2016.

[DLN+00] Cynthia Dwork, Michael Langberg, Moni Naor, Kobbi Nissim, and Omer Reingold. Succinct proofs for NP ander spooky interactions. manuscript, `http://www.wisdom.weizmann.ac.il/\~naor/PAPERS/spooky.pdf`, 2000.

[DNR16] Cynthia Dwork, Moni Naor, and Guy N. Rothblum. Spooky interaction and its discontents: Compilers for succinct two-message argument systems. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 123–145, 2016.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178, 2009.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4):205–214, 1998.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92, 2013.

[HR18] Justin Holmgren and Ron Rothblum. Delegating computations with (almost) minimal time and space overhead. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 124–135, 2018.

[KPY19] Yael Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC*, 2019.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. Delegation for bounded space. In *STOC*, pages 565–574, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: the power of no-signaling proofs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 485–494, 2014.

[PR17] Omer Paneth and Guy N. Rothblum. On zero-testable homomorphic encryption and publicly verifiable non-interactive arguments. In *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, pages 283–315, 2017.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, pages 1–18, 2008.