# Compact Proofs of Partial Knowledge for Overlapping CNF Formulae

Gennaro Avitabile[*1], Vincenzo Botta[†2], Daniele Friolo[‡2], Daniele Venturi[§2], and Ivan Visconti[¶2]

[1]IMDEA Software Institute, Madrid, Spain
[2]Sapienza University of Rome, Rome, Italy

**Abstract**

At CRYPTO '94, Cramer, Damgård, and Schoenmakers introduced a general technique for constructing honest-verifier zero-knowledge proofs of partial knowledge (PPK), where a prover Alice wants to prove to a verifier Bob she knows $\tau$ witnesses for $\tau$ claims out of $k$ claims without revealing the indices of those $\tau$ claims. Their solution starts from a base honest-verifier zero-knowledge proof of knowledge $\Sigma$ and requires to run in parallel $k$ execution of the base protocol, giving a complexity of $O(k\gamma(\Sigma))$, where $\gamma(\Sigma)$ is the communication complexity of the base protocol. However, modern practical scenarios require communication-efficient zero-knowledge proofs tailored to handle partial knowledge in specific application-dependent formats.

In this paper we propose a technique to compose a large class of $\Sigma$-protocols for atomic statements into $\Sigma$-protocols for PPK over formulae in conjunctive normal form (CNF) that overlap, in the sense that there is a common subset of literals among all clauses of the formula. In such formulae, the statement is expressed as a conjunction of $m$ clauses, each of which consists of a disjunction of $k$ literals (i.e., each literal is an atomic statement) and $\ell$ literals are shared among clauses. The prover, for a threshold parameter $\tau \leq k$, proves knowledge of at least $\tau$ witnesses for $\tau$ distinct literals in each clause.

At the core of our protocol, there is a new technique to compose $\Sigma$-protocols for regular CNF relations (i.e., when $\tau = 1$) that exploits the overlap among clauses and that we then generalize to formulae where $\tau > 1$ providing improvements over state-of-the-art constructions.

## 1 Introduction

Interactive proof systems [20] allow a prover to convince a verifier that a given instance $x$ belongs to an NP language $\mathcal{L}$. Besides completeness (namely, the honest prover always convinces the honest verifier when the statement is true), interactive proof systems typically enjoy two other properties known as *soundness* and *zero knowledge*. The former means that a malicious prover cannot convince the honest verifier about a claim $x$ when $x \notin \mathcal{L}$; the latter means that a malicious verifier talking to an honest prover learns nothing from the interaction (beyond $x \in \mathcal{L}$). The last few years have seen tremendous progress in the design of (sometimes even non-interactive) proof systems for languages of practical interest, with good efficiency in terms of round, communication, and computational complexity (see, e.g., [21, 9, 8, 10]). As a result, proof systems became a fundamental building block of many modern cryptographic applications, including electronic voting, cryptocurrencies, threshold cryptography, and distributed ledger technologies.

[*]gennaro.avitabile@imdea.org
[†]botta@di.uniroma1.it. The majority of the work was carried out whilst the author was working at the University of Warsaw, Warsaw, Poland.
[‡]friolo@di.uniroma1.it
[§]venturi@di.uniroma1.it
[¶]ivan.visconti@uniroma1.it. Work done while the author was affiliated with the University of Salerno, Italy.

**$\Sigma$-protocols.** $\Sigma$-protocols, introduced in Cramer's PhD thesis [13], are the central abstraction at the heart of several widely used zero-knowledge proofs. A $\Sigma$-protocol is a 3-round, public-coin,[1] interactive proof system in which the transcript has the form $(a, c, z)$, where $a$ and $z$ are called the prover's first-round and third-round messages, and where $c$ is the verifier's random challenge.

$\Sigma$-protocols typically satisfy two security properties known as *special soundness* and *honest-verifier zero knowledge* (HVZK). The former says that given two accepting transcripts for a statement $x$ of the form $(a, c, z)$ and $(a, c', z')$ with $c \neq c'$, one can efficiently extract a witness $w$ corresponding to the commonly known instance $x$; this implies that $\Sigma$-protocols are also *proofs of knowledge* [7] when the length of the challenge is large enough. The latter says that the protocol is zero knowledge assuming the verifier is benign and therefore picks the challenge $c$ from her random tape. Using the Fiat-Shamir heuristic [16], $\Sigma$-protocols can be made non-interactive enjoying zero-knowledge proofs of knowledge, therefore providing zero knowledge also against malicious verifiers, in the random oracle model. This is done by computing the challenge as the hash of the prover's first-round message concatenated to the statement and allowing the simulator and the extractor to program the random oracle.

**Composition of $\Sigma$-protocols.** Given $\Sigma$-protocols for different atomic statements, it is natural to ask whether they can be efficiently composed to prove compound NP statements. More in detail, given $n \in \mathbb{N}$ instances $x_1, \ldots, x_n$ belonging to a given[2] language $\mathcal{L}$, and given individual $\Sigma$-protocols for proving $x_i \in \mathcal{L}$, there are many results in literature to design a $\Sigma$-protocol for a compound relation $\mathcal{R}^*$ somewhat preserving efficiency. $\mathcal{R}^*$ can be expressed in different forms, and we consider two major ones in practical scenarios:

- **Disjunctive Normal Form ($(k, m)$-DNF):** Here, for parameters $k, m \in \mathbb{N}$, $(k, m)$-DNF is a disjunction (i.e., logical $\vee$) of $m$ clauses, where each clause is made of the conjunction (i.e., logical $\wedge$) of $k$ literals. For instance, by setting $k = 1$ and $m = n$, the resulting $(1, n)$-DNF represents the special case of a disjunction, in which the prover wants to convince the verifier it knows one out of $n$ witnesses. $\Sigma$-protocols for $(k, m)$-DNF relations whenever $m = \binom{n}{k}$ are often referred to as $k$-out-of-$n$ *proofs of partial knowledge* ($(k, n)$-PPK), where the prover proves to the verifier she holds $k$ witnesses, each of them corresponding to a different instance out of the $n$ possible instances.

- **Conjunctive Normal Form ($(k, m)$-CNF):** Here, for parameters $k, m \in \mathbb{N}$, the $(k, m)$-CNF is the conjunction (i.e., logical $\wedge$) of $m$ clauses, where each clause is made of the disjunction (i.e., logical $\vee$) of $k$ literals. CNF relations occur naturally in many applications, for instance when describing the access policies of certain attribute-based encryption schemes [27].

Which of the above representations is preferable depends on the application. Some policies are represented more succinctly using DNFs, whereas others using CNFs. Switching between these representations can involve an exponential increase in size [26].

Several prior works studied $\Sigma$-protocols composition for both DNF and CNF relations. The seminal work of Cramer *et al.* [14] initiated the study of $\Sigma$-protocols composition, providing a general solution that works for all monotone access structures. A particular case of those policies is disjunctions, which are a great tool to build new cryptographic primitives and protocols (see, e.g., [25, 15]) and have been the focus of many subsequent works [3, 22, 11, 18]. With the increasing interest in distributed systems (like blockchains), threshold variants of the disjunction composition (i.e., proofs of partial knowledge) became also very relevant [12, 4, 5].

## 1.1 Our Contribution and Related Work

Several recent papers focus on optimizing the composition of $\Sigma$-protocols for DNF and CNF relations. The goal of such works is to reduce the communication complexity by exploiting literals that are repeated across the clauses [1, 2, 29]. In this vein, we initiate, somewhat naturally, the study of threshold variants of $\Sigma$-protocols for CNF relations. In particular, we associate a proof of partial knowledge to each clause of the

[1] This refers to the fact that the verifier during the protocol sends random bits only.
[2] More generally, the statements could even belong to different languages.

CNF: the prover proves knowledge of $\tau$ different witnesses for each clause of the $(k, m)$-CNF relation. We name the resulting relation a $(\tau, k, m)$-CNF relation.[3]

We focus in particular on the case of *overlapping* CNF formulae in the sense that clauses of the formula share some literals. Our main contribution is a novel technique to compose $\Sigma$-protocols for overlapping $(\tau, k, m)$-CNF relations. In particular, our composition technique starts from any so-called *stackable* $\Sigma$-protocol[4] [18] for the underlying base instances, with communication complexity $\gamma(\Sigma)$, and produces a stackable $\Sigma$-protocol for the $(\tau, k, m)$-CNF relation with communication complexity

$$O(\tau \cdot (\log k + m\gamma(\Sigma) + \max\{m(\log(k-\ell)), \log(\tfrac{k}{k-\ell})\}))$$

whenever there is a common prefix of size $\ell \in [0, k-1]$, among the $m$ clauses. We use the word prefix solely to simplify the presentation. Observe that the order of literals within a clause is not important. Indeed, common literals among the clauses can be easily rearranged to become a common prefix without changing the meaning of the statement being proven.

In Tab. 1 and Tab. 2, we compare our results to prior $(1, k, m)$-CNF and $(\tau, k, m)$-CNF (with $\tau > 1$), respectively. The comparison considers communication complexity, supported languages, and whether the protocol resulting from the composition is a $\Sigma$-protocol or not. As $(\tau, k, m)$-CNF relations are not directly captured by some of the previous works, for such protocols we start from the natural approach of repeating a $\tau$-out-of-$k$ proof of partial knowledge $m$ times in parallel. Some works can directly handle $(\tau, k, m)$-CNF relations. For example, [14] handles $(\tau, k, m)$-CNF relations with a communication complexity of $O(m \cdot k \cdot \gamma(\Sigma))$ which is the same of the case of $\tau = 1$. However, the $m$-fold parallel repetition of [17, Section 9] or [5] outperform such protocols [14, 1] whenever $\tau = o(k/\log k)$. Therefore, starting with such communication-efficient proofs[5] (i.e., [17, Section 9] and [5]) prior works yield a complexity of $O(\tau \cdot m \cdot (\gamma(\Sigma) + \log k))$, which is worse than ours[6] for every $\ell \geq k - \log k$ and $m \geq \log k$. Furthermore, it is unclear whether one can generalize the result for $(k, m)$-CNF relations of Zeng *et al.* [29] to the setting of $(\tau, k, m)$-CNF relations.

A building block of our construction is a new technique to compose $\Sigma$-protocols for $(k, m)$-CNF relations (i.e., when $\tau = 1$) that we call $\Pi^{\mathsf{cnf}}$. $\Pi^{\mathsf{cnf}}$ is a component of our protocol for $(\tau, k, m)$-CNF relations that we call $\Pi^{\mathsf{cnf}}_{\tau,k}$. When $\Pi^{\mathsf{cnf}}$ is repeated $\tau$ times along with our ordering protocol $\Pi^{\mathsf{ord\text{-}cnf}}$ (built upon Avitabile *et al.* [5]), we obtain $\Pi^{\mathsf{cnf}}_{\tau,k}$. $\Pi^{\mathsf{ord\text{-}cnf}}$ is used to ensure that every run of $\Pi^{\mathsf{cnf}}$ within $\Pi^{\mathsf{cnf}}_{\tau,k}$ uses a *different* witness. Our composition technique for $(k, m)$-CNF relations yields a communication complexity of

$$O(\log k + m \cdot (\gamma(\Sigma) + \log(k-\ell))).$$

In contrast, the best previous protocols for the same setting [2, 29] had a communication complexity of $O(\gamma(\Sigma) \cdot (\ell + m \cdot (k-\ell)))$.[7] In particular, the works of Abe *et al.* [2] and Zeng *et al.* [29] follow the same approach (from which we deviate) of converting the CNF formula into a graph to then run a $\Sigma$-protocol over the resulting graph. Contrary to [2], the resulting protocol of [29] is a $\Sigma$-protocol.

It turns out that the landscape of $\Sigma$-protocols for CNF relations is rather complex and identifying the most communication-efficient solution depends on the application (i.e., the format of the statement). For example, the previous protocols designed to take advantage of repeating literals in the clauses [1, 2, 29] have a communication complexity generally worse than parallel repetition of communication-efficient protocols for disjunctions [4, 18] whenever the clauses share a prefix of few to no literals among each other. On the other hand, savings in [1, 2, 29] in terms of communication complexity are considerable whenever more literals are shared among the clauses.

[3]Our technique can handle different threshold values $\tau_i$, with $i \in [m]$, for each clause. The modified protocol does not incur in any additional overhead. To simplify the description, we will focus on the case of a single threshold value $\tau$ for all the clauses.

[4]Stackable $\Sigma$-protocols are $\Sigma$-protocols that in addition must satisfy some other properties (see Sec. 3.3). Concretely, this is not a significant restriction since all the practically relevant $\Sigma$-protocols are also stackable.

[5]In this comparison, we are only considering $\Sigma$-protocols composition techniques. We are not considering the use of generic succinct proof systems (e.g., SNARKs or STARKs), of which we discuss the limitations later. Furthermore, this comparison assumes that all $\Sigma$-protocols of interest are also stackable (as it is currently known). We are also not considering composition techniques tailored for specific languages (e.g., the discrete log setting considered in [4]).

[6]For the sake of simplicity, we omit the security parameter $\lambda$ in the communication complexities. In particular, the complexity of [17, 5] is $O(\tau \cdot m \cdot (\gamma(\Sigma) + \lambda \log k))$, and $\gamma(\Sigma)$ is dominated by $O(\lambda \log k)$ for sufficiently large $k$.

[7]This analysis assumes that there are no other repeating literals except the ones in the prefix.

| Reference | Language | Σ-protocol? | Communication ($\tau = 1$) |
|---|---|---|---|
| Cramer *et al.* [14] | Any $\Sigma$ | ✓ | $O(m \cdot k \cdot \gamma(\Sigma))$ |
| Groth *et al.* [22] | DL | ✗[¶] | $O(m \cdot \log k) \cdot (\|\mathbb{G}\| + \|\mathbb{Z}_p^*\|)$ |
| Abe *et al.* [1] | Any $\Sigma$ | ✗[†] | $O((m+n) \cdot \gamma(\Sigma))$ |
| Abe *et al.* [2] | Any $\Sigma$ | ✗[†] | $O(\gamma(\Sigma) \cdot (n))$ |
| Attema *et al.* [4] | DL | ✗[‡] | $O(\log(m \cdot k)) \cdot \|\mathbb{G}\|$ |
| Goel *et al.* [18] | Stackable $\Sigma$ | ✓ | $O(m \cdot (\gamma(\Sigma) + \log k))$ |
| Avitabile *et al.* [5] | Stackable $\Sigma$ | ✓ | $O(m \cdot (\gamma(\Sigma) + \log k))$ |
| Zeng *et al.* [29] | Chameleon $\Sigma$ | ✓ | $O(\gamma(\Sigma) \cdot \|V\|)$ [♮] |
| Ours | Stackable $\Sigma$ | ✓ | $O(\log k + m \cdot (\gamma(\Sigma) + \log(k - \ell)))$ |

Table 1: Comparison of techniques for composing Σ-protocols for $(\tau = 1, k, m)$-CNF relations. $\gamma(\Sigma)$ denotes the communication complexity of the base Σ-protocol, $n$ is the number of distinct literals in the formula, $m$ is the number of clauses, $k$ is the length of each clause, and $\ell$ is the length of the common prefix across all clauses. If the protocol is tailored to the discrete log setting, we write the communication complexity as the number of field and group elements sent by the prover. Whenever repeating literals are not considered [14, 22, 4, 17, 18, 5], we report the complexity of repeating its disjunction protocol $m$ times in parallel. In the third column, we use ✓ if the protocol resulting from the composition is an interactive 3-round public coin protocol with (computational) special soundness, while we use ✗ otherwise. [¶] [22] is not special sound. [†] [1, 2] require converting the Σ-protocol in a non-interactive protocol with the aid of a (non-programmable) random oracle. [‡] [4] requires a logarithmic number of rounds. [♮] The complexity of [29] depends on the number of vertices $V$ of the graph defined in their paper.

| Reference | Communication ($\tau > 1$) |
|---|---|
| Cramer *et al.* [14] | $O(m \cdot k \cdot \gamma(\Sigma))$ |
| Groth *et al.* [22] | ✗ |
| Abe *et al.* [1] | $O((m+n) \cdot \gamma(\Sigma))$ |
| Abe *et al.* [2] | $\gamma(\Sigma)\|\Gamma\|$[¶] |
| Attema *et al.* [4] | $O(\log(m \cdot k)) \cdot \|\mathbb{G}\|$ |
| Goel *et al.* Sec. 9 of [17] | $O(\tau \cdot m \cdot (\gamma(\Sigma) + \log k))$ |
| Avitabile *et al.* [5] | $O(\tau \cdot m \cdot (\gamma(\Sigma) + \log k))$ |
| Zeng *et al.* [29] | ✗ |
| Ours | $O(\tau \cdot (\log k + m\gamma(\Sigma) + \max\{m(\log(k-\ell)), \log(\frac{k}{k-\ell})\}))$ |

Table 2: Comparison of techniques for composing Σ-protocols for $(\tau > 1, k, m)$-CNF relations. If a work does not take into account repeating literals, we consider the complexity of repeating its $\tau$-out-of-$k$ proof of partial knowledge $m$ times. In the last column, ✗ denotes that there is no known black-box way to get a protocol for $(\tau > 1, k, m)$-CNF relations from their protocol for $\tau = 1$. [¶] We denote by $|\Gamma|$ the size of the acyclicity program $\Gamma$ representing the $(\tau, k, m)$-CNF relation, see [2] for details.

As an additional example, consider the case where $\ell = k - 1$, $m = \frac{k}{\log\log k}$, and there are no repeating literals except the ones in the prefix. Parallel repetition of the protocol of [18] leads to a communication complexity of roughly $O(\frac{k}{\log\log k}(\gamma(\Sigma)+\log k))$, while the protocols of [2, 29] have a communication complexity of $O(k \cdot \gamma(\Sigma))$. Interestingly, in this case (and many others), even though our technique is built upon the techniques of [18], we improve over the state-of-the-art protocols for CNF relations [2, 29] even when simple parallel repetition of [18] is outperformed by such protocols. Indeed, the communication complexity of our protocol when $\ell = k - 1$ and $m = \frac{k}{\log\log k}$ is $O(\frac{k}{\log\log k} \cdot \gamma(\Sigma))$. Beyond the specific case of $\ell = k - 1$, our technique outperforms the communication complexity of [2, 29, 18] for all $\ell \geq k - \log k$, when $m$ is in $O(\frac{k}{\log\log k})$. In conclusion, while our protocol for $(k, m)$-CNF is always superior (or equivalent) to parallel repetition of [18], there exist specific parameters for which [1, 2, 29] perform better than our protocol. Since no single solution outperforms the others in all possible scenarios, we find that it is more viable to optimize specific configurations of parameters, so that protocol designers can pick the constructions that better suit a particular application.

**Applications.** All the applications mentioned in prior works about $(k, m)$-CNF [1, 2, 29] directly carry over to our setting as we consider the same class of statements. Additionally, we are the first to provide direct support for threshold variants of those statements.

Nevertheless, one might wonder whether there exist application scenarios that naturally involve multiple clauses sharing several literals. We now recall an example of such a scenario by describing an application proposed in [2], namely proofs of possession of white money. Consider a cryptocurrency with a transaction graph whose nodes represent public keys, and whose edges represent money flows. Proofs of possession of white money are used to certify the integrity of a party. By integrity we mean that the money owned by a party was received from a whitelisted organization (e.g., an accredited institution). Proofs of possession of white money can be used to keep the organization's identity pseudo-anonymous by not disclosing the nodes in the network controlled by the organization. The organizations can certify the user's integrity by proving knowledge of valid credentials associated with nodes that form a cut from the cryptocurrency genesis node to the node of interest, without revealing any information about the actual cut. More concretely, every distinct path that reaches the node of interest can be modeled as a clause of a CNF, where the literals of the clause are the public keys corresponding to each of the nodes in the path, and a witness is one of the secret keys. If accredited institutions give valid proof for such a CNF, then all the money owned by the final node has, at some point, gone through a whitelisted organization. At the same time, such proof does not directly reveal the public keys controlled by the proof-giving entity. Note that these types of claims involve the entire history of the coins arriving at the final node, therefore the clauses of the CNF will likely be huge, and since many users make transactions with the same accredited entities (e.g., payments hubs, banks) it is likely to have very long prefixes as well. In this scenario, our technique for CNF relations (i.e., for $\tau = 1$) comes in handy. Additionally, our threshold techniques (i.e., for $\tau > 1$) also allow us to prove, again preserving anonymity, that in each path, the money has gone through whitelisted organizations at least $\tau$ times. These proofs can be made non-interactive and thus publicly verifiable using the Fiat-Shamir (FS) transform. Moreover, through classic transforms (e.g., carefully appending a message to the input of a cryptographic hash function when implementing the FS transform), one directly gets a ring signature for arbitrary monotone access policies.

**Other related work.** Previous $\Sigma$-protocols compositions had also other goals (in contrast to just focusing on optimizing the communication complexity) such as enhancing the security properties [11, 12], or speeding up the proving time [19]. In particular, [11] and [12] deal with delayed-instance specifications in disjunctions and proofs of partial knowledge respectively. In [19], a stacking compiler that can be applied to constant-round (succinct) argument systems is proposed. The goal of [19] is to obtain a faster prover exploiting the fact that the stacked prover executes the simulator on all the false statements. In some argument systems, the simulator is much faster than the prover and thus such technique in beneficial. Unfortunately, this is in general not the case for $\Sigma$-protocols.

Other works consider disjunctions and proofs of partial knowledge for binary or arithmetic circuits. Heath *et al.* [24] extended a previous result based on stacked garbled circuits [23] to proofs of partial knowledge while maintaining the same communication advantage of [23]. Baum *et al.* [6] proposed an interactive commit-and-prove zero-knowledge proof system for arithmetic circuits based on Vector Oblivious Linear Evaluation (VOLE), called Mac'n'Cheese [6]. Both approaches have a communication complexity proportional to $\tau$ times the longest circuit (or branch), and [6] has an additive term which is logarithmic in $k$. More recently, Yang et al. [28] also proposed a VOLE-based protocol for disjunctions called Robin with improved communication complexity w.r.t. Mac'n'Cheese. Notice that [6] and [28] are inherently private coin, therefore they are not $\Sigma$-protocols.

Alternatively, one could employ succinct arguments such as STARKs [8] or SNARKs [21, 9, 10]. On the other hand, these techniques have limitations on their own, such as demanding extensive computational efforts and memory requirements from the prover and relying on strong assumptions or problematic trusted setups. Moreover, although all the mentioned approaches can be applied to NP-complete languages, their adaptation to arbitrary languages is not a straightforward task. In comparison, $\Sigma$-protocol composition has specific advantages for protocol designers. If there exists a $\Sigma$-protocol for the base language, the designer can directly use the composition technique without resorting to potentially costly NP reductions.

## 2 Technical Overview

We start from the observation that a $\Sigma$-protocol for the conjunction of $m$ statements can be obtained by running in parallel a $\Sigma$-protocol for each of the underlying statements. Hence, given a $\Sigma$-protocol for the disjunction of $k$ literals, we immediately obtain a $\Sigma$-protocol for any $(k, m)$-CNF relation, with communication complexity $m$ times the communication complexity of the underlying $\Sigma$-protocol for the disjunction of $k$ literals. However, this approach treats all the disjunctions as if they were independent of each other, neglecting any communication saving that could result from their common structure. The main idea behind our result is that we can reduce the proof size for $(k, m)$-CNF relation whenever the disjunctions share a common prefix; in particular, we build upon the disjunction composition of [18] to reduce the proof size.

In a similar spirit to [5], we exploit the structure of the parameters of the commitment scheme used in [18]. In what follows, we denote the disjunction protocol of [18] as $\Pi_{\mathsf{OR}}$, and we denote our protocol for $(k, m)$-CNF relations as $\Pi^{\mathsf{cnf}}$. Finally, to get our protocol $\Pi^{\mathsf{cnf}}_{\tau,k}$ for $(\tau, k, m)$-CNF relations, we repeat $\Pi^{\mathsf{cnf}}$ $\tau$ times along with our ordering protocol $\Pi^{\mathsf{ord\text{-}cnf}}$ that we build upon $\Pi^{\mathsf{ord}}$ of [5]. To give a more detailed description of our techniques, we first give a high-level overview of the protocol $\Pi_{\mathsf{OR}}$ presented in [18]. Then, we show how to stack $m$ executions of $\Pi_{\mathsf{OR}}$ to obtain our $\Sigma$-protocol for any $(k, m)$-CNF relation ($\Pi^{\mathsf{cnf}}$) achieving considerable communication savings when the clauses share a common prefix. Finally, we show how to combine our protocol for $(k, m)$-CNF relations with our ordering protocol $\Pi^{\mathsf{ord\text{-}cnf}}$ to get our protocol for $(\tau, k, m)$-CNF relations ($\Pi^{\mathsf{cnf}}_{\tau,k}$).

### 2.1 High-Level Overview of $\Pi_{\mathsf{OR}}$

We first describe the main ingredients of $\Pi_{\mathsf{OR}}$, namely stackable $\Sigma$-protocols (Sec. 3.3) and 1-out-of-2 equivocal commitments (Sec. 3.1). We then show how these tools are combined in [18] to get a communication-efficient $\Sigma$-protocol for disjunctions.

**Stackable $\Sigma$-protocols.** A stackable $\Sigma$-protocol has two non-standard properties called (i) Extended HVZK (EHVZK), and (ii) recyclable third-round messages. Property (i) requires that the simulator, given a third-round message $z$ as additional input, along with statement $x$ and challenge $c$, is able to output a first-round message $a$ such that $(a, c, z)$ is an accepting transcript for $x \in \mathcal{L}$; moreover, the simulator is *deterministic*. Property (ii) requires that, given a fixed challenge $c$, the distributions of all the possible third-round messages for every pair of instances in the language are indistinguishable. As shown in [18], many natural and relevant $\Sigma$-protocols are already stackable. Moreover, any HVZK $\Sigma$-protocol can be converted into an EHVZK one. Formal definitions of the properties required by (stackable) $\Sigma$-protocols can be found in Sec. 3.2, Sec. 3.3. In particular, they consider a slightly weaker notion of special soundness, called computational special soundness ([5, App. A], also reported in Sec. 3.2), that says that whenever a PPT adversary outputs two accepting transcripts for a statement $x$ with the same first-round messages and different challenges, we are guaranteed to extract a valid witness for $x$ except with negligible probability.

**1-out-of-2 equivocal commitments.** 1-out-of-2 equivocal commitments allow a prover to commit to a pair of elements, in such a way that one of the two elements is binding, while the other one can be equivocated given a trapdoor. To do so, the sender generates a pair of commitment parameters $(p_0, p_1)$, where $p_0$ is used to commit to the element in the first position, while $p_1$ is used for the element in the other position. Without knowledge of a trapdoor for parameter $p_b$, with $b \in \{0, 1\}$, the message committed in position $b$ is binding. Conversely, knowing a trapdoor $\mathsf{td}_b$ for $p_b$, the prover can run an equivocation algorithm and open the commitment in position $b$ to an arbitrarily chosen message by generating an opening randomness which we call equivocation randomness. 1-out-of-2 equivocal commitments guarantee that the prover can know the trapdoor for one position at most. Additionally, such position remains hidden from the verifier, even after the commitment is opened. 1-out-of-2 equivocal commitments further provide an algorithm to commit to a pair of messages without using any trapdoor in a way that its output is indistinguishable from the one generated with the equivocation algorithm leveraging the trapdoor $\mathtt{td}_b$. Notice that the construction of [18] requires that the size of the equivocal commitment must be independent of the size of the committed value. A simple way to satisfy this requirement is to compress the committed values down to a constant size using a collision-resistant hash function.

**Description of $\Pi_{\mathsf{OR}}$.** Let us assume w.l.o.g. that $k$ is a power of two. We first show how to compose two executions of a stackable $\Sigma$-protocol $\Pi$ for a language $\mathcal{L}$, into a stackable $\Sigma$-protocol for the disjunction $(x_1 \in \mathcal{L}) \vee (x_2 \in \mathcal{L})$.[8] We call the resulting protocol $\Pi_{1,2}$. For simplicity, let us assume that the prover knows a witness $w_1$ for statement $x_1$; $\Pi_{1,2}$ works as follows:

- In the first round, the prover $\mathsf{P}_{1,2}$ computes the first-round message $a_1$ related to $x_1$ using witness $w_1$, and commits to it using a 1-out-of-2 equivocal commitment scheme having $a_1$ in the binding position, and the dummy value 0 in the other position. The resulting commitment is denoted as $\mathtt{com}$. The first-round message $a$ of the composed protocol $\Pi_{1,2}$ includes $\mathtt{com}$ and the commitment scheme parameters $(p_0, p_1)$.

- Upon receiving a challenge $c$ from the verifier $\mathsf{V}_{1,2}$, the prover $\mathsf{P}_{1,2}$ computes $z'$ using $w_1$, and equivocates the equivocal position of the commitment with a simulated message $a_2$ generated by running the EHVZK simulator of $\Pi$ with input $(x_2, c, z')$. The prover $\mathsf{P}_{1,2}$ sends $z'$ and the opening values of $\mathtt{com}$ to the verifier $\mathsf{V}_{1,2}$ as the third-round message $z$ of $\Pi_{1,2}$. The value $z$ also includes the commitment scheme parameters.

- Finally, the verifier $\mathsf{V}_{1,2}$ reconstructs $a_1$ and $a_2$ by running the EHVZK simulator of $\Pi$. The verifier also checks that both $(a_1, c, z')$ and $(a_2, c, z')$ are accepting transcripts of $\Pi$, and that $\mathtt{com}$ indeed opens to $a_1$ and $a_2$. $\mathsf{V}_{1,2}$ also checks the correctness of the commitment parameters and that the parameters sent in the first round are equal to the ones sent in the third round.

At first glance, it might seem superfluous to send the commitment parameters $(p_0, p_1)$ in both the first and the last round. However, including $(p_0, p_1)$ in the third-round message is essential to make the composed protocol $\Pi_{1,2}$ stackable, as the commitment parameters are necessary to deterministically compute the first-round message which contains a commitment. One might also think of removing the parameters from the first-round message, however, the security properties of 1-out-of-2 equivocal commitments do not rule out the possibility for a malicious prover to explain the same commitment with different pairs of parameters that can potentially allow him to equivocate the same commitment at different positions. This would make it difficult to prove the special soundness of $\Pi_{1,2}$ via a reduction to the special soundness of the underlying $\Sigma$-protocol.

Before proceeding with the description, let us first show why $\Pi_{1,2}$ is a stackable $\Sigma$-protocol. We recall that stackability requires the existence of an EHVZK simulator. The EHVZK simulator $\mathcal{S}_{1,2}^{\mathsf{EHVZK}}$ of $\Pi_{1,2}$ works as follows. The input of $\mathcal{S}_{1,2}^{\mathsf{EHVZK}}$ is $((x_1, x_2), c, z = (z', r, p_0, p_1))$. Let $\mathcal{S}$ be the EHVZK simulator of $\Pi$, then $\mathcal{S}_{1,2}^{\mathsf{EHVZK}}$ computes $a_0 = \mathcal{S}(x_1, c, z')$ and $a_1 = \mathcal{S}(x_2, c, z')$. The returned value is $a = (\mathtt{com}, p_0, p_1)$, where $\mathtt{com}$ is computed as the commitment of $(a_1, a_2)$ under randomness $r$ and parameters $p_0, p_1$.

Being $\Pi_{1,2}$ a stackable $\Sigma$-protocol, two executions of $\Pi_{1,2}$ can be composed in the same way to obtain a stackable $\Sigma$-protocol for disjunctions of the type $x_1 \vee x_2 \vee x_3 \vee x_4$. By recursion, this approach yields a stackable $\Sigma$-protocol for general $k \in \mathbb{N}$ that here we call $\Pi_{\mathsf{OR}}$. Its EHVZK simulator $\mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}$ simply consists in the recursive invocation of $\mathcal{S}_{1,2}^{\mathsf{EHVZK}}$.

Following Avitabile *et al.* [5], an execution of $\Pi_{\mathsf{OR}}$ can be represented using the so-called composition tree. This is a binary tree whose leaves represent the base statements $(x_1, \ldots, x_k)$ and the first-round messages $(a_1, \ldots, a_k)$ of the underlying $\Sigma$-protocol $\Pi$. Given two siblings nodes $(x_i, a_i)$ and $(x_j, a_j)$, the parent node

[8] In the remainder of the paper, we will simply write $x_1 \vee x_2$ for brevity.

of $(x_i, a_i)$ and $(x_j, a_j)$ is $(x_t, a_t)$, where the disjunctive statement $x_t$ and the first-round message $a_t$ are the ones resulting from the 1-out-of-2 composition explained above. Moreover, edges $((x_t, a_t), (x_i, a_i))$ and $((x_t, a_t), (x_j, a_j))$ ($(t, i)$ and $(t, j)$ for short) are labeled as follows: if the prover knows a witness for the instance $x_i$, then the edge $(t, i)$ is labeled with B to indicate that, in the commitment computed by the prover, the position where $x_i$ is used is binding. Similarly, the edge $(t, j)$ is labeled with T to indicate that the position where $x_j$ is used is equivocal. If the prover knows a witness for $x_j$ instead, then the opposite holds. An example of a composition tree is reported in Fig. 1. Therefore, we can assign a bit to each level of the tree as follows: if the left edge is labeled as B (and the right edge as T) we assign the bit 0 to that level, and we assign the bit 1 otherwise. Observe that the leaf for which the prover knows the witness uniquely determines the string composed of the bits assigned from the first to the last level. For example, if $k = 8$ and the prover knows the witness for the seventh instance, the corresponding string is 110 (i.e., 7 in binary). We recall that the third-round message $z$ of $\Pi_{1,2}$ also contains the commitment parameters and the commitment openings (i.e., the equivocation randomnesses) for the 1-out-of-2 equivocal commitment. Therefore, when $\Pi_{1,2}$ is stacked to get a disjunction protocol with four literals since the exact same $z$ is used for the two stacked executions of $\Pi_{1,2}$, the last level of the composition tree will share the *same* parameters and the *same* openings. This generally holds when recursively stacking protocols to get disjunctions of any length. As a result, the *same* commitment parameters and openings are shared across each level of the composition tree. Therefore, the tree for a clause of length $k$ has $\log k$ levels, which lead to $\log k$ pairs of commitment parameters and $\log k$ equivocation randomnesses (i.e., the commitment openings) that will be provided by the prover during the execution of $\Pi_{\mathsf{OR}}$.

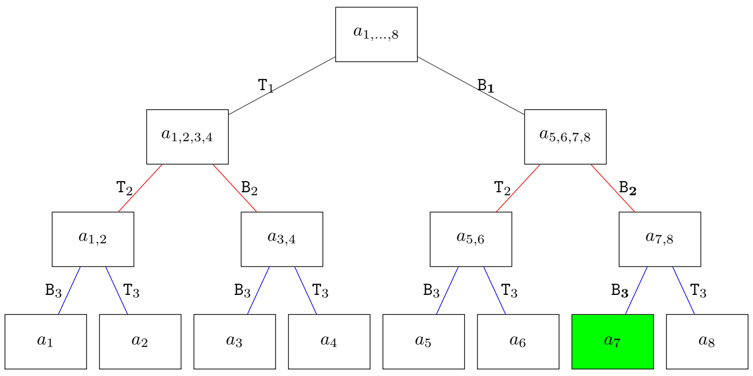

Figure 1: The composition tree induced by the recursive application of the stackable $\Sigma$-protocol $\Pi_{1,2}$ from [18], in which 8 base $\Sigma$-protocols are stacked to obtain a stackable $\Sigma$-protocol for a disjunction involving 8 literals. Here, the prover knows a witness for the instance $x_7$. This implies that going from the root to the leaves, the non-trapdoor edges must be arranged as depicted. Additionally, commitment openings and parameters are re-used across the same level of the composition tree. This is emphasized by using the same index and the same color for all the edges within a level.

**On the need of stackable $\Sigma$-protocols.** The use of an equivocal commitment to get a disjunction composition for $\Sigma$-protocols was already explored by [12] in the setting of delayed instance specification. Their technique is very similar to the one of [18] where the first-round message consists of a 1-out-of-$k$ binding commitment to the first-round messages of the base $\Sigma$-protocols, and in the third round all of the instances for which the prover does not have a witness for are equivocated with the values given in output by the simulator. Furthermore, delayed instance specification is achieved starting from delayed-input $\Sigma$-protocols (i.e., $\Sigma$-protocols where the prover does not have to know either the instance or the witness to compute the first-round message), and by randomly picking the binding position and assigning a random position to all the non-active instances received in the third round. The main difference with [18] is that [12] does not consider stackable $\Sigma$-protocols or recursive composition. Thus, even if their compiler basically coincides with the compiler of [18], the result of [12] does not afford any communication complexity savings when compared to classical techniques. This is because even if they implemented their commitment with the recursive approach of [18], it would not be possible to re-use the same third-round message for multiple instances, or the same commitment parameters and equivocation randomnesses for multiple levels of the composition tree.

## 2.2 Batching Commitment Parameters

We consider CNF formulas of the form:

$$\bigwedge_{i\in m}\left(\bigvee_{j\in\mathcal{J}}(x_j)\vee\bigvee_{u\in\mathcal{I}_i}(x_u)\right),$$

where $\mathcal{J}\subset[n]$ denotes the set of indexes in common between the different clauses (i.e., $|\mathcal{J}|<k$) and $\mathcal{I}_1,\ldots,\mathcal{I}_m\subset[n]\setminus\mathcal{J}$ represent the indexes that change among clauses. Notice that $|\mathcal{J}|+|\mathcal{I}_i|=k$, for all $i\in m$. If the prover has a witness for an instance $x_j$ with $j\in\mathcal{J}$, then we may w.l.o.g. assume that the prover uses this witness to satisfy all of the clauses, i.e. the formula is satisfied with the *minimal assignment*.

To illustrate our idea we now consider a simple CNF, but the technique is easy to generalize. Consider the following CNF with two clauses: $(x_1\vee x_2\vee x_3\vee x_4\vee x_5\vee x_6\vee x_7\vee x_8)\wedge(x_1\vee x_2\vee x_3\vee x_4\vee x_5\vee x_6\vee x_9\vee x_{10})$, where the instances $x_1$, $x_2$, $x_3$, $x_4$, $x_5$, $x_6$ are shared across the two clauses. We observe that the prover in $\Pi_{\mathsf{OR}}$ generates the parameters of each level solely based on the position of the leaf it has the witness for. Let us now consider two executions of $\Pi_{\mathsf{OR}}$ for the two above clauses. Starting from the root of the tree, if the prover has the witness for $x_5\vee x_6\vee x_7\vee x_8$, it would generate the pair of parameters for the first level having the right position as binding in both cases. On the other hand, if it holds a witness for $x_1\vee x_2\vee x_3\vee x_4$, it would generate the commitment parameters for the first level so that the left position is binding in both cases. Since the ability to equivocate a specific position (depending on the witness) is the only requirement to set a pair of parameters in the execution of $\Pi_{\mathsf{OR}}$, and since both executions have to equivocate the same position at the first level, we can use the same pair of parameters for the first level in both the executions. It is straightforward to notice that the same reasoning applies to the second level of the tree, indeed considering $x_5\vee x_6\vee x_7\vee x_8$ and $x_5\vee x_6\vee x_9\vee x_{10}$ the prover needs to equivocate either the right or the left position in both clauses. In this way, we can reduce the amount of information being sent to the verifier by re-using the same commitment parameters for the first two levels of both trees. However, even if this small modification produces a concrete improvement in the proof size, the improvement is only by a constant factor. Indeed, the prover still needs to send $\log k$ equivocation randomnesses per each clause of the CNF, as those values cannot be reused. This is because, even if in the first level of both trees the same position needs to be equivocated, the equivocations involve two different commitments and two different committed values. Therefore, the equivocation algorithm of the commitment scheme will generally produce different equivocation randomnesses.

## 2.3 Batching Equivocation Randomnesses

To get asymptotic communication savings, we need to find a way to send a single set of equivocation randomnesses for all the levels of the trees sharing the same parameters. To illustrate our technique for achieving this goal, let us consider the same CNF relation from the previous paragraph. W.l.o.g., we assume the prover holds a witness for $x_7$ and $x_{10}$. Let us consider the second clause. As in a regular execution of $\Pi_{\mathsf{OR}}$, the prover would first compute the first-round message $a_{9,10}$ related to $x_9\vee x_{10}$. Now, instead of committing to $a_{9,10}$ separately in a 1-out-of-2 equivocal commitment to create $a_{5,6,9,10}$ as usual, we merge the two composition trees into one by committing to the concatenation $a_{7,8}||a_{9,10}$ in the binding position of a single commitment. To be more specific, we now have a single commitment corresponding to the concatenation of two values in its binding position, instead of two separate commitments sharing the same parameters and committing to $a_{7,8}$ and $a_{9,10}$ in their own binding positions.

In Fig. 2, we report a graphical representation of the merged composition tree corresponding to the above example. More generally, let $\ell \geq k/2$ and $k - \ell$ be a power of 2, we can compute the number of levels of the composition trees that can share the same parameters as $d = \lfloor \log k - \log(k - \ell) \rfloor$. Given $d$, for all the clauses, the prover computes the corresponding 1-out-of-2 commitment for all the sub-trees at level $d$ as in a regular run of $\Pi_{\mathsf{OR}}$. Then, after having obtained all these intermediate sub-trees, the prover merges them in a single composition tree by concatenating all the sub-trees at level $d$ and then building all the nodes up until the root using a shared set of parameters that works for all clauses. Indeed if the prover has a witness for a literal that is in the prefix, we make it re-use such witness when proving all the clauses. Therefore, the first $d$ levels always need to be equivocated in the same way. We remark that to have any communication saving this technique requires the prefix to be $\ell \geq k/2$. The root commitment $a_{\mathsf{cnf}}$ will be sent to the verifier as part of the first-round message. The prover behave as follows:

- On input $(x_7, x_8, w_7)$ and $(x_9, x_{10}, w_{10})$, compute $a_{7,8}$ and $a_{9,10}$ by using $\mathsf{P}_{\mathsf{OR}}$ for statements $(x_7, x_8)$ and $(x_9, x_{10})$, witnesses $w_7$ and $w_{10}$ and parameters $(p_0^3, p_1^3)$ and $({p'}_0^3, {p'}_1^3)$ respectively.
- Compute an equivocal commitment $a_{5,6,7,8}$ with parameters $(p_0^2, p_1^2)$ where $a_{7,8}||a_{9,10}$ is in the binding position.
- Similarly, compute an equivocal commitment $a_{\mathsf{cnf}}$ with parameters $(p_0^1, p_1^1)$ where $a_{5,6,7,8}$ is in the binding position.
- When receiving the verifier's challenge, do the following (we implicitly assume that all the algorithms invoked from now on take the challenge as input):
  1. Compute the third-round messages $z_{7,8} = (z_7, r_3, p_0^3, p_1^3)$ and $z_{9,10} = (z_{10}, r'_3, {p'}_0^3, {p'}_1^3)$ of $\Pi_{\mathsf{OR}}$ for the statements $(x_7, x_8)$ and $(x_9, x_{10})$, where $z_7$, and $z_{10}$ are computed running the base prover of the underlying $\Sigma$-protocol.
  2. Compute $a_{5,6} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_5, x_6), c, z_{7,8})$ and $a'_{5,6} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_5, x_6), c, z_{9,10})$, where $\mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}$ is the EHVZK simulator of $\Pi_{\mathsf{OR}}$.
  3. Compute the equivocation randomness $r_2$ (using the trapdoor related to $p_0^2$) to open the equivocal position of $a_{5,6,7,8}$ to $a_{5,6}||a'_{5,6}$.
  4. Compute $a_{1,2} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_1, x_2), c, z_{7,8})$ and $a'_{1,2} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_1, x_2), c, z_{9,10})$.
  5. Repeat step 4 for $a_{3,4}$ and $a'_{3,4}$ where the statement used by $\mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}$ is $(x_3, x_4)$.
  6. Compute $a_{1,2,3,4}$ by using the committing algorithm with no trapdoor with inputs $a_{1,2}||a'_{1,2}$ and $a_{3,4}||a'_{3,4}$, parameters $(p_0^2, p_1^2)$ and randomness $r_2$.
  7. Generate the equivocation randomness $r_1$ (using the trapdoor related to $p_0^1$) to open the equivocal position of $a_{\mathsf{cnf}}$ to $a_{1,2,3,4}$.
- Send a value $z = (z_1, z_2)$ to the verifier where $z_1 = ((z_7, r_3, p_0^3, p_1^3), r_2, p_0^2, p_1^2), r_1, p_0^1, p_1^1))$ and $z_2 = (z_{10}, r'_3, {p'}_0^3, {p'}_1^3)$. Notice that $z_2$ contains less data than $z_1$ as we are able to re-use the parameters and the equivocation randomnesses from $z_2$.

Finally, the verifier behaves as follows (we implicitly assume that all the algorithms take the verifier's challenge as input):

1. Parse $z_1$ as $((z'_1, r_3, p_0^3, p_1^3), r_2, p_0^2, p_1^2), r_1, p_0^1, p_1^1))$ and $z_2$ as $(z'_2, r'_3, {p'}_0^3, {p'}_1^3)$.
2. Unpack the merged trees and verify them individually. Let $\tilde{z} = (z'_1, r_3, p_0^3, p_1^3)$ and $\tilde{z'} = (z'_2, r'_3, {p'}_0^3, {p'}_1^3)$:
   - Compute $\hat{a}_{1,2} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_1, x_2), c, \tilde{z})$ and $\hat{a}_{3,4} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_3, x_4), c, \tilde{z})$.
   - Compute $\hat{a'}_{1,2} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_1, x_2), c, \tilde{z'})$ and $\hat{a'}_{3,4} = \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}((x_3, x_4), c, \tilde{z'})$.

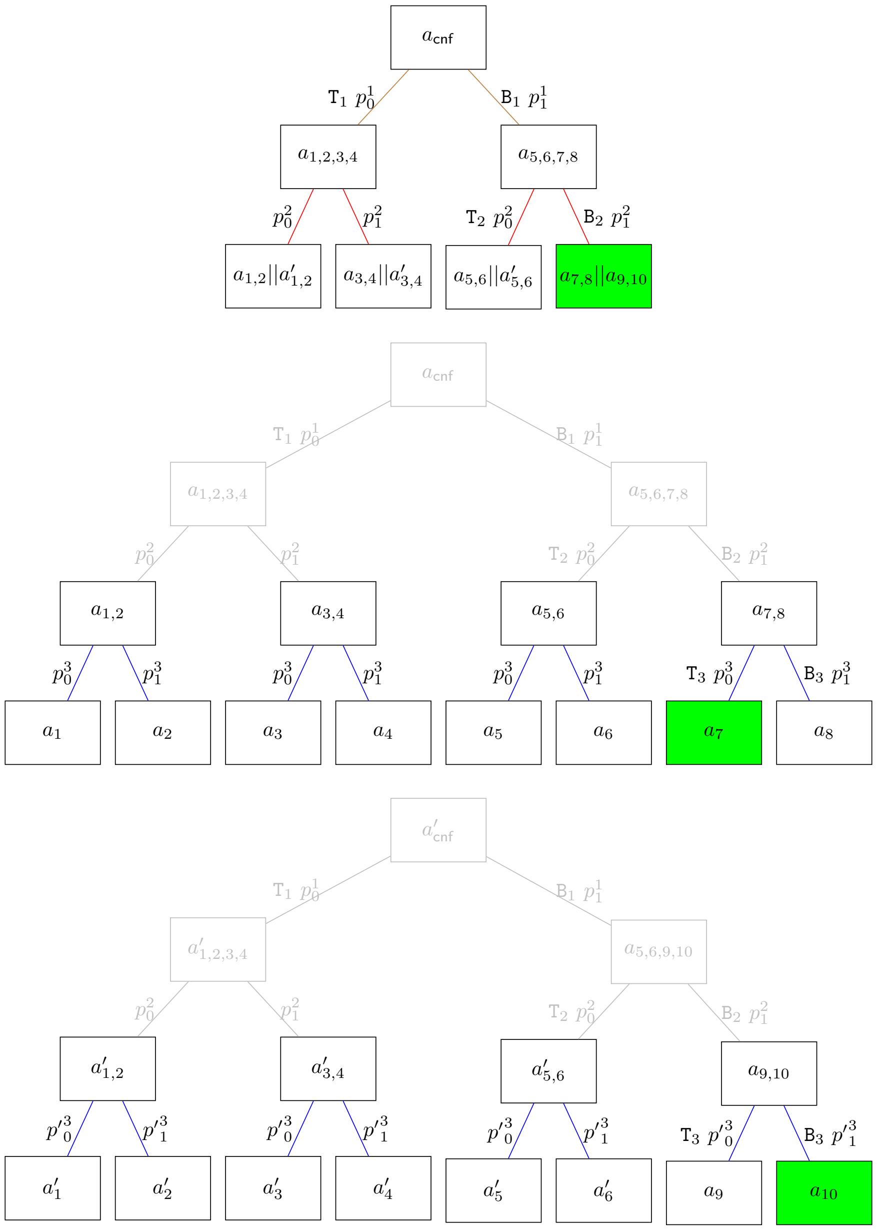

Figure 2: At the top of the figure, the merged composition tree resulting from two clauses of 8 literals sharing a prefix of 6 literals is represented. Below are the two composition trees of the two clauses, where the grey-shaded levels represent the levels that are not computed anymore since such commitment parameters are shareable across the two clauses, and thus such levels are computed only in the merged composition tree depicted on top. The solid nodes and edges represent the remaining part of the trees where the commitment parameters are *not* shared between each other. These sub-trees are indeed computed and then merged in the corresponding level of the merged composition tree depicted on top of the figure.

- Check that the verifier algorithm of $\Pi_{\mathsf{OR}}$ outputs 1 for all the unmerged trees, i.e., $\mathsf{V}_{\mathsf{OR}}((x_1, x_2), \hat{a}_{1,2}, c, \tilde{z}) = 1$, $\mathsf{V}_{\mathsf{OR}}((x_1, x_2), \hat{a'}_{1,2}, c, \tilde{z'}) = 1$, $\mathsf{V}_{\mathsf{OR}}((x_3, x_4), \hat{a}_{3,4}, c, \tilde{z}) = 1$, and $\mathsf{V}_{\mathsf{OR}}((x_3, x_4), \hat{a'}_{3,4}, c, \tilde{z'}) = 1$.

3. Verify correctness of merging:

   - Compute $a^*_{1,2,3,4}$ as a 1-out-of-2 commitment of $a_{1,2}||a'_{1,2}$ and $a_{3,4}||a'_{3,4}$ using parameters $(p^2_0, p^2_1)$ and randomness $r_2$ and check that $a^*_{1,2,3,4} = a_{1,2,3,4}$.

4. Repeat step 2 to 3 for the right sub-tree of $a_{\mathsf{cnf}}$, i.e. recompute $\hat{a}_{5,6} = \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}((x_5, x_6), c, \tilde{z})$, $\hat{a}_{7,8} = \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}((x_7, x_8), c, \tilde{z})$, $\hat{a'}_{5,6} = \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}((x_5, x_6), c, \tilde{z'})$, and $\hat{a}_{9,10} = \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}((x_9, x_{10}), c, \tilde{z'})$, and check that $\mathsf{V}_{\mathsf{OR}}((x_5, x_6), \hat{a}_{5,6}, c, \tilde{z}) = 1$, $\mathsf{V}_{\mathsf{OR}}((x_5, x_6), \hat{a'}_{5,6}, c, \tilde{z'}) = 1$, $\mathsf{V}_{\mathsf{OR}}((x_7, x_8), \hat{a}_{7,8}, c, \tilde{z}) = 1$ and $\mathsf{V}_{\mathsf{OR}}((x_9, x_{10}), \hat{a}_{9,10}, c, \tilde{z'}) = 1$. Then, recompute $a^*_{5,6,7,8}$ as the 1-out-of-2 commitment of $a_{5,6}||a'_{5,6}$ and $a_{7,8}||a_{9,10}$ using parameters $(p^2_0, p^2_1)$ and randomness $r_2$, and check that $a^*_{5,6,7,8} = a_{5,6,7,8}$.

5. Finally, compute $a^*_{\mathsf{cnf}}$ as a 1-out-of-2 commitment of $a_{1,2,3,4}$ and $a_{5,6,7,8}$ using parameters $(p^1_0, p^1_1)$ and randomness $r_1$ and check that $a^*_{\mathsf{cnf}} = a_{\mathsf{cnf}}$.

In a nutshell, our protocol merges all the levels that share the same commitment parameters and this allows to use a unique set of equivocation randomnesses for such levels. In the first round, the prover sends a single commitment encoding the 2 composition trees. Then, the verifier is able to reconstruct the composition trees and verify them using the verification algorithm of $\Pi_{\mathsf{OR}}$.

## 2.4 Our Σ-Protocol for $(\tau, k, m)$-CNF Relations

To design our Σ-protocol for $(\tau, k, m)$-CNF relations $\Pi^{\mathsf{cnf}}_{\tau,k}$, we adapt some techniques from [5] to our protocol for $(k, m)$-CNF relations from above. In [5], Avitabile *et al.* show how to compose the disjunction protocol of [18] to get a Σ-protocol for $(\tau, k, 1)$-CNF relations (i.e., a $\tau$-out-of-$k$ proof of partial knowledge). In a nutshell, they repeat $\tau$ times the protocol $\Pi_{\mathsf{OR}}$ for the same statement, with additional proof certifying that each execution uses a different witness. Their main observation is that the commitment parameters of an execution of $\Pi_{\mathsf{OR}}$ can be uniquely mapped to a binary string depending on which witness was used to generate the proof. Hence, they provide a communication-efficient stackable Σ-protocol to sort such strings in zero knowledge, proving that there is a certain ordering among the strings representing the commitment parameters while hiding the digits in which the strings differ (i.e., hiding which witnesses were used in the $\tau$ executions of $\Pi_{\mathsf{OR}}$). They call such protocol $\Pi^{\mathsf{ord}}$. In order to get a Σ-protocol for $(\tau, k, 1)$-CNF relations, they simply run $\Pi^{\mathsf{ord}}$ using the commitment parameters of each of the $\tau$ executions of $\Pi_{\mathsf{OR}}$ as a statement, and the trapdoors of every parameter pair as a witness.

Similarly to [5], we could repeat our $\Pi^{\mathsf{cnf}}$ for $\tau$ times, and then prove that a different witness was used for each clause in each of the $\tau$ executions of $\Pi^{\mathsf{cnf}}$. This latter proof would be done by running $m$ times $\Pi^{\mathsf{ord}}$ using the $\tau$ different *full* vectors of commitment parameters related to each of the $m$ clauses as the statements. To be more specific, we would need to consider the recycled parameters in *all* the vectors of commitment parameters related to clauses of the same execution of $\Pi^{\mathsf{cnf}}$. However, analogously to what happens when only batching commitment parameters (and not the equivocation randomnesses) in a $(k, m)$-CNF (Sec. 2.3), the benefit of having recycled some commitment parameters would not be enough to produce asymptotic communication savings w.r.t. the naive repetition of $m$ times of the $(\tau, k)$-PPK of [18, 5]. This is because the size of the ordering proof of [5] is linear in the size of the statement, and every ordering proof would involve vectors of commitment parameters of full length, just like in the solution of [5].

Notice that the naive approach of producing an ordering proof for the $\tau$ vectors of parameters of length $\log k$ related to the first clause, and $m - 1$ proofs for the $\tau$ vectors of parameters of length $\log(k - \ell)$ related to of the remaining $m - 1$ clauses of the CNF does not work. Indeed, the resulting protocol would not even be complete. For example, let us consider a CNF with two clauses $x_1, x_2$, with $k = 16$, $\ell = 12$, and $\tau = 2$. Let the binary string representing the composition tree w.r.t. $x_2$ in the first execution be 0000 (i.e., the prover has the witness for position 1), and the binary string w.r.t. $x_2$ in the second execution be 1100 (i.e., the prover has the witness for position 13). Since in this case $\Pi^{\mathsf{ord}}$ is only executed over the commitment parameters related to the suffix composed by $k-\ell$ literals (i.e., to the last $\log 4 = 2$ pair of parameters), $\Pi^{\mathsf{ord}}$ would have to prove that $00 < 00$, which is impossible. We overcome this problem by slightly changing how the ordering relation is proved for the remaining $m-1$ clauses. In particular, regarding the above example, let $\mathbf{p}_1$ and $\mathbf{p}_2$ be the parameters w.r.t. the prefix of $x_2$ in the first and second execution respectively, and $\mathbf{s}_1$ and $\mathbf{s}_2$ be the ones for the suffix. The prover would prove a modified ordering relation[9] of the form $\mathbf{s}_1 < \mathbf{s}_2 \vee \mathbf{p}_1 < \mathbf{p}_2$. This modified version of the ordering protocol allows the prover to produce a satisfying ordering proof by using the witness for the branch of the OR comparing the prefixes (i.e., proving that $00 < 11$). As detailed in Sec. 5, thanks to the efficiency of disjunctions, this modified version of the ordering protocol allows us to build a $(\tau, k, m)$-CNF retaining the efficiency gains of our $(k, m)$-CNF.

**On getting $(\tau, k, m)$-CNF from $(\tau, k)$-PPK.** We observe that it is possible to build a communication-efficient protocol for $(1, k, m)$-CNF relations from a $(1, k)$-PPK in a black-box way. Indeed, we can re-write the $(k, m)$-CNF into a single disjunction with more complex base statements:

$$\bigvee_{j \in \mathcal{J}} (x_j) \vee \left( \bigwedge_{i \in [m]} \left( \bigvee_{u \in \mathcal{I}_i} (x_u) \right) \right). \tag{1}$$

The latter formula is a disjunction of the type $\bigvee_{j\in\mathcal{J}}(x_j) \vee x'$ where $x' = \bigwedge_{i\in[m]}(\bigvee_{u\in\mathcal{I}_i}(x_u))$. Notice that literals repeating across the clauses appear in the modified formula only once. Hence, to get a protocol for the $(k, m)$-CNF relation, it suffices to prove the modified statement using a suitable disjunction protocol. The protocol resulting from applying this observation together with the disjunction of [18] already outperforms [1, 2, 29] for the same set of parameters discussed in Sec. 1.1.

Unfortunately, the above approach does not immediately carry over to the more interesting case of $(\tau, k, m)$-CNF with $\tau > 1$. In particular, it is unclear whether there are clever ways to take advantage of repeating literals while getting a protocol for $(\tau, k, m)$-CNF relations from a $(\tau, k)$-PPK in a black-box way for general values of $\tau$. Indeed, when considering the formula above $\bigvee_{j\in\mathcal{J}}(x_j) \vee x'$, we face the following issue when trying to obtain a proof for a $(\tau, k, m)$-CNF.

Recall that the idea of [5] is to generate for each clause $\tau$ different executions of the $(1, k)$-PPK and prove that a different witness is used to compute each of the $\tau$ PPKs. If we want to rely on a similar technique for the formula above, it is not clear how to cover the case in which the prover used two different witnesses inside $x'$. This problem arises from the fact that the composition tree of $\bigvee_{j\in\mathcal{J}}(x_j) \vee x'$ has the *same* configuration of parameters for any of the (many) witnesses for $x'$ that the prover may use. The same problem arises when using the $(\tau, k)$-PPK proposed in [18]. An idea would be to rewrite the formula in such a way that the issue described above can be circumvented. However, we did not find a way to rewrite a suitable formula to avoid any blow-up in the number of clauses. Therefore, we propose a novel approach that does not involve simply re-using known $\Sigma$-protocol composition techniques out of the box.

## 3 Preliminaries

Let $\mathbb{N}$ be the set of all natural numbers; for $n \in \mathbb{N}$, we write $[n]$ for the set $\{1, \ldots, n\}$ and $[m, n]$ for the set $\{m, m+1, \ldots, n\}$, where $n > m$.

Throughout this paper, we use the abbreviation PPT to denote probabilistic polynomial time. Given a PPT algorithm $\mathcal{A}$, let $\mathcal{A}(x)$ be the probability distribution of the output of $\mathcal{A}$ when run with $x$ as input. We use $\mathcal{A}(x; r)$, instead, to denote the output of $\mathcal{A}$ when run on input $x$ and coin tosses $r$. We denote with $\lambda \in \mathbb{N}$ the security parameter and with $\mathsf{poly}(\cdot)$ an arbitrary positive polynomial. Every algorithm takes as input the security parameter $\lambda$ (in unary, i.e. $1^\lambda$). When an algorithm takes more than one input, $1^\lambda$ is omitted. We say that a function $\nu : \mathbb{N} \to \mathbb{R}$ is negligible in the security parameter $\lambda \in \mathbb{N}$ if it vanishes faster than the

[9]Whenever we compare two lists of commitment parameters, we actually refer to the comparison of the associated binary strings.

inverse of any polynomial in $\lambda$, i.e. $\nu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$ for all positive polynomials $\mathsf{poly}(\lambda)$. We use $\leftarrow$ when the variable on the left side is assigned with the output value of the algorithm on the right side. Similarly, when using $\leftarrow\!\!\$$, we mean that the variable on the left side is assigned a value sampled randomly according to the distribution on the right side.

A *polynomial-time* relation $\mathcal{R}$ is a relation for which membership of $(x, w)$ in $\mathcal{R}$ can be decided in time polynomial in $|x|$. If $(x, w) \in \mathcal{R}$, then we say that $w$ is a *witness* for the *instance* $x$. A polynomial-time relation $\mathcal{R}$ is naturally associated with the NP language $\mathcal{L}$ and we denote it by writing $\mathcal{R}_{\mathcal{L}}$.

A distribution ensemble $\{X(\lambda)\}_{\lambda\in\mathbb{N}}$ is an infinite sequence of probability distributions, where a distribution $X(\lambda)$ is associated with each value of $\lambda \in \mathbb{N}$. We say that two distribution ensembles $\{X(\lambda)\}_{\lambda\in\mathbb{N}}$ and $\{Y(\lambda)\}_{\lambda\in\mathbb{N}}$ are *computationally indistinguishable* if for every PPT distinguisher $\mathcal{D}$, there exists a negligible function $\nu$ such that:

$$\mathsf{Pr}\left[\, \mathcal{D}(1^\lambda, X(\lambda)) = 1 \,\right] - \mathsf{Pr}\left[\, \mathcal{D}(1^\lambda, Y(\lambda)) = 1 \,\right] \leq \nu(\lambda).$$

$\{X(\lambda)\}_{\lambda\in\mathbb{N}}$ and $\{Y(\lambda)\}_{\lambda\in\mathbb{N}}$ are statistically indistinguishable if the above holds for computationally unbounded $\mathcal{D}$. We use $\equiv$ to denote that two distribution ensembles are identical.

Throughout the paper, we refer to vectors using boldface. Moreover, we use $\mathbf{v}[i]$ to denote the $i$-th position of $\mathbf{v}$, $\mathbf{v}[i, \ldots, j]$ to denote the subvector of $\mathbf{v}$ from position $i$ to position $j$, $\mathbf{v}.\mathit{append}(x)$ to denote that we are appending $x$ as last element of $\mathbf{v}$, $\mathbf{v}.\mathit{length}$ to denote the function that returns the number of elements in $\mathbf{v}$. The first position of the array is indexed with 1. We use the symbol "$||$" to denote the concatenation of binary strings. In some cases, we use the ternary operator (? :) to denote the conditional expression in the algorithms (i.e., we write "condition?evaluated-when-true : evaluated-when-false").

## 3.1 1-out-of-2 Equivocal Commitments

A 1-out-of-2 equivocal commitment scheme is composed of a tuple of PPT algorithms $\mathcal{CS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{BindCom}, \mathsf{EquivCom}, \mathsf{Equiv})$, along with a polynomial-time relation $\mathcal{R}$, specified as follows:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda; r)$: upon input the security parameter, and randomness $r$, generates public parameters $\mathsf{pp}$; we denote by $\mathcal{Y}^{\mathsf{pp}}$ the space of well-formed commitment parameters w.r.t. $\mathsf{pp}$, and require that membership in $\mathcal{Y}^{\mathsf{pp}}$ can be checked efficiently.

- $(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b; r)$: upon input public parameters $\mathsf{pp}$, binding position $b \in \{0, 1\}$, and randomness $r$, returns the commitment parameters $(p_0, p_1) \in \mathcal{Y}^{\mathsf{pp}}$ and the trapdoor $\mathsf{td}$ for parameter $p_{1-b}$ such that $(p_{1-b}, \mathsf{td})$ belongs to $\mathcal{R}$.[10]

- $\mathtt{com} \leftarrow \mathsf{BindCom}(\mathsf{pp}, p_0, p_1, m_0, m_1; r)$: upon input public parameters $\mathsf{pp}$, commitment parameters $p_0$, $p_1$, messages $m_0$, $m_1$, and randomness $r$ outputs a commitment $\mathtt{com}$.

- $(\mathtt{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, b, m, p_0, p_1, \mathsf{td}; r)$: upon input public parameters $\mathsf{pp}$, binding position $b$, message of the binding position $m$, commitment parameters $p_0$, $p_1$, trapdoor $\mathsf{td}$, and randomness $r$ returns a commitment $\mathtt{com}$ and auxiliary information $\mathsf{aux}$.

- $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, b, m_0, m_1, p_0, p_1, \mathsf{td}, \mathsf{aux})$: upon input public parameters $\mathsf{pp}$, binding position $b$, messages $m_0$, $m_1$, commitment parameters $p_0$, $p_1$, trapdoor $\mathsf{td}$, and auxiliary information $\mathsf{aux}$, deterministically returns an equivocation randomness $r$.

Here we report the definitions of 1-out-of-2 equivocal commitments as defined in [5]. The description of the algorithms is reported in Sec. 3.1. In the following, we will omit the randomness from the input of the algorithms, except when it is relevant. A sender and a receiver interact using the commitment scheme as follows.

**Commit Phase:** The sender, on input $m$ and binding position $b$, computes $(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b)$, $(\mathtt{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, b, m, p_0, p_1, \mathsf{td})$. The sender sends $(\mathtt{com}, p_0, p_1)$ to the receiver.

[10]The statement for $\mathcal{R}$ may also depend from $\mathsf{pp}$. We will omit this dependence to simplify the notation.

**Reveal Phase:** The sender, on input $m^*$, computes $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, b, m_0, m_1, p_0, p_1, \mathsf{td}, \mathsf{aux})$ where $m_b = m$ and $m_{1-b} = m^*$, and sends $(r, m_0, m_1)$ to the receiver. The receiver computes $\mathtt{com}' \leftarrow \mathsf{BindCom}(\mathsf{pp}, p_0, p_1, m_0, m_1; r)$ and accepts if $\mathtt{com}' = \mathtt{com}$ and $(p_0, p_1) \in \mathcal{Y}^{\mathsf{pp}}$; it rejects otherwise.

We state below the properties we require for 1-out-of-2 equivocal commitment schemes.

**Partial Equivocation:** For all $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, $b \in \{0,1\}$, $(p_0, p_1) \in \mathcal{Y}^{\mathsf{pp}}$, $(m_0, m_1) \in \{0,1\}^{2\lambda}$, $\mathsf{td}$ such that $(p_{1-b}, \mathsf{td}) \in \mathcal{R}$ the following holds:

$$\mathsf{Pr}\left[\begin{array}{c}\mathsf{BindCom}(\mathsf{pp}, p_0, p_1, \\ m_0, m_1; r) = \mathtt{com}\end{array} \middle| \begin{array}{c}(\mathtt{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, b, m_b, p_0, p_1, \mathsf{td}); \\ r \leftarrow \mathsf{Equiv}(\mathsf{pp}, b, m_0, m_1, p_0, p_1, \mathsf{td}, \mathsf{aux}).\end{array}\right] = 1.$$

**Computational Fixed Equivocation:** Given the experiment $\mathsf{ExpFixEquiv}(\lambda)$ below, for every PPT $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that $\mathsf{Pr}[\, \mathsf{ExpFixEquiv}_{\mathcal{A}}(\lambda) = 1\,] \leq \nu(\lambda)$.

$\mathsf{ExpFixEquiv}_{\mathcal{A}}(\lambda)$

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. $(p_0, p_1, r^1, r^2, r^3, r^4, m_0^1, m_0^2, m_1^1, m_1^2, m_0^3, m_0^4, m_1^3, m_1^4) \leftarrow \mathcal{A}(\mathsf{pp})$.
3. Return 1 if $\exists b \in \{0,1\}$ such that

$$\begin{aligned}(\mathsf{BindCom}(\mathsf{pp}, p_0, p_1, m_0^1, m_1^1; r^1) = \mathsf{BindCom}(\mathsf{pp}, p_0, p_1, m_0^2, m_1^2; r^2) \wedge \\ \big(\mathsf{BindCom}(\mathsf{pp}, p_0, p_1, m_0^3, m_1^3; r^3) = \mathsf{BindCom}(\mathsf{pp}, p_0, p_1, m_0^4, m_1^4; r^4)) \wedge \\ (m_{1-b}^1 \neq m_{1-b}^2) \wedge (m_b^3 \neq m_b^4) \wedge ((p_0, p_1) \in \mathcal{Y}^{\mathsf{pp}}).\end{aligned}$$

Return 0 otherwise.

Moreover, the protocol achieves perfect fixed equivocation if for any unbounded $\mathcal{A}$ it holds that $\mathsf{Pr}[\, \mathsf{ExpFixEquiv}_{\mathcal{A}}(\lambda) = 1\,] = 0$.

**Computational Position Hiding:** Given the experiment $\mathsf{ExpHid}(\lambda)$ below, for every PPT $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that
$\mathsf{Pr}[\, \mathsf{ExpHid}_{\mathcal{A}}(\lambda) = 1\,] \leq \frac{1}{2} + \nu(\lambda)$.

$\mathsf{ExpHid}_{\mathcal{A}}(\lambda)$

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. Sample $b \leftarrow_{\$} \{0,1\}$ and compute $(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b)$.
3. $b' \leftarrow \mathcal{A}(\mathsf{pp}, p_0, p_1)$.
4. Return 1 if $b' = b$ and 0 otherwise.

Moreover, if $\mathcal{A}$ is unbounded and $\nu(\lambda) = 0$ we say that the scheme is perfect position hiding.

**Computational Trapdoorness:** Given the experiment $\mathsf{ExpTrap}$, for every PPT $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that $\mathsf{Pr}[\, \mathsf{ExpTrap}_{\mathcal{A}}(\lambda) = 1\,] \leq \frac{1}{2} + \nu(\lambda)$.

$\mathsf{ExpTrap}_{\mathcal{A}}(\lambda)$

1. $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
2. $(m_0, m_1, p_0, p_1, \mathsf{td}, b) \leftarrow \mathcal{A}(\mathsf{pp})$.
3. If $(p_0, p_1) \notin \mathcal{Y}^{\mathsf{pp}}$ or $(p_{1-b}, \mathsf{td}) \notin \mathcal{R}$ abort the experiment.
4. Sample $\beta \leftarrow\$ \{0,1\}$. If $\beta = 0$, set $(\mathtt{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, b, m_b, p_0, p_1, \mathsf{td})$ and set $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, b, m_0, m_1, p_0, p_1, \mathsf{td}, \mathsf{aux})$. If $\beta = 1$, sample $r \leftarrow\$ \mathtt{D}$ and set $\mathtt{com} \leftarrow \mathsf{BindCom}(\mathsf{pp}, p_0, p_1, m_0, m_1; r)$.
5. $\beta' \leftarrow \mathcal{A}(\mathsf{pp}, m_0, m_1, p_0, p_1, \mathsf{td}, b, \mathtt{com}, r)$.
6. Return 1 if $\beta = \beta'$, return 0 otherwise.

Moreover, if $\mathcal{A}$ is unbounded and $\nu(\lambda) = 0$ we say that the protocol achieves perfect trapdoorness.

An instantiation of the 1-out-of-2 equivocal commitment scheme is given in [18], where the instantiated scheme enjoys partial equivocation, computational fixed equivocation, perfect position hiding, and perfect trapdoorness under the discrete logarithm assumption.

In the following, we assume that $\mathsf{pp}$ was already generated by a trusted third party using the algorithm $\mathsf{Setup}$. Additionally, to simplify the notation, we will consider $\mathsf{pp}$ as an implicit input of every algorithm that internally uses a 1-out-of-2 equivocal commitment.

## 3.2 Σ-Protocols

We consider a *3-round* public-coin protocol $\Pi$ for an NP language $\mathcal{L}$ with a poly-time relation $\mathcal{R}_{\mathcal{L}}$. $\Pi = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$ is run by a prover running auxiliary algorithms $\mathsf{P}_0, \mathsf{P}_1$ and a verifier running an auxiliary algorithm $\mathsf{V}$. The prover and the verifier receive common input $x$ and the security parameter $1^\lambda$. The prover receives as an additional private input a witness $w$ for $x$. Prover and verifier use the auxiliary algorithms $\mathsf{P}_0, \mathsf{P}_1, \mathsf{V}$ in the following way:

1. The prover runs $\mathsf{P}_0$ on common input $x$, private input $w$, randomness $R$, and outputs a message $a$. The prover sends $a$ to the verifier;

2. The verifier samples a random challenge $c \leftarrow\$ \{0,1\}^\lambda$ and sends $c$ to the prover;

3. The prover runs $\mathsf{P}_1$ on common input $x$, private input $w$, first-round message $a$, randomness $R$, and challenge $c$, and outputs the third-round message $z$, which is then sent to the verifier;

4. The verifier outputs 1 if $\mathsf{V}(x, a, c, z) = 1$, and rejects otherwise.

The transcript $(a, c, z)$ for the protocol $\Pi = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$, and common statement $x$ is called *accepting* if $\mathsf{V}(x, a, c, z) = 1$.

**Definition 1 (Σ-protocol)** *A 3-round public-coin protocol* $\Pi = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$, *is a Σ-protocol for an NP language* $\mathcal{L}$ *with a poly-time relation* $\mathcal{R}_{\mathcal{L}}$ *iff the following holds*

**Completeness:** *For all* $x \in \mathcal{L}$ *and* $w$ *such that* $(x, w) \in \mathcal{R}_{\mathcal{L}}$ *it holds that:*

$$\mathsf{Pr}\left[ \mathsf{V}(x, a, c, z) = 1 \;\middle|\; \begin{array}{c} R \leftarrow\$ \{0,1\}^\lambda; c \leftarrow\$ \{0,1\}^\lambda; \\ a \leftarrow \mathsf{P}_0(x, w; R); \\ z \leftarrow \mathsf{P}_1(x, w, a, c; R) \end{array} \right] = 1.$$

**Special Soundness:** *There exists a PPT extractor* $\mathsf{Ext}$*, such that on input* $x$ *and two accepting transcripts* $(a, c_0, z_0)$ *and* $(a, c_1, z_1)$ *for* $x$*, where* $c_0 \neq c_1$*, it holds that*

$$\Pr\left[\, (x, w) \in \mathcal{R}_{\mathcal{L}} | w \leftarrow \mathsf{Ext}(x, a, c_0, c_1, z_0, z_1) \,\right] = 1.$$

**Special HVZK (SHVZK):** *There exists a PPT simulator* $\mathcal{S}$ *that, on input an instance* $x \in \mathcal{L}$ *and challenge* $c$*, outputs* $(a, z)$ *such that* $(a, c, z)$ *is an accepting transcript for* $x$*. Moreover, the distribution of the output of* $\mathcal{S}$ *on input* $(x, c)$ *is computationally/statistically/perfectly indistinguishable from the distribution obtained when the verifier sends* $c$ *as a challenge and the prover runs on common input* $x$ *and any private input* $w$ *such that* $(x, w) \in \mathcal{R}_{\mathcal{L}}$*.*

We will also consider Σ-protocols where prover and verifier also use some trusted parameters, and in this case we replace the above special soundness with the following computational special soundness.

**Computational Special Soundness:** There exists a $PPT$ extractor $\mathsf{Ext}$ such that $\forall$ $PPT$, $\mathcal{A}$, $\lambda \in \mathbb{N}$ $\exists$ a negligible function $\nu(\cdot)$ such that $\Pr\left[\, \mathsf{ExpExt}_{\mathcal{A},\mathsf{Ext}}(\lambda) = 1 \,\right] \leq \nu(\lambda)$.

$\mathsf{ExpExt}_{\mathcal{A},\mathsf{Ext}}(\lambda)$

1. $(x, a, c_0, c_1, z_0, z_1) \leftarrow \mathcal{A}(\lambda)$.
2. If $c_0 = c_1$, or $\mathsf{V}(x, a, c_0, z_0) = 0$, or $\mathsf{V}(x, a, c_1, z_1) = 0$ return 0.
3. $w \leftarrow \mathsf{Ext}(x, a, c_0, c_1, z_0, z_1)$.
4. Return 1 if $(x, w) \notin \mathcal{R}_{\mathcal{L}}$. Otherwise, return 0.

In the above definition of computational special soundness, one might think $\nu(\lambda)$ must correspond to 0 since otherwise there always is an $\mathcal{A}$ having the succeeding tuple $(x, a, c_0, c_1, z_0, z_1)$ hardwired in its code. However, notice that as in [18, 5] we are assuming that there is a trusted parameter $\mathsf{pp}$ implicitly received as input. This can allow the design of a computationally special-sound Σ-protocol ruling out the above adversary even for $\nu(\lambda) > 0$.

## 3.3 Stackable Σ-Protocols

Stackable Σ-protocols are a strengthening of the standard Σ-protocols (Sec. 3.2). The extended honest-verifier zero-knowledge (EHVZK) property requires the existence of an *extended simulator*, that, given a third-round message sampled from the space of admissible third-round messages, outputs a unique first-round message[11]. Additionally, the simulator shall be able to reuse a specific third-round message across different statements, so that multiple first-round messages can be derived from the same third-round message. Goel et al. [18] showed that all Σ-protocols can be made EHVZK and that usually Σ-protocols are already stackable.

**Definition 2 (Computational EHVZK)** *Let* $\Sigma = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$*, be a* Σ*-protocol for an NP language* $\mathcal{L}$*.* Σ *is EHVZK if there exists a deterministic polynomial-time algorithm* $\mathcal{S}^{\mathsf{EHVZK}}$ *such that for all PPT* $\mathcal{D}$*, for all* $\lambda \in \mathbb{N}$*, and* $c \in \{0,1\}^\lambda$*, there exists an efficiently samplable distribution* $D^{(z)}_{x,c}$ *and a negligible function* $\nu(\cdot)$ *such that*

$$\left| \Pr\left[\, \mathsf{ExpEHVZK}_{(\mathsf{P}_0,\mathsf{P}_1),\mathcal{D},D^{(z)}_{x,c}}(\lambda, c) = 1 \,\right] - \Pr\left[\, \mathsf{ExpEHVZK}_{\mathcal{S}^{\mathsf{EHVZK}},\mathcal{D},D^{(z)}_{x,c}}(\lambda, c) = 1 \,\right] \right| \leq \nu(\lambda).$$

The experiment $\mathsf{ExpEHVZK}$ for EHVZK follows.

[11] A similar definition was introduced by Abe et al. [3] in their $\mathsf{type\text{-}T}$ signatures schemes.

$\mathsf{ExpEHVZK}_{\mathsf{P}',\mathcal{D},D^{(z)}_{x,c}}(\lambda, c)$

1. $(x, w) \leftarrow \mathcal{D}(\lambda, c)$.
2. If $(x, w) \notin \mathcal{R}_{\mathcal{L}}$, return 0.
3. If $\mathsf{P}' = \mathcal{S}^{\mathsf{EHVZK}}$, sample $z \leftarrow\!\!\$\ D^{(z)}_{x,c}$ and compute $a \leftarrow \mathcal{S}^{\mathsf{EHVZK}}(x, c, z)$.
4. Otherwise, sample $R \leftarrow\!\!\$\ \{0,1\}^{\lambda}$, compute $a \leftarrow \mathsf{P}_0(x, w; R)$ and $z \leftarrow \mathsf{P}_1(x, w, a, c; R)$.
5. Return $\mathcal{D}(x, w, a, c, z)$.

**Definition 3 (Statistical/Perfect EHVZK)** *Let $\Sigma = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$, be a $\Sigma$-protocol for an $NP$ language $\mathcal{L}$. $\Sigma$ is statistical/perfect EHVZK if there exists a deterministic polynomial-time algorithm $\mathcal{S}^{\mathsf{EHVZK}}$ such that for all $\lambda \in \mathbb{N}$, and $c \in \{0,1\}^{\lambda}$, there exists an efficiently samplable distribution $D^{(z)}_{x,c}$ such that for all $x \in \mathcal{L}$*

$$\Big\{(a, c, z) \mid R \leftarrow\!\!\$\ \{0,1\}^{\lambda}, a \leftarrow \mathsf{P}_0(x, w; R), z \leftarrow \mathsf{P}_1(x, w, a, c; R)\Big\} \approx \Big\{(a, c, z) \mid z \leftarrow\!\!\$\ D^{(z)}_{x,c}, a \leftarrow \mathcal{S}^{\mathsf{EHVZK}}(x, c, z)\Big\},$$

*where $\approx$ can either denote statistical or perfect indistinguishability.*

**Definition 4 ($\Sigma$-protocol with recyclable third messages)** *Let $\Sigma = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$ be a $\Sigma$-protocol for an $NP$ language $\mathcal{L}$, $\Sigma$ has recyclable third messages if for every $c \in \{0,1\}^{\lambda}$, there exists an efficiently samplable distribution $D^{(z)}_c$, such that for all $(x, w) \in \mathcal{R}_{\mathcal{L}}$, it holds that $D^{(z)}_c \approx \{z | R \leftarrow\!\!\$\ \{0,1\}^{\lambda}; a \leftarrow \mathsf{P}_0(x, w; R); z \leftarrow \mathsf{P}_1(x, w, c; R)\}$.*

**Definition 5 (Stackable $\Sigma$-protocol)** *We say that a $\Sigma$-protocol $\Sigma = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$, is stackable, if it is a EHVZK $\Sigma$-protocol and has recyclable third messages.*

The instantiation of the stackable $\Sigma$-protocol for disjunctions of [18] (and hence ours) requires the existence of a transparent setup. As it is done in [18], we consider a definition of stackable $\Sigma$-protocols that does not contemplate a setup. We find that this omission does not affect the protocol description and the security proofs while it improves the overall readability of the paper.

## 3.4 The Disjunction Composition of [18]

We give a description of the disjunction compiler of [18]. Let us start by describing the compiler for a disjunction featuring just two base statements. Let $\Pi = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$ be a stackable $\Sigma$-protocol for language $\mathcal{L}$ and let $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{base}}$ be the simulator for $\Pi$. Let $x = (x_0, x_1)$ be the public input. Let $w = (w_\alpha, \alpha)$, with $\alpha \in \{0,1\}$, so that $(x_\alpha, w_\alpha) \in \mathcal{R}_{\mathcal{L}}$ be the prover's private input. The compiler gives a stackable $\Sigma$-protocol $\Pi_{1,2} = (\mathsf{P}^{1,2}_0, \mathsf{P}^{1,2}_1, \mathsf{V}^{1,2})$ for the relation $\mathcal{R}_{1,2} = \{(x = (x_0, x_1), w_\alpha) | (x_0, w_\alpha) \in \mathcal{R}_{\mathcal{L}} \ \vee (x_1, w_\alpha) \in \mathcal{R}_{\mathcal{L}}\}$. We report such compiler in Fig. 3.

Given the compiler in Fig. 3 it is straightforward to extend this protocol to handle disjunctions with $k$ base statements using recursion as described in [18]. We call the resulting protocol $\Pi_{\mathsf{OR}} = (\mathsf{P}^{\mathsf{OR}}_0, \mathsf{P}^{\mathsf{OR}}_1, \mathsf{V}^{\mathsf{OR}})$. Notice that, to avoid to later re-compute such data re-using the randomness, we can extend the output of $\mathsf{P}^{\mathsf{OR}}_0((x_1, \ldots, x_k), (w, \alpha); \mathtt{rand})$ to include additional data aside the first-round message $a$. In particular the extended output can be of the form $(a, \mathbf{p} = ((p^1_0, p^1_1), \ldots, (p^{\log k}_0, p^{\log k}_1), \mathbf{td} = (\mathsf{td}^1, \ldots, \mathsf{td}^{\log k}))$, where $a$ is the first-round message, $\mathbf{p}$ and $\mathbf{td}$ the tuple of $\log k$ parameters and the respective trapdoors to be used by the underlying composition 1-out-of-2 equivocal commitment. The algorithm $\mathsf{P}^{\mathsf{OR}}_1((x_1, \ldots, x_k), (w, \alpha), a, c; \mathtt{rand})$ outputs the third-round message $z$, and $\mathsf{V}^{\mathsf{OR}}((x_1, \ldots, x_k), a, c, z)$ outputs $1/0$.

Let $\Pi = (\mathsf{P}_0, \mathsf{P}_1, \mathsf{V})$ be a stackable $\Sigma$-protocol for relation $\mathcal{R}_{\mathcal{L}}$ and $(\mathsf{Gen}, \mathsf{BindCom}, \mathsf{EquivCom}, \mathsf{Equiv})$ the algorithms of a 1-out-of-2 equivocal commitment scheme. The following compiler produces a stackable $\Sigma$-protocol $\Pi^{1,2} = (\mathsf{P}_0^{1,2}, \mathsf{P}_1^{1,2}, \mathsf{V}^{1,2})$ for relation $\mathcal{R}_{\mathsf{OR}} = \{((x_0, x_1), w) | (x_0, w) \in \mathcal{R}_{\mathcal{L}} \vee (x_1, w) \in \mathcal{R}_{\mathcal{L}}\}$. Let $\mathsf{pp}$ be public parameters of the 1-out-of-2 equivocal commitment scheme.

**First round:** The prover, on input $(x, (w, b), \mathtt{rand}_\mathsf{P})$, runs $\mathsf{P}_0^{1,2}(x, (w, b); \mathtt{rand}_\mathsf{P})$ as follows:

- Parse $\mathtt{rand}_\mathsf{P} = (\mathtt{rand}^b || \mathtt{rand}_0 || \mathtt{rand}_1)$;
- $v_b \leftarrow \mathsf{P}_0(x_b, w_b; \mathtt{rand}^b)$;
- $(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b; \mathtt{rand}_0)$;
- $(\mathsf{com}, \mathsf{aux}) \leftarrow \mathsf{EquivCom}(\mathsf{pp}, b, v_b, p_0, p_1, \mathsf{td}; \mathtt{rand}_1)$;
- Output $(\sigma_1, \mathsf{td})$ where $\sigma_1 = (\mathsf{com}, p_0, p_1)$.

Finally, the prover sends $a = \sigma_1$ to the verifier.

**Second round:** The verifier samples a challenge $c \leftarrow_\$ \{0, 1\}^\lambda$ and sends $c$ to the prover.

**Third round:** The prover runs $\mathsf{P}_1^{1,2}(x, (w, b), a, c; \mathtt{rand}_\mathsf{P})$ as follows:

- Parse $\mathtt{rand}_\mathsf{P} = (\mathtt{rand}^b || \mathtt{rand}_0 || \mathtt{rand}_1)$ and $a = (\mathsf{com}, p_0, p_1)$;
- Compute $z^* \leftarrow \mathsf{P}_1(x_b, w_b, v_b, c; \mathtt{rand}^b)$ (where $v_b$ is re-computed as in the first round);
- $v^*_{1-b} \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{base}}(x_{1-b}, c, z^*)$;
- Set $v^*_b = v_b$;
- $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, b, v^*_0, v^*_1, p_0, p_1, \mathsf{td}, \mathsf{aux})$ (where $\mathsf{td}$ and $\mathsf{aux}$ are computed with $\mathtt{rand}_\mathsf{P}$);
- Output $z = (z^*, r, p_0, p_1)$.

Finally, the prover sends $z$ to $\mathsf{V}^{1,2}$.

**Verification:** $\mathsf{V}^{1,2}$, in input $(x, a, c, z)$, does as follows:

- Parse $a = (\mathsf{com}, p_0, p_1)$ and $z = (z^*, r, p'_0, p'_1)$;
- If $(p_0, p_1) \neq (p'_0, p'_1)$ return 0, otherwise continue to next step;
- If $(p_0, p_1) \notin \mathcal{Y}^{\mathsf{pp}}$ return 0, otherwise continue to next step;
- Set $v_i \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{base}}(x_i, c, z^*)$, for $i \in \{0, 1\}$;
- Compute $\mathsf{com}' \leftarrow \mathsf{BindCom}(p_0, p_1, v_0, v_1, c; r)$. If $\mathsf{com}' = \mathsf{com}$, return $\mathsf{V}(x_0, v_0, c, z^*) \wedge \mathsf{V}(x_1, v_1, c, z^*)$, otherwise output 0.

Figure 3: Disjunction composition of [18] for $k = 2$ from stackable $\Sigma$-protocols.

We recall that $\Pi_{\mathsf{OR}}$ of [18] is a stackable $\Sigma$-protocol with computational special soundness. If the employed 1-out-of-2 equivocal commitment scheme satisfies perfect trapdoorness and perfect position hiding, then the EHVZK flavor (i.e., perfect, statistical, or computational) of the base $\Sigma$-protocol is preserved.

## 3.5 The Ordering Protocol of [5]

Here, we give a brief overview of the ordering protocol $\Pi^{\mathsf{ord}}$ of [5]. We refer to [5] for more details. $\Pi^{\mathsf{ord}} = (\mathsf{P}_0^{\mathsf{ord}}, \mathsf{P}_1^{\mathsf{ord}}, \mathsf{V}^{\mathsf{ord}})$ is a stackable $\Sigma$-protocol. Given $k$ vectors $\mathbf{p}_1, \ldots, \mathbf{p}_k$ of $n$ pairs of parameters of a 1-out-of-2 equivocal commitment and $k$ vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k$ of witnesses (i.e., the corresponding trapdoors), the relation proved by $\Pi^{\mathsf{ord}}$ is $\mathcal{R}_{\mathsf{ord}}((\mathbf{p}_1, \ldots, \mathbf{p}_k), (\mathbf{w}_1, \ldots, \mathbf{w}_k)) = \bigwedge_{i=1}^{k-1} \mathcal{R}_{\mathsf{ord}'}((\mathbf{p}_i, \mathbf{p}_{i+1}), (\mathbf{w}_i, \mathbf{w}_{i+1}))$. Informally, the two sequences of parameters $\mathbf{p}, \mathbf{q}$ (together with their trapdoors) satisfy relation $\mathcal{R}_{\mathsf{ord}'}$ whenever the binary string corresponding to $\mathbf{p}$ is smaller than the one corresponding to $\mathbf{q}$ (Sec. 2). As a result, a witness for $\mathcal{R}_{\mathsf{ord}}$ is a set of trapdoors each of which corresponds to a different all-binding path in its composition tree. More in details, [5] is instantiated with the equivocal commitment scheme presented in [18] based on the discrete logarithm assumption that is a variation of the Pedersen commitment. Concretely, the case of the 1-out-of-2 equivocal commitment scheme, instead of having only generators $g$ and $h$, for which the sender does not know the discrete logarithm of $g$ with base $h$, has three generators $g_1, g_2, h$ such that the sender knows the discrete logarithm of either $g_1$ or $g_2$ in base $h$. Let us consider the relation $\mathcal{R}_{\mathsf{DL}}(p_b^i, w_p^i)$ as the function evaluating to 1 if $w_p^i$ is the discrete logarithm of $p_b^i$ with regard to $h$ and 0 otherwise. Notice that $\mathcal{R}_{\mathsf{DL}}$ can be substituted by any relation that returns 1 if the parameter $p_b^i$ in input is the parameter associated with the trapdoor $w_p^i$ and 0 otherwise. Let us consider two vectors of parameters $\mathbf{p} = ((p_0^1, p_1^1), \ldots, (p_0^n, p_1^n))$ and $\mathbf{q} = ((q_0^1, q_1^1), \ldots, (q_0^n, q_1^n))$, and two vectors of trapdoors $\mathbf{w}_p = (w_p^1, \ldots, w_p^n)$ and $\mathbf{w}_q = (w_q^1, \ldots, w_q^n)$, then $\mathcal{R}_{ord'}((\mathbf{p}, \mathbf{q}), (\mathbf{w}_p, \mathbf{w}_q))$ is defined as follows:

$$\bigvee_{i=1}^{n} \left( \left( \bigwedge_{j=0}^{i-1} (((\mathcal{R}_{\mathsf{DL}}(p_0^j, w_p^j) \wedge \mathcal{R}_{\mathsf{DL}}(q_0^j, w_q^j)) \vee (\mathcal{R}_{\mathsf{DL}}(p_1^j, w_p^j) \wedge \mathcal{R}_{\mathsf{DL}}(q_1^j, w_q^j))) \right) \right.$$
$$\left. \wedge\, (\mathcal{R}_{\mathsf{DL}}(p_1^i, w_p^i) \wedge \mathcal{R}_{\mathsf{DL}}(q_0^i, w_q^i)) \right). \quad (2)$$

In a nutshell, to check whether the string corresponding to a parameters vector $\mathbf{p}$ is smaller than another one $\mathbf{q}$, $\mathcal{R}_{\mathsf{ord}'}$ checks that either there exist a bit within the $\mathbf{p}$ which is smaller than the corresponding one in $\mathbf{q}$, while all the other digits are identical in both strings.

The ordering protocol instantiation of [5] satisfies computational special soundness and perfect HVZK. The communication complexity is $O((k-1)n)$.

# 4 Our $(k, m)$-CNF Composition

To construct our $(\tau, k, m)$-CNF composition technique (Sec. 5), we first propose a novel stackable $\Sigma$-protocol $\Pi^{\mathsf{cnf}}$ for the following relation:

$$\mathcal{R}_{1,k}^{\mathsf{cnf}} = \{((x^1, \ldots, x^m), (w_1, \ldots, w_m)) :\ \forall\, i \in [m] : (x^i, w_i) \in \mathcal{R}_{1,k}\},$$

where $\mathcal{R}_{1,k} = \{((x_1, \ldots, x_k), (w, \alpha)) | 1 \leq \alpha \leq k \wedge\ (x_\alpha, w) \in \mathcal{R}_{\mathcal{L}}\}$.

$\Pi^{\mathsf{cnf}}$ is obtained via parallel execution of $\Pi_{\mathsf{OR}}$. Additionally, in one of these executions of $\Pi_{\mathsf{OR}}$ (the one related to the merged composition tree), we slightly modify the internal recursive composition to account for commitment parameters and equivocation randomnesses batching. We report our stackable $\Sigma$-protocol $\Pi^{\mathsf{cnf}}$ for $\mathcal{R}_{1,k}^{\mathsf{cnf}}$ in Fig. 4 for a $(k, m+1)$-CNF to improve its readability[12]. In Fig. 4 we consider the following values:

[12]Notice that we do not cover the case in which $\ell = k - 1$ since it is trivial and adding it will make the scheme less readable.

- Let $k$ be the length of the clauses that we assume to be a power of 2. Consequently, $\log k$ will be the height of the merged composition tree.
- Let $m+1$ be the total number of clauses in the CNF formula, i.e., $\mathsf{P}_0^{\mathsf{cnf}}$ takes in input a list of $m+1$ set of literals $(x, x^1, \dots x^m)$, where $x = (x_1, \dots, x_k)$ and $x^t = (x_1^t, \dots, x_k^t)$ with $t \in [m]$;
- Let $\ell$ be the number of literals that are shared (i.e., are in the prefix) of the $m+1$ clauses. Then, $k-\ell$ is the number of literals which are different between the clauses, we require that it is a power of 2;
- Let $d$ be equal to $\lfloor \log(\frac{k}{k-\ell}) \rfloor$. The value $d$ represents the number of levels of the composition tree in which the parameters are reused across the clauses. Notice that $\log(k-\ell)$ is the height of the composition sub-trees rooted at level $d$ in the $m$ clauses $(x^1, \dots x^m)$. Such sub-trees do not share parameters with the other sub-trees of the other clauses;
- We define the values $e_t$ and $d_t$ as follows. Recall from Sec. 2 that for the $m$ clauses $(x^1, \dots x^m)$ we need to compute the protocol of [18] on the $k-\ell$ literals in the composition tree that includes the witness $w_{\alpha_t}$, for $t \in [m]$. To do so, $\mathsf{P}_0^{\mathsf{cnf}}$ computes $e_t = \lfloor \alpha_t/(k-\ell) \rfloor$ that is the index of the sub-tree of $k-\ell$ leaves that contains the literal indexed $\alpha_t \in [k]$. i.e., the statement for which $\mathsf{P}_0^{\mathsf{cnf}}$ knows the witness $w_{\alpha_t}$. After having computed $e_t$, $\mathsf{P}_0^{\mathsf{cnf}}$ computes the index of the first statement among $(x_1^t, \dots, x_k^t)$ that corresponds to the $e_t$-th sub-tree. This index is $d_t = e_t(k-\ell)+1$ (it is trivial to see that the last statement of this sub-tree has index $d_t + k - \ell - 1$). The last thing that $\mathsf{P}_0^{\mathsf{cnf}}$ has to compute is the new index to assign to the witness within the sub-tree $w_{\alpha_t}$, that is $\alpha_t = \alpha_t - d_t + 1$ (i.e., to match the interface of $\mathsf{P}_{\mathsf{OR}}$ this index has to be re-scaled w.r.t. the length of the sub-clause of the sub-tree).

In particular, we introduce three auxiliary algorithms $\mathsf{First}$, $\mathsf{Third}$, and $\mathsf{Verify}$, formally described in Fig. 5, 6, and 7 respectively. $\mathsf{First}$ and $\mathsf{Third}$ are run by the prover to produce the first and third-round messages to be sent to the verifier. $\mathsf{Verify}$ is used by the verifier to verify the proof. These algorithms take as inputs a set of clauses[13] $x, x^1, \dots, x^m$ sharing a common prefix of length $\ell$. Moreover, these algorithms take as input an additional value called $d > 0$ which indicates the number of levels of the trees that can share the commitment parameters[14]. Let us consider the simple example of Fig. 2. In this case, $d = 2$. Now, the job of $\mathsf{First}$ is to create the merged composition tree depicted on top of the figure. Before invoking $\mathsf{First}$, the prover has to compute the intermediate commitments $\{a_t\}_{t\in[m]}$ to be merged in the composition tree (i.e., the root $a_{9,10}$ of the tree depicted in the bottom of Fig. 2). This is done by running $\mathsf{P}_0^{\mathsf{OR}}$ on the appropriate portions of each clause $\{x^t\}_{t\in[m]}$ (i.e., $x_9 \vee x_{10}$ in Fig. 2). Then, the prover invokes $\mathsf{First}$ with the additional inputs $\{a_t\}_{t\in[m]}$, and an empty vector $\mathbf{td}$ that will be updated by $\mathsf{First}$ with the trapdoors of the commitment parameters. $\mathsf{First}$ works as follows:

**Recursive phase:** divide the instance in half as follows:

- Pick the part of the instance containing the base instance the prover holds the witness for (we call it the active instance);
- At each recursive call, decrement the value of $d$ by 1;
- Once the base case of $d = 1$ is reached, call $\Pi_{\mathsf{OR}}$ on the portion of the instance containing the active instance, obtaining an internal node $a'_d$ (i.e., $a_{7,8}$ in Fig. 2);

**Combine phase:** commitments at level $d$ are then recombined as follows:

- All $\{a_t\}_{t\in[m]}$ (i.e., $a_{9,10}$ in Fig. 2) are appropriately concatenated with $a'_d$ to create the final node $a_d$ at level $d$;
- Finally, the merged composition tree is finalized as in a regular execution of $\Pi_{\mathsf{OR}}$.

[13]Here we are constructing a $(k, m+1)$-CNF, where we call the first statement $x$ and the remaining $x^1, \dots, x^m$. The same holds also for the witnesses. We use this notation for readability.

[14]We do not consider the case of $d = 0$, since it is equivalent to repeating $\Pi_{\mathsf{OR}}$ in parallel $m+1$ times.

Let $\Pi_{\mathsf{OR}} = (\mathsf{P}_0^{\mathsf{OR}}, \mathsf{P}_1^{\mathsf{OR}}, \mathsf{V}^{\mathsf{OR}})$ be the stackable $\Sigma$-protocol of [18] for relation $\mathcal{R}_{\mathsf{OR}}$. In the following we describe our stackable $\Sigma$-protocol $\Pi^{\mathsf{cnf}} = (\mathsf{P}_0^{\mathsf{cnf}}, \mathsf{P}_1^{\mathsf{cnf}}, \mathsf{V}^{\mathsf{cnf}})$ for relation $\mathcal{R}_{1,k}^{\mathsf{cnf}}$. Let $x^1, \ldots, x^m$ be the clauses sharing the first $2^d$ base statements with the first clause $x$. Let $d = \lfloor \log(\frac{k}{k-\ell}) \rfloor$ represent the number of levels that share parameters across the composition trees of each clause (i.e., $0 < d \leq \log k$).

**First round:** The algorithm $\mathsf{P}_0^{\mathsf{cnf}}((x, x^1, \ldots x^m), (w, w_1, \ldots, w_m); \mathtt{rand})$, where $x^t = (x_1^t, \ldots, x_k^t)$ for $t \in [m]$, works as follows:

- Parse $w$ as $(w_\alpha, \alpha)$, and for $t \in [m]$ parse $w_t$ as $(w_{\alpha_t}, \alpha_t)$;
- Parse $\mathtt{rand}$ as $(\mathtt{rand}', \mathtt{rand}_1, \ldots, \mathtt{rand}_m)$;
- Set $\mathbf{td} = []$;
- Let $\ell$ be the common prefix of the $m+1$ clauses, let $e_t = \lfloor \alpha_t/(k-\ell) \rfloor$, compute $d_t = e_t(k-\ell)+1$, set $w_\beta = w_{\alpha_t}$ and $\alpha_t = \alpha_t - d_t + 1$. Compute $(a_t, \cdot) \leftarrow \mathsf{P}_0^{\mathsf{OR}}((x_{d_t}^t, \ldots, x_{d_t+k-\ell-1}^t), (w_\beta, \alpha_t); \mathtt{rand}_t)$ for $t \in [m]$;
- Parse $a_t = (\mathtt{com}_t, (p_0^{t,1}, p_1^{t,1}), \ldots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)}))$, for $t \in [m]$;
- Run $(\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k)}, p_1^{\log(k)})) \leftarrow \mathsf{First}(x, w_\alpha, \alpha, d, \{\mathtt{com}_t\}_{t\in[m]}, \mathtt{rand}', \mathbf{td})$.

The prover sends $a = (\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k)}, p_1^{t,\log(k)}))$ to the verifier.

**Second round:** The verifier samples a challenge $c \leftarrow_\$ \{0,1\}^\lambda$ and sends $c$ to the prover.

**Third round:** The algorithm $\mathsf{P}_1^{\mathsf{cnf}}((x, x^1, \ldots x^m), (w, w_1, \ldots, w_m), a, c, \mathbf{td}; \mathtt{rand})$ works as follows:

- Parse $\mathtt{rand}$ as $(\mathtt{rand}', \mathtt{rand}_1, \ldots, \mathtt{rand}_m)$;
- For each $t \in [m]$, parse $w_t$ as $(w_{\alpha_t}, \alpha_t)$;
- Recompute $v_\alpha$ as the first-round message related to $x$ committed in the binding position at level $d$ (e.g., $a_{7,8}$ in Fig. 2);
- For each $t \in [m]$, set $\alpha_t = \alpha_t - d_t + 1$ compute $z_t \leftarrow \mathsf{P}_1^{\mathsf{OR}}((x_{d_t}^t, \ldots, x_{d_t+e}^t), (w_t, \alpha_t), a_t, c; \mathtt{rand}_t)$.
- Parse $a_t = (\mathtt{com}_t, (p_0^{t,1}, p_1^{t,1}), \ldots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)}))$, for $t \in [m]$;
- $\tilde{z} \leftarrow \mathsf{Third}(x, w_\alpha, \alpha, d, \{x^t\}_{t\in[m]}, \{\mathtt{com}_t\}_{t\in[m]}, \{z_t\}_{t\in[m]}, v_\alpha, c, \mathtt{rand}', \mathbf{td})$.

The prover sends $z = (\tilde{z}, z_1, \ldots, z_m)$ to the verifier.

**Verification:** $\mathsf{V}^{\mathsf{cnf}}(x, x^1, \ldots x^m, a, c, \tilde{z}, z_1, \ldots, z_m)$, works as follows:

- $(\mathsf{bool}, \cdot, \cdot, \cdot) \leftarrow \mathsf{Verify}(x, \{x^t\}_{t\in[m]}, d, a, \tilde{z}, \{z_t\}_{t\in[m]}, c)$;
- If $\mathsf{bool} = \mathsf{true}$ output 1, and 0 otherwise.

Figure 4: Our stackable $\Sigma$-protocol $\Pi^{\mathsf{cnf}}$ for $\mathcal{R}_{1,k}^{\mathsf{cnf}}$.

After having received the challenge from the verifier, the prover first computes, by running $\mathsf{P}_1^{\mathsf{OR}}$, the third-round messages for all the portions of the clauses $\{x^t\}_{t\in[m]}$ identified in the first round. As a result, the prover gets $m$ third-round messages $\{z_t\}_{t\in[m]}$ that will be given in input to $\mathsf{Third}$. The third-round messages $\{z_t\}_{t\in[m]}$ already contain the equivocation randomnesses, from the leaves up to level $d$, of the unmerged composition trees. The job of $\mathsf{Third}$ is to compute, along with the third-round message related to the active base instance of $x$, the equivocation randomnesses of the merged composition tree. Up to level $d$, such randomnesses are computed as in a regular execution of $\Pi_{\mathsf{OR}}$. Then from level $d$ up to the root of the merged composition tree, Third takes into account the third-round messages of the other clauses $\{z_t\}_{t\in[m]}$, to appropriately equivocate the merged nodes. Notice that this is basically a regular execution of $\Pi_{\mathsf{OR}}$ where the trapdoor branches are equivocated up until the root, with the exception that the value to equivocate at level $d$ not only comes from the output of the EHVZK simulator w.r.t. (a portion of) clause $x$, but also from the output of the EHVZK simulator w.r.t. the clauses $\{x^t\}_{t\in[m]}$. Third works as follows:

**Recursive phase:** divide the instances $x, \{x^t\}_{t\in[m]}$ in half as follows:

- Consider the index $\alpha$ of the active instance in clause $x$. Let us update $x$ and $x^t$, $t \in [m]$, to contain half of $x$ and $x^t$ respectively. The selected half contains the index $\alpha$ of active instance of $x$, while $\bar{x}$ and $\bar{x}^t$ contain the other halves;
- At each recursive call, decrement the value of $d$ by 1;
- Once the base case of $d = 1$ is reached, the algorithm handles the equivocation of the merged nodes as follows:
  - Compute the third-round message $z^*$ related to $x$ using $\mathsf{P}_1^{\mathsf{OR}}$;
  - Compute the first-round message $v_{1-b}$ related to $\bar{x}$ using the EHVZK simulator on input $z^*$;
  - Compute all the other first-round messages related to $\{\bar{x}^t\}_{t\in[m]}$ using the EHVZK simulator on input the corresponding $z_t$;
  - Concatenate such first-round messages and equivocate the node at level $d$ of the merged composition tree accordingly.

**Combine phase:** Finally, $z^*$ at level $d$ is treated as in a regular execution of $\Pi_{\mathsf{OR}}$ until the execution is terminated when the root of the merged composition tree is reached.

The verification algorithm Verify is very straightforward. The goal of Verify is to reconstruct the individual composition trees related to all the clauses, and then to simply run $\mathsf{V}_{\mathsf{OR}}$ on all of them. It recursively divides all the clauses in half and explores both halves, decrementing the value of $d$ at each step and checking the consistency of the parent commitment as $\mathsf{V}_{\mathsf{OR}}$ would do. Once the base case of $d = 1$ is reached, the merged node is unpacked, so that $\mathsf{V}_{\mathsf{OR}}$ can be called on all the reconstructed composition trees related to each individual clause.

**Communication complexity.** The first and second-round messages are composed of two constant-size values, while the third-round message is $z = (\tilde{z}, z_1, \ldots, z_m)$, where $\tilde{z}$ is, in turn, composed of the third-round message of the base sigma protocol $\Sigma$, $\log k$ pairs of commitment parameters, and $\log k$ equivocation randomnesses. Additionally, each $z_i$, with $i \in [m]$ is composed of a third-round message of the base sigma protocol $\Sigma$, $\log(k-\ell)$ pairs of commitment parameters, and $\log(k-\ell)$ equivocation randomnesses. Therefore, the overall communication complexity of $\Pi^{\mathsf{cnf}}$ is $O(\log k + m \cdot (\gamma(\Sigma) + \log(k-\ell)))$.

**Theorem 1** *Assuming that $\Pi_{\mathsf{OR}}$ (Fig. 3) is a stackable $\Sigma$-protocol (Def. 5) for the relation $\mathcal{R}_{\mathsf{OR}}$, and assuming the existence of a 1-out-of-2 equivocal commitment scheme* (Setup, Gen, BindCom, EquivCom, Equiv) *as defined in Sec. 3.1, $\Pi^{\mathsf{cnf}}$ (Fig. 4) is a stackable $\Sigma$-protocol for the relation $\mathcal{R}^{\mathsf{cnf}}$. If the 1-out-of-2 equivocal commitment scheme has perfect trapdoorness, then $\Pi^{\mathsf{cnf}}$ preserves the EHVZK flavor of $\Pi_{\mathsf{OR}}$.*

We prove Thm. 1 using lemmas Lem. 1, Lem. 2, Lem. 3 for which we require the same assumptions stated in Thm. 1.

**Lemma 1** *$\Pi^{\mathsf{cnf}}$ is complete.*

**Proof 1 (Completeness)** *Completeness follows from the completeness of $\Pi_{\mathsf{OR}}$ and of the 1-out-of-2 equivocal commitment scheme. Given $m+1$ clauses $(x, x^1, \ldots x^m)$ and $m+1$ witnesses $(w, w_1, \ldots, w_m)$, the protocol generates a transcript $(a, c, z)$, where $a = ((com, p_0^1, p_1^1), \ldots, p_0^{\log k}, p_1^{\log k})$ and $z = (\tilde{z}, z_1, \ldots, z_m)$, where $\tilde{z} = ((z', p_0^1, p_1^1), \ldots, p_0^{\log k}, p_1^{\log k}))$ and $z_t = ((z'_t, p_0^1, p_1^1), \ldots, p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$ with $t \in [m]$.*

```
$\mathsf{First}(x, w_\alpha, \alpha, d, \{\mathtt{com}_t\}_{t\in[m]}, \mathtt{rand}_{\mathsf{P}}, \mathbf{td})$
parse $\mathtt{rand}_{\mathsf{P}} = (\mathtt{rand}^b||\mathtt{rand}_0||\mathtt{rand}_1)$
if $\alpha \le |x|/2 : b = 0$
else $b = 1$
$(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b; \mathtt{rand}_0)$
$\mathbf{td}.append(\mathsf{td})$
if $\alpha \le |x|/2 : x = \{x_1, \dots, x_{|x|/2}\}$
else $x = \{x_{|x|/2+1}, \dots, x_{|x|}\}$
$\alpha = \alpha \mod |x|$
if $d = 1$ :
    $(a, \mathbf{td}') \leftarrow \mathsf{P}_0^{\mathsf{OR}}(x, w_\alpha, \alpha, \mathtt{rand}^\alpha)$
    $\mathbf{td} = \mathbf{td}.append(\mathbf{td}')$
    parse $a = (\mathtt{com}, (p_0^1, p_1^1), \dots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$
    return $(\mathsf{EquivCom}(\mathsf{pp}, b, \mathtt{com}||\mathtt{com}_1||\dots||\mathtt{com}_m, p_0, p_1, \mathsf{td}; \mathtt{rand}_1), (p_0^1, p_1^1), \dots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}), (p_0, p_1))$
else    // i.e., $d > 1$
    $a \leftarrow \mathsf{First}(x, w_\alpha, \alpha, d-1, \{\mathtt{com}_t\}_{t\in[m]}, \mathtt{rand}^\alpha, \mathbf{td})$
    parse $a = (\mathtt{com}, (p_0^1, p_1^1), \dots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}))$
    return $(\mathsf{EquivCom}(\mathsf{pp}, b, \mathtt{com}, p_0, p_1, \mathsf{td}; \mathtt{rand}_1), (p_0^1, p_1^1), \dots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}), (p_0, p_1))$
```

Figure 5: Auxiliary algorithm used to compute the first-round message of $\Pi^{\mathsf{cnf}}$.

*From $z$ and the commitment parameters $((p_0^1, p_1^1), \dots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$, $\mathsf{V}^{\mathsf{cnf}}$ recomputes the values $(a_1, \dots, a_{k/(k-\ell)})$, that are the $k/(k-\ell)$ commitments at level $d$ in the composition tree derived by $x$, and $(a_1^t, \dots, a_{k/(k-\ell)}^t)$, $t \in [m]$, that are the $k/(k-\ell)$ commitments at level $d$ in the composition tree derived by $x^t$.*

*For completeness of $\Pi_{\mathsf{OR}}$ the transcripts $((x_i, \dots, x_j), a, c, ((z', p_0^1, p_1^1), \dots, p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$, with $i = s(k-\ell)+1$, $s \in \{0, \dots, k/(k-\ell)-1\}$, and $j = i + k - \ell - 1$, are accepting transcripts. The same holds for $((x_i, \dots, x_j), a_t, c, ((z'_t, p_0^1, p_1^1), \dots, p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$, for $t \in [m]$ and $i, j$ computed as before.*

*Then from level $\log(k-\ell-1)$ to the root, $\mathsf{V}^{\mathsf{cnf}}$ checks that the commitment is computed correctly and follows from the partial equivocation of the equivocal commitment scheme. The last two points to check are that each pair of parameters belongs to $\mathcal{Y}^{\mathsf{pp}}$ and that the parameters in the first round message are the same as the last round message, followed by inspecting the code.*

**Lemma 2** $\Pi^{\mathsf{cnf}}$ *is computational special sound.*

**Proof 2 (Computational Special Soundness)** *Let $\mathcal{A}$ be an admissible adversary for the experiment of computational special soundness and* $\mathsf{Ext}_{\mathsf{OR}}$ *be the extractor of the underlying $\Pi_{\mathsf{OR}}$. Let $((x, x_1, \dots, x_m), a, c_1, c_2, (\tilde{z}^1, z_1^1, \dots, z_m^1), (\tilde{z}^2, z_1^2, \dots, z_m^2))$ be the tuple produced by $\mathcal{A}$. The extractor* $\mathsf{Ext}_{\mathsf{cnf}}$ *of our protocol $\Pi^{\mathsf{cnf}}$ calls* RecExt *(see Fig. 8) with inputs the statements $(x, x_1, \dots, x_m)$ packed as $(x, \{x_t\}_{t\in[m]})$, $d$, and the two transcripts $(a = (\tilde{a}, a_1, \dots, a_m), c, c', z = (\tilde{z}, z_1, \dots, z_m)), z' = (\tilde{z}', z'_1, \dots, z'_m))$ packed as $(a, c, \tilde{z}, \{z_t\}_{t\in[m]}, c', \tilde{z}', \{z'_t\}_{t\in[m]})$.* RecExt *will return either $\perp$, or a list containing $m+1$ elements. We now show that* RecExt *outputs a correct witness $(w, w_1, \dots, w_m)$ with overwhelming probability when* RecExt *is fed with two accepting transcripts and the correct $d$.* RecExt *recursively traverses all the pairs of composition trees by opening the commitment of each level of the trees (using the openings contained in $z$ and $z'$). Each pair of composition trees is expanded by opening only the child having the same $v_i$, with $i \in \{0, 1\}$. At each recursive call $d$ is decremented by 1. When $d = 1$, it means that we traversed all the levels sharing the same commitment parameters of the composition trees for all the statements $\{x_t\}_{t\in[m]}$ (line 4 of* RecExt*). At this*

```
$\mathsf{Third}(x, w_\alpha, \alpha, d, \{x^t\}_{t\in[m]}, \{\mathtt{com}_t\}_{t\in[m]}, \{z_t\}_{t\in[m]}, v_\alpha, c, \mathtt{rand}_\mathsf{P}, \mathbf{td})$
1 : **parse** $\mathtt{rand}_\mathsf{P} = (\mathtt{rand}^\alpha || \mathtt{rand}_0 || \mathtt{rand}_1)$
2 : **if** $\alpha \le |x|/2 : b = 0$
3 : **else** $b = 1$
4 : $(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b; \mathtt{rand}_0)$
5 : **if** $\alpha < |x|/2$ :
6 :     $x = \{x_1, \dots, x_{|x|/2}\}, \bar{x} = \{x_{|x|/2+1}, \dots, x_{|x|}\}$
7 :     **for** $t \in [m]$ :
8 :         $x^t = \{x^t_1, \dots, x^t_{|x^t|/2}\}, \bar{x}^t = \{x^t_{|x^t|/2+1}, \dots, x^t_{|x^t|}\}$
9 : **else**
10 :     $x = \{x_{|x|/2+1}, \dots, x_{|x|}\}, \bar{x} = \{x_1, \dots, x_{|x|/2}\}$
11 :     **for** $t \in [m]$ :
12 :         $\bar{x}^t = \{x^t_1, \dots, x^t_{|x|/2}\}, x^t = \{x^t_{|x^t|/2+1}, \dots, x^t_{|x^t|}\}$
13 : $v_b = v_\alpha,$
14 : $\alpha = \alpha \mod |x|$
15 : **if** $d = 1$ :
16 :     $z^* \leftarrow \mathsf{P}_1^{\mathsf{OR}}(x, (w_\alpha, \alpha), v_b, c, \mathtt{rand}_\mathsf{P})$
17 :     $v_{1-b} \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z^*)$
18 :     **for** $t \in [m]$ :
19 :         $\bar{v} \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}_t, c, z_t), v_{1-b} = v_{1-b}||\bar{v}, v_b = v_b||\mathtt{com}_t$
20 : **else**    // i.e., $d > 1$
21 :     $z^* \leftarrow \mathsf{Third}(x, w_\alpha, \alpha, d-1, \{x^t\}_{t\in[m]}, \{\mathtt{com}_t\}_{t\in[m]}, v_b, c, \mathtt{rand}^\alpha, \mathbf{td})$
22 :     $v_{1-b} \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z^*)$
23 : $\mathsf{td} = \mathbf{td}[1], \mathbf{td} = \mathbf{td}[2, \dots, \mathbf{td}.length]$
24 : $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, b, v_0, v_1, p_0, p_1, \mathsf{td}, \mathsf{aux})$    // where $\mathsf{aux}$ is computed with $\mathtt{rand}_1$
25 : **return** $(z^*, r, p_0, p_1)$
```

Figure 6: Auxiliary algorithm used to compute the first-round message of $\Pi^{\mathsf{cnf}}$.

*point, we have $m + 1$ (i.e., one per clause) different pairs of accepting transcripts of $\Pi_{\mathsf{OR}}$ with each pair sharing the same first-round message. Finally, we can call $\mathsf{Ext}_{\mathsf{OR}}$ on input each of those pairs. Thanks to the computational special soundness of $\Pi_{\mathsf{OR}}$, each one of these calls to $\mathsf{Ext}_{\mathsf{OR}}$ would return a witness for the corresponding clause with non-negligible probability. We have so far assumed that every time we call $\mathsf{Ext}_{\mathsf{OR}}$ we are actually feeding it with a pair of accepting transcripts w.r.t. the same first round message. This may not hold in the bad event that none of the two children of the nodes of the two composition trees have the same value, i.e. whenever the checks of lines 10, 19, 29 in $\mathsf{RecExt}$ go through and $\mathsf{RecExt}$ output $\perp$. However, thanks to the fixed equivocation property of the underlying $(1, 2)$-equivocal commitment scheme, such an event happens with negligible probability. For example, since we start from a pair of accepting transcripts, we are guaranteed that, regarding the checks at lines 10 and 19, $\mathsf{BindCom}(p_0, p_1, v_0||v_0^1|| \dots ||v_0^m, v_1||v_1^1|| \dots ||v_1^m; r) = \mathsf{BindCom}(p'_0, p'_1, v'_0||v'^1_0|| \dots ||v'^m_0, v'_1||v'^1_1|| \dots ||v'^m_1; r)$. Thus, thanks to the fixed equivocation property of the commitment scheme, we have that either $v_0||v_0^1|| \dots ||v_0^m = v'_0||v'^1_0|| \dots ||v'^m_0$ or $v_1||v_1^1|| \dots ||v_1^m = v'_1||v'^1_1|| \dots ||v'^m_1$ with non-negligible probability (notice that thanks to a similar argument, the check at line 29 is satisfied only with negligible probability, hence either $v_0 = v'_0$ or $v_1 = v'_1$). It follows that the pairs of accepting transcripts for all the $m + 1$ clauses are all accepting w.r.t. their common first-round message with non-negligible probability.*

$\mathsf{Verify}(x, \{x^t\}_{t\in[m]}, d, a, \tilde{z}, \{z_t\}_{t\in[m]}, c)$

```
$x = \{x_1, \ldots, x_{|x|/2}\}, \bar{x} = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
for $t \in [m]$ :
  $x^t = \{x^t_1, \ldots, x^t_{|x^t|/2}\}, \bar{x}^t = \{x^t_{|x^t|/2+1}, \ldots, x^t_{|x^t|}\}$
if $d = 1$ :
  parse $\tilde{z} = (z^*, r, p_0, p_1)$
  $a_0 = (v_0, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x, c, z^*)$
  $a_1 = (v_1, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z^*)$
  for $t \in [m]$ :
    parse $z_t = (z^*_t, r, p^t_0, p^t_1)$
    $a^t_0 = (v^t_0, (p_0^{t,1}, p_1^{t,1}), \ldots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x^t, c, z^*_t)$
    $a^t_1 = (v^t_1, (p_0^{t,1}, p_1^{t,1}), \ldots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}^t, c, z^*_t)$
  $a^* \leftarrow \mathsf{BindCom}(p_0, p_1, v_0||v^1_0|| \ldots ||v^m_0, v_1||v^1_1|| \ldots ||v^m_1, c; r)$
  parse $a = (\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}), (p_0, p_1))$
  return $(\mathtt{com} = a^* \wedge \mathsf{V}^{\mathsf{OR}}(x, a_0, c, \tilde{z}) \wedge \mathsf{V}^{\mathsf{OR}}(\bar{x}, a_1, c, \tilde{z}) \wedge$
      $(\bigwedge_{t\in[m]} (\mathsf{V}^{\mathsf{OR}}(x^t, a^t_0, c, z_t) \wedge \mathsf{V}^{\mathsf{OR}}(\bar{x}^t, a^t_1, c, z_t))), a^*, p_0, p_1)$
else    // i.e., $d > 1$
  parse $a = (\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}), (p'_0, p'_1))$
  $a' = (\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}))$
  parse $\tilde{z} = (z^*, r, p_0, p_1)$
  $(b_0, v_0, \cdot, \cdot) \leftarrow \mathsf{Verify}(x, x^t, d-1, a', z^*, \{z_t\}_{t\in[m]}, c)$
  $(b_1, v_1, \cdot, \cdot) \leftarrow \mathsf{Verify}(\bar{x}, \bar{x}^t, d-1, a', z^*, \{z_t\}_{t\in[m]}, c)$
  $a^* \leftarrow \mathsf{BindCom}(p_0, p_1, v_0, v_1, c; r)$
  if $a^* = \mathtt{com} \wedge ((p'_0, p'_1) = (p_0, p_1)) \wedge b_0 \wedge b_1 \wedge (p_0, p_1) \in \mathcal{Y}^{\mathsf{pp}}$ :
      return $(\mathsf{true}, a^*, p_0, p_1)$
  else return $(\mathsf{false}, a^*, p_0, p_1)$
```

Figure 7: Verification algorithm of $\Pi^{\mathsf{cnf}}$.

**Lemma 3** $\Pi^{\mathsf{cnf}}$ *is extended honest verifier zero knowledge.*

**Proof 3 (EHVZK)** *We build the EHVZK simulator $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}$ of $\Pi^{\mathsf{cnf}}$ as follows. The simulator $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}$ takes in input values*[15] $(x, x_1, \ldots, x_m), c, \tilde{z} = (z^*, (p_0^1, p_1^1), \ldots, (p_0^{\log k}, p_1^{\log k}), r_1, \ldots, r_{\log k})$*, a set of values* $\{z_t\}_{t\in[m]}$*, where* $z_t = (z^*_t, (p_0^{d+1}, p_1^{d+1}), \ldots, (p_0^{\log k}, p_1^{\log k}), r_{d+1}, \ldots, r_{\log k})$*.* $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}$ *computes* $d = \lfloor \log(\frac{k}{k-\ell}) \rfloor$ *and returns the output of* $\mathsf{FirstSim}(x, \{x^t\}_{t\in[m]}, d, \tilde{z}, \{z_t\}_{t\in[m]}, c)$*.*

*Let $\mathcal{D}_{1,k}$ be the third-round message distribution of protocol $\Pi_{1,k}$ (Sec. 3.3, [18]) for relation $\mathcal{R}_{1,k}$. We now define $\mathcal{D}_{\mathsf{cnf}}$, the third-round message distribution of $\Pi^{\mathsf{cnf}}$. The values $\tilde{z}, z_1, \ldots, z_m$ are sampled by the distribution $\mathcal{D}_{\mathsf{cnf}} = \{\tilde{z}, z_1, \ldots, z_m | \tilde{z} \leftarrow \mathcal{D}_{1,k}, \{z_t \leftarrow \mathcal{D}_{1,(k-\ell)}\}_{t\in[m]}\}$. The proof goes through hybrid hopping, modifying at each step the algorithm* $\mathsf{First}$ *(Fig. 5) run by the prover until we end up with the simulator. We report the code of the algorithm in each hybrid, and we highlight the changes using black squares.*

$\mathcal{H}_0$**:** *This hybrid is identical to the real game except that the prover of hybrid $\mathcal{H}_0$ takes in input $((x, x_1, \ldots x_m), (w, w_1, \ldots, w_m), c, (\tilde{z}, z_1, \ldots, z_m); \mathtt{rand})$, where $(\tilde{z}, z_1, \ldots, z_m) \in \mathcal{D}_{\mathsf{cnf}}$. The additional inputs $(\tilde{z}, z_1, \ldots,$*

[15]For clarity, we will slightly abuse the notation in representing $\tilde{z}$ by rearranging the order of its components.

```
$\mathsf{RecExt}(x, \{x^t\}_{t\in[m]}, d, a, c, \tilde{z}, \{z_t\}_{t\in[m]}, c', \tilde{z}', \{z'_t\}_{t\in[m]})$
$x = \{x_1, \dots, x_{|x|/2}\}, \bar{x} = \{x_{|x|/2+1}, \dots, x_{|x|}\}$
**for** $t \in [m]$ :
  $x^t = \{x^t_1, \dots, x^t_{|x^t|/2}\}, \bar{x}^t = \{x^t_{|x^t|/2+1}, \dots, x^t_{|x^t|}\}$
**if** $d = 1$ :
  **parse** $\tilde{z} = (z^*, r, p_0, p_1), z' = (z'^*, r', p'_0, p'_1)$
  $a_0 = (v_0, (p^1_0, p^1_1), \dots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x, c, z^*)$
  $a'_0 = (v'_0, (p_0'^{,1}, p_1'^{,1}), \dots, (p_0'^{,\log(k-\ell)}, p_1'^{,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x, c, z'^*)$
  $a_1 = (v_1, (p^1_0, p^1_1), \dots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z^*)$
  $a'_1 = (v'_1, (p_0'^{,1}, p_1'^{,1}), \dots, (p_0'^{,\log(k-\ell)}, p_1'^{,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z'^*)$
  **if** $(v_0 \neq v'_0) \wedge (v_1 \neq v'_1)$ :
    **return** $\perp$
  $(v_0 = v'_0)?w \leftarrow \mathsf{Ext}_{\mathsf{OR}}(x, a_0, c, z^*, c', z'^*) : w \leftarrow \mathsf{Ext}_{\mathsf{OR}}(\bar{x}, a_1, c, z^*, c', z'^*)$
  **for** $t \in [m]$ :
    **parse** $z_t = (z^*_t, r, p^t_0, p^t_1)$, **parse** $z'_t = (z'^*_t, r, p'^t_0, p'^t_1)$
    $a^t_0 = (v^t_0, (p_0^{t,1}, p_1^{t,1}), \dots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x^t, c, z^*_t)$
    $a'^t_0 = (v'^t_0, (p_0'^{t,1}, p_1'^{t,1}), \dots, (p_0'^{t,\log(k-\ell)}, p_1'^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x^t, c, z^*_t)$
    $a^t_1 = (v^t_1, (p_0^{t,1}, p_1^{t,1}), \dots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}^t, c, z^*_t)$
    $a'^t_1 = (v'^t_1, (p_0'^{t,1}, p_1'^{t,1}), \dots, (p_0'^{t,\log(k-\ell)}, p_1'^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}^t, c, z^*_t)$
    **if** $(v^t_0 \neq v'^t_0) \wedge (v^t_1 \neq v'^t_1)$ :
      **return** $\perp$
    $(v^t_0 = v'^t_0)?w_t \leftarrow \mathsf{Ext}_{\mathsf{OR}}(x^t, a^t_0, c, z_t, c', z'_t) : w_t \leftarrow \mathsf{Ext}_{\mathsf{OR}}(\bar{x}^t, a^t_1, c, z_t, c', z'_t)$
  **return** $[w, w_1, \dots, w_m]$
**else**    // i.e., $d > 1$
  **parse** $z = (z^*, r, p_0, p_1), z' = (z'^*, r', p'_0, p'_1)$
  $a_0 = (v_0, (p^1_0, p^1_1), \dots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x, c, z^*)$
  $a'_0 = (v'_0, (p_0'^{1}, p_1'^{1}), \dots, (p_0'^{\log(k-\ell)+d-1}, p_1'^{\log(k-\ell)+d-1})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x, c, z'^*)$
  $a_1 = (v_1, (p^1_0, p^1_1), \dots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z^*)$
  $a'_1 = (v'_1, (p_0'^{1}, p_1'^{1}), \dots, (p_0'^{\log(k-\ell)+d-1}, p_1'^{\log(k-\ell)+d-1})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z'^*)$
  **if** $(v_0 \neq v'_0) \wedge (v_1 \neq v'_1)$ :
    **return** $\perp$
  **if** $(v_0 = v'_0)$ :
    **return** $\mathsf{RecExt}(x, x^t, d-1, a_0, c, z^*, \{z_t\}_{t\in[m]}, c', z'^*, \{z'_t\}_{t\in[m]})$
  **else**
    **return** $\mathsf{RecExt}(\bar{x}, \bar{x}^t, d-1, a_1, c, z^*, \{z_t\}_{t\in[m]}, c', z'^*, \{z'_t\}_{t\in[m]})$
```

Figure 8: Auxiliary algorithm run by the extractor $\mathsf{Ext}_{\mathsf{cnf}}$ of $\Pi^{\mathsf{cnf}}$.

*$z_m$) are ignored during the execution by the prover of hybrid $\mathcal{H}_0$. Therefore the transcript is produced by running $a \leftarrow \mathsf{P}^{\mathsf{cnf}}_0((x, x_1, \dots x_m), (w, w_1, \dots, w_m); \mathtt{rand})$ and $z \leftarrow \mathsf{P}^{\mathsf{cnf}}_1((x, x_1, \dots, x_m), (w, w_1, \dots, w_m), a, c, \mathbf{td}; \mathtt{rand})$. The prover returns $(a, c, z)$. We notice that the output of $\mathcal{H}_0$ is identically distributed to the output of the real game. This is because we only give more inputs to the prover who ignores them in its execution.*

```
FirstSim$(x, \{x^t\}_{t\in[m]}, d, \tilde{z}, \{z_t\}_{t\in[m]}, c)$
$x = \{x_1, \ldots, x_{|x|/2}\}, \bar{x} = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
for $t \in [m]$ :
    $x^t = \{x^t_1, \ldots, x^t_{|x^t|/2}\}, \bar{x}^t = \{x^t_{|x^t|/2+1}, \ldots, x^t_{|x^t|}\}$
parse $\tilde{z} = (z^*, r, p_0, p_1)$
if $d = 1$ :
        $(v_0, (p^1_0, p^1_1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x, c, z^*)$,
        $(v_1, (p^1_0, p^1_1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z^*)$
    for $t \in [m]$ :
        parse $z_t = (z^*_t, r, p^t_0, p^t_1)$
        $(v^t_0, (p^{t,1}_0, p^{t,1}_1), \ldots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(x^t, c, z^*_t)$,
        $(v^t_1, (p^{t,1}_0, p^{t,1}_1), \ldots, (p_0^{t,\log(k-\ell)}, p_1^{t,\log(k-\ell)})) \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}^t, c, z^*_t)$
    $a \leftarrow \mathsf{BindCom}(\mathsf{pp}, p_0, p_1, v_0||v^1_0|| \ldots ||v^m_0, v_1||v^1_1|| \ldots ||v^m_1, c; r)$
    return $(a, (p^1_0, p^1_1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}), (p_0, p_1))$
else    // i.e., $d > 1$
    $(v_0, (p^1_0, p^1_1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1})) \leftarrow \mathsf{FirstSim}(x, x^t, d-1, z^*, \{z_t\}_{t\in[m]}, c)$
    $(v_1, (p^1_0, p^1_1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1})) \leftarrow \mathsf{FirstSim}(\bar{x}, \bar{x}^t, d-1, z^*, \{z_t\}_{t\in[m]}, c)$
    return $(\mathsf{BindCom}(\mathsf{pp}, p_0, p_1, v_0, v_1; r), (p^1_0, p^1_1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}), (p_0, p_1))$
```

Figure 9: Auxiliary algorithm used by the simulator $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}$.

**$\mathcal{H}_1$:** *This hybrid (Fig. 10) is identical to $\mathcal{H}_0$ except that we modify the interface of* First *to take in input $c$ and $(\tilde{z}, z_1, \ldots, z_m)$ and we add a call to $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z^*)$ at line 11 of* First*. The transcript is produced as follows $a \leftarrow \mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_1}((x, x_1, \ldots x_m), (w, w_1, \ldots, w_m), c; \mathtt{rand})$ and $z \leftarrow \mathsf{P}^{\mathsf{cnf}}_1((x_1, \ldots, x_m), (w_1, \ldots, w_m), a, c, \mathbf{td}; \mathtt{rand})$. Where $\mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_1}$ is identical to $\mathsf{P}^{\mathsf{cnf}}_0$ except that it calls $\mathsf{First}_{\mathcal{H}_1}$ instead of* First*. The outputs of $\mathcal{H}_0$ and $\mathcal{H}_1$ are identically distributed as the above change does not have any impact on the output's distribution.*

**$\mathcal{H}_2$:** *This hybrid (Fig. 11) is identical to $\mathcal{H}_1$ except that we remove the input parameters $\{\mathtt{com}_t\}_{t\in[m]}$ from $\mathsf{First}_{\mathcal{H}_1}$ and add the computation of $v^t_{b'}, v^t_{1-b'}$ at line 16 of $\mathsf{First}_{\mathcal{H}_1}$. $v^t_{b'}$ and $v^t_{1-b'}$ are computed using the extended simulator guaranteed by [18]. The transcript is produced as follows: $a \leftarrow \mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_2}((x, x_1, \ldots, x_m), (w, w_1, \ldots, w_m), c; \mathtt{rand})$ and $z \leftarrow \mathsf{P}^{\mathsf{cnf}}_1((x_1, \ldots, x_m), (w_1, \ldots, w_m), a, c, \mathbf{td}; \mathtt{rand})$, where $\mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_2}$ is identical to $\mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_1}$ except that it calls $\mathsf{First}_{\mathcal{H}_2}$ instead of $\mathsf{First}_{\mathcal{H}_1}$. $\mathcal{H}_1$ and $\mathcal{H}_2$ are indistinguishable thanks to the extended honest-verifier zero-knowledge property of $\Pi_{\mathsf{OR}}$. In case $\Pi_{\mathsf{OR}}$ is only computational EHVZ, the reduction is obvious and thus omitted.*

**$\mathcal{H}_3$:** *This hybrid (Fig. 12) is identical to $\mathcal{H}_2$ except that we replace the calls to* EquivCom *in $\mathsf{First}_{\mathcal{H}_2}$ with calls to* BindCom*. The transcript is produced as follows: $(a, z) \leftarrow \mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_3}((x, x_1, \ldots, x_m), (w, w_1, \ldots, w_m), c; \mathtt{rand})$, where $\mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_3}$ is identical to $\mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_2}$ except that it calls $\mathsf{First}_{\mathcal{H}_3}$ instead of $\mathsf{First}_{\mathcal{H}_2}$ and returns also the third-round message computed by $\mathsf{First}_{\mathcal{H}_3}$. $\mathcal{H}_2$ and $\mathcal{H}_3$ are indistinguishable thanks to the trapdoorness property of the commitment scheme. In case the 1-out-of-2 equivocal commitment scheme has computational trapdoorness the reduction is obvious and thus omitted.*

**$\mathcal{H}_4$:** *This hybrid (Fig. 13) is identical to $\mathcal{H}_3$ except that we replace the calls to $\Pi_{\mathsf{OR}}$ with calls to $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}$. The transcript is produced as follows $a \leftarrow \mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_4}((x, x_1, \ldots, x_m), c; \mathtt{rand})$, where $\mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_4}$ is identical to $\mathsf{P}^{\mathsf{cnf}}_{0,\mathcal{H}_3}$ except that it calls $\mathsf{First}_{\mathcal{H}_4}$ instead of $\mathsf{First}_{\mathcal{H}_3}$. $z$ is equal to the tuple $(z, z_1, \ldots, z_m)$ taken in input by the prover. $\mathcal{H}_3$ and $\mathcal{H}_4$ are indistinguishable thanks to the extended EHVZK property of $\Pi_{\mathsf{OR}}$. In* *case $\Pi_{\mathsf{OR}}$ is only computational EHVZ, the reduction is obvious and thus omitted.*

$\mathcal{H}_5$**:** *This hybrid is identical to $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}$. Notice that $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}$ only differs from $\mathcal{H}_4$ since $b$ is not used at all in* FirstSim*. The computation of* $\mathsf{First}_{\mathcal{H}_4}$ *does not depend on $b$, therefore the outputs of the two hybrids are identically distributed.*

$\mathsf{First}_{\mathcal{H}_1}(x, w_b, b, d, \{\mathtt{com}_t\}_{t\in[m]}, \mathtt{rand}_{\mathsf{P}}, \mathbf{td}, \boxed{\tilde{z}, \{z_t\}_{t\in[m]}, c})$

```
1 :  $\mathbf{parse}\ \mathtt{rand}_{\mathsf{P}} = (\mathtt{rand}^b || \mathtt{rand}_0 || \mathtt{rand}_1)$
2 :  $\mathbf{if}\ b \leq |x|/2 : b' = 0$
3 :  $\mathbf{else}\ b' = 1$
4 :  $(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b'; \mathtt{rand}_0)$
5 :  $\mathbf{td}.append(\mathsf{td})$
6 :  $\boxed{\mathbf{if}\ b \leq |x|/2 :}$
7 :    $\boxed{x = \{x_1, \ldots, x_{|x|/2}\}}, \boxed{\bar{x} = \{x_{|x|/2+1}, \ldots, x_{|x|}\}}$
8 :  $\boxed{\mathbf{else}}$
9 :    $\boxed{\bar{x} = \{x_1, \ldots, x_{|x|/2}\}}, \boxed{x = \{x_{|x|/2+1}, \ldots, x_{|x|}\}}$
10 : $b = b \mod |x|$
11 : $\mathbf{if}\ d = 1 :$
12 :   $(a_{b'}, \mathbf{td}') \leftarrow \mathsf{P}^{\mathsf{OR}}_0(x, w_b, b, \mathtt{rand}^b)$
13 :   $\boxed{z_{\mathsf{curr}} \leftarrow \mathsf{P}^{\mathsf{OR}}_1(x, w_b, b, a_{b'}, c, \mathtt{rand}^b)}, \boxed{a_{1-b'} \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{OR}}(\bar{x}, c, z_{curr})}$
14 :   $\mathbf{td} = \mathbf{td}.append(\mathbf{td}')$
15 :   $\mathbf{parse}\ a_{b'} = (\mathtt{com}, (p^1_0, p^1_1), \ldots, (p^{\log(k-\ell)}_0, p^{\log(k-\ell)}_1))$
16 :   $\mathbf{return}\ (\mathsf{EquivCom}(\mathsf{pp}, b', \mathtt{com}||\mathtt{com}_1|| \ldots ||\mathtt{com}_m, p_0, p_1, \mathsf{td}; \mathtt{rand}_1), (p^1_0, p^1_1), \ldots, (p^{\log(k-\ell)}_0, p^{\log(k-\ell)}_1), (p_0, p_1))$
17 : $\mathbf{else}$  // i.e., $d > 1$
18 :   $\boxed{\mathbf{parse}\ \tilde{z} = (z^*, \cdot, \cdot, \cdot)}$
19 :   $\boxed{a \leftarrow \mathsf{First}_{\mathcal{H}_1}(x, w_b, b, d-1, \{\mathtt{com}_t\}_{t\in[m]}, \mathtt{rand}^b, \mathbf{td}, z^*, \{z_t\}_{t\in[m]}, c)}$
20 :   $\mathbf{parse}\ a = (\mathtt{com}, (p^1_0, p^1_1), \ldots, (p^{\log(k-\ell)+d-1}_0, p^{\log(k-\ell)+d-1}_1))$
21 :   $\mathbf{return}\ (\mathsf{EquivCom}(\mathsf{pp}, b, \mathtt{com}, p_0, p_1, \mathsf{td}; \mathtt{rand}_1), (p^1_0, p^1_1), \ldots, (p^{\log(k-\ell)+d-1}_0, p^{\log(k-\ell)+d-1}_1), (p_0, p_1))$
```

Figure 10: Algorithm $\mathsf{First}_{\mathcal{H}_1}$ of hybrid $\mathcal{H}_1$.

## 5 Our $(\tau, k, m)$-CNF Composition

We can now deal with conjunctions of proofs of partial knowledge, meaning that for each clause the prover has to know at least $\tau$ witnesses for different base statements within the clause, out of all $k$ literals that are in the clause. In particular, we repeat our protocol $\Pi^{\mathsf{cnf}}$ $\tau$ times and then prove that a different witness for each clause is used in each of the $\tau$ executions. We call such a proof $\Pi^{\mathsf{ord\text{-}cnf}}$. Let us first describe how we instantiate our ordering protocol $\Pi^{\mathsf{ord\text{-}cnf}}$, built upon $\Pi^{\mathsf{ord}}$ of [5]. Let us consider the $i$-th execution (out of $\tau$) of our $\Pi^{\mathsf{cnf}}$. The prover algorithm of the $i$-th execution will produce $m$ lists of commitment parameters, where the first $d$ values of each of these lists (which we call $\mathbf{q}_i$) are related to the parameters re-used across the $m$ clauses, while the remaining parameters of each of the $m$ clauses (which we call $\mathbf{s}^t_i$ for $t \in [m]$) differ

```
$\mathsf{First}_{\mathcal{H}_2}(x, w_b, b, d, \mathtt{rand}_{\mathsf{P}}, \mathbf{td}, \tilde{z}, \{z_t\}_{t\in[m]}, c)$
$\textbf{parse } \mathtt{rand}_{\mathsf{P}} = (\mathtt{rand}^b || \mathtt{rand}_0 || \mathtt{rand}_1)$
$\textbf{if } b \le |x|/2 : b' = 0$
$\textbf{else } b' = 1$
$(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b'; \mathtt{rand}_0)$
$\mathbf{td}.\mathit{append}(\mathsf{td})$
$\textbf{if } b \le |x|/2 :$
    $x = \{x_1, \ldots, x_{|x|/2}\}, \bar{x} = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
$\textbf{else}$
    $\bar{x} = \{x_1, \ldots, x_{|x|/2}\}, x = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
$b = b \mod |x|$
$\textbf{if } d = 1 :$
    $(a_{b'}, \mathbf{td}') \leftarrow \mathsf{P}_0^{\mathsf{OR}}(x, w_b, b, \mathtt{rand}^b)$
    $z_{\mathsf{curr}} \leftarrow \mathsf{P}_1^{\mathsf{OR}}(x, w_b, b, c, \mathtt{rand}^b), a_{1-b'} \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(\bar{x}, c, z_{\mathsf{curr}})$
    $\boxed{\textbf{for } t \in [m] :}$
        $\boxed{\textbf{parse } z_t = (z_t^*, r, p_0^t, p_1^t)},$
    $\boxed{(v_{b'}^t, \cdot, \ldots, \cdot) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(x^t, c, z_t^*)}, \quad \boxed{(v_{1-b'}^t, \cdot, \ldots, \cdot) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(\bar{x}^t, c, z_t^*)}$
    $\mathbf{td} = \mathbf{td}.\mathit{append}(\mathbf{td}')$
    $\textbf{parse } a_{b'} = (\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$
    $\boxed{\textbf{return } (\mathsf{EquivCom}(\mathsf{pp}, b', \mathtt{com}_{b'} || v_{b'}^1 || \ldots || v_{b'}^m, p_0, p_1, \mathsf{td}; \mathtt{rand}_1), (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}), (p_0, p_1))}$
$\textbf{else}$    // i.e., $d > 1$
    $\textbf{parse } \tilde{z} = (z^*, \cdot, \cdot, \cdot)$
    $a_{b'} \leftarrow \mathsf{First}_{\mathcal{H}_2}(x, w_b, b, d-1, \{v_{b'}^t\}_{t\in[m]}, \mathtt{rand}^b, \mathbf{td}, z^*, \{z_t\}_{t\in[m]}, c)$
    $\boxed{a_{1-b'} \leftarrow \mathsf{First}_{\mathcal{H}_2}(\bar{x}, w_b, b, d-1, \{v_{1-b'}^t\}_{t\in[m]}, \mathtt{rand}^b, \mathbf{td}, z^*, \{z_t\}_{t\in[m]}, c)}$
    $\textbf{parse } a_{b'} = (\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}))$
    $\textbf{return } (\mathsf{EquivCom}(\mathsf{pp}, b', \mathtt{com}, p_0, p_1, \mathsf{td}; \mathtt{rand}_1), (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}), (p_0, p_1))$
```

Figure 11: Algorithm $\mathsf{First}_{\mathcal{H}_2}$ of hybrid $\mathcal{H}_2$.

from each other. It suffices for us to separately prove the two following properties. For the first clause of each of the $\tau$ executions, we prove that $(\mathbf{q}_1||\mathbf{s}_1^1, \ldots, \mathbf{q}_\tau||\mathbf{s}_\tau^1)$ are ordered (i.e., we are proving that we satisfy the first clause of the CNF $\tau$ times, using a different witness every time). For each of the consecutive pair of executions of $\Pi^{\mathsf{cnf}}$ on indexes $i$ and $i+1$ for $i \in [\tau - 1]$ and for each clause $t \in [2, m]$, we prove that either the commitment parameters related to the shared prefixes $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$ satisfy the ordering relation, or that it is satisfied by the suffixes $\mathbf{s}_i^t$ and $\mathbf{s}_{i+1}^t$. Let $\mathbf{qw}_i$ and $\mathbf{sw}_i^t$ be the witnesses related to $\mathbf{q}_i$ and $\mathbf{s}_i^t$ respectively. The resulting relation $\mathcal{R}_{\mathsf{ord\text{-}cnf}}((\{\mathbf{q}_i\}_{i\in[\tau]}, \{\mathbf{s}_i^1, \ldots, \mathbf{s}_i^m\}_{i\in[\tau]}), (\{\mathbf{qw}_i\}_{i\in[\tau]}, \{\mathbf{sw}_i^1, \ldots, \mathbf{sw}_i^m\}_{i\in[\tau]})$ follows:

$$\mathcal{R}_{\mathsf{ord}}((\mathbf{q}_1||\mathbf{s}_1^1, \ldots, \mathbf{q}_\tau||\mathbf{s}_\tau^1), (\mathbf{qw}_1||\mathbf{sw}_1^1, \ldots, \mathbf{qw}_\tau||\mathbf{sw}_\tau^1))\wedge$$
$$\bigwedge_{i=1}^{\tau-1} \left( \mathcal{R}_{\mathsf{ord}}((\mathbf{q}_i, \mathbf{q}_{i+1}), (\mathbf{qw}_i, \mathbf{qw}_{i+1})) \vee \left( \bigwedge_{t=2}^{m} \mathcal{R}_{\mathsf{ord}}((\mathbf{s}_i^t, \mathbf{s}_{i+1}^t), (\mathbf{sw}_i^t, \mathbf{sw}_{i+1}^t)) \right) \right).$$

A stackable $\Sigma$-Protocol $\Pi^{\mathsf{ord\text{-}cnf}}$ for $\mathcal{R}_{\mathsf{ord\text{-}cnf}}$ can be straightforwardly obtained via simple AND/OR composition of $\Pi^{\mathsf{ord}}$. Notice that as we compose $\Pi^{\mathsf{ord}}$ with the cross-stacking compiler of [18], $\Pi^{\mathsf{ord\text{-}cnf}}$ preserves

```
$\mathsf{First}_{\mathcal{H}_3}(x, w_b, b, d, \mathtt{rand}_\mathsf{P}, \mathbf{td}, \tilde{z}, \{z_t\}_{t\in[m]}, c)$
1 :  parse $\mathtt{rand}_\mathsf{P} = (\mathtt{rand}^b || \mathtt{rand}_0 || \mathtt{rand}_1)$
2 :  if $b \le |x|/2 : b' = 0$
3 :  else $b' = 1$
4 :  $(p_0, p_1, \mathsf{td}) \leftarrow \mathsf{Gen}(\mathsf{pp}, b'; \mathtt{rand}_0)$
5 :  $\mathbf{td}.append(\mathsf{td})$
6 :  if $b \le |x|/2$ :
7 :    $x = \{x_1, \ldots, x_{|x|/2}\}, \bar{x} = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
8 :  else
9 :    $\bar{x} = \{x_1, \ldots, x_{|x|/2}\}, x = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
10 : $b = b \mod |x|$
11 : if $d = 1$ :
12 :   $(a_{b'}, \mathbf{td}') \leftarrow \mathsf{P}_0^{\mathsf{OR}}(x, w_b, b, \mathtt{rand}^b)$
13 :   $z_{curr} \leftarrow \mathsf{P}_1^{\mathsf{OR}}(x, w_b, b, c, \mathtt{rand}^b), a_{1-b'} \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(\bar{x}, c, z_{curr})$
14 :   for $t \in [m]$ :
15 :     parse $z_t = (z_t^*, r, p_0^t, p_1^t)$
16 :     $(v_{b'}^t, \cdot, \ldots, \cdot) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(x^t, c, z_t^*), (v_{1-b'}^t, \cdot, \ldots, \cdot) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(\bar{x}^t, c, z_t^*)$
17 :   $\mathbf{td} = \mathbf{td}.append(\mathbf{td}')$
18 :   parse $a_{0'} = (\mathtt{com}_0, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$
19 :   parse $a_{1'} = (\mathtt{com}_1, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}))$
20 :   $\boxed{\textbf{return } (\mathsf{BindCom}(\mathsf{pp}, p_0, p_1, \mathtt{com}_0 || v_0^1 || \ldots || v_0^m, \mathtt{com}_1 || v_1^1 || \ldots || v_1^m, c; r), (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}), (p_0, p_1))}$
21 : else   // i.e., $d > 1$
22 :   parse $\tilde{z} = (z^*, \cdot, \cdot, \cdot)$
23 :   $a_{b'} \leftarrow \mathsf{First}_{\mathcal{H}_3}(x, w_b, b, d-1, \{a_t\}_{t\in[m]}, \mathtt{rand}^b, \mathbf{td}, z^*, \{z_t\}_{t\in[m]}, c)$
24 :   $a_{1-b'} \leftarrow \mathsf{First}_{\mathcal{H}_3}(\bar{x}, w_b, b, d-1, \{a_t\}_{t\in[m]}, \mathtt{rand}^b, \mathbf{td}, z^*, \{z_t\}_{t\in[m]}, c)$
25 :   parse $a_0 = (\mathtt{com}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}))$
26 :   parse $a_1 = (\mathtt{com}_1, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}))$
27 :   $\boxed{\textbf{return } (\mathsf{BindCom}(\mathsf{pp}, p_0, p_1, \mathtt{com}_0, \mathtt{com}_1; \mathtt{rand}_1), (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}), (p_0, p_1))}$
```

Figure 12: Algorithm $\mathsf{First}_{\mathcal{H}_3}$ of hybrid $\mathcal{H}_3$.

the same properties flavour of $\Pi^{\mathsf{ord}}$ (i.e., computational special soundness and perfect EHVZK). In Fig. 14, we give a detailed description of our compiler that uses $\Pi^{\mathsf{cnf}}$ of Sec. 4 and $\Pi^{\mathsf{ord\text{-}cnf}}$ to construct a stackable $\Sigma$-protocol $\Pi^{\mathsf{cnf}}_{\tau,k}$ for the relation $\mathcal{R}^{\mathsf{cnf}}_{\tau,k} = \{((x_1, \ldots x_m), (w_1, \ldots, w_m)) : \forall\, i \in [m] : (x_i, w_i) \in \mathcal{R}_{\tau,k}\}$, where $\mathcal{R}_{\tau,k} = \{((x_1, \ldots, x_k), ((w_1, \alpha_1), \ldots, (w_t, \alpha_t))) \mid 1 \le \alpha_1 < \ldots < \alpha_t \le k \wedge \forall\, j \in [t] : (x_{\alpha_j}, w_j) \in \mathcal{R}_{\mathcal{L}}\}$. Informally, $\Pi^{\mathsf{cnf}}_{\tau,k}$ is special sound because the first comparison on the full composition trees (i.e., $\mathcal{R}_{\mathsf{ord}}((\mathbf{q}_1||\mathbf{s}_1^1, \ldots, \mathbf{q}_\tau||\mathbf{s}_\tau^1), (\mathbf{qw}_1||\mathbf{sw}_1^1, \ldots, \mathbf{qw}_\tau||\mathbf{sw}_\tau^1)))$ guarantees that $\mathbf{q}_1 \le \ldots \le \mathbf{q}_\tau$. Then, in the case $\mathbf{q}_i = \mathbf{q}_{i+1}$ the prover can prove that all the pairs of suffixes related to the same clause in two different executions (i.e., $\mathbf{s}_i^t < \mathbf{s}_{i+1}^t, i \in [\tau-1], t \in [2, m]$) satisfy an order relation[16].

[16]To simplify our description, we implicitly assumed that all the clauses are already arranged so that $(\mathbf{q}_1||\mathbf{s}_1^1 < \ldots < \mathbf{q}_\tau||\mathbf{s}_\tau^1)$. However, the prover can always sort the $\tau$ sequences (according to the trapdoors it holds) and communicate this new ordering to the verifier succinctly. Similarly, when comparing $\mathbf{s}_i^t$ with $\mathbf{s}_{i+1}^t, i \in [\tau-1], t \in [2, m]$, the prover can also prove $\mathbf{s}_{i+1}^t < \mathbf{s}_i^t$ if needed. In the case $\mathbf{s}_i^t = \mathbf{s}_{i+1}^t$, the prover will pick a random order for the comparison involving the two sequences of parameters. Notice that this comparison is part of a disjunction. Therefore, in this case, a witness for the other branch (i.e., $\mathbf{q}_i, \mathbf{q}_{i+1}$) will

$\mathsf{First}_{\mathcal{H}_4}(x, b, d, z, \{z_t\}_{t\in[m]}, c)$

```
1 : if $b \leq |x|/2 : b' = 0$
2 : else $b' = 1$
3 : if $b \leq |x|/2$ :
4 :   $x = \{x_1, \ldots, x_{|x|/2}\}, \bar{x} = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
5 : else
6 :   $\bar{x} = \{x_1, \ldots, x_{|x|/2}\}, x = \{x_{|x|/2+1}, \ldots, x_{|x|}\}$
7 : $b = b \mod |x|$
8 : parse $z = (z^*, r, p_0^*, p_1^*)$
9 : if $d = 1$ :
10 :   $\boxed{(v_{b'}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)})) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(x, c, z^*)}, (v_{1-b'}, \cdot, \ldots, \cdot) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(\bar{x}, c, z^*)$
11 :   for $t \in [m]$ :
12 :     parse $z_t = (z_t^*, r, p_0^t, p_1^t)$
13 :     $(v_{b'}^t, \cdot, \ldots, \cdot) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(x^t, c, z_t^*), (v_{1-b'}^t, \cdot, \ldots, \cdot) \leftarrow \mathcal{S}_{\mathsf{OR}}^{\mathsf{EHVZK}}(\bar{x}^t, c, z_t^*)$
14 :   return $(\mathsf{BindCom}(\mathsf{pp}, p_0^*, p_1^*, v_0||v_0^1||\ldots||v_0^m, v_1||v_1^1||\ldots||v_1^m, c; r), (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)}, p_1^{\log(k-\ell)}), (p_0^*, p_1^*))$
15 : else   // i.e., $d > 1$
16 :   $a_{b'} \leftarrow \mathsf{First}_{\mathcal{H}_4}(x, b, d-1, \{a_t\}_{t\in[m]}, z^*, \{z_t\}_{t\in[m]}, c)$
17 :   $a_{1-b'} \leftarrow \mathsf{First}_{\mathcal{H}_4}(\bar{x}, b, d-1, \{a_t\}_{t\in[m]}, z^*, \{z_t\}_{t\in[m]}, c)$
18 :   parse $a_{b'} = (v_{b'}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}))$
19 :   parse $a_{1-b'} = (v_{1-b'}, (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}))$
20 :   return $(\mathsf{BindCom}(\mathsf{pp}, p_0^*, p_1^*, v_0', v_1'; r), (p_0^1, p_1^1), \ldots, (p_0^{\log(k-\ell)+d-1}, p_1^{\log(k-\ell)+d-1}), (p_0^*, p_1^*))$
```

Figure 13: Algorithm $\mathsf{First}_{\mathcal{H}_4}$ of hybrid $\mathcal{H}_4$.

**Communication complexity.** The communication complexity of $\Pi^{\mathsf{cnf}}_{\tau,k}$ can be obtained by multiplying the complexity of $\Pi^{\mathsf{cnf}}$ $\tau$ times, that is $O(\tau(\log k + m \cdot (\gamma(\Sigma) + \log(k-\ell)))))$ and adding the complexity of $\Pi^{\mathsf{ord\text{-}cnf}}$, that is $O(\tau(\log k + \max\{\log(\frac{k}{k-\ell}), m \cdot \log(k-\ell))\})$ (i.e., applying the cross-stacking compiler of [18]). If we consider $m \geq \log k$ and $\ell \geq k - \log k$, the complexity of $\Pi^{\mathsf{ord\text{-}cnf}}$ is $O(\tau \cdot \log k \cdot \log\log k)$, while the complexity of $\Pi^{\mathsf{cnf}}$ is $O(\gamma(\Sigma) \log k + (\log k \cdot \log\log k))$. It follows that the complexity of $\Pi^{\mathsf{cnf}}_{\tau,k}$ is $O(\tau(\log k \cdot \log\log k))$.

**Theorem 2** *Assuming that $\Pi^{\mathsf{cnf}}$ (Fig. 4) and $\Pi^{\mathsf{ord\text{-}cnf}}$ are stackable $\Sigma$-protocols as in Def. 5 and that the commitment scheme used in $\Pi^{\mathsf{cnf}}$ is a 1-out-of-2 equivocal commitment scheme as defined in Sec. 3.1, $\Pi^{\mathsf{cnf}}_{\tau,k}$ (Fig. 14) is a stackable $\Sigma$-protocol for the relation $\mathcal{R}^{\mathsf{cnf}}_{\tau,k}$. If $\Pi^{\mathsf{ord\text{-}cnf}}$ is perfect EHVZK, then $\Pi^{\mathsf{cnf}}_{\tau,k}$ preserves the EHVZK flavor of $\Pi^{\mathsf{cnf}}$.*

We prove Thm. 2 using lemmas Lem. 4, Lem. 5, Lem. 6 for which we require the same assumptions stated in Thm. 2.

**Lemma 4** $\Pi^{\mathsf{cnf}}_{\tau,k}$ *is complete.*

**Proof 4 (Completeness)** *It follows from the completeness of $\Pi^{\mathsf{cnf}}$ and $\Pi^{\mathsf{ord\text{-}cnf}}$. The completeness of $\Pi^{\mathsf{ord\text{-}cnf}}$ follows from the completeness of $\Pi^{\mathsf{ord}}$.*

**Lemma 5** $\Pi^{\mathsf{cnf}}_{\tau,k}$ *is computationally special sound.*

be used to satisfy the proof.

Let $\Pi^{\mathsf{cnf}} = (\mathsf{P}_0^{\mathsf{cnf}}, \mathsf{P}_1^{\mathsf{cnf}}, \mathsf{V}^{\mathsf{cnf}})$ be the stackable $\Sigma$-protocol of Fig. 4 and $\Pi^{\mathsf{ord\text{-}cnf}} = (\mathsf{P}_0^{\mathsf{ord\text{-}cnf}}, \mathsf{P}_1^{\mathsf{ord\text{-}cnf}}, \mathsf{V}^{\mathsf{ord\text{-}cnf}})$ be the stackable $\Sigma$-protocol for $\mathcal{R}_{\mathsf{ord\text{-}cnf}}$. The following compiler produces a stackable $\Sigma$-protocol $\Pi^{\mathsf{cnf}}_{\tau,k} = (\mathsf{P}_0^{(\tau,k)\text{-}\mathsf{cnf}}, \mathsf{P}_1^{(\tau,k)\text{-}\mathsf{cnf}}, \mathsf{V}^{(\tau,k)\text{-}\mathsf{cnf}})$ for relation $\mathcal{R}^{\mathsf{cnf}}_{\tau,k} = \{((x_1,\dots x_m),(w_1,\dots,w_m)) :\ \forall\, i \in [m] : (x_i,w_i) \in \mathcal{R}_{\tau,k}\}$.

**First round:** The algorithm $\mathsf{P}_0^{(\tau,k)\text{-}\mathsf{cnf}}((x_1,\dots x_m),(w_1,\dots,w_m);\mathtt{rand})$ works as follows:

- Parse $\mathtt{rand}$ as $((\mathtt{rand}^i_j)_{i\in[\tau],j\in[m]}, \mathtt{rand}^{\mathsf{ord\text{-}cnf}})$.
- For $i \in [\tau]$: $(a_i, \mathbf{p}_i, \tilde{\mathbf{td}}_i) \leftarrow \mathsf{P}_0^{\mathsf{cnf}}((x_1,\dots x_m),(w_1[i],\dots,w_m[i]);(\mathtt{rand}^i_j)_{j\in[m]})$.
- For $i \in [\tau]$: parse $\mathbf{p}_i$ and $\mathbf{td}_i$ as $(\mathbf{p}_i^1,\dots,\mathbf{p}_i^m)$, $(\mathbf{td}_i^1,\dots,\mathbf{td}_i^m)$.
- For each $i \in [\tau]$ set $\mathbf{q}_i = \mathbf{p}_i^1[1,d]$, $\mathbf{qw}_i = \mathbf{td}_i^1[1,d]$, $\mathbf{s}_i^1 = \mathbf{p}_i^1[d+1,\log k]$, $\mathbf{sw}_i^1 = \mathbf{td}_i^1[d+1,\log k]$, and for $t \in [2,m]$ set $\mathbf{s}_i^t = \mathbf{p}_i^t$, $\mathbf{sw}_i^t = \mathbf{td}_i^t$.
- Compute $a_{\mathsf{ord\text{-}cnf}} \leftarrow \mathsf{P}_0^{\mathsf{ord\text{-}cnf}}((\{\mathbf{q}_i\}_{i\in[\tau]}, \{\mathbf{s}_i^1,\dots,\mathbf{s}_i^m\}_{i\in[\tau]}), (\{\mathbf{qw}_i\}_{i\in[\tau]}, \{\mathbf{sw}_i^1,\dots,\mathbf{sw}_i^m\}_{i\in[\tau]}); \mathtt{rand}^{\mathsf{ord\text{-}cnf}})$.

The prover sends $a = (a_1,\dots,a_\tau, a_{\mathsf{ord\text{-}cnf}}, \mathbf{p}_1,\dots,\mathbf{p}_\tau)$ to the verifier.

**Second round:** The verifier samples a challenge $c \leftarrow\!\!\$\ \{0,1\}^\lambda$ and sends $c$ to the prover.

**Third round:** The algorithm $\mathsf{P}_1^{(\tau,k)\text{-}\mathsf{cnf}}((x_1,\dots x_m),(w_1,\dots,w_m),a,c;\mathtt{rand})$ works as follows:

- For $i \in [\tau]$: $(z_i) \leftarrow \mathsf{P}_1^{\mathsf{cnf}}((x_1,\dots x_m),(w_1[i],\dots,w_m[i]);(\mathtt{rand}^i_j)_{j\in[m]})$.
- Compute $z_{\mathsf{ord\text{-}cnf}} \leftarrow \mathsf{P}_0^{\mathsf{ord\text{-}cnf}}((\{\mathbf{q}_i\}_{i\in[\tau]}, \{\mathbf{s}_i^1,\dots,\mathbf{s}_i^m\}_{i\in[\tau]}), (\{\mathbf{qw}_i\}_{i\in[\tau]}, \{\mathbf{sw}_i^1,\dots,\mathbf{sw}_i^m\}_{i\in[\tau]}), a_{\mathsf{ord\text{-}cnf}}, c; \mathtt{rand}^{\mathsf{ord\text{-}cnf}})$.

The prover sends $z = (z_1,\dots,z_\tau, z_{\mathsf{ord\text{-}cnf}})$ to the verifier.

**Verification:** $\mathsf{V}^{(\tau,k)\text{-}\mathsf{cnf}}(x_1,\dots x_m,a,c,z)$ works as follows:

- Parse $a = (a_1,\dots,a_\tau, a_{\mathsf{ord\text{-}cnf}}, \mathbf{p}_1,\dots,\mathbf{p}_\tau)$, and $z = (z_1,\dots,z_\tau, z_{\mathsf{ord\text{-}cnf}})$.
- For each $i \in [\tau]$ check that $\mathsf{V}^{\mathsf{cnf}}(x_1,\dots x_m, a_i, c, z_i) = 1$, parse $\mathbf{p}_i$ as $(\mathbf{p}_i^1,\dots,\mathbf{p}_i^m)$, set $\mathbf{q}_i = \mathbf{p}_i^1[1,d]$, $\mathbf{s}_i^1 = \mathbf{p}_i^1[d+1,\log k]$, and for $t \in [2,m]$ set $\mathbf{s}_i^t = \mathbf{p}_i^t$.
- Check that $\mathsf{V}^{\mathsf{ord\text{-}cnf}}((\{\mathbf{q}_i\}_{i\in[\tau]}, \{\mathbf{s}_i^1,\dots,\mathbf{s}_i^m\}_{i\in[\tau]}), a_{\mathsf{ord\text{-}cnf}}, c, z_{\mathsf{ord\text{-}cnf}}) = 1$.
- If all the previous checks are successful, output 1. Otherwise, output 0.

Figure 14: Our stackable $\Sigma$-protocol for $\mathcal{R}^{\mathsf{cnf}}_{\tau,k}$.

**Proof 5 (Computational special soundness)** *Let $\mathcal{A}$ be an admissible adversary of the experiment of computational special soundness. Let $(x,a,c_1,c_2,z_1,z_2)$ be the tuple produced by $\mathcal{A}$. The tuple contains a statement $x$ and two accepting transcripts of $\Pi^{\mathsf{cnf}}_{\tau,k}$ w.r.t $x$ with common first-round message $a$. The components of the tuple can be parsed as $a = (a^1,\dots,a^\tau, a_{\mathsf{ord\text{-}cnf}})$, $z_1 = (z^1_{\mathsf{cnf},1},\dots,z^1_{\mathsf{cnf},\tau}, z_{\mathsf{ord\text{-}cnf},1})$ with $z^1_{\mathsf{cnf},i} = ((z^{*,1}_{i,j})_{j\in[m]}, \mathbf{p}_i, r^{i,1},\dots,r^{i,\log k})$, and $z_2 = (z^2_{\mathsf{cnf},1},\dots,z^2_{\mathsf{cnf},\tau}, z_{\mathsf{ord\text{-}cnf},2})$ with $z^2_{\mathsf{cnf},i} = ((z^{*,2}_{i,j})_{j\in[m]}, \bar{\mathbf{p}}_i, \bar{r}^{i,1}, \dots, \bar{r}^{i,\log k})$, with $i \in [\tau]$. We recall that $\mathbf{p}_i = [[(p_0^{1,i}, p_1^{1,i}),\dots,(p_0^{\log k,i}, p_1^{\log k,i})], [(p_0^{d,i,j}, p_1^{d,i,j}),\dots,(p_0^{\log k,i,j}, p_1^{\log k,i,j})]_{j\in[2,m]}]$ and the same holds for $\bar{\mathbf{p}}_i$.*

*Additionally, notice that the lists of parameters in the two transcripts are equal since they are part of the first-round message $a$. We now create from this pair of accepting transcripts, a pair of accepting transcripts* *for* $\Pi^{\mathsf{ord\text{-}cnf}}$.

*For each* $i \in [\tau]$ *set* $\mathbf{q}_i = \mathbf{p}_i^1[1,d]$, $\mathbf{s}_i^1 = \mathbf{p}_i^1[d+1, \log k]$, *and* $t \in [2,m]$ *set* $\mathbf{s}_i^t = \mathbf{p}_i^t$. *The two transcripts for the statement* $x_{\mathsf{ord\text{-}cnf}} = (\{\mathbf{q}_i\}_{i\in[\tau]}, \{\mathbf{s}_i^1, \ldots, \mathbf{s}_i^m\}_{i\in[\tau]})$ *for* $\Pi^{\mathsf{ord\text{-}cnf}}$ *are* $(a_{\mathsf{ord\text{-}cnf}}, c_1, z_{\mathsf{ord\text{-}cnf},1})$, $(a_{\mathsf{ord\text{-}cnf}}, c_2, z_{\mathsf{ord\text{-}cnf},2})$. *We first run* $\mathsf{Ext}_{\Pi^{\mathsf{ord\text{-}cnf}}}$ *on input such pairs of accepting transcripts sharing the same first-round message of* $\Pi^{\mathsf{ord\text{-}cnf}}$ *(i.e.,* $\mathsf{Ext}_{\Pi^{\mathsf{ord\text{-}cnf}}}(x_{\mathsf{ord\text{-}cnf}}, a_{\mathsf{ord\text{-}cnf}}, c_1, c_2, z_{\mathsf{ord\text{-}cnf},1}, z_{\mathsf{ord\text{-}cnf},2})$*). Except with negligible probability, the extractor returns a (subset of) the trapdoors of the composition tree of each clause. In particular, for the first clause, the extractor returns the* $\tau$ *sequences of trapdoors related to the merged composition trees (each of length* $\log k$*)* $(\mathbf{qw}_1||\mathbf{sw}_1^1, \ldots, \mathbf{qw}_\tau||\mathbf{sw}_\tau^1)$. *On the other hand, for the other clauses, it returns* $\tau \cdot (m-1)$ *trapdoors that can either be related to the prefix (i.e.,* $(\mathbf{qw}_i, \mathbf{qw}_{i+1})_{i\in[\tau]}$*), or to the suffix (i.e.,*$(\mathbf{sw}_i^t, \mathbf{sw}_{i+1}^t)_{i\in[\tau], t\in[2,m]}$*) of the clause itself. The extracted trapdoors sequences satisfy an order relation.*

*Notice that* $(\mathbf{qw}_1||\mathbf{sw}_1^1, \ldots, \mathbf{qw}_\tau||\mathbf{sw}_\tau^1)$ *can be used to extract the binding positions* $\alpha_{1,j}$*, with* $j \in [\tau]$*, of the first clause for the* $\tau$ *executions. Moreover, for each of the remaining* $x_2, \ldots, x_m$ *clauses we extract the binding position* $\alpha_{i,j}$*, with* $i \in [m]$ *and* $j \in [\tau]$*, either at the leaves level or at level* $d$ *of the composition tree for* $x_i$*, with* $i \in [2,m]$*. Notice that whenever we extract the trapdoors of the suffix, we automatically get the binding position at the leaves by combining them with the trapdoors extracted from the merged composition tree of the first clause, since the prefix is shared. Due to the relation proved by* $\Pi^{\mathsf{ord\text{-}cnf}}$*, all the extracted binding positions for the same clause must be different, thus pointing at a different base statement. We will extract a witness for such base statements using the pairs of accepting transcripts for* $\Pi^{\mathsf{cnf}}$ *contained in the accepting transcripts for* $\Pi^{\mathsf{cnf}}_{\tau,k}$*. Let us first deal the with case of an extracted binding position pointing at the leaves level.*

*Let* $\mathcal{E}$ *be the set of all clauses for which we extracted a binding position pointing at the leaves level. For* $i \in \mathcal{E}$ *and* $j \in [\tau]$, $\mathsf{Ext}_{\Pi^{\mathsf{cnf}}_{\tau,k}}$ *runs* $a \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{base}(x_{\alpha_{i,j}}, c_1, z^{*,1}_{j,i})$ *and* $\bar{a} \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{base}(x_{\alpha_{i,j}}, c_2, z^{*,2}_{j,i})$[17]. *If* $a = \bar{a}$*, it runs* $\mathsf{Ext}_{\Pi_{base}}(x_{\alpha_{i,j}}, a, c_1, z^{*,1}_{j,i}, c_2, z^{*,2}_{j,i})$ *and store the output of* $\mathsf{Ext}_{\Pi_{base}}$ *for* $\alpha_{1,j}$ *(i.e.,* $\tau$ *witnesses of* $\Pi_{base}$ *for the first clause). On the other end, If* $a \neq \bar{a}$, $\mathsf{Ext}_{\Pi^{\mathsf{cnf}}_{\tau,k}}$ *aborts.*

*We now prove that* $\mathsf{Ext}_{\Pi^{\mathsf{cnf}}_{\tau,k}}$ *aborts only with negligible probability. Let us focus on a fixed value of* $j \in [\tau]$*, as we can simply repeat the following argument for all the values of* $j$*. Let us assume there exists a* $x_{\alpha_i}$ *(with* $i \in [m]$*) such that* $a \neq \bar{a}$ *and let us denote by* $x_s$ *the statement of its sibling node in the composition tree. We compute* $a_s \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_i(x_s, c_1, z^{*,1}_i)$ $\bar{a}_s \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_i(x_s, c_2, z^{*,2}_i)$*. Compute* $a' \leftarrow \mathsf{BindCom}(p_0^{\log k,i}, p_1^{\log k,i}, a, a_s, r^{\log k,i})$ *and* $a'' \leftarrow \mathsf{BindCom}(p_0^{\log k,i}, p_1^{\log k,i}, \bar{a}, \bar{a}_s, \bar{r}^{i,\log k})$[18]. *If* $a' = a''$*, we break the fixed equivocation of the commitment scheme. Indeed, in the transcripts we would have recovered a 1-out-of-2 equivocal commitment that is equivocated in one position, while at the same time, we hold a trapdoor (extracted thanks to* $\Pi^{\mathsf{ord}}$*) that allows us to create a fresh commitment w.r.t. the same parameters and equivocate it in the other position.*

*If* $a' \neq a''$*, we can repeat the same step until level* $d$*. At level* $d$*, we just have to take care of the commitment parameters sharing, which effectively merges all the composition trees of the CNF into one. Let* $a'_1, \ldots, a'_m$ *be the values computed at level* $d$ *for each clause in the CNF in the first transcript and* $a''_1, \ldots, a''_m$ *be the values computed at level* $d$ *for each clause in the CNF in the second transcript. The* $a'$ *at level* $d-1$ *is computed as* $\mathsf{BindCom}(p_0^{d-1,i}, p_1^{d-1,i}, a'_1||\ldots||a'_m, a'_{s,1}||\ldots||a'_{s,m}, r^{i,d-1})$*, while the value* $a''$ *for the second transcript at level* $d-1$ *is* $\mathsf{BindCom}(p_0^{d-1,i}, p_1^{d-1,i}, a''_1||\ldots||a''_m, a''_{s,1}||\ldots||a''_{s,m}, r^{i,d-1})$*. After having dealt with this special case, we can continue the same procedure as before until we are able to break the fixed equivocation property at some level. This will happen at the root at worst, where by definition the two accepting transcripts share the same value* $a$*.*

*What remains to analyze is the extraction on the clauses* $x_i$ *such that* $i \in \{1, \ldots, m\} \setminus \mathcal{E}$*. The extraction procedure is basically the same, with the sole difference that we start from the binding position at level* $d$*. For this reason, the extended simulator is called on the disjunctive statement corresponding to that node, along with the appropriate third-round message. If* $a = \bar{a}$ *we can run the extractor of* $\Pi_{\mathsf{OR}}$ *and get the witness, while if* $a \neq \bar{a}$*, we can break the fixed equivocation property as shown before. Finally,* $\mathsf{Ext}_{\Pi^{\mathsf{cnf}}_{\tau,k}}$ *returns the witnesses*

[17]Here, with an abuse of notation, by $x_{\alpha_{i,\cdot}}$ we indicate the base statement in position $\alpha_{i,\cdot}$ within the $i$-th clause of the CNF.

[18]Without loss of generality here we are assuming that $x_s$ is always the right sibling of $x_{\alpha_i}$. In the actual extraction procedure, this depends on the value of $\alpha_i$.

*extracted for all the clauses, that thanks to the relation enforced by $\Pi^{\mathsf{ord\text{-}cnf}}$ are all related to different binding positions.*

**Lemma 6** *$\Pi^{\mathsf{cnf}}_{\tau,k}$ is extended honest verifier zero knowledge.*

**Proof 6 (EHVZK)** *Let $\mathcal{D}_{\mathsf{cnf}}$ and $\mathcal{D}_{\mathsf{ord\text{-}cnf}}$ be the third-round message distributions of $\Pi^{\mathsf{cnf}}$ and $\Pi^{\mathsf{ord\text{-}cnf}}$ respectively. We can define the third-round message distribution of $\Pi^{\mathsf{cnf}}_{\tau,k}$ as $\mathcal{D}^{\tau,k}_{\mathsf{cnf}} = \{\{z_t \leftarrow \mathcal{D}_{\mathsf{cnf}}\}_{t\in[\tau]}, z_{\mathsf{ord\text{-}cnf}} \leftarrow \mathcal{D}_{\mathsf{ord\text{-}cnf}}\}$. The simulator $\mathcal{S}^{\mathsf{cnf}}_{\tau,k}(x_1,\ldots,x_m,c,z_1\ldots,z_\tau,z_{\mathsf{ord\text{-}cnf}})$, for each $i \in [\tau]$, computes, $a_i \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}(x_1,\ldots,x_m,c,z_i)$, parses $z_i$ to get the parameter list $\mathbf{p}_i$, sets $\mathbf{q}_i = \mathbf{p}^1_i[1,d]$, $\mathbf{s}^1_i = \mathbf{p}^1_i[d+1,\log k]$, and for $t \in [2,m]$ set $\mathbf{s}^t_i = \mathbf{p}^t_i$. Then, for $j \in [m]$, computes $a^j_{ord} \leftarrow \mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{ord\text{-}cnf}}((\{\mathbf{q}_i\}_{i\in[\tau]},\{\mathbf{s}^1_i,\ldots,\mathbf{s}^m_i\}_{i\in[\tau]}),c,z^j_{ord})$, and finally returns $a = (a_1,\ldots,a_\tau,a^1_{ord},\ldots,a^m_{ord},\mathbf{p}_i)$. We prove that $\Pi_{\tau,k}$ is EHVZK.*

$\mathcal{H}_0$**:** *This is equal to the real game with honest prover, except that the prover of hybrid $\mathcal{H}_0$ takes in input $(x_1,\ldots,x_m,c,z_1\ldots,z_\tau,z_{\mathsf{ord\text{-}cnf}})$, where all the $z_1,\ldots,z_\tau,z_{\mathsf{ord\text{-}cnf}}$ are drawn from $\mathcal{D}^{\tau,k}_{\mathsf{cnf}}$. The additional inputs $c$ and $(z_1\ldots,z_\tau,z_{\mathsf{ord\text{-}cnf}})$ are ignored during the execution by the prover of hybrid $\mathcal{H}_0$. We notice that the output of $\mathcal{H}_0$ is distributed identically to the output of the real game. This is because we only give more inputs to the prover who ignores them in its execution.*

$\mathcal{H}_j$**:** *It is equal to $\mathcal{H}_{j-1}$ except that for each $j \in [m]$, $a_{\mathsf{ord\text{-}cnf}}$ is computed using $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{ord\text{-}cnf}}((\{\mathbf{q}_i\}_{i\in[\tau]},\{\mathbf{s}^1_i,\ldots,\mathbf{s}^m_i\}_{i\in[\tau]}),c,z_{\mathsf{ord\text{-}cnf}})$, where are defined as in the simulator reported above, while $c$ and $z_{\mathsf{ord\text{-}cnf}}$ are taken from the prover's additional input specified in $\mathcal{H}_0$. Recall that $z_i$, for $i \in [\tau]$, contains also $\mathbf{p}_i$. Clearly, $\mathcal{H}_j$ and $\mathcal{H}_{j-1}$ are indistinguishable thanks to the EHVZK property of $\Pi^{\mathsf{ord\text{-}cnf}}$.*

$\mathcal{H}_{m+i}$**:** *For each $i \in [\tau]$, this is equal to $\mathcal{H}_{m+i-1}$ except that all values $a_1,\ldots,a_i$ are computed using $\mathcal{S}^{\mathsf{EHVZK}}_{\mathsf{cnf}}(x_1,\ldots,x_m,c,z_i)$ where $c$ and $z_i$ are taken from the prover's additional input specified in $\mathcal{H}_0$. Clearly, $\mathcal{H}_{m+i}$ and $\mathcal{H}_{m+i-1}$ are indistinguishable thanks to the EHVZK property of $\Pi^{\mathsf{cnf}}$. In case $\Pi^{\mathsf{cnf}}$ is only computational EHVZK, the reduction is obvious and thus omitted. Notice that the final hybrid is identical to the simulator.*

## Acknowledgements

Gennaro Avitabile received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), and from the Spanish Government under projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00). The last two projects are co-funded by European Union EIE, and NextGenerationEU/PRTR funds. Vincenzo Botta received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PROCONTRA (grant agreement No. 885666) and by the project PARTHENON (B53D23013000006), under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. Daniele Friolo is supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union—NextGenerationEU. Daniele Venturi is member of the Gruppo Nazionale Calcolo Scientifico Istituto Nazionale di Alta Matematica (GNCS-INdAM). His reaserch was supported by project SERICS (PE00000014) and by project PARTHENON (B53D23013000006), under the MUR National Recovery and Resilience Plan funded by the European Union—NextGenerationEU. Ivan Visconti is member of the Gruppo Nazionale Calcolo Scientifico Istituto Nazionale di Alta Matematica (GNCS-INdAM) and his research contribution on this work is financially supported under the National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.1, Call for tender No. 104 published on 2.2.2022 by the Italian Ministry of University and Research (MUR), funded by the European Union -NextGenerationEU - Project Title "PARTHENON" - CUP D53D23008610006 - Grant Assignment Decree No. 959 adopted on June 30, 2023 by the Italian Ministry of Ministry of University and Research (MUR).

## References

[1] Masayuki Abe, Miguel Ambrona, Andrej Bogdanov, Miyako Ohkubo, and Alon Rosen. Non-interactive composition of sigma-protocols via share-then-hash. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 749–773, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[2] Masayuki Abe, Miguel Ambrona, Andrej Bogdanov, Miyako Ohkubo, and Alon Rosen. Acyclicity programming for sigma-protocols. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 435–465, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

[3] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 415–432, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany.

[4] Thomas Attema, Ronald Cramer, and Serge Fehr. Compressing proofs of k-out-of-n partial knowledge. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 65–91, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[5] Gennaro Avitabile, Vincenzo Botta, Daniele Friolo, and Ivan Visconti. Efficient proofs of knowledge for threshold relations. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part III*, volume 13556 of *LNCS*, pages 42–62, Copenhagen, Denmark, September 26–30, 2022. Springer, Heidelberg, Germany.

[6] Carsten Baum, Alex J. Malozemoff, Marc B. Rosen, and Peter Scholl. Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 92–122, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[7] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 390–420, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.

[8] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

[9] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[10] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092, London, UK, November 11–15, 2019. ACM Press.

[11] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Improved OR-composition of sigma-protocols. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 112–141, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[12] Michele Ciampi, Giuseppe Persiano, Alessandra Scafuro, Luisa Siniscalchi, and Ivan Visconti. Online/offline OR composition of sigma protocols. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 63–92, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[13] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, University of Amsterdam, January 1997.

[14] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany.

[15] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 103–118, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[16] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[17] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose Σ-protocols for disjunctions. Cryptology ePrint Archive, Report 2021/422, 2021. https://eprint.iacr.org/2021/422.

[18] Aarushi Goel, Matthew Green, Mathias Hall-Andersen, and Gabriel Kaptchuk. Stacking sigmas: A framework to compose $\Sigma$-protocols for disjunctions. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 458–487, Trondheim, Norway, May 30 – June 3, 2022. Springer, Heidelberg, Germany.

[19] Aarushi Goel, Mathias Hall-Andersen, Gabriel Kaptchuk, and Nicholas Spooner. Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 347–378, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany.

[20] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press.

[21] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[22] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 253–280, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[23] David Heath and Vladimir Kolesnikov. Stacked garbling for disjunctive zero-knowledge proofs. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 569–598, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

[24] David Heath, Vladimir Kolesnikov, and Stanislav Peceny. Garbling, stacked and staggered - faster k-out-of-n garbled function evaluation. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 245–274, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

[25] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 143–154, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[26] Peter Bro Miltersen, Jaikumar Radhakrishnan, and Ingo Wegener. On converting CNF to DNF. *Theor. Comput. Sci.*, 347(1-2):325–335, 2005.

[27] Rotem Tsabary. Fully secure attribute-based encryption for t-CNF from LWE. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 62–85, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

[28] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. Batchman and robin: Batched and non-batched branching for interactive ZK. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1452–1466, Copenhagen, Denmark, November 26–30, 2023. ACM Press.

[29] Gongxian Zeng, Junzuo Lai, Zhengan Huang, Yu Wang, and Zhiming Zheng. DAG-$\Sigma$: A DAG-based sigma protocol for relations in CNF. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 340–370, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany.