DAG-Σ: A DAG-based Sigma Protocol for Relations in CNF

Gongxian Zeng[1], Junzuo Lai[2], Zhengan Huang[1], Yu Wang[1], and Zhiming Zheng[1,3]

[1] Peng Cheng Laboratory, Shenzhen, China
[2] College of Information Science and Technology, Jinan University, Guangzhou, China
[3] Institute of Artificial Intelligence, LMIB, NLSDE, Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, Beijing, China
gxzeng@cs.hku.hk, laijunzuo@gmail.com, zhahuang.sjtu@gmail.com, wangy12@pcl.ac.cn, zzheng@pku.edu.cn

Abstract. At CRYPTO 1994, Cramer, Damgård and Schoenmakers proposed a general method to construct proofs of knowledge (PoKs), especially for k-out-of-n partial knowledge, of which relations can be expressed in disjunctive normal form (DNF). Since then, proofs of k-out-of-n partial knowledge have attracted much attention and some efficient constructions have been proposed. However, many practical scenarios require efficient PoK protocols for partial knowledge in other forms.
In this paper, we mainly focus on PoK protocols for k-conjunctive normal form (k-CNF) relations, which have n statements and can be expressed as follows: (i) k statements constitute a clause via "OR" operations, and (ii) the relation consists of multiple clauses via "AND" operations. We propose an alternative Sigma protocol (called DAG-Σ protocol) for k-CNF relations (in the discrete logarithm setting), by converting these relations to directed acyclic graphs (DAGs). Our DAG-Σ protocol achieves less communication cost and smaller computational overhead compared with Cramer et al.'s general method.

Keywords: Sigma protocol · Proof of partial knowledge · Conjunctive normal form · Directed acyclic graph · Disjunctive normal form.

1 Introduction

Proofs of partial knowledge demonstrate the possession of certain subsets of witnesses for a given collection of statements. In 1994, Cramer, Damgård and Schoenmakers [14] showed a *general* method with access structures to construct proofs of partial knowledge for compound statements, from "atomic" *Sigma protocols* for the individual statements.

During the last decades, most works of proofs of partial knowledge [21,1,2,5] focus on k-out-of-n partial knowledge (i.e., proving knowledge of witnesses for k out of n statements). The relations of k-out-of-n partial knowledge can be

expressed in the following *disjunctive normal form (DNF)* on n statements: every k different statements are combined with operation "AND" (we call such a combination of k statements a "Type-$\wedge$ clause"), and C_n^k different Type-$\wedge$ clauses are combined with operation "OR". An informal expression when $k = 2$ and $n = 3$ is $(y_1 \wedge y_2) \vee (y_1 \wedge y_3) \vee (y_2 \wedge y_3)$, where y_1, y_2, y_3 are 3 statements, and $(y_1 \wedge y_2)$, $(y_1 \wedge y_3)$, $(y_2 \wedge y_3)$ are 3 Type-$\wedge$ clauses. We call this kind of relations *complete k-DNF relations*, since each of them contains C_n^k Type-$\wedge$ clauses for some specific k and n.

However, many practical scenarios require proofs of partial knowledge in other forms, such as a variant of the aforementioned DNF relations, which are very similar to complete k-DNF relations but the number of Type-$\wedge$ clauses is smaller than C_n^k (e.g., when $k = 2$ and $n = 3$, $(y_1 \wedge y_2) \vee (y_1 \wedge y_3)$). We call this kind of relations *incomplete k-DNF relations*.

Relations expressed in *conjunctive normal form (CNF)* are another important collection of relations in practice. For instance, many access control policies are naturally set in CNF and they have been discussed in some attribute-based encryption schemes [26,28,8,34]. Another class of examples is the collection of instances of the k-SAT problem [25], e.g., a start-up company wants to show the investors a business plan (building at least a shopping mall in every k neighbouring blocks) in a zero-knowledge manner, avoiding the business roadmap being leaked. Some other applications about relations in CNF are also mentioned in [2], e.g., proof of possession of white money, where given a transaction graph, a user proves that the money are transferred among some white organizations while preserving the organizations' pseudonymity.

In this paper, we mainly focus on *k-CNF relations*[1]: k different statements are combined with operation "OR" (similarly, we call such a combination a "Type-$\vee$" clause), and many Type-$\vee$ clauses are combined with operation "AND". An example expression when $k = 2$ and $n = 3$ is $(y_1 \vee y_2) \wedge (y_1 \vee y_3)$, where $(y_1 \vee y_2)$ and $(y_1 \vee y_3)$ are 2 Type-$\vee$ clauses.

Note that given some witnesses and statements, in order to determine whether they belongs to a k-CNF relation, one has to check every Type-$\vee$ clause. But if for a k-DNF relation, once a Type-$\wedge$ clause is satisfied, the other Type-$\wedge$ clauses do not need to be checked anymore. It seems that the above difference results in the failure of applying most approaches of Sigma protocols for complete k-DNF relations to k-CNF relations.

To the best of our knowledge, only Cramer et al. [14] shows constructions of Sigma protocols for k-CNF relations. However, it may lead to super-polynomial communication cost. Acyclicity program, proposed by Abe et al. [2], also works for k-CNF relations, but it is designed for non-interactive zero-knowledge proofs (NIZK), not Sigma protocols. More importantly, it seems impossible to transfer their scheme [2] into a standard Sigma protocol, so acyclicity program [2] does

[1] In this paper, when we refer to k-CNF relations, we usually mean incomplete k-CNF relations (i.e., the number of Type-$\vee$ clauses num is smaller than C_n^k), since complete k-CNF relations (i.e., $num = C_n^k$) can be trivially converted to complete $(n - k + 1)$-DNF relations.

not have the strengths of Sigma protocols. For example, Sigma protocols often enjoy low soundness error by design, have high efficiency relative to their generic counterparts, and are more flexible. Using the Fiat-Shamir transform [15], Sigma protocols can be transferred to NIZK, so they are widely adopted in both non-interactive algorithms [7,32,4] and interactive protocols [10,19]. Some protocols [10,19] even enjoy round complexity improvement benefit from delayed-input Sigma protocols, which can be transferred from ordinary Sigma protocols using the method in [11]. But the acyclicity program does not enjoy these advantages.

Therefore, a question is raised naturally: *Is it possible to construct a more efficient Sigma protocol for k-CNF relations?*

Our Contributions. This paper gives an affirmative answer to the above question in the discrete-logarithm (DL) setting. More concretely, we systematically study proofs of partial knowledge for k-CNF relations, showing constructions of Sigma protocols for these relations and extensions.

We firstly formally define *partial knowledge for k-CNF relations*. Then, we propose a construction of a Sigma protocol for k-CNF relations and we call it DAG-Σ protocol. More specifically, we first put forth an efficient deterministic algorithm $\mathsf{kCNFtoDAG}$ to convert a k-CNF relation to a directed acyclic graph (DAG). Then, we construct the DAG-Σ protocol by composing a collection of Schnorr's Sigma protocols [33] according to the DAG. With this approach, we succeed in reducing the size of the transcripts and improving the efficiency.

As an extension, we apply our DAG-Σ protocols to construct Sigma protocols for incomplete k-DNF relations. We prove theoretically that a Sigma protocol for incomplete k-DNF relations can be obtained from two Sigma protocols: one for k-CNF relations and the other one for complete k-DNF relations. Then we construct a Sigma protocol for incomplete k-DNF relations in the DL setting, by restricting the choices of statements.

A comparison of communication costs of some existing protocols for three kinds of relations (k-CNF, incomplete k-DNF and complete k-DNF) is shown in Table 1. To compare these schemes, we consider them in the DL setting where given a group $\mathbb{G}$ of order p, the secret (or witness) of each statement is the corresponding discrete logarithm. For the Sigma protocols (i.e., except [2]), we consider the size of the data transmitted during the communication between the prover and the verifier. For the others (i.e., [2]), we consider the proof size.

For k-CNF relations, the communication cost of our protocol (in Sec. 5.2) is $O(n-k)|\mathbb{G}| + O(|V|)|\mathbb{Z}_p^*|$. Note that V in Table 1 denotes the vertices of the DAG in our DAG-Σ protocol. A discussion on upper bound of $|V|$ shows that the size of our solution is smaller (actually is much smaller in most cases) than that of [14], which implies that our solution enjoys a better performance when compared with [14]. Although the communication cost of [2] is linear in n, it is a non-interactive protocol, so it lacks some general extensions for standard Sigma protocols as discussed before.

For incomplete k-DNF relations, only a few protocols work for them. As shown in Table 1, the communication costs of our protocol (in Sec. 6) and [1]

Table 1: Comparison of some existing protocols (in the DL setting)*

Schemes	Σ protocol?	k-CNF	incomplete k-DNF	complete k-DNF
Cramer et al.[14]	Yes	$O(k \cdot num)(\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p^*\rvert)$	$O(k \cdot num)(\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p^*\rvert)$	$O(n)(\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p^*\rvert)$
Groth et al.[21]**	Yes	\	\	$O(\log n)(\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p^*\rvert)$
Abe et al.[1]	Yes	\	$O(n)\lvert\mathbb{G}\rvert + O(num)\lvert\mathbb{Z}_p^*\rvert$	$O(n)\lvert\mathbb{G}\rvert + O(C_n^k)\lvert\mathbb{Z}_p^*\rvert$
Abe et al.[2]	No	$O(n)(\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_p^*\rvert)$	\	\
Attema et al.[5]	Yes	\	\	$O(\log(2n-k))\lvert\mathbb{G}\rvert + 4 \times \lvert\mathbb{Z}_p^*\rvert$
Goel et al.[18]	Yes	\	\	$O(k \cdot n)^{\star\star\star}$
Ours (Sec. 5.2)	Yes	$O(n-k)\lvert\mathbb{G}\rvert + O(\lvert V\rvert)\lvert\mathbb{Z}_p^*\rvert$	\	$O(k)\lvert\mathbb{G}\rvert + O(\lvert V\rvert)\lvert\mathbb{Z}_p^*\rvert^{\dagger}$
Ours (Sec. 6)‡	Yes	\	$O(n)\lvert\mathbb{G}\rvert + O(\lvert V\rvert)\lvert\mathbb{Z}_p^*\rvert$	\

* The results here are obtained by trivially applying the corresponding protocols. There are n statements and num clauses in the expression of the k-CNF or (in)complete k-DNF relations, where each clause contains k different statements. V denotes the vertices of the DAG in our DAG-Σ protocol ($|V| \leq k \cdot num$).
** The solution in [21] only works for $k = 1$.
*** [18] presents a discussion on this kind of relation and the result is directly obtained from the discussion. It involves a special commitment scheme, so we do not have $|\mathbb{G}|$ and $|\mathbb{Z}_p^*|$ here.
† The result is obtained from Remark 1.
‡ Our solution in Sec. 6 only works for special language.

are both less than [14]. In the case of $|V| < num$, our protocol (in Sec. 6) has less communication cost than that of [1].

Compared with those protocols for complete k-DNF relations with general k ([14,5]), [5] does not consider k-CNF relations, and the protocol in [14] for k-CNF relations has more communication cost than ours.

Finally, we provide an implementation of our DAG-Σ protocol based on elliptic curve groups with key size of 512 bits. It shows that our DAG-Σ protocol saves more than 95% communication costs and more than 90% running time, compared with [14], when proving the relations in our experiments.

Discussion: non-discrete-logarithm setting. In this paper, we mainly focus on the DL setting (exactly running Schnorr's Sigma protocol [33] for each statement). Our solution can be extended to non-discrete-logarithm setting. We describe the DAG-Σ protocol by using many algorithm interfaces of a modified Schnorr's Sigma protocol. If similar modification can also be applied to other non-discrete-logarithm-based Sigma protocols [30,9,24], then using the framework of our DAG-Σ protocol and embedding other non-discrete-logarithm Sigma protocols, the new protocol can work in non-discrete-logarithm setting.

Technical overview. Recall that a Sigma protocol is an interactive protocol run by a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, and during the execution, a commitment a, a challenge c and a response z are sent in turn by $\mathcal{P}$ and $\mathcal{V}$, where c is randomly picked by $\mathcal{V}$. In the literature, a composite Sigma protocol for compound NP relations is constructed by composing "atomic" Sigma protocols for the individual relations securely. Our DAG-Σ protocol follows this general idea. Generally, to run the composite Sigma protocol, $\mathcal{P}$ firstly runs each of the "atomic" Sigma protocols to generate the individual commitment a_{atm}, and then sends a to $\mathcal{V}$, where a derives from all the a_{atm}'s as per the rule of the composite protocol. After receiving a randomly sampled c from $\mathcal{V}$, $\mathcal{P}$ prepares the challenges c_{atm}'s, based on what she sees (including c), for all the "atomic" Sigma protocols to generate the responses z_{atm}'s for all statements. Finally, $\mathcal{P}$ packs the responses

z_{atm}'s and some c_{atm}'s as z (e.g., [14,1]) and sends z to $\mathcal{V}$. Correctness usually requires that having c and z, $\mathcal{V}$ can compute a result a' that equals a.

Our starting point is the most trivial solution, i.e., a contains all commitments a_{atm}'s, and z contains all challenges c_{atm}'s and all responses z_{atm}'s. Then, we show step by step how to reduce the size of the communication, i.e., reducing the numbers of a_{atm}'s in a, and the number of c_{atm}'s and z_{atm}'s in z.

Step I: reduce the number of a_{atm}'s and c_{atm}'s. Inspired by the ring signature [4], in a Type-$\vee$ clause with k statements, we take the hash value of the commitment for the $(j+1)^{th}$ statement as the challenge for the j^{th} $(1 \leq j < k)$ statement, i.e., $c_j = \mathsf{Hash}(a_{j+1})$, where c_j denotes one of c_{atm}'s and a_{j+1} denotes one of a_{atm}'s. Further, all Type-$\vee$ clauses share the challenge c picked by $\mathcal{V}$, and for each Type-$\vee$ clause, the k^{th} statement takes c as the challenge. The method leads to the following benefits.

1. Only *one* challenge is in the transcript. Following the above method, i.e., "$c_j = \mathsf{Hash}(a_{j+1})$", $\mathcal{V}$ can also compute all the challenges (i.e., c_{atm}'s for all "atomic" Sigma protocols) by himself/herself when verification, except the challenge for the k^{th} statement in each clause. Hence, only one challenge (i.e., c) needs to be transmitted, reducing the number of c_{atm}'s in z.
2. Only the commitments of the first statement in all Type-$\vee$ clauses are in the transcript, if we informally require that the verifiers in the underlying "atomic" Sigma protocols can compute a_{atm}, given the corresponding c_{atm} and z_{atm}. The reason is simple. In a Type-$\vee$ clause, given c and z_k for the k^{th} statement, if the verifier can compute a_k, then he can know c_{k-1} via $c_{k-1} = \mathsf{Hash}(a_k)$. Following the method, the verifier can compute a_1 by himself/herself. Thus, we only need to send the commitments of the first statement in all Type-$\vee$ clauses to the verifier for verification, which reduces the number of a_{atm}'s in z.

To guarantee the correctness, we employ a variant of Schnorr's Sigma protocol and following we take the proof of 1-out-of-k partial knowledge (i.e., there is only one Type-$\vee$ clause) for example to highlight the main idea.

An example relation in the DL setting is in Fig. 1, where $\mathbf{x} = (x_1, \ldots, x_k)$ and $\mathbf{y} = (y_1, \ldots, y_k)$ denote the witnesses and statements respectively, and the witness x_μ for statement y_μ is known by the prover. In Fig. 1, the prover in the first step of the Sigma protocol (i.e., $\mathcal{P}_1$) randomly picks $(z_1, \ldots, z_{k-1}, r)$ to compute $(a_1, \ldots, a_k)$, and then sends only a_1 as the commitment a to the verifier. Note that except the last statement, we take the hash value of commitment a_{j+1} $(1 \leq j < k)$ as the challenge for the j^{th} statement, i.e., $c_j = \mathsf{H}(a_{j+1})$, where $\mathsf{H} : \mathbb{G} \to \mathbb{Z}_p^*$ is a collision-resistance hash function. After receiving the challenge c from the verifier, the prover in the third step of the Sigma protocol (i.e., $\mathcal{P}_2$) computes as follows.

1. For $i = k$ to $\mu + 1$ (where the witness x_μ for statement y_μ is known to the prover), randomly re-computes the commitments for these statements by randomly picking $z'_k, \ldots, z'_{\mu+1}$, following the similar method of $\mathcal{P}_1$;

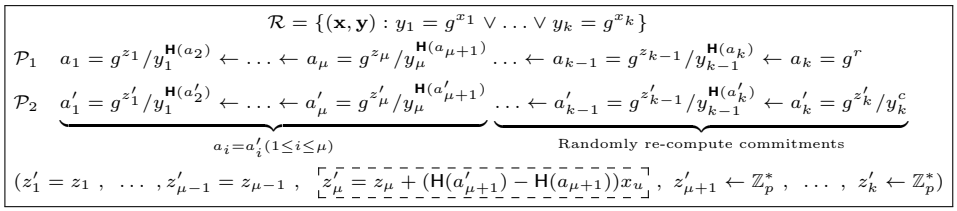


Fig. 1: An example of the proof of 1-out-of-k partial knowledge

2. For the μ^{th} statement y_μ, given the commitment a_μ, the challenge $\mathsf{H}(a_{\mu+1})$ and the witness x_μ, we can re-compute z'_μ for y_μ by the property of *Chameleon* Σ*-protocol* [12] (Schnorr's Sigma protocol is also a Chameleon Σ-protocol and more details are in Sec. 5.1), such that the value of z'_μ guarantees $a_\mu = a'_\mu$.
3. For $i = \mu - 1$ to 1, we just set $z'_i = z_i$.

A simple analysis on the correctness is presented here. Recall that for the μ^{th} statement, we have $a_\mu = a'_\mu$. Then when $i = \mu - 1$, the equal challenges $\mathsf{H}(a_{i+1}) = \mathsf{H}(a'_{i+1})$ and the equal responses $z_i = z'_i$ imply the equality of commitments $a_i = a'_i$, and the latter further implies $\mathsf{H}(a_i) = \mathsf{H}(a'_i)$. By induction on $1 \le i < \mu - 1$, we have $a = a_1 = a'_1$, which implies that the verifier will accept the proof. The detailed algorithm can be found in Sec. 5.1.

If applying the above method directly to each Type-$\vee$ clause of a k-CNF relation, then the size of the response z (resp., the commitment a) would be $O(k \cdot num)$ (resp., $O(num)$), where num is the number of Type-$\vee$ clauses. Hence, the complexity is theoretically equal to that of [14] as shown in Table 1. Therefore, we further consider to reduce the number of a_{atm}'s and z_{atm}'s.

Step II: reduce the number of a_{atm}'s and z_{atm}'s. Given a k-CNF relation, there may be many duplicate statements in different Type-$\vee$ clauses. If these duplicate statements can share the commitments a_{atm}'s and responses z_{atm}'s, then we can reduce the numbers. To this end, we convert the relation to a DAG, requiring that (i) every Type-$\vee$ clause is converted to a directed path with k vertices and each vertex represents a statement; (ii) the maximum length of paths is k, and the number of paths with length k equals the number of the Type-$\vee$ clauses num. We merge the vertices in the graph while the above requirements are preserved. For the details of the rules of merging, please refer to the transfer algorithm kCNFtoDAG in Sec. 4. Our composite Sigma protocol is run over the DAG. As a result, the size of the commitment a is $O(n - k)$, and the size of the response z is $O(|V|)$, where V is the vertex set of the DAG. Through a theoretic analysis (in Appendix C), we show that $|V| \le (k \cdot num)$, even $|V| \ll (k \cdot num)$ in most cases.

To illustrate the idea more clearly, we take the k-CNF relation in Eq. (1) for example and the relation is informally denoted as

$$(y_1 \vee y_2) \wedge (y_2 \vee y_3) \wedge (y_3 \vee y_4) \wedge (y_1 \vee y_4). \tag{1}$$

Fig. 2 is the DAG output by kCNFtoDAG when inputting the relation in Eq. (1), which has 4 directed paths, just equal to the number of the Type-$\vee$ clauses in Eq. (1). Node i, i' ($i \in [1, 4]$) represents the corresponding statement y_i. For

each Type-$\vee$ clause, we have a corresponding directed path with length k (e.g., for $(y_1 \vee y_2)$, we have path $2 \to 1$). There are 4 different statements and each has 2 duplicates in Eq. (1). Note that in Fig. 2, there is only one node representing y_1 (similar for y_4), because we merge some nodes by the algorithm $\mathsf{kCNFtoDAG}$. We also note that not all nodes corresponding to the duplicate statements can be merged, e.g., node 3 and node $3'$ for y_3.

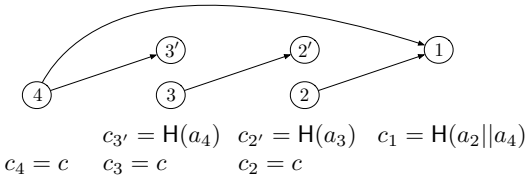


Fig. 2: An example of our scheme

Based on the DAG output by $\mathsf{kCNFtoDAG}$, we compose the "atomic" Sigma protocols for individual relations. Informally, we run a "atomic" Sigma protocol over each node in the DAG. In a nutshell, for each node, we generate a commitment for the corresponding statement of the node, and then generate a response after receiving the challenge.

Note that the DAG affects the generation of the challenges for statements. In Fig. 1, we note that the challenges are generated sequentially and only one commitment influences the computation of a challenge (i.e., informally, $c_j = \mathsf{H}(a_{j+1})$). However, in Fig. 2, there may be multiple arrows pointing to a node v (e.g., node 1). For convenience, we call the nodes that these arrows point from *the predecessor nodes* of node v. So here we have the challenge for the corresponding statement of node v being influenced by multiple commitments, which are generated for the statements of the predecessor nodes of v. More exactly, to compute the challenge, the hash function will take these commitments as the input (e.g., $c_1 = \mathsf{H}(a_2||a_4)$). For those nodes that no arrows point to, we directly take c as the challenge for their corresponding statements (e.g., $c_4 = c$). With this approach, we preserve the effect of Step I for reducing the number of a_{atm}'s and c_{atm}'s.

Related works. A general composition technique of Sigma protocol was proposed by Cramer, Damgård and Schoenmakers [14]. The idea is to secret-share the challenge according to the access structure and then use the shares as challenges in the corresponding Sigma protocols for each of the "atomic" statements. Another composition technique, to sequentially generate the challenge as we do in Step I, is introduced in [4] and recently revisited in [16,2]. Some more discussion on constructing proofs for k-CNF relations using the techniques of [14] and [2] are placed in Appendix A.

Composition is also a hot topic in NIZKs in the common reference string model. Numbers of works [20,22,3,29,31] are proposed to implement disjunctive relations for the Groth-Sahai proofs [23] and Quasi-Adaptive NIZKs [27].

Composite Sigma protocol for 1-out-of-n partial knowledge (or complete k-DNF relations) have been studied for a long time, since Cramer et al. [14] achieves

linear communication complexity. Later, Groth and Kohlweiss [21] show how to achieve logarithmic (in n) communication when $k = 1$, while Attema, Cramer and Fehr [5] achieve logarithmic communication for general k and n in the DL setting. Recently, Aarushi Goel et al. [18] propose stacking Sigmas to compose Sigma protocols for disjunctions. The resulting Sigma protocol has communication complexity proportional to the communication required by the largest clause.

Roadmap. The rest of paper is organised as follows. We review preliminaries in Sec. 2. The definition of k-CNF relations is introduced in Sec. 3 and a transfer algorithm $\mathsf{kCNFtoDAG}$ is presented in Sec. 4. We formally present the DAG-Σ protocol in Sec. 5 and an extension on incomplete k-DNF relations in Sec. 6. Finally, we show the experimental results in Sec. 7.

2 Preliminary

Notations. Throughout this paper, let λ denote the security parameter. For any $k \in \mathbb{N}$, let $[k] := \{1, 2, \cdots, k\}$. For a finite set S, we denote by $a \leftarrow S$ the process of uniformly sampling a from S. For a distribution X, we denote by $a \leftarrow X$ the process of sampling a from X. For any probabilistic polynomial-time (PPT) algorithm Alg, we write $\mathsf{Alg}(x; r)$ for the process of Alg on input x and with inner randomness r, and use $y \leftarrow \mathsf{Alg}(x)$ to denote the process of running Alg on input x and with uniformly sampled inner randomness r, and assigning y the result. We also use the symbol "$\leftarrow$" to assign the value of a variable or the result of a formula on the right-hand side to the variable on the left-hand side. We write vectors in $\mathbb{Z}_q^n$ or $\mathbb{G}^n$ in boldface, e.g., $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_q^n$. In addition, let $(a||b)$ denote the concatenation of a and b.

Sigma protocol. Let $\mathcal{R}$ be a polynomial-time-decidable binary relation. The corresponding language L consists of statement y such that there exists a witness x satisfying $(x, y) \in \mathcal{R}$. We specify L as an NP language. A Sigma protocol $\Sigma = (\mathcal{P}, \mathcal{V})$ for polynomial-time-decidable relation $\mathcal{R}$ is a three-move protocol and consists of two efficient interactive protocol algorithms $(\mathcal{P}, \mathcal{V})$, where $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2)$ is the prover and $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2)$ is the verifier, associated with a challenge space $\mathcal{CL}$. Specifically, for any $(x, y) \in \mathcal{R}$, the commitment a, the challenge c and the response z are sent in turn by the prover and verifier, where c is randomly picked over $\mathcal{CL}$ by the verifier. It enjoys completeness if for any $(x, y) \in \mathcal{R}$ and any transcript (a, c, z) output by the protocol, the verifier (i.e., $\mathcal{V}_2$) outputs 1. It also has the security requirements of knowledge soundness, special honest verifier zero knowledge (special HVZK) and witness indistinguishability. In this paper, we relax the requirement of knowledge soundness to *computational* knowledge soundness. Due to page limitations, formal definitions of these security requirements will be given in Appendix B. Without loss of generality, when there are multiple Sigma protocols, for $\Sigma = (\mathcal{P}, \mathcal{V})$, we use $\Sigma.\mathcal{P}$ and $\Sigma.\mathcal{V}$ to specify the prover and verifier of Σ, respectively.

Graphs. A directed graph is a tuple $G = (V, E)$ where V is a set of elements called vertices (or nodes) and E is a set of vertices pairs, $E \subseteq V \times V$, called

directed edges or arrows. Given an edge $e = (u, v)$, it is pointed from vertex u to vertex v, and u is called the head of e and v is called the tail of e. A cycle in G is a finite sequence of edges $(e_1, \ldots, e_l)$ satisfying that the tail of edge e_i is the head of edge e_{i+1} for $\forall i \in [l]$ (we set $e_{l+1} = e_1$). A graph with no cycles is called acyclic. Given an acyclic graph G, we define a vertex sequence $(v_1, \ldots, v_l)$ as a path, where there is an edge $e = (v_i, v_{i+1})$ for every pair of neighboring vertices (v_i, v_{i+1}) for $i \in [l-1]$. The number of edges pointed to vertex v is called the in-degree of vertex v and we denote it as $\mathsf{in\text{-}deg}(v)$. Similarly, the number of edges pointed from vertex v is called the out-degree of vertex v and we denote it as $\mathsf{out\text{-}deg}(v)$. Given a vertex v, we call it a *source* if $\mathsf{in\text{-}deg}(v) = 0$ and call it a *sink* if $\mathsf{out\text{-}deg}(v) = 0$. In addition, we define some operations for a directed acyclic graph G: (1) $\mathsf{sink}(G)$ outputs a vertex set S^{sink} that contains all sinks; (2) similarly, $\mathsf{source}(G)$ outputs a vertex set S^{source} that contains all sources; (3) for any vertex v, $\mathsf{pred}(v)$ outputs a vertex set S_v^{pred} where the elements are the head of the edges that are pointed to vertex v.

3 Definition of k-CNF relations

In this section, we formally define partial knowledge for k-CNF relations. Let y denote a statement, and S_k denote the universal set of which the elements are k-size subsets of $[n]$, i.e., $S_k := \{\{i_1, \ldots, i_k\} | \ 1 \leq i_1 < \ldots < i_k \leq n, \{i_1, \ldots, i_k\} \subset [n]\}$. Besides, $(x_l, y_l) \in \mathcal{R}_l$ ($l \in [n]$) denotes a valid witness-statement pair belonging to a relation $\mathcal{R}_l$. Then, we define the following partial knowledge for compound statements.

Definition 1. (Partial knowledge for k-CNF). *Given n different statements $(y_l)_{l\in[n]}$, n sub-relations $(\mathcal{R}_l)_{l\in[n]}$, and $S'_k \subseteqq S_k$, the prover proves that for all $\{i_1, \ldots, i_k\} \in S'_k$, she knows the witnesses for at least one of $y_{i_1}, \cdots, y_{i_k}$.*

The relation can be presented in CNF as follows,

$$\mathcal{R}_{k\text{-CNF},S'_k} = \{(\mathbf{x}, \mathbf{y}) : \wedge_{\{i_1,\ldots,i_k\}\in S'_k}(\vee_{j\in[k]}(x_{i_j}, y_{i_j}) \in \mathcal{R}_{i_j})\}, \tag{2}$$

where $\mathbf{x}$, $\mathbf{y}$ are two n-dimension vectors, and $\mathcal{R}_{i_j} \in \{\mathcal{R}_l \mid l \in [n]\}$ is a sub-relation. We call $(\vee_{j\in[k]}(x_{i_j}, y_{i_j}) \in \mathcal{R}_{i_j})$ a "Type-$\vee$" clause, where $\{i_1, \ldots, i_k\} \in S_k$. Let num denote the number of Type-$\vee$ clauses in $\mathcal{R}_{k\text{-CNF},S'_k}$, i.e., $num = |S'_k|$. Note that $num \leq C_n^k$, and we only consider polynomial-time relation, so it is required that the membership of $(\mathbf{x}, \mathbf{y})$ to $\mathcal{R}_{k\text{-CNF},S'_k}$ can be determined in polynomial time in $|\mathbf{y}|$. We denote the (*polynomial-time*) relation defined in Eq. (2) as a **k-CNF relation**.

We stress that *not* all the $\mathcal{R}_{k\text{-CNF},S'_k}$ defined in Eq. (2) can be decided in polynomial time. For example, when k is about $\frac{n}{3}$ and num is close to C_n^k, generally the complexity of determining whether $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}_{k\text{-CNF},S'_k}$ is $O(k \cdot num) = O(\frac{n}{3} \cdot C_n^{\frac{n}{3}})$, so it is super-polynomial.

In this paper, we focus on k-CNF relations that can be determined in polynomial time, e.g., (i) $|S'_k|$ is polynomial in $|\mathbf{y}|$, and (ii) k is a constant. Specifically, when $|S'_k|$ is polynomial in $|\mathbf{y}|$, the time for determining $\mathcal{R}_{k\text{-CNF},S'_k}$ is

linear in $|S'_k|$, so it is also polynomial. On the other hand, when k is a constant, $O(k \cdot num) = O(num)$, where num is polynomial in n in the worst case.

Remark 1. When $num = C_n^k$, a proof for a k-CNF relation can be transferred into a proof of $(n-k+1)$-out-of-n partial knowledge. Then there exists some trivial and efficient solutions, e.g., [5]. Thus, without loss of generality, when we refer to k-CNF relations, we usually mean "incomplete" k-CNF relations (i.e., $num < C_n^k$). It also can be inferred that a proof of k-out-of-n partial knowledge can be transferred into a proof for a $(n-k+1)$-CNF relation with C_n^{n-k+1} clauses.

Throughout this paper, we mainly focus on the discrete logarithm (DL) setting. In other words, the prover aims to convince the verifier that she knows the discrete logarithms of some statements (i.e., the group elements). Formally, let $\mathbb{G}$ be a cyclic group of order p, and g be a generator of $\mathbb{G}$. Following Def. 1 and the DL setting, we define the relation $\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}$ as follows:

$$\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k} = \{(\mathbf{x},\mathbf{y}) : \wedge_{\{i_1,\ldots,i_k\}\in S'_k}(\vee_{j\in[k]} y_{i_j} = g^{x_{i_j}})\}, \tag{3}$$

where $\mathbf{x} \in (\mathbb{Z}_p^* \cup \{\perp\})^n \setminus \{(\perp)^n\}$, $\mathbf{y} \in \mathbb{G}^n$, S'_k is defined as in Def. 1, and for all $\{i_1,\ldots,i_k\} \in S'_k$, $1 \le i_1 < \ldots < i_k \le n$. Furthermore, for any $\mathbf{x} \in (\mathbb{Z}_p^* \cup \{\perp\})^n \setminus \{(\perp)^n\}$, let $S^{\mathrm{w}}_{\mathbf{x}} := \{i \in [n] \mid y_i = g^{x_i}\}$. In other words, $S^{\mathrm{w}}_{\mathbf{x}}$ contains the indices that prover knows the corresponding witnesses.

4 Converting k-CNF relations into DAGs

Before constructing our Sigma protocol for k-CNF relations, we firstly introduce a deterministic transfer algorithm kCNFtoDAG, which can convert a k-CNF relation $\mathcal{R}_{k\text{-CNF},S'_k}$ (in Eq. (2)) to a directed acyclic graph (DAG). In Sec. 5, we will show a Sigma protocol (DAG-Σ) based on the DAG output by the algorithm kCNFtoDAG.

We require that the DAG output by kCNFtoDAG should have the following properties:

- **Property-(i):** Each node in some path corresponds to a statement in the corresponding Type-$\vee$ clause.
- **Property-(ii):** The number of paths from the nodes in S^{source} to the nodes in S^{sink} equals the number of Type-$\vee$ clauses in the expression of $\mathcal{R}_{k\text{-CNF},S'_k}$, and the lengths of these paths are k.

Furthermore, we require that the number of vertices in the DAG should be *as few as possible*. That's because in Sec. 5, we will show that the communication complexity of our DAG-Σ protocol depends on the number of the vertices of the DAG output by kCNFtoDAG.

Now, we turn to the details of algorithm kCNFtoDAG.

For simplicity, we require that the statements in each Type-$\vee$ clause are sorted from the smallest index to the largest, e.g., $\mathcal{R}_1$ in Eq. (4) (for simplicity, we use Σ to denote $(x, y) \in \mathcal{R}$).

$$\begin{aligned}\mathcal{R}_1 = \{(\mathbf{x},\mathbf{y}) : (\Sigma_1 \vee \Sigma_2 \vee \Sigma_3) \wedge (\Sigma_1 \vee \Sigma_2 \vee \Sigma_4) \\ \wedge (\Sigma_2 \vee \Sigma_3 \vee \Sigma_5) \wedge (\Sigma_3 \vee \Sigma_4 \vee \Sigma_5)\}\end{aligned} \tag{4}$$

A simple idea to implement kCNFtoDAG is to build a separate directed path for each Type-$\vee$ clause. However, it would result in $(k \cdot num)$ nodes in the graph, where num is the number of Type-$\vee$ clauses. As shown in Fig. 3, we draw a DAG for $\mathcal{R}_1$ in Eq. (4), using the simple idea. It is clear that the DAG has the above two properties, and there are totally $3 \times 4 = 12$ nodes in the graph.

To reduce the number of nodes, we consider the following method first. We scan the relation and let every statement have at most three states, i.e., beginning, middle, ending. The beginning state shows that the statement is the last statement of some Type-$\vee$ clause so the corresponding node is the head of some path. The middle state indicates that the statement is placed in the middle of some Type-$\vee$ clause. The ending state is that the statement is the first statement of some Type-$\vee$ clause (note that in Sec. 5, the prover will compute a commitment for each node, and only the commitments for the nodes indicating statements with ending state will be sent to the verifier). Then for every Type-$\vee$ clause, we have a path in G from a node indicating the beginning state of some statement to a node indicating the ending state of some statement.

Thus, we merge the nodes with the same state in Fig. 3, then obtain another DAG in Fig. 4. We use a_l, b_l, e_l ($l \in [1, 5]$) to denote the beginning, middle, ending state of the l^{th} statement respectively. When describing the DAG here, for convenience, we also use these notations (i.e., a_l, b_l and e_l) to represent the head nodes, middle nodes and tail nodes respectively. In addition, we may use superscripts to indicate different duplicate nodes (e.g., nodes b_3^1 and b_3^2 in Fig. 6 represent the different duplicates). When talking about the paths in the DAG, we sometimes write the path with nodes and arrows (e.g., for the path (a_3, b_2, e_1) in Fig. 4, we write it as $a_3 \to b_2 \to e_1$). In Fig. 4, the number of vertices is 9, which is smaller than that in Fig. 3.

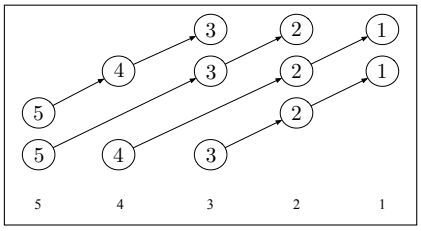


Fig. 3: A simple idea

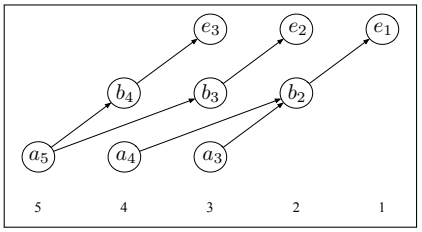


Fig. 4: An example for CNF

However, the above approach cannot handle all cases. A counter example is

$$\begin{aligned}\mathcal{R}_2 = \{(\mathbf{x},\mathbf{y}) : (\Sigma_1 \vee \Sigma_2 \vee \Sigma_3) \wedge (\Sigma_1 \vee \Sigma_2 \vee \Sigma_4) \wedge (\Sigma_1 \vee \Sigma_3 \vee \Sigma_4) \\ \wedge (\Sigma_2 \vee \Sigma_3 \vee \Sigma_5) \wedge (\Sigma_3 \vee \Sigma_4 \vee \Sigma_5)\}\end{aligned} \tag{5}$$

and we try to draw a DAG as shown in Fig. 5, using the above approach.

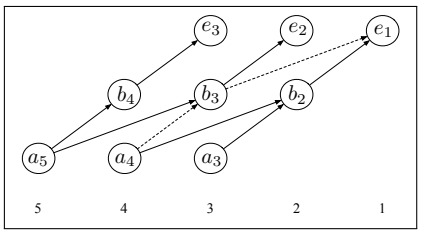


Fig. 5: A counter exam

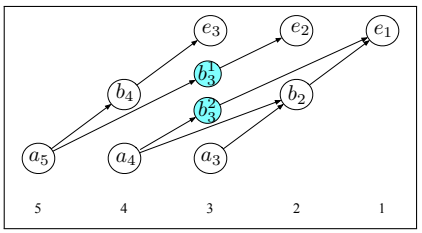


Fig. 6: A fixed graph

Compared with relation $\mathcal{R}_1$ in Eq. (4), one more Type-$\vee$ clause is added in Eq. (5) (i.e., $(\Sigma_1 \vee \Sigma_3 \vee \Sigma_4)$), and we use the dashed arrows in Fig. 5 to show the difference compared with Fig. 4. Note that there is a "crossing edge" (i.e., in node b_3) in Fig. 5. It implies two more directed paths (i.e., $a_4 \to b_3 \to e_2$ and $a_5 \to b_3 \to e_1$) are introduced in Fig. 5, while $(\Sigma_2 \vee \Sigma_3 \vee \Sigma_4)$ and $(\Sigma_1 \vee \Sigma_3 \vee \Sigma_5)$ are not in Eq. (5). Hence, the obtaining DAG does not have the above two properties. Essentially, a "wrong" crossing edge may introduce nonexistent Type-$\vee$ clauses. Thus, to output a correct DAG, a duplicate node for b_3 is needed in this case, as shown in Fig. 6.

Next, we present the formal description of algorithm kCNFtoDAG, which is constructed with the above approach. We also take relation $\mathcal{R}_2$ in Eq. (5) as an example, to show how kCNFtoDAG works step by step.

Algorithm description. Inputting a k-CNF relation $\mathcal{R}_{k\text{-CNF},S'_k}$ (in Eq. (2)), the deterministic transfer algorithm kCNFtoDAG runs in the following steps and finally outputs a DAG $G = (V, E)$:

1. **Preparing nodes.** For each Type-$\vee$ clause in $\mathcal{R}_{k\text{-CNF},S'_k}$, draw a separate directed path $(v_1, \ldots, v_k)$ with length k and each node represents a statement. For each path, we require that the indices of their corresponding statements are from the largest to the smallest. In other words, given a function $f : V \to [n]$, mapping the nodes to the indices of the corresponding statements, we have $f(v_1) > \ldots > f(v_k)$.
   As shown in Fig. 7, for every Type-$\vee$ clause of the expression of $\mathcal{R}_2$ in Eq. (5), we draw a path. There are 5 paths and 15 nodes in total. The numbers in the bottom of Fig. 7 (i.e., $5, \ldots, 1$) indicate the statements that the above nodes map to, e.g., node a_3 represents statement y_3. It is clear that given any path (v_1, v_2, v_3) in Fig. 7, the indices of the corresponding statements are in descending order, e.g., for the path which is denoted as $a_3 \to b_2^2 \to e_1^3$, we have $f(v_1) = 3 > f(v_2) = 2 > f(v_3) = 1$.
2. **Merging prefixes.** For any node v_l $(l \in [k])$ in some path $(v_1, \ldots, v_k)$, we define the *prefix* of v_l as $(v_1, \ldots, v_{l-1})$. For any v_l and v'_l, if their prefixes $(v_1, \ldots, v_{l-1})$ and $(v'_1, \ldots, v'_{l-1})$ correspond to the same statements, then for all $i \in [l-1]$, we merge v_i and v'_i into one node. Here, we merge the nodes in descending order of the indices of the statements, i.e., from the largest index to the least index.

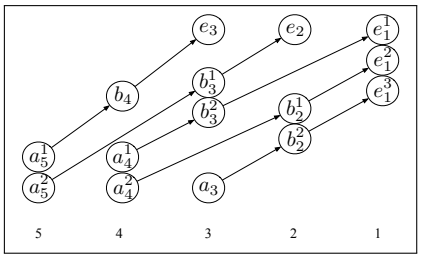


Fig. 7: Graph after step 1

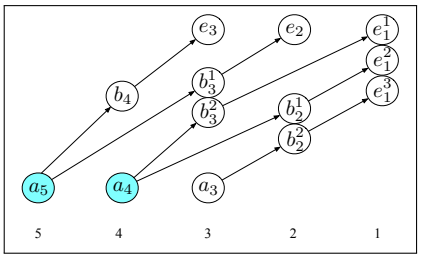


Fig. 8: Graph after step 2

For example, in Fig. 7, node b_4 (in path $a_5^1 \to b_4 \to e_3$) and node b_3^1 (in path $a_5^2 \to b_3^1 \to e_2$) have the same prefix (i.e., node a_5^1 and node a_5^2). Thus, we merge them into one node (i.e., the blue node a_5 in Fig. 8). Similarly, we merge node a_4^1 and node a_4^2 into another blue node a_4 in Fig. 8. Finally, we obtain Fig. 8 after merging prefixes and there are totally 5 paths and 13 nodes.

3. **Merging suffixes.** For any node v_l ($l \in [k]$) in some path $(v_1, \ldots, v_k)$, we define the *suffix* of v_l as $(v_{1+1}, \ldots, v_k)$. Note that a node may have multiple suffixes after merging prefixes. For any v_l and v_l', we will merge them into one node, if they satisfy the following conditions: i) they correspond to the same statement; ii) the numbers of suffixes of v_l and v_l' are the same (if the suffix is empty, the number of suffixes is 0); iii) when the numbers of suffixes are greater than 0, for each suffix of v_l, there is suffix of v_l' such that the corresponding statements of the suffixes are the same. Here, we merge the nodes in ascending order of the indices of the statements, i.e., from the least index to the largest index. Finally, output the graph G.

   In Fig. 8, the suffix of the node e_1^1 in path $a_4 \to b_3^2 \to e_1^1$, the suffix of node e_1^2 in path $a_4 \to b_2^1 \to e_1^2$ and the suffix of node e_1^3 in path $a_3 \to b_2^2 \to e_1^3$, are all empty. Thus, we merge them into one node, as the blue node e_1 in Fig. 9.

   After that, node b_2^1 (in path $a_4 \to b_2^1 \to e_1$) and node b_2^2 (in path $a_3 \to b_2^2 \to e_1$) share the same suffixes (i.e., node e_1). Thus, we merge node b_2^1 and node b_2^2 into one node (i.e., the blue node b_2 in Fig. 10). Finally, we can see that the graphs in Fig. 10 and Fig. 6 are identical. There are 5 paths and 10 nodes in total in Fig. 10, and the number of the nodes in Fig. 10 are much smaller than that in Fig. 7.

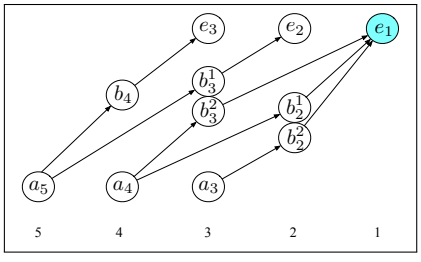


Fig. 9: Merging nodes to e_1

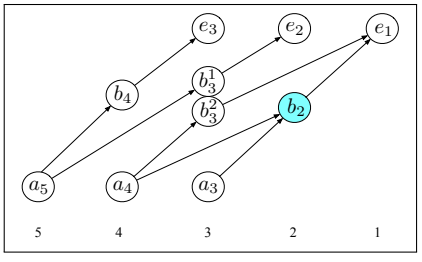


Fig. 10: Graph after step 3

That's the description of the deterministic transfer algorithm kCNFtoDAG.

Now we turn to discuss the properties that kCNFtoDAG has. Formally, we have the following two theorems. Due to space limitations, the proofs of these two theorems will be postponed in Appendix C.

Theorem 1. *Given a k-CNF relation, the DAG output by algorithm* kCNFtoDAG *has the aforementioned Property-(i) and Property-(ii).*

Theorem 2. *Given a k-CNF relation $\mathcal{R}_{k\text{-CNF},S'_k}$ for n statements, the number of vertices $|V|$ in the DAG, output by the above transfer algorithm* kCNFtoDAG*, satisfies that $|V| \leq \mathsf{Min}(V_{\text{bound}}, (k \cdot num))$, where num is the number of the clauses in the expression of $\mathcal{R}_{k\text{-CNF},S'_k}$, and*

$$V_{\text{bound}} = 2^d + 2(n-2d+1) + (n-2d+2)C_n^{\lfloor \frac{d}{2} \rfloor + 1} \begin{cases} d = k & (2 \leq k < \frac{n+1}{2}) \\ d = n-k+1 & (\frac{n+1}{2} \leq k \leq n-1) \end{cases} \tag{6}$$

In addition, if we just prepare as many nodes as the theoretical result (i.e., V_{bound}), then we can further reduce the running time and memory space when invoking kCNFtoDAG. An improved algorithm is attached in Appendix D.

Discussion on V_{bound}. Here, we also provide another method to analyze the upper bound V_{bound}, which may be more easier to understand. Given an integer n and an integer k (for simplicity, here we firstly assume that k is an even number), then we prepare $k \cdot C_n^{k/2}$ nodes in the following way:

1. Prepare two same subgraphs and each subgraph has $C_n^{k/2}$ paths with length $k/2$, so there are $2 \cdot (k/2) \cdot C_n^{k/2} = k \cdot C_n^{k/2}$ nodes in total.
2. Each path in the subgraph corresponds to a combination of $k/2$ distinct numbers chosen from $[n]$. For simplicity, the indices of the nodes in each path are the numbers of the corresponding combination sorted from smallest to the largest.

We note that these nodes can represent the expression of any k-CNF relation. For a clause $(y_1 \vee y_2 \vee \cdots \vee y_k)$ in the expression, we divide it into two parts $(y_1 \vee y_2 \vee \cdots \vee y_{k/2})$ and $(y_{k/2+1} \vee y_{k/2+2} \vee \cdots \vee y_k)$. Then we can find a path in the first subgraph, representing the first part $(y_1 \vee y_2 \vee \cdots \vee y_{k/2})$, according to the indices of the statements. Similarly, we can find a path in the second subgraph for the second part. Finally, we add an edge between them so that they form a new path with length k. After all the clauses in the expression are dealt with like this, we remove all paths with length $k/2$ and then obtain a DAG for the k-CNF relation.

It is obvious that all the clauses have the corresponding paths, and the obtained DAG satisfies the aforementioned Property-(i) and Property-(ii).

When k is an odd number, we can deal with the case by adjusting the lengths of the paths in each subgraph (i.e., let the lengths of the paths in the two subgraphs be $\lceil k/2 \rceil$ and $\lfloor k/2 \rfloor$ respectively).

It seems not easy to explore the size relation between $k \cdot C_n^{k/2}$ and V_{bound} in Eq. (6). For instance, when $k \geq (n+1)/2$, $k \cdot C_n^{k/2}$ could be larger than $V_{\text{bound}} \approx (2k-n) \cdot C_n^{\lfloor \frac{n-k+1}{2} \rfloor + 1}$ in Eq. (6), since $(2k-n) < k$ and $C_n^{\lfloor \frac{n-k+1}{2} \rfloor + 1} \leq C_n^{k/2}$ (the latter is because $\frac{n-k+1}{2} \leq k/2 < (n+1)/2$ when $k \geq (n+1)/2$). In any case, when we compare the upper bound of the number of vertices in the DAG output by kCNFtoDAG with the upper bound of the number of statements in the original expression of a k-CNF relation (i.e., $k \cdot C_n^k$), both the two kinds of analysis (i.e., $(k \cdot C_n^{k/2})$ and V_{bound}) show that our method may have a remarkable improvement.

5 DAG-Σ protocol for k-CNF

In this section, we construct a Sigma protocol for k-CNF relations. Specifically, we first show a Sigma protocol for k-CNF relations based on a Sigma protocol for 1-out-of-k relations in Sec. 5.1. Further, we convert the k-CNF relations to directed acyclic graphs (DAGs), and then show a DAG-based Sigma protocol (DAG-Σ protocol) in Sec. 5.2.

5.1 Warm-up

Here we describe a Sigma protocol for k-CNF relations. Part of the ideas will be adopted in our later DAG-Σ protocol.

Framework. Let $\mathcal{R}_{\text{1-OR}}$ be a 1-out-of-k relation in the DL setting, i.e.,

$$\mathcal{R}_{\text{1-OR}} = \{(\mathbf{x}, \mathbf{y}) : y_1 = g^{x_1} \vee \ldots \vee y_k = g^{x_k}\}, \tag{7}$$

where $\mathbf{x} \in (\mathbb{Z}_p^* \cup \{\perp\})^k \setminus \{(\perp)^k\}$ and $\mathbf{y} \in \mathbb{G}^k$. We will firstly construct a Sigma protocol $\Sigma^{\mathcal{R}_{\text{1-OR}}}$ for $\mathcal{R}_{\text{1-OR}}$. Then, with $\Sigma^{\mathcal{R}_{\text{1-OR}}}$ as an ingredient, we construct a composite Sigma protocol $\Sigma_{\text{plain}}^{\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}}$ for $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$ (Eq. (3)) in this way:

1. For each Type-$\vee$ clause in $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$, the prover $\mathcal{P}_1$ calls $\Sigma^{\mathcal{R}_{\text{1-OR}}}.\mathcal{P}_1$ to generate a commitment; then she sends all the commitments to the verifier.
2. The verifier $\mathcal{V}_1$ picks a random number from $\mathbb{Z}_p^*$ as a challenge and sends it to the prover.
3. The prover $\mathcal{P}_2$ calls $\Sigma^{\mathcal{R}_{\text{1-OR}}}.\mathcal{P}_2$ to generate responses and then sends them to the verifier.

Finally, the verifier $\mathcal{V}_2$ outputs 1 if and only if $\Sigma^{\mathcal{R}_{\text{1-OR}}}.\mathcal{V}_2$ accepts all the transcripts (for all the Type-$\vee$ clauses in $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$).

Completeness, computational knowledge soundness and special HVZK property of this composite Sigma protocol are trivially based on that of $\Sigma^{\mathcal{R}_{\text{1-OR}}}$. So we omit the analysis here, and turn to the construction of $\Sigma^{\mathcal{R}_{\text{1-OR}}}$.

Sigma protocol $\Sigma^{\mathcal{R}_{\text{1-OR}}}$. Before describing the protocol $\Sigma^{\mathcal{R}_{\text{1-OR}}}$, we firstly recall Schnorr's Sigma protocol [33] $\Sigma^{\mathcal{R}}_{\text{Sch}} = (\mathcal{P}, \mathcal{V})$ for relation $\mathcal{R} = \{(x, y) :$

$y = g^x\}$ in Fig. 11, where the description of the HVZK simulator Sim is also presented. Observe that the witness x is not needed for $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathcal{P}_1$, so we write $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathcal{P}_1(\perp, y)$ directly in Fig. 11. Note that Schnorr's Sigma protocol $\Sigma^{\mathcal{R}}_{\text{Sch}}$ is a *Chameleon* Σ-protocol [12] (see the definition in Appendix B). Generally, in a Chameleon Σ-protocol, the prover can compute the commitment a by using the simulator (taking a statement y and an arbitrary challenge c' as input). Once the challenge c has been received, the prover can compute the response z by using the witness x and the randomness which is used by the simulator to compute a. Thus, a Chameleon Σ-protocol for $\mathcal{R}$ has two modes: standard mode when $\mathcal{P}$ runs $\mathcal{P}_1$ and $\mathcal{P}_2$, and a Chameleon mode when $\mathcal{P}$ runs the simulator. It is required that for all $(x, y) \in \mathcal{R}$, the transcript output in the standard mode and that output in the Chameleon mode are indistinguishable. As pointed out in [12], $\Sigma^{\mathcal{R}}_{\text{Sch}}$ is a Chameleon Σ-protocol, so we provide another proving algorithm $\mathcal{P}' = (\mathcal{P}'_1, \mathcal{P}'_2)$ for $\Sigma^{\mathcal{R}}_{\text{Sch}}$ in Fig. 11.

In fact, Schnorr's Sigma protocol is a perfect Chameleon Σ-protocol, so for all $(x, y) \in \mathcal{R}$, the transcripts generated by $(\mathcal{P}, \mathcal{V})$ and that generated by $(\mathcal{P}', \mathcal{V})$ are distributed identically.

Standard mode:	$\mathcal{V}_2(y, a, c, z)$:	Chameleon mode:
(1) $\mathcal{P}_1(\perp, y)$	$a' \leftarrow g^z / y^c$	(1) $\mathcal{P}'_1(\perp, y)$:
$r \leftarrow \mathbb{Z}^*_p$, $a \leftarrow g^r$	Return $(a' \stackrel{?}{=} a)$	$c' \leftarrow \mathbb{Z}^*_p$
Send a to $\mathcal{V}$		$r \leftarrow \mathbb{Z}^*_p$, $a \leftarrow g^r / y^{c'}$ //$\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathsf{Sim}(y, c')$
(2) $\mathcal{V}_1(a)$:	Simulator $\mathsf{Sim}(y, c)$:	Send a to $\mathcal{V}$
$c \leftarrow \mathbb{Z}^*_p$	$z \leftarrow \mathbb{Z}^*_p$, $a \leftarrow g^z / y^c$	(2) $\mathcal{V}_1(a)$:
Send c to $\mathcal{P}$	Return (a, z)	$c \leftarrow \mathbb{Z}^*_p$, Send c to $\mathcal{P}$
(3) $\mathcal{P}_2(a, c, x, y)$		(3) $\mathcal{P}'_2(a, c, c', x, y)$:
$z \leftarrow r + cx$		$z \leftarrow r + (c - c')x$
Send z to $\mathcal{V}$		Send z to $\mathcal{V}$

Fig. 11: Schnorr's Sigma protocol $\Sigma^{\mathcal{R}}_{\text{Sch}}$

Now, we turn to the construction of Sigma protocol $\Sigma^{\mathcal{R}_{\text{1-OR}}}$.

Let $\Sigma^{\mathcal{R}}_{\text{Sch}}$ be Schnorr's Sigma protocol as shown in Fig. 11, and $\varphi : \{0,1\}^* \to \mathbb{Z}^*_p$ be a collision-resistant hash function. The Sigma protocol $\Sigma^{\mathcal{R}_{\text{1-OR}}} = (\mathcal{P}, \mathcal{V})$ for $\mathcal{R}_{\text{1-OR}}$ is as follows (and the detailed algorithms are shown in Fig. 12).

1. $\underline{\mathcal{P} \to \mathcal{V}.}$ The prover $\mathcal{P}_1$ computes the commitment as follows. First, $\mathcal{P}_1$ calls $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathcal{P}_1(\perp, y_k)$ to generate a random commitment a_k for the k^{th} statement y_k. Then for $l = k - 1$ to 1, $\mathcal{P}_1$ invokes the HVZK simulator $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathsf{Sim}$, feeding it with $\varphi(a_{l+1})$ as the challenge, to generate a_l for the l^{th} statement y_l. Finally, $\mathcal{P}_1$ sends $a = a_1$ to the verifier $\mathcal{V}$.
2. $\underline{\mathcal{V} \to \mathcal{P}.}$ Receiving a, $\mathcal{V}_1$ samples $c \leftarrow \mathbb{Z}^*_p$ and sends it to $\mathcal{P}$.
3. $\underline{\mathcal{P} \to \mathcal{V}.}$ Receiving c, $\mathcal{P}_2$ proceeds to compute the response. We denote the largest component in $S^{\text{w}}_{\mathbf{x}}$ as μ, i.e., the witness x_μ for y_μ is known by the prover. For every $l > \mu$, $\mathcal{P}_2$ invokes the HVZK simulator $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathsf{Sim}$ to generate another commitment a'_l for each statement y_l. Then, for $l = \mu$, $\mathcal{P}_2$ calls $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathcal{P}'_2(a_\mu, \varphi(a'_{\mu+1}), \varphi(a_{\mu+1}), x_\mu, y_\mu)$ (or $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathcal{P}_2(a_k, c, x_k, y_k)$ if $\mu = k$) to generate a valid response. For every $l < \mu$, we just set the responses equal

to those responses output by the HVZK simulator in the first step. Finally, $\mathcal{P}_2$ sends $z = \{z_l\}_{l\in[k]}$ to the verifier.

The verification is as follows. The verifier $\mathcal{V}_2$ invokes the codes in $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathcal{V}_2$ to compute the commitments for every statement. Then he compares the computed commitment of the first statement with the commitment a sent by $\mathcal{P}_1$. If they are equal, $\mathcal{V}_2$ outputs 1.

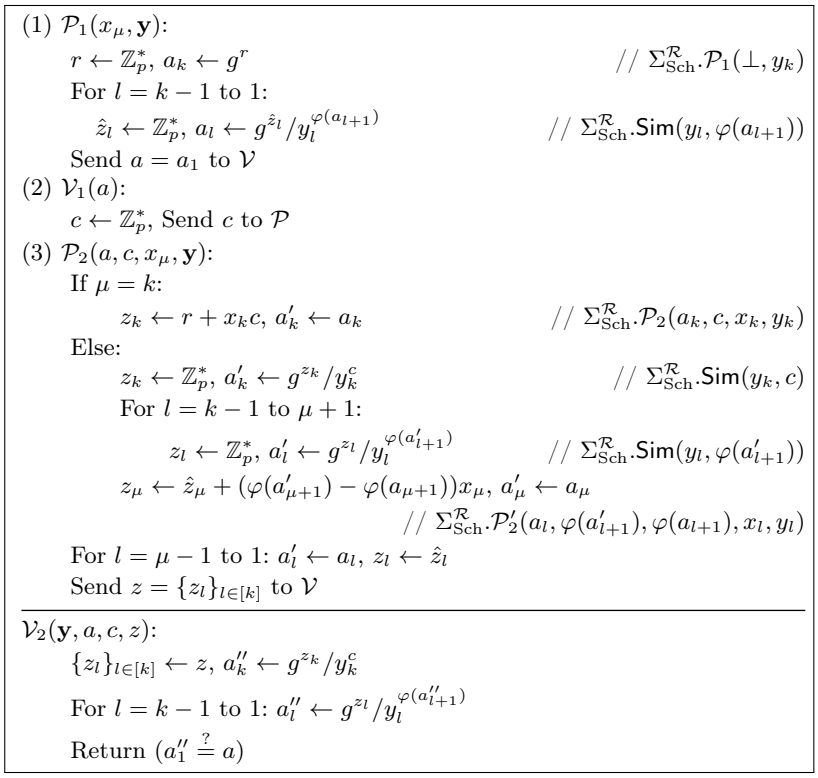


Fig. 12: Algorithms of $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ (μ is the largest component in $S^{\text{w}}_{\mathbf{x}}$, i.e., the prover knows x_μ for y_μ.)

Completeness. Now we analyze the completeness of $\Sigma^{\mathcal{R}_{1\text{-OR}}}$. For any $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}_{1\text{-OR}}$, denote the largest component in $S^{\text{w}}_{\mathbf{x}}$ as μ. If $\mu = k$, we have $a''_k = g^{z_k}/y_k^c = g^{r+x_k c}/y_k^c = g^r = a_k = a'_k$. If $\mu < k$, we have $a''_k = g^{z_k} y_k^c = a'_k$ and then by mathematical induction we have $a''_{\mu+1} = a'_{\mu+1}$. Further, we have

$$\begin{aligned} a''_\mu &= g^{z_\mu}/y_\mu^{\varphi(a''_{\mu+1})} = g^{z_\mu}/y_\mu^{\varphi(a'_{\mu+1})} \\ &= g^{\hat{z}_\mu + (\varphi(a'_{\mu+1}) - \varphi(a_{\mu+1}))x_\mu}/y_\mu^{\varphi(a'_{\mu+1})} = g^{\hat{z}_\mu}/y_\mu^{\varphi(a_{\mu+1})} = a_\mu = a'_\mu. \end{aligned}$$

Therefore, when $l < \mu$, we can prove the following recursively: $a''_l = g^{z_l}/y_l^{\varphi(a''_{l+1})} = g^{z_l}/y_l^{\varphi(a'_{l+1})} = g^{\hat{z}_l}/y_l^{\varphi(a_{l+1})} = a_l = a'_l$. It implies that $a''_1 = a'_1 = a_1 = a$, so $\mathcal{V}_2$ outputs 1.

The completeness implies some special features of $\Sigma^{\mathcal{R}_{1\text{-OR}}}$:

1. For every statement, the commitment computed by $\mathcal{P}_2$ equals that computed by $\mathcal{V}_2$, i.e, $a'_l = a''_l$ $(l \in [k])$.
2. For the statement of which the prover knows the witness, the corresponding commitments in different steps are the same, i.e., $a_\mu = a'_\mu = a''_\mu$.
3. If $a_{l+1} \neq a''_{l+1}$ $(l \in [k-1])$ and the prover does not know the witness of y_l, then it holds that $a_l \neq a''_l$ with overwhelming probability.

Due to page limitations, the analysis of computational knowledge soundness, special HVZK and witness indistinguishability of $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ are given in Appendix E.

With this Sigma protocol $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ as a building block, we can obtain a composite Sigma protocol $\Sigma_{\text{plain}}^{\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}}$ for $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$ (Eq. (3)) following the framework as mentioned before. We note that the communication complexity of the composite Sigma protocol for $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$ is $O(k \cdot num)$, which theoretically equals the complexity of [14].

5.2 Description of DAG-Σ protocols

Here, we construct a more efficient Sigma protocol for $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$ in Eq. (3). Informally, we construct this protocol following the main idea of $\Sigma^{\mathcal{R}_{1\text{-OR}}}$, except that (i) we firstly convert the relation to a directed acyclic graph (DAG), and generate a commitment for each node v of the DAG (instead of generating a_l for each statement y_l in $\Sigma^{\mathcal{R}_{1\text{-OR}}}$), and (ii) the value of commitment for node v depends on all the commitments for the nodes in S_v^{pred} (while the value of commitment a_l depends on a single statement a_{l+1} for statement y_{l+1}). Furthermore, the communication complexity of the DAG-based protocol depends on the number of vertices of the DAG.

Building blocks. Let $\Sigma^{\mathcal{R}}_{\text{Sch}}$ be Schnorr's Sigma protocol as shown in Fig. 11, and $\varphi : \{0,1\}^* \to \mathbb{Z}_p^*$ be a collision-resistant hash function. Let kCNFtoDAG be the deterministic transfer algorithm presented in Sec. 4, which takes a k-CNF relation $\mathcal{R}_{k\text{-CNF},S'_k}$ (i.e., relation of the form like Eq. (2)) as input and outputs a directed acyclic graph $G = (V, E)$. As in the description of kCNFtoDAG, we can have a function $f : V \to [n]$ such that if there is an edge from v_1 to v_2 in the graph, then $f(v_1) > f(v_2)$.

Overview. We firstly run the transfer algorithm kCNFtoDAG to convert the relation $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$ to a DAG $G = (V, E)$. Note that a node in G represents only one statement, while a statement may correspond to multiple nodes, since there are multiple Type-$\vee$ clauses in the expression of $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$. Recall that in the Sigma protocol $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ in Fig. 12, for each statement y_l, a corresponding commitment a_l is generated. Here, with similar approach, for each node of G, we compute a commitment for the corresponding statement. For a node v, the commitment computed with the algorithm $\mathcal{P}_1$ of the DAG-Σ protocol is denoted as a_v if

$v \in S^{\mathsf{source}}$, or b_v if $(v \notin S^{\mathsf{source}}) \wedge (v \notin S^{\mathsf{sink}})$, or e_v if $v \in S^{\mathsf{sink}}$. In other words, it is denoted according to the in-degree and out-degree of node v. Note that the in-degree and out-degree cannot both be zero when $k \geq 2$ (it is a trivial problem when $k = 1$). In addition, the values of these commitments will not be changed once they are assigned.

On the other hand, recall that in $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ (as shown in Fig. 12), commitment a_l is computed based on $\varphi(a_{l+1})$, i.e., the underlying hash function φ takes only one commitment as input. In our DAG-Σ protocol, when computing the commitment for the statement corresponding to node v (hereinafter, we sometimes directly write it as the commitment for node v for simplicity), the hash function φ would take all the commitments for the nodes in S_v^{pred} as input. Specifically, for the algorithm $\mathcal{P}_1$ of the DAG-Σ protocol, we provide an algorithm $\mathsf{msg}(G, v)$ to "splice" the commitments computed by $\mathcal{P}_1$, denoting the output of $\mathsf{msg}(G, v)$ as m_v, such that φ will directly take m_v as input. We assume that msg always "splice" the commitments from the smallest index to the largest one. So for any fixed node v in G, $\mathsf{msg}(G, v)$ is also a fixed value. The detailed description of msg will be given in Fig. 14.

Analogously, in the description of the DAG-Σ protocol (which will be shown in Fig. 13 and Fig. 14), the commitments computed by $\mathcal{P}_2$ (resp., $\mathcal{V}_2$) are denoted as a'_v, b'_v or e'_v (resp., a''_v, b''_v or e''_v). Respectively, we also provide msg' and msg'', and the detailed descriptions will be given in Fig. 14.

Note that in $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ (as shown in Fig. 12), the corresponding commitments computed in $\Sigma^{\mathcal{R}_{1\text{-OR}}}.\mathcal{P}_1$ and in $\Sigma^{\mathcal{R}_{1\text{-OR}}}.\mathcal{P}_2$ are equal (i.e., $a_l = a'_l$ in Fig. 12), only when the prover knows the witness x_l or $a_{l+1} = a'_{l+1}$. Comparatively, in our DAG-Σ protocol, the commitments (for a node v) computed in $\mathcal{P}_1$ and in $\mathcal{P}_2$ are equal, only when the prover knows the witness (of the statement corresponding to v) or $\mathsf{msg}(G, v) = \mathsf{msg}'(G, v)$.

In addition, as described in $\Sigma_{\text{plain}}^{\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}}$ in Sec. 5.1, $\Sigma_{\text{plain}}^{\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}}.\mathcal{P}_1$ sends all the a_1's of different Type-$\vee$ clauses to $\Sigma_{\text{plain}}^{\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}}.\mathcal{V}_1$, and then $\Sigma_{\text{plain}}^{\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}}.\mathcal{V}_2$ computes all the corresponding (a''_1)'s and compare them with a_1's for verification. Comparatively, in our DAG-Σ protocol, $\mathcal{P}_1$ sends all the $\{e_v\}_{v \in S^{\mathsf{sink}}}$ to $\mathcal{V}_1$, and then $\mathcal{V}_2$ computes all the $\{e''_v\}_{v \in S^{\mathsf{sink}}}$ and compares them with $\{e_v\}_{v \in S^{\mathsf{sink}}}$ for verification.

Next, we turn to the detailed description of our DAG-Σ protocol.

Description. Our DAG-based Sigma protocol $\Sigma_{\text{DAG}}^{\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}}$ for relation $\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k}$ is as follows. The detailed algorithms are shown in Fig. 13 and Fig. 14.

1. $\mathcal{P} \to \mathcal{V}$. The prover $\mathcal{P}_1$ first calls $\mathsf{kCNFtoDAG}(\mathcal{R}^{\text{dl}}_{k\text{-CNF},S'_k})$ to get a directed acyclic graph $G = (V, E)$, and then generates the commitment a as follows: for every node v in G,
   (a) if v is a source (i.e., $\mathsf{in\text{-}deg}(v) = 0$), then $\mathcal{P}_1$ calls $\Sigma^{\mathcal{R}}_{\text{Sch}}.\mathcal{P}_1$ to generates a commitment for this node, i.e., $a_v = g^{r_v}$, where $r_v \leftarrow \mathbb{Z}_p^*$.

(b) if v is neither a source nor a sink (i.e., $\mathsf{in\text{-}deg}(v) \neq 0$ and $\mathsf{out\text{-}deg}(v) \neq 0$), $\mathcal{P}_1$ invokes the HVZK simulator $\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathsf{Sim}$ to generate the commitment b_v for node v (i.e., $b_v \leftarrow \Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathsf{Sim}(y_{f(v)}, \varphi(m_v))$), where $m_v \leftarrow \mathsf{msg}(G, v)$.

(c) if v is a sink (i.e., $\mathsf{out\text{-}deg}(v) = 0$), $\mathcal{P}_1$ computes a commitment for node v similar to step (b), and the only difference is that we denote the commitment as e_v here.

Finally, $\mathcal{P}_1$ sends $a = \{e_v\}_{v \in S^{\mathsf{sink}}}$ to the verifier $\mathcal{V}$.

2. $\underline{\mathcal{V} \rightarrow \mathcal{P}.}$ Receiving a, $\mathcal{V}_1$ samples $c \leftarrow \mathbb{Z}_p^*$ and sends it to $\mathcal{P}$.
3. $\underline{\mathcal{P} \rightarrow \mathcal{V}.}$ Receiving c, $\mathcal{P}_2$ proceeds to compute the response. In a nutshell, for every $v \in V$: if the prover knows $x_{f(v)}$ of the corresponding statement $y_{f(v)}$, she calls $\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathcal{P}_2$ to compute a response if $v \in S^{\mathsf{source}}$, or calls $\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathcal{P}'_2$ if $(v \notin S^{\mathsf{source}}) \wedge (m_v \neq m'_v)$; otherwise (i.e., the prover does not know any witness of the corresponding statement), she calls $\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathsf{Sim}$ to generate a response and re-generate the commitment once $v \in S^{\mathsf{source}}$ or $m_v \neq m'_v$. In the above cases, if $(v \notin S^{\mathsf{source}}) \wedge (m_v = m'_v)$, then we just set the response equal to that output by the simulator in $\mathcal{P}_1$. Note that if for some $v \in S^{\mathsf{sink}}$, the prover does not know $x_{f(v)}$, and $m_v \neq m'_v$, then the protocol aborts, because we can find a Type-$\vee$ clause such that the prover does not know any witness of the statements in it, which implies that $(\mathbf{x}, \mathbf{y}) \notin \mathcal{R}^{\mathrm{dl}}_{k\text{-CNF}, S'_k}$. Finally, $\mathcal{P}_2$ sends $z = \{z_v\}_{v \in V}$ to $\mathcal{V}$.

The verification is as follows. $\mathcal{V}_2$ invokes the codes in $\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathcal{V}_2$ to compute the commitments for every node in G according to the edges in G. If the commitments of the nodes in S^{sink} are equal to the corresponding commitments sent by $\mathcal{P}_1$, then $\mathcal{V}_2$ accepts, otherwise he rejects.

We provide some more explanations about the algorithms here.

Given a node v, $\mathsf{msg}(G, v)$ will always succeed in returning the same value, because (i) G is a directed acyclic graph, there are no inter-dependent nodes, i.e., no endless loops exist; (ii) its predecessor nodes can have correct assignments, which can be achieved by adopting recursion or a special node sequence (for the code "For $v \in V$" in $\mathcal{P}_1$ and we omit the details here). In addition, the "For loops" in the msg are executed following a deterministic sequence of nodes, e.g., from the smallest index to the largest. Similar explanations are also applied to $\mathcal{P}_2$ and $\mathcal{V}_2$ with msg' and msg'' respectively.

For all $v \in V$, a_v (or b_v or e_v) and $\hat{z}_v$ are generated by $\mathcal{P}_1$, a'_v (or b'_v or e'_v) and z_v are generated by $\mathcal{P}_2$, and a''_v (or b''_v or e''_v) is generated by $\mathcal{V}_2$. $\mathcal{P}$ knows some witness of $y_{f(v)}$ if and only if $f(v) \in S^{\mathrm{w}}_{\mathbf{x}}$. Moreover, algorithm $\mathcal{P}_2$ has the following properties.

***(I)*:** For any $v \in V$, if $f(v) \in S^{\mathrm{w}}_{\mathbf{x}}$, then $a'_v = a_v$ or $b'_v = b_v$ or $e'_v = e_v$. In other words, if $a'_v \neq a_v$ or $b'_v \neq b_v$ or $e'_v \neq e_v$, then $f(v) \notin S^{\mathrm{w}}_{\mathbf{x}}$.

***(II)*:** For any $v \in V \setminus S^{\mathsf{source}}$, if $f(v) \notin S^{\mathrm{w}}_{\mathbf{x}}$, and for all $v' \in S^{\mathsf{pred}}_v$, $a'_{v'} = a_{v'}$ or $b'_{v'} = b_{v'}$ (i.e., $m_v = m'_v$), then $b'_v = b_v$ or $e'_v = e_v$.

***(III)*:** Implied by *(I)* and *(II)*, if $a'_v \neq a_v$ or $b'_v \neq b_v$ or $e'_v \neq e_v$ for some $v \in V \setminus S^{\mathsf{source}}$, then there must be some $v' \in S^{\mathsf{pred}}_v$ such that $a'_{v'} \neq a_{v'}$ or $b'_{v'} \neq b_{v'}$ (which further implies $f(v') \notin S^{\mathrm{w}}_{\mathbf{x}}$ according to Property *(I)*).

(1) $\mathcal{P}_1(\mathbf{x}, \mathbf{y})$:
  $G = (V, E) \leftarrow \mathsf{kCNFtoDAG}(\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k})$ // convert the relation into a DAG
  For $v \in V$:
    If $\mathsf{in\text{-}deg}(v) = 0$: $r_v \leftarrow \mathbb{Z}^*_p$, $a_v \leftarrow g^{r_v}$ // $\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathcal{P}_1(\perp, y_{f(v)})$
    Else If $\mathsf{out\text{-}deg}(v) \neq 0$:
      $m_v \leftarrow \mathsf{msg}(G, v)$, $\hat{z}_v \leftarrow \mathbb{Z}^*_p$, $b_v \leftarrow g^{\hat{z}_v} / y_{f(v)}^{\varphi(m_v)}$ //$\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathsf{Sim}(y_{f(v)}, \varphi(m_v))$
    Else $m_v \leftarrow \mathsf{msg}(G, v)$, $\hat{z}_v \leftarrow \mathbb{Z}^*_p$, $e_v \leftarrow g^{\hat{z}_v} / y_{f(v)}^{\varphi(m_v)}$ //$\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathsf{Sim}(y_{f(v)}, \varphi(m_v))$
  Send $a = \{e_v\}_{v \in S^{\mathsf{sink}}}$ to $\mathcal{V}$
(2) $\mathcal{V}_1(a)$: $c \leftarrow \mathbb{Z}^*_p$, Send c to $\mathcal{P}$
(3) $\mathcal{P}_2(a, c, \mathbf{x}, \mathbf{y})$:
  For $v \in V$:
    If $f(v) \in S^{\mathrm{w}}_{\mathbf{x}}$: //$\mathcal{P}$ knows witness of $y_{f(v)}$
      If $\mathsf{in\text{-}deg}(v) = 0$: $z_v \leftarrow r_v + x_{f(v)}c$, $a'_v \leftarrow a_v$ //$\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathcal{P}_2(a_v, c, x_{f(v)}, y_{f(v)})$
      Else If $(m'_v \leftarrow \mathsf{msg}'(G, v), m_v \neq m'_v)$:
        $z_v \leftarrow \hat{z}_v + (\varphi(m'_v) - \varphi(m_v))x_{f(v)}$//$\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathcal{P}'_2(a_v, \varphi(m'_v), \varphi(m_v), x_{f(v)}, y_{f(v)})$
        Else $z_v \leftarrow \hat{z}_v$
        If $\mathsf{out\text{-}deg}(v) \neq 0$: $b'_v \leftarrow b_v$
        Else $e'_v \leftarrow e_v$
    Else //$\mathcal{P}$ does not know witness of $y_{f(v)}$
      If $\mathsf{in\text{-}deg}(v) = 0$: $z_v \leftarrow \mathbb{Z}^*_p$, $a'_v \leftarrow g^{z_v} / y^c_{f(v)}$ //$\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathsf{Sim}(y_{f(v)}, c)$
      Else If $(m'_v \leftarrow \mathsf{msg}'(G, v), m_v \neq m'_v)$:
        If $\mathsf{out\text{-}deg}(v) \neq 0$:
          $z_v \leftarrow \mathbb{Z}^*_p$, $b'_v \leftarrow g^{z_v} / y_{f(v)}^{\varphi(m'_v)}$ //$\Sigma^{\mathcal{R}}_{\mathrm{Sch}}.\mathsf{Sim}(y_{f(v)}, \varphi(m'_v))$
        Else Return $\perp$
      Else
        If $\mathsf{out\text{-}deg}(v) \neq 0$: $b'_v \leftarrow b_v$, $z_v \leftarrow \hat{z}_v$
        Else $e'_v \leftarrow e_v$, $z_v \leftarrow \hat{z}_v$
  Send $z = \{z_v\}_{v \in V}$ to $\mathcal{V}$

Fig. 13: Generation algorithms of $\Sigma^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}_{\mathrm{DAG}}$ (Assume that the "For loops" are executed following a deterministic sequence of nodes.)

***(IV)*:** Implied by *(III)* and by induction on path, if $a'_{\tilde{v}} \neq a_{\tilde{v}}$ or $b'_{\tilde{v}} \neq b_{\tilde{v}}$ or $e'_{\tilde{v}} \neq e_{\tilde{v}}$ for some $\tilde{v} \in V$, then there must be some path such that for any vertex v in the path from a source to $\tilde{v}$, $f(v) \notin S^{\mathrm{w}}_{\mathbf{x}}$.

***(V)*:** As a special case of *(IV)*, if $e'_{\tilde{v}} \neq e_{\tilde{v}}$ for some $\tilde{v} \in S^{\mathsf{sink}}$, there must be some path such that for any vertex v in this path, $f(v) \notin S^{\mathrm{w}}_{\mathbf{x}}$, which also implies that there is a Type-$\vee$ clause $\vee_{j \in [k]} y_{i_j}$ such that $\mathcal{P}$ does not know any witness of $(y_{i_j})_{j \in [k]}$, i.e., $(\mathbf{x}, \mathbf{y}) \notin \mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}$.

Note that when the event mentioned in Property *(V)* occurs, $\mathcal{P}_2$ will return $\perp$, as shown in Fig. 13.

$\mathcal{V}_2(\mathbf{y}, a, c, z)$:
  $G = (V, E) \leftarrow \mathsf{kCNFtoDAG}(\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k})$
  $\{e_v\}_{v\in S^{\mathsf{sink}}} \leftarrow a$, $\{z_v\}_{v\in V} \leftarrow z$
  For $v \in V$:
    If $\mathsf{in\text{-}deg}(v) = 0$: $a''_v \leftarrow g^{z_v}/y^{c}_{f(v)}$
    Else If $\mathsf{out\text{-}deg}(v) \neq 0$:
      $m''_v \leftarrow \mathsf{msg}''(G, v)$
      $b''_v \leftarrow g^{z_v}/y^{\varphi(m''_v)}_{f(v)}$
    Else
      $m''_v \leftarrow \mathsf{msg}''(G, v)$
      $e''_v \leftarrow g^{z_v}/y^{\varphi(m''_v)}_{f(v)}$
  If $\forall v \in S^{\mathsf{sink}}$, $e''_v = e_v$:
    Return 1
  Else
    Return 0

$\mathsf{msg}(G, v)$:
  $m_v \leftarrow \perp$, $S^{\mathsf{pred}}_v \leftarrow \mathsf{pred}(v)$
  For $v' \in S^{\mathsf{pred}}_v$:
    If $\mathsf{in\text{-}deg}(v') = 0$: $m_v \leftarrow (m_v||a_{v'})$
    Else $m_v \leftarrow (m_v||b_{v'})$
  Return m_v

$\mathsf{msg}'(G, v)$:
  $m'_v \leftarrow \perp$, $S^{\mathsf{pred}}_v \leftarrow \mathsf{pred}(v)$
  For $v' \in S^{\mathsf{pred}}_v$:
    If $\mathsf{in\text{-}deg}(v') = 0$: $m'_v \leftarrow (m'_v||a'_{v'})$
    Else $m'_v \leftarrow (m'_v||b'_{v'})$
  Return m'_v

$\mathsf{msg}''(G, v)$:
  $m''_v \leftarrow \perp$, $S^{\mathsf{pred}}_v \leftarrow \mathsf{pred}(v)$
  For $v' \in S^{\mathsf{pred}}_v$:
    If $\mathsf{in\text{-}deg}(v') = 0$: $m''_v \leftarrow (m''_v||a''_{v'})$
    Else $m''_v \leftarrow (m''_v||b''_{v'})$
  Return m''_v

Fig. 14: Verification algorithm of $\Sigma^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}_{\mathrm{DAG}}$ and other auxiliary algorithms (Assume that the "For loops" are executed following a deterministic sequence of nodes.)

Completeness. For any $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}$, let (a, c, z) denote the transcript generated by the protocol. Now we consider the computation of $\mathcal{V}_2(\mathbf{y}, a, c, z)$. Note that for all $v \in S^{\mathsf{source}}$, $a''_v = g^{z_v}/y^{c}_{f(v)} = a'_v$. By induction on path, we have the following claim, the formal proof of which will be given in Appendix F.

Claim. For all $v \in V \setminus S^{\mathsf{sink}}$, $a''_v = a'_v$ or $b''_v = b'_v$.

According to above claim, for any $v \in V$, $m''_v = m'_v$. For each $v \in S^{\mathsf{sink}}$ satisfying $f(v) \in S^{\mathrm{w}}_{\mathbf{x}}$, we have:

(1) If $m_v \neq m'_v$, then

$$\begin{aligned} e''_v &= g^{z_v}/y^{\varphi(m''_v)}_{f(v)} = g^{z_v}/y^{\varphi(m'_v)}_{f(v)} \\ &= g^{\hat{z}_v+(\varphi(m'_v)-\varphi(m_v))x_{f(v)}}/y^{\varphi(m'_v)}_{f(v)} = g^{\hat{z}_v}/y^{\varphi(m_v)}_{f(v)} = e_v. \end{aligned}$$

(2) If $m_v = m'_v$, then according to the procedures of $\mathcal{P}_2$, we have $z_v = \hat{z}_v$, so $e''_v = g^{z_v}/y^{\varphi(m''_v)}_{f(v)} = g^{z_v}/y^{\varphi(m'_v)}_{f(v)} = g^{\hat{z}_v}/y^{\varphi(m_v)}_{f(v)} = e_v$.

For each $v \in S^{\mathsf{sink}}$ satisfying $f(v) \notin S^{\mathrm{w}}_{\mathbf{x}}$, we have:

(1) If $m_v \neq m'_v$, then according to Property *(IV)*, there is some path such that for any vertex $\tilde{v}$ in the path from a source to v', $f(\tilde{v}) \notin S^{\mathrm{w}}_{\mathbf{x}}$ (we denote these $k-1$ vertices as S_{pa}). Note that $v \in S^{\mathsf{sink}}$ and $f(v) \notin S^{\mathrm{w}}_{\mathbf{x}}$, so $S_{\mathsf{pa}} \cup \{v\}$ constitute a path such that for any vertex $\tilde{v}$ in the path,

$f(\tilde{v}) \notin S^{\mathrm{w}}_{\mathbf{x}}$. According to Property *(V)*, $(\mathbf{x}, \mathbf{y}) \notin \mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}$, contradicting the assumption that $(\mathbf{x}, \mathbf{y}) \in \mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}$. So we don't need to consider this case in completeness analysis.

(2) If $m_v = m'_v$ then according to the procedures of $\mathcal{P}_2$, we have $e'_v = e_v$ and $z_v = \hat{z}_v$. Since $m''_v = m'_v$, we derive $e''_v = g^{z_v}/y_{f(v)}^{\varphi(m''_v)} = g^{\hat{z}_v}/y_{f(v)}^{\varphi(m_v)} = e_v$.

Other properties. For computational knowledge soundness and special HVZK property, we have the following theorem. Due to space limitations, we provide the proof in Appendix G.

Theorem 3. *If φ is a collision-resistant hash function, $\Sigma_{\mathrm{DAG}}^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}$ provides computational knowledge soundness and is special HVZK.*

Communication complexity. It is clear that there are $|S^{\mathsf{sink}}|$ group elements and $(|V|+1)$ elements in $\mathbb{Z}_p^*$ in the communication of the 3-move Sigma protocol $\Sigma_{\mathrm{DAG}}^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}$. If we apply Fiat-Shamir transform [15], the total proof would be $|V|$ elements in $\mathbb{Z}_p^*$.

According to Theorem 2, $|V| \leq \mathsf{Min}(V_{\mathrm{bound}}, (k \cdot num))$, which implies that $|V| \leq k \cdot num$. Note that the communication complexity of [14] is $O(k \cdot num)$, so we can draw such a conclusion that the communication complexity of $\Sigma_{\mathrm{DAG}}^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}$ is better than that of [14]. A further analysis of V_{bound} in Appendix C.2 will show that *generally* $V_{\mathrm{bound}} \ll k \cdot num$. It implies that *generally* $\Sigma_{\mathrm{DAG}}^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}$ protocol based on $\mathsf{kCNFtoDAG}$ has a remarkable performance improvement on proving k-CNF relations, when compared with [14].

6 Extension: incomplete k-DNF relations

In Sec. 5.2, we have shown a Sigma protocol for k-CNF relations. However, in some scenarios, the required relations of partial knowledge are formalized in disjunctive normal form (DNF) [13,5] , i.e., each clause combines the statements using "AND" operation, and then the formula of the relation combines the clauses using "OR" operation. If every clause has k statement, we call it k-DNF relations and then we further classify them into *complete* ones and *incomplete* ones. In this section, we show a construction of Sigma protocols for incomplete k-DNF relations, partially based on DAG-Σ protocol in Sec. 5.

6.1 Problem definition

First, please refer to Sec. 3 for the notations of S_k and $\mathcal{R}_l$ ($l \in [n]$). Then, we define the following partial knowledge for compound statements.

Definition 2. (Complete k-out-of-n partial knowledge for DNF). *Given n different statements $\{y_l\}_{l\in[n]}$ and n sub-relations $\{\mathcal{R}_l\}_{l\in[n]}$, the prover proves that she knows k witnesses among the n statements. In other words, she knows some $(y_{i_1}, \cdots, y_{i_k})$ are true, where $\{i_1, \cdots, i_k\} \in S_k$.*

The relation can be presented in DNF as follows,

$$\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k} = \{(\mathbf{x},\mathbf{y}) : \vee_{\{i_1,\ldots,i_k\}\in S_k}(\wedge_{j\in[k]}(x_{i_j},y_{i_j}) \in \mathcal{R}_{i_j})\}, \tag{8}$$

where $\mathbf{x},\mathbf{y}$ are two n-dimension vectors, and $\mathcal{R}_{i_j} \in \{\mathcal{R}_l\}_{l\in[n]}$ is a sub-relation. For simplicity, we denote the relation in disjunctive normal form where every clause has k statements as **complete k-DNF relation**. Furthermore, we stress that $|S_k| = C^k_n$.

Then similarly, we define the incomplete k-out-of-n partial knowledge relation in DNF as follows.

Definition 3. (Incomplete k-out-of-n partial knowledge for DNF). *Given n different statements $\{y_l\}_{l\in[n]}$, n sub-relations $\{\mathcal{R}_l\}_{l\in[n]}$, and a subset $S''_k \subsetneqq S_k$, the prover proves that she knows some $(y_{i_1},\cdots,y_{i_k})$ are true, where $\{i_1,\cdots,i_k\} \in S''_k$.*

Similarly, the relation can be presented in DNF as follows,

$$\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k} = \{(\mathbf{x},\mathbf{y}) : \vee_{\{i_1,\ldots,i_k\}\in S''_k}(\wedge_{j\in[k]}(x_{i_j},y_{i_j}) \in \mathcal{R}_{i_j})\}, \tag{9}$$

where $\mathbf{x},\mathbf{y}$ are two n-dimension vectors, and $\mathcal{R}_{i_j} \in \{\mathcal{R}_l\}_{l\in[n]}$ is a sub-relation. Note that $|S''_k| < C^k_n$. We denote the relation in Eq. (9) as **incomplete k-DNF relation** and we also focus on the incomplete k-DNF relations that can be decided in polynomial time.

6.2 A transfer for special cases

Following $\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}$ (Eq. (8)) and $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$ (Eq. (9)), we further consider the following relations,

$$\mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\backslash S''_k} = \{(\mathbf{x},\mathbf{y}) : \wedge_{\{i_1,\ldots,i_k\}\in S_k\backslash S''_k}(\vee_{j\in[k]}(x_{i_j},y_{i_j}) \notin \mathcal{R}_{i_j})\}, \tag{10}$$

$$\mathcal{R}^{\text{tsf}} = \mathcal{R}^{\text{com}}_{k\text{-DNF},S_k} \cap \mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\backslash S''_k}, \tag{11}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two n-dimension vectors, $S''_k \subsetneqq S_k$, and we assume that $1 \le i_1 < \ldots < i_k \le n$ without loss of generality. Obviously, we have that $\mathcal{R}^{\text{tsf}} \subset \mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\backslash S''_k}$.

Now we show $\mathcal{R}^{\text{tsf}} \subset \mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$. Specifically, for any pair $(\mathbf{x},\mathbf{y})$ belonging to $\mathcal{R}^{\text{tsf}}$, $(\mathbf{x},\mathbf{y}) \in \mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}$ and $(\mathbf{x},\mathbf{y}) \in \mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\backslash S''_k}$. In other words, at least one clause labeled in S_k with respect to $\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}$, e.g., $(\wedge_{j\in[k]}(x_{i_j},y_{i_j}) \in \mathcal{R}_{i_j})$, is true, while the clauses labeled in $S_k\backslash S''_k$ with respect to $\mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\backslash S''_k}$, e.g., $(\vee_{j\in[k]}(x_{i_j},y_{i_j}) \notin \mathcal{R}_{i_j})$, are all true. It means that the clauses labeled in $S_k\backslash S''_k$ with respect to $\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}$ are all false. In all, at least one clause labeled in S''_k with respect to $\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}$ is true, which implies that $(\mathbf{x},\mathbf{y}) \in \mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$.

We claim that a Sigma protocol $\Sigma^{\mathcal{R}^{\text{tsf}}}$ for relation $\mathcal{R}^{\text{tsf}}$ can be transferred to a Sigma protocol for relation $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$. Given a witness-statement pair $(\mathbf{x},\mathbf{y}) \in$

$\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$, we know that one of the clauses with respect to $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$ is true. The prover chooses one among the true clauses, and then she only preserves the witnesses for the statements in this clause and set the others empty. Therefore, we get an $\mathbf{x}'$. It is clear that $(\mathbf{x}', \mathbf{y}) \in \mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$ and $(\mathbf{x}', \mathbf{y}) \in \mathcal{R}^{\text{tsf}}$. Thus, if $\Sigma^{\mathcal{R}^{\text{tsf}}}$ with the input $(\mathbf{x}', \mathbf{y})$ outputs a proof and the verifier accepts the proof together with input $\mathbf{y}$, then the accepting proof indicates that the prover knows the partial knowledge of $\mathbf{y}$ as per the relation $\mathcal{R}^{\text{tsf}}$, which implies that the prover knows the partial knowledge of $\mathbf{y}$ as per the relation $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$.

The Sigma protocol $\Sigma^{\mathcal{R}^{\text{tsf}}}$ can be obtained from $\Sigma^{\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}}$ and $\Sigma^{\mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\setminus S''_k}}$ using "AND"-proof construction [6]. Therefore, we have the following theorem.

Theorem 4. *The proof for an incomplete* k-DNF *relation* $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$ *can be obtained from a proof for a complete* k-DNF *relation* $\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}$ *and a proof for a* k-CNF *relation* $\mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\setminus S''_k}$.

In other words, a Sigma protocol $\Sigma^{\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}}$ can be obtained from $\Sigma^{\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}}$ and $\Sigma^{\mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\setminus S''_k}}$. Since there are some efficient constructions for $\Sigma^{\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}}$, e.g., [14], what remains is to construct $\Sigma^{\mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\setminus S''_k}}$ efficiently. However, it seems difficult to prove a "NOT" statement (e.g., $(x_{i_j}, y_{i_j}) \notin \mathcal{R}_{i_j}$) generally.

Here, we discuss this problem in the *discrete logarithm* setting for some special cases. More specifically, in the following, we show a construction of a Sigma protocol for $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$ *under the conditions (defined by Eq. (12)-(13)) in the discrete logarithm setting.*

We firstly introduce the definition of ρ-type pairs as follows.

Definition 4. (ρ-type pair). *Let* $\mathbb{G}$ *be a cyclic group of prime order* p *generated by* $g \in \mathbb{G}$. *Let* $h \in \mathbb{G}$ *be some arbitrary non-identity element and* $\log_g h$ *is unknown. Then we call* $(x, y = g^x h^\rho) \in \mathbb{Z}_p \times \mathbb{G}$ *a* ρ*-type pair, where* $\rho \in \mathbb{Z}_p$.

We stress that for any distinct ρ_1, ρ_2, when $x_1, x_2 \leftarrow \mathbb{Z}_p$, $y_1 = g^{x_1} h^{\rho_1}$ and $y_2 = g^{x_2} h^{\rho_2}$ are distributed identically.

Then, we consider the following two conditions for relations: 1) every statement is obtained from a 0-type or 1-type pair, as shown in Eq. (12); 2) further there are only k 0-type pairs among all witness-statement pairs, as shown in Eq. (12)-(13).

$$\mathcal{R}_{\text{con1}} = \{(\mathbf{x}, \mathbf{y}) : \wedge_{l\in[n]}(y_l = g^{x_l} \vee y_l/h = g^{x_l})\}, \tag{12}$$

$$\mathcal{R}_{\text{con2}} = \{(\mathbf{x}, \mathbf{y}) : (\prod_{l=1}^{n} y_l)/h^{n-k} = g^{\sum_{l=1}^{n} x_l}\}. \tag{13}$$

In the discrete logarithm setting, $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$, $\mathcal{R}^{\text{com}}_{k\text{-DNF},S_k}$ and $\mathcal{R}^{\text{not}}_{k\text{-CNF},S_k\backslash S''_k}$ can be written as

$$\mathcal{R}^{\text{incom,dl}}_{k\text{-DNF},S''_k} = \{(\mathbf{x},\mathbf{y}) : \vee_{\{i_1,\ldots,i_k\}\in S''_k}(\wedge_{j\in[k]}\ y_{i_j} = g^{x_{i_j}})\}, \tag{14}$$

$$\mathcal{R}^{\text{com,dl}}_{k\text{-DNF},S_k} = \{(\mathbf{x},\mathbf{y}) : \vee_{\{i_1,\ldots,i_k\}\in S_k}(\wedge_{j\in[k]}\ y_{i_j} = g^{x_{i_j}})\}, \tag{15}$$

$$\mathcal{R}^{\text{not,dl}}_{k\text{-CNF},S_k\backslash S''_k} = \{(\mathbf{x},\mathbf{y}) : \wedge_{\{i_1,\ldots,i_k\}\in S_k\backslash S''_k}(\vee_{j\in[k]}\overline{y_{i_j} = g^{x_{i_j}}})\}. \tag{16}$$

Under the conditions defined by Eq. (12)-(13), $\mathcal{R}^{\text{incom,dl}}_{k\text{-DNF},S''_k}$ further becomes

$$\mathcal{R}^{\text{incom}}_{k} = \mathcal{R}^{\text{incom,dl}}_{k\text{-DNF},S''_k} \cap \mathcal{R}_{\text{con1}} \cap \mathcal{R}_{\text{con2}}. \tag{17}$$

Note that $\mathcal{R}^{\text{incom,dl}}_{k\text{-DNF},S''_k}$ indicates that at least one clause labeled in S''_k is true, and $\mathcal{R}^{\text{incom}}_{k}$ means that only one clause labeled in S''_k is true.

Now we turn to $\mathcal{R}^{\text{not,dl}}_{k\text{-CNF},S_k\backslash S''_k}$ in Eq. (16) under the conditions defined by Eq. (12)-(13). Firstly, because of Eq. (12), a "NOT" statement, i.e., $\overline{y_{i_j} = g^{x_{i_j}}}$ here, can be transferred into $y_{i_j}/h = g^{x_{i_j}}$. Secondly, Eq. (12)-(13) guarantee that once $(\mathbf{x},\mathbf{y}) \in \mathcal{R}^{\text{incom}}_{k}$, for every $\{i_1,\ldots,i_k\} \in S_k\backslash S''_k$, there is at least one of the indices of the $(n-k)$ 1-type pairs falling in $\{i_1,\ldots,i_k\}$. Therefore, $\mathcal{R}^{\text{not,dl}}_{k\text{-CNF},S_k\backslash S''_k}$ in Eq. (16) under the conditions defined by Eq. (12)-(13) becomes

$$\mathcal{R}^{\text{not},\rho\text{-type}}_{k\text{-CNF},S_k\backslash S''_k} = \{(\mathbf{x},\mathbf{y}) : \wedge_{\{i_1,\ldots,i_k\}\in S_k\backslash S''_k}(\vee_{j\in[k]}\ y_{i_j}/h = g^{x_{i_j}})\}. \tag{18}$$

Considering relation

$$\mathcal{R}^{\text{tsf}}_{k} = \mathcal{R}^{\text{com,dl}}_{k\text{-DNF},S_k} \cap \mathcal{R}^{\text{not},\rho\text{-type}}_{k\text{-CNF},S_k\backslash S''_k} \cap \mathcal{R}_{\text{con1}} \cap \mathcal{R}_{\text{con2}}, \tag{19}$$

it is easy to see that $\mathcal{R}^{\text{tsf}}_{k} = \mathcal{R}^{\text{incom}}_{k}$. We note that $\mathcal{R}^{\text{com,dl}}_{k\text{-DNF},S_k}$ in Eq. (15) indicates that at least one clause labeled in S_k is true, while $\mathcal{R}^{\text{tsf}}_{k}$ in Eq. (19) implies that only one clause labeled in S_k is true and it is not labeled in $S_k\backslash S''_k$.

Hence, in order to construct a Sigma protocol for $\mathcal{R}^{\text{incom}}_{k}$ in Eq. (17), we need to construct a Sigma protocol for $\mathcal{R}^{\text{tsf}}_{k}$, which can be obtained from $\Sigma^{\mathcal{R}^{\text{com,dl}}_{k\text{-DNF},S_k}}$, $\Sigma^{\mathcal{R}^{\text{not},\rho\text{-type}}_{k\text{-CNF},S_k\backslash S''_k}}$, $\Sigma^{\mathcal{R}_{\text{con1}}}$ and $\Sigma^{\mathcal{R}_{\text{con2}}}$ using "AND" operation [6]. Moreover, $\Sigma^{\mathcal{R}_{\text{con1}}}$ and $\Sigma^{\mathcal{R}_{\text{con2}}}$ can be obtained from Schnorr's Sigma protocol and "AND/OR" proof construction. As for $\Sigma^{\mathcal{R}^{\text{com,dl}}_{k\text{-DNF},S_k}}$, there are existing constructions, e.g. [5,14]. As for $\Sigma^{\mathcal{R}^{\text{not},\rho\text{-type}}_{k\text{-CNF},S_k\backslash S''_k}}$, note that for each sub-relation with respect to $\mathcal{R}^{\text{not},\rho\text{-type}}_{k\text{-CNF},S_k\backslash S''_k}$, Schnorr's Sigma protocol can be applied; then $\Sigma^{\mathcal{R}^{\text{not},\rho\text{-type}}_{k\text{-CNF},S_k\backslash S''_k}}$ can be obtained from the Sigma protocol in Sec. 5.2.

Thus, we obtain an efficient Sigma protocol for $\mathcal{R}^{\text{incom}}_{k}$ in Eq. (17), i.e., the incomplete k-DNF relation $\mathcal{R}^{\text{incom}}_{k\text{-DNF},S''_k}$ under the conditions defined by Eq. (12)-(13).

Remark 2. Note that besides ρ-type pairs, the statements can be other kinds of elements, e.g., DDH and OneNDH in [13]. Roughly, we only require that they are indistinguishable and the conditions will be modified accordingly.

7 Experiments

In this section, we show the performance of our DAG-Σ protocol $\Sigma_{\mathrm{DAG}}^{\mathcal{R}_{k\text{-CNF},S'_k}^{\mathrm{dl}}}$ and to have a more straight view, we compare it with that of [14]. Note that we implement [14] in its simplest way as mentioned in the introduction section.

We implement our experiments in Golang language (version 16.6) based on elliptic curve groups with key size of 512 bits. The experiments are conducted on a docker, over Pengcheng Cloud Brain[2], running Ubuntu 16.04 on two Intel® Xeon$^{\mathrm{TM}}$ Gold 6248 CPUs@2.50 GHz and using 64 GB memory in total. We are interested in the space overhead in communications as well as the timing overhead in running the Sigma protocols. To this end, we present microbenchmarks to evaluate the overhead costs. There are two factors k and n, that will affect the performance greatly. For simplicity, we have n vary from 10 to 50 and we choose $4 \leq k \leq n/3$ in most cases (if $n/3 < 4$, we just set $k = 4$, e.g., in Fig. 15, there is only one data when $n = 10$). Since we find no k-CNF relations in use in the real world, we construct some different relations in the DL setting for our experiments. Given n and k, the number of clauses in a k-CNF relation expression num also has an influence on the performance, but the range of num is large. Similar to the theoretical analysis in Appendix C, here we set $num = C_n^k - \chi$, where χ is a random number in $[50, 200]$, which is nearly the worst case and can reflect the worst performance (i.e., the most space and running time that the tested Sigma protocols need)[3]. In Appendix H, we draw some 3D figures to show the complete and thorough influence of k and n over the performance. Here, we pick some experimental data and draw some 2D figures, e.g., stacked bar charts and line charts, for better comparison. Following are the experimental analysis.

Communication costs. The communication costs are measured by the bit length of all the messages between the prover and the verifier when running the Sigma protocols. A theoretical comparison is displayed in Table 1 in Sec. 1. Here, we make a quantitative comparison. In Table 2, we show that the communication size when $k = 4$. It is clear that our scheme saves more than 97% space overhead compared with [14].

[2] https://cloudbrain.pcl.ac.cn/

[3] Although our experiments mainly consider the setting of "dense-CNF" (informally, where num is almost C_n^k), in fact it also implies that our protocol performs better in most cases (including "non-dense-CNFs"). Given the performance of CDS for "dense-CNFs", it is easier for us to infer the performance of CDS for other k-CNFs ("non-dense-CNFs" with the same k), because given k, the performance of CDS is liner in the number of clauses num. Besides, the performance of our protocol for "non-dense-CNFs" will be better than that for "dense-CNFs". When considering the performance of "non-dense-CNFs", we can compare the worst performance of our protocol with the performance of CDS. For example, when num is about $1/2 \cdot C_n^k$ (in this case, the running time and space overhead of CDS is about half of that of CDS for "dense-CNFs"), it can be implied that our protocol can save about 80% running time and space overhead.

[4] ratio $= 1 - \frac{\text{bits of our scheme}}{\text{bits of [14]}} \times 100\%$

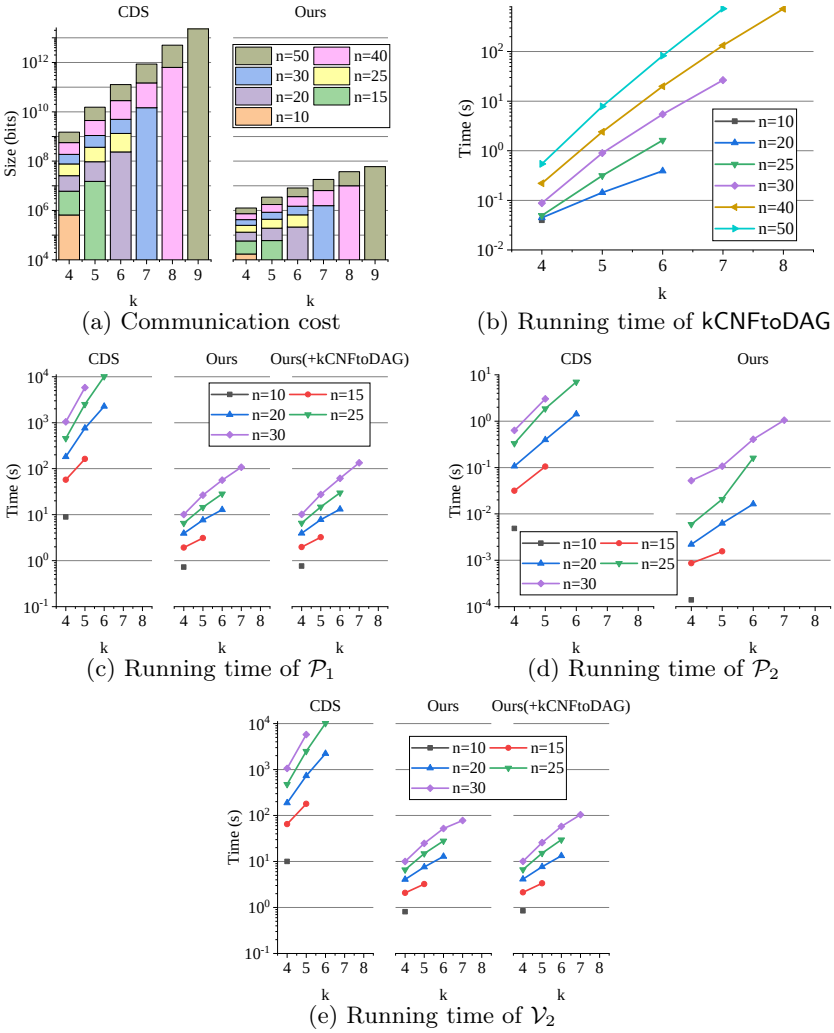


Fig. 15: Figures for the experiments (CDS is the solution [14] proposed by Cramer, Damgård and Schoenmakers. The number of clauses in one relation is $C_n^k - \chi$, where χ is a random number in $[50, 200]$.)

Table 2: Communication cost when $k = 4$ ($\times 10^4$ bits)[4]

n	[14]	Our scheme	ratio
10	65.54	1.72	97.37% ↓
15	538.62	4.07	99.24% ↓
20	1964.03	7.45	99.62% ↓
25	5160.96	11.90	99.77% ↓
30	11204.6	17.48	99.84% ↓
40	37412.9	31.92	99.91% ↓
50	94310.4	50.92	99.94% ↓

For more cases, we draw a stacked bar chart as shown in Fig. 15a, where k varies from 4 to 9. It is clear that our solution has a remarkable decrease on the communication costs compared with [14]. In addition, the figure shows that the effect on decrease would be better as n and k get larger.

Running time. We evaluate the running time of $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{V}_2$ in Fig. 15. Note that when testing our solution, we also record the running time of kCNFtoDAG for special interest. In fact, the directed acyclic graphs can be pre-computed. Thus, when recording the running time of $\mathcal{P}_1$ or $\mathcal{V}_2$ in our scheme, we have two versions: one includes the running time of kCNFtoDAG and the other one does not. Here we implement kCNFtoDAG using the improved algorithm in Appendix D.

We planned to evaluate both schemes with the same range of n and k. However, the running time of [14] grows so fast that the program was killed when n and k are set relatively large numbers. Therefore, in the experiment of [14], we set n from 10 to 33 and k from 4 to 7. In the experiment of our scheme, n varies from 10 to 50 and k varies from 4 to 10. More detailed experimental results can be found in Appendix H. Here, we just pick some data for analysis.

The running time of kCNFtoDAG is presented in Fig. 15b. It can be expected that as k and n get larger, the running time increases very quickly, since the number of vertices grows fast. If we compare it with the running time of $\mathcal{P}_1$ and $\mathcal{V}_2$ of our scheme (as shown in Fig. 15c and Fig. 15e), kCNFtoDAG performs reasonably well.

Table 3: Running time when $k = 4$ (s)[5]

n	$\mathcal{P}_1$			$\mathcal{P}_2$			$\mathcal{V}_2$		
	[14]	Ours	ratio	[14]	Ours	ratio	[14]	Ours	ratio
10	8.91	0.72	91.87% ↓	0.0049	1.40×10^{-4}	97.11% ↓	10.04	0.85	91.56% ↓
15	57.47	1.92	96.66% ↓	0.033	8.63×10^{-4}	97.27% ↓	65.08	2.13	96.72% ↓
20	182.23	3.91	97.85% ↓	0.11	2.20×10^{-3}	97.95% ↓	187.41	4.13	97.80% ↓
25	456.37	6.54	98.57% ↓	0.33	5.97×10^{-3}	98.20% ↓	477.74	6.66	98.61% ↓
30	1046.45	10.09	99.04% ↓	0.63	5.21×10^{-2}	91.78% ↓	1058.25	10.08	99.05% ↓

For the running time of $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{V}_2$, we draw a table (Table 3) to present the running time when $k = 4$. The table tells that our scheme saves more than 90% running time, compared with [14]. More cases (i.e., n varies from 10 to 30 and the range of k is $[4, 8]$) are shown in Fig. 15 (Fig. 15c - Fig. 15e). They also indicates that the running time of $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{V}_2$ of our scheme outperforms [14]. Note that counting in the running time of kCNFtoDAG or not does not affect the performance a lot, since it only occupies a limited percentage of the total running time and the time of commitment generation in $\mathcal{P}_1$ and verification in $\mathcal{V}_2$ of our scheme dominate the whole performance. In addition, Table 3, Fig. 15c and Fig. 15e show that the running time of $\mathcal{P}_1$ and $\mathcal{V}_2$ have similar performance. It is because in both [14] and our scheme, $\mathcal{P}_1$ and $\mathcal{V}_2$ have similar computation for the commitments.

In all, according to the experiment results, when compared with [14], our scheme achieves a remarkable performance improvement on proving k-CNF relations, no matter from the view of communication costs or running time.

Acknowledgements. We would like to express our sincere appreciation to the anonymous reviewers for their valuable comments and suggestions. Especially, we appreciate that the anonymous reviewers of ASIACRYPT 2022 comment on the upper bound and kindly point out another method to analyze it, which we supplement in the discussion part in Sec. 4.

References

1. Abe, M., Ambrona, M., Bogdanov, A., Ohkubo, M., Rosen, A.: Non-interactive Composition of Sigma-Protocols via Share-then-Hash. In: ASIACRYPT 2020. pp. 749–773. Springer (2020)
2. Abe, M., Ambrona, M., Bogdanov, A., Ohkubo, M., Rosen, A.: Acyclicity Programming for Sigma-Protocols. In: TCC 2021. pp. 435–465. Springer (2021)
3. Abe, M., Chase, M., David, B., Kohlweiss, M., Nishimaki, R., Ohkubo, M.: Constant-size structure-preserving signatures: Generic constructions and simple assumptions. Journal of Cryptology **29**(4), 833–878 (2016)
4. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: ASIACRYPT 2002. pp. 415–432. Springer (2002)
5. Attema, T., Cramer, R., Fehr, S.: Compressing proofs of k-out-of-n partial knowledge. In: CRYPTO 2021. pp. 65–91. Springer (2021)
6. Boneh, D., Shoup, V.: A graduate course in applied cryptography. Draft 0.5 (2020)
7. Camenisch, J.: Efficient and generalized group signatures. In: EUROCRYPT 1997. pp. 465–479. Springer (1997)
8. Canard, S., Trinh, V.C.: Constant-size ciphertext attribute-based encryption from multi-channel broadcast encryption. In: ICISS 2016. pp. 193–211. Springer (2016)
9. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO 1992. pp. 89–105. Springer (1992)
10. Choudhuri, A.R., Ciampi, M., Goyal, V., Jain, A., Ostrovsky, R.: Round optimal secure multiparty computation from minimal assumptions. In: TCC 2020. pp. 291–319. Springer (2020)

[5] Here, we count in the running time of kCNFtoDAG when running $\mathcal{P}_1$ and $\mathcal{V}_2$ in our scheme. ratio $= 1 - \frac{\text{time of our scheme}}{\text{time of [14]}} \times 100\%$.

11. Ciampi, M., Parisella, R., Venturi, D.: On adaptive security of delayed-input sigma protocols and Fiat-Shamir NIZKs. In: Security and Cryptography for Networks 2020. pp. 670–690. Springer (2020)
12. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Improved OR-Composition of Sigma-Protocols. In: TCC 2016. pp. 112–141. Springer (2016)
13. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Online/offline OR composition of Sigma protocols. In: EUROCRYPT 2016. pp. 63–92. Springer (2016)
14. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: CRYPTO 1994. pp. 174–187. Springer (1994)
15. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO 1986. pp. 186–194. Springer (1986)
16. Fischlin, M., Harasser, P., Janson, C.: Signatures from Sequential-OR Proofs. In: EUROCRYPT 2020. pp. 212–244. Springer (2020)
17. Garay, J.A., MacKenzie, P., Yang, K.: Strengthening zero-knowledge protocols using signatures. Journal of Cryptology **19**(2), 169–209 (2006)
18. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Stacking Sigmas: A Framework to Compose Σ-Protocols for Disjunctions. In: EUROCRYPT 2022. pp. 458–487. Springer (2022)
19. Goyal, V., Richelson, S.: Non-malleable commitments using goldreich-levin list decoding. In: FOCS 2019. pp. 686–699. IEEE (2019)
20. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: ASIACRYPT 2006. pp. 444–459. Springer (2006)
21. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: EUROCRYPT 2015. pp. 253–280. Springer (2015)
22. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: EUROCRYPT 2006. pp. 339–358. Springer (2006)
23. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: EUROCRYPT 2008. pp. 415–432. Springer (2008)
24. Guillou, L.C., Quisquater, J.J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: EUROCRYPT 1998. pp. 123–128. Springer (1988)
25. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. Journal of Computer and System Sciences **62**(2), 367–375 (2001)
26. Junod, P., Karlov, A.: An efficient public-key attribute-based broadcast encryption scheme allowing arbitrary access policies. In: Proceedings of the tenth annual ACM workshop on Digital rights management. pp. 13–24 (2010)
27. Jutla, C.S., Roy, A.: Shorter quasi-adaptive NIZK proofs for linear subspaces. Journal of Cryptology **30**(4), 1116–1156 (2017)
28. Lai, J., Deng, R.H., Li, Y.: Fully secure cipertext-policy hiding CP-ABE. In: Information Security Practice and Experience 2011. pp. 24–39. Springer (2011)
29. Malkin, T., Teranishi, I., Vahlis, Y., Yung, M.: Signatures resilient to continual leakage on memory and computation. In: TCC 2011. pp. 89–106. Springer (2011)
30. Okamoto, T.: An efficient divisible electronic cash scheme. In: CRYPTO 1995. pp. 438–451. Springer (1995)
31. Ràfols, C.: Stretching Groth-Sahai: NIZK proofs of partial satisfiability. In: TCC 2015. pp. 247–276. Springer (2015)
32. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: ASIACRYPT 2001. pp. 552–565. Springer (2001)

33. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (1991)
34. Tsabary, R.: Fully secure attribute-based encryption for t-CNF from LWE. In: CRYPTO 2019. pp. 62–85. Springer (2019)

A More discussions about [14] and [2] on k-CNF relations

For the Cramer et al.'s scheme [14], we often use its simplest way. In a nutshell, for the i^{th} clause in the expression of a k-CNF relation, the prover shares a given challenge c into k challenges $c_1^i, \ldots, c_k^i$ under the constraint that $c = c_1^i \oplus \ldots \oplus c_k^i$ and uses c_j^i $(1 \leq j \leq k)$ as the challenge for the j^{th} statement in the i^{th} clause.

Therefore, given a k-CNF relation, duplicate statements are independent from each other and the challenges for the statements in one clause are restricted via exclusive OR. We also take the relation in Eq. (1) for example and draw Fig. 16. As shown in Fig. 16, in each clause, there is a separate challenge for each statement and the result of exclusive OR of these challenges in each clause is restricted to the random challenge c picked by the verifier. As a result, the size of the communication is proportional to $(k \cdot num)$, where num is the number of clauses.

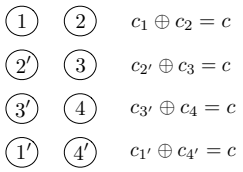
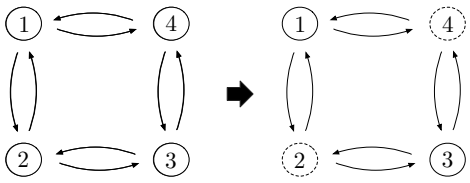


Fig. 16: An example of [14]

In [2], Abe et al. propose another method called *acyclicity program* to deal with CNF relations and construct a non-interactive zero knowledge proof (NIZK) scheme. As shown in Fig. 17, the relation in Eq. (1) is firstly transferred into the left graph of Fig. 17. For each clause, the graph has a directed cycle, e.g., $(y_1 \vee y_2)$ is turned into $1 \rightarrow 2 \rightarrow 1$. For each statement y_i, it only has one challenge c_i and the challenge satisfies the equation $c_i = \mathsf{Hash}(a_j||...)$ where a_j is a commitment of the statement that has the arrow that is pointed to node i, e.g., $c_1 = \mathsf{Hash}(a_2||a_4)$. Therefore, unlike [14] and our scheme, each statement (no matter having duplicates or not) only needs one node such that it reduces the proof size. Thus, the size of the communication is proportional to n, where n is the number of total different statements in the relation. The satisfying witnesses can make the graph have no cycles. For example, in the right graph of Fig. 17, if the prover knows the witnesses of y_2 and y_4 (the corresponding nodes with dotted line) and we remove these nodes and corresponding arrows in the graph, the graph has no cycles.

However, it needs to be very careful to generate the graph, e.g., the node order in the cycle. For a more complicated clause $(y_1 \vee y_2 \vee y_3)$, we can have

two kinds of cycles: $1 \to 2 \to 3 \to 1$ and $1 \to 3 \to 2 \to 1$. As pointed in [2], a seemingly intuitive approach to "convert" the relation to an acyclicity program does not work, i.e., a satisfying witnesses may still have cycles in the converted graph even after removing the corresponding nodes and arrows. To improve the robustness, Abe et al. propose a complex method in [2]. Since the scheme is non-interactive and no open source codes are found, we will not take it as a contrast experiment in the later experiment section.



$c_1 = \mathsf{Hash}(a_2||a_4)$ $\quad$ $c_4 = \mathsf{Hash}(a_1||a_3)$

$c_2 = \mathsf{Hash}(a_1||a_3)$ $\quad$ $c_3 = \mathsf{Hash}(a_2||a_4)$

Fig. 17: An example of [2]

B Preliminary of Sigma protocols

A *polynomial-time* relation $\mathcal{R}$ is a subset of $\mathcal{X} \times \mathcal{Y}$ for which membership of (x, y) to $\mathcal{R}$ can be decided in time polynomial in $|y|$, where $\mathcal{X}$ and $\mathcal{Y}$ are the witness space and statement space respectively. If $(x, y) \in \mathcal{R}$, we say that x is a *witness* for statement y. We define the NP-language $L_{\mathcal{R}}$ as $L_{\mathcal{R}} = \{y \in \mathcal{Y} | \exists\ x \in \mathcal{X} : (x, y) \in \mathcal{R}\}$. Following [17], let $\hat{L}_{\mathcal{R}}$ be the *input language* including both $L_{\mathcal{R}}$ and all well formed statements that do not have a witness. It follows that $L_{\mathcal{R}} \subseteq \hat{L}_{\mathcal{R}}$ and membership in $\hat{L}_{\mathcal{R}}$ can be tested in polynomial time. We implicitly assume that the verifier of a protocol for relation $\mathcal{R}$ runs the protocol only if the common input $y \in \hat{L}_{\mathcal{R}}$ and rejects immediately if $y \notin \hat{L}_{\mathcal{R}}$.

A Sigma protocol $\Sigma = (\mathcal{P}, \mathcal{V})$ for polynomial-time relation $\mathcal{R}$ consists of two efficient interactive protocol algorithms $(\mathcal{P}, \mathcal{V})$, where $\mathcal{P} = (\mathcal{P}_1, \mathcal{P}_2)$ is the prover and $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2)$ is the verifier, associated with a challenge space $\mathcal{CL}$. Specifically, for any $(x, y) \in \mathcal{R}$, the input of the prover (resp. verifier) is (x, y) (resp. y). The prover first computes $(a, \mathsf{aux}) \leftarrow \mathcal{P}_1(x, y)$ and sends the commitment a to the verifier. The verifier (i.e., $\mathcal{V}_1$) returns a challenge $c \leftarrow \mathcal{CL}$. Then the prover replies with $z \leftarrow \mathcal{P}_2(a, c, x, y, \mathsf{aux})$. Receiving z, the verifier (i.e., $\mathcal{V}_2$) outputs $b \in \{0, 1\}$. The tuple (a, c, z) is called a *transcript*. We require that $\mathcal{V}$ does not make any random choices other than the selection of c. For any fixed (a, c, z), if the final output of $\mathcal{V}(y)$ is 1, (a, c, z) is called an *accepting transcript* for y. For simplicity, we denote by $\langle \mathcal{P}(x), \mathcal{V} \rangle(y)$ the final output of $\mathcal{V}$ when running the protocol $(\mathcal{P}, \mathcal{V})$ on common input y with $\mathcal{P}$ running on additional input x.

$(\mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ is called *complete* if for all $(x, y) \in \mathcal{R}$, $\langle \mathcal{P}(x), \mathcal{V} \rangle(y) = 1$. The corresponding security notions are as follows.

Definition 5. (Knowledge soundness). *We say that a Sigma protocol* $(\mathcal{P}, \mathcal{V})$ *for* $\mathcal{R}$ *provides* knowledge soundness*, if there is an efficient deterministic algorithm* Ext *such that on input* $y \in \mathcal{Y}$ *and two accepting transcripts* $(a, c, z), (a, c', z')$ *where* $c \neq c'$*,* Ext *always outputs an* $x \in \mathcal{X}$ *satisfying* $(x, y) \in \mathcal{R}$*.*

Here, we consider computational knowledge soundness.

Definition 6. (Computational knowledge soundness). *We say that a Sigma protocol* $(\mathcal{P}, \mathcal{V})$ *for* $\mathcal{R}$ *provides* computational knowledge soundness*, if there is a deterministic algorithm* Ext *such that on input* $y \in \mathcal{Y}$ *and two accepting transcripts* $(a, c, z), (a, c', z')$ *where* $c \neq c'$*, the probability that* Ext *cannot output an* $x \in \mathcal{X}$ *satisfying* $(x, y) \in \mathcal{R}$ *is negligible.*

In other words, Ext will output an x with overwhelming probability.

Definition 7. (Special HVZK). *We say that a Sigma protocol* $(\mathcal{P}, \mathcal{V})$ *for* $\mathcal{R}$ *with challenge space* $\mathcal{CL}$ *is* special honest verifier zero knowledge (special HVZK)*, if there is a PPT simulator* Sim *which takes* $(y, c) \in \mathcal{Y} \times \mathcal{CL}$ *as input and satisfies the following properties:*

(i) for all $(y, c) \in \mathcal{Y} \times \mathcal{CL}$*,* Sim *always outputs a pair* (a, z) *such that* (a, c, z) *is an accepting transcript for* y*;*

(ii) for all $(x, y) \in \mathcal{R}$*, the tuple* (a, c, z)*, generated via* $c \leftarrow \mathcal{CL}$ *and* $(a, z) \leftarrow \mathsf{Sim}(y, c)$*, has the same distribution as that of a transcript between* $\mathcal{P}(x, y)$ *and* $\mathcal{V}(y)$*.*

The Sigma protocols can be combined using "AND/OR"-proof construction and the combined protocol is also a Sigma protocol satisfying the above security properties [6]. In addition, the OR-proof construction also satisfies witness indistinguishability.

Definition 8. (Witness indistinguishability). *We say that a Sigma protocol* $(\mathcal{P}, \mathcal{V})$ *for* $\mathcal{R}$ *with challenge space* $\mathcal{CL}$ *is* witness indistinguishable (WI)*, if for any PPT malicious verifier* $\mathcal{V}^*$*, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* x_0, x_1, y *satisfying* $(x_0, y) \in \mathcal{R}$ *and* $(x_1, y) \in \mathcal{R}$*,*

$$|\Pr[\langle \mathcal{P}(x_0), \mathcal{V}^* \rangle(y) = 1] - \Pr[\langle \mathcal{P}(x_1), \mathcal{V}^* \rangle(y) = 1]| \leq \mathsf{negl}(\lambda).$$

If the two distributions are identical, we say that the Sigma protocol is perfect *WI.*

Chameleon Σ-protocol. In a Chameleon Σ-protocol, the prover can compute the commitment a by using the simulator and thus knowing only the input but not the witness. Once the challenge c has been received, the prover can compute the response z by using the witness x (which is thus used only to compute the response) and the coin tosses used by the simulator to compute the commitment.

Definition 9. (Chameleon Σ-protocol [12]). *A* Σ*-protocol* Π *for polynomial-time relation* $\mathcal{R}$ *is a Chameleon* Σ*-protocol if there exists an special HVZK simulator* Sim *and an algorithm* $\mathcal{P}_2'$ *satisfying the following property:*

Delayed Indistinguishability*: for all pairs of of challenges c and c' and for all $(x, y) \in \mathcal{R}$, the following two distributions* $\{r \leftarrow \{0,1\}^{|x|^d}; (a, z') \leftarrow \mathsf{Sim}(y, c'; r); z \leftarrow \mathcal{P}_2'(a, c, c', x, y) : (y, a, c, z)\}$ *and* $\{(a, z) \leftarrow \mathsf{Sim}(y, c) : (y, a, c, z)\}$ *are indistinguishable, where* Sim *is the special HVZK simulator and* d *is such that* Sim*, on input an* λ*-bit instance, uses at most* λ^d *random coin tosses. If the two distributions above are identical then we say that delayed indistinguishability is perfect, and* Π *is a Perfect Chamelleon* Σ*-protocol.*

Note that a Chameleon Σ-protocol Π has two modes: the standard mode when $\mathcal{P}$ runs $\mathcal{P}_1$ and $\mathcal{P}_2$, and a *delayed* mode (in Sec. 5.1, we also call it Chameleon mode) when $\mathcal{P}$ uses Sim and $\mathcal{P}_2'$. Moreover, observe that since Sim is a simulator for Π, it follows from the delayed-indistinguishability property that, for all challenges c and $\tilde{c}$ and common inputs y, distribution

$$\{r \leftarrow \{0,1\}^{|x|^d}; (a, \tilde{z}) \leftarrow \mathsf{Sim}(y, \tilde{c}; r); z \leftarrow \mathcal{P}_2'(a, c, \tilde{c}, x, y) : (a, c, z)\}$$

is indistinguishable from

$$\{r \leftarrow \{0,1\}^{|x|^d}; a \leftarrow \mathcal{P}_1(y, x; r) z \leftarrow \mathcal{P}_2(a, c, x, y) : (a, c, z)\}$$

.

That is, the two modes of operations of Π are indistinguishable. This property make us able to claim that if Π is WI when a WI challenger interacts with an adversary using $(\mathcal{P}_1, \mathcal{P}_2)$, then Π is WI even when the pair $(\mathsf{Sim}, \mathcal{P}_2')$ is used. Finally, we observe that Chameleon Σ-protocols do exist and Schnorr's protocol [33] is one example. When considering the algorithms associated to a Chameleonn Σ-protocol, we will add $\mathcal{P}_2'$.

C Analysis of kCNFtoDAG

C.1 Proof of Theorem 1

Proof (Proof of Theorem 1). It is clear the result of Step 1 (i.e., preparing nodes) satisfies the requirements of Property-(i) and Property-(ii) of kCNFtoDAG:

1. every clause is mapped to a path, and all nodes in this path correspond to the statements in the clause;
2. every k-size path is mapped to a clause, and the number of paths is exactly equals the number of clauses.

Hence, the DAG after Step 1 (and before Step 2) has Property-(i) and Property-(ii).

Note that the merging operations in Step 2 (i.e., merging prefixes) and Step 3 (i.e., merging suffixes) will not change any statement a node corresponds to. Further, we claim that these merging operations do not lead to the changes in the number of k-size paths either. The reason is as follows.

We first consider the merging operations in Step 2. As shown in Fig. 18, there are two cases. In Case 1, we only merge the beginning nodes (e.g., v_1 and v'_1) corresponding to the same statement (e.g., y_5); in Case 2, we merge nodes $(v_1, \cdots, v_{l'})$ and $(v'_1, \cdots, v'_{l'})$ where $l' < k$, and for all $i \in [n]$, v_i and v'_i correspond to the same statement. Since two same clauses will not occur in the expression of $\mathcal{R}_{k\text{-CNF},S'_k}$, there are no prefixes with length k, which implies that the number of k-size paths will not decrease. In addition, the merging operation is from the beginning node, so the length of each path is still k.

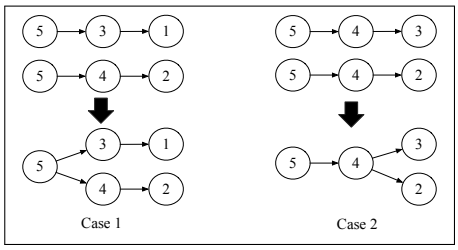


Fig. 18: Merging prefixes of kCNFtoDAG
(the numbers in the cycles represent the statements)

Secondly, we note that after Step 3 the DAG also has Property-(i) and Property-(ii). If not, then there are "wrong" crossing edges (since the length of paths and statements that nodes correspond to will not be changed by the operation of merging suffixes). In other words, there exists a node v_l such that a path (consisting of some prefix of v_l, v_l itself, and some suffix of v_l) is a new path when compared with the DAG after Step 2 (and before Step 3). However, it is contradictory to the condition iii) of merging suffixes (i.e., Step 3) in Sec. 4.

Therefore, the claim holds. In all, the output of kCNFtoDAG satisfies the requirements of Property-(i) and Property-(ii). □

C.2 Proof of Theorem 2

Proof (Proof of Theorem 2). Given a graph G output by kCNFtoDAG in Sec. 4, we analyze the upper bound of the number of nodes.

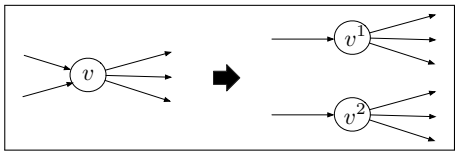


Fig. 19: A node with multiple paths

We consider the problem in another way. Take the node v in Fig. 19 for example. The in-degree of node v is 2 and out-degree is 3. If we do not want to have "wrong" crossing edges, then we need to duplicate the node at most $\mathsf{Min}(\mathsf{in\text{-}deg}(v), \mathsf{out\text{-}deg}(v))$ times, i.e., 2 times here, where Min returns the minimum value.

Then we discuss how many nodes the prover needs to duplicate at most for each statement. Given statement y_l ($l \in [n]$), we first consider the case when $2 \leq k < \frac{n+1}{2}$.

1. When $1 \leq l < k$, recalling that the statements in each Type-$\vee$ clause are sorted from the smallest to the largest, we assume that there are i statements before y_l in some clauses. and then, we have $0 \leq i \leq l-1$. Given a fixed value i, in the graph G output by $\mathsf{kCNFtoDAG}$, the out-degree of the node representing the statement y_l is at most C_{l-1}^{i} as we pick i statements among the statements labelled in $[1, l-1]$, and its in-degree is at most C_{n-l}^{k-1-i} as we pick $(k-1-i)$ statements among the statements labelled in $[l+1, n]$, so we need to duplicate at most $\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i})$. Therefore, the total number of nodes duplicated for y_l would be at most $\sum_{i=0}^{l-1} \mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i})$.
2. When $k \leq l \leq n-k+1$, similar to the above case, the total number of nodes would be at most $\sum_{i=0}^{k-1} \mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i})$.
3. When $n-k+1 < l \leq n$, considering that there are i statements after y_l in some Type-$\vee$ clauses, we have $0 \leq i \leq n-l$. Similarly, given i, i.e., y_l is the $(k-i)^{th}$ statements in the clause, the out-degree is at most C_{l-1}^{k-1-i} as we pick $(k-1-i)$ statements among the statements labelled in $[1, l-1]$, and the in-degree is at most C_{n-l}^{i} as we pick i statements among the statements labelled in $[l+1, n]$. Thus, we need to duplicate at most $\mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^{i})$ when given i. Therefore, the total number of nodes would be at most $\sum_{i=0}^{n-l} \mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^{i})$.

Thus, the total number of nodes in G is $|V| \leq \sum_{l=1}^{k-1} \sum_{i=0}^{l-1} \mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) + \sum_{l=k}^{n-k+1} \sum_{i=0}^{k-1} \mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) + \sum_{l=n-k+2}^{n} \sum_{i=0}^{n-l} \mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^{i})$.

In the following, we show that We have the following lemma.

Lemma 1. *When $2 \leq k < \frac{n+1}{2}$, we have $|V| \leq 2^k + 2(n-2k+1) + (n-2k+2)C_n^{\lfloor \frac{k}{2} \rfloor + 1}$.*

Proof. First, we have

$$
\begin{aligned}
|V| =& \sum_{l=1}^{k-1}\sum_{i=0}^{l-1}\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) + \sum_{l=k}^{n-k+1}\sum_{i=0}^{k-1}\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) \\
&+ \sum_{l=n-k+2}^{n}\sum_{i=0}^{n-l}\mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^{i}) && (20)\\
\leq& \sum_{l=1}^{k-1}\sum_{i=0}^{l-1} C_{l-1}^{i} + \sum_{l=k}^{n-k+1}\sum_{i=0}^{\lfloor\frac{k}{2}\rfloor}(C_{l-1}^{i} + C_{n-l}^{i}) + \sum_{l=n-k+2}^{n}\sum_{i=0}^{n-l} C_{n-l}^{i} && (21)\\
=& \sum_{l=1}^{k-1} 2^{l-1} + \sum_{l=k}^{n-k+1}\sum_{i=0}^{\lfloor\frac{k}{2}\rfloor}(C_{l-1}^{i} + C_{n-l}^{i}) + \sum_{l=n-k+2}^{n} 2^{n-l} && (22)\\
=& 2^{k} - 2 + \sum_{l=k}^{n-k+1}\sum_{i=0}^{\lfloor\frac{k}{2}\rfloor}(C_{l-1}^{i} + C_{n-l}^{i}). && (23)
\end{aligned}
$$

We explain why Eq. (21) holds. For the first and third parts, i.e., the sum $\sum_{l=1}^{k-1}\sum_{i=0}^{l-1}\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i})$ and $\sum_{l=n-k+2}^{n}\sum_{i=0}^{n-l}\mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^{i})$, since $\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) \leq C_{l-1}^{i}$ and $\mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^{i}) \leq C_{n-l}^{i}$, we have

$$
\begin{aligned}
\sum_{l=1}^{k-1}\sum_{i=0}^{l-1}\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) \leq \sum_{l=1}^{k-1}\sum_{i=0}^{l-1} C_{l-1}^{i},\\
\sum_{l=n-k+2}^{n}\sum_{i=0}^{n-l}\mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^{i}) \leq \sum_{l=n-k+2}^{n}\sum_{i=0}^{n-l} C_{n-l}^{i}.
\end{aligned}
$$

For the second part, i.e., the sum $\sum_{l=k}^{n-k+1}\sum_{i=0}^{k-1}\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i})$, we divide the range of i, i.e., $[0, k-1]$, into two parts for every $l \in [k, n-k+1]$: (1) when $0 \leq i \leq \lfloor\frac{k}{2}\rfloor$, it is clear that $\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) \leq C_{l-1}^{i}$; (2) when $\lfloor\frac{k}{2}\rfloor + 1 \leq i \leq k-1$, it is clear that $\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) \leq C_{n-l}^{k-1-i}$. When $\lfloor\frac{k}{2}\rfloor + 1 \leq i \leq k-1$, if we set $t = k-1-i$, then we have $C_{n-l}^{k-1-i} = C_{n-l}^{t}$. Then, the range of t is $0 \leq t \leq k-2-\lfloor\frac{k}{2}\rfloor \leq \lfloor\frac{k}{2}\rfloor$. Therefore, we have $\sum_{l=k}^{n-k+1}\sum_{i=0}^{k-1}\mathsf{Min}(C_{l-1}^{i}, C_{n-l}^{k-1-i}) \leq \sum_{l=k}^{n-k+1}\sum_{i=0}^{\lfloor\frac{k}{2}\rfloor}(C_{l-1}^{i} + C_{n-l}^{i})$.

Then, we have the following Lemma.

Lemma 2. *If $1 \leq i \leq \mathsf{Min}(l-1, n-l)$, then we have $C_{l-1}^{i} + C_{n-l}^{i} \leq C_{n-i}^{i}$.*

Proof. For some m ($m > i$), we first define a function $f(x) = (i-1+x)(i-2+x)\ldots(1+x)x$ where $x \in [0, m-i]$. Then we define another function $g(x) = f(x) + f(m-i-x)$ where $x \in [0, m-i]$. We can rewrite $f(x)$ as $f(x) = \sum_{j=0}^{i} a_j x^j$ where $a_j \geq 1$, which implies that its first derivative $f'(x) > 0$ and its second derivative $f''(x) \geq 0$ for all $x \in [0, m-i]$. Hence, the second derivative of $g(x)$ is $g''(x) = f''(x) + f''(m-i-x)$, which is greater than or equal to 0. Therefore,

$g(x)$ is a convex function over $[0, m-i]$ and we have the following formula with respect to the definition of convex function,

$$\forall \lambda \in [0,1] : g(\lambda(m-i)) \leq (1-\lambda)g(0) + \lambda g(m-i).$$

It is clear that $g(0) = g(m-i)$, so for all $\lambda \in [0,1] : g(\lambda(m-i)) \leq g(0) = (m-1)\dots(m-i)$, which implies that the maximum value of $g(x)$ over $[0, m-i]$ is $(m-1)\dots(m-i)$.

Let $m = n-i+1$, then $m-i = n-2i+1 = n-l+l-1-2i+2$ for some l such that $1 \leq l \leq n$. Since $i \leq \mathsf{Min}(l-1, n-l)$, we have $i \leq l-1$ and $i \leq n-l$. Therefore, $m-i = (n-l-i)+(l-1-i)+2 > 0$. Then, we have

$$\begin{aligned}
&C_{l-1}^i + C_{n-l}^i \\
&= \frac{(l-1)(l-1-1)\dots(l-1-i+1) + (n-l)(n-l-1)\dots(n-l-i+1)}{i!} \\
&= \frac{f(l-i) + f(n-l-i+1)}{i!} && //\text{let } m = n-i+1 \\
&= \frac{f(l-i) + f((n-i+1)-i-(l-i))}{i!} \\
&= \frac{g(l-i)}{i!} && //(l-i) \in [0, (n-i+1)-i] \\
&\leq \frac{(n-i)(n-i-1)\dots(n-2i+1)}{i!} \\
&= C_{n-i}^i.
\end{aligned}$$

Note that for $g(l-i)$, it is required that $(l-i) \in [0, (n-i+1)-i]$ according to the definition of $g(x)$. Since $i \leq \mathsf{Min}(l-1, n-l)$, then we have $i \leq (l-1)$ and $i \leq (n-l)$, so we have $0 < 1 \leq (l-i)$ and $(l-i) \leq (n-2i) < (n-i+1)-i$. Therefore, $(l-i) \in [0, (n-i+1)-i]$. Hence, we complete the proof. □

When $2 \leq k \leq l \leq n-k+1$, we have $\lfloor \frac{k}{2} \rfloor \leq k-1 \leq l-1 \leq n-k$ and $\lfloor \frac{k}{2} \rfloor \leq k-1 \leq n-l \leq n-k$. Therefore, when $1 \leq i \leq \lfloor \frac{k}{2} \rfloor$, we have $1 \leq i \leq \mathsf{Min}(l-1, n-l)$. Thus, by Lemma 2, we have

$$\begin{aligned}
|V| &\leq 2^k - 2 + \sum_{l=k}^{n-k+1} \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} (C_{l-1}^i + C_{n-l}^i) \leq 2^k - 2 + \sum_{l=k}^{n-k+1} (2 + \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} C_{n-i}^i) \\
&= 2^k + 2(n-2k+1) + \sum_{l=k}^{n-k+1} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} C_{n-i}^i.
\end{aligned}$$

In addition, we have

$$\begin{aligned}
1 \le i &\le \lfloor \frac{k}{2} \rfloor \\
&\le \frac{n}{2} - \lfloor \frac{k}{2} \rfloor && //k < \frac{n+1}{2} \Rightarrow 2\lfloor \frac{k}{2} \rfloor \le k \le \frac{n}{2} \\
&\le \frac{n-k+1}{2} && //\lfloor \frac{k}{2} \rfloor \ge \frac{k-1}{2} \\
&\le \frac{n-i}{2} && //i \le \lfloor \frac{k}{2} \rfloor \le k-1
\end{aligned}$$

so we hold $i \le \lfloor \frac{k}{2} \rfloor \le \frac{n-i}{2}$. Further, because of the monotonicity of combinatorial number, we get

$$\begin{aligned}
|V| &\le 2^k + 2(n-2k+1) + \sum_{l=k}^{n-k+1} \sum_{i=1}^{\lfloor \frac{k}{2} \rfloor} C_{n-i}^{\lfloor \frac{k}{2} \rfloor} \\
&\le 2^k + 2(n-2k+1) + \sum_{l=k}^{n-k+1} \sum_{i=1}^{n-\lfloor \frac{k}{2} \rfloor} C_{n-i}^{\lfloor \frac{k}{2} \rfloor}.
\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
C_n^m =& C_{n-1}^m + C_{n-1}^{m-1} = C_{n-1}^m + C_{n-2}^{m-1} + C_{n-2}^{m-2} = \dots \\
=& C_{n-1}^m + C_{n-2}^{m-1} + C_{n-3}^{m-2} + \dots + C_{n-m}^{m-(m-1)} + C_{n-m}^{m-m} \\
=& C_{n-1}^{m-1+1} + C_{n-2}^{m-2+1} + C_{n-3}^{m-3+1} + \dots + C_{n-m}^{m-m+1} + C_{n-(m+1)}^{m-(m+1)+1} \\
& && //C_{n-m}^{m-m} = 1 = C_{n-(m+1)}^{m-(m+1)+1} \\
=& \sum_{i=1}^{m+1} C_{n-i}^{m-i+1} = \sum_{i=1}^{m+1} C_{n-i}^{n-m-1}.
\end{aligned}$$

In other words, $\sum_{i=1}^{n-t} C_{n-i}^t = C_n^{n-t-1} = C_n^{t+1}$ if we set $t = n-m-1$. Therefore, $|V| \le 2^k + 2(n-2k+1) + \sum_{l=k}^{n-k+1} C_n^{\lfloor \frac{k}{2} \rfloor + 1} = 2^k + 2(n-2k+1) + (n-2k+2) C_n^{\lfloor \frac{k}{2} \rfloor + 1}$. □

Analogous to the analysis of the case when $2 \le k < \frac{n+1}{2}$, we have the following result for $\frac{n+1}{2} \le k \le n-1$. When $1 \le l < n-k+1$, the total number of nodes would be at most $\sum_{i=0}^{l-1} \mathsf{Min}(C_{l-1}^i, C_{n-l}^{k-1-i})$. When $n-k+1 \le l \le k$, the total number of nodes would be at most $\sum_{i=k-l}^{n-l} \mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^i)$. When $k < l \le n$, the total number of nodes would be at most $\sum_{i=0}^{n-l} \mathsf{Min}(C_{l-1}^{k-1-i}, C_{n-l}^i)$. Hence, with similar analysis, $|V| \le 2^{n-k+1} + 2(2k-n-1) + (2k-n) C_n^{\lfloor \frac{n-k+1}{2} \rfloor + 1}$. In all, $|V|$ satisfies the following inequality.

$$|V| \le V_{\text{bound}} = 2^d + 2(n-2d+1) + (n-2d+2) C_n^{\lfloor \frac{d}{2} \rfloor + 1} \begin{cases} d = k & (2 \le k < \frac{n+1}{2}) \\ d = n-k+1 & (\frac{n+1}{2} \le k \le n-1) \end{cases} \tag{24}$$

On the other hand, from the description of kCNFtoDAG in Sec. 4, it can be inferred that there will be $(k \cdot num)$ nodes after the first step, so $|V| \leq (k \cdot num)$ after the merging operations. Hence, $|V| \leq \mathsf{Min}(V_{\text{bound}}, (k \cdot num))$. □

The upper bound V_{bounds} in Eq. (24) only depends on k and n, while num depends on the specific structure of $\mathcal{R}$, so it seems difficult to directly compare V_{bound} in Eq. (24) with $(k \cdot num)$. When given k and n, V_{bound} is fixed and num can range from 1 to C_n^k, so here we make a rough comparison in the worst case (i.e., we set $num = C_n^k$). Fig. 20 shows the comparison when $2 \leq k < \frac{n+1}{2}$ (for the case $\frac{n+1}{2} \leq k \leq n-1$, the comparison is similar), it shows that V_{bound} is much smaller than $k \cdot C_n^k$ in most cases, and the value $\frac{V_{\text{bound}}}{k \cdot C_n^k}$ drops rapidly as k increases. Thus, we can draw such a conclusion that V_{bound} is smaller than $k \cdot num$ in most cases.

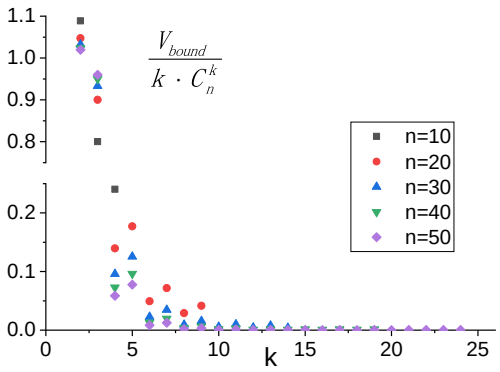


Fig. 20: $\frac{V_{\text{bound}}}{k \cdot C_n^k}$ $(2 \leq k < \frac{n+1}{2}$, and since the y-axis values of the points do not fall in the range $[0.25, 0.75]$, we remove this part in the figure for simplicity.)

In fact, the exact number $|V|$ can be further smaller than the theoretical result in Eq. (24), because some inequality tricks in the proof (of Lemma 1) make the theoretical result larger. Hence, the protocol can have a better performance in practice than in theoretical analysis.

D Improved kCNFtoDAG

Given the relation $\mathcal{R}_{k\text{-CNF},S'_k}$ as input, kCNFtoDAG outputs a directed acyclic graph, which satisfies the security requirements as needed in our Sigma protocol. However, the complexity of the running time of the algorithm introduced in Sec. 4 is $O(k \cdot num)$, where k is the number of statement in each clause and num is the number of clauses, so it is time consuming when k and num get very large. Hence, in this section, we improve the algorithm.

The main problem is that merging is a time consuming operation, even compared with the total running time of scalar computation over the elliptic curve.

kCNFtoDAG($\mathcal{R}$):
//$\mathcal{R}.clauses$: parse $\mathcal{R}$ as clauses and every clause contains the statement labels $(i_1, \ldots, i_k)$ where $1 \leq i_1 < \ldots < i_k \leq n$.
  $G \leftarrow$ Nodes(n, k)
  For every $(i_1, \ldots, i_k) \in \mathcal{R}.clauses$:
    If Label$(i_1, 1) = 0$:
      $G[i_1]$[Prefix$((i_1, \ldots, i_k), i_1)] = 1$
      $current \leftarrow (i_1,$ Prefix$((i_1, \ldots, i_k), i_1))$
    Else
      $G[i_1]$[Suffix$((i_1, \ldots, i_k), i_1)] = 1$
      $current \leftarrow (i_1,$ Suffix$((i_1, \ldots, i_k), i_1))$
    For $j = 2$ to k:
      If Label$(i_j, j) = 0$:
        $G[i_j]$[Prefix$((i_1, \ldots, i_k), i_j)] = 1$
        $next \leftarrow (i_j,$ Prefix$((i_1, \ldots, i_k), i_j))$
      Else
        $G[i_j]$[Suffix$((i_1, \ldots, i_k), i_j)] = 1$
        $next \leftarrow (i_j,$ Suffix$((i_1, \ldots, i_k), i_j))$
    Draw an arrow from $current$ to $next$
    $current \leftarrow next$
  Return G

AddNodes(l,t,G):
  $label =$ Label(l, t) //$i_t = l$
  If $label = 0$:
    For every $(i_1, \ldots, i_{t-1}) \in [n]^{t-1}$:
    //$1 \leq i_1 < \ldots < i_{t-1} \leq l$
      $G[l][(i_1, \ldots, i_{t-1})] = 0$
  If $label = 1$:
    For every $(i_{t+1}, \ldots, i_k) \in [n]^{k-t}$:
    //$l \leq i_{t+1} < \ldots < i_k \leq n$
      $G[l][(i_{t+1}, \ldots, i_k)] = 0$
  Return G

Nodes(n,k):
  $G \leftarrow \perp$
  For $l = 1$ to n:
    If $2 \leq k < \frac{n+1}{2}$:
      If $1 \leq l < k$:
        For $t = 1$ to l:
          $G \leftarrow$ AddNodes(l, t, G)
      If $k \leq l \leq n - k + 1$:
        For $t = 1$ to k:
          $G \leftarrow$ AddNodes(l, t, G)
      If $n - k + 1 < l \leq n$:
        For $t = k - n + l$ to k:
          $G \leftarrow$ AddNodes(l, t, G)
    If $\frac{n+1}{2} \leq k \leq n - 1$:
      If $1 \leq l < n - k + 1$:
        For $t = 1$ to l:
          $G \leftarrow$ AddNodes(l, t, G)
      If $n - k + 1 \leq l \leq k$:
        For $t = k - n + l$ to l:
          $G \leftarrow$ AddNodes(l, t, G)
      If $k < l \leq n$:
        For $t = k - n + l$ to k:
          $G \leftarrow$ AddNodes(l, t, G)
  Return G

Prefix$((i_1, \ldots, i_k), i_j)$:
//$1 \leq i_1 < \ldots < i_k \leq n$ and $j \in [k]$
  Return $(i_1, \ldots, i_{j-1})$

Suffix$((i_1, \ldots, i_k), i_j)$:
  If $j = k$: Return "$"
  //"$" denotes the empty string of suffixes
  Else Return $(i_{j+1}, \ldots, i_k)$

Label(l, j):
//the l^{th} statement is in the j^{th} place in some clause, $l \in [n]$ and $j \in [k]$
  If $(C_{l-1}^{j-1} \leq C_{n-l}^{k-j})$: Return 0
  Else Return 1

Fig. 21: Algorithm description of improved kCNFtoDAG

The difference between the algorithms in Sec. 4 and here is that the improved kCNFtoDAG here does not need to merge as many nodes as the kCNFtoDAG in Sec. 4 does.

Based on the analysis in Appendix C, it can be referred that usually the total number of the nodes in the directed acyclic graph is much smaller than $(k \cdot num)$, especially when num is close to C_n^k. Therefore, if we just prepare as many nodes as the analysis, then we can reduce merging operations. As a result, no matter from the view of storage or from the view of computational complexity, the algorithm kCNFtoDAG is improved a lot. The detailed algorithms are introduced in Fig. 21.

E Security proof of $\Sigma^{\mathcal{R}_{1\text{-OR}}}$

We analyze computational knowledge soundness, special HVZK and witness indistinguishability of Sigma protocol $\Sigma^{\mathcal{R}_{1\text{-OR}}} = (\mathcal{P}, \mathcal{V})$ constructed in Sec. 5.1.

Computational knowledge soundness. A deterministic algorithm Ext is constructed as follows. For two accepting transcripts (a, c, z) and $(a, \tilde{c}, \tilde{z})$ where $c \neq \tilde{c}$, we parse $\{z_l\}_{l\in[k]} \leftarrow z$ and $\{\tilde{z}_l\}_{l\in[k]} \leftarrow \tilde{z}$. Note that taking $(\mathbf{y}, a, c, z)$ (resp., $(\mathbf{y}, a, \tilde{c}, \tilde{z})$) as input, $\mathcal{V}_2$ will compute $\{a''_l\}_{l\in[k]}$ (resp., $\{\tilde{a}''_l\}_{l\in[k]}$). Now, we provide the following claim with a postponed proof.

Claim. With overwhelming probability, there is some index $\hat{l}$ ($\hat{l} \in [k]$) such that $z_{\hat{l}} \neq \tilde{z}_{\hat{l}}$ and $a''_{\hat{l}} = \tilde{a}''_{\hat{l}}$.

We find the largest index $l \in [k]$ satisfying $z_l \neq \tilde{z}_l$ and $a''_l = \tilde{a}''_l$, and also denote it as $\hat{l}$ for simplicity. If $\hat{l} = k$, we extract x_k: $x_k \leftarrow (\tilde{z}_k - z_k)/(\tilde{c} - c)$, else we extract the witness $x_{\hat{l}}(\hat{l} < k)$ in this way: $x_{\hat{l}} \leftarrow (\tilde{z}_{\hat{l}} - z_{\hat{l}})/(\varphi(\tilde{a}''_{\hat{l}+1}) - \varphi(a''_{\hat{l}+1}))$, where $a''_{\hat{l}+1}$ and $\tilde{a}''_{\hat{l}+1}$ are computed by $\mathcal{V}_2$ when taking $(\mathbf{y}, a, c, z)$ and $(\mathbf{y}, a, \tilde{c}, \tilde{z})$ as input respectively (see Fig. 12).

Finally, Ext outputs $x_{\hat{l}}$. When $\hat{l} = k$, $a''_k = \tilde{a}''_k$ implies $g^{z_k}/y_k^c = g^{\tilde{z}_k}/y_k^{\tilde{c}}$, so we have $y_k^{\tilde{c}-c} = g^{\tilde{z}_k - z_k}$. Therefore, $\log_g y_k = (\tilde{z}_k - z_k)/(\tilde{c} - c) = x_k$. Hence, the extracted witness x_k is the witness of the statement y_k. When $\hat{l} < k$, $a''_{\hat{l}} = \tilde{a}''_{\hat{l}}$ implies $g^{z_{\hat{l}}}/y_{\hat{l}}^{\varphi(a''_{\hat{l}+1})} = g^{\tilde{z}_{\hat{l}}}/y_{\hat{l}}^{\varphi(\tilde{a}''_{\hat{l}+1})}$, so we have $y_{\hat{l}}^{\varphi(\tilde{a}''_{\hat{l}+1}) - \varphi(a''_{\hat{l}+1})} = g^{\tilde{z}_{\hat{l}} - z_{\hat{l}}}$. Therefore, if $\varphi(\tilde{a}''_{\hat{l}+1}) \neq \varphi(a''_{\hat{l}+1})$, then we have $\log_g y_{\hat{l}} = (\tilde{z}_{\hat{l}} - z_{\hat{l}})/(\varphi(\tilde{a}''_{\hat{l}+1}) - \varphi(a''_{\hat{l}+1})) = x_{\hat{l}}$. Hence, when $\hat{l} < k$, the extracted witness $x_{\hat{l}}$ is the witness of the statement $y_{\hat{l}}$ if $\varphi(\tilde{a}''_{\hat{l}+1}) \neq \varphi(a''_{\hat{l}+1})$.

Therefore, the output $x_{\hat{l}}$ of Ext is the witness of the statement $y_{\hat{l}}$.

In the following, we analyze the success probability of Ext.

Firstly, the inequality $c \neq \tilde{c}$ guarantees that we can extract x_k when $\hat{l} = k$. Now we show that $\varphi(\tilde{a}''_{\hat{l}+1}) \neq \varphi(a''_{\hat{l}+1})$ with overwhelming probability. Note that if $a''_{\hat{l}+1}$ equals $\tilde{a}''_{\hat{l}+1}$ when $\hat{l} < k$, then we get $z_{\hat{l}+1} = \tilde{z}_{\hat{l}+1}$ according to the definition of index $\hat{l}$, which implies that $a''_{\hat{l}+2}$ equals $\tilde{a}''_{\hat{l}+2}$ otherwise we find a collision for collision-resistant hash function φ. By recursion, we can get $a''_k = \tilde{a}''_k$ and

$z_k = \tilde{z}_k$, which is contradictory to that $c \neq \tilde{c}$. Thus, $a''_{\hat{l}+1} \neq \tilde{a}''_{\hat{l}+1}$, which implies that $\varphi(\tilde{a}''_{\hat{l}+1}) \neq \varphi(a''_{\hat{l}+1})$ with overwhelming probability. Therefore, when $\hat{l} < k$, it is also guaranteed that we can extract $x_{\hat{l}}$ successfully.

Secondly, we prove the above claim.

Proof. We assume that there is not any index $\hat{l}$ ($\hat{l} \in [k]$) such that $z_{\hat{l}} \neq \tilde{z}_{\hat{l}}$ and $a_{\hat{l}} = \tilde{a}_{\hat{l}}$. In other words, for every $1 \leq l \leq k$, there are two cases:

1. $a''_l = \tilde{a}''_l$ and $z_l = \tilde{z}_l$;
2. or $a''_l \neq \tilde{a}''_l$.

Note that $a''_1 = a = \tilde{a}''_1$, so according to the assumption, we have $z_1 = \tilde{z}_1$. Further, $a''_1 = g^{z_1}/y_1^{\varphi(a''_2)}$ and $\tilde{a}''_1 = g^{\tilde{z}_1}/y_1^{\varphi(\tilde{a}''_2)}$ implies that $g^{z_1}/y_1^{\varphi(a''_2)} = g^{\tilde{z}_1}/y_1^{\varphi(\tilde{a}''_2)}$. Thus, we can know that $a''_2 = \tilde{a}''_2$ with overwhelming probability. So, according to the assumption, we have $z_2 = \tilde{z}_2$. By recursion, we have $a''_k = \tilde{a}''_k$ and $z''_k = \tilde{z}''_k$, which is contradictory to $c \neq \tilde{c}$. Therefore, the assumption does not hold. In other words, there is some index $\hat{l}$ ($\hat{l} \in [k]$) such that $z_{\hat{l}} \neq \tilde{z}_{\hat{l}}$ and $a_{\hat{l}} = \tilde{a}_{\hat{l}}$ with overwhelming probability.

In all, the algorithm Ext can extract a witness with overwhelming probability, which implies that the Sigma protocol $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ enjoys computational knowledge soundness.

Special HVZK. The simulator Sim is shown in Fig. 22. It is easy to check that the transcript generated by Sim can be accepted by the honest verifier $\mathcal{V}$. So we just prove that the distribution of the transcript generated by Sim is the same as that of the transcript between $\mathcal{P}(x, y)$ and $\mathcal{V}(y)$.

$\mathsf{Sim}(\mathbf{y}, c)$:
$z_k \leftarrow \mathbb{Z}_p^*$, $a_k \leftarrow g^{z_k}/y_k^c$
For $l = k-1$ to 1: $z_l \leftarrow \mathbb{Z}_p^*$, $a_l \leftarrow g^{z_l}/y_l^{\varphi(a_{l+1})}$
$a \leftarrow a_1$, $z \leftarrow (\{z_l\}_{l\in[k]})$, Return (a, z)

Fig. 22: HVZK simulator of $\Sigma^{\mathcal{R}_{1\text{-OR}}}$

For the transcript (a, c, z) generated in $\Sigma^{\mathcal{R}_{1\text{-OR}}}$, we claim that c and z_l ($1 \leq l \leq k$) are independent, with c uniformly distributed over $\mathcal{CL}$ (i.e., $\mathbb{Z}_p^*$ here) and z_l uniformly distributed over $\mathbb{Z}_p^*$. Note that c is randomly picked by $\mathcal{V}$ and for $l \neq \mu$, it is easy to get that z_l is uniformly distributed over $\mathbb{Z}_p^*$. When $l = \mu$ and $\mu = k$, $z_k = r + x_k c$ and r is randomly picked over $\mathbb{Z}_p^*$, so z_k is uniformly distributed over $\mathbb{Z}_p^*$. When $l = \mu$ but $\mu \neq k$, $z_\mu = \hat{z}_\mu + (\varphi(a'_{\mu+1}) - \varphi(a_{\mu+1}))x_\mu$, so z_μ is also uniformly distributed over $\mathbb{Z}_p^*$. Therefore, our claim holds.

Then, given z_k and c, a''_k is uniquely determined by $a''_k = g^{z_k}/y_k^c$. By recursion, for every $l < k$, given z_l, a''_l is uniquely determined by $a''_l = g^{z_l}/y_l^{\varphi(a''_{l+1})}$.

For the transcript (a, c, z) generated in the simulation, c and z_l $(1 \leq l \leq k)$ are randomly picked over $\mathbb{Z}_p^*$. On the other hand, $a_k = g^{z_k}/y_k^c$ and $a_l = g^{z_l}/y_l^{\varphi(a_{l+1})}$ for every $l < k$.

Thus, the distribution of the transcript generated by Sim is the same as that of the transcript between $\mathcal{P}(x, y)$ and $\mathcal{V}(y)$. So the Sigma protocol $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ is special HVZK.

Witness indistinguishability. According to the following theorem, the Sigma protocol $\Sigma^{\mathcal{R}_{1\text{-OR}}}$ is perfect witness indistinguishable (WI).

Theorem 5. ([14]). *Every Sigma protocol is perfect WI.*

F Proof of the Claim in Sec. 5.2

Recall that the claim in Sec. 5.2 is that for all $v \in V \backslash S^{\mathsf{sink}}$, $a''_v = a'_v$ or $b''_v = b'_v$. The proof is as follows.

Proof. Let $S^{(0)} := S^{\mathsf{source}}$, and for all $i \leq k-1$, $S^{(i)}$ denote the vertex set that contains all tails of the edges that are pointed from the vertices of $S^{(i-1)}$ (i.e., $S^{(i)}$ is the set of all the $(i+1)^{th}$ vertices in all the paths from S^{source} to S^{sink}). Note that $S^{(i)} \cap S^{(j)} = \emptyset$ for all $0 \leq i < j \leq k-1$, since it is required that the lengths of all paths from the node in S^{source} to the node in S^{sink} are k. Specially, $S^{(k-1)} = S^{\mathsf{sink}}$, and for this claim, we only need to prove it in the cases of $0 \leq i \leq k-2$.

First of all, we note that for any $v_0 \in S^{(0)}$ (i.e., $v_0 \in S^{\mathsf{source}}$), $a''_{v_0} = g^{z_{v_0}}/y^c_{f(v_0)} = a'_{v_0}$.

Then, for $0 \leq i \leq k-3$, assume that for any $v_i \in S^{(i)}$, $a''_{v_i} = a'_{v_i}$ or $b''_{v_i} = b'_{v_i}$. Now, we aim to show that for any $v_{i+1} \in S^{(i+1)}$, $b''_{v_{i+1}} = b'_{v_{i+1}}$.

We firstly notice that in this case, $m''_{v_{i+1}} = m'_{v_{i+1}}$, since for any $v_i \in S^{(i)}$, $a''_{v_i} = a'_{v_i}$ or $b''_{v_i} = b'_{v_i}$.

1. If $f(v_{i+1}) \in S^{\mathrm{w}}_{\mathbf{x}}$, there are two cases:
   (1) if there is some $v' \in S^{\mathsf{pred}}_{v_{i+1}}$ (note that $S^{\mathsf{pred}}_{v_{i+1}} \subset S^{(i)}$) such that $a'_{v'} \neq a_{v'}$ or $b'_{v'} \neq b_{v'}$ (i.e., $m_{v_{i+1}} \neq m'_{v_{i+1}}$), then

$$
\begin{aligned}
b''_{v_{i+1}} &= g^{z_{v_{i+1}}}/y_{f(v_{i+1})}^{\varphi(m''_{v_{i+1}})} && (25)\\
&= g^{z_{v_{i+1}}}/y_{f(v_{i+1})}^{\varphi(m'_{v_{i+1}})} && (26)\\
&= g^{\hat{z}_{v_{i+1}}+(\varphi(m'_{v_{i+1}})-\varphi(m_{v_{i+1}}))x_{f(v_{i+1})}}/y_{f(v_{i+1})}^{\varphi(m'_{v_{i+1}})} && (27)\\
&= g^{\hat{z}_{v_{i+1}}}/y_{f(v_{i+1})}^{\varphi(m_{v_{i+1}})} = b_{v_{i+1}} && (28)\\
&= b'_{v_{i+1}}, && (29)
\end{aligned}
$$

   where Eq. (25) and (28) are trivial, Eq. (26) is because $m''_{v_{i+1}} = m'_{v_{i+1}}$, and Eqs. (27) and (29) are both obtained from the procedures of $\mathcal{P}_2$;

(2) if for all $v' \in S^{\mathsf{pred}}_{v_{i+1}}$, $a'_{v'} = a_{v'}$ or $b'_{v'} = b_{v'}$ (i.e., $m_{v_{i+1}} = m'_{v_{i+1}}$), then according to the procedures of $\mathcal{P}_2$, we have $z_{v_{i+1}} = \hat{z}_{v_{i+1}}$ and $b'_{v_{i+1}} = b_{v_{i+1}}$, so $b''_{v_{i+1}} = g^{z_{v_{i+1}}} / y_{f(v_{i+1})}^{\varphi(m''_{v_{i+1}})} = g^{\hat{z}_{v_{i+1}}} / y_{f(v_{i+1})}^{\varphi(m_{v_{i+1}})} = b_{v_{i+1}} = b'_{v_{i+1}}$.

2. If $f(v_{i+1}) \notin S^{\mathrm{w}}_{\mathbf{x}}$, then we have:
   (1) If there is some $v' \in S^{\mathsf{pred}}_{v_{i+1}}$ satisfying $a'_{v'} \neq a_{v'}$ or $b'_{v'} \neq b_{v'}$ (i.e., $m_{v_{i+1}} \neq m'_{v_{i+1}}$), then according to the procedures of $\mathcal{P}_2$, we have $b'_{v_{i+1}} = g^{z_{v_{i+1}}} / y_{f(v_{i+1})}^{\varphi(m'_{v_{i+1}})} = g^{z_{v_{i+1}}} / y_{f(v_{i+1})}^{\varphi(m''_{v_{i+1}})}$, so $b''_{v_{i+1}} = b'_{v_{i+1}}$.
   (2) If for all $v' \in S^{\mathsf{pred}}_{v_{i+1}}$, $a'_{v'} = a_{v'}$ or $b'_{v'} = b_{v'}$ (i.e., $m_{v_{i+1}} = m'_{v_{i+1}}$), then according to the procedures of $\mathcal{P}_2$, we have $z_{v_{i+1}} = \hat{z}_{v_{i+1}}$ and $b'_{v_{i+1}} = b_{v_{i+1}}$, so $b''_{v_{i+1}} = g^{z_{v_{i+1}}} / y_{f(v_{i+1})}^{\varphi(m''_{v_{i+1}})} = g^{\hat{z}_{v_{i+1}}} / y_{f(v_{i+1})}^{\varphi(m_{v_{i+1}})} = b_{v_{i+1}} = b'_{v_{i+1}}$.

Hence, we have shown that for any $v_{i+1} \in S^{(i+1)}$, $b''_{v_{i+1}} = b'_{v_{i+1}}$. This completes the induction step, and the proof. □

G Proof of Theorem 3

Computational knowledge soundness. To show that the Sigma protocol $\Sigma_{\mathrm{DAG}}^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}$ provides computational knowledge soundness, we construct an efficient extractor as follows.

Given the statement vector $\mathbf{y}$ and two accepting transcripts for the same commitment a, i.e., (a, c, z) and $(a, \tilde{c}, \tilde{z})$, the extractor firstly sets that $\mathbf{x} := (\perp)^n$, and runs $\mathsf{kCNFtoDAG}(\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k})$ to obtain $G = (V, E)$.

Then, it parses $\{e_v\}_{v \in S^{\mathsf{sink}}} \leftarrow a$, $\{z_v\}_{v \in V} \leftarrow z$ and $\{\tilde{z}_v\}_{v \in V} \leftarrow \tilde{z}$. For every path in G, (i) we denote it as $(v_1, \ldots, v_k)$, where $v_1 \in S^{\mathsf{source}}$ and $v_k \in S^{\mathsf{sink}}$, and we have $f(v_1) > \ldots > f(v_k)$; (ii) for every $v \in \{v_j\}_{j \in [k]}$, the extractor computes a''_v, b''_v, or e''_v (resp. $\tilde{a}''_v$, $\tilde{b}''_v$, or $\tilde{e}''_v$) according to $\mathcal{V}_2$, and finds the largest $f(v)$ such that $z_v \neq \tilde{z}_v$ and $a''_v = \tilde{a}''_v$ (or $b''_v = \tilde{b}''_v$, or $e''_v = \tilde{e}''_v$) (for convenience, denote this vertex by $\hat{v} \in \{v_j\}_{j \in [k]}$); and (iii)

- if $\hat{v} \in S^{\mathsf{source}}$, then the extractor updates $x_{f(\hat{v})} \leftarrow (\tilde{z}_{\hat{v}} - z_{\hat{v}})/(\tilde{c} - c)$;
- otherwise, it updates $x_{f(\hat{v})} \leftarrow (\tilde{z}_{\hat{v}} - z_{\hat{v}})/(\varphi(\tilde{m}''_{\hat{v}}) - \varphi(m''_{\hat{v}}))$, where both $\tilde{m}''_{\hat{v}}$ and $m''_{\hat{v}}$ are computed with msg'' according to $\mathcal{V}_2$ in Fig. 14.

At last, the extractor outputs $\mathbf{x}$.

That's the construction of the extractor. Now we analyze its success probability.

Obviously, for any path $(v_1, \ldots, v_k)$ in G, if there is some $v_{j'}$ ($j' \in [k]$) satisfying $z_{v_{j'}} \neq \tilde{z}_{v_{j'}}$ and $a''_{v_{j'}} = \tilde{a}''_{v_{j'}}$ (or $b''_{v_{j'}} = \tilde{b}''_{v_{j'}}$, or $e''_{v_{j'}} = \tilde{e}''_{v_{j'}}$), and $\varphi(m''_{v_{j'}}) \neq \varphi(\tilde{m}''_{v_{j'}})$ when $j' > 1$, then the extracted (updated) $x_{f(v_{j'})}$ is a witness for $y_{f(v_{j'})}$. Hence, if for each path $(v_1, \ldots, v_k)$ in G, there is such a $v_{j'}$, then the output $\mathbf{x}$ by the extractor is a witness for $\mathbf{y}$.

Therefore, what remains is to show that *(1)* for any path $(v_1, \ldots, v_k)$ in G, the probability that there is some $v \in \{v_j\}_{j \in [k]}$ satisfying $z_v \neq \tilde{z}_v$ and $a''_v = \tilde{a}''_v$

(or $b''_v = \tilde{b}''_v$, or $e''_v = \tilde{e}''_v$) is overwhelming, and *(2)* for this kind of v satisfying that $f(v)$ is the largest in this path, when $v \notin S^{\mathsf{source}}$, $\varphi(m''_v) \neq \varphi(\tilde{m}''_v)$ with overwhelming probability.

To prove *(1)*, assume that for some path $(v_1, \ldots, v_k)$, there is no such vertex (i.e., for all $j \in [k]$, if $a''_{v_j} = \tilde{a}''_{v_j}$ or $b''_{v_j} = \tilde{b}''_{v_j}$ or $e''_{v_j} = \tilde{e}''_{v_j}$, then $z_{v_j} = \tilde{z}_{v_j}$). In this case, note that both (a, c, z) and $(a, \tilde{c}, \tilde{z})$ are accepted transcripts, so $e''_{v_k} = e_{v_k} = \tilde{e}''_{v_k}$, which implies that $z_{v_k} = \tilde{z}_{v_k}$. For those vertices $v \in \{v_j\}_{j\in[k]\setminus\{1\}}$ satisfying both $b''_v = \tilde{b}''_v$ (or $e''_v = \tilde{e}''_v$) and $z_v = \tilde{z}_v$, let $v_{j''}$ $(1 < j'' \leq k)$ denote the vertex where $f(v_{j''})$ is the largest. The fact $b''_{v_{j''}} = \tilde{b}''_{v_{j''}}$ (or $e'_{v_{j''}} = \tilde{e}'_{v_{j''}}$) suggests $\varphi(m''_{v_{j''}}) = \varphi(\tilde{m}''_{v_{j''}})$. Now we consider the following cases:

- if $j'' - 1 = 1$: if $a''_{v_1} = \tilde{a}''_{v_1}$, according to the assumption, we derive that $z_{v_1} = \tilde{z}_{v_1}$. However, note that $c \neq \tilde{c}$, so $z_{v_1} = \tilde{z}_{v_1}$ will imply $a''_{v_1} = g^{z_{v_1}}/y^{c}_{f(v_1)} \neq g^{\tilde{z}_{v_1}}/y^{\tilde{c}}_{f(v_1)} = \tilde{a}''_{v_1}$. Hence, we obtain that $a''_{v_1} \neq \tilde{a}''_{v_1}$.
- if $j'' - 1 > 1$: if $b''_{v_{j''-1}} = \tilde{b}''_{v_{j''-1}}$, according to the assumption, we derive that $z_{v_{j''-1}} = \tilde{z}_{v_{j''-1}}$, contradicting that "$f(v_{j''})$ is the largest" since $f(v_{j''-1}) > f(v_{j''})$. Hence, we obtain that $b''_{v_{j''-1}} \neq \tilde{b}''_{v_{j''-1}}$.

The fact $a''_{v_1} \neq \tilde{a}''_{v_1}$ or $b''_{v_{j''-1}} \neq \tilde{b}''_{v_{j''-1}}$ implies $m''_{v_{j''}} \neq \tilde{m}''_{v_{j''}}$. So we find a collision of φ, the probability of which should be negligible.

Next, we turn to prove *(2)*. For those vertices $v \in \{v_j\}_{j\in[k]}$ satisfying $z_v \neq \tilde{z}_v$ and $a''_v = \tilde{a}''_v$ (or $b''_v = \tilde{b}''_v$, or $e''_v = \tilde{e}''_v$), let $\hat{v} \in \{v_j\}_{j\in[k]}$ denote the vertex where $f(\hat{v})$ is the largest. What remains is to show that when $\hat{v} \notin S^{\mathsf{source}}$, $\varphi(m''_{\hat{v}}) \neq \varphi(\tilde{m}''_{\hat{v}})$ with overwhelming probability. When $\hat{v} \notin S^{\mathsf{source}}$, consider the following two cases:

- if $m''_{\hat{v}} \neq \tilde{m}''_{\hat{v}}$: the collision resistance of φ guarantees that $\varphi(m''_{\hat{v}}) \neq \varphi(\tilde{m}''_{\hat{v}})$ with overwhelming probability.
- if $m''_{\hat{v}} = \tilde{m}''_{\hat{v}}$: let $\widehat{j} \in [k] \setminus \{1\}$ denote the index of $\hat{v}$, i.e., $\hat{v} = v_{\widehat{j}}$. Note that $m''_{\hat{v}} = \tilde{m}''_{\hat{v}}$ implies that $a''_{v_{\widehat{j}-1}} = \tilde{a}''_{v_{\widehat{j}-1}}$ or $b''_{v_{\widehat{j}-1}} = \tilde{b}''_{v_{\widehat{j}-1}}$. Then we derive that $z_{v_{\widehat{j}-1}} = \tilde{z}_{v_{\widehat{j}-1}}$, since $f(\hat{v})$ (i.e., $f(v_{\widehat{j}})$) is the largest. Note that if $\widehat{j} - 1 \neq 1$, $b''_{v_{\widehat{j}-1}} = \tilde{b}''_{v_{\widehat{j}-1}}$ and $z_{v_{\widehat{j}-1}} = \tilde{z}_{v_{\widehat{j}-1}}$ implies that $\varphi(m''_{v_{\widehat{j}-1}}) = \varphi(\tilde{m}''_{v_{\widehat{j}-1}})$, which suggests that $m''_{v_{\widehat{j}-1}} = \tilde{m}''_{v_{\widehat{j}-1}}$ with overwhelming probability. Hence, by recursion, we can get $a''_{v_1} = \tilde{a}''_{v_1}$ and $z_{v_1} = \tilde{z}_{v_1}$, which is contradictory to that $c \neq \tilde{c}$. Therefore, this case occurs with negligible probability.

So we conclude that $\varphi(m''_{\hat{v}}) \neq \varphi(\tilde{m}''_{\hat{v}})$ with overwhelming probability, finishing the proof of *(2)*.

Special HVZK. We construct a PPT simulator Sim as in Fig. 23, where the underlying function msg'' is shown in Fig. 14. It is easy to check that the output of Sim can be accepted by the honest verifier. Now we show that the output distribution of Sim is the same as that of $\Sigma_{\mathrm{DAG}}^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}$.

In $\Sigma_{\mathrm{DAG}}^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}$, we claim that c and z_v $(v \in V)$ are distributed independently, with c uniformly distributed over $\mathcal{CL}$ (i.e., $\mathbb{Z}_p^*$ here) and z_v uniformly distributed

$\underline{\mathsf{Sim}(\mathbf{y}, c)}$:
$G = (V, E) \leftarrow \mathsf{kCNFtoDAG}(\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k})$
For $v \in V$:
If $\mathsf{in\text{-}deg}(v) = 0$: $z_v \leftarrow \mathbb{Z}_p^*$, $a''_v = g^{z_v}/y^c_{f(v)}$
Else If $\mathsf{out\text{-}deg}(v) \neq 0$:
$m''_v \leftarrow \mathsf{msg}''(G, v)$, $z_v \leftarrow \mathbb{Z}_p^*$, $b''_v = g^{z_v}/y^{\varphi(m''_v)}_{f(v)}$
Else $m''_v \leftarrow \mathsf{msg}''(G, v)$, $z_v \leftarrow \mathbb{Z}_p^*$, $e''_v = g^{z_v}/y^{\varphi(m''_v)}_{f(v)}$
Return $a \leftarrow \{e''_v\}_{v \in S^{\mathsf{sink}}}$, $z \leftarrow \{z_v\}_{v \in V}$

Fig. 23: HVZK simulator of $\Sigma^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}_{\mathrm{DAG}}$ (Assume that the "For loops" are executed following a deterministic sequence of nodes.)

over $\mathbb{Z}_p^*$. Note that c is randomly picked by $\mathcal{V}$ and for every $v \in V$, if $f(v) \notin S^{\mathrm{w}}_{\mathbf{x}}$, it is easy to get that z_v is uniformly distributed over $\mathbb{Z}_p^*$. When $f(v) \in S^{\mathrm{w}}_{\mathbf{x}}$ and $v \in S^{\mathsf{source}}$, $z_v = r_v + x_{f(v)}c$ and r_v is randomly picked over $\mathbb{Z}_p^*$, so z_v is uniformly distributed over $\mathbb{Z}_p^*$. When $f(v) \in S^{\mathrm{w}}_{\mathbf{x}}$, $v \notin S^{\mathsf{source}}$ and one of the commitments for the predecessor nodes of node v is changed, $z_v = \hat{z}_v + (\varphi(m'_v) - \varphi(m_v))x_{f(v)}$, so z_v is also uniformly distributed over $\mathbb{Z}_p^*$. When $f(v) \in S^{\mathrm{w}}_{\mathbf{x}}$, $v \notin S^{\mathsf{source}}$ and all the commitments for the predecessor nodes of node v are unchanged, z_v is also unchanged and is randomly picked over $\mathbb{Z}_p^*$ by $\mathcal{P}_1$. Therefore, our claim holds.

Then, for every $v \in S^{\mathsf{source}}$, given z_v and c, a''_v is uniquely determined by $a''_v = g^{z_v}/y^c_{f(v)}$. By recursion, for every $v \in V \backslash S^{\mathsf{source}}$, given z_v, b''_v or e''_v is uniquely determined by b''_v or $e''_v = g^{z_v}/y^{\varphi(m''_v)}_{f(v)}$.

In the simulation, c and z_v ($v \in V$) are randomly picked over $\mathbb{Z}_p^*$. On the other hand, $a''_v = g^{z_v}/y^c_{f(v)}$ for $v \in S^{\mathsf{source}}$ and b''_v or $e''_v = g^{z_v}/y^{\varphi(m''_v)}_{f(v)}$ for $v \in V \backslash S^{\mathsf{source}}$.

Thus, the output distributions of $\Sigma^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}_{\mathrm{DAG}}$ and the simulator are identical. Therefore, we have an efficient simulator, which implies that the Sigma protocol $\Sigma^{\mathcal{R}^{\mathrm{dl}}_{k\text{-CNF},S'_k}}_{\mathrm{DAG}}$ is special HVZK.

H Some 3D figures for the experiments

Here, we present some 3D figures for the experiments. Fig. 24 is the communication cost of [14] and Fig. 25 is the communication cost of our scheme. Fig. 26 is the running time of $\mathsf{kCNFtoDAG}$. Fig. 27 contains the experimental results of running time of each step of [14] and our scheme. The above figures are the running time of $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{V}_2$ of [14], and the below figures are of our scheme.

Communication costs. We set a reasonable range for each parameter, i.e., n varies from 10 to 50 and k varies from 4 to $n/3$ (if $4 > n/3$, then we just set $k = 4$). Besides, the total number of clauses would be $C^k_n - \chi$, where χ is a random number in $[50, 200]$. From the figures, it is clear that our solution has a remarkable decrease on the communication costs compared with [14]. In

addition, the figures show that the effect on decrease would be better as n and k get larger.

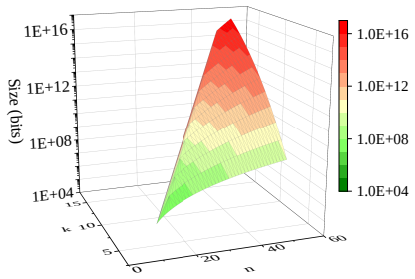


Fig. 24: Communication cost of [14]

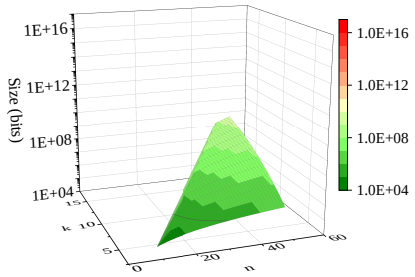


Fig. 25: Communication cost of our scheme

Running time. We show the running time of $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{V}_2$ separately. Note that when testing our solution, we also record the running time of kCNFtoDAG for special interest and do not count it in the time of $\mathcal{P}_1$ or $\mathcal{V}_2$, in order to compare our scheme with [14] more fairly, as kCNFtoDAG can be pre-computed and is not adopted in [14].

We evaluate both schemes with the same range of n and k. However, the running time of [14] grows so fast that the program was killed when n and k are set relatively large numbers. Therefore, in the experiment of [14], we set n from 10 to 33 and k from 4 to 7. In the experiment of our scheme, n varies from 10 to 50 and k varies from 4 to 10. The total number of clauses also is $C_n^k - \chi$, where χ is a random number in $[50, 200]$. Fig. 26 and Fig. 27 display the results.

Fig. 26 is the running time of kCNFtoDAG. As k and n get larger, the running time increases very quickly, since the number of vertices grows fast. But

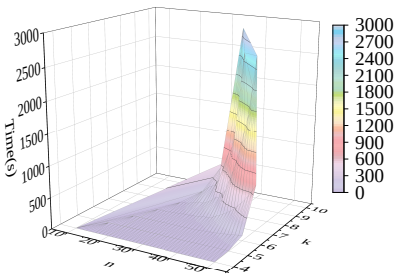


Fig. 26: Running time of kCNFtoDAG

in fact, if we compare it with the running time of $\mathcal{P}_1$ and $\mathcal{V}_2$ of our scheme, kCNFtoDAG performs reasonably well. Fig. 27 presents the running time of the main algorithms of both Sigma protocols. Fig. 27 points that the running time of $\mathcal{P}_2$ only occupies a limited percentage of the total running time. Thus, it shows that the running time of $\mathcal{P}_1$ and $\mathcal{V}_2$ dominates the running performance. From the perspective of the whole, our scheme saves lots of time compared with [14].

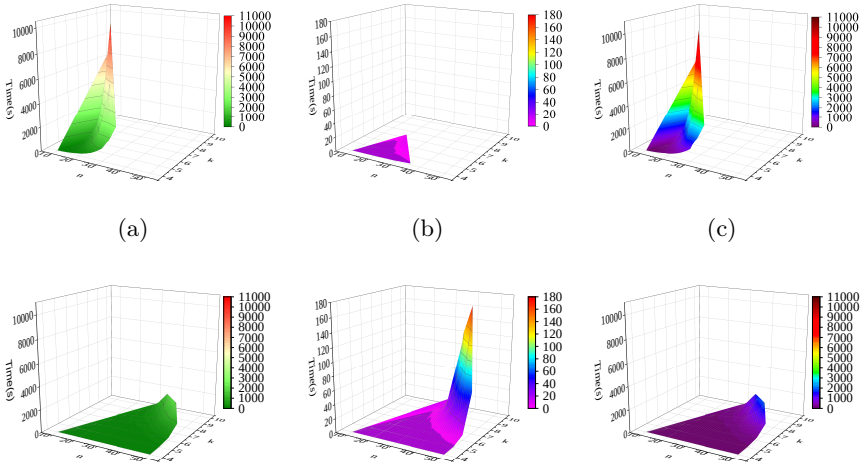


Fig. 27: The above figures are the running time of $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{V}_2$ of [14] separately and the below figures are of our scheme