Acyclicity Programming for Sigma-Protocols

Masayuki Abe[1], Miguel Ambrona[1], Andrej Bogdanov[2(✉)], Miyako Ohkubo[3], and Alon Rosen[4]

[1] NTT Laboratories, Tokyo, Japan
{abe.masayuki.cp,miguel.ambrona.fu}@hco.ntt.co.jp
[2] Chinese University of Hong Kong, Hong Kong, China
andrejb@cse.cuhk.edu.hk
[3] Security Fundamentals Laboratory, CSR, NICT, Tokyo, Japan
m.ohkubo@nict.go.jp
[4] Bocconi University, Milan, Italy and IDC Herzliya, Herzliya, Israel
alon.rosen@idc.ac.il

Abstract. Cramer, Damgård, and Schoenmakers (CDS) built a proof system to demonstrate the possession of subsets of witnesses for a given collection of statements that belong to a prescribed access structure $\mathcal{P}$ by composing so-called sigma-protocols for each atomic statement. Their verifier complexity is linear in the size of the monotone span program representation of $\mathcal{P}$.

We propose an alternative method for combining sigma-protocols into a single non-interactive system for a compound statement in the random oracle model. In contrast to CDS, our verifier complexity is linear in the size of the *acyclicity program* representation of $\mathcal{P}$, a complete model of monotone computation introduced in this work. We show that the acyclicity program size of a predicate is polynomially equivalent to the branching-program size of its monotone dual and hence polynomially incomparable to its monotone span program size. We additionally present an extension of our proof system, with verifier complexity linear in the *monotone circuit size* of $\mathcal{P}$, in the common reference string model.

Finally, considering the types of statement that naturally reduce to acyclicity programming, we discuss several applications of our new methods to protecting privacy in cryptocurrency and social networks.

Keywords: sigma-protocols · Zero-knowledge proofs · Random oracles

1 Introduction

The feasibility of proving every NP-statement in zero-knowledge was first established by Goldreich, Micali, and Wigderson [GMW86]. For decades, such generic protocols have been perceived as mere proof-of-concept results, in part because they involve costly NP reductions and also require multiple independent repetitions for the sake of soundness error reduction.

K. Nissim and B. Waters (Eds.): TCC 2021, LNCS 13042, pp. 435–465, 2021.
https://doi.org/10.1007/978-3-030-90459-3_15

More recently, we have witnessed a widespread push towards the efficient realization of general-purpose zero-knowledge proofs, with varying degrees of practicality [Gro16, AHIV17, BCG+17, GKM+18, BBB+18, BCR+19, CFQ19, MBKM19, LMR19]. Despite significant progress, the efforts are still ongoing, with performance a far cry from traditional cryptographic constructs such as public-key cryptography.

In contrast, special purpose protocols for specific hard-on-average languages within NP have seen wide practical deployment, most notably in the form of standardized digital signatures and anonymous credentials. An abstraction that is at the heart of this approach is so-called *sigma-protocols* [Cra97]. A sigma-protocol is a three-move interactive protocol between a prover and a verifier where the prover sends an initial message, a, to the verifier who replies with a random challenge, e, and the prover then responds with an answer, z, based on which the verifier accepts or rejects.

Sigma-protocols often enjoy low soundness error by design, resulting in high efficiency relative to their generic counterparts. Their zero-knowledge property is typically only guaranteed against honest verifiers, but they can be made secure against malicious verifiers using the Fiat-Shamir (FS) heuristic [FS87]. The FS heuristic transforms the protocol into a non-interactive argument by generating the challenge e via an application of a hash function to the initial message a.

1.1 Sigma-Protocol Composition

Importantly, sigma-protocols can be used to prove compound NP relations, by composing several sigma-protocols for "atomic" statements. Cramer, Damgård, and Schoenmakers (CDS) showed that sigma-protocols can be generically composed [CDS94], in the sense that given a collection of sigma-protocols for NP statements $x_1, \ldots, x_n$, one can obtain a zero-knowledge proof of knowledge for any subset of statements that belong to a prescribed access structure $\mathcal{P}$.

The idea in CDS composition is to secret-share the challenge e according to the access structure $\mathcal{P}$ and then use the shares as challenges in the corresponding sigma-protocols for each of the atomic statements. It works for any $\mathcal{P}$ recognized by a monotone span program [CDM00]. It results in sigma-protocols that can be made non-interactive, using the FS heuristic, and proved secure in the programmable random oracle model [BR93]. Due to its simplicity and generality, the CDS paradigm is popular both in theory and in practice.

Another composition technique, which is more closely related to this work, was introduced in [AOS02] and recently revisited in [FHJ20]. The idea, inspired by the ring signatures of [RST01], is to sequentially generate a challenge e_i to a statement x_i by hashing the first message a_{i-1} from the proof for the statement x_{i-1}. This sequence ends when the first challenge e_1 is generated from a_n. The method is referred to as a *sequential composition* in [FHJ20], who show that it can be turned into a signature scheme whose unforgeability can be proven in the non-programmable random oracle model (NPROM) [Nie02] (see also [Lin15, CPSV16]). In contrast, applying CDS+FS results in a signature scheme that is not provable in the NPROM in a black-box manner.

1.2 Our Contributions

We propose a new method for combining sigma-protocols for statements $x_1, \ldots, x_n$ into a non-interactive zero-knowledge proof of knowledge of witnesses w_i for subsets of the statements that belong to a prescribed access structure $\mathcal{P}$ (Sect. 3). The complexity of the proof system is dictated by the *acyclicity program* (ACP) size of $\mathcal{P}$, a new model of monotone computation introduced in this work.

In contrast, CDS composition complexity depends on the monotone span program size of $\mathcal{P}$. Relying on monotone complexity theory results, we show the existence of function families whose monotone span program size is inherently superpolynomial in their acyclicity program size (Sect. 4). Acyclicity programs can be polynomially simulated by monotone circuits. As there exist functions whose minimum monotone circuit size is superpolynomial in their span program size [BGW99], there is also a superpolynomial gap in the other direction. Thus the two composition methods are of incomparable complexity.

We prove the knowledge soundness of our acyclicity programming proof system in the programmable random oracle model (Theorem 1). We also obtain regular soundness in the non-programmable random oracle model (Theorem 2). Unlike CDS, our approach makes inherent use of the random oracle. We do not know how to implement such a transformation, even interactively, without recourse to the FS heuristic.

In Sect. 5 we describe a variant of the proof system whose complexity is determined by the monotone circuit size of $\mathcal{P}$. Monotone circuit size is at most polynomial in acyclicity program size but maybe superpolynomially smaller. The knowledge soundness of this protocol requires that the prover sample pairs of instances, out of which he can certify knowledge of exactly one. This is possible assuming the hardness of discrete logarithms.

Composition Via Acyclicity Programming. Our starting point is the sequential composition method of [AOS02,FHJ20]. They represent a disjunction of statements $x_1 \vee \cdots \vee x_n$ by a cycle on the nodes $1, \ldots, n$ with the interpretation that the cycle is "satisfied" if the cycle can be traversed starting at any node labeled by a true statement with a known witness. An acyclicity program extends these semantics to general directed graphs that may contain multiple and overlapping cycles. The nodes of an acyclicity program are labeled by statements, allowing repetitions. A satisfying set of witnesses corresponds to a set of nodes that cover every cycle.

Composition for Monotone Formulas. In Sect. 4.2 we show how to convert any monotone formula describing an access structure into an acyclicity program of the same size. This reduction from monotone formulas enjoys concrete efficiency and thus may be of practical interest. For relatively simple representations such as those in disjunctive normal form over distinct instances, compound proofs using our ACP method tend to be shorter than those by CDS+FS as challenges are not necessarily included in the proofs.

Composition for Monotone Circuits. In Sect. 5 we address a more general composition, represented by monotone circuits consisting of AND and THRESHOLD gates. This method compiles sigma-protocols into non-interactive systems with a common reference string (CRS), assuming the added property that the CRS allows oblivious sampling of instances from a hard language having a sigma-protocol. We note that oblivious sampling is often possible without CRS in the random oracle model, e.g. [GPS08], which is used in our basic ACP composition.

Unlike their non-monotone counterparts, monotone circuits are not universal for efficient computation of monotone functions [Raz85, AB87, Tar88]. Non-monotone circuits on n inputs, however, can be turned into monotone circuits on their $2n$ literals (variables and their negations) while at most doubling their size. Therefore, assuming the additional availability of sigma-protocols for proving *ignorance* [BM90, DK18] of the base statements, our composition method is essentially universal. Succinct proofs of ignorance, for discrete logarithms under a CRS, are used in our circuit protocol construction. (The CRS we consider is simply a group element with an unknown discrete logarithm.)

Other Advantages Over Related Works. CDS composition essentially relies on the fact that the composed sigma-protocols are 2-special sound, i.e., two colliding transcripts allow to extract a witness with probability one efficiently. Our construction can be used to compose more general κ-special sound protocols since the challenges to every run of the underlying protocol can be directly reassigned in the programmable random oracle model, and we can rewind the execution until κ valid transcripts are obtained. This allows the composition of a broader class of sigma-protocols, including, for instance, Stern's protocol [Ste96], which is 3-special sound and is used in lattice/code-based constructions [LLNW16, LLNW17, NTWZ19, FLWL20]. Other examples are protocols for proofs of binary secrets [BCC+15] which are also 3-special sound and useful for proving the correctness of inputs to a circuit.

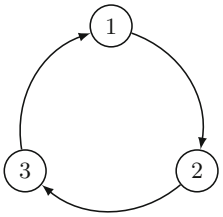
A recent work [AAB+20] enhances the CDS composition paradigm in the random oracle model, yielding shorter proofs and weakening the soundness requirements of the constituent sigma-protocols. Like CDS itself, their complexity is governed by the access structure's span program size and is therefore of incomparable complexity to the results in this work.

1.3 Technical Overview

Let $(\mathcal{C}_1, \mathcal{Z}_1, \mathcal{V}_1)$ and $(\mathcal{C}_2, \mathcal{Z}_2, \mathcal{V}_2)$ be sigma-protocols for relations R_1 and R_2 respectively. In the following, we show how to build a non-interactive zero-knowledge argument system (in the random oracle model) for the disjunctive combination of the above relations, i.e., for the language:

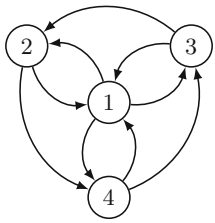
$$L_{R_1} \vee L_{R_2} \coloneqq \{(x_1, x_2) \,|\, \exists w : (x_1, w) \in R_1 \vee (x_2, w) \in R_2\} \ .$$

Let H be a random oracle that maps arbitrarily large inputs into $\{0,1\}^\lambda$ (w.l.o.g., the challenge space of both sigma-protocols). A proof on instance



$e_1 = H(a_3)$
$e_2 = H(a_1)$
$e_3 = H(a_2)$

(a) Sequential composition for the disjunctive relation $(1 \vee 2 \vee 3)$.



$e_1 = H(a_2 \| a_3 \| a_4)$
$e_2 = H(a_1 \| a_3)$
$e_3 = H(a_1 \| a_4)$
$e_4 = H(a_1 \| a_2)$

(b) Composition for a relation in CNF $(1 \vee 2) \wedge (1 \vee 3) \wedge (1 \vee 4) \wedge (2 \vee 3 \vee 4)$.

Fig. 1. Graphs inducing a compound ZK proof system.

(x_1, x_2) will consist of two accepting transcripts (a_1, e_1, z_1) and (a_2, e_2, z_2) for the respective sigma-protocols, where the challenges are computed as $e_1 = H(a_2)$, $e_2 = H(a_1)$, where $e_i = H(a)$ is an abbreviation of $e_i = H(x_1 \| x_2 \| i \| a)$.[1] Note that an honest prover in possession of a valid witness, say w_1 such that $(x_1, w_1) \in R_1$, can produce both transcripts by selecting $a_1 \leftarrow \mathcal{C}_1(x_1, w_1; r_1)$, computing $e_2 = H(a_1)$, running the simulator of the second sigma-protocol, obtaining $(a_2, z_2) \leftarrow \mathcal{S}_2(x_2, e_2)$ and using w_1 to complete the transcript for the first protocol on challenge $e_1 = H(a_2)$, producing an accepting z_1. The soundness of the protocol can be proven based on the soundness of the underlying sigma-protocols in the random oracle model, since the structure of our scheme and the fact that H is a random oracle guarantee that either e_1 will be uniformly chosen after having fixed a_1 or that e_2 will be uniformly chosen after having fixed a_2. Finally, zero-knowledge can be argued by defining a simulator with the ability to program the oracle H.

Observe that this technique can be generalized to combine in disjunction more than two statements, e.g., for combining three sigma-protocols we can define challenges as $e_1 = H(a_3)$, $e_2 = H(a_1)$ and $e_3 = H(a_2)$; but it can also be used to prove more complex compound statements. For example, given a monotone formula f in conjunctive normal form (CNF), consider a directed graph (where nodes are labelled as statements) that satisfies the following property: for every disjunctive clause in f there exists a cycle in the graph consisting of

[1] To highlight the role of a_1 and a_2, we use this shorthand in this section. We however remind readers the importance of including instances x_1 and x_2 as pointed out in [BPW12, BDG20].

nodes associated to all the statements in the clause, and there are not other cycles that correspond to a clause not implied by the original formula. We can build a proof system for f where proofs consist of an accepting sigma-protocol transcript for every node in the graph, and where for every node i, the challenge e_i is computed based on all nodes $j_1, \ldots, j_k$ that have an edge pointing to i, that is, $e_i = H(a_{j_1} \| \ldots \| a_{j_k})$. See Fig. 1 for two examples. Graph 1a corresponds to the already mentioned disjunctive predicate of three statements. On the other hand, graph 1b corresponds to the formula $(1 \vee 2) \wedge (1 \vee 3) \wedge (1 \vee 4) \wedge (2 \vee 3 \vee 4)$; there exist a cycle in the graph between all nodes that appear in the same clause.

It turns out that every graph induces a proof system of a certain monotone compound predicate over the base statements. We refer to Sect. 3 for more details about how to characterize the predicate associated with a graph.

1.4 Potential Applications

An immediate application of our novel composition method is a ring signature scheme [RST01] where an ACP access structure can describe the admissible group of signers. It extends the result of [BSS02] where admissible signers are described by a monotone formula, since ACP can simulate monotone formulas (Sect. 4.2). Unforgeability of the signature scheme could be proven in the programmable random oracle model following standard techniques.

As we have already mentioned, the complexity of our new model of computation is incomparable to the complexity of monotone span programs. There exist specific predicates that support superpolynomially more compact ACP representations. We describe some scenarios where the structure of ACPs could yield substantial improvements in communication and computation.

Our ACP proof system is especially suitable for predicates related to *disconnecting a graph*. One can prove knowledge of witnesses associated to certain nodes in an *acyclic* graph that represent a (node) *cut* from a node s to a node t (by artificially adding an edge between t and s and arguing the absence of cycles in the new graph). Such a proof would hide which subset of nodes has been used among all *cuts* that disconnect s from t. Even when the original graph is not acyclic, in Proposition 2 we show that the separation of s from t can be enforced with ACP at a cubic cost in the size of the graph, thereby retaining superpolynomial advantage over monotone span programs.

While the focus of our work is theoretical, we envision potential applications relating to certifying complex statements about social or financial activities. For example, consider a cryptocurrency system with a *transaction graph* whose nodes represent public keys, and whose (directed) edges represent money flows. Our proof system could apply to the following types of claims:

- *Proof of possession of white money.* Users may want to certify that the money associated with a given node (logically, owned by them) has been transferred (possibly not directly) from a set of whitelisted organizations (such as banks or well-reputed companies). To preserve the organizations' pseudonymity (not disclosing the nodes controlled by them) the organizations can certify the

user's integrity by proving knowledge of valid credentials associated with nodes that form a cut from the cryptocurrency genesis node to the node of interest, without revealing any information about the actual cut.
- *Proof of self-transaction.* A common practice in pseudonymous networks to increase anonymity is to transfer assets to freshly generated identities. Repeating such self-transactions makes the graph complex, and it is believed to be good for privacy. The mechanism described above would allow a user to certify a statement of the form *"this node belongs to me, and all of its money was once owned by myself or by the subset of parties who are colluding with me to create this proof"*, while retaining privacy on the origin nodes.

Besides scenarios where the graph represents transactions, we expect that our proof system may find more applications in other settings like social networks. Other possible applications based on the expressivity of our new approach, i.e., arguing knowledge of witnesses that make a graph acyclic, are:

- *Resolving circular software dependencies.* A user may be willing to pay money to a company for resolving a circular dependency issue in their updated software. In the paradigm of *fair exchange*, the user wants to pay after the problem is solved, but the company wants the money upfront. They can leverage the approach devised by Maxwell based on contingent payment [Max11]. This approach requires the company first proving in zero-knowledge that they can resolve the circular dependency, a statement that could be easily expressed with an ACP program.
- *st-unreachability in ℓ or fewer steps.* As a generalization of the *directed st-cut* predicate, our method can be used to prove unreachability between two nodes in a directed graph by considering paths of limited length ℓ. This could be useful on applications where the graph represents a map or, as above, in graphs of transactions where the transactions' lifetime matters. This application would require reducing the original graph G with n vertices into a graph H with $n\ell$ vertices so that G has a path from s to t of length at most ℓ if and only if H has a cycle. Such reduction could be devised as in the proof of Proposition 2 if distances were tracked as a separate parameter.

2 Preliminaries

For a finite set S, we write $a \leftarrow S$ to denote that a is uniformly sampled from S. We denote the security parameter by $\lambda \in \mathbb{N}$. Given two functions $f, g : \mathbb{N} \to [0, 1]$, we write $f \approx g$ if the difference $|f(\lambda) - g(\lambda)|$ is asymptotically smaller than the inverse of any polynomial. A function f is said to be *negligible* if $f \approx 0$, whereas it is said to be *overwhelming* when $f \approx 1$. For integers m, n, such that $m \leq n$, we denote by $[m, n]$ the range $\{m, m+1, \ldots, n\}$. We denote by $[n]$ the range $[1, n]$. By $\mathbb{N}^*$ we denote the space of arbitrarily-long sequences of numbers in $\mathbb{N}$. We use multiplicative notation for groups. When $\mathcal{A}$ is a probabilistic algorithm, we denote by $\mathcal{A}(x; r)$ an execution of $\mathcal{A}$ on input x and random coins r taken from an appropriate domain defined for $\mathcal{A}$. If the random coins are not important, we

simply write $\mathcal{A}(x)$. We generally assume stateful adversaries, e.g., in the game execution $a \leftarrow \mathcal{A}(x)$; $e \leftarrow \{0,1\}^\lambda$; $z \leftarrow \mathcal{A}(e)$ the adversary $\mathcal{A}$ in the second call knows the state of $\mathcal{A}$ after the first call (in particular, it knows x and a). Let $R : \mathcal{X} \times \mathcal{W} \to \{0,1\}$ be a binary relation defined over a set of instances $\mathcal{X}$ and a set of witnesses $\mathcal{W}$. We denote by L_R be the language defined as $L_R \coloneqq \{x \in \mathcal{X} \mid \exists w \in \mathcal{W} : R(x,w) = 1\}$.

2.1 Sigma-protocols

A sigma-protocol, introduced in [Cra97], is a public-coin interactive proof system that consists of only three data transfers between the prover and the verifier, and it satisfies a weaker notion of zero-knowledge and special soundness. In this work we employ a generalized definition of special soundness as in [AAB+20], where $\kappa \geq 2$ colliding transcripts with distinct (uniformly sampled) challenge lead to efficient witness extraction with probability $1 - \epsilon$ for a negligible ϵ. Compared to the original special soundness, where $\kappa = 2$ and $\epsilon = 0$, it captures a broader class of schemes, e.g. the parallel version of Stern's protocol [Ste96], where an exponential number (but still negligible compared to the size of the challenge space) of colliding transcripts can be prepared without knowing the witness. (In plain Stern's protocol, the challenge is chosen from $\{0,1,2\}$ and it is always possible to answer two preliminary chosen challenges. In its parallel version, a prover with no witness could answer up to 2^λ challenges from $\{0,1,2\}^\lambda$.)

Definition 1 (Sigma-protocol). *A sigma-protocol for R is a triple of PPT algorithms $(\mathcal{C}, \mathcal{Z}, \mathcal{V})$ associated to the following execution:*

(i) $a \leftarrow \mathcal{C}(x, w\,; r)$*: given* $(x,w) \in \mathcal{X} \times \mathcal{W}$*, outputs an initial message.*
(ii) $e \leftarrow \{0,1\}^\lambda$*: the verifier's challenge is uniformly sampled.*
(iii) $z \leftarrow \mathcal{Z}(x, w, r, e)$*: the prover answers to the challenge.*
(iv) $1/0 \leftarrow \mathcal{V}(x, a, e, z)$*: the verifier returns* 1 *(acceptance) or* 0 *(rejection).*

A sigma-protocol must satisfy completeness, special soundness, and special HVZK:

Perfect Completeness. *For every $\lambda \geq 1$ and every pair $(x,w) \in R$,*

$$\Pr\left[e \leftarrow \{0,1\}^\lambda; \begin{array}{l} a \leftarrow \mathcal{C}(x, w\,; r) \\ z \leftarrow \mathcal{Z}(x, w, r, e) \end{array} : \mathcal{V}(x, a, e, z) = 1\right] = 1\,.$$

κ-Special Soundness. *There exists a predicate $\Phi : (\{0,1\}^\lambda)^\kappa \to \{0,1\}$ and there exists a deterministic polynomial-time extractor $\mathcal{E}$ such that:*

- *On input $(x, a, \{e_1, z_1\}, \ldots, \{e_\kappa, z_\kappa\})$ such that $\mathcal{V}(x, a, e_i, z_i) = 1$ for all $i \in [\kappa]$ and $\Phi(e_1, \ldots, e_\kappa) = 1$, $\mathcal{E}$ outputs a witness w s.t. $R(x,w) = 1$.*
- *$\Pr\left[e_1, \ldots, e_\kappa \leftarrow \{0,1\}^\lambda : \Phi(e_1, \ldots, e_\kappa) = 0\right]$ is negligible in λ. (This probability is referred to as the special-soundness error.)*

It is perfectly *κ-special sound if Φ only checks for the challenges being different.*

Special Honest Verifier Zero-Knowledge (HVZK). *There exists a PPT algorithm $\mathcal{S}$ that, for every stateful PPT distinguisher $\mathcal{D}$,*

$$\begin{aligned} & \Pr\left[(x,w,e) \leftarrow \mathcal{D}(1^\lambda);\ a \leftarrow \mathcal{C}(x,w;r);\ z \leftarrow \mathcal{Z}(x,w,r,e) : \mathcal{D}(a,z) = 1\right] \\ \approx\ & \Pr\left[(x,w,e) \leftarrow \mathcal{D}(1^\lambda);\ (a,z) \leftarrow \mathcal{S}(x,e) \qquad\qquad : \mathcal{D}(a,z) = 1\right] \end{aligned}$$

where $\mathcal{D}$ must output values such that $(x,w) \in R$ and $e \in \{0,1\}^\lambda$.

The following notion is implied by κ-special soundness, for any constant κ.

Definition 2 (Soundness). *A sigma-protocol for L_R is said to be* sound with error ϵ_{snd} *if, for any PPT stateful adversary $\mathcal{A}$ and for any $x \notin L_R$, it holds* $\Pr[a \leftarrow \mathcal{A}(x);\ e \leftarrow \{0,1\}^\lambda;\ z \leftarrow \mathcal{A}(e) : \mathcal{V}(x,a,e,z) = 1] < \epsilon_{\mathsf{snd}}$. *We say the protocol is* sound *if ϵ_{snd} is negligible in λ.*

If there existed an adversary $\mathcal{A}$ against the soundness game, winning with non-negligible probability, by the Forking Lemma (Lemma 1), there would exist another algorithm $\mathcal{B}$ producing κ accepting transcripts with the same first message with non-negligible probability (there would be a probability loss with κ in the exponent, but κ is constant). Because the challenges in those transcripts would be sampled uniformly and independently, $\Phi(e_1,\ldots,e_\kappa)$ would be 1 with overwhelming probability. On the other hand, $\mathcal{E}$ cannot extract a valid witness for x, since such a witness does not exist. This contradicts κ-special soundness.

For an example of our generalized version of special soundness, think of the parallel version of Stern's protocol, with challenge space $\{0,1,2\}^\lambda$. It achieves 3-special soundness under our definition, for condition $\Phi(e,e',e'')$ defined as $\exists i \in [\lambda]$ *such that* $\{e_i,e'_i,e''_i\} = \{0,1,2\}$, where the equality is between (unordered) sets. In this case, the special soundness error would be $(7/9)^\lambda$.

2.2 Non-interactive Arguments

We define non-interactive argument systems in the random oracle model.

Definition 3 (Non-Interactive Argument System). *A non-interactive argument system for relation R in the random oracle model is a pair of polynomial-time oracle algorithms* $(\mathsf{Prove},\mathsf{Verify})$ *that, for random oracle H:*

- $\mathsf{Prove}^H(x,w) \to \pi$ *is a probabilistic algorithm that takes an instance x and a witness w as input, and outputs a proof π.*
- $\mathsf{Verify}^H(x,\pi) \to 0/1$ *is a deterministic algorithm that takes x and π, and outputs either 1 or 0 representing acceptance or rejection, respectively.*

It is correct if, for every sufficiently large $\lambda \in \mathbb{N}$, all H, all $(x,w) \in R$, $\mathsf{Verify}^H(x,\mathsf{Prove}^H(x,w))$ outputs 1 except with negligible probability in λ.

Definition 4 (Zero-Knowledge). *A non-interactive argument system* $(\mathsf{Prove},\mathsf{Verify})$ *for R is zero-knowledge in the random oracle model if there exists a probabilistic polynomial-time algorithm* $\mathsf{Simulator}$ *that for every PPT distinguisher $\mathcal{D}$, the following difference is negligible in λ:*

$$\Pr[1 \leftarrow \mathcal{D}^{H,\mathsf{Prove}^H}(1^\lambda)] - \Pr[1 \leftarrow \mathcal{D}^{\mathcal{S}}(1^\lambda)]\ .$$

$\mathcal{D}$ makes two types of queries; i) proof queries *on $(x, w) \in R$, and ii)* hash queries *on any string. H is a random oracle that returns an independently and uniformly chosen value in an appropriate domain for every distinct input. $\mathcal{S}$ is a simulator oracle that, given a proof query, forwards x to* Simulator *and returns the output. It also forwards hash queries transparently to* Simulator.

Definition 5 (Witness Indistinguishability). *A non-interactive argument system* $(\mathsf{Prove}, \mathsf{Verify})$ *for R is witness indistinguishable if for every PPT adversary $\mathcal{A}$, any polynomial-size string h, any x, w_1, w_2 with $R(x, w_1) = R(x, w_2) = 1$, and any random oracle H, the following difference is negligible in λ:*

$$\Pr[1 \leftarrow \mathcal{A}(x, w_1, \mathsf{Prove}^H(x, w_1), h)] - \Pr[1 \leftarrow \mathcal{A}(x, w_1, \mathsf{Prove}^H(x, w_2), h)] \ .$$

Definition 6 (Soundness). *A non-interactive argument system* $(\mathsf{Prove}, \mathsf{Verify})$ *for L_R is sound if for any PPT oracle algorithm $\mathcal{A}$, any $x \notin L_R$, and a random oracle H,* $\Pr[\pi \leftarrow \mathcal{A}^H(x) \ : \ 1 = \mathsf{Verify}^H(x, \pi)]$ *is negligible in λ. The probability is taken over the coins of $\mathcal{A}$ and H.*

The following definition models the fact that provers who create a valid proof must know a witness for the statement being proven. This is enforced by the existence of an extractor (that can invoke the prover in an oracle manner) that extracts a witness whenever the prover produces a valid proof.

Definition 7 (Knowledge Soundness). *A non-interactive argument system* $(\mathsf{Prove}, \mathsf{Verify})$ *for R is knowledge sound with knowledge error ϵ in the random oracle model if there exists an expected polynomial-time algorithm $\mathcal{E}$ such that for any probabilistic polynomial-time (potentially cheating) prover P^*, it holds that:*

$$\Pr\left[\begin{array}{l}(x, \pi) \leftarrow P^{*H}(1^\lambda) \\ w \leftarrow \mathcal{E}^{H, P^*}(x, \pi)\end{array} : \mathsf{Verify}^H(x, \pi) = 1 \ \wedge \ R(x, w) = 0\right] \leq \epsilon(\lambda) \ ,$$

where the probability is taken over the coins of P^*, $\mathcal{E}$, and random oracle H. The system is said to be knowledge sound *if ϵ is negligible in λ.*

2.3 Graphs

A *directed graph* is a tuple $G = (V, E)$ where V is a set of elements called vertices and E is a set of ordered pairs, $E \subseteq V \times V$, called *directed edges* or *arrows*. Throughout the paper, we use N to denote the size of V. Given an edge $e = (u, v)$, u is called the *head* of e, denoted by $head(e)$ and v is called the *tail* of e, denoted by $tail(e)$. We usually assume an ordering on the vertices V and denote by v_ℓ (or simply ℓ when it is clear from the context) the ℓ-th vertex, for every $\ell \in [N]$. For $v \in V$, we denote by $pred(v)$ the set of *predecessors* of v, $pred(v) := \{u \in V \mid (u, v) \in E\}$. Given a subset of vertices $U \subset V$, we define the subgraph $G(U)$ as (U, E_U), where $E_U := \{(u, v) \in E \mid u \in U \wedge v \in U\}$. A *cycle* in G is a finite sequence of edges $(e_1, \ldots, e_\ell)$ satisfying $tail(e_i) = head(e_{i+1})$ for every $i \in [\ell - 1]$ and $tail(e_\ell) = head(e_1)$. A graph with no cycles is called *acyclic*.

3 ACP Composition

3.1 Construction

In this section we describe and analyze our ACP composition for zero-knowledge composition of sigma-protocols. The common input to the prover and verifier consists of n statements $x_1, \ldots, x_n$ and a monotone set system $\mathcal{P}$ over base set $[n]$ represented by a monotone acyclicity program, which we will define shortly. A prover wants to convince a verifier that he knows valid witnesses w_i, $\forall i \in S$ for some set S in $\mathcal{P}$. We assume the availability of special-sound, HVZK sigma-protocols $\Sigma_i = (\mathcal{C}_i, \mathcal{Z}_i, \mathcal{V}_i, \mathcal{S}_i)$ for verifying that w_i is a witness for x_i.

Inputs: $\boldsymbol{x} = x_1, \ldots, x_n$, proof π, and acyclicity program A (with N nodes).
Random oracle: H.

1. Parse π as $((a_1, e_1, z_1), \ldots, (a_N, e_N, z_N))$.
2. Verify that $e_j = H(\boldsymbol{x} \| j \| \{a_p \,|\, p \in \mathit{pred}_A(j)\})$ for all $j \in [N]$.
3. Accept iff $\mathcal{V}_i(x_i, a_j, e_j, z_j)$ accepts for all $j \in [N]$, where $i = \mathit{var}(j)$.

Fig. 2. The ACP verifier.

Definition 8 (Monotone acyclicity program). *A* monotone acyclicity program *A is a directed graph G whose nodes are labeled by the* variables $1, \ldots, n$, *allowing repetitions. We use $\mathit{var}(j) \in [n]$ to denote the label of node $j \in [N]$, and $\mathit{pred}(j)$ to denote the set of nodes pointing to j. A set $S \subseteq [n]$ is accepted by A if every (directed) cycle in G contains a node j such that $\mathit{var}(j) \in S$.*

Equivalently, a set $S \subseteq [n]$ is accepted by A if the subgraph $G(\bar{S})$ induced by the complement set of nodes $\{j \,|\, \mathit{var}(j) \notin S\}$ contains no cycles. The sets accepted by A form a monotone set system $\Pi(A)$. We will measure the size of A by the number of nodes N and, as a complexity parameter of secondary interest, the number of edges M. The number of nodes determines the communication complexity of the protocol.

We now describe the ACP verifier (Fig. 2). A proof π consists of N parts $(a_1, e_1, z_1), \ldots, (a_N, e_N, z_N)$, where (a_j, e_j, z_j) has the format of a transcript for $\Sigma_{\mathit{var}(j)}$. The verifier accepts if all transcripts are accepting. The key idea is to enforce a specific choice of challenges e_j that allows the prover to pass verification only when knowing a sufficiently large set of witnesses S for which the graph $G(\bar{S})$ is acyclic. To accomplish this, we require that e_j be the value of the random oracle H evaluated at all a_p for $p \in \mathit{pred}(j)$ (ordered canonically).

As an example, consider the acyclicity program in Fig. 1a, where the challenges e_j must satisfy the constraints $e_1 = H(\boldsymbol{x} \| 1 \| a_3)$, $e_2 = H(\boldsymbol{x} \| 2 \| a_1)$ and $e_3 = H(\boldsymbol{x} \| 3 \| a_2)$. A prover that knows a witness, say w_1, can pass verification by first choosing $a_1 \leftarrow \mathcal{C}_1(x_1, w_1 \,; r_1)$, then simulating (a_2, z_2) given e_2, then simulating (a_3, z_3) given e_3, and finally completing the proof of knowledge

$\mathcal{Z}_1(x_1, w_1, r_1, e_1)$ for w_1 upon challenge e_1. This strategy can be generalized to an arbitrary acyclicity program A, as long as the complement of the set of known witnesses is acyclic, or equivalently the set of known witnesses is accepting for A. The general prover is given in Fig. 3.

Inputs: $\boldsymbol{x} = x_1, \ldots, x_n$, witnesses $w_i : i \in S$, acyclicity program A (with N nodes).
Precondition: A accepts S.
Random oracle: H.

1. Compute $a_j \leftarrow \mathcal{C}_j(x_i, w_i; r_i)$ for all $j \in [N] : i = var(j) \in S$.
2. Compute a topological sort $j(1), \ldots, j(t_{\max})$ of the vertices in $G(\bar{S})$.
3. For t ranging from 1 to $t_{\max}$, do:
   (a) Set $e_{j(t)} \coloneqq H(\boldsymbol{x} \| j(t) \| \{a_p \mid p \in pred(j(t))\})$.
   (b) Compute $(a_{j(t)}, z_{j(t)}) \leftarrow \mathcal{S}_{j(t)}(x_i, e_{j(t)})$, where $i = var(j(t))$.
4. For all j such that $var(j) \in S$, do:
   (a) Set $e_j \coloneqq H(\boldsymbol{x} \| j \| \{a_p \mid p \in pred(j)\})$.
   (b) Compute $z_j \leftarrow \mathcal{Z}_j(x_i, w_i, r_j, e_j)$, for all $j \in [N] : i = var(j) \in S$.
5. Output $\pi \coloneqq ((a_1, e_1, z_1), \ldots, (a_N, e_N, z_N))$.

Fig. 3. The ACP prover.

The communication complexity of this protocol is the sum of the communication complexities of the sigma-protocols $\Sigma_{var(1)}, \ldots, \Sigma_{var(N)}$. In fact, the challenges $e_1, \ldots, e_N$ do not have to be sent explicitly as the ACP verifier can turn step 2 from a verification into a calculation. The running time of the verifier is $O(Nt_{\mathcal{V}} + M)$ where $t_{\mathcal{V}}$ is the maximum running time of the verifiers $\mathcal{V}_i$, assuming H can be evaluated in linear time.

3.2 Security

The *composition* of witness relations R_i with the monotone set system $\mathcal{P}$ is the witness relation $R_{\mathcal{P}}$ in which $(w_i : i \in S)$ is a witness for $\boldsymbol{x} = (x_1, \ldots, x_n)$ if $S \in \mathcal{P}$ and w_i is an R_i-witness for x_i for all $i \in S$.

Theorem 1. *If $\Sigma_i = (\mathcal{C}_i, \mathcal{Z}_i, \mathcal{V}_i, \mathcal{S}_i)$ is a sigma-protocol for R_i for all $i \in [n]$ over the common challenge space $\{0,1\}^\lambda$, $\mathcal{P}$ is the set system computed by acyclicity program A, and H is a programmable random oracle, then ACP is a non-interactive zero-knowledge argument of knowledge for $R_{\mathcal{P}}$.*

We now prove completeness, zero-knowledge, and knowledge soundness of ACP.

Completeness. Assume that the prover knows witnesses $w_i : i \in S$ for some accepting set S of A. By definition, the graph $G(\bar{S})$ is acyclic and admits a topological sort.[2] Therefore all the inputs to H in step 3a are well-defined. By

[2] A topological sort is a linear ordering of the vertices of the graph satisfying that for every edge (u, v), u comes before v in the ordering.

Input: Acyclicity program A with graph G.
Oracle access to: P^*.

1. Set $S = \varnothing$.
2. While $G(\bar{S})$ has a directed cycle C, repeat the following:
3. Keep sampling random $\boldsymbol{h}^1$ until $P^*(\boldsymbol{h}^1) = (\boldsymbol{x}^1, \pi^1)$ is valid.
4. Let $j^* = last(\boldsymbol{h}^1; C, \boldsymbol{x}^1)$ and $i = var(j^*)$.
5. Remember $x_i = \boldsymbol{x}^1_{j^*}$ and $(a^1_{j^*}, e^1_{j^*}, z^1_{j^*}) = (j^*\text{th component of } \pi^1)$.
6. Set $r = 2$ and repeat the following until $r = \kappa$ but at most $4\kappa q/\delta$ times:
7. Sample $\boldsymbol{h}^r$ at random conditioned on $h^r_k = h^1_k$ for $1 \le k < det(\boldsymbol{h}^1; C, \boldsymbol{x}^1)$.
8. If $P^*(\boldsymbol{h}^r) = (\boldsymbol{x}^r, \pi^r)$ is valid and $det(\boldsymbol{h}^r; C, \boldsymbol{x}^r) = det(\boldsymbol{h}^1; C, \boldsymbol{x}^1)$,
9. Remember $(e^r_{j^*}, z^r_{j^*}) = (\text{last 2 terms in } j^*\text{th component of } \pi^r)$.
10. Increment r.
11. If $r = \kappa$ and $\mathcal{E}_i(x_i, a^1_{j^*}, \{e^1_{j^*}, z^1_{j^*}\}, \ldots, \{e^r_{j^*}, z^r_{j^*}\})$ outputs a valid witness w_i,
12. Remember x_i, w_i and add i to S.
13. Output the statement-witness pairs $(x_i, w_i : i \in S)$.

Fig. 4. The ACP extractor.

the HVZK property of Σ_i for $i = var(j) \notin S$, all the HVZK simulators $\mathcal{S}_i$ run in step 3b (on random challenges) yield accepting transcripts. By the completeness of Σ_i for $i = var(j) \in S$, all the transcripts computed in steps 1 and 4 are also accepting. It follows that the ACP prover passes verification with overwhelming probability. The completeness error of ACP is at most the sum of the larger one among the completeness and simulation errors of the input protocols Σ_i.

Zero-Knowledge. The zero-knowledge simulator runs independent copies of the simulators $\mathcal{S}_i(x_i, e_j)$, for all $j \in [N]$ where $i = var(j)$ and e_j is uniformly sampled, obtaining transcripts (a_j, e_j, z_j). It then outputs their concatenation as the proof π. The random oracle H is programmed so that e_j equals $H(\boldsymbol{x} \,\|\, j \,\|\, \{a_p \,|\, p \in pred(j)\})$. The produced simulated proof is distributed as a genuine proof owing to the indistinguishability of the output by $\mathcal{S}_i$ from regular transcripts of Σ_i. By a hybrid argument, the simulation error is at most the sum of the simulation errors of the input protocols Σ_i. Perfect zero-knowledge is achieved if every Σ_i is perfect special honest verifier zero-knowledge.

Knowledge Soundness. Let P^* be a potentially cheating prover that produces statement $\boldsymbol{x} = (x_1, \ldots, x_n)$ and a valid proof π with probability $\delta + \binom{q}{2}2^\lambda$, for some $0 < \delta \le 1$. P^* runs in time t, has query complexity q and is given oracle answers $\boldsymbol{h} = (h_1, \ldots, h_q) \in (\{0,1\}^\lambda)^q$ to its q queries. We will show that if all the Σ_i protocols are κ-special sound with knowledge extractor $\mathcal{E}_i$ with (negligible) error $\epsilon_i \le (\delta/2)^\kappa / 2q^{\kappa-1}$, then our ACP extractor, defined in Fig. 4, produces a valid witness for $R_{\mathcal{P}}$ with probability at least $\delta/2$, in expected time $O(\kappa q n t/\delta)$.

We will assume that (1) all prover queries to the random oracle H are distinct, and (2) if $(\boldsymbol{x}, \pi) \leftarrow P^*(\boldsymbol{h})$ for $\pi = ((a_i, e_i, z_i) : 1 \le i \le N)$ then P^* made all

queries of the form $(\boldsymbol{x}\|j\|\{a_p : p \in pred(j)\})$. Assumption (1) and (2) are without loss of generality as these constraints can be enforced while preserving prover efficiency. Assumption (2) ensures that all oracle queries of the ACP verifier must have also been made by P^*. We say that the output $(\boldsymbol{x}, \pi)$ of $P^*(\boldsymbol{h})$ is *valid* if the ACP verifier accepts $(\boldsymbol{x}, \pi)$ when interacting with any oracle that answers the k-th query of P^* by h_k, for all $k \in [q]$.[3]

We will also assume that (3) all oracle answers are distinct. Assumption (3) incurs a loss of $\binom{q}{2}2^{-\lambda}$ in the probability of $P^*(\boldsymbol{h})$ being valid (where $\boldsymbol{h}$ is now a sequence q uniformly sampled distinct values from $\{0,1\}^\lambda$), so from here on we will assume that success probability of P^* is at least δ.

The ACP extractor greedily collects witnesses from every cycle C of A. To do so, it seeks to find a query index Q such that the Q-th query is the one that determines the commitments and challenges along the cycle C. By forking at this query Q, which is prefixed by $\boldsymbol{x}\|j$ for some node j, a witness for $i = var(j)$ can be extracted. We define *"the query that determines commitments and challenges along C"* as follows. Assume $(\boldsymbol{x}, \pi) \leftarrow P^*(\boldsymbol{h})$ is a valid statement-proof pair, all challenges e_j are represented in the oracle answers $\boldsymbol{h}$, and all entries h_k in $\boldsymbol{h}$ are distinct (thus, every challenge e_j is uniquely represented as $h_{k(j)}$ in $\boldsymbol{h}$). Let $det(\boldsymbol{h}; C, \boldsymbol{x})$ be the smallest k_C such that $k(j) \leq k_C$ for all nodes j in cycle C, and $last(\boldsymbol{h}; C, \boldsymbol{x})$ be the node j_C so that the k_C-th query is prefixed by $\boldsymbol{x}\|j_C$. We argue that if $(\boldsymbol{x}, \pi) \leftarrow P^*(\boldsymbol{h})$ is valid then the first $k_C = det(\boldsymbol{h}; C, \boldsymbol{x})$ entries of $\boldsymbol{h}$ determine the commitment a_{j_C} in π in the following sense:

Claim 1. *Assume P^* is deterministic. If $\boldsymbol{h}$ and $\boldsymbol{h}'$ are two sequences, each of which has pairwise distinct entries, $(\boldsymbol{x}, \pi) := P^*(\boldsymbol{h})$ and $(\boldsymbol{x}', \pi') := P^*(\boldsymbol{h}')$ are both valid, $k_C = det(\boldsymbol{h}; C, \boldsymbol{x}) = det(\boldsymbol{h}'; C, \boldsymbol{x}')$, and $h_k = h'_k$ for all $1 \leq k < k_C$, then $\boldsymbol{x} = \boldsymbol{x}'$ and $a_{j_C} = a'_{j_C}$.*[4]

Proof. As $P^*(\boldsymbol{h})$ is valid, for every $j \in C$ there is a unique query step $k(j)$ such that the query $(\boldsymbol{x}\|j\|\{a_p : p \in pred(j)\})$ must have been made by $P^*(\boldsymbol{h})$ at step $k(j)$ and answered by $h_{k(j)} = e_j$. The same occurs analogously for $P^*(\boldsymbol{h}')$, denote such query step by $k'(j)$ for all $j \in C$, in this case.

Since the k-th answers h_k and h'_k are identical for all $1 \leq k < k_C$ and P^* is deterministic, the k-th queries of $P^*(\boldsymbol{h})$ and $P^*(\boldsymbol{h}')$ must be identical for all $1 \leq k \leq k_C$, and so $last(\boldsymbol{h}; C, \boldsymbol{x}) = last(\boldsymbol{h}'; C, \boldsymbol{x}') =: j_C$. By definition of det, we have $k(j), k'(j) < k_C$ for all $j \in C \setminus \{j_C\}$, so $k(j) = k'(j)$ and therefore $e_j = h_{k(j)} = h'_{k'(j)} = e'_j$ for all $j \in C \setminus \{j_C\}$.

In particular, $e_{j^+} = e'_{j^+}$ for the *successor* j^+ of j_C in C. Consequently, the corresponding query $k(j^+) = k'(j^+)$ made by P^* is then of the form $(\boldsymbol{x}\|j^+\|\{a_p \,|\, p \in pred(j^+)\}) = (\boldsymbol{x}'\|j^+\|\{a'_p \,|\, p \in pred(j^+)\})$. It follows that $\boldsymbol{x} = \boldsymbol{x}'$ and, as $j_C \in pred(j^+)$, also $a_{j_C} = a'_{j_C}$. □

[3] This is a slight abuse of terminology for the sake of readability as validity is not determined by $\boldsymbol{x}$ and π only, but may also depend on $\boldsymbol{h}$.

[4] For $\pi = ((a_1, e_1, z_1), \ldots, (a_N, e_N, z_N))$ and $\pi' = ((a'_1, e'_1, z'_1), \ldots, (a'_N, e'_N, z'_N))$.

Claim 2. *Assume $P^*(\boldsymbol{h})$ is valid with probability δ (over the internal coins of P^* and the sampling of $\boldsymbol{h}$: q uniformly chosen distinct values from $\{0,1\}^\lambda$) and the special soundness error of all Σ_i is at most $(\delta/2)^\kappa/4q^{\kappa-1}$. The ACP extractor outputs a witness for $R_{\mathcal{P}}$ in expected time $O(\kappa qnt/\delta)$ with probability at least $\delta/2$ over the randomness of P^*.*

Proof. The ACP extractor interacts in a black-box manner with a P^* whose internal randomness τ has been fixed (after being sampled from the appropriate distribution). We first argue that, with reasonable probability, fixing P^*'s internal randomness results in a "decent" prover algorithm. More concretely, let X be the random variable that maps τ to $\Pr[P^* \text{ succeeds} \mid \tau]$ we have that for every $\alpha \in [0,1]$, $\Pr[X \geq \alpha] \geq \delta - \alpha$, since

$$\begin{aligned}\delta &= \Pr[P^* \text{succeeds}] \\ &= \Pr[P^* \text{ succeeds} \mid X < \alpha] \Pr[X < \alpha] + \Pr[P^* \text{ succeeds} \mid X \geq \alpha] \Pr[X \geq \alpha] \\ &\leq \alpha + \Pr[X \geq \alpha] \quad .\end{aligned}$$

In particular, for $\alpha = \delta/2$, we have $\Pr[X \geq \delta/2] \geq \delta/2$. Consequently, with probability $\delta/2$, fixing P^*'s internal randomness will result in a deterministic prover with success probability (over the choice of $\boldsymbol{h}$) of at least $\delta/2$. From now own, we assume P^* has its randomness fixed.

Fix an arbitrary directed cycle C in G. Let Valid_r be the event that $P^*(\boldsymbol{h}^r)$ is valid and Match_r be the event $\mathit{det}(\boldsymbol{h}^r; C, \boldsymbol{x}^r) = \mathit{det}(\boldsymbol{h}^1; C, \boldsymbol{x}^1)$. By assumption, $\Pr[\mathit{Valid}_1] \geq \delta/2$. Applying the Forking Lemma 1 below (with $\kappa = 2$, $F(\boldsymbol{h}) = \mathit{det}(\boldsymbol{h}; C, \boldsymbol{x}^1)$, and $A =$ "$P^*(\boldsymbol{h})$ is valid"), we get that

$$\Pr[\mathit{Valid}_1 \cap \mathit{Valid}_r \cap \mathit{Match}_r] = \Pr[\boldsymbol{h}^1 \in A \wedge \boldsymbol{h}^r \in A \wedge F(\boldsymbol{h}^1) = F(\boldsymbol{h}^r)]$$

is lower bounded by $\Pr[\mathit{Valid}_1]^2/q$ and, consequently,

$$\Pr[\mathit{Valid}_r \cap \mathit{Match}_r \mid \mathit{Valid}_1] \geq \frac{\delta}{2q} \quad .$$

(By Claim 1, $\mathit{Valid}_r \cap \mathit{Valid}_1 \cap \mathit{Match}_r$ implies that $\boldsymbol{x}^r = \boldsymbol{x}^1$ and $a^r_{j_C} = a^1_{j_C}$.) The event that the ACP extractor reaches step 9 in the r-th iteration of loop 6 (given that it completed step 3) is precisely $\mathit{Valid}_r \cap \mathit{Match}_r$ (conditioned on Valid_1). If T_r is the number of executions of loop 6 with a given value of r, then $T_r \mid \mathit{Valid}_1$ is a geometric random variable with success parameter at least $\delta/2q$ so $\mathrm{E}[T_r \mid \mathit{Valid}_1] \leq q/\delta$ and, by linearity of expectation,

$$\mathrm{E}[T_1 + \cdots + T_{\kappa-1} \mid \mathit{Valid}_1] = \mathrm{E}[T_1 \mid \mathit{Valid}_1] + \cdots + \mathrm{E}[T_{\kappa-1} \mid \mathit{Valid}_1] \leq \frac{2\kappa q}{\delta} \quad .$$

Let *Reach* be the event that r reaches value κ in step 11. By Markov's inequality,

$$\Pr[\mathit{Reach}] = \Pr[T_1 + \cdots + T_{\kappa-1} \leq 4\kappa q/\delta \mid \mathit{Valid}_1] \geq \frac{1}{2} \quad .$$

We now upper bound the probability of the event *Fail* that $\mathcal{E}_i$ *fails to extract a witness* w_i for x_i *from these transcripts*, given *Reach*. The transcripts

$\boldsymbol{h}^1, \ldots, \boldsymbol{h}^r$ before step 11 of the ACP Extractor is executed are marginally random, identical in the first $det(\boldsymbol{h}^1; C, \boldsymbol{x}^1)$ entries, and independent in the other entries. The event *Reach* is the same as $AllValid \cap Fork$, where *AllValid* is the event "$P^*(\boldsymbol{h}^r)$ *is valid for all* r" and *Fork* is the event "$det(\boldsymbol{h}^r; C, \boldsymbol{x}^r) = det(\boldsymbol{h}^1; C, \boldsymbol{x}^1)$ *for all* r". Therefore:

$$\Pr[Fail \mid Reach] \leq \Pr[\neg A \mid Reach] \leq \frac{\Pr[\neg A \cap Fork]}{\Pr[AllValid \cap Fork]} \ ,$$

where A is the set of admissible transcripts for $\mathcal{E}_i$. Assuming *Fork*, by Claim 1, for $i = var(j_C)$, the i-th component x_i^r of the input $\boldsymbol{x}^r$ is equal to x_i^1 and the commitment $a_{j_C}^r$ is equal to $a_{j_C}^1$, for all r. The challenges $e_{j_C}^r$ are however independent of one another (and distributed uniformly) because the sequences $\boldsymbol{h}^r$ fork in position $k_C = det(\boldsymbol{h}^1; C, \boldsymbol{x}^1)$. So the probability of $\neg A$ and *Fork* happening at the same time is exactly the κ-special soundness error ϵ_i of $\mathcal{E}_i$: $\Pr[\neg A \cap Fork] = \epsilon_i$. On the other hand, by the Forking Lemma 1,

$$\Pr[AllValid \cap Fork] \geq \frac{(\delta/2)^\kappa}{q^{\kappa-1}} \ .$$

By our assumption on ϵ_i, we conclude that $\Pr[Fail \mid Reach] \leq 1/2$.

In conclusion, in any cycle C there exists a node j_C such that loop 2 succeeds in extracting a solution $(x_i, w_i) \in R_i$ for $i = var(j_C)$ with probability at least

$$\Pr[Reach \cap \neg Fail] = \Pr[Reach](1 - \Pr[Fail \mid Reach]) \geq \frac{1}{2} \cdot \frac{1}{2} \geq \frac{1}{4} \ .$$

It follows that the expected number of iterations to extract a solution to $R_{\mathcal{P}}$, namely a solution to $R_{var(j_C)}$ for some j_C in every cycle, is at most $4n$. As each loop 2 iteration has expected running time $O(\kappa q t/\delta)$, the total expected running time of the extractor is $O(\kappa q n t/\delta)$ as desired. □

By Claim 2, the ACP extractor succeeds with probability at least $1/2\delta$ over the combined randomness of the extractor and P^*. To extract a witness with probability $1 - \delta$ we run the ACP extractor independently $O((1/\delta)\log(1/\delta))$ times (with different fixed randomness for P^*), thereby establishing knowledge soundness of ACP.

It remains to establish the following version of the Forking Lemma, first introduced by Pointcheval and Stern [PS00] and later generalized by Bellare and Neven [BN06], here restated and proved in our notation.

Lemma 1 (Forking Lemma). *Let F be a possibly randomized function from q-term sequences to the set $\{1, \ldots, q\}$ and A be a subset of q-term sequences. The probability that $\boldsymbol{h}^1, \ldots, \boldsymbol{h}^\kappa$ are all in A and $F(\boldsymbol{h}^1) = \cdots = F(\boldsymbol{h}^\kappa)$, when $\boldsymbol{h}^1, \ldots, \boldsymbol{h}^\kappa$ are marginally uniform, mutually identical in the first $F(\boldsymbol{h}^1)$ entries, and mutually independent in the remaining ones, is at least $\delta(A)^\kappa/q^{\kappa-1}$, where $\delta(A)$ is the measure of A.*

Proof. Let $\boldsymbol{h} = (h_1, \ldots, h_q)$ be a uniformly random sequence.

$$\begin{aligned}
&\Pr\left[\boldsymbol{h}^1, \ldots, \boldsymbol{h}^\kappa \in A \text{ and } F(\boldsymbol{h}^1) = \cdots = F(\boldsymbol{h}^\kappa)\right] \\
&\quad = \textstyle\sum_{f=1}^{q} \Pr\left[\boldsymbol{h}^r \in A \text{ and } F(\boldsymbol{h}^r) = f \text{ for all } r \le \kappa\right] \\
&\quad = \textstyle\sum_{f=1}^{q} \mathrm{E}\left[\Pr[\boldsymbol{h} \in A \text{ and } F(\boldsymbol{h}) = f \mid h_1, \ldots, h_{f-1}]^\kappa\right] \\
&\quad \ge \textstyle\sum_{f=1}^{q} \Pr[\boldsymbol{h} \in A \text{ and } F(\boldsymbol{h}) = f]^\kappa, \\
&\quad \ge q \cdot \left(\textstyle\sum_{f=1}^{q} \frac{1}{q} \cdot \Pr[\boldsymbol{h} \in A \text{ and } F(\boldsymbol{h}) = f]\right)^\kappa = \frac{\Pr[\boldsymbol{h} \in A]^\kappa}{q^{\kappa-1}}.
\end{aligned}$$

The last two inequalities are both applications of Jensen's inequality. □

3.3 Security in the Non-programmable Random Oracle Model

Sigma-protocols with the Fiat-Shamir transform provide soundness when the hash function in the transformation is modeled as a non-programmable random oracle. Here we show that an analogue statement holds for ACP as well. Roughly, in the NPROM, the same random oracle is given to every algorithm including *the reduction* built in the security proof. Thus, the reduction does not have the advantage of being able to program the input-output correspondence of the random oracle. For a more formal definition we refer to [Nie02].

Theorem 2. *If $\Sigma_i = (\mathcal{C}_i, \mathcal{Z}_i, \mathcal{V}_i, \mathcal{S}_i)$ is a sigma-protocol for L_{R_i} for all $i \in [n]$ over the common challenge space $\{0,1\}^\lambda$, $\mathcal{P}$ is the set system computed by acyclicity program A, then ACP is a non-interactive witness indistinguishable argument for $L_{R_\mathcal{P}}$ in the non-programmable random oracle model.*

Proof. Witness indistinguishability is shown by a hybrid argument. Given two distinct sets of witnesses, $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ for $\boldsymbol{x}$, we build a hybrid starting from a proof obtained by running the ACP prover for input $(\boldsymbol{x}, \boldsymbol{w}_1, A)$. Adding a witness from $\boldsymbol{w}_2 \setminus \boldsymbol{w}_1$ one by one, it reaches to a proof for $(\boldsymbol{x}, \boldsymbol{w}_1 \cup \boldsymbol{w}_2, A)$ as input to the prover. Then, removing a witness belonging to $\boldsymbol{w}_1 \setminus \boldsymbol{w}_2$ one by one, it reaches to a proof for $(\boldsymbol{x}, \boldsymbol{w}_2, A)$. Any noticeable change of probability of a distinguisher outputting 1 at any point of the hybrid reduces to violating the special honest verifier zero-knowledge property of the corresponding Σ_i. Suppose that the gap happens between hybrids with respect to witness set $\boldsymbol{x}^*$ and $\boldsymbol{x}^* \cup \{w_{i^*}\}$. The reduction executes the prover algorithm with $\boldsymbol{x}^*$ using the zero-knowledge challenger that, given e_{i^*}, returns (a_{i^*}, z_{i^*}), instead of zero-knowledge simulator in step 3(b) in Fig. 3. Thus the proof distributes in the same way as prover with $\boldsymbol{x}^*$ if (a_{i^*}, z_{i^*}) is a simulated one while it is the same as prover with $\boldsymbol{x}^* \cup \{w_{i^*}\}$ if (a_{i^*}, z_{i^*}) is created using w_{i^*}. Accordingly, the witness indistinguishabilty error is at most $2N\epsilon_{\mathsf{zk}}$ where ϵ_{zk} is the largest zero-knowledge error among Σ_i.

For soundness, suppose that there is an adversary $\mathcal{A}$ that outputs, with probability $\epsilon_\mathcal{A}$, an instance $\boldsymbol{x} := (x_1, \ldots, x_n)$ together with a valid proof on $\boldsymbol{x}$, $\pi := ((a_1, e_1, z_1), \ldots, (a_N, e_N, z_N))$ that satisfies:

- for every qualified set S, there exists i that $x_i \notin L_{R_i}$,

- $e_j = H(\boldsymbol{x}\|j\|\{a_p \,|\, p \in pred(j)\})$ for all $j \in [N]$, and
- $\mathcal{V}_i(x_i, a_j, e_j, z_j)$ accepts for all $j \in [N]$, where $i = var(j)$.

Let $e_j \leftarrow H_j$ denote the event that $H(\boldsymbol{x}\|j\|\{a_p \,|\, p \in pred(j)\})$ is evaluated and e_j is returned. Let $H_k \leftarrow a_j$ denote the event that $\boldsymbol{x}\|k\|\{a_p \,|\, p \in pred(k)\}$ satisfying $a_j \in pred(k)$ is queried to H. Over the sequence of q distinct queries from $\mathcal{A}$ to the random oracle, we define function τ that, given an event E, outputs $\ell \in [q]$ such that the ℓ-th query to the random oracle made event E happened in the view of $\mathcal{A}$. (Define the output of τ as 0 if E did not occur at any query).

Suppose that there exist $j, k \in [N]$ that either $e_j \leftarrow H_j$ or $H_k \leftarrow a_j$ do not happen in the view of $\mathcal{A}$. Then the probability that the randomly assigned hash output (drawn in verification for the first time) satisfies the corresponding relation with respect to the already fixed $(x_{var(j)}, a_j, z_j)$, is bounded by the soundness error of Σ_j. Let ϵ_{snd} be the maximum soundness error, taking union bound for all q possible queries, the probability for this event is upper-bounded by $q \cdot \epsilon_{\mathsf{snd}}$.

Now, consider the case where for all $j, k \in [N]$, both events $e_j \leftarrow H_j$ and $H_k \leftarrow a_j$ happen in the view of $\mathcal{A}$. For every cycle C in graph G, let (j^*, k^*) be the arrow in C that satisfies $\tau(H_{k^*} \leftarrow a_{j^*}) = \min(\{\tau(H_k \leftarrow a_j) \,|\, (j,k) \in C\})$. Observe that $\tau(H_k \leftarrow a_{j^*}) = \tau(e_{j^*} \leftarrow H_{j^*})$ for some $k \neq k^*$ and therefore, $\tau(H_{k^*} \leftarrow a_{j^*}) < \tau(e_{j^*} \leftarrow H_{j^*})$. (Equivalently, a_{j^*} is fixed before e_{j^*} in the view of the adversary.) Let J be the set of such index j^* over all cycles in G. Then, by definition, $S := \{var(j) \,|\, j \in J\}$ is a qualified set. To see this, suppose that there exists $i \in S$ with $x_i \notin L_{R_i}$ and let j be such that $i = var(j)(\in J)$. For fixed x_i and a_j, the probability that, e_j is sampled uniformly at random (by H_j) and the adversary outputs z_j passing the verification is bounded by the soundness error ϵ_{snd} of Σ_i. Consequently, the probability that $\mathcal{A}$ produces a valid proof where there exists $i \in S$ that $x_i \notin L_{R_i}$ is upper-bound by $q \cdot \epsilon_{\mathsf{snd}}$.

Accumulating the above bounds, we have $\epsilon_{\mathcal{A}} < 2q \cdot \epsilon_{\mathsf{snd}}$, which concludes the proof of soundness in NPROM. □

4 The Expressive Power of Acylicity Programs

The communication and verification efficiencies of ACP composition grow linearly in the monotone acyclicity program size of the set system $\mathcal{P}$. To this end, in this section we study the complexity of acyclicity programs. In Sect. 4.1 we show that monotone acyclicity program size is polynomially equivalent to monotone branching program size [GS92].

As a corollary, leveraging known results from monotone complexity theory in Sect. 4.1 we conclude that there exist families whose monotone span program size grows superpolynomially in their monotone acyclicity program size. As the efficiency of the Cramer, Damgård, Schoenmakers sigma-protocol composition [CDS94] (CDS composition) is dictated by the former parameter, we obtain concrete examples of set systems for which ACP composition is asymptotically more efficient than CDS composition.

In Sect. 4.2 we demonstrate a simulation of de Morgan formulas by acyclicity programs of the same size. This is a consequence of known simulations of formulas by branching programs. In this section, it will be useful to view (monotone) set systems $\mathcal{P}$ over $[n]$ as (monotone) functions $\mathcal{P}\colon \{0,1\}^n \to \{0,1\}$ via the usual identification of a set by its indicator vector. Recall that the *monotone dual* $\mathcal{P}^\dagger$ of $\mathcal{P}$ is the function $\neg\mathcal{P}(\neg x_1, \ldots, \neg x_n)$. An acyclicity program for $\mathcal{P}$ can then be interpreted as a "cyclicity program" for $\mathcal{P}^\dagger$:

Definition 9 (Monotone cyclicity program). *A* monotone cyclicity program *is a directed graph whose nodes are labeled by variables $x_1, \ldots, x_n$. The program accepts a given input if the subgraph induced by the true-valued nodes has a cycle.*

The size of a cyclicity program is the number of nodes. We may, without affecting the size, allow for nodes labeled by the constant `true`. (These nodes can be bypassed by contracting all their incoming and outgoing edge pairs.)

4.1 Polynomial Equivalence with Branching Programs

A *monotone branching program* is also a directed acyclic graph with distinguished start and accept nodes whose vertices are labeled by variables or the constant 1. The program accepts a given input if the subgraph induced by the 1-valued vertices has a path from the start state to the accept state.

Let $mCP(\mathcal{P})$, $mBP(\mathcal{P})$ be the size of the smallest monotone cyclicity program, smallest monotone branching program for $\mathcal{P}$. The efficiency of the ACP verifier for the composed relation $R_{\mathcal{P}}$ in Sect. 1 is proportional to $mCP(\mathcal{P})$.

The following proposition shows that the complexity of our proof system is polynomially related to the monotone branching program size of the monotone dual $\mathcal{P}^\dagger$ of the composition predicate. As monotone branching programs can polynomially simulate monotone formulas [GS92], the complexity is also upper-bounded by the formula size $mF(\mathcal{P}^\dagger)$ of $\mathcal{P}^\dagger$. We elaborate on this simulation in the next section.

Proposition 1. *For every monotone $\mathcal{P}$, $mBP(\mathcal{P}) \le mCP(\mathcal{P}) \le mBP(\mathcal{P})^3 + 2$.*

On the other hand, the prover complexity in the NIZK composition framework of Cramer, Damgård, and Schoenmakers [CDS94] is proportional to the monotone span program complexity $mSP_{\mathbb{F}}(\mathcal{P}^\dagger)$ times some polylogarithmic factor in $\mathbb{F}$, over any finite field $\mathbb{F}$.[5] Let stC denote the directed st-connectivity family of functions, Pitassi and Robere [PR18] showed that:

$$mSP_{\mathbb{F}}(\mathsf{stC}) = mBP(\mathsf{stC})^{\Omega(mBP(\mathsf{stC}))} \ ,$$

for every $\mathbb{F}$. The monotone dual $\mathsf{stC}^\dagger$ is a family of predicates on n inputs for which ACP composition has complexity linear in n, but CDS composition

[5] Both mF and mSP [KW93] are in fact invariant under taking monotone duals, but mBP is not [GS92].

requires $n^{\Omega(\log n)}$ communication for any potential implementation of $\mathsf{stC}^\dagger$ by a monotone span program (over any finite field). In the other direction, it is known that:

$$mSP_{\mathbb{F}}(\mathcal{P}) \leq mF(\mathcal{P}) \leq mBP(\mathcal{P})^{O(mBP(\mathcal{P}))} \ ,$$

for every $\mathcal{P}$ [GS92] so the quasipolynomial separation is optimal.

Proof of Proposition 1. In the course of proving Proposition 1 we establish the equivalence of several natural directed graph problems with respect to suitable reductions. A *labeled graph* is a directed graph whose nodes are labeled by variables $x_1, \ldots, x_n$ or the constant 1. A *projection reduction* from graph property P to graph property Q is an algorithm that converts a labeled graph G into another labeled graph H with the same set of labels such the for any assignment to the variables, the subgraph induced by the 1-labels of G has property P if and only if the subgraph induced by the 1-labels of H has property Q. We consider the following graph properties:

- CYCLICITY: the graph has a (directed) cycle.
- *st*-CONNECTIVITY: the graph has a (directed) path from vertex s to vertex t.
- ACYCLIC *st*-CONNECTIVITY: *st*-connectivity under the promise that the graph is acyclic.

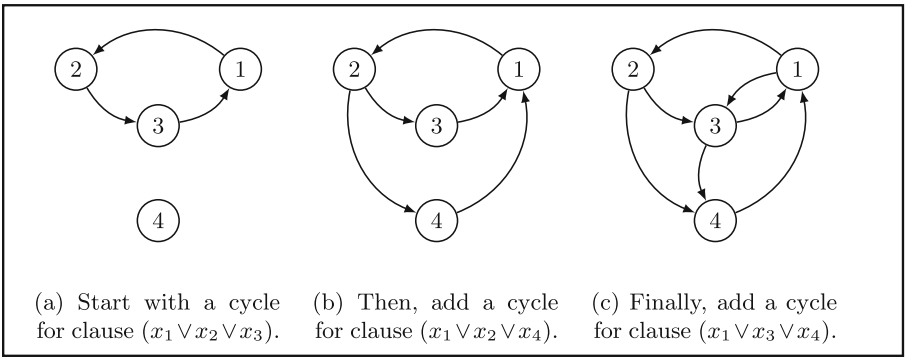
Proposition 2. CYCLICITY*,* *st*-CONNECTIVITY*, and* ACYCLIC *st*-CONNECTIVITY *are polynomial-time equivalent with respect to projection reductions.*

Proof. ACYCLIC *st*-CONNECTIVITY reduces to CYCLICITY by adding a back edge from t to s and trivially reduces to *st*-CONNECTIVITY.

In the other direction, we view CYCLICITY and *st*-CONNECTIVITY as special cases of the following property PATHS: Given a set of triplets (s_i, t_i, ℓ_i) does there exist a path from s_i to t_i of length ℓ_i for some i? In CYCLICITY $s_i = t_i$ ranges over all vertices and ℓ_i ranges over all values from 1 to n. In *st*-CONNECTIVITY $s_i = s, t_i = t$, and ℓ_i ranges over all values from 0 to $n-1$.

The reduction from PATHS to ACYCLIC *st*-CONNECTIVITY consists of building the reachability graph for the natural guess-and-verify nondeterministic logspace algorithm for PATHS. Let V be the vertex set of the PATHS instance G. For every pair $(v, w) \in V \times V$ and ℓ between 0 and n create a node (v, w, ℓ) in the ACYCLIC *st*-CONNECTIVITY instance H with the same label as w and two special nodes s and t with label 1. Also, for every edge (w, w') in G, every $v \in V \backslash \{w'\}$ and all $\ell \geq 1$, create an edge between node $(v, w, \ell-1)$ node (v, w', ℓ) in H. Finally, connect s to $(s_i, s_i, 0)$ and connect (s_i, t_i, ℓ_i) to t for all i in H. By construction, H is acyclic and its st-paths are all of the form $s, (s_i, v_0, 0), (s_i, v_1, 1), \ldots, (s_i, v_{\ell_i}, \ell_i), t$, where $v_0, \ldots, v_{\ell_i}$ is a path from s_i to t_i in G. □

As the reduction from ACYCLIC *st*-CONNECTIVITY to CYCLICITY preserves the number of vertices and the reverse reduction maps an n-vertex graph into an $n^3 + 2$-vertex graph (after shortcutting the bottom-layer vertices) we obtain Proposition 1.



(a) Start with a cycle for clause $(x_1 \vee x_2 \vee x_3)$. (b) Then, add a cycle for clause $(x_1 \vee x_2 \vee x_4)$. (c) Finally, add a cycle for clause $(x_1 \vee x_3 \vee x_4)$.

Fig. 5. Not an acyclicity program for $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$.

4.2 Acyclicity Programs for Monotone Formulas

A monotone *(de Morgan) formula* for a n-variate function $\mathcal{P}$ is a tree whose internal nodes are labeled by AND/OR and whose leaves are labeled by inputs, allowing repetitions. The size of a formula is the number of leaves. Monotone formulas naturally compute monotone functions. The monotone formula size remains invariant under duality by de Morgan's laws.

By a standard simulation of (monotone) formulas by (monotone) branching programs (e.g., [GS92]) it is known that every monotone formula can be simulated by a branching program of the same size, i.e., $mF(\mathcal{P}) \geq mBP(\mathcal{P})$. It follows that both $mF(\mathcal{P})$ and $mF(\mathcal{P}^\dagger)$ are lower-bounded by $mCP(\mathcal{P})$, i.e. a formula (and its dual) can be computed by an acyclicity program of the same size:

Proposition 3. *A formula for $\mathcal{P}$ can be converted to an acyclicity program for $\mathcal{P}$ of the same size in at most quadratic time in the size.*

Before we describe this simulation, we discuss an alternative seemingly intuitive approach that **does not** work. To be specific, let us look at the 4-variate CNF formula $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$. It may be tempting to "convert" this formula to the acyclicity program in Fig. 5c, computed by greedily including a cycle for each of the clauses. The two are, however not equivalent, because the assignment $x_1x_2x_3x_4 = \texttt{ftft}$ (`t`, `f` stand for *true*, *false*) is satisfying for the CNF but not for the acyclicity program. (Note that an unintended cycle has been created between nodes 1 and 3, unavoidably.)

To prove Proposition 3 we first convert the formula to a branching program, add a back edge from the accept state to the start state, and then short-circuit all states labeled by the constant 1. The branching program is constructed inductively. A variable x_i is represented by the branching program $\texttt{s(tart)} \to i \to \texttt{a(ccept)}$. Their AND and their OR are represented by the branching programs:

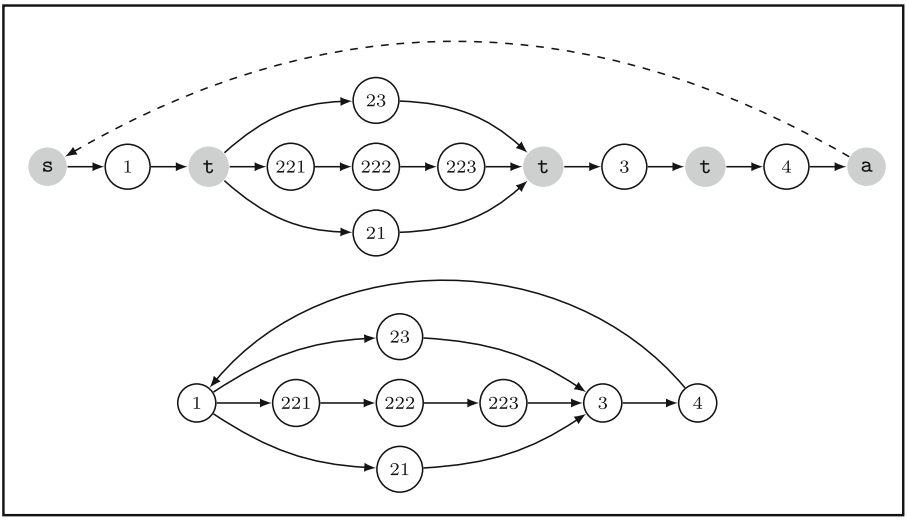


Fig. 6. Branching program representation of $\mathcal{F}^\dagger$ (above) and its simplified representation (below). The acyclicity program is obtained by incorporating the dashed back edge.

$$\mathtt{s} \to A \to \mathtt{t(rue)} \to B \to \mathtt{a} \qquad \begin{array}{ccc} \mathtt{s} & \longrightarrow & A \\ \downarrow & & \downarrow \\ B & \longrightarrow & \mathtt{a} \end{array}$$

In the AND construction, the accept state of A is fused with the start state of B into the `true` constant. To illustrate the transformation, consider the formula:

$$\mathcal{F} = x_1 \vee (x_{21} \wedge (x_{221} \vee x_{222} \vee x_{223}) \wedge x_{23}) \vee x_3 \vee x_4 \ .$$

To convert it into an acyclicity program, we work with its dual:

$$\mathcal{F}^\dagger = x_1 \wedge (x_{21} \vee (x_{221} \wedge x_{222} \wedge x_{223}) \vee x_{23}) \wedge x_3 \wedge x_4 \ .$$

Figure 6 (above) shows the branching program representation of $\mathcal{F}^\dagger$ including the constant states. The cyclicity program representation of $\mathcal{F}^\dagger$, which is the same as the acyclicity program representation of $\mathcal{F}$ is then obtained by adding a back edge from the accept state to the start state. Figure 6 (below) shows the simplification that results from short-cutting the constant states.

We note that this reduction from monotone formulas makes it easy to turn the witness indistinguishability property into zero-knowledge in the common reference string model by following the techniques from [FLS90, CPSV16]: select a random instance x from a hard language as a common reference string and prove the extended statement *"$(x \in L)$ $\vee$ (the original statement)"*.

5 Composition for Predicates Represented by Circuits

In this section we present a sigma-protocol composition scheme with respect to predicates represented by monotone circuits. Unlike Theorem 1, this scheme relies on additional computational assumptions, specifically, we propose an instantiation based on the hardness of computing discrete logarithms.

A direct simulation of monotone circuits by acyclicity programs is inherently inefficient, since $mBP(\mathsf{stC}^\dagger) = mBP(\mathsf{stC})^{\Omega(mBP(\mathsf{stC}))}$, as shown by Grigni and Sipser [GS92]. By Proposition 1, the acyclicity program size of stC is superpolynomial in its branching program size and thus, in its monotone circuit size.

Instead, we emulate a circuit C on n inputs $x_1, \ldots, x_n$ by an acyclicity program A that, in addition to these n inputs as nodes, contains paired auxiliary nodes $(y_1, z_1), \ldots, (y_m, z_m)$. Accepting inputs to C yield accepting inputs to A in which at most one of the two nodes in each pair (y_i, z_i) is set to true (and vice versa). Each auxiliary pair represents a pair of sigma-protocols for proving knowledge of at most one out of two auxiliary witnesses. Such protocols can be constructed and proved secure in the common random string model from the discrete logarithm assumption.

For simplicity, we apply the construction to monotone circuits in the AND/OR basis, but we note that the method can be extended to other threshold gates.

Definition 10 (Boolean circuit). *An* AND/OR boolean circuit *is a directed acyclic graph with n sources, and other nodes of in-degree 2, called internal gates. Sources are labeled by inputs $x_1, \ldots, x_n$, all other nodes are labeled by AND/OR.*

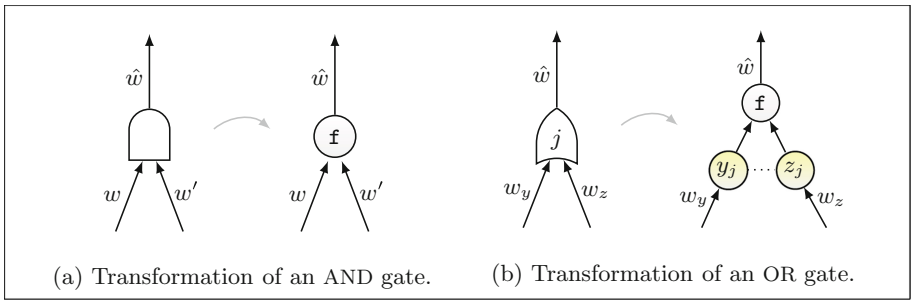
The size of the circuit is the total number of nodes. We consider circuits with a single output node, which compute a monotone function from $\{0,1\}^n$ to $\{0,1\}$: after assigning the input nodes, the internal gates can be evaluated in any order consistent with the underlying graph, determining in a unique output value.

A *tangled assignment* to a pair of boolean variables (y, z) is an assignment in which at most one of the values is true.

Proposition 4. *Given a circuit C with n sources $x_1, \ldots, x_n$ and m OR nodes there exists an acyclicity program A with $n+2m$ nodes $x_1, \ldots, x_n, y_1, z_1, \ldots, y_m, z_m$ such that $x_1, \ldots, x_n$ satisfies C if and only if there exists a tangled assignment to all (y_j, z_j) so that $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ satisfies A. Moreover, A (and $\boldsymbol{y}, \boldsymbol{z}$) can be computed from C (and $\boldsymbol{x}$) in time at most $n + O(m^2)$.*

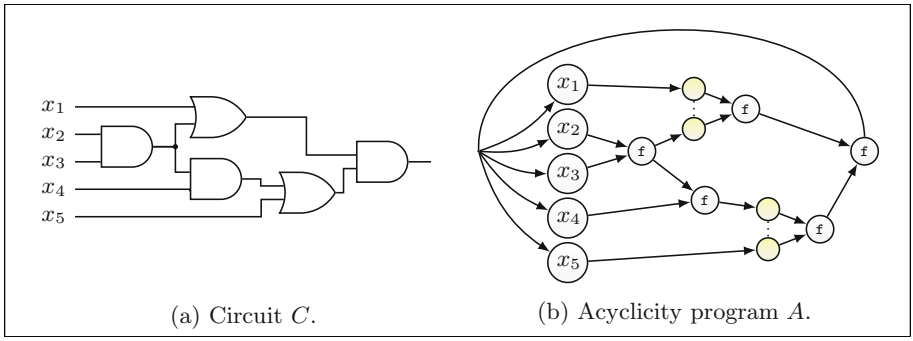
Proof. We replace each AND gate by a `false` node and the j-th OR gate by a gadget consisting of (y_j, z_j) and a `false` node connected as in Fig. 7. We add back edges from the output node to all n input nodes. See Fig. 8 for an example of this transformation.

If we ignore the back edges, we need to argue that satisfying assignments to C can be extended to cover all source-to-sink paths in A and vice versa. First suppose $\boldsymbol{x}$ is a satisfying assignment for C. For each true OR gate j with input wires w_y, w_z, set y_j to true if w_y is false and set z_j to true if w_z is false.



(a) Transformation of an AND gate. (b) Transformation of an OR gate.

Fig. 7. From a circuit to an acyclicity program with tangled nodes.



(a) Circuit C. (b) Acyclicity program A.

Fig. 8. Circuit to acyclicity program conversion example.

(The values to the false OR gates can be set arbitrarily.) We now argue that all source-sink paths in A are covered by induction on circuit size. The claim is vacuous for a circuit with one node. Since the assignment is satisfying, the top gate *top* evaluates to true. If *top* is AND, then by the inductive hypothesis the paths in both subcircuits are covered. If *top* is OR, then the true subcircuit paths are covered by the inductive hypothesis, while the false subcircuit paths (if any) are covered by our definition of y_j and z_j.

We prove the converse also by induction on circuit size. The claim is vacuous for a circuit with one node. Now assume $\boldsymbol{x}$ is not satisfying so the top gate *top* evaluates to false. If *top* is AND, then at least one of the subcircuits is false and the paths in it cannot all be covered. If *top* is OR, then both subcircuits are false, so at least one of them cannot be covered regardless of the choice of y_j and z_j.

The false nodes in A can be short-circuited resulting in an acyclicity program of size $n + 2m$ as desired. □

In order to instantiate the ACP compiler with the acyclicity program A from Proposition 4, we need to describe proof relations and sigma-protocols for each tangled pair. Given two tangled inputs y and z, the prover ought to be able to prove knowledge of a witness for y or a witness for z but not both. The instances

y and z should therefore be chosen in a correlated manner. In order to provide a general definition, we define the notion of *restrictive sampling*. We then show how to build a restrictive sampling scheme from the dlog assumption.

Definition 11 (Restrictive sampling). *A restrictive sampling scheme is a triple of PPT algorithms* $\Upsilon = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Verify})$*:*

- $\mathsf{Setup}(1^\lambda) \to pp$ *is a probabilistic algorithm that outputs a set of public parameters, including an NP-relation* R^* *and a sigma-protocol* Σ^* *for it.*
- $\mathsf{Gen}(pp, b) \to (h_0, h_1, w)$ *is a probabilistic algorithm that, on input* pp *and a bit* b*, outputs two instances* h_0, h_1 *and a witness* w *such that* $R^*(h_b, w) = 1$*.*
- $\mathsf{Verify}(pp, h_0, h_1) \to 0/1$ *is a deterministic algorithm that, on input* pp *and two instances* h_0, h_1 *outputs either* 1 *(acceptance) or* 0 *(rejection).*

We require that a restrictive sampling scheme satisfy the following properties:

Partial Knowledge. *For every PPT algorithm* $\mathcal{A}$*, the following probability is negligible in the security parameter* λ*:*

$$\Pr\left[\begin{array}{rl} pp \leftarrow \mathsf{Setup}(1^\lambda) & \\ (h_0, h_1, w_0, w_1) \leftarrow \mathcal{A}(pp) & : \quad R^*(h_0, w_0) = R^*(h_1, w_1) = 1 \\ & \wedge \quad \mathsf{Verify}(pp, h_0, h_1) = 1 \end{array}\right] .$$

Witness Independence. *For* $b \in \{0,1\}$*, let* $\mathcal{D}_b$ *be the distribution defined as:*

$$\mathcal{D}_b := \left(pp \leftarrow \mathsf{Setup}(1^\lambda);\ (h_0, h_1, w) \leftarrow \mathsf{Gen}(pp, b);\ \mathsf{return}\ (pp, h_0, h_1)\right) .$$

We require distributions $\mathcal{D}_0$ *and* $\mathcal{D}_1$ *be identical.*

Theorem 3. *For some* $\lambda, n \in \mathbb{N}$*, and all* $i \in [n]$*, let* Σ_i *be a sigma-protocol for relation* R_i*, with common challenge space* $\{0,1\}^\lambda$ *and let* Υ *be a restrictive sampling scheme. Let* C *be a monotone* AND/OR *boolean circuit for the set system* $\mathcal{P}$ *and let* $H : \{0,1\}^* \to \{0,1\}^\lambda$ *be a random oracle. The scheme described in Fig. 9 is a NIZK proof of knowledge for* $R_{\mathcal{P}}$*.*

Proof. Completeness, follows from the completeness of the ACP proof system and the *"only if"* part of Proposition 4. The *zero-knowledge* simulator samples random pairs h_j^0, h_j^1 by running $\Upsilon.\mathsf{Gen}$ on input[6] $(pp, 0)$ to simulate the first part of the view of the prover. It then runs the ACP simulator on inputs x_i, h_j^0, h_j^1, inheriting its indistinguishability.

For *knowledge soundness*, given the *partial knowledge* property of Υ, we can assume that algorithms running in expected time t cannot find h_0, h_1, w_0, w_1 such that $R^*(h_0, w_0) = R^*(h_1, w_1) = 1$ and $\Upsilon.\mathsf{Verify}(pp, h_0, h_1) = 1$, where t is the running time of the ACP extractor $\mathcal{E}$. By soundness of the ACP extractor, if t is sufficiently large in terms of the cheating prover complexity, $\mathcal{E}$ produces witnesses for some satisfying set S in A. Our assumption guarantees that this set cannot include a pair of tangled claims h_j^0, h_j^1 such that $\Upsilon.\mathsf{Verify}(pp, h_j^0, h_j^1) = 1$ and the

[6] The simulator may alternatively choose input $(pp, 1)$, in both cases the simulation is perfect due to the *witness independence* property of Υ.

Inputs: Claims $x_1, \ldots, x_n$, circuit C represented by acyclicity program A with tangled pairs $(y_1^0, y_1^1), \ldots, (y_m^0, y_m^1)$.

Common reference string: pp generated with $\Upsilon.\mathsf{Setup}$, random oracle H.

Prover:

Extra input: Witnesses $w_i : i \in S \subseteq [n]$ and $b(j) \in \{0,1\}$ for $j \in [m]$ such that the input set $\{x_i : i \in S\} \cup \{y_j^{b(j)} : j \in [m]\}$ is satisfying for A.

1. For each tangled pair, run $(h_j^0, h_j^1, w_j^*) \leftarrow \Upsilon.\mathsf{Gen}(pp, b(j))$ and set values h_j^0, h_j^1 as statements for the claims associated to nodes $y_j^{b(j)}$ and $y_j^{1-b(j)}$, respectively.
2. Emulate the ACP prover for acyclicity program A on claims $x_1, \ldots, x_n$ with witnesses $w_i : i \in S \subseteq [n]$ and claims $h_1^0, h_1^1, \ldots, h_m^0, h_m^1$ with witnesses w_j^*.

Verifier:

1. Upon receiving instances h_j^0, h_j^1, verify that $\Upsilon.\mathsf{Verify}(pp, h_j^0, h_j^1) = 1$, $\forall j \in [m]$.
2. Emulate the ACP verifier for acyclicity program A, claims x_i w.r.t. relation R_i, and claims h_j^0, h_j^1 w.r.t. relation R^* (defined in pp).

Fig. 9. Composition for predicates represented by circuits.

corresponding witnesses w_j^0, w_j^1 satisfy $R(h_j^0, w_j^0) = R(h_j^1, w_j^1) = 1$. Consequently, the extraction of witnesses for S provided at most one witness for each pair of tangled claims, so by the knowledge soundness of $\mathcal{E}$ and the *"if"* part of Proposition 4, the subset of S indexed by nodes $x_1, \ldots, x_n$ is satisfying for $\mathcal{P}$. □

5.1 Restrictive sampling from the Discrete Logarithm Assumption

Consider the following restrictive sampling scheme:

- $\mathsf{Setup}(1^\lambda)$: sample a λ-bits prime p and select a group $\mathbb{G} = \langle g \rangle$ of order p. Sample $\tau \leftarrow \mathbb{Z}_p$ and let $h := g^\tau$. Define $R^*(h, x) := (h = g^x)$ and let Σ^* be the Schnorr sigma-protocol for $\mathrm{PoK}\{(x) : R^*(h, x) = 1\}$. Return $pp := (p, \mathbb{G}, g, h)$.
- $\mathsf{Gen}(pp, b)$: sample $w \leftarrow \mathbb{Z}_p$, set $h_b := g^w$, $h_{1-b} := hg^{-w}$. Return (h_0, h_1, w).
- $\mathsf{Verify}(pp, h_0, h_1)$: output 1 if $h_0 h_1 = h$, output 0 otherwise.

It is not hard to see that the scheme satisfies the *partial knowledge* property, because an algorithm $\mathcal{A}$ that, given (g, h), outputs h_0, h_1, w_0, w_1 satisfying that $h_0 = g^{w_0}$, $h_1 = g^{w_1}$ and $h_0 h_1 = h$, can be used to compute discrete logarithms, since $w_0 + w_1$ is the discrete logarithm of h in base g.

Furthermore, the scheme satisfies the *witness independence* property, since for any $g, h \in \mathbb{G}$, the following two distributions are identical:

$$\left(w \leftarrow \mathbb{Z}_p;\ \mathsf{return}\ (g^w, hg^{-w})\right) \equiv \left(w \leftarrow \mathbb{Z}_p;\ \mathsf{return}\ (hg^{-w}, g^w)\right) .$$

5.2 Security in the Non-programmable Random Oracle Model

We can prove the security of our construction from Fig. 9 in the NPROM, at the cost of having *soundness* (Definition 6) instead of *knowledge soundness*. For that, observe that the proof of Theorem 3 relies almost completely on the ACP composition security, which can be proven secure in the non-programmable random oracle model. Nevertheless, as shown in Sect. 3.3, in the NPROM the ACP composition achieves witness indistinguishability (and not ZK).

Our construction from Fig. 9 can, however, achieve zero-knowledge if the restrictive sampling scheme is equipped with an extra algorithm that, on input pp and the randomness used to compute it (which can be seen as a trapdoor), can indeed "violate" the *partial knowledge* property. Namely, it can produce instances h_0 and h_1 and valid witnesses for both them, such that $\mathsf{Verify}(pp, h_0, h_1) = 1$. The zero-knowledge simulator could leverage this algorithm to sample the instances associated to tangled pairs, knowing both witnesses of every pair, and then emulate the ACP prover for the acyclicity program on claims $x_1, \ldots, x_n$ (with not witnesses for them) and claims $h_1^0, h_1^1, \ldots, h_m^0, h_m^1$ (with all witnesses for them).

Observe that this extra algorithm can be defined for (but it is not specific to) the instantiation of restrictive sampling based on discrete logarithm. In particular, knowing τ computed during Setup, one can generate $w \leftarrow \mathbb{Z}_p$, compute $h_0 \coloneqq g^w$, $h_1 \coloneqq g^{\tau - w}$ and output $(h_0, h_1, w, \tau - w)$, and the above distribution on (h_0, h_1) is identical to $\mathcal{D}_0$ and $\mathcal{D}_1$ (from the *witness independence* property).

6 Concluding Remarks

Constructing zero-knowledge proof systems by combining sigma-protocols in a compound statement is a powerful technique used for many applications, including anonymous credentials or ring signatures. The most famous and widely used of such composition techniques is the celebrated CDS composition [CDS94], which can be used for compound statements expressed as monotone span programs.

In this work, we have presented a novel technique for combining sigma-protocols into a single non-interactive system for a compound statement. Unlike CDS, our scheme looses the structure of sigma-protocol and it is proven secure in the random oracle model. However, our new methodology enhances the CDS composition in several flavors, including: *new expressivity* (the complexity of our system is linear in the size of the *acyclicity program* representation of the access structure, incomparable to monotone span programs), *more generality* (it is not limited to 2-special sound atomic protocols) and can often lead to *more compact* proofs (as in our circuit composition, where one single transcript is present per atomic statement). Consequently, our results arguably complement previous composition techniques.

Exploring whether our techniques can lead to more efficient zero-knowledge systems achieving post-quantum security is an appealing target for future work.

Observe that the only part of the presented work that relies on classical assumptions is the instantiation of the restrictive sampling scheme. But even this primitive could be instantiated under post-quantum assumptions. For example, it could be instantiated under the Short Integer Solution (SIS) assumption [GPV08]. Note that the map $f_A(\boldsymbol{x}) = A\boldsymbol{x} \pmod q$ is almost surjective, so given $\boldsymbol{y}$, one could sample $\boldsymbol{y_0}$ and $\boldsymbol{y_1}$ such that $\boldsymbol{y_0} + \boldsymbol{y_1}$, and know a short $\boldsymbol{x_b}$ such that $A\boldsymbol{x_b} = \boldsymbol{y_b}$. For that, sample a short $\boldsymbol{x_b}$ first and set $\boldsymbol{y_b} \coloneqq A\boldsymbol{x_b}$, $\boldsymbol{y_{1-b}} \coloneqq \boldsymbol{y} - A\boldsymbol{x_b}$. The *partial knowledge* property would hold because if, given $\boldsymbol{y}$, one could find short $\boldsymbol{x_0}, \boldsymbol{x_1}$ such that $A\boldsymbol{x_0} + A\boldsymbol{x_1} = \boldsymbol{y}$, then $\boldsymbol{x_0} + \boldsymbol{x_1}$ would be a short preimage of $\boldsymbol{y}$ under A. The *witness independence* is guaranteed by the fact that the distribution of $\boldsymbol{y} = A\boldsymbol{x}$ for a randomly chosen short $\boldsymbol{x}$ is close to uniform.

Acknowledgment. We would like to thank Gautam Prakriya for helpful discussions on monotone span programs and Gregory Neven for very fruitful discussions and all his feedback. Finally, we would like to thank all anonymous reviewers for their valuable time and useful comments. The third and fifth authors are supported by Hong Kong RGC GRF grant CUHK 14209419, and ISF grant No. 1399/17 and Project PROMETHEUS (Grant 780701), respectively.

References

[AAB+20] Abe, M., Ambrona, M., Bogdanov, A., Ohkubo, M., Rosen, A.: Non-interactive composition of sigma-protocols via Share-then-Hash. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12493, pp. 749–773. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64840-4_25

[AB87] Alon, N., Boppana, R.B.: The monotone circuit complexity of Boolean functions. Combinatorica **7**(1), 1–22 (1987)

[AHIV17] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 2087–2104. ACM Press, October/November 2017

[AOS02] Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 415–432. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36178-2_26

[BBB+18] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press, May 2018

[BCC+15] Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C.: Short accountable ring signatures based on DDH. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS 2015. LNCS, vol. 9326, pp. 243–265. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24174-6_13

[BCG+17] Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. Part III, volume 10626 of LNCS, pp. 336–365. Springer, Heidelberg (2017)

[BCR+19] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4

[BDG20] Bellare, M., Davis, H., Günther, F.: Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 3–32. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_1

[BGW99] Babai, L., Gál, A., Wigderson, A.: Superpolynomial lower bounds for monotone span programs. In: Combinatorica, vol. 19, pp. 301–319 (1999). https://doi.org/10.1007/s004930050058

[BM90] Bellare, M., Micali, S.: Non-interactive oblivious transfer and applications. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 547–557. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_48

[BN06] Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006, pp. 390–399. ACM Press, October/November 2006

[BPW12] Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: pitfalls of the fiat-Shamir heuristic and applications to Helios. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 626–643. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_38

[BR93] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93, pp. 62–73. ACM Press, November 1993

[BSS02] Bresson, E., Stern, J., Szydlo, M.: Threshold ring signatures and applications to ad-hoc groups. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 465–480. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_30

[CDM00] Cramer, R., Damgård, I., MacKenzie, P.: Efficient zero-knowledge proofs of knowledge without intractability assumptions. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 354–372. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46588-1_24

[CDS94] Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_19

[CFQ19] Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2075–2092. ACM Press, November 2019

[CPSV16] Ciampi, M., Persiano, G., Siniscalchi, L., Visconti, I.: A transform for NIZK almost as efficient and general as the Fiat-Shamir transform without programmable random oracles. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 83–111. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49099-0_4

[Cra97] Cramer, R.: Modular Design of Secure yet Practical Cryptographic Protocols. PhD thesis, University of Amsterdam, January 1997

[DK18] Deshpande, A., Kalai, Y.: Proofs of ignorance and applications to 2-message witness hiding. Cryptology ePrint Archive, Report 2018/896 (2018)

[FHJ20] Fischlin, M., Harasser, P., Janson, C.: Signatures from sequential-OR Proofs. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 212–244. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_8

[FLS90] Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: 31st FOCS, pp. 308–317. IEEE Computer Society Press, October 1990

[FLWL20] Feng, H., Liu, J., Wu, Q., Li, Y.-N.: Traceable ring signatures with post-quantum security. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 442–468. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40186-3_19

[FS87] Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_12

[GKM+18] Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 698–728. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_24

[GMW86] Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: 27th FOCS, pp. 174–187. IEEE Computer Society Press, October 1986

[GPS08] Galbraith, S.D., Paterson, K.G., Smart, N.P.: Pairings for cryptographers. Discret. Appl. Math. **156**(16), 3113–3121 (2008)

[GPV08] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 197–206. ACM Press, May 2008

[Gro16] Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11

[GS92] Grigni, M., Sipser, M.: Monotone complexity, pp. 57–75. In: Proceedings of the London Mathematical Society, Symposium on Boolean Function Complexity (1992)

[KW93] Karchmer, M., Wigderson, A.: On span programs. In: Proceedings of the Eighth Annual Structure in Complexity Theory Conference, pp. 102–111. IEEE Computer Society (1993)

[Lin15] Lindell, Y.: An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9014, pp. 93–109. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46494-6_5

[LLNW16] Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 1–31. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_1

[LLNW17] Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based PRFs and applications to E-Cash. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10626, pp. 304–335. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70700-6_11

[LMR19] Lai, R.W.F., Malavolta, G., Ronge, V.: Succinct arguments for bilinear group arithmetic: practical structure-preserving cryptography. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2057–2074. ACM Press, November 2019

[Max11] Maxwell, G.: Zero knowledge contingent payment. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment (2011)

[MBKM19] Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2111–2128. ACM Press, November 2019

[Nie02] Nielsen, J.B.: Separating random oracle proofs from complexity theoretic proofs: the non-committing encryption case. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 111–126. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_8

[NTWZ19] Nguyen, K., Tang, H., Wang, H., Zeng, N.: New code-based privacy-preserving cryptographic constructions. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11922, pp. 25–55. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_2

[PR18] Pitassi, T., Robere, R.: Lifting nullstellensatz to monotone span programs over any field. In: Diakonikolas, I., Kempe, D., Henzinger, M. (eds.) 50th ACM STOC, pp. 1207–1219. ACM Press, June 2018

[PS00] Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptol. **13**(3), 361–396 (2000)

[Raz85] Razborov, A.A.: Lower bounds on monotone complexity of the logical permanent. Math. Notes Acad. Sci. USSR **37**(6), 485–493 (1985)

[RST01] Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_32

[Ste96] Stern, J.: A new paradigm for public key identification. IEEE Trans. Inf. Theory **42**(6), 1757–1768 (1996)

[Tar88] Tardos, É.: The gap between monotone and non-monotone circuit complexity is exponential. Combinatorica **8**(1), 141–142 (1988)