Efficient NIZK Arguments via Parallel Verification of Benes Networks

Helger Lipmaa

Institute of Computer Science, University of Tartu, Estonia

Abstract. We work within the recent paradigm, started by Groth (ASIACRYPT 2010), of constructing short non-interactive zero knowledge arguments from a small number basic arguments in a modular fashion. The main technical result of this paper is a new permutation argument, by using product and shift arguments of Lipmaa (2014) and a parallelizable variant of the Beneš network. We use it to design a short non-interactive zero knowledge argument for the **NP**-complete language CircuitSAT with $\Theta(n \log^2 n)$ prover's computational complexity, where n is the size of the circuit. The permutation argument can be naturally used to design direct NIZK arguments for many other **NP**-complete languages.
Keywords. Beneš networks, modular NIZK arguments, perfect zero knowledge, product argument, shift argument, shuffle.

1 Introduction

To construct a cryptographic protocol secure against malicious adversaries, one needs to employ zero-knowledge proofs [14]. To enable verifiability even in the case the prover is not online, zero-knowledge proofs must be non-interactive. Since in many applications, the proof will be verified by many independent verifiers (e.g., by many voters in an e-voting applications), then it is also desirable that the proofs be short. Finally, in several applications (like delegation of computation) one needs to construct a short non-interactive zero-knowledge (NIZK) proofs for generic **NP**-complete languages (like CircuitSAT).

Motivated by such applications, Groth [16] constructed a non-interactive zero knowledge and computationally sound NIZK proof (i.e., an NIZK argument) for CircuitSAT with communication that is a small constant number of group elements. Groth's CircuitSAT argument is constructed in a modular fashion from a small number of more basic arguments. More precisely, it consists of less than 10 basic Hadamard product (given three committed vectors, one of them is an entry-wise product of the other two) and permutation (given two committed vectors and a public permutation ϱ, the coefficients of one vector are ϱ-permuted coefficients of the second vector) arguments. Unfortunately, both basic arguments have CRS length (in group elements) and prover's computation (in exponentiations) quadratic in the vector dimension n, and as the result, the corresponding complexity parameters of the CircuitSAT argument are quadratic in the circuit size $|C|$. Due to this, Groth's original argument can only be used for relatively small $|C|$.

Subsequent research has improved on Groth's modular approach by both increasing the efficiency of Groth's basic arguments and studying the possibility to implement arguments for **NP**-complete languages by using different (hopefully more efficient) basic arguments. Lipmaa [19] improved Groth's basic arguments, by using progression-free sets (see Sect. 2). For vectors of length n, Lipmaa's basic arguments have CRS length of $\Theta(r_3^{-1}(n))$ group elements, while the prover's computation is dominated by $\Theta(n^2)$ scalar additions. Here, $r_3(N)$ is the size of some progression-free subset of $[N] = \{1, \ldots, N\}$. However, due to quadratic prover's computation, also Lipmaa's CircuitSAT argument is useful only for small $|C|$. In [12], Fauzi, Lipmaa and Zhang implemented Lipmaa's product argument in prover's computational complexity $\Theta(r_3^{-1}(n) \log r_3^{-1}(n))$ multiplications. Finally, in [21], Lipmaa used a different approach to construct a product argument with the CRS length of $\Theta(n)$ group elements and prover's computation of $\Theta(n \log n)$ non-cryptographic and $\Theta(n)$ cryptographic operations. His product argument is based on the so called *interpolating commitment scheme*.

However, none of the subsequent work improved on the asymptotic computation of the permutation argument. Instead, [12] proposed a new z-left/right shift argument (given two committed vectors, one of them is a coordinate-wise z-shift of the second vector) with linear CRS size and prover's computation. They then used the product and shift arguments to implement computationally more efficient modular NIZK arguments for the **NP**-complete languages SetPartition, SubsetSum and DecisionKnapsack. (For the latter, they also need an efficient NIZK range argument, [8], which can be constructed by using product and shift arguments as shown in [12,21].) Lipmaa showed in [21] how to use the interpolating commitment scheme to efficiently implement the shift arguments, and thus also the corresponding **NP**-complete languages. Since the prover's computation in the arguments from [21] is $\Theta(n \log n)$, these arguments are usable for much larger input sizes n than the CircuitSAT arguments from [16,19].

On the other hand, while [12] proposed computationally efficient NIZK arguments for SetPartition, SubsetSum and DecisionKnapsack, it did not propose one for CircuitSAT. CircuitSAT is a natural language to be used in various cryptographic applications, like verifiable computation. Moreover, CircuitSAT is often used as a benchmark to show the efficiency of new zero-knowledge techniques. Therefore, construction of a CircuitSAT argument with subquadratic CRS length and prover's computation and short (i.e., polylogarithmic) communication is still an important open problem.

Finally, any further advances will help one to better understand the power and limitations of the modular approach. It is likely (we will motivate this intuition later) that efficient modular arguments for other **NP**-complete languages can be constructed by using the permutation argument, and thus it is important to construct a permutation argument with better prover's computation.

Our contributions. We construct a new permutation argument for n-bit vectors, by using Beneš network [4] to combine $\Theta(\log n)$ product and shift arguments. This results in a permutation argument with $\Theta(n \log^2 n)$ prover's com-

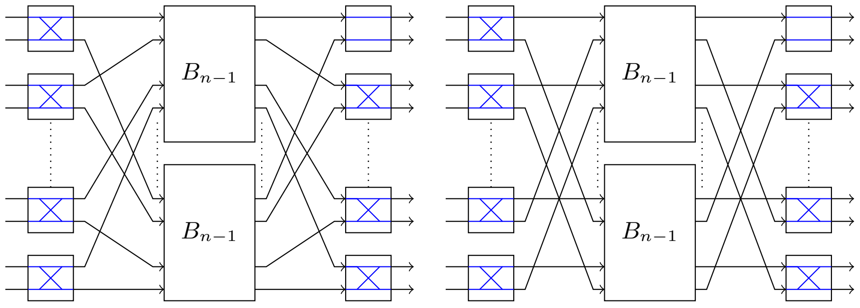

Fig. 1. The usual definition of Beneš network B_n (left), and our definition (right). The boxes with blue X denote crossbars in the input and output stage

putation and logarithmic communication. We then plug the new permutation argument in to the NIZK arguments of [16,19] to obtain a CIRCUITSAT argument with the same asymptotic complexity. Apart from being the most efficient CIRCUITSAT argument following from the modular paradigm, the new CIRCUITSAT argument is the most complex known yet still competitively efficient NIZK argument in the modular framework. We also outline how to use the new permutation argument to construct efficient NIZK arguments for other **NP**-complete languages.

Let S_n denote the symmetric group. The Beneš network $B_{\log_2 n}$, see Fig. 1, is a multistage interconnection network [18]. Every stage consists of a number of crossbars, the outputs of which are wired to the inputs of the next stage's crossbars. The only changeable elements are the S_2-permutations implemented by each of the crossbars. Assuming that n is a power of 2, the definition of a Beneš network $B_{\log_2 n}$ is usually given recursively, from two copies of $B_{\log_2 n-1}$, an input stage (that has $n/2$ crossbars) and an output stage (that has $n/2-1$ crossbars; one crossbar can be fixed to implement the identity map), as in Fig. 1 (left). The network B_1 consists of a single crossbar. Each crossbar implements a permutation from S_2.

The Beneš network $B_{\log_2 n}$ implements a permutation from S_n by using nearly optimal $n \cdot \log_2 n - n + 1$ crossbars and $2\log_2 n - 1$ stages ($\log_2 n$ input stages, followed by a single stage that implements B_1, followed by $\log_2 n$ output stages). A stage here consists of $n/2$ parallel crossbars, followed by wirings from the outputs of the crossbars of the current stage to the inputs of the next stage.

A straightforward use of Beneš networks results in a permutation (and shuffle) argument of length $\Theta(n \log n)$ [1,2]. We achieve logarithmic communication and $\Theta(n \log^2 n)$ prover's computation by verifying *in parallel* that all crossbars of the same stage of the Beneš network have been implemented correctly; this seems to be a novel use of Beneš networks. We use a small number of product and shift arguments from [21] (each of which has $\Theta(n \log n)$ prover's computation and constant communication) at every stage. Since there are $\Theta(\log n)$ stages, the resulting permutation argument consists of $\Theta(\log n)$ basic arguments.

It is not immediately clear from the standard recursive definition of Beneš networks (Fig. 1, left) how to do parallel verification. We use a slight variant (that we call a *parallelizable Beneš network*) of the usual Beneš network, see Fig. 1 (right), that is also a well-known but not the usually presented definition. In that variant, at every stage, input elements of the stage are permuted twice.

First, in crossbars, the ith input of a stage will become to the jth input of the wirings, where depending on the S_2-permutation implemented by the concrete crossbar, $j \in \{i-1 \mod n, i, i+1 \mod n\}$. That is, the jth output bit of the crossbars is either equal to the jth bit of the input $\boldsymbol{a}$ of the crossbars, or it is equal to the jth bit of either 1-left or 1-right shift of $\boldsymbol{a}$. Which case happens depends on the number of the stage, the index i, and the input to the network.

Second, after the wiring, the ith output of the crossbars will go to the jth input of the next stage, where $j \in \{i-z \mod n, i, i+z \mod n\}$, where z is a stage-dependent constant. That is, the jth output bit of the wiring is either equal to jth input bit of the wiring, or to the jth bit of a z-left or z-right shift of the input of the wiring. Importantly, which case happens depends only on the number of the stage and on i, and not on the input to the network, that is, not on the concrete permutation. In the usual variant of the Beneš networks as in Fig. 1 (left), all possible z-shifts, $z \leq n/2$, are used in wirings of every stage.

These observations make it possible to verify correct implementation of one stage of the Beneš network by using a small constant number of shift and product arguments. Therefore, correct implementation of the full Beneš network can be verified by using a logarithmic number of shift and product arguments. This results in a permutation argument with related complexity parameters.

While the resulting new permutation argument, has larger communication than Groth's and Lipmaa's permutation arguments [19], it has smaller prover's computation.[1] Since quadratic prover's computation is the main obstacle in actually applying Groth's short arguments, this means that now one can use a significantly larger value of n. The relatively minor increase in the communication (from $\Theta(1)$ to $\Theta(\log n)$, where n is the circuit size) is comparatively less important.

Both the prover and the verifier have to execute an online routing algorithm that outputs the necessary shift amounts. For (the standard variant of the) Beneš network, routing can be done in time $\Theta(n \log n)$ on a single processor [28,24], or $\Theta(\log^2 n)$ on a parallel computer [23]. Clearly, routing algorithms can be modified to work with the parallelizable Beneš networks without any loss in efficiency.

We can use the methodology of Groth [16] and Lipmaa [19] to construct a CircuitSAT argument, given the product argument of [21] and the permutation argument of the current paper. Hence, one can construct modular NIZK arguments of knowledge for **NP**-complete languages SetPartition [12], SubsetSum [12], DecisionKnapsack [12] and CircuitSAT ([16], [19], and the current paper) that are all based on the simple "parallel programming lan-

[1] It also has verifier's computation of $\Theta(n \log n)$ multiplications. This is larger than in [16], but smaller than in [19], where one needed $\Theta(n)$ exponentiations. (Since one exponentiation takes $\Theta(\log p)$ multiplications, and $n \ll p$.)

guage" consisting of two arguments, Hadamard product and (z-)shift. The fact that one can construct a CircuitSAT argument from the weaker set of basic primitives than in previous papers [16,19], where the programming language consisted of the product and (arbitrary) permutation arguments, can be seen as an additional contribution of the current paper. Since every shift argument is quadratically more efficient than the permutation argument of [16,19], using $\Theta(\log n)$ of them results still in a major win in efficiency.

Interestingly, the arguments for SetPartition, SubsetSum and DecisionKnapsack [12,21] are more efficient than the new argument for CircuitSAT, and the new CircuitSAT argument is by far the most complex existing program in such a language. We leave it as an open question to design efficient direct (i.e., one obtained without a reduction to another **NP**-complete language) NIZK arguments for other **NP**-complete languages, and to study why some languages have more efficient arguments than others.

However, one can use the permutation argument and the product argument to construct NIZK argument for many **NP**-complete languages as follows. First, use a permutation argument (by using a *secret* permutation) to permute the inputs randomly. (See App. B for a description of how to efficiently modify the new permutation argument to handle secret permutations. The resulting argument can be seen as a *committed shuffle* argument that one committed vector is a shuffle of another vector.) Second, use a product argument to clear the bits of the inputs that are not needed to verify the witness (this can be done by checking that the permuted input $\boldsymbol{a}$ and the cleared version $\boldsymbol{b}$ satisfy $a_i \cdot c_i = b_i$, where c_i is a publicly known Boolean vector). After that, one reveals the cleared version of the inputs, from which one can — in the case of many **NP**-complete languages — directly verify the witness. This results in direct NIZK arguments for a large family of **NP**-complete languages.

As a high-level example, in the case of the HamiltonianPath argument, the prover has to show that there exists a path that visits each vertex once. Here, the prover chooses a random secret permutation ϱ that permutes the input (which is usually represented as an $n \times n$ adjacency matrix), modulo the restriction that the Hamiltonian path visits vertices in the order $1 \to 2 \to 3 \ldots \to n \to 1$. The prover uses a committed shuffle argument to show that the (secret) permutation was done correctly. The prover then uses a product argument to clear $n^2 - n$ elements of the adjacency matrix, and then opens the cleared adjacency matrix. In the case that the graph had an Hamiltonian path, the opened adjacency matrix has 1-s in positions $(i, i + 1 \mod n)$. The language Clique has a very similar argument, except that one verifies that the opened adjacency matrix starts with an all-1 $m \times m$ matrix for some parameter m. We leave the precise details together with bringing further examples to the future work.

We remark that the parallel programming model that consists of shift and entry-wise addition and product is well-known both in theoretical computer science [26] and parallel computing [6], but up to our knowledge it has not been applied before the current line of work (starting with [16] and made explicit in [19,12]) to verify the solutions of **NP**-complete languages — even without requiring the zero knowledge property.

Comparison to the QSP/QAP-Based Approach. In [13], Gennaro, Gentry, Parno, and Raykova showed how to construct a more efficient (linear CRS, $\Theta(n \log^3 n)$ prover's computation, constant communication, and linear verifier's computation) NIZK argument for CircuitSAT. The prover's computation can be improved to $\Theta(n \log^2 n)$ when one uses bilinear groups of well-chosen prime order p [13,20,21]. Their — based either on Quadratic Span Programs or Quadratic Arithmetic Programs — argument has been further improved in say [5,3,20].

First, the arguments of [13] are directly tailored for (arithmetic) CircuitSAT. It is unclear how to use these arguments for any **NP**-complete language $\mathcal{L}$, except via a potentially costly polynomial-time reduction. Even if this reduction takes time say $\Theta(n \log^2 n)$, the resulting argument for $\mathcal{L}$ becomes too slow (in prover's computation).

Second, our approach is completely different and therefore still interesting by itself. It is clearly beneficial to study different approaches to the same problem.

2 Preliminaries

Let $[L, H] = \{L, L+1, \dots, H-1, H\}$ and $[H] = [1, H]$. By $\boldsymbol{a}$, we denote the vector $\boldsymbol{a} = (a_1, \dots, a_n)$. For a group $\mathbb{G}$, we utilize the fact that $\mathbb{G}^2 = \mathbb{G} \times \mathbb{G}$ is a group and thus aggressively use notation like $(g, h)^a$ or $(g_1, h_1) \cdot (g_2, h_2)$. If $y = h^x$, then let $\log_h y := x$. We abbreviate probabilistic polynomial-time as PPT, and let $\mathrm{negl}(\kappa)$ be an arbitrary negligible function.

Preliminaries on Interconnection Networks. The basic components of a switching network [18] (e.g., see Fig. 1 or Fig. 2) are crossbars, and links that connect crossbars. A crossbar with n inlets and m outlets is denoted by X_{nm}. Any matching (one-to-one mapping) between the inlets and the outlets of a crossbar is considered routable, that is, a crossbar is *nonblocking*. By using different terminology, a crossbar X_{nn} can be fixed to implement any permutation from S_n. Two sets of crossbars are connected to outside world. One set of such crossbars is called input crossbars and another set output crossbars. The links on an input (output) crossbar linking to outside world are called inputs (outputs) of the network. An (N, M)-network has N inputs and M outputs, and will be called an N-network if $M = N$.

A network is (i) strict-sense nonblocking if one can, given the values of ϱ for some i together with corresponding routes in the network, always find non-intersecting routes for the rest of the values of ϱ, (ii) wide-sense nonblocking if there is an algorithm for establishing paths in the network one after another, so that after each path is established, it is still possible to connect any unused input to any unused output, (iii) *rearrangeable* if it is only required that such choice of routes can be done for all i at the same time.

An (m, n)-Clos network [9,18] for a permutation $\varrho \in S_N$, where $N = mn$, is a three-stage network to implement ϱ, in which each stage is composed in a number of smaller crossbars. The first stage has m crossbars X_{nn}, the second stage has n crossbars X_{mm}, and the third stage has m crossbars X_{nn}. Each input crossbar is connected to each middle stage crossbar, and each middle stage crossbar is connected to each output crossbar. See Fig. 2. To implement an arbitrary permutation from S_{mn} it suffices to use a rearrangeable non-blocking network [9,4,18], for which one can choose an (m, n)-Clos network. (In fact, the Clos network is strict-sense nonblocking.) For this one just has to choose the $2m + n$ small permutations accordingly.

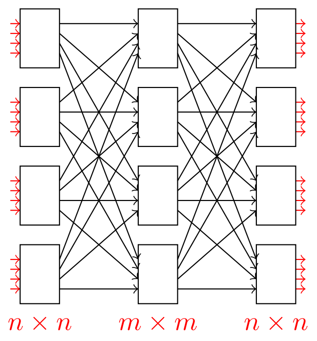

Fig. 2. An $(4, 4)$-Clos network for permutations from S_{16}

Beneš networks [4,18] implement an arbitrary permutation $\varrho : [N] \to [N]$ by using $2 \log_2 N - 1$ stages of X_{22} crossbars. Beneš network B_n (for permutation from S_{2^n}) is usually defined recursively by connecting an initial stage of $n/2$ X_{22} crossbars, two B_{n-1} networks and a final stage of $(n/2-1)$ X_{22} crossbars, as in Fig. 1, left. Clearly, the Beneš network $B_{\log_2 N}$ has $(N \log_2 N - N + 1)$ X_{22} crossbars. While the Clos network is strict-sense nonblocking, the Beneš network is only rearrangeable. Recall that X_{22} implements either an identity function id, $\mathsf{id}(x, y) = (x, y)$, or a flip flip, $\mathsf{flip}(x, y) = (y, x)$.

Waksman [28] and Opferman and Tsao-Wu [24] proposed efficient routing algorithms for the standard variant (Fig. 1, left) of the Beneš network. Their algorithm instantiates the $N \log_2 N - N + 1$ crossbars $\varrho_{i,j}$, given the input permutation ϱ, in time $\Theta(N \log N)$. Nassimi and Sahni [23] proposed a parallel routing algorithm for the Beneš network that works in time $\Theta(\log^2 N)$, given N parallel processors.

Cryptographic Preliminaries. On input 1^κ, where κ is the security parameter, a *bilinear map generator* returns $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g_1, g_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are three multiplicative cyclic groups of prime order p, g_z is a generator of $\mathbb{G}_z$ for $z \in \{1, 2\}$, and $\hat{e}$ is an efficient bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that satisfies in particular the following two properties: (i) $\hat{e}(g_1, g_2) \neq 1$, and (ii) $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$. Thus, if $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1^c, g_2^d)$ then $ab = cd \mod p$.

The security of the arguments of the current paper depends on the q-type computational and knowledge [10] assumptions, variants of which have been studied and used in say [15,11,16,8,19,5,12]. In fact, all known (to us) adaptive short NIZK arguments are based on q-type assumptions about genbp. We refer to [21] for a description of these assumptions.

Trapdoor commitment scheme is a randomized cryptographic primitive in the CRS model [7] that takes a message and outputs its commitment. It consists of two efficient algorithms gencom (that outputs a CRS and a trapdoor) and $\mathcal{C}\mathrm{om}$ (that, given the CRS, a message and a randomizer, outputs a commitment), and must satisfy the following three security properties. **Computational binding:** without access to the trapdoor, it is intractable to open the same commitment to two different messages. **Perfect hiding:** commitments of any two messages have the same distribution. **Trapdoor:** given an access to the original message, the randomizer and the trapdoor, one can open a commitment to (say) 0 to an arbitrary message. See, e.g., [16] for formal definitions.

We use the following *interpolating commitment scheme* from [21]. Assume n is a power of 2, and assume that the group order p is such that there exist a nth primitive unit of root modulo p [21]. Let this unit be ω. Let $f_0(X) = Z(X)$ and $f_i(X) = \ell_i(X)$ be polynomials, defined as follows:

1. $Z(X) = \prod_{j=1}^{n}(X - \omega^{j-1}) = X^n - 1$, i.e., $Z(\omega^{i-1}) = 0$,
2. $\ell_i(X) = \prod_{j \neq i}(X - \omega^{j-1}) / \prod_{j \neq i}(\omega^{i-1} - \omega^{j-1})$ is the ith Lagrange basic polynomial, i.e., $\ell_i(\omega^{i-1}) = 1$ and $\ell_i(\omega^{j-1}) = 0$ for $i \neq j$.

The CRS generation algorithm $\mathsf{gencom}(1^\kappa)$ first sets $\mathsf{gk} \leftarrow \mathsf{genbp}(1^\kappa)$, and then generates the trapdoor $(\sigma, \alpha) \leftarrow \mathbb{Z}_p^2$ (with $\sigma \neq 0$). It then outputs the common reference string $\mathsf{ck} = ((g_1, g_2^\alpha)^{f(\sigma)})_{f \in \{Z, \ell_1, \dots, \ell_n\}}$. The commmon reference string ck is made public, while the trapdoor (σ, α) is only used in security proofs. Define[2] $\mathcal{C}\mathrm{om}_{\mathsf{ck}}((a_1, \dots, a_k); r) := \prod_{i=1}^{k}((g_1, g_2^\alpha)^{\ell_i(\sigma)})^{a_i} \cdot ((g_1, g_2^\alpha)^{Z(\sigma)})^r = (g_1, g_2^\alpha)^{rZ(\sigma) + \sum_{i=1}^{k} a_i \ell_i(\sigma)}$. The computation of $\mathcal{C}\mathrm{om}$ can be sped up by using efficient multi-exponentiations algorithms [27,25]. We denote the output of the commitment either by (A_1, A_2^α) or by $(A, \hat{A})$.

As shown in [21], the interpolating commitment scheme is perfectly hiding, and computationally binding. Moreover, if a suitable knowledge assumption holds, then for any non-uniform PPT $\mathcal{A}$ that outputs a valid commitment C, there exists a non-uniform PPT extractor that, given $\mathcal{A}$'s input together with $\mathcal{A}$'s random coins, extracts a valid opening of C.

An NIZK argument for a language L consists of three algorithms, gencrs, $\mathcal{P}$ and $\mathcal{V}$. The CRS generation algorithm gencrs takes as input 1^κ (and possibly some other, public, language-dependent information) and outputs the CRS crs and the trapdoor td. The prover's algorithm $\mathcal{P}$ takes as an input crs together with a statement x and a witness w, and outputs an argument π. The verifier's algorithm $\mathcal{V}$ takes as an input crs together with a statement x and an argument π, and either accepts or rejects.

We expect the argument to be (i) perfectly complete (the honest verifier always accepts the honest prover), (ii) perfectly zero knowledge (there exists an efficient simulator who can, given x, crs and td, output an argument that comes from the same distribution as the argument produced by the prover), and (iii) computationally sound (if $x \notin L$, then an arbitrary non-uniform PPT prover has only a negligible success in creating a satisfying argument). We refer to say [16,19] for formal definitions.

[2] Here and in what follows, elements of the form $(g, g^\alpha)^x$, where α is a secret random key, can be thought of as a *linear-only encoding* of x, see [5] for a discussion

Assume that Γ is a (trapdoor) commitment scheme that commits to elements $\boldsymbol{a} = (a_1, \dots, a_n) \in \mathbb{Z}_p^n$ for a prime p and integer $n \geq 1$. In an *Hadamard product argument* [16,19], the prover aims to convince the verifier that given commitments A, B and C, he can open them as $A = \mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{a}; r_a)$, $B = \mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{b}; r_b)$, and $C = \mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{c}; r_c)$, such that $c_i = a_i b_i$ for $i \in [n]$. In a *z-right shift argument* [12], the prover aims to convince the verifier that for two commitments A and B, he knows how to open them as $A = \mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{a}; r_a)$ and $B = \mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{b}; r_b)$, such that $a_i = b_{i+z}$ for $i \in [n-z]$, and $a_i = 0$ for $i \in [n-z+1, n]$. That is, $(a_1, \dots, a_n) = (b_z, \dots, b_n, 0, \dots, 0)$. We define the z-left shift argument dually. In a *permutation argument* [16,19], the prover aims to convince the verifier that given commitments A and B and a permutation $\varrho \in S_n$, he can open them as $A = \mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{a}; r_a)$ and $B = \mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{b}; r_b)$, such that $b_i = a_{\varrho(i)}$ for $i \in [n]$.

We recall the following theorem from [21], but we only give the details that are needed in the following (for example, we state the precise security guarantees, but we leave out the description of the arguments themselves and the precise security assumptions). We do it to avoid overdependence on the concrete implementation of the basic arguments. We remark that the condition (2) in the theorem statement is called co-soundness, see [17] for an explanation.

Theorem 1 (Security of the product argument [21]). *Let $n = \mathrm{poly}(\kappa)$. Let $\mathcal{C}om$ be the interpolating commitment scheme.*

1. *The product argument from [21] is perfectly complete and perfectly witness-indistinguishable.*
2. *(*Co-soundness:*) If genbp satisfies a q-type computational assumption from [21], then a non-uniform probabilistic polynomial-time adversary against the product argument from [21] has negligible chance, given $\mathsf{crs} \leftarrow \mathsf{gencrs}(1^\kappa, n)$ as an input, of outputting $inp_\times = (A, \hat{A}, B, \hat{B}, C, \hat{C})$ and an accepting argument $\pi_\times$ together with a witness $w_\times = (\boldsymbol{a}, r_a, \boldsymbol{b}, r_b, \boldsymbol{c}, r_c)$, such that*
   - *(i) $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{Z}_p^n$ and $r_a, r_b, r_c \in \mathbb{Z}_p$,*
   - *(ii) $(A, \hat{A}) = \mathcal{C}om(\mathsf{ck}; \boldsymbol{a}; r_a)$, $(B, \hat{B}) = \mathcal{C}om(\mathsf{ck}; \boldsymbol{b}; r_b)$, and $(C, \hat{C}) = \mathcal{C}om(\mathsf{ck}; \boldsymbol{c}; r_c)$, and*
   - *(iii) for some $i \in \{1, \dots, n\}$, $a_i b_i \neq c_i$.*

The product argument of [21] has CRS of $\Theta(n)$ group elements, prover's computation $\Theta(n \log n)$ multiplications, verifier's computation 3 pairings, and communication of 1 group element. We denote this argument as $[\![(A, \hat{A})]\!] \circ [\![(B, \hat{B})]\!] = [\![(C, \hat{C})]\!]$.

For a vector $\boldsymbol{a} = (a_1, \dots, a_n)$, denote $\mathsf{lsft}_1(\boldsymbol{a}) := (a_2, a_3, \dots, a_n, 0)$ and $\mathsf{rsft}_1(\boldsymbol{a}) := (0, a_1, a_2, \dots, a_{n-1})$. For $z > 1$, let $\mathsf{lsft}_z(\boldsymbol{a}) := \mathsf{lsft}_1(\mathsf{lsft}_{z-1}(\boldsymbol{a}))$ and $\mathsf{rsft}_z(\boldsymbol{a}) := \mathsf{rsft}_1(\mathsf{rsft}_{z-1}(\boldsymbol{a}))$. In the case of the shift argument, [21] proved an even weaker version of soundness, see [16] for explanation.

Theorem 2 (Security of the right shift argument [21]). *Let $n = \mathrm{poly}(\kappa)$. Let $\mathcal{C}om$ be the interpolating commitment scheme.*

1. *The shift argument of [21] is perfectly complete and perfectly witness-indistinguishable.*
2. *(*Weak soundness:*) Let Φ^z_{rsft} be as in [21]. If* genbp *satisfies a q-type computational assumption from [21], then a non-uniform probabilistic polynomial time adversary against the shift argument of [21] has negligible chance, given* $\mathsf{crs} \leftarrow \mathsf{gencrs}(1^\kappa, n)$ *as an input, of outputting* $inp_{\mathsf{rsft}} \leftarrow (A, \hat{A}, B, \hat{B})$ *and an accepting argument* (π, π^β) *together with a witness* $w_{\mathsf{rsft}} \leftarrow (\boldsymbol{a}, r_a, \boldsymbol{b}, r_b, (f^*_\varphi)_{\varphi \in \Phi^z_{\mathsf{rsft}}})$*, such that*
   (i) $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^n$*,* $r_a, r_b \in \mathbb{Z}_p$*,* $f^*_\varphi \in \mathbb{Z}_p$ *for* $\varphi \in \Phi^z_{\mathsf{rsft}}$*,*
   (ii) $(A, \hat{A}) = \mathcal{C}om(\mathsf{ck}; \boldsymbol{a}; r_a)$*,* $(B, \hat{B}) = \mathcal{C}om(\mathsf{ck}; \boldsymbol{b}; r_b)$*,*
   (iii) $\log_{g_2} \pi = \log_{g_2^\alpha} \pi^\beta = \sum_{\varphi \in \Phi^z_{\mathsf{rsft}}} f^*_\varphi \cdot \varphi(\sigma)$*, and*
   (iv) $(a_n, \dots, a_1) \neq (0, \dots, 0, b_n, \dots, b_{z+1})$*.*

The communication (argument size) of the right shift argument from [21] is 2 elements from $\mathbb{G}_2$. The prover's computation is dominated by $\Theta(n)$ multiplications in $\mathbb{Z}_p$ and two $(n+2)$-wide multi-exponentiations. The verifier's computation is dominated by 4 bilinear pairings. The CRS consists of $\Theta(n)$ group elements.

The left shift argument is very similar to the right shift argument, see [12,21].

3 New Permutation Argument

We now propose a new permutation argument that has $\Theta(n \log^2 n)$ prover's computation, as compared to $\Theta(n^2)$ in previous work. The main drawback of the new argument, compared to say permutation arguments from [16,19], is increased communication.

We will use parallelizable Beneš network [4] (see Sect. 2) to implement an arbitrary permutation argument. Beneš networks have been used before in the context of zero knowledge, see, e.g., [1,2,16]. In App. A, we explain why the approach of [16] of using Clos networks does not give the gain in efficiency that is necessary for our purposes, even if we use Beneš networks instead. Similar reasoning applies to the approach of [1,2]. To simplify the presentation, we will assume from now on that n is a power of 2.

More precisely, we show how to verify all basic permutations of the same step of the Beneš network simultaneously. That is, for every step of the Beneš network, we construct a short argument that this step was performed correctly by the prover. We emphasize that we use the variant of the Beneš permutation from Fig. 1 (right).

In a nutshell, at every stage we show separately that the crossbars are implemented correctly and that the wiring is implemented correctly. As we explained in the introduction, at both substages, the inputs will either stay at their original position, or will be shifted up or down by a constant that only depends on the stage. Next, we will show that one can compute efficiently the indexes of elements that are not shifted at all, shifted up, or shifted down. (In the crossbar substage, the prover and the verifier have to both use a routing algorithm for this.) Given corresponding index vectors (that we call masks), one can then show

— by using a small number of product and shift arguments — the correctness of each substage.

Consider the crossbar substage of the jth stage of the Beneš network, where we start counting with $j = 0$. Clearly, after the crossbar, the pair of elements indexed by $(2i, 2i + 1)$ will now be indexed by either $(2i, 2i + 1)$ or $(2i + 1, 2i)$, depending on whether the ith crossbar $C_{j,i}$ of this stage implements id or flip.

We prove the following technical lemma. As usually, for a predicate $P(X)$, let $[P(X)] = 1$ if $P(X)$ is true, and $[P(x)] = 0$ if $P(X)$ is false.

Lemma 1. *After following the crossbars, the vector* $\boldsymbol{a_j}$ *of intermediate values of all wires will change to a vector* $\boldsymbol{b_{j+1}}$*, where* $\boldsymbol{b_{j+1}} := (\boldsymbol{1} - \boldsymbol{m^{c,\ell}_{j+1}} - \boldsymbol{m^{c,r}_{j+1}}) \circ \boldsymbol{a_j} + \boldsymbol{m^{c,\ell}_{j+1}} \circ \mathsf{lsft}_1(\boldsymbol{a_j}) + \boldsymbol{m^{c,r}_{j+1}} \circ \mathsf{rsft}_1(\boldsymbol{a_j})$*. Here,*

$$\begin{aligned} m^{c,\ell}_{j+1,2i} := [C_{ji} = \mathsf{flip}] \ , \quad m^{c,r}_{j+1,2i+1} := [C_{ji} = \mathsf{flip}] \ , \\ m^{c,\ell}_{j+1,2i+1} := 0 \ , \quad m^{c,r}_{j+1,2i} := 0 \ . \end{aligned} \tag{1}$$

Proof. Denote $\boldsymbol{\ell^c_{j+1}} \leftarrow \mathsf{lsft}_1(\boldsymbol{a_j})$ and $\boldsymbol{r^c_{j+1}} \leftarrow \mathsf{rsft}_1(\boldsymbol{a_j})$. Clearly, the pair $(b_{j+1,2i}, b_{j+1,2i+1})$ depends only on $(a_{j,2i}, a_{j,2i+1})$ and C_{ji}. More precisely, if $C_{j+1} = \mathsf{id}$, then $(b_{j+1,2i}, b_{j+1,2i+1}) = (a_{j,2i}, a_{j,2i+1})$, and if $C_{j+1} = \mathsf{flip}$, then $(b_{j+1,2i}, b_{j+1,2i+1}) = (a_{j,2i+1}, a_{j,2i})$. Thus, $b_{j+1,2i}$ obtains the value $a_{j,2i}$ if $C_{ij} = \mathsf{id}$, and the value $\ell^c_{j+1,2i} = a_{j,2i+1}$ otherwise. Analogously, $b_{j+1,2i+1}$ obtains the value $a_{j,2i+1}$ if $C_{ij} = \mathsf{id}$, and the value $r^c_{j+1,2i} = a_{j,2i}$ otherwise. Thus,

$$b_{j+1,2i} = a_{j,2i} \cdot [C_{ij} = \mathsf{id}] + \ell^c_{j+1,2i} \cdot [C_{ij} \neq \mathsf{id}] + r^c_{j+1,2i} \cdot 0$$

and

$$b_{j+1,2i+1} = a_{j,2i+1} \cdot [C_{ij} = \mathsf{id}] + \ell^c_{j+1,2i+1} \cdot 0 + r^c_{j+1,2i+1} \cdot [C_{ij} \neq \mathsf{id}] \ .$$

Therefore, $\boldsymbol{b_{j+1}}$ can be expressed as in Eq. (1). □

Here, the values $\boldsymbol{m^{c,\ell}_{j+1}}$ and $\boldsymbol{m^{c,r}_{j+1}}$ are publicly known but ϱ-dependent. Thus, they have to be computed as part of the argument. Their computation takes time $\Theta(n \log n)$, by using a standard routing algorithm.

Let $z \leftarrow n/2^j$ for $j < \log_2 n$ and $z \leftarrow 2^j/n$ for $j \geq \log_2 n$. That is, $z \leftarrow 2^{|j - \log_2 n|}$. After additionally following the wiring substage, the new value of the value vector is equal to $\boldsymbol{a_{j+1}}$, where $a_{j+1,2i+1} = b_{j+1,2i+1}$, and, letting $\oplus$ to denote the bitwise XOR, $b_{j+1,2i \oplus z}$. This follows directly from the definition of the parallelizable Beneš network as given in Fig. 1, right. Thus, for some (public) masks $\boldsymbol{m^{w,\ell}_{j+1}}$ and $\boldsymbol{m^{w,r}_{j+1}}$,

$$\boldsymbol{a_{j+1}} := (\boldsymbol{1} - \boldsymbol{m^{w,\ell}_{j+1}} - \boldsymbol{m^{w,\ell}_{j+1}}) \circ \boldsymbol{b_{j+1}} + \boldsymbol{m^{w,\ell}_{j+1}} \circ \mathsf{lsft}_z(\boldsymbol{b_{j+1}}) + \boldsymbol{m^{w,r}_{j+1}} \circ \mathsf{rsft}_z(\boldsymbol{b_{j+1}}) \ .$$

The vectors $\boldsymbol{m^{w,\ell}_j}$ and $\boldsymbol{m^{w,r}_j}$ are both publicly known and not dependent on ϱ, therefore they can be precomputed (if necessary, as a part of the CRS). We note that since there are no wirings at the last stage, we will just have $\boldsymbol{a_{2\log_2 n}} = \boldsymbol{b_{2\log_2 n}}$.

Let $\mathsf{gk} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathsf{genbp}(1^\kappa, n)$, $g_1 \leftarrow \mathbb{G}_1 \setminus \{1\}$, $g_2 \leftarrow \mathbb{G}_2 \setminus \{1\}$; Let $\sigma \leftarrow \mathbb{Z}_p$; Generate the CRS of the following basic arguments separately, except that use the same values $(\mathsf{gk}, \sigma, \alpha)$ in all of them:

(i) the product argument,
(ii) the 2^i-left shift argument for $i < \log_2 n$,
(iii) the 2^i-right shift argument for $i < \log_2 n$.

The CRS of the permutation argument is equal to the union of all basic arguments.

Algorithm 1: CRS generation on input $(1^\kappa, n)$

Based on the explanation above, we now construct the permutation argument $\varrho([\![(B, \hat{B})]\!]) = [\![(A, \hat{A})]\!]$, where $(B, \hat{B}) = (A_0, \hat{A}_0)$ and $(A, \hat{A}) = (B_{2\log_2 n}, \hat{B}_{2\log_2 n})$. Note that we can remove $(B_{2\log_2 n}, \hat{B}_{2\log_2 n})$ from π^ϱ. See Prot. 1, Prot. 2 and Prot. 3. Here, the prover commits to all values $\boldsymbol{a_j}$ and $\boldsymbol{b_j}$, and then shows, by using intermediate masks $m_j^{\cdot;\cdot}$ and left and right shifts, that $\boldsymbol{b_{j+1}}$ is correctly computed from $\boldsymbol{a_j}$ and that $\boldsymbol{a_{j+1}}$ is correctly computed from $\boldsymbol{a_{j+1}}$. The (weak) soundness of the argument follows directly from the (weak) soundness of the product and permutation argument. The concrete computational assumption is used to individually guarantee the (weak) soundness of the product argument and 2^i-left and 2^i-right shift arguments for $i < \log_n$.

Theorem 3. *Let $\mathcal{C}om$ be the interpolating commitment scheme.*

(1) The new permutation argument is perfectly complete and perfectly witness-indistinguishable.

(2) Let Φ_z^{rsft} be as in Thm. 2, Φ_z^{lsft} be as in Sect. 2, and $\Phi^{\mathsf{perm}} := \bigcup_{i=0}^{\log_2 n - 1} \Phi_{2^i}^{\mathsf{lsft}} \cup \bigcup_{i=0}^{\log_2 n - 1} \Phi_{2^i}^{\mathsf{rsft}}$. If genbp satisfies an appropriate computational assumption, then a non-uniform PPT adversary against the new permutation argument has negligible chance, given a correctly formed CRS crs as an input, of outputting $inp^{\mathsf{perm}} \leftarrow (A, \tilde{A}, B, \tilde{B}, \varrho)$ and an accepting argument $\pi^{\mathsf{perm}} \leftarrow (\pi, \tilde{\pi})$ together with a witness $w^{\mathsf{perm}} \leftarrow (\boldsymbol{a}, r_a, \boldsymbol{b}, r_b, (f_\phi^*)_{\phi \in \Phi^{\mathsf{perm}}})$, such that

(i) $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_p^n$, $r_a, r_b \in \mathbb{Z}_p$, and $f_\phi^* \in \mathbb{Z}_p$ for $\phi \in \Phi^{\mathsf{perm}}$,

(ii) $(A, \tilde{A}) = \mathcal{C}om(\widetilde{\mathsf{ck}}; \boldsymbol{a}; r_a)$, $(B, \tilde{B}) = \mathcal{C}om(\widetilde{\mathsf{ck}}; \boldsymbol{b}; r_b)$, $\varrho \in S_n$,

(iii) $\log_{g_2} \pi = \log_{\tilde{g}_2} \tilde{\pi} = \sum_{\phi \in \Phi^{\mathsf{perm}}} f_\phi^* \cdot \phi(\sigma)$, and

(iv) $a_{\varrho(i)} \neq b_i$ for some $i \in [n]$.

Proof. Completeness follows from the previous discussion. Witness-indistinguishability follows from the fact that the argument π^{perm} is uniquely defined, given the witness.

Computational soundness. Given an adversary $\mathcal{A}$ who can break the soundness property, we construct an adversary $\mathcal{A}^*$ that either breaks either a computational or a knowledge assumption, as follows. For each stage, the adversary $\mathcal{A}^*$ uses the knowledge extractor to open the following commitments (the adversary also obtains the used randomizers, which we will not specify):

```
Let $(A_0, \hat{A}_0) = (A, \hat{A})$ be the input commitment;
Let $(B_{2\log_2 n}, \hat{B}_{2\log_2 n}) = (B, \hat{B})$ be the output commitment;
Use a routing algorithm to find out the value of $C_{ij}$ for every crossbar;
for $i \leftarrow 0$ to $2\log_2 n - 1$ do
    /* Handling crossbars */
    Construct a commitment $(B_{i+1}, \hat{B}_{i+1})$ to $\boldsymbol{b_{i+1}}$;
    Construct a commitment $(L^c_{i+1}, \hat{L}^c_{i+1})$ to $\boldsymbol{\ell^c_{i+1}} \leftarrow \mathsf{lsft}_1(\boldsymbol{a_i})$;
    Construct a commitment $(R^c_{i+1}, \hat{R}^c_{i+1})$ to $\boldsymbol{r^c_{i+1}} \leftarrow \mathsf{rsft}_1(\boldsymbol{a_i})$;
    Construct an argument $\pi_{i+1,1}$ that $\mathsf{lsft}_1([\![A_i]\!]) = [\![L^c_{i+1}]\!]$;
    Construct an argument $\pi_{i+1,2}$ that $\mathsf{rsft}_1([\![A_i]\!]) = [\![R^c_{i+1}]\!]$;
    Use Eq. (1) to construct valid masks $\boldsymbol{m^{c,\ell}_{i+1}}, \boldsymbol{m^{c,r}_{i+1}} \in \{0,1\}^n$;
    Construct a commitment $(D^{c,\ell}_{i+1}, \hat{D}^{c,\ell}_{i+1})$ to $\boldsymbol{d^{c,\ell}_{i+1}} \leftarrow \boldsymbol{m^{c,\ell}_{i+1}} \circ \boldsymbol{\ell^c_{i+1}}$;
    Construct a commitment $(D^{c,r}_{i+1}, \hat{D}^{c,r}_{i+1})$ to $\boldsymbol{d^{c,\ell}_{i+1}} \leftarrow \boldsymbol{m^{c,r}_{i+1}} \circ \boldsymbol{r^c_{i+1}}$;
    Construct a product argument $\pi_{i+1,3} \leftarrow [\![L^c_i]\!] \circ [\![\mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{m^{c,\ell}_{i+1}}; 0)]\!] = [\![D^{c,\ell}_{i+1}]\!]$;
    Construct a product argument $\pi_{i+1,4} \leftarrow [\![R^c_i]\!] \circ [\![\mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{m^{c,r}_{i+1}}; 0)]\!] = [\![D^{c,r}_{i+1}]\!]$;
    Construct a prod. arg. $\pi_{i+1,5}$ that
    $\boldsymbol{a_i} \circ (\boldsymbol{1} - \boldsymbol{m^{c,\ell}_{i+1}} - \boldsymbol{m^{c,r}_{i+1}}) = \boldsymbol{b_{i+1}} - \boldsymbol{d^{c,\ell}_{i+1}} - \boldsymbol{d^{c,r}_{i+1}}$;
    $\pi^c_{i+1} \leftarrow (B_{i+1}, \hat{B}_{i+1}, L^c_{i+1}, \hat{L}^c_{i+1}, R^c_{i+1}, \hat{R}^c_{i+1}, D^{c,\ell}_{i+1}, \hat{D}^{c,\ell}_{i+1}, D^{c,r}_{i+1}, \hat{D}^{c,r}_{i+1},$
        $(\pi_{i+1,j})_{j\in\{1,2\}}, (\pi_{i+1,j})_{j\in\{3,4,5\}})$;
    if $i < 2\log_2 n - 1$ then
        /* Handling wirings */
        Construct a commitment $(A_{i+1}, \hat{A}_{i+1})$ to $\boldsymbol{a_{i+1}}$ ;    /* Interim values
        after the wirings */
        if $i < \log_2 n$ then  $z \leftarrow n/2^i$ ;
        else  $z \leftarrow 2^i/n$ ;
        Construct a commitment $(L^w_{i+1}, \hat{L}^w_{i+1})$ of $\boldsymbol{\ell^w_{i+1}} = \mathsf{lsft}_z(\boldsymbol{b_{i+1}})$;
        Construct a commitment $(R^w_{i+1}, \hat{R}^w_{i+1})$ of $\boldsymbol{r^w_{i+1}} = \mathsf{rsft}_z(\boldsymbol{b_{i+1}})$;
        /* $\boldsymbol{m^{w,\ell}_{i+1}}$ and $\boldsymbol{m^{w,r}_{i+1}}$ can be part of CRS */
        Construct valid masks $\boldsymbol{m^{w,\ell}_{i+1}}, \boldsymbol{m^{w,r}_{i+1}} \in \{0,1\}^n$;
        Construct a shift argument $\pi_{i+1,6}$ for $\mathsf{lsft}_z([\![B_{i+1}]\!]) = [\![L^w_{i+1}]\!]$;
        Construct a shift argument $\pi_{i+1,7}$ for $\mathsf{rsft}_z([\![B_{i+1}]\!]) = [\![R^w_{i+1}]\!]$;
        Construct a commitment $(D^{w,\ell}_{i+1}, \hat{D}^{w,\ell}_{i+1})$ to $\boldsymbol{d^{w,\ell}_{i+1}} \leftarrow \boldsymbol{m^{w,\ell}_{i+1}} \circ \boldsymbol{\ell^w_{i+1}}$;
        Construct a commitment $(D^{r,\ell}_{i+1}, \hat{D}^{r,\ell}_{i+1})$ to $\boldsymbol{d^{w,\ell}_{i+1}} \leftarrow \boldsymbol{m^{w,r}_{i+1}} \circ \boldsymbol{r^w_{i+1}}$;
        Construct a product argument
        $\pi_{i+1,8} \leftarrow [\![L^w_i]\!] \circ [\![\mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{m^{w,\ell}_{i+1}}; 0)]\!] = [\![D^{w,\ell}_{i+1}]\!]$;
        Construct a product argument
        $\pi_{i+1,9} \leftarrow [\![R^w_i]\!] \circ [\![\mathcal{C}\mathrm{om}(\mathsf{ck}; \boldsymbol{m^{w,r}_{i+1}}; 0)]\!] = [\![D^{w,r}_{i+1}]\!]$;
        Construct a prod. arg. $\pi_{i+1,10}$ that
        $\boldsymbol{a_{i+1}} - \boldsymbol{d^{w,\ell}_{i+1}} - \boldsymbol{d^{w,r}_{i+1}} = (\boldsymbol{1} - \boldsymbol{m^{w,\ell}_{i+1}} - \boldsymbol{m^{w,r}_{i+1}}) \circ \boldsymbol{b_{i+1}}$;
        $\pi^w_{i+1} \leftarrow (A_{i+1}, \hat{A}_{i+1}, L^w_{i+1}, \hat{L}^w_{i+1}, R^w_{i+1}, \hat{R}^w_{i+1}, D^{w,\ell}_{i+1}, \hat{D}^{w,\ell}_{i+1}, D^{w,r}_{i+1}, \hat{D}^{w,r}_{i+1},$
            $(\pi_{i+1,6})_{j\in\{6,7\}}, (\pi_{i+1,6})_{j\in\{8,9,10\}})$;
    end
end
The argument is $\pi^{\varrho} \leftarrow ((\pi^c_{i+1}, \pi^w_{i+1})_{i=0}^{2\log_2 n-2}, \pi^c_{2\log_2 n})$;
```

Algorithm 2: Prover's algorithm on input $(A, \hat{A}; \boldsymbol{a}, r_a, \boldsymbol{b}, r_b)$

```
Use routing algorithm to compute $C_{ij}$ for all $i, j$;
for $i \leftarrow 0$ to $2\log_2 n - 1$ do
    Verify that $B_{i+1}, L^c_{i+1}, R^c_{i+1}, D^{c,\ell}_{i+1}, D^{c,r}_{i+1}$ are all group elements, and that their knowledge component is correctly formed;
    Construct valid masks $\boldsymbol{m^{c,\ell}_{i+1}}$ and $\boldsymbol{m^{c,r}_{i+1}}$ following Eq. (1);
    Compute valid masks $\boldsymbol{m^{w,\ell}_{i+1}}$ and $\boldsymbol{m^{w,r}_{i+1}}$;
    Verify 2 shift and 3 product arguments $\pi_{j+1,i}$ for $i \leq 5$;
    if $i < 2\log_2 n - 1$ then
        Verify that $A_{i+1}, L^w_{i+1}, R^w_{i+1}, D^{w,\ell}_{i+1}, D^{w,r}_{i+1}$ are all group elements, and that their knowledge component is correctly formed;
        Verify 2 shift and 3 product arguments $\pi_{j+1,i}$ for $i \geq 6$;
    end
end
```

Algorithm 3: Verifier's algorithm on input $(A, \hat{A}, B, \hat{B}; \pi^{\varrho})$

(a) $(B_{i+1}, \hat{B}_{i+1})$ to $\boldsymbol{b_{i+1}}$, (b) $(L^c_{i+1}, \hat{L}^c_{i+1})$ to $\boldsymbol{\ell^c_{i+1}}$, (c) $(R^c_{i+1}, \hat{R}^c_{i+1})$ to $\boldsymbol{r^c_{i+1}}$, (d) $(D^{c,\ell}_{i+1}, \hat{D}^{c,\ell}_{i+1})$ to $\boldsymbol{d^{c,\ell}_{i+1}}$, (e) $(D^{c,r}_{i+1}, \hat{D}^{c,r}_{i+1})$ to $\boldsymbol{d^{c,r}_{i+1}}$, (f) $(A_{i+1}, \hat{A}_{i+1})$ to $\boldsymbol{a_{i+1}}$, (g) $(L^w_{i+1}, \hat{L}^w_{i+1})$ to $\boldsymbol{\ell^w_{i+1}}$, (h) $(R^w_{i+1}, \hat{R}^w_{i+1})$ to $\boldsymbol{r^w_{i+1}}$, (i) $(D^{w,\ell}_{i+1}, \hat{D}^{w,\ell}_{i+1})$ to $\boldsymbol{d^{w,\ell}_{i+1}}$, (j) $(D^{w,r}_{i+1}, \hat{D}^{w,r}_{i+1})$ to $\boldsymbol{d^{w,r}_{i+1}}$. $\mathcal{A}^*$ verifies that all the openings were successful. If not (that is, $\mathcal{A}$ was *not* successful), she aborts.

Now, assume that $\mathcal{A}^*$ has returned $(inp^{\mathsf{perm}}, \pi^{\mathsf{perm}}, w^{\mathsf{perm}})$ that satisfy conditions (2i–2iv). Thus, in particular, $a_{\varrho(i)} \neq b_i$ for some $i \in [n]$. Since we obtained by using the extractor the intermediate values $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$, $\mathcal{A}^*$ verifies starting from $i = 0$ and ending with $i = 2\log_2 n - 1$ where is the first point where one of those two values was computed incorrectly.

W.l.o.g., assume that for some i_0, $\boldsymbol{a_i}$ and $\boldsymbol{b_i}$ have always been computed correctly for $i < i_0$, but $\boldsymbol{b_{i_0}}$ is computed incorrectly. That is, $\boldsymbol{b_{i_0}}$ is not equal to the expression given in Lem. 1. But it must be the case that all $\pi_{i_0,3}$, $\pi_{i_0,4}$, and $\pi_{i_0,5}$ verify while $\boldsymbol{b_{i_0}}$. Now we also have a case analysis depending on whether $\boldsymbol{d^{c,\ell}_{i_0}} = \boldsymbol{m^{c,\ell}}_{i_0} \cdot \boldsymbol{\ell^c_{i_0}}$ and/or $\boldsymbol{d^{c,r}_{i_0}} = \boldsymbol{m^{c,r}}_{i_0} \cdot \boldsymbol{r^c_{i_0}}$, or not. Suppose, w.l.o.g., that both equalities hold (the adversary $\mathcal{A}^*$ can check it, because due to the extractor she knows all the elements). In this case, $\boldsymbol{d^{c,\ell}_{i_0}}$ and $\boldsymbol{d^{c,r}_{i_0}}$ were correctly computed and $\pi_{i_0,5}$ verifies, but $(\boldsymbol{a_{i_0}} - \boldsymbol{d}^{c,\ell}_{i_0} - \boldsymbol{d}^{c,r}_{i_0}) \neq (1 - \boldsymbol{d}^{c,\ell}_{i_0} - \boldsymbol{d}^{c,r}_{i_0}) \circ \boldsymbol{a_{i_0}}$. But then by contradicting Thm. 1, the adversary has broken an appropriate computational assumption.

By an analogous case analysis, we obtain that the permutation argument is weakly sound, unless the weak soundness (and then the corresponding computational assumption) of one of the underlying product or shift arguments is broken. □

Efficiency. As seen from Prot. 2 and Prot. 3, every step can be done by using a small number of commitments, product, and rotation arguments. Thus, the computational complexity of the whole permutation argument will be dominated by $\Theta(\log n)$ more basic product and rotation arguments. Since the product argument can be computed in $\Theta(n \log n)$ non-cryptographic operations and $\Theta(n)$ cryptographic operations, then the new permutation argument can be computed in $\Theta(n \log^2 n)$ non-cryptographic operations and $\Theta(n \log n)$ cryptographic operations. This improves on prover's computation of $\Theta(n^2)$ exponentiations in [16].

The verifier has to perform $\Theta(\log n)$ pairings in total. Moreover, the verifier has to run a routing algorithm to establish the values C_{ij}, which takes $\Theta(n \log n)$ *non-cryptographic* operations. This is fine, since in all known applications of the permutation argument, the verifier's computational complexity is $\Omega(n)$ (much more expensive) cryptographic operations.

We also note that the second component of all product arguments used in Prot. 2 is a commitment to the masks. Both the prover and the verifier know the masks, and thus the corresponding commitments do not have to be transferred. On the other hand, both parties also have to compute these commitments, and this takes $\Theta(n \log n)$ multiplications in total.

The CRS length will be $\Theta(n \log n)$ group elements because all $\Theta(\log n)$ different shift arguments have to use a linear-length (and different) CRS.

4 CircuitSAT Argument

As shown in [16], one can construct a CircuitSAT argument out of $\Theta(1)$ product and permutation arguments. Since Groth's product and permutation arguments had quadratic prover's computation complexity, so did his CircuitSAT argument. By using the product argument proposed in [21] and the permutation argument of this paper, we end up with a CircuitSAT argument with prover's computation dominated by $\Theta(n \log^2 n)$ non-cryptographic operations and $\Theta(n \log n)$ cryptographic operations. As shown in [16,19], witness-indistinguishability and (weak) soundness of the underlying arguments is sufficient to prove zero-knowledge and soundness of the CircuitSAT argument. In fact, they can also be used to prove that the CircuitSAT argument is an argument of knowledge. Again, since we do not want to introduce overdependence on concrete basic arguments, also here we omit the precise description of the underlying assumptions; they can be derived directly from the assumptions behind the product and shift arguments.

Theorem 4. *There exists a perfectly complete and zero-knowledge* CircuitSAT *argument. Under certain q-type computational and knowledge assumptions, this argument is also computationally sound and an argument of knowledge.*

Proof. The CircuitSAT argument will be exactly the same as in [19], except that it uses the new permutation argument from Sect. 3 together with the product argument from [19]. The soundness of the CircuitSAT argument follows from q-type computational and knowledge assumptions, as in [19]. We also get that the CircuitSAT argument is an argument of knowledge, since during the soundness proof of [19], the adversary uses the knowledge extractor to extract the whole witness. □

Acknowledgements. The author was supported by Estonian Research Council and European Union through the European Regional Development Fund.

References

1. Abe, M.: Mix-Networks on Permutation Networks. In: Lam, K.Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 258–273. Springer, Heidelberg
2. Abe, M., Hoshino, F.: Remarks on Mix-Network Based on Permutation Networks. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 317–324. Springer, Heidelberg
3. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In: Canetti, R., Garay, J. (eds.) CRYPTO (2) 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg
4. Beneš, V.E.: Mathematical Theory of Connecting Networks and Telephone Traffic. Academic Press (Aug 28, 1965)
5. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct Non-interactive Arguments via Linear Interactive Proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg
6. Blelloch, G.: Vector Models for Data-Parallel Computing. MIT Press (1990)
7. Blum, M., Feldman, P., Micali, S.: Non-Interactive Zero-Knowledge and Its Applications. In: STOC 1988. pp. 103–112. ACM Press
8. Chaabouni, R., Lipmaa, H., Zhang, B.: A Non-Interactive Range Proof with Constant Communication. In: Keromytis, A. (ed.) FC 2012. LNCS, vol. 7397, pp. 179–199. Springer, Heidelberg
9. Clos, C.: A Study of Non-Blocking Switching Networks. Bell System Technical Journal 32(2), 406–424 (Mar 1953)
10. Damgård, I.: Towards Practical Public Key Systems Secure against Chosen Ciphertext Attacks. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 445–456. Springer, Heidelberg
11. Di Crescenzo, G., Lipmaa, H.: Succinct NP Proofs from an Extractability Assumption. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.) Computability in Europe, CIE 2008. LNCS, vol. 5028, pp. 175–185. Springer, Heidelberg
12. Fauzi, P., Lipmaa, H., Zhang, B.: Efficient Modular NIZK Arguments from Shift and Product. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 92–121. Springer, Heidelberg
13. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg
14. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof-Systems. In: Sedgewick, R. (ed.) STOC 1985. pp. 291–304. ACM Press
15. Golle, P., Jarecki, S., Mironov, I.: Cryptographic Primitives Enforcing Communication and Storage Complexity. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 120–135. Springer, Heidelberg

16. Groth, J.: Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg
17. Groth, J., Lu, S.: A Non-interactive Shuffle with Pairing Based Verifiability. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 51–67. Springer, Heidelberg
18. Hwang, F.K.M.: The Mathematical Theory of Nonblocking Switching Networks, Series on Applied Mathematics, vol. 15. World Scientific Publishing Co Pte Ltd, 2 edn. (Oct 1, 2004)
19. Lipmaa, H.: Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 169–189. Springer, Heidelberg
20. Lipmaa, H.: Succinct Non-Interactive Zero Knowledge Arguments from Span Programs and Linear Error-Correcting Codes. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013 (1). LNCS, vol. 8269, pp. 41–60. Springer, Heidelberg
21. Lipmaa, H.: Almost Optimal Short Adaptive Non-Interactive Zero Knowledge. Tech. Rep. 2014/396, International Association for Cryptologic Research (2014), available at `http://eprint.iacr.org/2014/396`
22. Lipmaa, H., Zhang, B.: A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. In: Visconti, I., Prisco, R.D. (eds.) SCN 2012. LNCS, vol. 7485, pp. 477–502. Springer, Heidelberg
23. Nassimi, D., Sahni, S.: Parallel Algorithms to Set Up the Benes Permutation Network. IEEE Trans. Computers 31(2), 148–154 (1982)
24. Opferman, D.C., Tsao-Wu, N.T.: On a Class of Rearrangeable Switching Networks. Part I: Control Algorithm. Bell System Technical Journal 50(5), 1579–1600 (May–June 1971)
25. Pippenger, N.: On the Evaluation of Powers and Monomials. SIAM J. Comput. 9(2), 230–250 (1980)
26. Pratt, V.R., Stockmeyer, L.J.: A Characterization of the Power of Vector Machines. Journal of Computer and System Sciences 12(2), 198–221 (1976)
27. Straus, E.G.: Addition Chains of Vectors. American Mathematical Monthly 70, 806–808 (1964)
28. Waksman, A.: A Permutation Network. Journal of the ACM 15(1), 159–163 (Jan 1968)

A On Groth's Use of Clos networks

One can implement a permutation argument by constructing an efficient rearrangeable interconnection network for that permutation, and then showing in zero knowledge that both the crossbars and the wiring steps of the network are followed correctly. In particular, Groth [16] used an (m, m)-Clos network for $m \approx \sqrt{n}$, combined with permutation arguments (for permutations from S_m) to show the correctness of crossbars, and with so called dispersion arguments [16] to show the correctness of the wirings. Since Groth's dispersion argument is significantly more efficient than Groth's permutation argument (or the permutation argument that we will construct in this paper), we will only count the cost induced by the S_m-permutation arguments.

Assume that we use some multistage interconnection network that uses, w.l.o.g.[3], crossbars X_{mm} for some $m \mid n$, has n/m crossbars per stage, and k stages. To implement a single crossbar (i.e., permutation from S_m) by using Groth's permutation argument, we need a CRS of size $\Theta(m^2)$, prover's computation $\Theta(m^2)$, verifier's computation $\Theta(m)$, and communication $\Theta(1)$. Thus, when we implement all k stages of the network (and kn/m crosspoints), the prover's computation grows to $\Theta(k \cdot \frac{n}{m} \cdot m^2) = \Theta(kmn)$, the verifier's computation to $\Theta(k \cdot \frac{n}{m} \cdot m) = \Theta(kn)$, and the communication to $\Theta(kn/m)$. On the other hand, the CRS length stays at $\Theta(m^2)$.

For example, if one uses the Clos network with $m = \sqrt{n}$ as in [16], then the CRS/the communication/the prover's computation/the verifier's computation become $\Theta(n)/\Theta(n^{1.5})/\Theta(n)/\Theta(\sqrt{n})$. In the case of the Beneš network, with $m = 2$ and $k = \log_2 n$, the corresponding complexities will be $\Theta(1)$, $\Theta(n \log n)$, $\Theta(n \log n)$, and $\Theta(n \log n)$. (Thus, it has the same computational and communication complexity as — interactive — shuffle by Abe [1,2], which is also based on the Beneš network.) The new method (as described below) provides a significant optimization over Groth's way of using interconnection networks in a permutation argument.

B Committed Shuffle Argument

B.1 Brief Idea

In a shuffle argument, the prover proves that two vectors of ciphertexts encrypt the same (unordered) multiset of plaintexts. Until now, only two efficient NIZK shuffle arguments have been published [17,22]. The argument of [17] is based on the Groth-Sahai proofs. The more efficient shuffle argument of [22] first uses two new basic arguments (zero argument and 1-sparsity argument), related to the techniques [16,19], to show that two commitments commit to the same (unordered) multiset of plaintexts. We call it the committed shuffle argument. After that, they use another argument to show that the same elements that were committed to were also encrypted by a ciphertext vector. We call it the commitment-ciphertext consistency argument. In [22], both arguments take linear computation and linear communication.

By using Beneš networks as in the case of the permutation argument, we implement the committed shuffle argument by using quasilinear CRS, prover's computation $\Theta(r_3^{-1}(n) \log r_3^{-1}(n) \cdot \log n)$, logarithmic verifier's computation, and logarithmic communication. The main additional technical challenge here, compared to the permutation argument, is that one has to prove in zero knowledge that the committed masks are correctly formed. We show that the latter can be done efficiently. The new committed shuffle argument is, up to our knowledge,

[3] One can use crossbars of different size, but the complexity is dominated by the largest crossbar in use. Moreover, all interesting multistage interconnection networks contain crossbars of the same size.

the first committed shuffle argument that requires only polylogarithmic communication. On the other hand, it requires larger prover's computation than the shuffle arguments of [17,22]. Alternatively, by using Groth's balancing technique, one can achieve sublinear CRS and communication at the same time.

On top of this, we can use the commitment-ciphertext consistency argument of [22]. The resulting shuffle argument will have smaller communication and verifier's computation than the shuffle argument from [22], but at the same time it will have higher prover's computation. We note that Beneš networks have been used before to construct shuffle arguments but with significantly larger communication $\Theta(n \log n)$, see for example [1,2].

B.2 Construction

The main difference with the permutation argument is that in the committed shuffle argument, the permutation should stay unknown to the verifier. We can still use Beneš networks, but in this case the masks $\boldsymbol{m}_{\boldsymbol{i+1}}^{\boldsymbol{c,r}}$ and $\boldsymbol{m}_{\boldsymbol{i+1}}^{\boldsymbol{c,\ell}}$ are not public, but they are committed to. The prover has to show that both masks belong to the set of allowed masks. The rest of the argument is pretty much the same as the permutation argument.

More precisely, to show that $\boldsymbol{m}_{\boldsymbol{i+1}}^{\boldsymbol{c,r}}$ and $\boldsymbol{m}_{\boldsymbol{i+1}}^{\boldsymbol{c,\ell}}$ are both valid, the prover proves that $\boldsymbol{m}_{\boldsymbol{i+1}}^{\boldsymbol{c,r}}$ and $\boldsymbol{m}_{\boldsymbol{i+1}}^{\boldsymbol{c,\ell}}$ satisfy Eq. (1). This can be done by showing that (here, $*$ denotes an arbitrary value) (i) $\boldsymbol{m}_{\boldsymbol{j+1}}^{\boldsymbol{c,\ell}} = (*, 0, *, 0, \ldots, *, 0)$ by using restriction argument [16], (ii) $\boldsymbol{m}_{\boldsymbol{j+1}}^{\boldsymbol{c,r}} = (0, *, 0, *, \ldots, 0, *)$ by using restriction argument [16], (iii) $\boldsymbol{m}_{\boldsymbol{j+1}}^{\boldsymbol{c,\ell}}$ is Boolean by using a product argument [16,19,12], (iv) $\boldsymbol{m}_{\boldsymbol{j+1}}^{\boldsymbol{r,\ell}} = \mathsf{lsft}_1(\boldsymbol{m}_{\boldsymbol{j+1}}^{\boldsymbol{c,\ell}})$ by using shift argument [12]. Thus, one has to do 2 restriction, 1 product and 1 shift argument. One can construct similar mask correctness arguments for any value of j. The committed shuffle argument itself takes $\Theta(\log n)$ group elements, as in the case of the permutation argument. Moreover, here the verifier does not have to execute the $\Theta(n \log n)$-time routing algorithm, and thus the verifier's computation is dominated by $\Theta(\log n)$ pairings.

Related Work. Abe and Hoshino [1,2] proposed a shuffle, where the underlying permutation is implemented by a Beneš network. Since there the correctness of every crossbar is verified individually, the resulting shuffle consists of $\Theta(n \log n)$ smaller zero-knowledge proofs and is thus less efficient than the new argument.