Efficient permutation protocol for MPC in the head*

Peeter Laud

Cybernetica AS
peeter.laud@cyber.ee

Abstract. The *MPC-in-the-head* construction (Ishai et al., STOC'07) give zero-knowledge proofs from secure multiparty computation (MPC) protocols. This paper presents an efficient MPC protocol for permuting a vector of values, making use of the relaxed communication model that can be handled by the MPC-in-the-head construction. Our construction allows more efficient ZK proofs for relations expressed in the Random Access Machine (RAM) model. As a standalone application of our construction, we present batch anonymizable ring signatures.

1 Introduction

Zero-knowledge proofs (ZKP) are cryptographic protocols that allow one party — the Prover — to convince another party — they Verifier — in the correctness of a statement, with the Verifier learning nothing besides the fact that the statement holds. The language of statements and their truth values are given in terms of a specified relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$. A statement is some $x \in \{0,1\}^*$, known both to the Prover and the Verifier. The Prover attempts to convince the Verifier that there exists some w (or: the Prover knows some w), such that $(x, w) \in R$.

There exist different techniques for turning the description of the relation R into a ZKP, based on various kinds of interactive proofs, or different secure multiparty computation techniques. This work best when R is represented as an arithmetic circuit, or a boolean circuit. The translation is less straightforward when R is represented as a computation in the Random Access Memory (RAM) model. In this case, if the whole computation is not translated first into a circuit (which has its own overheads), one will separately translate the behaviour of the processing unit, and the behaviour of the memory. These two behaviours have to be related to each other, and this requires showing that the load- and store-commands read and write the same values at both sides. Showing the equality

* This research has been funded by the Defense Advanced Research Projects Agency (DARPA) under contract HR0011-20-C-0083. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This research has also been supported by European Regional Development Fund through the Estonian Centre of Excellence in ICT Research (EXITE).

of loaded and stored values requires us to sort these actions by the memory addresses; in ZKP, this amounts to a permutation, and to a sortedness check.

A universal representation for permutations works by fixing a routing network [34, 4], and giving the bits that state how each switching element must route its two incoming values. This representation is equally well usable with any ZKP technique. If there have been n memory operations in the program, then the size of the routing network is $O(n \log n)$. If n is close to the total number of operations needed to express the relation R, then the size of the routing network may be the dominant component in the complexity of the ZK proofs for R.

MPC-in-the-head [23] is a ZKP technique that internally makes use of secure multiparty computation protocols. Compared to other techniques, it has good running time for the Prover, a decent running time for the Verifier, but longer proofs. Nevertheless, there are a number of ZK proof systems built upon this technique [17, 9, 1]. The MPC-in-the-head technique is also expected to compose well with other ZKP techniques.

In this paper, we propose a $O(n)$-complexity MPC-in-the-head based method to verify the correctness of the application of a permutation to a vector of values. Our method, which is basically a secure multiparty computation protocol for a communication model that fits into the MPC-in-the-head technique, can be composed with other protocols in the same communication model, hence bringing down the complexity of ZKP protocols for relations represented in the RAM model. We present our construction in Sec. 4, after discussing related work in Sec. 2 and giving the preliminaries in Sec. 3.

The main application of our construction would be in a ZK proof system, where it would support the encoding of interesting relations R. In this paper, we demonstrate a more stand-alone application of it — *batch anonymizable ring signatures*. In this setting there are a number of message digests and a number of public (verification) keys, and one party has a signature to each of the digests with respect to one of the keys. This party wants to prove that it has these signatures, but does not want to reveal them, nor does he want to reveal which digest has been signed with which key. If there are m keys and ℓ digests, then the existing techniques allow such proof to be created with the complexity $O(m\ell)$. We show how to bring the complexity down to $O(m + \ell)$. We present this construction in Sec. 5.

2 Related Work

Zero-knowledge proofs were first proposed in [20]. In this section, we cannot hope to give an overview of all the advancements thereafter. Rather, we refer to the course notes [32] discussing interactive proofs and their zero-knowledge variants.

The MPC-in-the-head construction was proposed in [23, 24]. A number of ZK proof systems have been built on top of this construction [17, 9, 1, 25].

Permutations in ZK proofs and MPC protocols have received their share of attention, and so have the means of connecting the processing unit and the memory unit in encoding RAM-based computations in both ZK proofs and MPC protocols. Laur et al. [27] were among the first to propose a composable MPC protocol for secret sharing based protocols; Laud [26] built oblivious reading and writing operations on top of it. For garbled circuits, Zahur and Evans [35] proposed similar constructions. For ZK proofs, Ben-Sasson et al. [2] used routing networks to connect the processing unit and the memory unit in a RAM-based computation. Bootle et al. [6] lifted a technique by Neff [28] for verifying that two encrypted vectors are permutations of each other, into the encodings of relations of ZK proofs; this technique is interactive and requires the operations in the encoding of the relation to work over large fields. Making proofs of permutations in private fashion has also been an important component of electronic voting systems; an overview of such *cryptographic mix-nets* is given in [21].

3 Preliminaries

In this paper, $[n]$ denotes the set $\{1, \ldots, n\}$. We use bold font to denote vectors: $\boldsymbol{v} = (v_1, \ldots, v_n)$ is a vector of length n.

3.1 Secure multiparty computation

A secure multiparty computation (MPC) protocol allows n parties $P_1, \ldots, P_n$ to jointly evaluate a publicly-known function $f : (\{0,1\}^m)^n \to \{0,1\}^\ell$, where the i-th party supplies the i-th argument of the function. All parties learn the output. An n-party protocol Π_f is *passively secure against k parties*, if for any $i_1, \ldots, i_k$, the view of the coalition of parties $\{P_{i_1}, \ldots, P_{i_k}\}$ can be simulated, given the inputs $x_{i_1}, \ldots, x_{i_k}$ of these parties, as well as the output of the function.

Let $\mathbb{A} \subseteq \{0,1\}^*$ be a finite set. Let $\mathbb{A}_\perp = \mathbb{A} \cup \{\perp\}$, where $\perp$ denotes the absence of a value. A (n,k)-*secret sharing scheme* for $\mathbb{A}$ consists of a randomized algorithm $\mathsf{Share} : \mathbb{A} \to \mathbb{A}^n$ and a deterministic algorithm $\mathsf{Combine} : \mathbb{A}_\perp^n \to \mathbb{A}_\perp$, such that the output of Share, where restricted to at most k positions, is independent from the input, and, for all $x \in \mathbb{A}$, for all $(x_1, \ldots, x_n)$ that can be output by $\mathsf{Share}(x)$, and for all $(x'_1, \ldots, x'_n) \in \mathbb{A}_\perp$, where $x'_i \in \{x_i, \perp\}$ and the number of non-$\perp$ elements x'_i is at least $(k+1)$, we have $\mathsf{Combine}(x'_1, \ldots, x'_n) = x$.

A (n,k)-secret sharing scheme may be a significant component of n-party MPC protocols secure against k parties. In this case, the private values are held by secret-sharing them among the n parties. For operations with private values, one needs cryptographic protocols that take the shares of the inputs of the operation as the input, and return to the parties the shares of the output [19, 16]. Typically, the function f is given by an arithmetic circuit that implements it. The inputs and outputs of f, as well as the intermediate values computed in the circuit are elements of $\mathbb{A}$, which is required to be an algebraic structure, typically a ring (or, more strongly, a field). The inputs of the circuit are shared by the parties holding them. The operations in the circuit are addition and multiplication in the ring $\mathbb{A}$. The parties execute a protocol for each operation in the circuit, eventually obtaining the shares of the output value, which they all learn by running the $\mathsf{Combine}$-algorithm.

Given a value $v \in \mathbb{A}$ that is held in secret-shared form as part of a MPC protocol, we denote the sharing by $[\![v]\!]$, and the individual share of the i-th party by $[\![v]\!]_i$. The write-up $[\![w]\!] \leftarrow [\![u]\!] + [\![v]\!]$ denotes the execution of the protocol for addition by all the parties, where the inputs are the shares of u and v, and the output shares define the value of w. Similar write-up is used for other operations with secret-shared data.

A secret sharing scheme over a ring $\mathbb{A}$ is *linear* if the operations Share and $\mathsf{Combine}$ are linear between the rings $\mathbb{A}$ and $\mathbb{A}^n$ [31, 12]. In this case, the protocol for $[\![u]\!] + [\![v]\!]$ is just the addition of the corresponding shares of u and v by each party. Similarly, the protocol for $c \cdot [\![u]\!]$, where $c \in \mathbb{A}$ is public, requires each party to multiply its share with c. The protocol for $[\![u]\!] \cdot [\![v]\!]$ is more complex; its details depend on the details of the secret sharing scheme, and it requires communication among participants.

Passive security for MPC protocol sets is defined through the simulation paradigm [18]. The *view* of a party in a protocol consists of the inputs of this party, the randomness this party generates, and the messages this party receives from other parties; these values allow one to perform all computations of that party, in particular find the messages it sends to other parties, and the values it outputs at the end of the protocol. The protocol Π_f for n parties is passively secure against the coalition $P_{i_1}, \ldots, P_{i_k}$, if there exists an algorithm $\mathcal{S}$ (the simulator), such that for any $x_1, \ldots, x_n$, the joint view of $P_{i_1}, \ldots, P_{i_k}$ in Π_f, where the input of P_j is x_j, is indistinguishable from the output of $\mathcal{S}(x_{i_1}, \ldots, x_{i_k}, f(x_1, \ldots, x_n))$.

3.2 Honest-verifier zero-knowledge proofs

Let $R \subseteq \{0,1\}^* \times \{0,1\}^*$, which we also think of as a function $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. Write $L_R = \{x \in \{0,1\}^* \,|\, \exists w \in \{0,1\}^* : (x,w) \in R\}$. We assume that R is a *NP-relation*, i.e. the function R is polynomial-time computable, and there exists a polynomial p, such for all $x \in L_R$, there exists $w \in \{0,1\}^*$, such that $(x,w) \in R$ and $|w| \leq p(|x|)$.

A protocol Π_R is a *Σ-protocol* for a given NP-relation R, if it is a protocol between two parties P and V with the following properties

- **Structure:** both P and V receive $x \in \{0,1\}^*$ as input. P also receives $w \in \{0,1\}^*$ as input. P sends the first message α to V. V generates a random β (does not depend on x or α), and sends it to P as the second message. P sends the third message γ to V. V runs a check on x, α, β, γ and either accepts or rejects.
- **Completeness:** if $(x,w) \in R$, then V definitely accepts.
- **Special soundness:** there exists a number s, such that if the transcripts $(x, \alpha, \beta_i, \gamma_i)$ for $i \in [s]$ with mutually different β_i-s are all accepted by V, then a w satisfying $(x,w) \in R$ can be efficiently found from these transcripts.

– **Special honest-verifier zero-knowledge:** there exists a simulator that on input $x \in L_R$ and a random β, outputs α, γ, such that the distribution of $(x, \alpha, \beta, \gamma)$ is indistinguishable from the transcripts of the real protocol.

A Σ-protocol is an instance of honest-verifier zero-knowledge (HVZK) proofs of knowledge (PoK). It can be turned into a non-interactive ZK PoK using the Fiat-Shamir heuristic [15]. The same heuristic is usable if the protocol has more rounds, as long as all challenges from the verifier are freshly generated random numbers. In this paper, we only consider honest verifiers, as the heuristic is already usable for them.

Let $\mathbb{G}$ be a cyclic group of size p, generated by g, and having a hard Discrete Logarithm problem. Camenisch and Stadler [8] studied HVZK PoKs in the setting of showing equalities among the elements of such groups. They have introduced a language for specifying the knowledge that the Prover claims to have. In contemporary notation, a problem description in their language has the form $\mathrm{PK}\{(x_1, \ldots, x_k) \mid \mathcal{F}\}$, where $x_1, \ldots, x_k$ are variables taking values in $\mathbb{Z}_p$, and $\mathcal{F}$ is a monotone Boolean formula, the atoms of which are statements that either an arithmetic expression (over $\mathbb{Z}_p$) involving $x_1, \ldots, x_k$ is zero, or a product of elements of $\mathbb{G}$ with exponents being arithmetic expressions involving $x_1, \ldots, x_k$, is the unit element of the group. Beside $x_1, \ldots, x_k$, the atomic statements in $\mathcal{F}$ use elements of $\mathbb{Z}_p$ and $\mathbb{G}$ which are known to both the Prover and the Verifier. The problem $\mathrm{PK}\{(x_1, \ldots, x_k) \mid \mathcal{F}\}$ requests the Prover to convince the Verifier that he knows values $v_1, \ldots, v_k \in \mathbb{Z}_p$ for $x_1, \ldots, x_k$, which make $\mathcal{F}$ true. The simplest example of expressing Prover's knowledge is the statement of knowing the discrete logarithm of an element $h \in \mathbb{G}$; the write-up is $\mathrm{PK}\{(x) \mid g^x = h\}$. Camenisch and Stadler [8], and Camenisch et al. [7] have given a general mechanism for constructing HVZK PoKs for the problems stated in this language; these protocols are actually Σ-protocols. Given a PoK problem $\mathrm{PK}\{(x_1, \ldots, x_k) \mid \mathcal{F}\}$ and values $v_1, \ldots, v_k$ known by Prover which fulfill that problem statement, we denote the execution of the Camenisch-Stadler protocol for this PoK problem with these values by $\mathrm{PK}\{(x_1, \ldots, x_k) \mid \mathcal{F}\}(v_1, \ldots, v_k)$.

3.3 The IKOS construction

Fix n and m, as well as a secret-sharing scheme. For the relation R, $n \in \mathbb{N}$ and $x \in \{0,1\}^*$, define the function $f_R^x : (\{0,1\}^m)^n \to \{0,1\}$ by $f_R^x(w_1, \ldots, w_n) = R(x, \mathsf{Combine}(w_1, \ldots, w_n))$. Let $\Pi_{f_R^x}$ be a MPC protocol for f_R^x, passively secure against $k \geq 2$ parties. The IKOS construction [23] turns the family of protocols $\{\Pi_{f_R^x}\}_x$ into a Σ-protocol for the relation R, also assuming the existence of commitments. In this construction, the prover first secret shares w by $(w_1, \ldots, w_n) \leftarrow \mathsf{Share}(w)$. He executes the protocol $\Pi_{f_R^x}$ with inputs $w_1, \ldots, w_n$ "in his head", i.e. simulates the views of all n parties. The Prover then commits to the views of all parties, and sends the commitments to the Verifier. The latter randomly picks a set of indices $\{i_1, \ldots, i_k\}$. The Prover opens the views of the i_1-th, i_2-th, $\ldots$, i_k-th simulated party to the Verifier, who checks that the obtained output is 1, and the views of the simulated parties are consistent with each other.

A MPC protocol consists of two kinds of steps. In the first kind, a party performs local computations. In the second kind, two parties perform a particular two-party computation, the sending and receiving of a message, which we could denote as $(x, \perp) \mapsto (\perp, x)$. In the IKOS construction, the verifier checks the correctness of both kinds of steps for all simulated parties whose views have been opened.

The correctness checks for the second kind of steps are possible for those pairs of simulated parties that both have their views opened to the verifier. For verifying these steps, the actual two-party functionality being executed makes no difference; it may be more complex than sending and receiving a message. This was used in the ZKBoo [17] ZK proof system, where an n-party MPC-in-the-head protocol for evaluating arithmetic circuits over a finite ring $\mathbb{A}$ was proposed, with passive security against $(n-1)$ parties. The two-party functionality used by their protocol is *oblivious linear evaluation*, where the first party ("sender") inputs a pair of values $(x, r) \in \mathbb{A}$, the second party ("receiver") inputs a value $y \in \mathbb{A}$, the sender learns nothing, and the receiver learns $xy - r$.

In the ZKBoo MPC-in-the-head protocol, private values are additively shared, i.e. $v \in \mathbb{A}$ is represented as $[\![v]\!]$, where $[\![v]\!]_i$ are random elements of $\mathbb{A}$ subject to the condition $\sum_{i=1}^{n} [\![v]\!]_i = v$. For adding two private values, or multiplying a private value with a constant, each party performs that same operation with his shares. For multiplying private values $[\![u]\!]$ and $[\![v]\!]$, the parties execute the protocol in Alg. 1. We see that each pair of parties (P_i, P_j) runs an instance of oblivious linear evaluation in order to share between themselves the product $[\![u]\!]_i \cdot [\![v]\!]_j$.

Data: private values $[\![u]\!], [\![v]\!]$
Data: private value $[\![w]\!]$, such that $w = uv$
foreach $i, j \in [n]$, $i \neq j$ **do**
  P_i picks a random $r_{ij}^{(i)} \xleftarrow{\$} \mathbb{A}$
  P_i and P_j run oblivious linear evaluation, with
    P_i (sender) inputs $([\![u]\!]_i, r_{ij}^{(i)})$
    P_j (receiver) inputs $[\![v]\!]_j$
    Output of P_j is $r_{ij}^{(j)} \leftarrow [\![u]\!]_i \cdot [\![v]\!]_j - r_{ij}^{(i)}$
foreach $i \in [n]$ **do**
  P_i computes $[\![w]\!]_i \leftarrow [\![u]\!]_i \cdot [\![v]\!]_i + \sum_{\substack{1 \leq j \leq n \\ j \neq i}} (r_{ij}^{(i)} + r_{ji}^{(i)})$
Return $[\![w]\!]$

Algorithm 1: Multiplying two private values in ZKBoo

The protocol in Alg. 1, together with the protocols for adding private values and multiplying them with public constants, as well as protocols for secret-sharing an input value (the party doing the sharing generates a random element of $\mathbb{A}$ as the share of each party, subject to their sum being equal to the value to be shared), and recovering an output of the computation (all parties send their shares to all other parties; each party adds up the shares), is a n-party protocol passively secure against $(n-1)$ parties. Indeed, all messages a party receives, either during the sharing an input value, or as the receiver in an oblivious linear evaluation functionality, or during the recovery of outputs, are uniformly random elements of $\mathbb{A}$ (in case of output recovery, subject to their sum being equal to the actual output, which is given to the simulator), hence can be simulated as such. These values remain uniformly random if we combine the views of up to $(n-1)$ parties.

3.4 Motivation: simulating computations

Existing MPC protocols, and ZK proof protocols built on top of them, are suitable if the computed function f or the relation R is represented as an arithmetic circuit. In practice, such f and R are usually represented differently. They are usually given in a format executable by a computer, i.e. as programs in an imperative language, i.e. as programs for a *Random Access Machine (RAM)*. These programs can invoke storing and loading operations against memory, the cells of which are addressable with the elements of $\mathbb{A}$. These operations, and the memory structure are not easily converted into an arithmetic circuit.

For verifying that $R(x, w) = 1$, where R is given as a RAM program, one commonly splits the execution of R on the RAM into two parts, proves the correctness of execution separately, and then shows that the two parts are connected in the right manner [2]. The first part of execution is the *processing unit*; the proof shows that at each execution step, the instruction was decoded correctly, and the result of the instruction was correctly computed from its inputs. The second part of the execution is the *memory*; the proof shows that for each memory cell, the value read from it is the same that was written to it previously. The two parts have to be connected — the sequence of load- and store-commands has to be the same at both sides. The ZK proof must check that the same sequence appears at both parts.

At processor side, it is natural to order the sequence of load- and store-commands by timestamps. When verifying the correctness of the steps made by the processor, at each execution step we need to know what value was load from the memory, or what value was stored there (if any). At memory side, it is natural to order this sequence first by memory address, and then by timestamps. In this manner, it is easy to verify that for each memory cell, the value loaded from there was the same that was either stored there, or loaded from there the previous time the same cell was accessed. Hence we need to show that two sequences are permutations of each other. For added flexibility, we want to have the permutation as a separate object, because we may need to show that several sequences are related to each other through the same permutation.

4 Our construction

We will now present our permutation protocol, which can be used to for the permutation functionality in a linear secret sharing based protocol set implementing the ABB for MPC-in-the-head. Let S_m denote the group of permutations of m elements. Given a private representation of a permutation $\sigma \in S_m$, and a vector of shared values $[\![\boldsymbol{v}]\!] = ([\![v_1]\!], \ldots, [\![v_m]\!])$, where $v_i \in \mathbb{A}$, we want to have a protocol for obtaining $[\![\sigma(\boldsymbol{v})]\!] = ([\![v_{\sigma(1)}]\!], \ldots, [\![v_{\sigma(m)}]\!])$. If the protocol is executed by n parties, then we want it to be passively secure against a coalition of $(n-1)$ parties.

The permutation σ is part of the witness, hence the Prover has to secret-share it among the n simulated parties. We let the private representation of σ to be $[\![\sigma]\!] = ([\![\sigma]\!]_1, \ldots, [\![\sigma]\!]_n)$, where $[\![\sigma]\!]_i$ is a random permutation of m elements, subject to the constraint $\sigma = [\![\sigma]\!]_n \circ \cdots \circ [\![\sigma]\!]_1$. Assuming that additive sharing is used for the values $v \in \mathbb{A}$, the protocol for obtaining $[\![\sigma(\boldsymbol{v})]\!]$ from $[\![\sigma]\!]$ and $[\![\boldsymbol{v}]\!]$ is given in Alg. 2.

Data: private vector $[\![\boldsymbol{v}]\!]$, private permutation $[\![\sigma]\!]$
Result: private vector $[\![\boldsymbol{w}]\!]$, where $w_i = v_{\sigma(i)}$
$[\![\boldsymbol{w}^{(0)}]\!] \leftarrow [\![\boldsymbol{v}]\!]$
for $i = 1$ **to** n **do**
    foreach $j \in [n] \backslash \{i\}$ **do**
        P_i generates random $\boldsymbol{r}_{ij}^{(i)} \in \mathbb{A}^m$
        Parties P_i and P_j run the following two-party functionality:
            P_i inputs $[\![\sigma]\!]_i$ and $\boldsymbol{r}_{ij}^{(i)}$
            P_j inputs $[\![\boldsymbol{w}^{(i-1)}]\!]_j$
            P_i obtains nothing
            P_j obtains $\boldsymbol{r}_{ij}^{(j)} \leftarrow [\![\sigma]\!]_i([\![\boldsymbol{w}^{(i-1)}]\!]_j) - \boldsymbol{r}_{ij}^{(i)}$
                `/* Elementwise subtraction of vectors */`
        P_j defines $[\![\boldsymbol{w}^{(i)}]\!]_j \leftarrow \boldsymbol{r}_{ij}^{(j)}$
    P_i defines $[\![\boldsymbol{w}^{(i)}]\!]_i \leftarrow [\![\sigma]\!]_i([\![\boldsymbol{w}^{(i-1)}]\!]_i) + \sum_{\substack{1 \leq j \leq n \\ j \neq i}} \boldsymbol{r}_{ij}^{(i)}$
Return $[\![\boldsymbol{w}^{(n)}]\!]$

Algorithm 2: Private permutation PrivPerm

The protocol in Alg. 2 is secure (passively, against $(n-1)$ parties) for the same reason the protocol in Alg. 1 is secure — any message a party receives through the two-party functionality are uniform random vectors over $\mathbb{A}$. Indeed, each such message is masked with freshly generated randomness. The simulator can again just replace these messages with freshly, uniformly generated elements of $\mathbb{A}$. We note that the two-party functionality we use here is the same as the Permute+Share functionality in [10].

We also see that **the protocol works**. Indeed, after the i-th iteration, the vector $\boldsymbol{w}^{(i)}$ is equal to $[\![\sigma]\!]_i([\![\sigma]\!]_{i-1}(\cdots [\![\sigma]\!]_1(\boldsymbol{v}) \cdots))$. Its private representation is constructed by permuting the shares of the private vector $[\![\boldsymbol{w}^{(i-1)}]\!]$ with the permutation $[\![\sigma]\!]_i$. The permutation of the i-th share is held by the i-th party, while the permutation of the j-th share $(j \neq i)$ is shared between the i-th and j-th parties.

The communication complexity of the protocol, which affects the size of the ZK proofs, is $O(n^2m)$. Indeed, at each round (of which there are n), each party, except for one, receives a vector of length m of elements of $\mathbb{A}$. Note that for MPC-in-the-head protocols, the round complexity is irrelevant. According to the calculations in [17], the protocol is most efficient (considering the amount of communication from the prover to the verifier in order to obtain a sufficiently small soundess error) when n is minimized, i.e. $n = 3$.

5 Application: multiple ring signatures

For demonstrating the usefulness, consider the following application, which may itself be part of a larger system. There are two parties, let us call them Prover and Verifier. There are ℓ message digests $d_1, \dots, d_\ell$, known to both of them.

There are also $m \approx \ell$ public keys $Q_1, \dots, Q_m$ for verifying signatures, known both to Prover and Verifier. Neither of them knows the corresponding signing keys. The Prover knows signatures $S_1, \dots, S_\ell$, such that S_i is the signature of d_i by one of the keys $Q_1, \dots, Q_m$. The Prover wants to convince the verifier that he knows these signatures, but does not want to reveal, which digest is verifiable by which public key.

The standard tools in situations like this are *group signatures* [11] and *ring signatures* [29]. In the latter, the signer can pick a number of public keys of other entities, in addition to his own, such that the signature can be verified against this set of public keys (including the public key of the signer), but does not reveal, the owner of which public key created the signature. In an *anonymizable ring signature* [22], the signing and ring creation functionalities are separated; anyone can turn a "normal" signature into a ring signature, adding more public keys to the set, against which the verification is done. Blazy et al. [5] showed that anonymizable ring signatures can be built on top of Schnorr signatures [30]. Their construction is based on a ZK proof for disjunction, which is made non-interactive using the Fiat-Shamir heuristic. If there are m public keys in the ring, then the size of the signature is $O(m)$. If we have ℓ messages, then we need ℓ signatures like that, hence their total size is $O(m\ell)$.

Schnorr signatures in a group $\mathbb{G}$ of size p (where p is prime) with hard Discrete Logratihm problem are defined as follows. Let $g \in \mathbb{G}$ be a fixed generator of the group. A signing key is a random $k \in \mathbb{Z}_p$. The corresponding public key is $Q = g^k$. In the signing operation $\mathsf{sig}(k, d)$ for a message (digest) d, the signer picks a random $r \in \mathbb{Z}_p$, computes $e = H(g^r, d)$, and $s = r - ke$. Here $H : \mathbb{G} \times \{0,1\}^* \to \mathbb{Z}_p$ is a hash function, modeled as a random oracle. The signature is the pair (s, e). The verification $\mathsf{ver}(d, Q, (s, e))$ consists of checking that $e = H(g^s Q^e, d)$.

5.1 Construction

In Alg. 4 we give a protocol for the Prover to convince the Verifier that it has a signature for each of the digests, where each signature can be verified against one of the given public keys. The protocol can be made non-interactive using the Fiat-Shamir transform. In Alg. 4, the Prover will tell the verifier the blinded version Z_i of the public key Q_{τ_i} for which he knows a signature (s_i, e_i) that verifies against d_i. The blinding is done by $Z_i = Q_{\tau_i}^{\lambda_i}$, where λ_i is a random value. As the size of $\mathbb{G}$ is prime, Z_i is independent of Q_{τ_i} from the point of view of the Verifier. On the other hand, the Prover, knowing the relationship, can convince the Verifier that he knows a signature (s_i, e_i) wrt. Q_{τ_i}, as well as the conversion factor λ_i. During this conviction, the Prover sends the verifier the value X_i, which is equal to g^r for the random r that the signer used in the signing operation. Hence X_i does not depend on Q_{τ_i}, either, and opening it to the Verifier does not help the latter to find out the value of τ_i.

Data: public $m, \ell \in \mathbb{N}$; public vector $\boldsymbol{Q} \in \mathbb{G}^m$
Data: private vector $[\![\boldsymbol{\lambda}]\!]$, additively shared over $\mathbb{Z}_p$, where $|\boldsymbol{\lambda}| = \ell$
Data: private vector $[\![\boldsymbol{\tau}]\!]$, additively shared over $\mathbb{Z}_2^{\lceil \log(m+\ell) \rceil}$, where $|\boldsymbol{\tau}| = \ell$
Data: private permutation $[\![\sigma]\!]$, which would sort $(1, \ldots, m, \tau_1, \ldots, \tau_\ell) \in \mathbb{N}^{m+\ell}$
Result: Vector $\boldsymbol{Z}$ of length ℓ, where $Z_i = Q_{\tau_i}^{\lambda_i}$
Result: Validity check of σ

```
⟦A_1⟧ ← Share(Q_1)                      /* Shared multiplicatively over 𝔾 */
⟦v_1⟧ ← Share(1)                        /* Shared additively over ℤ_2^⌈log(m+ℓ)⌉ */
⟦t_1⟧ ← Share(0)                        /* Shared additively over ℤ_2 */
for i = 2 to m do
    ⟦A_i⟧ ← Share(Q_i · Q_{i−1}^{−1})
    ⟦v_i⟧ ← Share(i)
    ⟦t_i⟧ ← Share(0)
for i = m + 1 to m + ℓ do
    ⟦A_i⟧ ← Share(1)
    ⟦v_i⟧ ← ⟦τ_{i−m}⟧
    ⟦t_i⟧ ← Share(1)
⟦B⟧ ← PrivPerm(⟦A⟧, ⟦σ⟧)
⟦w⟧ ← PrivPerm(⟦v⟧, ⟦σ⟧)
⟦u⟧ ← PrivPerm(⟦t⟧, ⟦σ⟧)
foreach i ∈ [m + ℓ − 1] do
    ⟦y_i⟧ ← (⟦w_i⟧‖⟦u_i⟧) ≤? (⟦w_{i+1}⟧‖⟦u_{i+1}⟧)
⟦C_1⟧ ← ⟦B_1⟧
for i = 2 to m + ℓ do
    ⟦C_i⟧ ← ⟦C_{i−1}⟧ · ⟦B_i⟧
⟦D⟧ ← PrivPerm^{−1}(⟦C⟧, ⟦σ⟧)
foreach i ∈ [ℓ] do
    ⟦Z_i⟧ ← ⟦D_{m+i}⟧^⟦λ_i⟧              /* Using Alg. 1 */
Return Combine(⟦Z⟧), Combine(⟦y⟧)
```

Algorithm 3: Internal MPC-in-the-head protocol for batch anonymizable ring signatures

The public keys $Q_{\tau_1}, \ldots, Q_{\tau_\ell}$ are picked out and blinded by the MPC-in-the-head protocol in Alg. 3, and the correctness of these operations is verified

Data: P and V have $Q_1, \ldots, Q_m \in \mathbb{G}$ and $d_1, \ldots, d_\ell \in \{0,1\}^*$
Data: P has signatures (s_i, e_i) for $i \in [\ell]$
Result: V is convinced that P has a signature for each d_i, with respect to one of the public keys Q_j
P finds $\tau_1, \ldots, \tau_\ell$, such that $\mathsf{ver}(d_i, Q_{\tau_i}, (s_i, e_i))$ holds for each i
P finds σ, which (stably) sorts $(1, \ldots, m, \tau_1, \ldots, \tau_\ell)$
P generates a random vector $\boldsymbol{\lambda} \in \mathbb{Z}_p^\ell$
$P \to V$: $Z_i \leftarrow Q_{\tau_i}^{\lambda_i}$ for $i \in [\ell]$
P and V run the IKOS protocol, executing the following multiple times:
  P runs in his head the protocol in Alg. 3, with public inputs m, ℓ, $\boldsymbol{Q}$, and private inputs $[\![\boldsymbol{\tau}]\!]$, $[\![\sigma]\!]$, $[\![\boldsymbol{\lambda}]\!]$
  P commits to the views of simulated parties, V requests the opening of some of them
  V checks the consistency of views, and that the output is $Z_1, \ldots, Z_\ell$ and a vector of true-s
foreach $i \in [\ell]$ **do**
  $P \to V$: $X_i \leftarrow g^{s_i} Q_{\tau_i}^{e_i}$
  P and V run the protocol $\mathrm{PK}\{(s, v) \mid g^s \cdot (Z_i^{H(X_i, d_i)})^v = X_i\}(s_i, 1/\lambda_i)$

Algorithm 4: Verification of batch anonymizable ring signatures

through the IKOS construction. The picking out is done by the operations that compute the vectors $[\![\boldsymbol{A}]\!]$, $[\![\boldsymbol{B}]\!]$, $[\![\boldsymbol{C}]\!]$, and $[\![\boldsymbol{D}]\!]$; the key Q_{τ_i} is then given as D_{m+i}. This part of Alg. 3 is the same as the $\mathsf{performRead}$ algorithm by Laud [26], and we refer to this paper for detailed explanations. We use the permutation protocol $\mathsf{PrivPerm}$ given in Alg. 2. We also use the protocol $\mathsf{PrivPerm}^{-1}$, which permutes the given vector with the *inverse* of the given private permutation $[\![\sigma]\!]$. It works the same way as $\mathsf{PrivPerm}$, except that it uses $[\![\sigma]\!]_i^{-1}$ as the share of the i-th party, and the main loop in Alg. 2 considers the parties in reverse order.

At the same time, the protocol in Alg. 3 verifies that σ is indeed the sorting permutation. It constructs the vector $[\![\boldsymbol{v}]\!]$, applies σ to it, and verifies that the resulting vector is sorted. We also need σ to perform stable sorting, or at least ensure that two equal values, where the first one originates among the first m elements of $\boldsymbol{v}$, and the second one among the last ℓ elements, will not be swapped. We use the vector $\boldsymbol{u}$ to record, where the element originated from, and append the bit u_i to the value w_i as the least significant bit, in order to ensure that the values from the first part of $\boldsymbol{v}$ are considered smaller. In order to compute the results of comparison $[\![y_i]\!]$, we have to compare two private values that have been bitwise shared. We can use any digital comparator circuit for this purpose.

At the end of Alg. 3, the keys D_{m+i} are blinded by raising them to the powers λ_i. Due to the sharings we have chosen for the elements of $\mathbb{G}$, and for the exponents (which are elements of $\mathbb{Z}_p$), this operation can be performed with the help of the multiplication protocol in Alg. 1. Indeed, the distributive laws we need for the correctness of this protocol hold for the exponentiation ($(g_1 g_2)^x = g_1^x g_2^x$ and $g^{x_1+x_2} = g^{x_1} g^{x_2}$). The ability to choose the most suitable sharing scheme for each data item, considering the operations we want to do with them, is one of the signs of versatility of the MPC-in-the-head protocols. This is bolstered by the permutation protocol (Alg. 2) being agnostic with respect to the sharing scheme chosen for the elements of the vector it is applied onto.

The computation and communication complexity of our multiple ring signature protocol is $O(m+\ell)$, if we consider the group operations to have constant complexity. When we are claiming linear complexity in total, we are also assuming $\log(m+\ell)$ to be constant, and the operations with bit-strings of length $\log(m+\ell)$ to take constant time. We believe this to be a fair simplification, because this value is expected to be less than $\log p$, which characterizes the complexity of multiplication and exponentiation in the group $\mathbb{G}$. With such simplification, one instance of Alg. 3 requires $O(m+\ell)$ computation and produces a trace of the same length; this protocol has to be executed a constant number of times in order to reduce its soundness error to acceptable levels. The rest of Alg. 4 has linear complexity, too.

5.2 Security

We desire our protocol to have privacy and soundness. Both properties can be stated through standard game-based definitions between the challenger (or environment) C and the adversary A. Starting with privacy, we want to state that an honest-but-curious verifier is not able to find out, which keys were actually used to sign the message digests. This property can be easily stated as an indistinguishability-type game; we give it in Alg. 5. We see that here the adversary comes up with two possible assignments of messages digests to the key that will sign them, and has to guess which assignment was actually used. We require that the adversary follows the protocol in Alg. 4, because we only provide privacy against an honest verifier.

C generates a random bit b and m public-private key pairs
$(Q_1, q_1), \dots, (Q_m, q_m)$
$C \to A$: $Q_1, \dots, Q_m$
A comes up with digests $d_1, \dots, d_\ell$, and maps $\pi_0, \pi_1 : [\ell] \to [m]$
C constructs signatures $(s_i, e_i) \leftarrow \mathsf{sig}(q_{\pi_b(i)}, d_i)$
C (as Prover) and A (as honest Verifier) run Alg. 4
A comes up with a bit b^*. A *wins* if $b = b^*$.

Algorithm 5: Security game for privacy

Theorem 1. *In the privacy game in Alg. 5, the adversary's probability of winning is at most negligibly higher than* $1/2$.

Proof sketch. Except for the public keys $Q_1, \dots, Q_m$ themselves, the rest of the view of the Verifier is independent of them, and can be generated from just $Q_1, \dots, Q_m, d_1, \dots, d_\ell$. Indeed, each Z_i is a random element of $\mathbb{G}$. The IKOS protocol is zero-knowledge, as long as the MPC protocol in Alg. 3 provides privacy against a sufficient number of parties (e.g. all but one of them). The protocol indeed has this privacy property, because all its steps are simulatable and the simulations can be composed; the output of the protocol is already knows to the Verifier. The values X_i that the Verifier learns in Alg. 4 are actually the values g^{r_i} that were constructing during the signing of the digests d_i; these values are random elements of $\mathbb{G}$, independent of both the digest and the key. The final proofs of knowledge are Zero-knowledge as well.

The soundness property is somewhat more complex to state. Here the adversary will win if the challenger accepts the signatures, while there exists a digest that has not been signed with one of the public keys. For setting up the public keys and the signatures, there is the *preparation phase* that controls the signing operations. In this phase, the adversary can cause a new public key to be generated. Note that the adversary does not obtain the corresponding private key, because these were supposed to be unknown to the Prover. In the preparation phase, the adversary can also get signatures with the generated keys. Before calling Alg. 4, the adversary can prune down the set of keys that have been generated. While executing Alg. 4, the adversary may deviate from the protocol.

C initializes $\boldsymbol{Q} \leftarrow \mathsf{NIL}$, $\boldsymbol{q} \leftarrow \mathsf{NIL}$, $n \leftarrow 0$
while A *stays in preparation phase* **do**
  switch A *queries* C *with...* **do**
    case *"generate key"* **do**
      $n \leftarrow n + 1$
      C generates keypair (Q_n, q_n) `/* Appended to` $\boldsymbol{Q}$ `and` $\boldsymbol{q}$ `*/`
      C initializes $\mathcal{D}_n \leftarrow \emptyset$
      $C \rightarrow A$: Q_n
    case *"sign d with i-th key"* $(1 \leq i \leq n)$ **do**
      $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{d\}$
      $C \rightarrow A$: $(s, e) \leftarrow \mathsf{sig}(q_i, d)$
A comes up with $i_1, \ldots, i_n \in [m]$, and with digests $d_1, \ldots, d_\ell$
C and A redefine $\boldsymbol{Q} \leftarrow (Q_{i_1}, Q_{i_2}, \ldots, Q_{i_n})$
A (as Prover) and C (as Verifier) run Alg. 4
A wins if C accepts and $\{d_1, \ldots, d_\ell\} \not\subseteq \bigcup_{j=1}^{n} \mathcal{D}_{i_j}$

Algorithm 6: Security game for soundness

Theorem 2. *In the soundness game in Alg. 6, the adversary's probability of winning is negligible.*

Proof sketch. The steps of Alg. 4 make sure that the Prover actually knows a signature of each digest with respect to one of the public keys in $\boldsymbol{Q}$. The IKOS proof ensures that Z_i are related to Q_i (by permuting and blinding), and the final PoK-s in Alg. 4 will verify that there exist signatures with respect to Z_i as the public keys. Combining these two, gives the signatures with respect to Q_i-s.

It is indeed possible to construct a knowledge extractor, by rewinding the Prover in the final PoK-s, as well as after it has committed to the views of simulated parties. The extractability of the PoK-s allows the values of s_i and

λ_i to be extracted, while the rewinding in the IKOS protocol extracts σ. The values e_i are computed as $e_i = H(X_i, d_i)$.

6 Discussion

We have proposed a passively secure MPC-in-the-head protocol for permutation. More efficient constructions of ZK proofs from MPC-in-the-head protocols are known, if the underlying protocols with several parties are *actively* secure for at least a constant fraction of the parties. Existing efficient linear secret sharing based MPC protocols [3, 14] make use of homomorphic MACs, which are updated by each operation in the arithmetic circuit encoding the computation. It is unclear, what would be a suitable MAC for permutation, as it would have to have suitable homomorphic properties with respect to the application of that permutation to a vector of values.

There exist methods to turn passively secure protocols into actively secure protocols with the help of replication [13]. Most probably, these methods will not help in increasing the efficiency of the resulting ZK proof, compared to the use of the underlying passively secure protocol, because the IKOS technique would dismantle the passive-to-active construction.

Still, our construction will be useful in encoding the relations represented in the RAM model as ZK proofs built using the IKOS technique.

References

1. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Thuraisingham et al. [33], pages 2087–2104.
2. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013*, pages 401–414. ACM, 2013.
3. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.
4. Václav E Beneš. *Mathematical theory of connecting networks and telephone traffic.* Academic press, 1965.
5. Olivier Blazy, Xavier Bultel, and Pascal Lafourcade. Anonymizable ring signature without pairing. In Frédéric Cuppens, Lingyu Wang, Nora Cuppens-Boulahia, Nadia Tawbi, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 9th International Symposium, FPS 2016, Québec City, QC, Canada, October 24-25, 2016, Revised Selected Papers*, volume 10128 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 2016.

6. Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune K. Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I*, volume 11272 of *Lecture Notes in Computer Science*, pages 595–626. Springer, 2018.
7. Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the portability of generalized schnorr proofs. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 425–442. Springer, 2009.
8. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.
9. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Thuraisingham et al. [33], pages 1825–1842.
10. Melissa Chase, Esha Ghosh, and Oxana Poburinnaya. Secret-shared shuffle. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 342–372. Springer, 2020.
11. David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.
12. Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.
13. Ivan Damgård, Claudio Orlandi, and Mark Simkin. Yet another compiler for active security or: Efficient MPC over arbitrary rings. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 799–829. Springer, 2018.
14. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

15. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.
16. Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111. ACM, 1998.
17. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 1069–1083. USENIX Association, 2016.
18. Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications.* Cambridge University Press, 2004.
19. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.
20. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.
21. Thomas Haines and Johannes Müller. Sok: Techniques for verifiable mix nets. In *33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020*, pages 49–64. IEEE, 2020.
22. Fumitaka Hoshino, Tetsutaro Kobayashi, and Koutarou Suzuki. Anonymizable signature and its construction from pairings. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *Pairing-Based Cryptography - Pairing 2010 - 4th International Conference, Yamanaka Hot Spring, Japan, December 2010. Proceedings*, volume 6487 of *Lecture Notes in Computer Science*, pages 62–77. Springer, 2010.
23. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 21–30. ACM, 2007.
24. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009.
25. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 525–537. ACM, 2018.
26. Peeter Laud. Parallel oblivious array access for secure multiparty computation and privacy-preserving minimum spanning trees. *Proc. Priv. Enhancing Technol.*, 2015(2):188–205, 2015.
27. Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In Xuejia Lai, Jianying Zhou, and Hui Li, editors, *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*, volume 7001 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2011.

28. C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001*, pages 116–125. ACM, 2001.
29. Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.
30. Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, 1991.
31. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
32. Justin Thaler. Proofs, Arguments, and Zero-Knowledge, 2021. Course notes, http://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html.
33. Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017.
34. Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, 1968.
35. Samee Zahur and David Evans. Circuit structures for improving efficiency of security and privacy tools. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 493–507. IEEE Computer Society, 2013.