Efficient oblivious permutations for privacy-preserving computations

Peeter Laud
peeter.laud@cyber.ee

Joint Estonian-Latvian Theory Days, 08.05.2022

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 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 (EXCITE), and Estonian Research Council through grant PRG920.

Privacy-preserving computations

- Parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$
- A computation $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$ to be done
  - f is described in some executable formalism
- Want: a *secure multiparty computation* (MPC) protocol that allows f to be evaluated, and
  - No party, or *tolerable coalition* learns nothing beyond their inputs and outputs
    - Formalisation: a *simulator* constructing the coalition's view from their inputs and outputs
  - If an *tolerable coalition* misbehaves, then honest parties will get correct outputs
    - I.e. the outputs of honest parties are possible, given their inputs
  - (Tolerable coalitions for curiosity and misbehaviour, for privacy and integrity may be different)

Zero-knowledge proofs

- Two-party protocols between Prover and Verifier
- There is a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$
- Prover and Verifier both have $x \in \{0,1\}^*$. Prover has $w \in \{0,1\}^*$
  - x is called "instance". w is called "witness"
- Prover wants to convince Verifier that [he knows | there exists] w, such that $(x, w) \in R$
  - If [there is | Prover knows] no such w, then Verifier should reject
- Verifier should learn nothing about w, besides that $(x, w) \in R$

Computation f for zero-knowledge proofs

- $P_1 \equiv$ Prover. $P_2 \equiv$ Verifier
- $f((x, w), x')$ works as follows:
  1. Check that $x = x'$. If not, return $(0, 0)$
  2. Check that $(x, w) \in R$. If not, return $(0, 0)$
  3. Return $(1, 1)$

R vs cryptographic techniques for ZKP

- The cryptographic techniques expect R as an arithmetic or boolean circuit
  - Arithmetic circuit over a finite ring (typically a field) $\mathbb{F}$
    - Not every field works for every technique
  - Inputs: Instance: m elements of $\mathbb{F}$. Witness: n elements of $\mathbb{F}$
  - Operations: binary addition and multiplication; constants
  - Outputs: some wires of the circuit.
    - Say that $(x, w) \in R$, if all output wires carry the value 0
- Verifier takes the description of R, and the instance
- Prover takes the description of R, the instance, and the witness

MPC in the head: set-up

- $R \subseteq \{0,1\}^* \times \{0,1\}^*$, $x \in \{0,1\}^*$. Define $f^x_{R,n} : (\{0,1\}^*)^n \to \{0,1\}^n$:
  - On inputs $w_1, \dots, w_n$: let $w \leftarrow w_1 \oplus \cdots \oplus w_n$
  - Let $r \leftarrow R(x, w)$
  - Output $(r, r, \dots, r)$
- Let $\Pi^{x,\nu}_{R,n}$ be a n-party MPC protocol that
  - computes $f^x_{R,n}$
  - is secure against passive coalitions of size $\leq \nu$
    - ... where $\nu \geq 2$

MPC in the head

- Prover generates random $w_1, \ldots, w_n$, s.t. $w_1 \oplus \cdots \oplus w_n = w$
- Prover executes all n parties in $\Pi^{x,\nu}_{R,n}(w_1, \ldots, w_n)$
  - Let $view_i$ be the view of the i-th party in the execution
- Prover defines $C_i \leftarrow \mathsf{Commit}(view_i)$, sends $C_1, \ldots, C_n$ to Verifier
- Verifier randomly picks $i_1, \ldots, i_\nu \in \{1, \ldots, n\}$, sends them to Prover
- Prover sends $view_{i_1}, \ldots, view_{i_\nu}$ to Verifier
- Verifier checks that for all $j, j' \in \{i_1, \ldots, i_\nu\}$:
  - C_j is a commitment to $view_j$
  - The output of j-th party is 1
  - Computations of j-th party are done correctly
  - Messages sent by j-th party to j'-th party are the same at both ends

MPC in the head: security

Zero-knowledge

Verifier only sees the view of a tolerable coalition

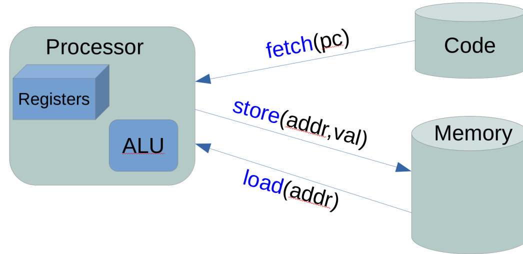
Soundness

- If $f^{x}_{R,n}(w_1,\dots,w_n)=0$, but $\Pi^{x,\nu}_{R,n}$ returns 1, then either
  - a protocol party did something wrong, or
  - a protocol message changed during the transmission
- Verifier has non-negligible chance of catching it
  - Run the whole ZK protocol repeatedly, such that the chance becomes overwhelming

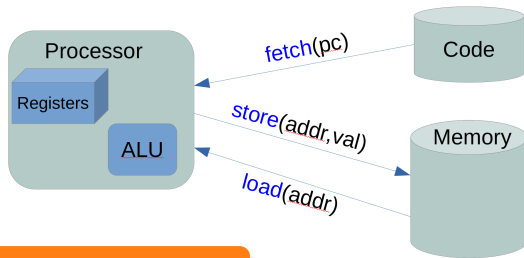
MPC in the head: usability

- Representation of R: must be supported by cryptographic techniques for $\Pi_{R,n}^{x,\nu}$
- In some popular instantiations (ZKBoo, ZKB++):
  - R is an *arithmetic circuit* over a *ring* $\mathbb{F}$
  - Several different rings (in the same circuit) can be supported

From a RAM program to R as a circuit



From a RAM program to R as a circuit

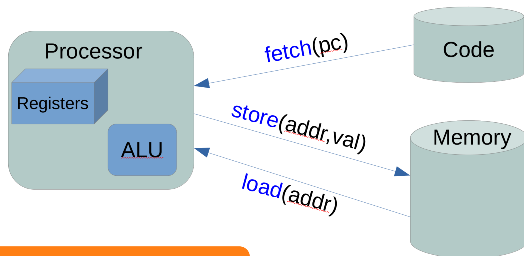


R states: for each time moment

- ALU computes correctly
- Registers' values are correct
- Correct store is generated

... if fetch and load work correctly

From a RAM program to R as a circuit



R states: for each time moment

- ALU computes correctly
- Registers' values are correct
- Correct store is generated

... if fetch and load work correctly

R states: for each time and address:

if a value is loaded from this address

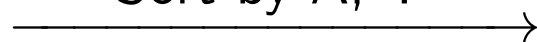
then it is equal to the most recent value stored at this address

loads and stores match

T	A	op	val
1	5	S	v_1
2	8	S	v_2
3	5	L	v_1
4	7	S	v_3
5	8	S	v_4
6	7	L	v_3
7	8	L	v_4
8	7	S	v_5
9	5	L	v_1
10	8	L	v_4
11	7	L	v_5

loads and stores match

T	A	op	val
1	5	S	v_1
2	8	S	v_2
3	5	L	v_1
4	7	S	v_3
5	8	S	v_4
6	7	L	v_3
7	8	L	v_4
8	7	S	v_5
9	5	L	v_1
10	8	L	v_4
11	7	L	v_5

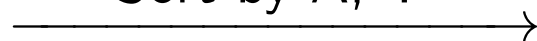


T	A	op	val
1	5	S	v_1
3	5	L	v_1
9	5	L	v_1
4	7	S	v_3
6	7	L	v_3
8	7	S	v_5
11	7	L	v_5
2	8	S	v_2
5	8	S	v_4
7	8	L	v_4
10	8	L	v_4

loads and stores match

T	A	op	val
1	5	S	v_1
2	8	S	v_2
3	5	L	v_1
4	7	S	v_3
5	8	S	v_4
6	7	L	v_3
7	8	L	v_4
8	7	S	v_5
9	5	L	v_1
10	8	L	v_4
11	7	L	v_5

Sort by A, T



T	A	op	val
1	5	S	v_1
3	5	L	v_1
9	5	L	v_1
4	7	S	v_3
6	7	L	v_3
8	7	S	v_5
11	7	L	v_5
2	8	S	v_2
5	8	S	v_4
7	8	L	v_4
10	8	L	v_4

R states:

For each row:

op = L

val = val_{prev}
and
A = A_{prev}

loads and stores match

T	A	op	val
1	5	S	v_1
2	8	S	v_2
3	5	L	v_1
4	7	S	v_3
5	8	S	v_4
6	7	L	v_3
7	8	L	v_4
8	7	S	v_5
9	5	L	v_1
10	8	L	v_4
11	7	L	v_5

Sort by A, T $\longrightarrow$

Witness w

Let it include the permutation π that sorts

Relation R

- applies π
- checks sortedness

T	A	op	val
1	5	S	v_1
3	5	L	v_1
9	5	L	v_1
4	7	S	v_3
6	7	L	v_3
8	7	S	v_5
11	7	L	v_5
2	8	S	v_2
5	8	S	v_4
7	8	L	v_4
10	8	L	v_4

R states:

For each row:

op = L

val = val_{prev} and A = A_{prev}

loads and stores match

T	A	op	val
1	5	S	v_1
2	8	S	v_2
3	5	L	v_1
4	7	S	v_3
5	8	S	v_4
6	7	L	v_3
7	8	L	v_4
8	7	S	v_5
9	5	L	v_1
10	8	L	v_4
11	7	L	v_5

Sort by A, T $\longrightarrow$

T	A	op	val
1	5	S	v_1
3	5	L	v_1
9	5	L	v_1
4	7	S	v_3
6	7	L	v_3
8	7	S	v_5
11	7	L	v_5
2	8	S	v_2
5	8	S	v_4
7	8	L	v_4
10	8	L	v_4

Witness w

Let it include the permutation π that sorts

Relation R

- **applies π**
- checks sortedness

R states:

For each row:

op = L

$\Downarrow$

val = val_{prev}

and

A = A_{prev}

Private values and permutations in $\Pi_{f,n}^{x,n-1}$

Additive sharing for $v \in \mathbb{F}$

$[\![v]\!] = ([\![v]\!]_1, \ldots, [\![v]\!]_n) \in \mathbb{F}^n$

- $[\![v]\!]_1, \ldots, [\![v]\!]_n$ are random elements of $\mathbb{F}$, such that $\sum_{i=1}^{n} [\![v]\!]_i = v$
- i-th party has $[\![v]\!]_i$

... sharing for $\pi \in S_m$

$[\![\pi]\!] = ([\![\pi]\!]_1, \ldots, [\![\pi]\!]_n) \in (S_m)^n$

- $[\![\pi]\!]_1, \ldots, [\![\pi]\!]_n$ are random permutations from S_m, such that $[\![\pi]\!]_n \circ \cdots \circ [\![\pi]\!]_1 = \pi$
- i-th party has $[\![\pi]\!]_i$

Two-party functionalities

In MPC in the head, Verifier had to check...

"Messages sent by j-th party to j'-th party are the same at both ends"
(for all pairs of parties (j, j') whose views were opened)

Send + receive

$(v, \bot) \mapsto (\bot, v)$

- Implementable over a network connection

Two-party functionalities

In MPC in the head, Verifier had to check...

"Messages sent by j-th party to j'-th party are the same at both ends"
(for all pairs of parties (j, j') whose views were opened)

Send + receive

$(v, \perp) \mapsto (\perp, v)$

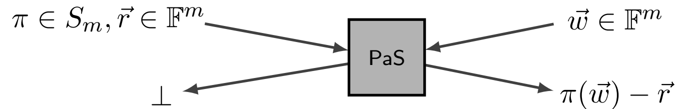
- Implementable over a network connection

Two-party functionality

$(\vec{x}, \vec{y}) \mapsto (f_1(\vec{x}, \vec{y}), f_2(\vec{x}, \vec{y}))$

- f_1, f_2 are arbitrary, deterministic functions
- Prover can run in his head
- Verifier can check that it was done correctly

Permute-and-Share



I.e. after running PaS, two parties have additively shared $\pi(\vec{w})$

Permute a *shared* vector with *private* permutation

Protocol

- There is $[\![\vec{v}]\!]$. Also, s-th party has π
- For each $i \in I = \{1, \ldots, n\} \setminus \{s\}$:
  - s-th party generates random $\vec{r}_i \in \mathbb{F}^m$
  - s-th and i-th party run PaS:
    - s-th party inputs π and $\vec{r}_i$
    - i-th party inputs $[\![\vec{v}]\!]_i$
    - i-th party receives $[\![\vec{w}]\!]_i$
- s-th party defines $[\![\vec{w}]\!]_s \leftarrow \pi([\![\vec{v}]\!]_s) + \sum_{i \in I} \vec{r}_i$
- $[\![\vec{w}]\!]$ is the result

Permute a *shared* vector with *private* permutation

Protocol

- There is $[\![\vec{v}]\!]$. Also, s-th party has π
- For each $i \in I = \{1, \ldots, n\} \setminus \{s\}$:
  - s-th party generates random $\vec{r}_i \in \mathbb{F}^m$
  - s-th and i-th party run PaS:
    - s-th party inputs π and $\vec{r}_i$
    - i-th party inputs $[\![\vec{v}]\!]_i$
    - i-th party receives $[\![\vec{w}]\!]_i$
- s-th party defines $[\![\vec{w}]\!]_s \leftarrow \pi([\![\vec{v}]\!]_s) + \sum_{i \in I} \vec{r}_i$
- $[\![\vec{w}]\!]$ is the result

Correctness

$[\![\vec{w}]\!]_i + \vec{r}_i = \pi([\![\vec{v}]\!]_i)$ for $i \in I$

CYBERNETICA

Permute a *shared* vector with *private* permutation

Protocol

- There is $[\![\vec{v}]\!]$. Also, s-th party has π
- For each $i \in I = \{1, \ldots, n\} \setminus \{s\}$:
  - s-th party generates random $\vec{r}_i \in \mathbb{F}^m$
  - s-th and i-th party run PaS:
    - s-th party inputs π and $\vec{r}_i$
    - i-th party inputs $[\![\vec{v}]\!]_i$
    - i-th party receives $[\![\vec{w}]\!]_i$
- s-th party defines $[\![\vec{w}]\!]_s \leftarrow \pi([\![\vec{v}]\!]_s) + \sum_{i \in I} \vec{r}_i$
- $[\![\vec{w}]\!]$ is the result

Correctness

$[\![\vec{w}]\!]_i + \vec{r}_i = \pi([\![\vec{v}]\!]_i)$ for $i \in I$

$[\![\vec{w}]\!] = \sum_{i=1}^{n} [\![\vec{w}]\!]_i$

Permute a *shared* vector with *private* permutation

Protocol

- There is $[\![\vec{v}]\!]$. Also, s-th party has π
- For each $i \in I = \{1, \dots, n\} \setminus \{s\}$:
  - s-th party generates random $\vec{r}_i \in \mathbb{F}^m$
  - s-th and i-th party run PaS:
    - s-th party inputs π and $\vec{r}_i$
    - i-th party inputs $[\![\vec{v}]\!]_i$
    - i-th party receives $[\![\vec{w}]\!]_i$
- s-th party defines $[\![\vec{w}]\!]_s \leftarrow \pi([\![\vec{v}]\!]_s) + \sum_{i \in I} \vec{r}_i$
- $[\![\vec{w}]\!]$ is the result

Correctness

$[\![\vec{w}]\!]_i + \vec{r}_i = \pi([\![\vec{v}]\!]_i)$ for $i \in I$
$[\![\vec{w}]\!] = \sum_{i=1}^{n} [\![\vec{w}]\!]_i = \sum_{i=1}^{n} \pi([\![\vec{v}]\!]_i)$

Permute a *shared* vector with *private* permutation

Protocol

- There is $[\![\vec{v}]\!]$. Also, s-th party has π
- For each $i \in I = \{1, \ldots, n\} \backslash \{s\}$:
  - s-th party generates random $\vec{r}_i \in \mathbb{F}^m$
  - s-th and i-th party run PaS:
    - s-th party inputs π and $\vec{r}_i$
    - i-th party inputs $[\![\vec{v}]\!]_i$
    - i-th party receives $[\![\vec{w}]\!]_i$
- s-th party defines $[\![\vec{w}]\!]_s \leftarrow \pi([\![\vec{v}]\!]_s) + \sum_{i \in I} \vec{r}_i$
- $[\![\vec{w}]\!]$ is the result

Correctness

$[\![\vec{w}]\!]_i + \vec{r}_i = \pi([\![\vec{v}]\!]_i)$ for $i \in I$

$[\![\vec{w}]\!] = \sum_{i=1}^{n} [\![\vec{w}]\!]_i = \sum_{i=1}^{n} \pi([\![\vec{v}]\!]_i) = \pi(\sum_{i=1}^{n} [\![\vec{v}]\!]_i) = \pi([\![\vec{v}]\!])$

Permute a *shared* vector with *private* permutation

Protocol

- There is $[\![\vec{v}]\!]$. Also, s-th party has π
- For each $i \in I = \{1, \ldots, n\} \setminus \{s\}$:
  - s-th party generates random $\vec{r}_i \in \mathbb{F}^m$
  - s-th and i-th party run PaS:
    - s-th party inputs π and $\vec{r}_i$
    - i-th party inputs $[\![\vec{v}]\!]_i$
    - i-th party receives $[\![\vec{w}]\!]_i$
- s-th party defines $[\![\vec{w}]\!]_s \leftarrow \pi([\![\vec{v}]\!]_s) + \sum_{i \in I} \vec{r}_i$
- $[\![\vec{w}]\!]$ is the result

Correctness

$[\![\vec{w}]\!]_i + \vec{r}_i = \pi([\![\vec{v}]\!]_i)$ for $i \in I$
$[\![\vec{w}]\!] = \sum_{i=1}^{n} [\![\vec{w}]\!]_i = \sum_{i=1}^{n} \pi([\![\vec{v}]\!]_i) = \pi(\sum_{i=1}^{n} [\![\vec{v}]\!]_i) = \pi([\![\vec{v}]\!])$

Privacy

- The coalition of everybody but the s-th party only learns random values
  - all outputs from PaS are masked with fresh randomnesses $\vec{r}_i$
- The s-th party learns nothing

Permute a *shared* vector with *shared* permutation

- Permute $[\![\vec{v}]\!]$ with $[\![\pi]\!]_1, \ldots, [\![\pi]\!]_n$, one after another
  - Each party will play the role of s-th party during their turn

Alternative approaches

- Permutation networks
  - Size $O(m \log m)$. Slower than our approach

Alternative approaches

- Permutation networks
  - Size $O(m \log m)$. Slower than our approach
- Checking an invariant for $\vec{v}, \vec{w} \in \mathbb{F}^m$

Alternative approaches

- Permutation networks
  - Size $O(m \log m)$. Slower than our approach
- Checking an invariant for $\vec{v}, \vec{w} \in \mathbb{F}^m$

Invariant polynomial

- Let $\vec{v} \in \mathbb{F}^m$. Define polynomial $p_{\vec{v}}(X) \in \mathbb{F}[X]$ as

$$p_{\vec{v}}(X) = \prod_{i=1}^{m} (X - v_i)$$

- $\vec{v}$ and $\vec{w}$ are permutations of each other, iff $p_{\vec{v}}(X) = p_{\vec{w}}(X)$

Alternative approaches

- Permutation networks
  - Size $O(m \log m)$. Slower than our approach
- Checking an invariant for $\vec{v}, \vec{w} \in \mathbb{F}^m$

Invariant polynomial

- Let $\vec{v} \in \mathbb{F}^m$. Define polynomial $p_{\vec{v}}(X) \in \mathbb{F}[X]$ as

$$p_{\vec{v}}(X) = \prod_{i=1}^{m} (X - v_i)$$

- $\vec{v}$ and $\vec{w}$ are permutations of each other, iff $p_{\vec{v}}(X) = p_{\vec{w}}(X)$
- If $|\mathbb{F}| \gg m$, then this equality check of polynomials can be done as follows:
  - Pick random $r \stackrel{\$}{\leftarrow} \mathbb{F}$. Check that $p_{\vec{v}}(r) = p_{\vec{w}}(r)$
  - Probability of false positive: $m/|\mathbb{F}|$

Alternative approaches

- Permutation networks
  - Size $O(m \log m)$. Slower than our approach
- Checking an invariant for $\vec{v}, \vec{w} \in \mathbb{F}^m$
  - Only if $\mathbb{F}$ is a sufficently large field
  - Introduces more communication rounds between Prover and Verifier
    - r has to be selected by Verifier

Invariant polynomial

- Let $\vec{v} \in \mathbb{F}^m$. Define polynomial $p_{\vec{v}}(X) \in \mathbb{F}[X]$ as

$$p_{\vec{v}}(X) = \prod_{i=1}^{m} (X - v_i)$$

- $\vec{v}$ and $\vec{w}$ are permutations of each other, iff $p_{\vec{v}}(X) = p_{\vec{w}}(X)$
- If $|\mathbb{F}| \gg m$, then this equality check of polynomials can be done as follows:
  - Pick random $r \stackrel{\$}{\leftarrow} \mathbb{F}$. Check that $p_{\vec{v}}(r) = p_{\vec{w}}(r)$
  - Probability of false positive: $m/|\mathbb{F}|$

SPDZ

- Secure multiparty computation protocol, based on secret sharing
- Uses additive secret sharing over a field $\mathbb{F}$
  - Executes arithmetic circuits over $\mathbb{F}$
- Tolerates dishonest majority
  - Up to $(n-1)$ parties out of n may be controlled by the adversary
- Gives *active security with abort*:
  - Active attacks give no information to attacker
  - Active attacks are detected, i.e. cannot change the outcome
    - The attacking party is not detected

SPDZ's verifiable secret sharing, and private operations

- i-th party has a private value $\alpha_i \in \mathbb{F}$
  - Denote $\alpha = \alpha_1 + \cdots + \alpha_n$

SPDZ's verifiable secret sharing, and private operations

- i-th party has a private value $\alpha_i \in \mathbb{F}$
  - Denote $\alpha = \alpha_1 + \cdots + \alpha_n$
- Private representation $[\![v]\!]$ of a value $v \in \mathbb{F}$ is the following:
  - i-th party privately holds $[\![v]\!]_i = ([v]_i, \langle v \rangle_i) \in \mathbb{F}^2$
  - $[v]_1 + \cdots + [v]_n = v$
  - $\langle v \rangle_1 + \cdots + \langle v \rangle_n = \alpha \cdot v$

SPDZ's verifiable secret sharing, and private operations

- i-th party has a private value $\alpha_i \in \mathbb{F}$
  - Denote $\alpha = \alpha_1 + \cdots + \alpha_n$
- Private representation $[\![v]\!]$ of a value $v \in \mathbb{F}$ is the following:
  - i-th party privately holds $[\![v]\!]_i = ([v]_i, \langle v \rangle_i) \in \mathbb{F}^2$
  - $[v]_1 + \cdots + [v]_n = v$
  - $\langle v \rangle_1 + \cdots + \langle v \rangle_n = \alpha \cdot v$
- Linear operations with private values are done locally by parties

SPDZ's verifiable secret sharing, and private operations

- i-th party has a private value $\alpha_i \in \mathbb{F}$
  - Denote $\alpha = \alpha_1 + \cdots + \alpha_n$
- Private representation $[\![v]\!]$ of a value $v \in \mathbb{F}$ is the following:
  - i-th party privately holds $[\![v]\!]_i = ([v]_i, \langle v \rangle_i) \in \mathbb{F}^2$
  - $[v]_1 + \cdots + [v]_n = v$
  - $\langle v \rangle_1 + \cdots + \langle v \rangle_n = \alpha \cdot v$
- Linear operations with private values are done locally by parties
- A private value can be *opened* to all parties, or to a single party
  - Inconsistencies are detected

SPDZ's verifiable secret sharing, and private operations

- i-th party has a private value $\alpha_i \in \mathbb{F}$
  - Denote $\alpha = \alpha_1 + \cdots + \alpha_n$
- Private representation $[\![v]\!]$ of a value $v \in \mathbb{F}$ is the following:
  - i-th party privately holds $[\![v]\!]_i = ([v]_i, \langle v \rangle_i) \in \mathbb{F}^2$
  - $[v]_1 + \cdots + [v]_n = v$
  - $\langle v \rangle_1 + \cdots + \langle v \rangle_n = \alpha \cdot v$
- Linear operations with private values are done locally by parties
- A private value can be *opened* to all parties, or to a single party
  - Inconsistencies are detected
- Multiplication triples ("Beaver triples") are used to multiply private values
  - Multiplication triples are generated during the *offline phase* of the protocol

Multiplication triples

$([\![a]\!], [\![b]\!], [\![c]\!])$, where $a, b \in \mathbb{F}$ are random and $c = ab$

Multiplication triples

$([\![a]\!], [\![b]\!], [\![c]\!])$, where $a, b \in \mathbb{F}$ are random and $c = ab$

Multiplying $[\![u]\!]$ and $[\![v]\!]$

- Compute $[\![u']\!] = [\![u]\!] - [\![a]\!]$ and $[\![v']\!] = [\![v]\!] - [\![b]\!]$
- *Open* u' and v' to all parties
  - This does not leak u, v, because they are masked with random a, b
- Result is $u' \cdot v' + u' \cdot [\![b]\!] + v' \cdot [\![a]\!] + [\![c]\!]$
- (Discard the multiplication triple)

Opening $[\![x]\!]$

- i-th party broadcasts $[x]_i$
  - They can be malicious
  - Let x'_i be the actually broadcast value
- Everybody computes $x' \leftarrow \sum_{i=1}^{n} x_i$
- i-th party computes $d_i \leftarrow \alpha_i \cdot x' - \langle x \rangle_i$
  - cheat with $d_i \equiv$ cheat with $\langle x \rangle_i$
- Parties *simultaneously* broadcast $d_1, \ldots, d_n$
  - Commit + open
- Check that $\sum_{i=1}^{n} d_i = 0$
- Accept $x = x'$

Opening $[\![x]\!]$

- i-th party broadcasts $[x]_i$
  - They can be malicious
  - Let x'_i be the actually broadcast value
- Everybody computes $x' \leftarrow \sum_{i=1}^{n} x_i$
- i-th party computes $d_i \leftarrow \alpha_i \cdot x' - \langle x \rangle_i$
  - cheat with $d_i \equiv$ cheat with $\langle x \rangle_i$
- Parties *simultaneously* broadcast $d_1, \ldots, d_n$
  - Commit + open
- Check that $\sum_{i=1}^{n} d_i = 0$
- Accept $x = x'$

Privacy

- x is supposed to be learned
- if $d = 0$, then α does not leak

Soundness

- any change is *additive change*
- Successful change of x to x' gives knowledge of $\alpha \cdot (x - x')$

Why (oblivious) permutations in MPC?

$[\![v_1]\!], [\![v_2]\!], \ldots, [\![v_m]\!]$

Processing, which selects some of the v_i, intended to be opened

Why (oblivious) permutations in MPC?

$[\![v_1]\!], [\![v_2]\!], \ldots, [\![v_m]\!]$

Processing, which selects some of the v_i, intended to be opened

$[\![b_1]\!], [\![b_2]\!], \ldots, [\![b_m]\!]$

Why (oblivious) permutations in MPC?

$[\![v_1]\!], [\![v_2]\!], \ldots, [\![v_m]\!]$

↓

Processing, which selects some of the v_i, intended to be opened

↓

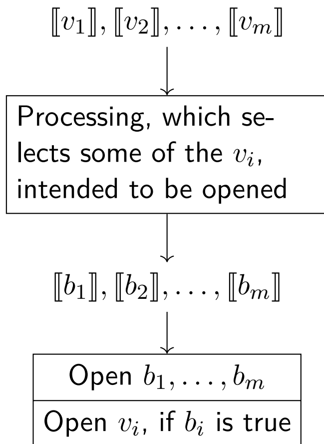
$[\![b_1]\!], [\![b_2]\!], \ldots, [\![b_m]\!]$

↓

Open $b_1, \ldots, b_m$
Open v_i, if b_i is true

- This reveals the locations of selected v_i

Why (oblivious) permutations in MPC?



- This reveals the locations of selected v_i
- Instead, one should
  - Generate a random, private permutation $[\![\pi]\!]$ for m elements
  - Compute $[\![\vec{c}]\!] \leftarrow [\![\pi]\!]([\![\vec{b}]\!])$ and $[\![\vec{w}]\!] \leftarrow [\![\pi]\!]([\![\vec{v}]\!])$
  - Open $c_1, \ldots, c_m$
  - Open w_i, if c_i is true

More uses for oblivious permutations

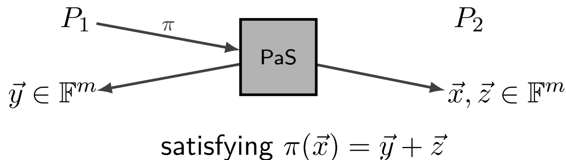
- More efficient sorting algorithms
  - Sorting networks have oblivious data access patterns
  - But quicksort has not
- Sorting + oblivious permutations give oblivious parallel RAM

Oblivious permutations: state of the art

- Constructions exist for passively secure MPC
  - A construction with $O(m) \cdot 2^{O(n)}$ complexity has found a lot of use
- For active security: protocols based on
  - Sorting networks — $O(m \log^2 m)$ complexity
  - ORAM — $O(m \log m)$ complexity
  - Mix-nets — heavyweight cryptography
  - post-execution verification — for different protocols

Permute-and-Share (again)

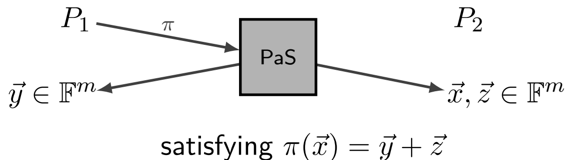
- P_1 has permutation π of m elements



- If one party is malicious, then still private, but not necessarily correct

Permute-and-Share (again)

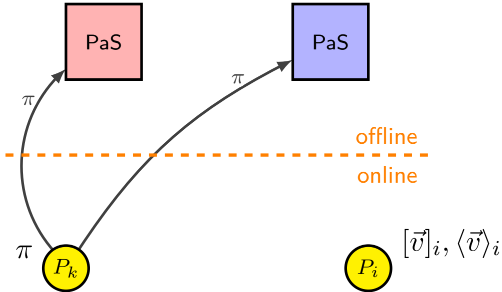
- P_1 has permutation π of m elements



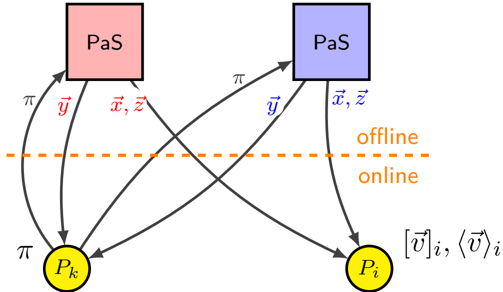
- If one party is malicious, then still private, but not necessarily correct
- Protocols for Permute-and-Share have been proposed
  - Also with optimizations for multiple instances using the same π

Applying a permutation known to k-th party

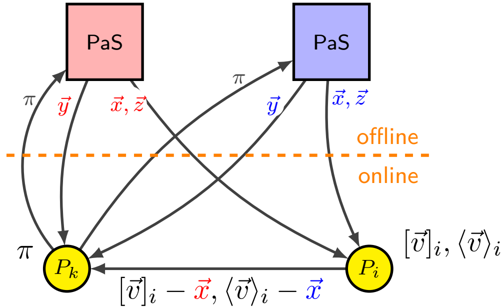
Applying a permutation known to k-th party



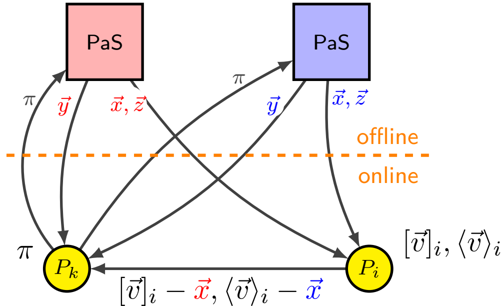
Applying a permutation known to k-th party



Applying a permutation known to k-th party



Applying a permutation known to k-th party

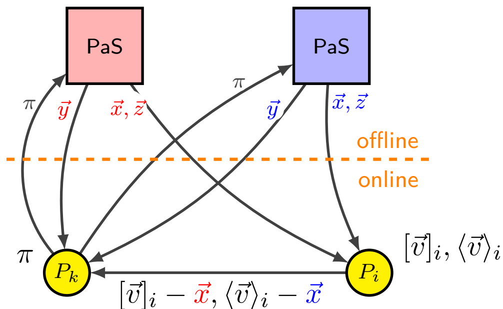


$[\vec{s}]_i \leftarrow \pi([\vec{v}]_i - \vec{x}) + \vec{y}$ $\qquad$ $[\vec{w}]_i \leftarrow \vec{z}$

$\langle\vec{s}\rangle_i \leftarrow \pi(\langle\vec{v}\rangle_i - \vec{x}) + \vec{y}$ $\qquad$ $\langle\vec{w}\rangle_i \leftarrow \vec{z}$

$[\![\vec{s}]\!]_i, [\![\vec{w}]\!]_i$ additively share $\pi([\![\vec{v}]\!]_i)$

Applying a permutation known to k-th party



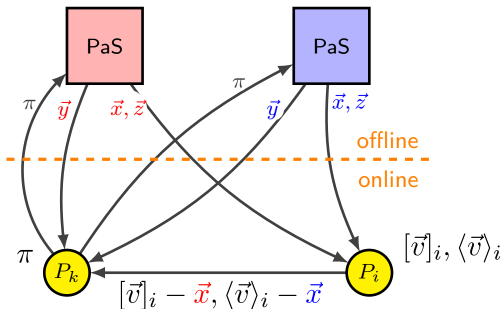
$[\vec{s}]_i \leftarrow \pi([\vec{v}]_i - \vec{x}) + \vec{y}$ $\quad$ $[\vec{w}]_i \leftarrow \vec{z}$

$\langle\vec{s}\rangle_i \leftarrow \pi(\langle\vec{v}\rangle_i - \vec{x}) + \vec{y}$ $\quad$ $\langle\vec{w}\rangle_i \leftarrow \vec{z}$

$[\![\vec{s}]\!]_i, [\![\vec{w}]\!]_i$ additively share $\pi([\![\vec{v}]\!]_i)$

- P_k runs this protocol with all P_i in parallel
  - $i \in \{1, \ldots, n\} \setminus \{k\}$

Applying a permutation known to k-th party



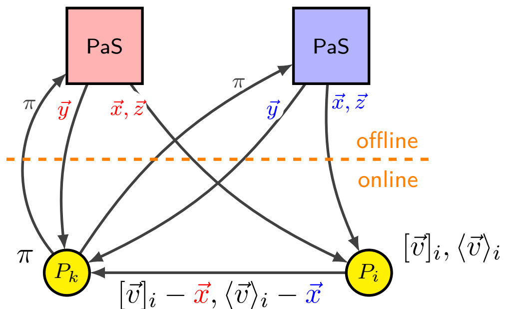
$[\vec{s}]_i \leftarrow \pi([\vec{v}]_i - \vec{x}) + \vec{y}$ $\quad$ $[\vec{w}]_i \leftarrow \vec{z}$

$\langle\vec{s}\rangle_i \leftarrow \pi(\langle\vec{v}\rangle_i - \vec{x}) + \vec{y}$ $\quad$ $\langle\vec{w}\rangle_i \leftarrow \vec{z}$

$[\![\vec{s}]\!]_i, [\![\vec{w}]\!]_i$ additively share $\pi([\![\vec{v}]\!]_i)$

- P_k runs this protocol with all P_i in parallel
  - $i \in \{1, \ldots, n\} \setminus \{k\}$
- P_i obtains $[\![\vec{w}]\!]_i$ as result
- P_k obtains $[\![\vec{s}]\!]_1, \ldots, [\![\vec{s}]\!]_n$ (except $[\![\vec{s}]\!]_k$)

Applying a permutation known to k-th party



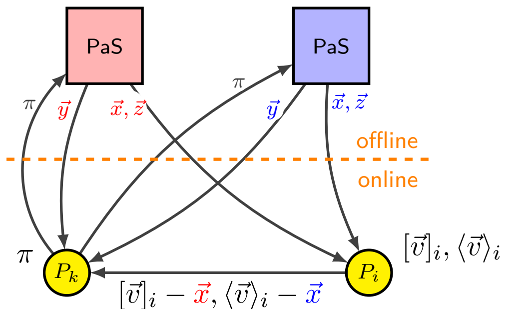
$[\vec{s}]_i \leftarrow \pi([\vec{v}]_i - \vec{x}) + \vec{y}$ $\qquad$ $[\vec{w}]_i \leftarrow \vec{z}$

$\langle\vec{s}\rangle_i \leftarrow \pi(\langle\vec{v}\rangle_i - \vec{x}) + \vec{y}$ $\qquad$ $\langle\vec{w}\rangle_i \leftarrow \vec{z}$

$[\![\vec{s}]\!]_i, [\![\vec{w}]\!]_i$ additively share $\pi([\![\vec{v}]\!]_i)$

- P_k runs this protocol with all P_i in parallel
  - $i \in \{1, \dots, n\} \setminus \{k\}$
- P_i obtains $[\![\vec{w}]\!]_i$ as result
- P_k obtains $[\![\vec{s}]\!]_1, \dots, [\![\vec{s}]\!]_n$ (except $[\![\vec{s}]\!]_k$)
- P_k defines $[\![\vec{s}]\!]_k \leftarrow \pi([\![\vec{v}]\!]_k)$
- P_k defines $[\![\vec{w}]\!]_k \leftarrow \sum_{i=1}^{n} [\![\vec{s}]\!]_i$

Applying a permutation known to k-th party



$[\vec{s}]_i \leftarrow \pi([\vec{v}]_i - \vec{x}) + \vec{y}$ $\quad$ $[\vec{w}]_i \leftarrow \vec{z}$

$\langle \vec{s} \rangle_i \leftarrow \pi(\langle \vec{v} \rangle_i - \vec{x}) + \vec{y}$ $\quad$ $\langle \vec{w} \rangle_i \leftarrow \vec{z}$

$[\![\vec{s}]\!]_i, [\![\vec{w}]\!]_i$ additively share $\pi([\![\vec{v}]\!]_i)$

- P_k runs this protocol with all P_i in parallel
  - $i \in \{1, \ldots, n\} \setminus \{k\}$
- P_i obtains $[\![\vec{w}]\!]_i$ as result
- P_k obtains $[\![\vec{s}]\!]_1, \ldots, [\![\vec{s}]\!]_n$ (except $[\![\vec{s}]\!]_k$)
- P_k defines $[\![\vec{s}]\!]_k \leftarrow \pi([\![\vec{v}]\!]_k)$
- P_k defines $[\![\vec{w}]\!]_k \leftarrow \sum_{i=1}^{n} [\![\vec{s}]\!]_i$
- Private, but not necessarily correct

Discussion

- We propose new kinds of precomputations to be done
  - P_k chooses π and gets $\vec{y}$. P_i gets $\vec{x}, \vec{z}$
    - Such that $\pi(\vec{x}) = \vec{y} + \vec{z}$
  - We do not say, what protocol is good for these precomputations
  - (permutation network + oblivious transfer would work)

Discussion

- We propose new kinds of precomputations to be done
  - P_k chooses π and gets $\vec{y}$. P_i gets $\vec{x}, \vec{z}$
    - Such that $\pi(\vec{x}) = \vec{y} + \vec{z}$
  - We do not say, what protocol is good for these precomputations
  - (permutation network + oblivious transfer would work)
- Permutation π has to be fixed already during precomputation. This is not a significant constraint, because
  - ...in applications, we usually use private permutations that are *random*
  - ...a particular permutation π' can be handled by $P_k \rightarrow P_i : \pi' \cdot \pi^{-1}$
    - $\langle(\pi, \vec{y}), (\vec{x}, \vec{z})\rangle$ can be turned to $\langle(\pi^{-1}, \pi^{-1}(-\vec{y})), (\vec{z}, \vec{x})\rangle$ without any communication

Oblivious permutation (1/2)

- Private representation $[\![\pi]\!]$ of permutation π is the following:
  - i-th party holds a random permutation π_i, subject to $\pi_1 \circ \cdots \circ \pi_n = \pi$

Oblivious permutation (1/2)

- Private representation $[\![\pi]\!]$ of permutation π is the following:
  - i-th party holds a random permutation π_i, subject to $\pi_1 \circ \cdots \circ \pi_n = \pi$
- Applying $[\![\pi]\!]$ to $[\![\vec{v}]\!]$:
  - Apply π_1 (known to P_1) to $[\![\vec{v}]\!]$,
  - Apply π_2 (known to P_2) to the result,
  - ...
  - Apply π_n (known to P_n) to the result, giving $[\![\vec{w}]\!]$

Oblivious permutation (1/2)

- Private representation $[\![\pi]\!]$ of permutation π is the following:
  - i-th party holds a random permutation π_i, subject to $\pi_1 \circ \cdots \circ \pi_n = \pi$
- Applying $[\![\pi]\!]$ to $[\![\vec{v}]\!]$:
  - Apply π_1 (known to P_1) to $[\![\vec{v}]\!]$,
  - Apply π_2 (known to P_2) to the result,
  - ...
  - Apply π_n (known to P_n) to the result, giving $[\![\vec{w}]\!]$
- This is private. But how to be sure that $\vec{v}$ and $\vec{w}$ have the same elements?

Oblivious permutation (2/2)

- Pick fresh random $[\![r]\!], [\![r']\!]$
- Compute

$$[\![r']\!] \cdot \left(\prod_{i=1}^{m}([\![r]\!] - [\![v_i]\!]) - \prod_{i=1}^{m}([\![r]\!] - [\![w_i]\!])\right)$$

- Open the result, abort if $\neq 0$
  - Random r' masks any possible leaks, if the result is not 0

Permuting two vectors with the same permutation

- $\vec{w}$, $\vec{w}'$ are the same permutation of $\vec{v}$, $\vec{v}'$, iff

$$\prod_{i=1}^{m}(X - v_i - Yv_i') = \prod_{i=1}^{m}(X - w_i - Yw_i')$$

- Hence, after applying the first half of the permutation protocol to both $\vec{v}$ and $\vec{v}'$, we
  - Pick fresh random $[\![r]\!], [\![s]\!], [\![r']\!]$
  - Open r and s
  - Compute

$$[\![r']\!] \cdot \left(\prod_{i=1}^{m}(r - [\![v_i]\!] - s[\![v_i']\!]) - \prod_{i=1}^{m}(r - [\![w_i]\!] - s[\![w_i']\!])\right)$$

  - Open the result, abort if $\neq 0$

(Online) complexity

- First part (permute-and-share):
  - Each party communicates $O(nm)$ elements of $\mathbb{F}$
    - $O(n^2m)$ in total
  - Number of communication rounds: $O(n)$
- Second part (evaluating polynomials):
  - $2m - 1$ multiplications with $\lceil \log m \rceil$ multiplicative depth