Secure Multi-Party Computation

CS-523

Wouter Lueks | February 25, 2025 | v1.1.0

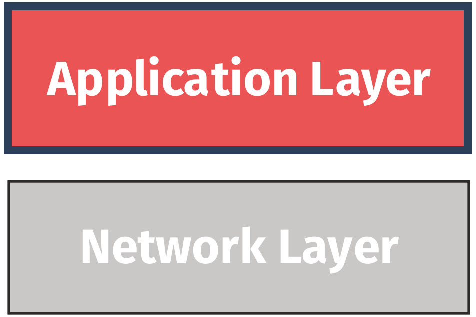
Some *slides inspired by Jean-Pierre Hubaux*

Introduction
Secure Multi-Party Computation (SMC)

Course aim: learn **toolbox for privacy engineering**

tool
for building PETS

cryptography
as main technique

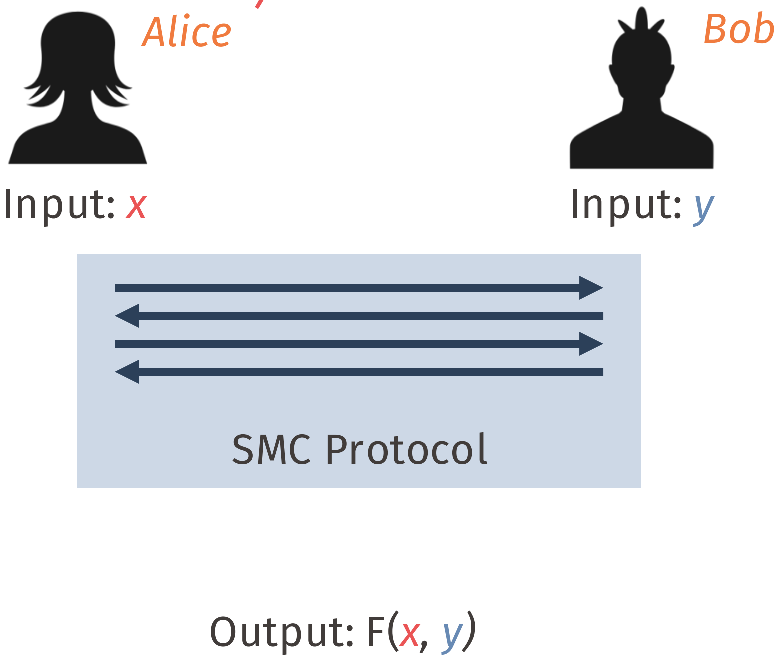


Goals
What should you learn today?

- Basic understanding of **two SMC techniques**
- Know **when** SMC is a **useful tool** in creating privacy-friendly systems
- Understand **how to express your problem as a circuit** to enable MPC
- Understand **key properties**:
  - Communication and computation cost
  - Trust assumptions
  - Guarantees with respect to inputs
- Be able to **use** SMC as a building block (by describing which **function it computes**)

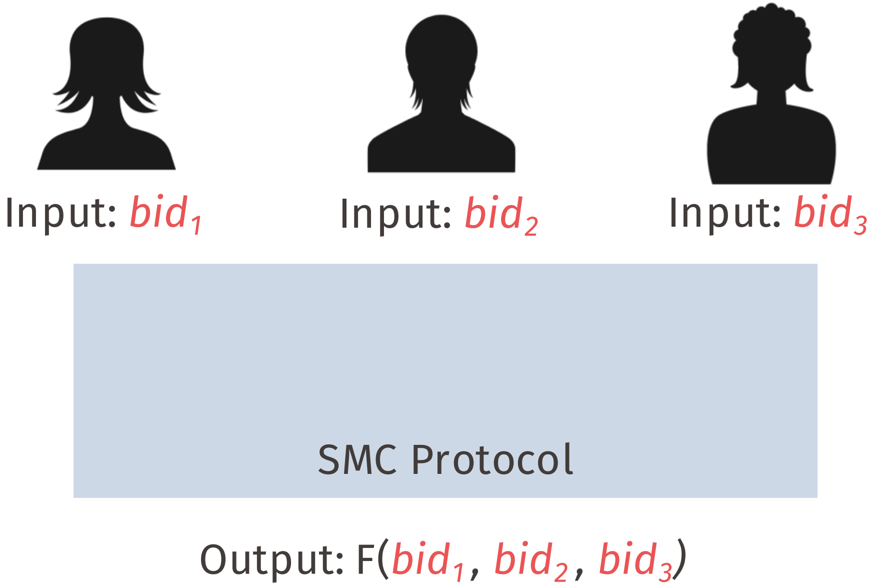
Overview

SMC on one-slide (2-party version)



- A secure 2-party protocol enables two parties with private inputs x and y to compute a function $F(x, y)$
- **Security property**: without either party learning anything more than what can be derived from the output
- **Correctness**: output is correct
- **Types of circuits** (today): Boolean circuits (with logic gates), and arithmetic circuits (with addition/multiplication gates)
- **Secure Multi-party Computation**: have n parties with private inputs $x_1, \ldots, x_n$ compute $F(x_1, \ldots, x_n)$

Example
Secure Auction



- In auction, multiple parties bidding for an item (say, a house). Want to find the party that bid highest. Privacy: **without revealing anybody else's bid**.
- Define **function**:
  $$F(bid_1, \ldots, bid_3) = (i, bid_i) \text{ s.t. } \forall j \; bid_j \leq bid_i$$
- *(This function is not really a circuit, we'll fix that later)*
- SMC protocol guarantees: none of the parties learn more than the output.

High Level Structure

High-Level description of function *F* to compute

Your Problem

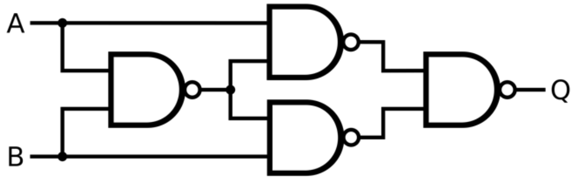
A boolean/arithmetic circuit *C* that realizes / implements *F*

Protocol for computing *C*

SMC Protocol/Transformation

Example function F:

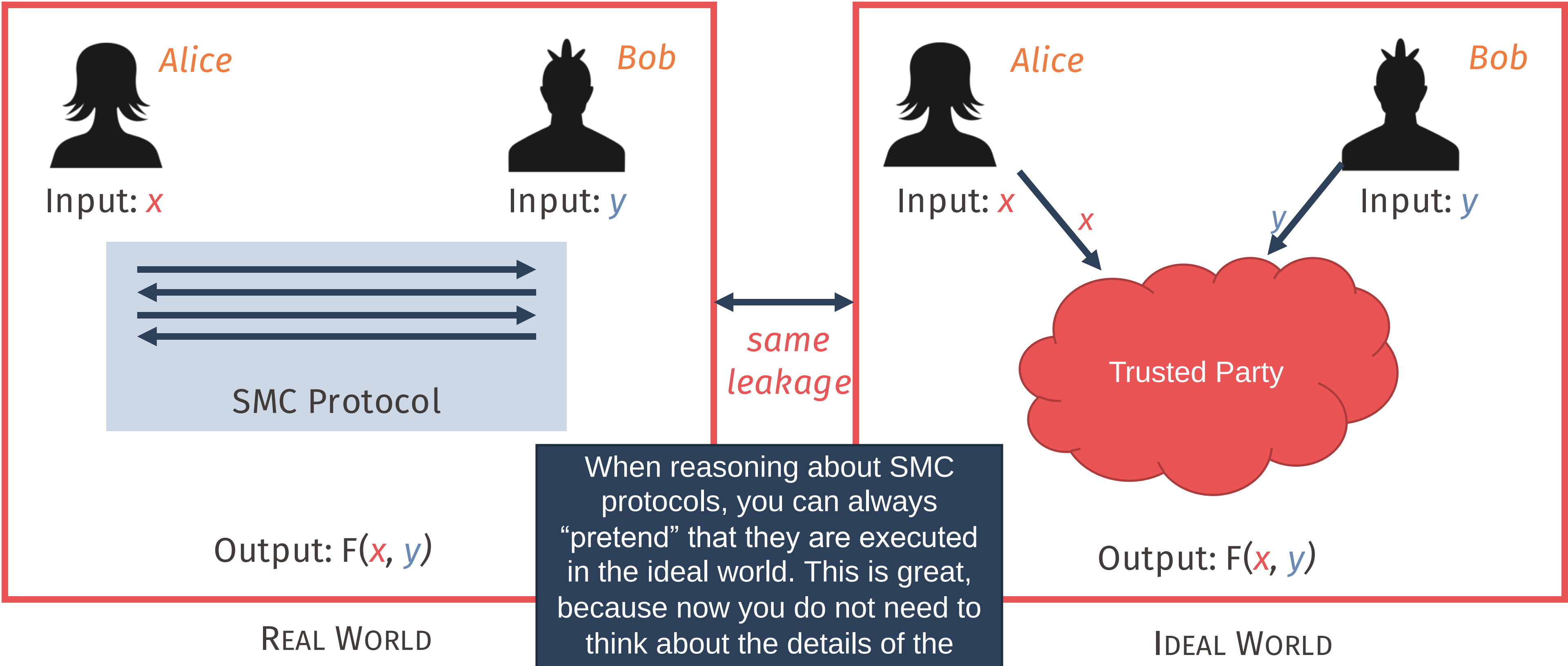
$$F(bid_1, \ldots, bid_3) = (i, bid_i)\ s.t. \forall j\ bid_j \leq bid_i$$



Example Logic Circuit C
(*Not implementing F on the left*)

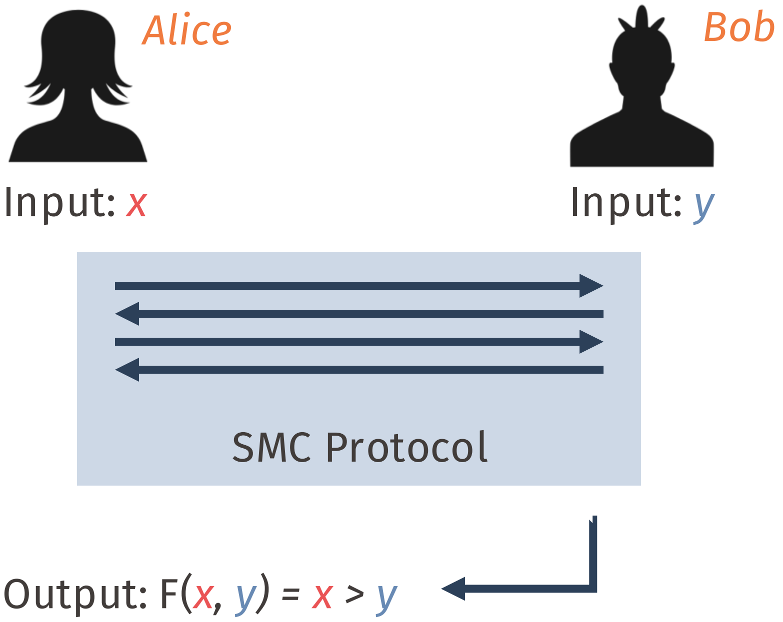
Security
Ideal-world equivalent

- **Security property**: without either party learning anything more than what can be derived from the output



When reasoning about SMC protocols, you can always "pretend" that they are executed in the ideal world. This is great, because now you do not need to think about the details of the construction.

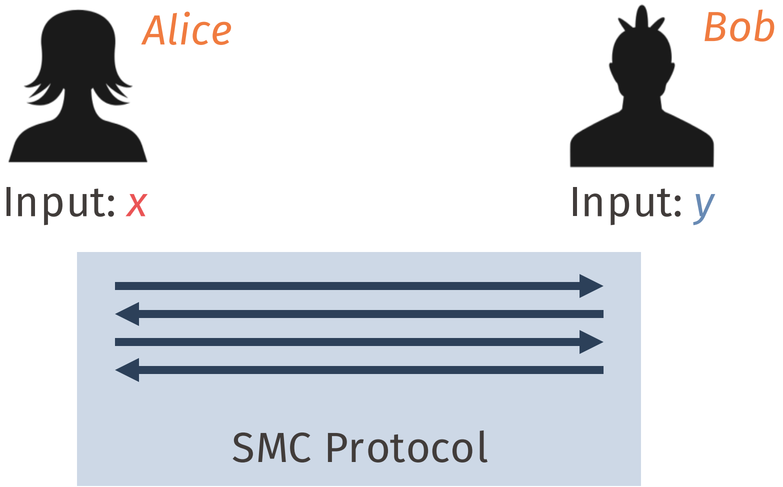
Example
What does Alice learn?



- Simple example function: $F(x, y) = x > y$
- Assume: SMC protocol is secure, i.e., we can reason about it using the ideal world.
- Assume: only Alice learns the output
- **Question:** what does Alice learn about Bob's input?

Answer: if the answer is true Alice learns an upper bound (x) on Bob's value, and a lowerbound otherwise.

Example and caveat
What does Alice learn?



- As on the previous slide, but now suppose Alice and Bob run the same protocol **several times.**
- Bob will use the same input *y* for every run.
- **Question:** what can Alice learn about Bob's value *y*?

Take Away
Even though the SMC protocol is itself secure, the composition of the SMC protocol with other parts of the system does not have to be secure

Answer: Alice can *change* her inputs, she can then use binary search to learn Bob's exact value.

Threat Models
Honest but Curious **vs** Malicious

- So far, treated the SMC protocol as a **black box** where parties can only control what they **input.**
- Must take into account **threat model** aka what can parties do. Today we consider two:
- **Honest but Curious*:** Parties will follow the SMC protocol honestly, but try to learn as much as possible from the messages they receive
- **Malicious:** Parties can arbitrarily deviate from the SMC protocol to learn as much as possible

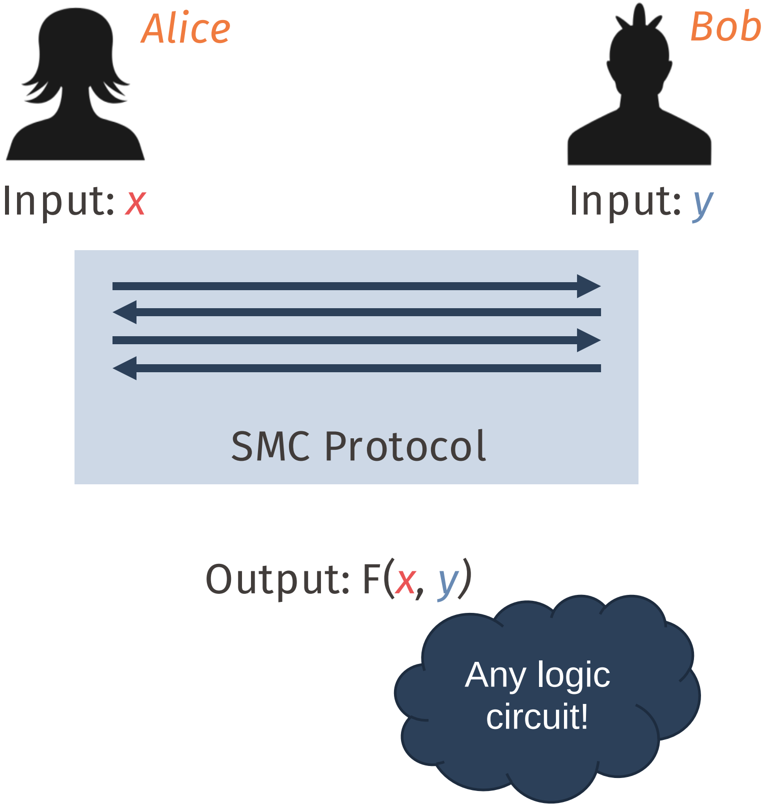
Today's Class

Mostly in the Honest-but-Curious setting.

This is not – usually – a realistic assumption! There are ways to fix this, but these are often costly.

Secure Multi-Party Computation
A generic solution?

- There exist SMC protocols (both 2 party and multi-party) that can compute ***any Boolean circuit***
- Therefore: SMC protocols are **universal**: can solve any problem!
- So why do we not always use them? **Custom protocols** are often “better”:
  - Might use less **bandwidth**
  - Might be faster, i.e., use less **computation**
  - Might use fewer **rounds of communication**
  - Might work even if all parties **are not online at the same time**

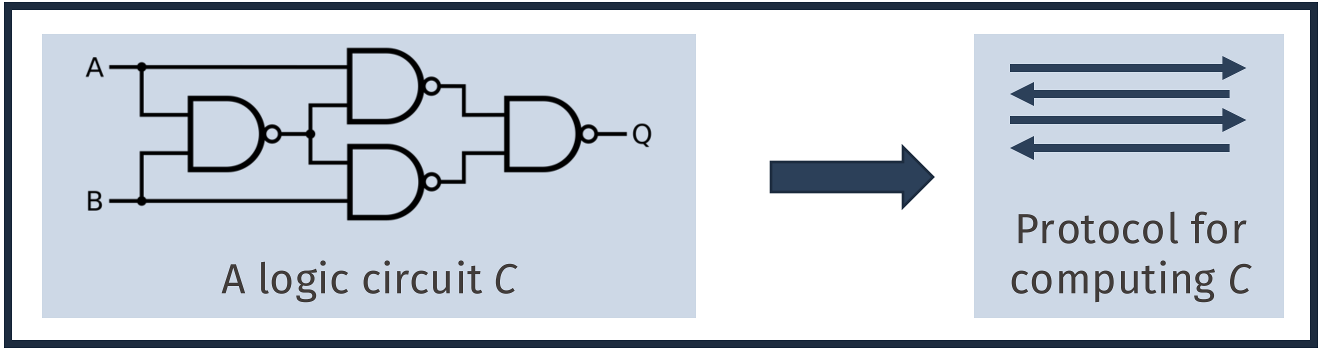


A two-party secure multi-party protocol: Yao’s garbled circuit

Overview
Yao's Garbled Circuits

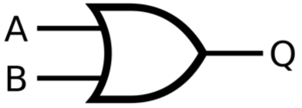
Two Parties

Parties are
Honest but Curious



Transformation

Key Idea
Gates as Truth Tables



Logic Gate

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table

Step 1: Assign **random labels** to each possible input value

A	label
0	W_A^0
1	W_A^1

B	label
0	W_B^0
1	W_B^1

Step 2: Create encrypted truth table

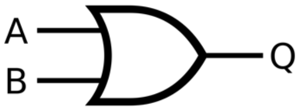
A	B	Q
W_A^0	W_B^0	Enc($W_A^0 \parallel W_B^0$, 0)
W_A^0	W_B^1	Enc($W_A^0 \parallel W_B^1$, 1)
W_A^1	W_B^0	Enc($W_A^1 \parallel W_B^0$, 1)
W_A^1	W_B^1	Enc($W_A^1 \parallel W_B^1$, 1)

Step 3: Shuffle output column to create a **garbled gate**

Q
Enc($W_A^1 \parallel W_B^0$, 1)
Enc($W_A^1 \parallel W_B^1$, 1)
Enc($W_A^0 \parallel W_B^0$, 0)
Enc($W_A^0 \parallel W_B^1$, 1)

Note: we use that if key is wrong Dec fails

Garbling a Single Gate
Yao's Garbled Circuits



Logic Gate

Input: a

Garbler

A	label
0	W_A^0
1	W_A^1

B	label
0	W_B^0
1	W_B^1

Input: b

Evaluator

Step 1. Compute garbled gate

Step 2. Send garbled gate and Garbler's input label

Q
Enc(W_A^1 ‖ W_B^0, 1)
Enc(W_A^1 ‖ W_B^1, 1)
Enc(W_A^0 ‖ W_B^0, 0)
Enc(W_A^0 ‖ W_B^1, 1)

, W_A^a

Depends on Garbler's input *a*

Requires **magic** protocol! (why?)
Next slide: Oblivious Transfer

Step 3. Get label W_B^b for Evaluator's input b

Step 4. Find row that decrypts for W_A^a ‖ W_B^b

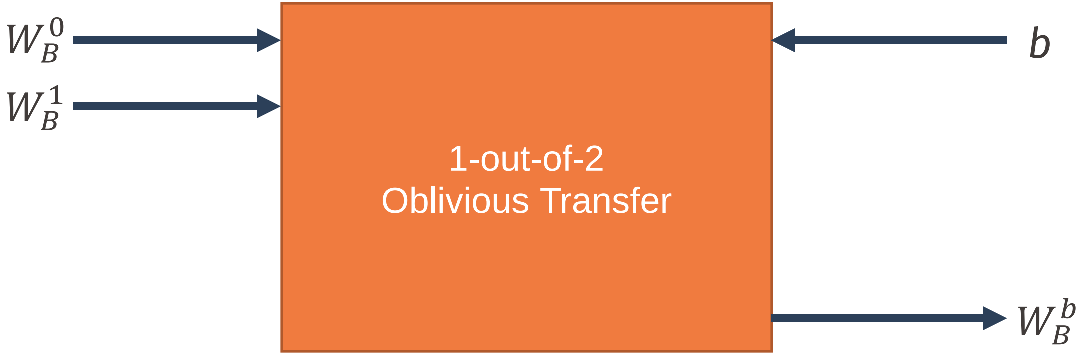
Intermezzo
Oblivious Transfer

Garbler/ Sender

B	label
0	W_B^0
1	W_B^1

Input: b

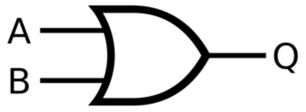
Evaluator Receiver



Key Properties:
- Sender learns nothing about the bit b
- Receiver learns nothing about W_B^{1-b}

Simple OT Protocols
(from Public Key encryption)

Garbling a Single Gate
Yao's Garbled Circuits



Logic Gate

Input: a

Garbler

A	label
0	W_A^0
1	W_A^1

B	label
0	W_B^0
1	W_B^1

Input: b

Evaluator

Step 1. Compute garbled gate

Step 2. Send garbled gate and Garbler's input label

Q
Enc(W_A^1 ‖ W_B^0, 1)
Enc(W_A^1 ‖ W_B^1, 1)
Enc(W_A^0 ‖ W_B^0, 0)
Enc(W_A^0 ‖ W_B^1, 1)

, W_A^a

Depends on Garbler's input *a*

W_B^0

W_B^1

1-out-of-2 Obvlious transfer

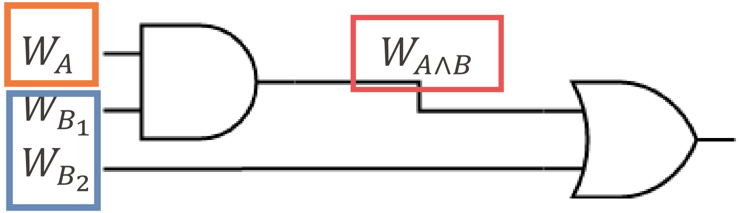
b

W_B^b

Step 3. Send label W_B^b for Evaluator's input b

Step 4. Find row that decrypts for W_A^a ‖ W_B^b

Full Circuits
Recursion to the rescue!



Generation (garbling)

Step 1. Assign **random labels** to each **wire**

	W_A	W_{B_1}	W_{B_2}	$W_{A \wedge B}$
0	W_A^0	$W_{B_1}^0$	$W_{B_2}^0$	$W_{A \wedge B}^0$
1	W_A^0	$W_{B_1}^1$	$W_{B_2}^1$	$W_{A \wedge B}^1$

Step 2. Generate **garbled gates** and send to the evaluator

AND
Enc($W_A^1 \parallel W_{B_1}^0$, $W_{A \wedge B}^0$)
Enc($W_A^1 \parallel W_{B_1}^1$, $W_{A \wedge B}^1$)
Enc($W_A^0 \parallel W_{B_1}^0$, $W_{A \wedge B}^0$)
Enc($W_A^0 \parallel W_{B_1}^1$, $W_{A \wedge B}^0$)

OR
Enc($W_{A \wedge B}^0 \parallel W_{B_2}^1$, 1)
Enc($W_{A \wedge B}^1 \parallel W_{B_2}^1$, 1)
Enc($W_{A \wedge B}^1 \parallel W_{B_2}^0$, 1)
Enc($W_{A \wedge B}^0 \parallel W_{B_2}^0$, 0)

Evaluation

Step 3. Obtain generator's inputs (W_A^a)

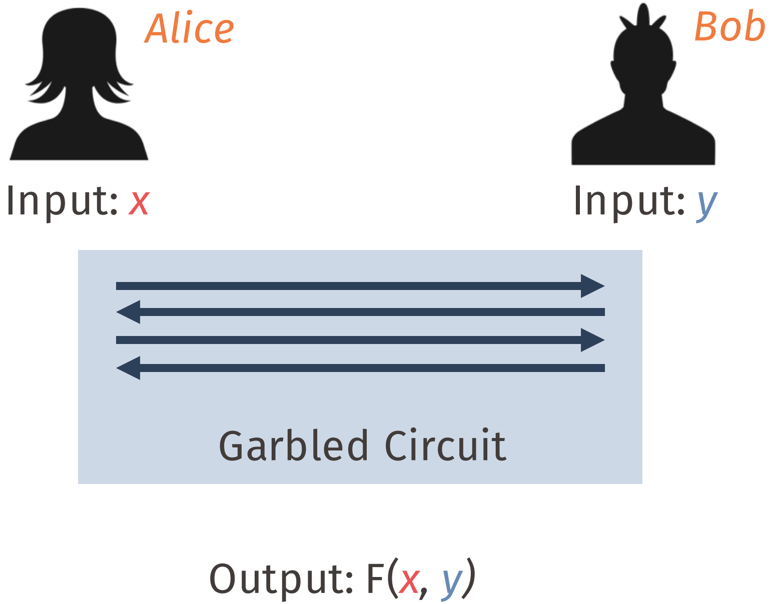
Step 4. Obtain evaluator's (own) inputs ($W_{B_1}^{b_1}, W_{B_2}^{b_2}$) via OT

Step 5. Evaluate gates in order to obtain intermediate wire values ($W_{A \wedge B}$) and final outputs

Yao's Garbled Circuits
Properties

- Evaluating any circuit requires only **a constant number of rounds** (one message to send the garbled circuit; and a bunch of (parallel) OTs to get the evaluator's inputs).
- Communication cost is **linear** in the number of gates
- Computation cost is is **linear** in the number of gates

Example: Circuits can be slow. For example, evaluating 1 AES block: 17 seconds and requires 77 MB of data.



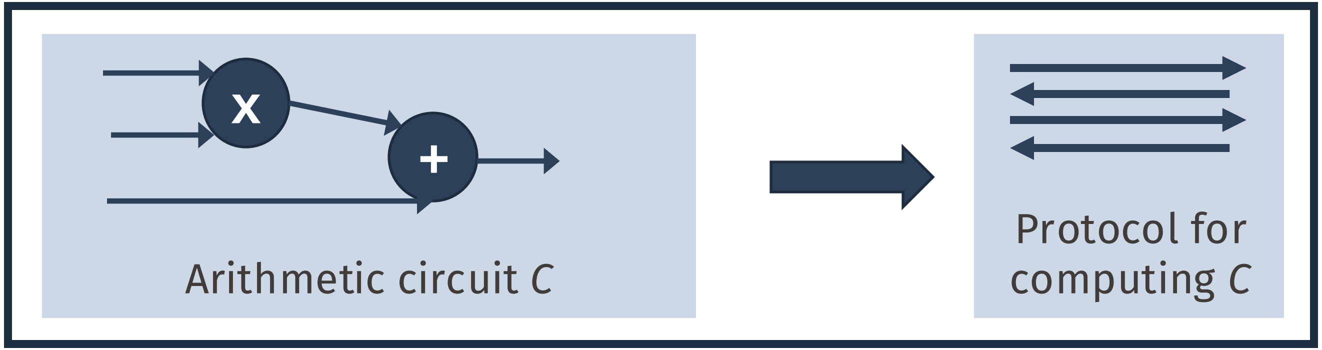
A multi-party secure multi-party protocol: Ben-Or, Goldwasser, Wigderson (BGW)

For arithmetic circuits

Overview
BGW Circuits

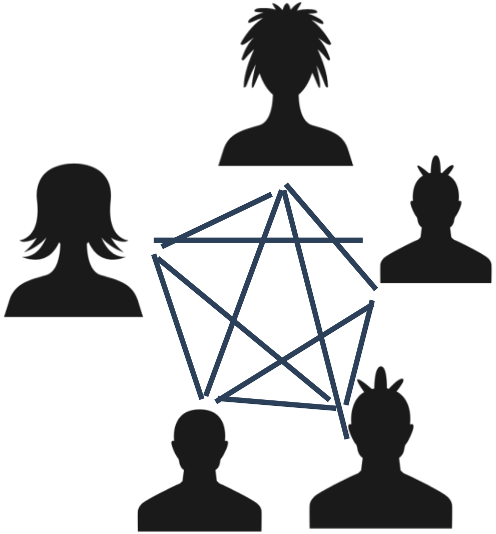
N Parties

Parties are
Honest but Curious

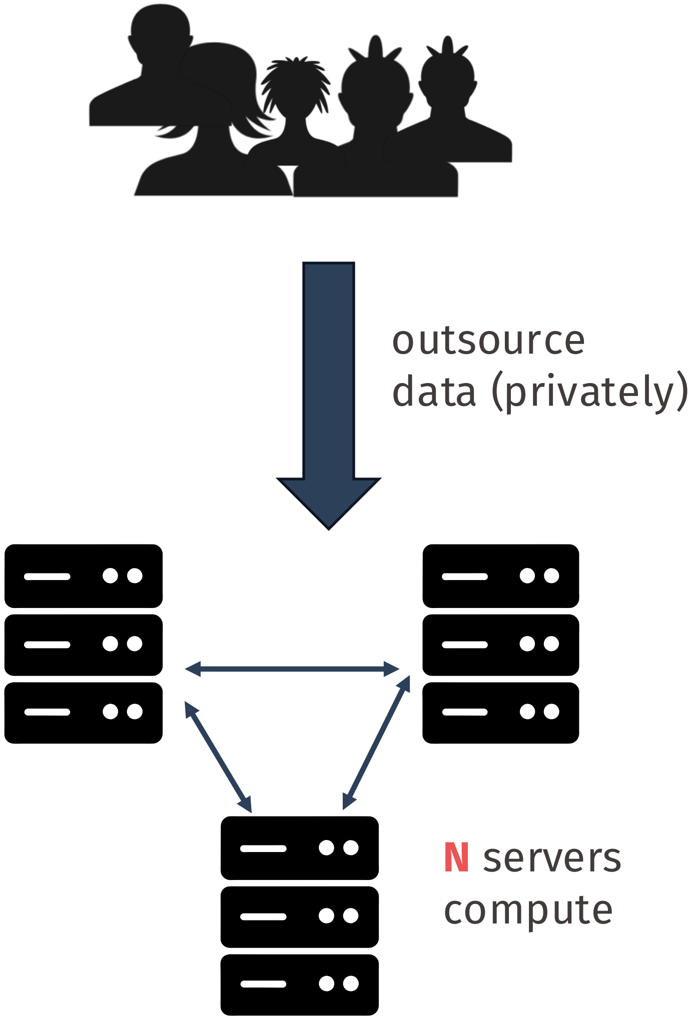


Transformation

Two Settings



N Parties. -- They compute themselves



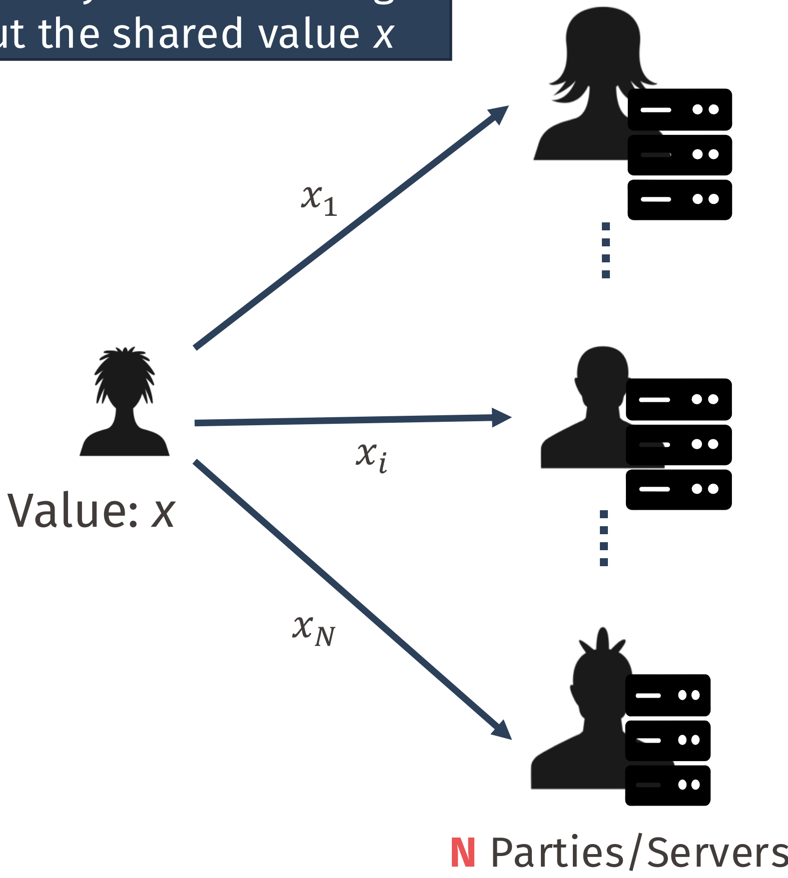
N servers compute

Building Block
Additive Secret Shares

Privacy Property: given at most *N – 1* shares, an adversary learns nothing about the shared value *x*

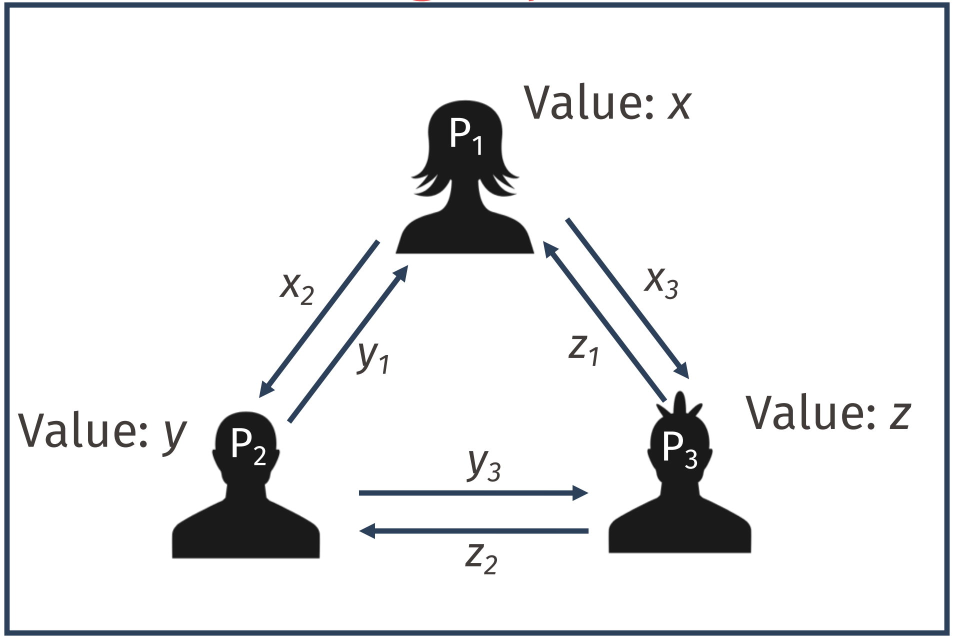
Operate over a field $\mathbb{F}$ (for example, integers modulo a prime *p*)

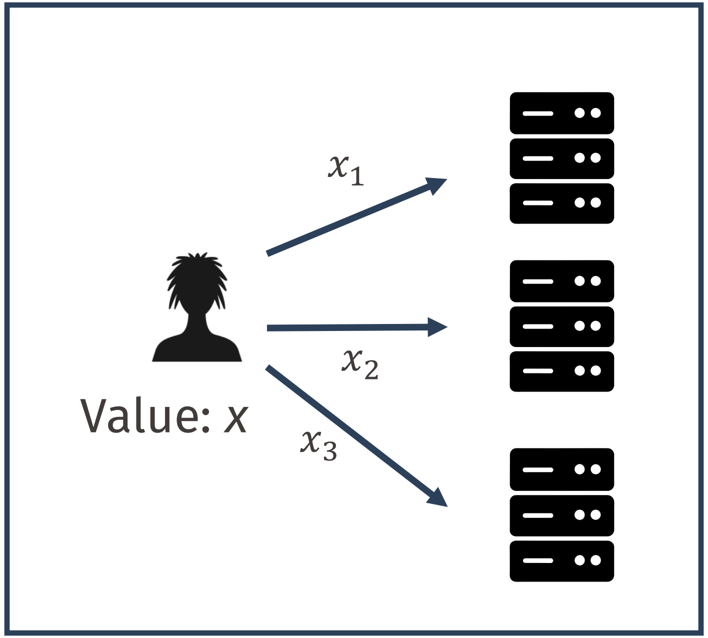
- **Share:** given a value $x \in \mathbb{F}$ we compute shares $x_1, \ldots, x_N$:
  - Sample $x_2, \ldots, x_N$ uniformly at random from $\mathbb{F}$
  - Set $x_1 = x - \sum_{\{i=2\}}^{N} x_i$ (over $\mathbb{F}$)
  - We denote $[x] = \{x_1, \ldots, x_N\}$ the sharing of x
- **Reconstruction:** given a sharing $[x] = \{x_1, \ldots, x_N\}$ output $x = \sum_{i=1}^{N} x_i$



First Step
Sharing Inputs

All drawings with **N=3** parties





Input sharing setting: each party secret shares their inputs to each of the other parties (and keeps one share)

Outsourced computation setting: parties with data secret share their inputs to the compute nodes

Computing on shares
Addition (Add-Protocol)

General Structure/Invariant: the parties in the protocol hold secret shares of the circuit wire values.

Here: Party *i* holds secret shares s_i, v_i such that: $s = \sum_i s_i$ and $v = \sum_i v_i$.

Goal: Each party must obtain t_i such that $t = \sum_i t_i = s + v$ or in other words $[t] = [s + v]$

Algorithm:

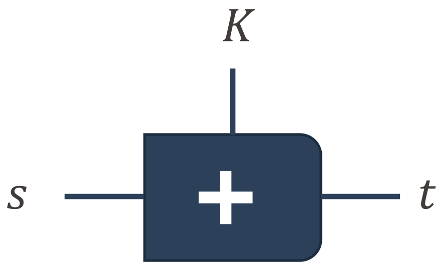
- Each party (locally!) sets $t_i = s_i + v_i$

Computing on shares
Addition (Add-K Protocol)

General Structure/Invariant: the parties in the protocol hold secret shares of the circuit wire values.

Here: Party *i* holds secret shares s_i such that: $s = \sum_i s_i$ and K is a public value

Goal: Each party must obtain t_i such that $t = \sum_i t_i = s + K$ or in other words $[t] = [s + K]$



Want to add a public value K

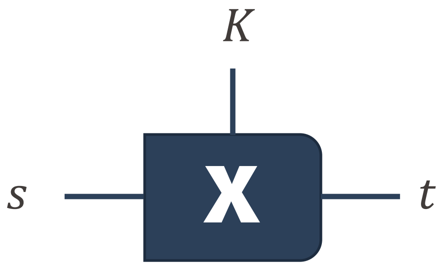
Algorithm:

- Party 1: locally sets $t_1 = s_1 + K$
- Other parties *i*: locally set $t_i = s_i$

Computing on shares
Multiplication (Mult-K Protocol)

General Structure/Invariant: the parties in the protocol hold secret shares of the circuit wire values.

Exercise :).



Want to multiply by a public value K

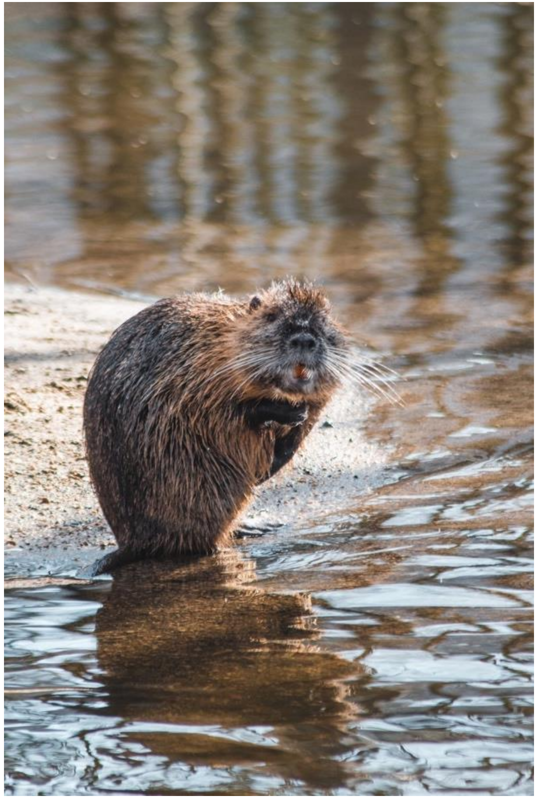
Intermezzo
Beaver Triplets

Computing **multiplication gates** is harder. We use a trick.

A **Beaver triplet** (a, b, c) such that a and b are random (in the field) and $c = ab$.

We assume that the parties hold secret shares of the Beaver triplet: $[a], [b], [c]$

For security it is essential that *no parties know the values a, b, c*. They can only know their secret share. As a result: **constructing Beaver triplets is hard.** Usual tricks: trusted third party, using Homomorphic Encryption, or from OT.



Computing on shares
Multiplication (Mul-Protocol)

Input: Party *i* holds secret shares s_i, v_i such that:

$$s = \sum_i s_i \text{ and } v = \sum_i v_i$$

as well as shares a_i, b_i, c_i, for a *fresh* Beaver Triplet (a, b, c)

Goal: Each party must obtain t_i such that $t = \sum_i t_i = sv$ or in other words $[t] = [sv]$

A useful identity:

$$\begin{aligned} sv &= (s - a + a)(v - b + b) \\ &= (\mathrm{d} + \mathrm{a})(\mathrm{e} + \mathrm{b}) \\ &= de + db + ae + ab \\ &= de + db + ea + c \end{aligned}$$

Algorithm:

1. Each party locally computes a share of *[d] = [s – a]* and broadcasts it. Each party reconstructs and learns *d*
2. Each party locally computes a share of *[e] = [v – b]* and broadcasts it. Each party reconstructs and learns *e*
3. Locally compute a share of:

   $$[sv] = de + d[b] + e[a] + [c]$$

   (note that this requires only additions and multiplications by constants)

Alternative
Shamir's Secret Sharing

Operate over a field $\mathbb{F}$ (for example, integers modulo a prime *p*)

Additive secret sharing requires all *N* shares to reconstruct. To add **robustness** (at the cost of **privacy**), could use Shamir's secret sharing so that you can reconstruct given only *t* values.

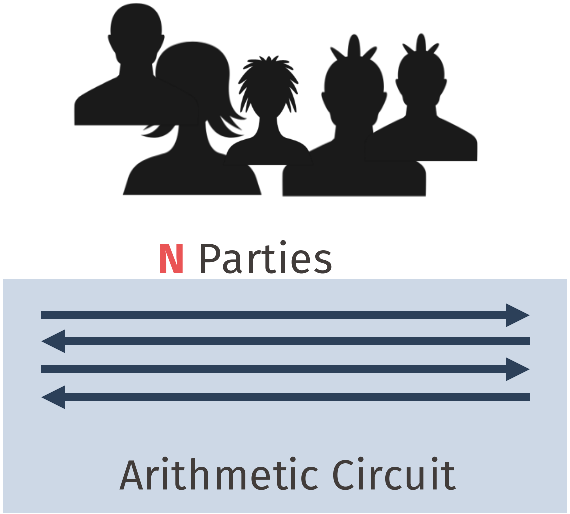
Privacy Property: in a *t-out-of-N* Shamir secret sharing scheme, an adversary given at most *t – 1* shares, learns nothing about the shared value *x*

- **Share:** given a value $x \in \mathbb{F}$ we compute shares $x_1, \dots, x_N$:
  - Sample $a_1, \dots, a_{t-1}$ uniformly at random from $\mathbb{F}$ to construct secret-sharing polynomial $f(X) = x + a_1 X + \cdots + a_{t-1} X^{t-1}$
  - Set $x_i = f(i)$ for $i \in \{1, \dots, N\}$
  - We denote $[x] = \{x_1, \dots, x_N\}$ the sharing of x
- **Reconstruction:** given *t* shares $x_{i_1}, \dots, x_{i_t}$ from parties $i_1, \dots, i_T$ reconstruct the secret through polynomial evaluation.

BGW Circuits
Properties

- The number of rounds is **linear in the circuit depth** (we need openings for each multiplication gate at each level)
- Computation cost is **linear** in the number of gates
- Communication cost is linear in the number of **multiplication** gates.

Practical Performance: Computing arithmetic circuits can be quite fast only a few modular computations per gate.

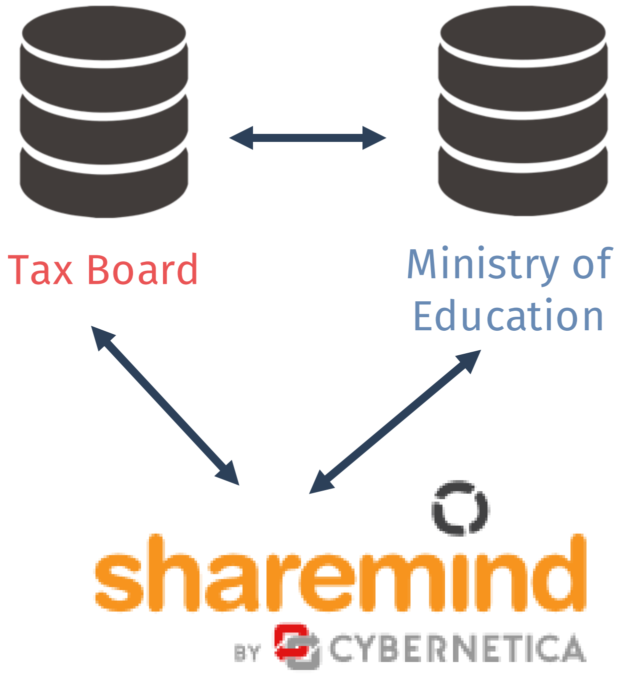


Output: $F(x_1, \ldots, x_N)$

Applications

Applications of MPC
Estonian Study

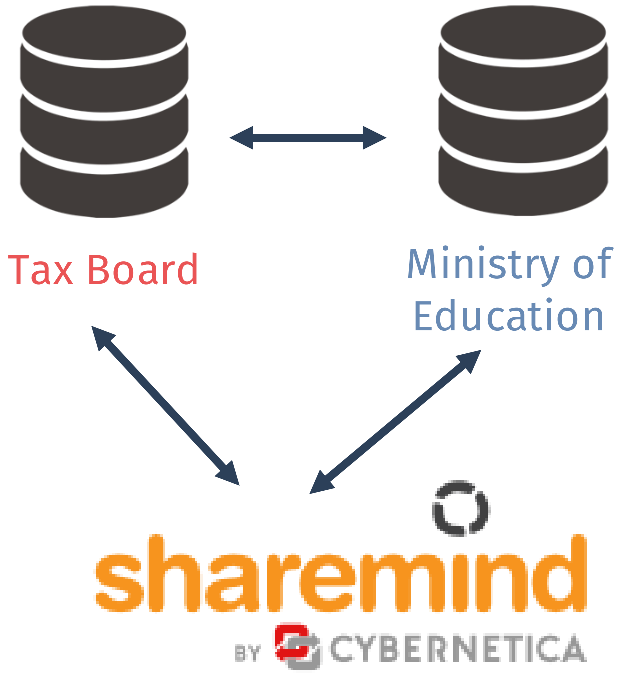
- Estonian CS programs: 43% of students failed to graduate. Question: Why?
- Hypothesis: everybody gets nice IT job before graduating
- Privacy legislation prevent sharing of data from Tax Board (10M records) and Ministry of Education (600k records)
- Use of MPC resulted in higher accuracy than using other data anonymization techniques (e.g., k-anonymity see Data Publishing Part I) that were also legally acceptable



Dan Bogdanov, Liina Kamm, Baldur Kubo, Reimo Rebane, Ville Sokk, Riivo Talviste: Students and Taxes: a Privacy-Preserving Study Using Secure Computation. Proc. Priv. Enhancing Technol. 2016(3): 117-135 (2016). https://eprint.iacr.org/2015/1159.pdf

Applications of MPC
Estonian Study II

- Technical solution built on Sharemind's MPC framework that operates on secret-shared data (e.g., see BGW before)
- Challenges faced by Cybernetica:
  - Technical implementation was difficult, especially to run at this scale
  - Convince stake-holders that this approach is actually secure
  - Operational support: ensure assumptions are met, manuals, deployment support
- Time to run is massive:
  - 384 hours
  - With 2x 2-core machine, and 1x 12-core machine



Contact

Wouter Lueks

CISPA Helmholtz Center for Information Security

lueks@cispa.de

https://wouterlueks.nl/