Actively Secure Dishonest Majority Table Join in the JOCONDE Project

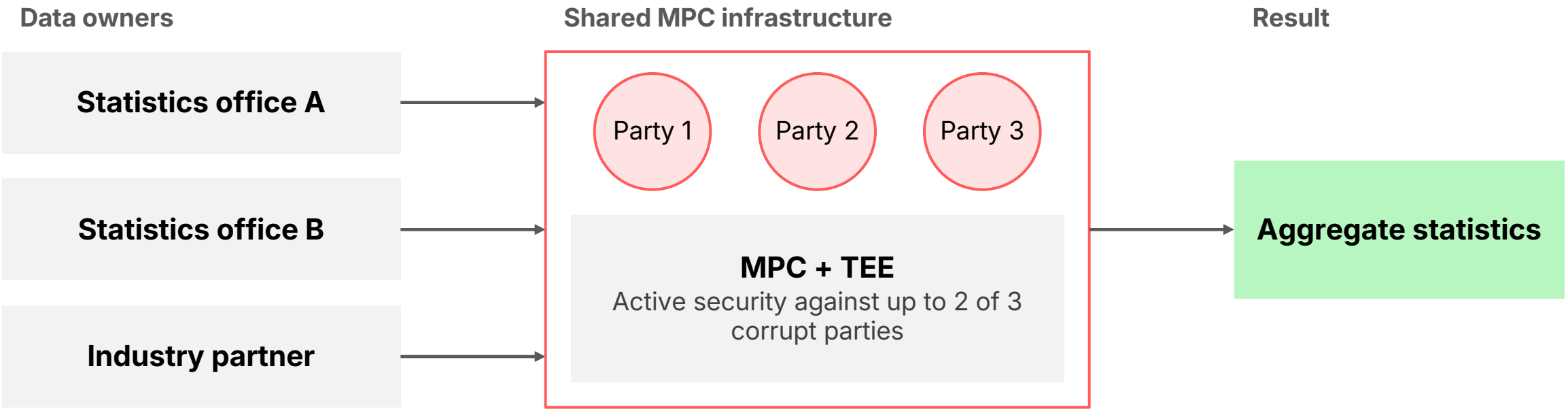
2026-10-27 @ NIST WPEC 2026

Hendrik Eerikson, Sona Kravtšenko, Pille Pullonen-Raudvere
Cybernetica AS / University of Tartu

CYBERNETICA

Eurostat JOCONDE project

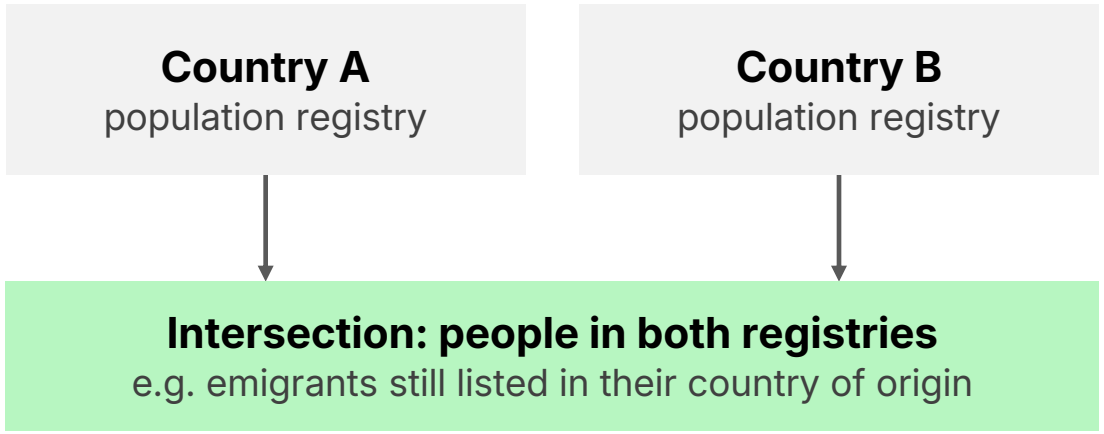
- Secure Multi-Party Computation as a Service infrastructure for official statistics
- Statistics offices and industry partners analyze confidential microdata **without revealing it**



CYBERNETICA

Example use case: EU population census

The computation



Conventional approach: one trusted party

- Both registries are **combined at one central party**
- Each data provider must trust that this party:
  - **protects** the data from internal and external intrusion
  - **deletes** it afterwards
  - is not subject to **laws requiring it to keep or share** the data

> This trust requirement is **too high a barrier** and out of proportion to the benefit of the computation.

Threat model

Security in JOCONDE

- JOCONDE can be instantiated with different MPC backends
- The **actively secure dishonest majority** backend gives the strongest guarantees
- MPC parties also run inside a **trusted execution environment (TEE)** as an additional layer of defense
- JOCONDE handles deploying MPC without any single point of trust

Adversary



- The adversary may corrupt up to **2 of 3** computing parties
- Corrupt parties may **deviate arbitrarily** from the protocol
- Cheating is **detected** and the computation aborts

Actively Secure MPC

Our starting point is the **SPDZ protocol** (Damgård, Pastro, Smart, Zakarias, CRYPTO 2012)

Authenticated secret sharing of a value x

Party 1	Party 2	Party 3
x_1 , m_1	x_2 , m_2	x_3 , m_3

Value: $x = x_1 + x_2 + x_3$

MAC tag: $\alpha \cdot x = m_1 + m_2 + m_3$

α is a global MAC key, itself secret shared

Properties of SPDZ:

Linear operations
computed locally, no communication

Multiplication
consumes one authenticated Beaver triple (a, b, a · b)

Opening a value
MAC check catches tampering by up to n – 1 parties

Preprocessing
triples are generated in advance and dominate the cost

Secure Table Join

1 Secret-share the inputs
DATA OWNERS

2 Join on a key column
MPC COMPUTING NODES

3 Process the result
MPC COMPUTING NODES

EXAMPLE:

Estonia registry

Full name	DoB	Nat.	Gender
Anna Kask	12-03-91	EST	F
Jaan Tamm	05-11-78	EST	M

Finland registry

Full name	DoB	Nat.	Gender
Anna Kask	12-03-91	EST	F
Mikko Laine	23-08-84	FIN	M

Fixed-width join key

name + DoB + nat. + gender

`Anna Kask····12-03-91ESTF`

`Jaan Tamm····05-11-78ESTM`

`Anna Kask····12-03-91ESTF`

`Mikko Laine··23-08-84FINM`

Joined table

Anna Kask, 12-03-91, EST, F
listed in both registries

The result is secret shared for further aggregation and group by

CYBERNETICA

Secure Join Outline

- Passively secure table join protocol **PrpJoin** (Laur, Talviste, Willemson, ACNS 2013)
- Deterministically but irreversibly encrypt the inputs so matching can be done **locally**
- Encrypt using a random secret-shared key
- Shuffling hides the connection between input rows and output rows



CYBERNETICA

Actively Secure PrpJoin

- Requires actively secure building blocks:
  - **a shuffle**
  - **a keyed pseudorandom permutation (PRP)**
- Leaks only the **number of entries in the join**, the same as the passively secure protocol
  - We consider this leakage acceptable
- Every multiplication on SPDZ shares consumes a **multiplication triple**
  - Generating triples is expensive, so we design protocols to use as few multiplication gates as possible

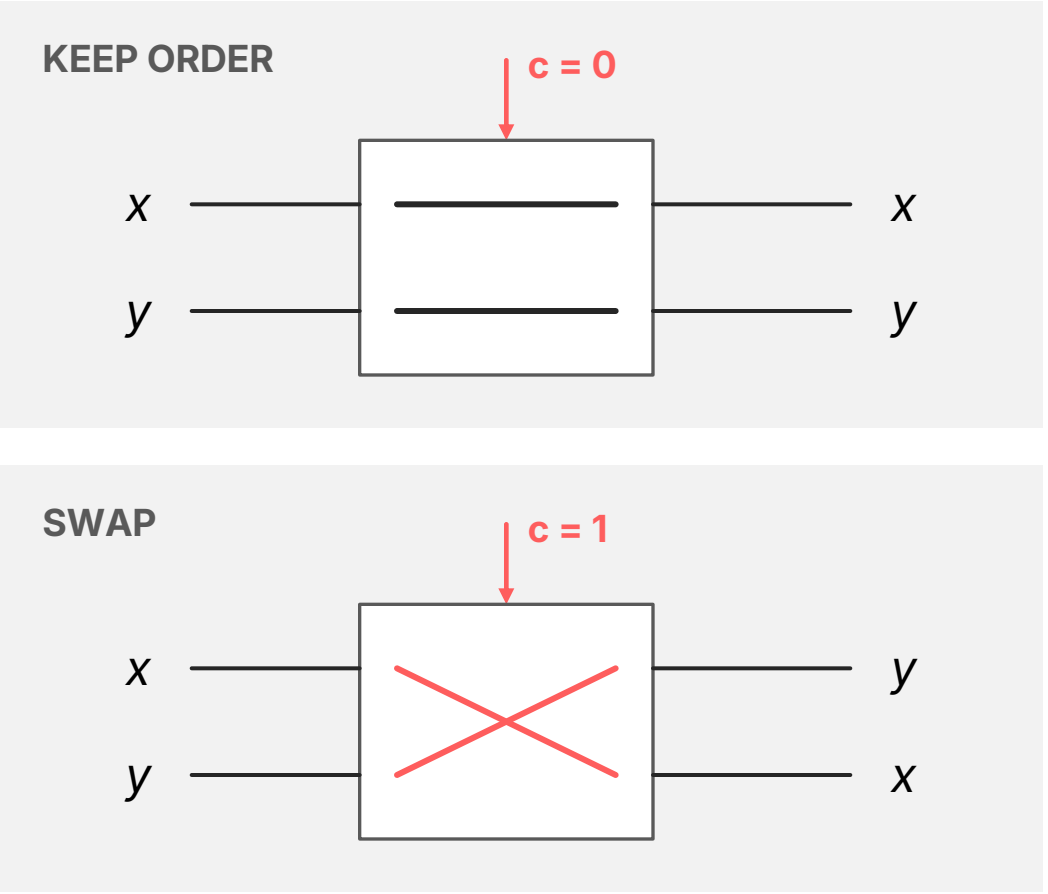
Pseudorandom permutation

- **AES: used in the original PrpJoin**
  - The passively secure PrpJoin evaluated AES on a secret-shared key and input
  - With SPDZ shares, AES would have to be built from $\mathbb{Z}_p$ arithmetic
  - A naive implementation needs about **33,000 secret-shared multiplications** per AES block, which is too inefficient
- **Poseidon2: the replacement**
  - Works natively over $\mathbb{Z}_p$ and needs simpler arithmetic
  - It is a sponge hash function: initializing the sponge with a key turns it into a keyed pseudorandom function
  - For a 64-bit prime *p* and a 4-element input, needs only **344 secret-shared multiplications**

Poseidon2 hash introduced in (Grassi, Khovratovich, Schofnegger, AFRICACRYPT 2023)

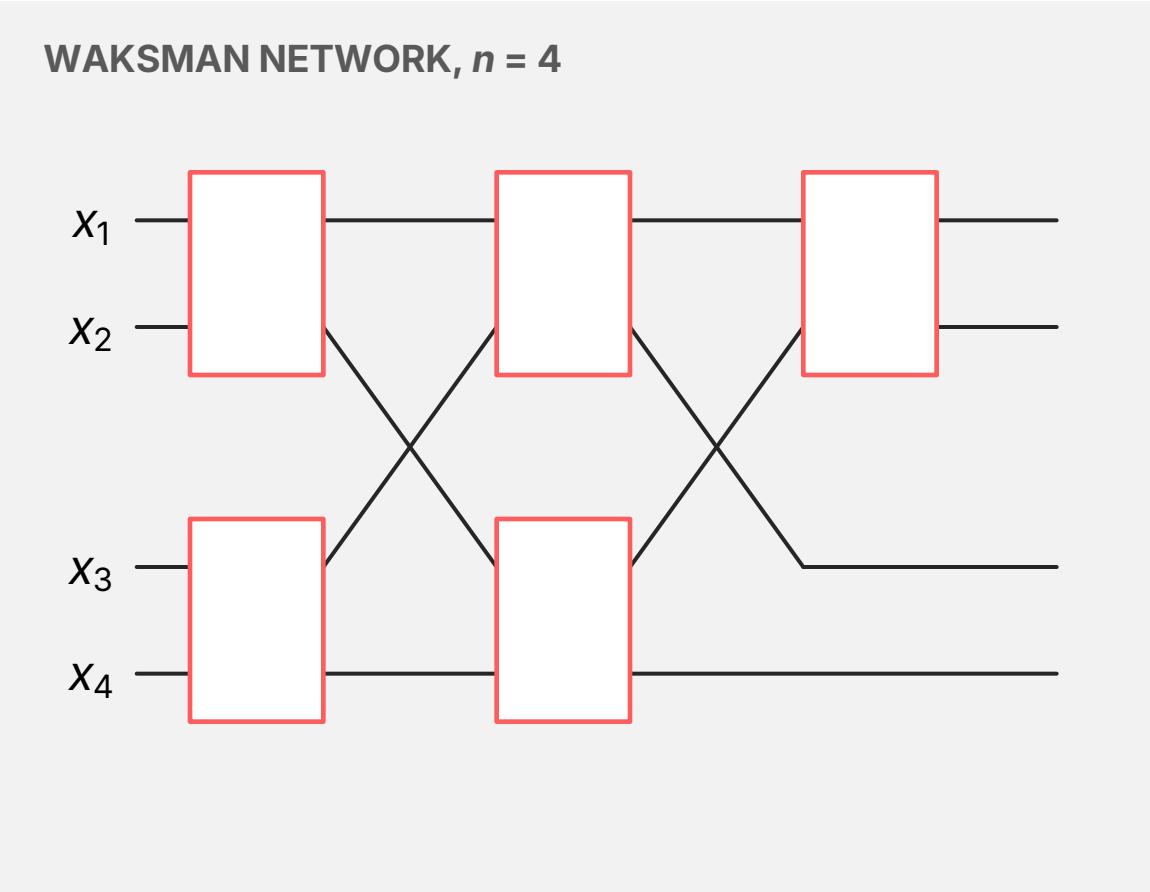
Secure shuffle: the switch box

- A switch box takes **two inputs** and a **control bit**
  - Control bit 0: inputs pass through in the same order
  - Control bit 1: inputs are swapped
- In the shuffle, each control bit is a **random bit**
- Switch boxes are the building block of the shuffle network
- One multiplication per switch box



CYBERNETICA

Secure shuffle: the Waksman network



- A **Waksman network** chains switch boxes to shuffle n inputs
- It uses $n \log_2 n - n + 1$ switch boxes
- It can produce all $n!$ permutations
- With random control bits, permutations are not uniformly random
  - The network has more configurations than there are permutations
  - In practice the distribution is close to uniform, and gets closer as *n* grows

Secure shuffle: computing a switch box

SWITCH BOX FUNCTION

Inputs: secret-shared values $[\![x]\!]$, $[\![y]\!]$ and control bit $[\![b]\!]$ shared over $\mathbb{Z}_p$

1. Compute $[\![d]\!] \leftarrow [\![b]\!] \cdot ([\![y]\!] - [\![x]\!])$
2. Return $([\![x]\!] + [\![d]\!], [\![y]\!] - [\![d]\!])$

Cost: one secure multiplication per switch box

RANDOM BIT IN $\mathbb{Z}_p$

1. Take a random shared value $[\![r]\!]$ from a multiplication triple
2. Compute $[\![u]\!] \leftarrow [\![r]\!] \cdot [\![r]\!]$
3. Open $[\![u]\!]$; it does not reveal the sign of r
4. Locally compute $v \leftarrow u^{-1/2} \bmod p$
5. Compute $[\![s]\!] \leftarrow v \cdot [\![r]\!]$, so $s \in \{-1, 1\}$
6. Return $[\![b]\!] \leftarrow ([\![s]\!] + 1)/2 \in \{0, 1\}$

Cost: one multiplication and one random value

Random bit protocol from (Catrina, de Hoogh, SCN 2010)

Putting the pieces together

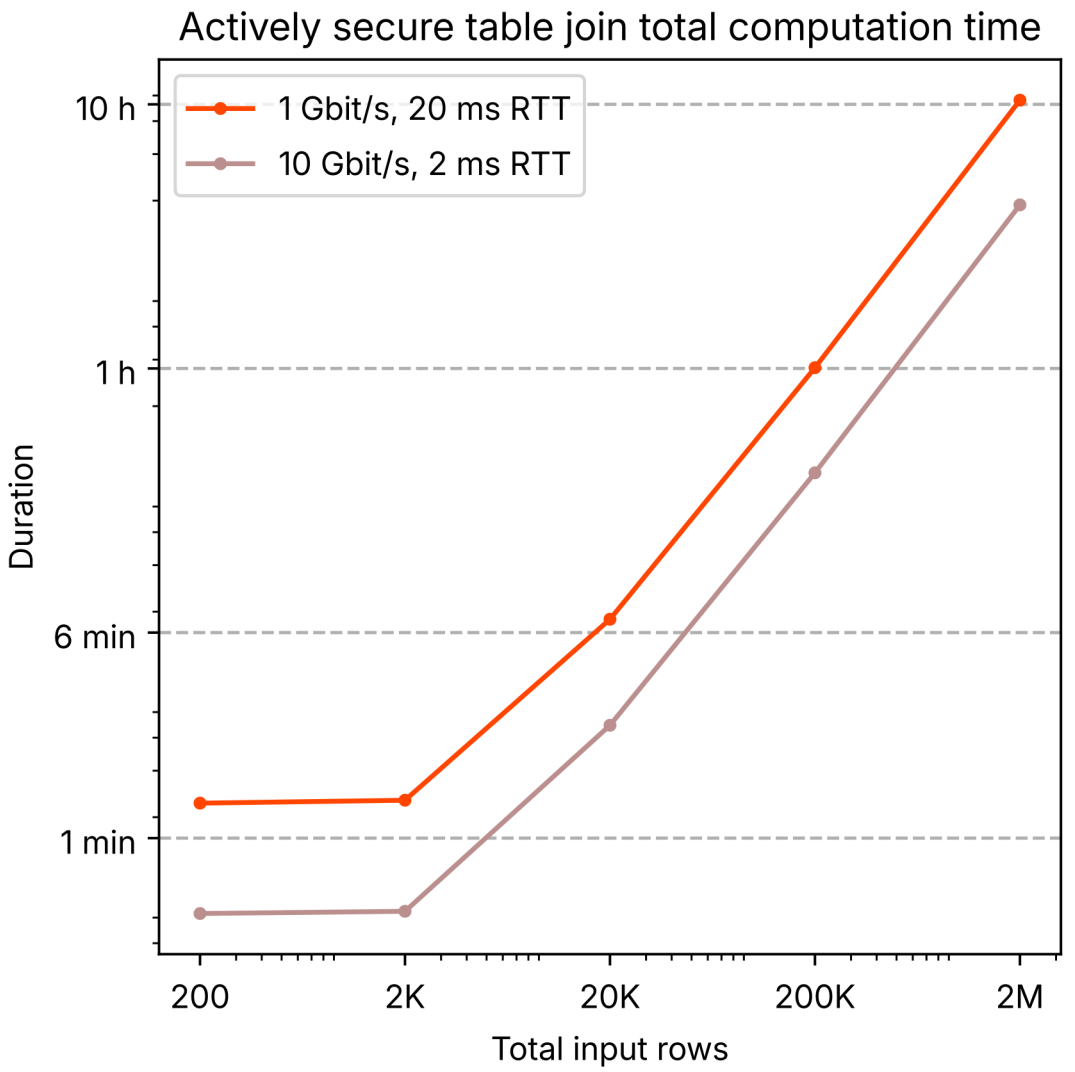
- Opening the secret-shared Poseidon2 hash entails checking all MAC tags
  - If an adversary deviated from the protocol, it would be detected before the hash is revealed
- As long as one party remains honest, the shuffle will be random
- $3 \cdot (n \log_2 n - n + 1)$ triples for shuffle
- Approx. 400 triples per row for keyed hashing
- For joining two tables of 1 million records each, 915 million triples used

Implementation

- The protocols were implemented in the Sharemind MPC framework
- The population register record matching use case was implemented in the SecreC language
- The multiplication triples were generated with the LowGear protocol (Keller, Pastro, Rotaru, EUROCRYPT 2018)
- All technical reports of the JOCONDE project are available online at https://cros.ec.europa.eu/book-page/joconde

Experimental results

- Joining 2 tables with 1M rows each
- With 10 Gbit/s bandwidth, 2 ms RTT took **4 hours 9 min**
  - of which 3 hours 55 min spent generating triples (94%)
  - **915 million triples** were used
  - approx. 4.8 TB of communication per party (~2.4 MB per row)
- With 1 Gbit/s bandwidth, 20 ms RTT took 10 hours 22 min
- Passively secure honest majority version took ~ 1 minute
- If the triples were generated in advance, would use 22 GB of storage at each computing node



CYBERNETICA

Conclusion

- We adapted a passively secure table join protocol to the actively secure dishonest majority setting
- Optimized for least number of multiplications
  - using a non-uniformly random shuffle
  - and $\mathbb{Z}_p$ friendly keyed hash function
- Still, generating multiplication triples takes >90% of the computation time
  - 250× overhead compared to passive security with an honest majority
- Future work:
  - further analysis of the security of the non-uniform shuffle
  - evaluate other options for $\mathbb{Z}_p$ friendly PRP/PRF
  - further engineering effort to improve performance
- Read more at https://cros.ec.europa.eu/book-page/joconde

Thank you

Hendrik Eerikson (hendrik.eerikson@cyber.ee)

https://cyber.ee/
info@cyber.ee
cybernetica
CyberneticaAS
cybernetica_ee
Cybernetica