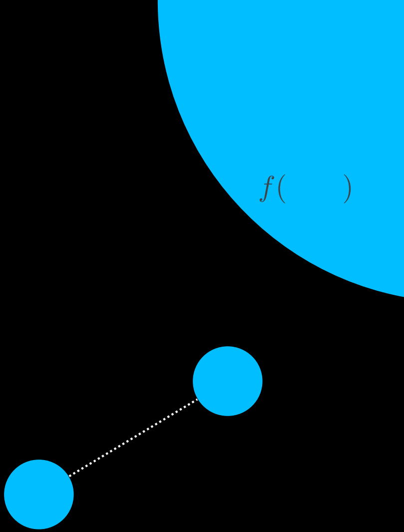
University of Stuttgart
Germany

Marc Rivinius

MPC with Publicly Identifiable Abort from Pseudorandomness and Homomorphic Encryption

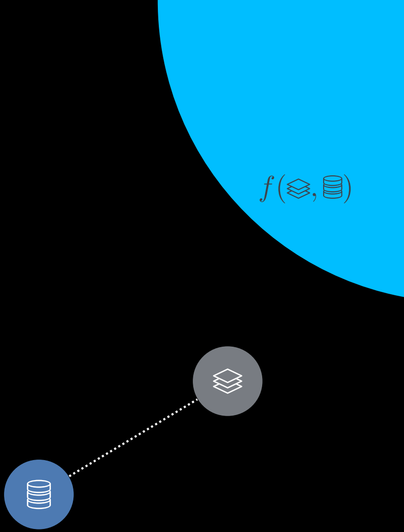
Multiparty Computation

- Multiple parties want to compute a function



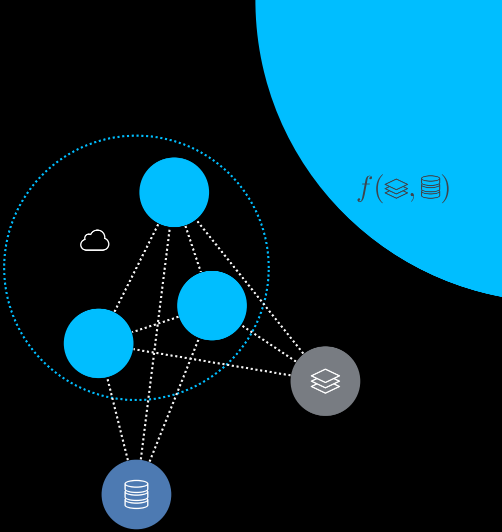
Multiparty Computation

- Multiple parties want to compute a function



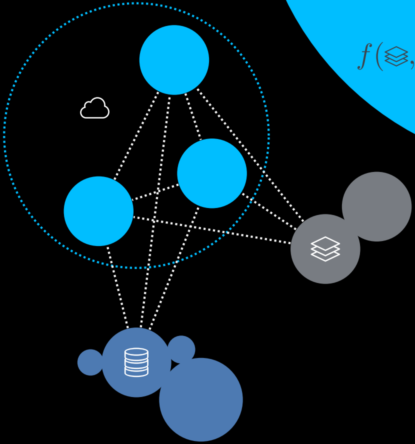
Multiparty Computation

- Multiple parties want to compute a function



Multiparty Computation

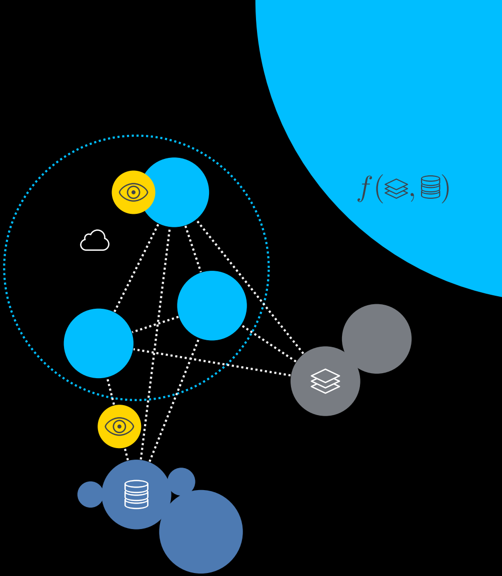
- Multiple parties want to compute a function



$f(\,\cdot\,,\,\cdot\,)$

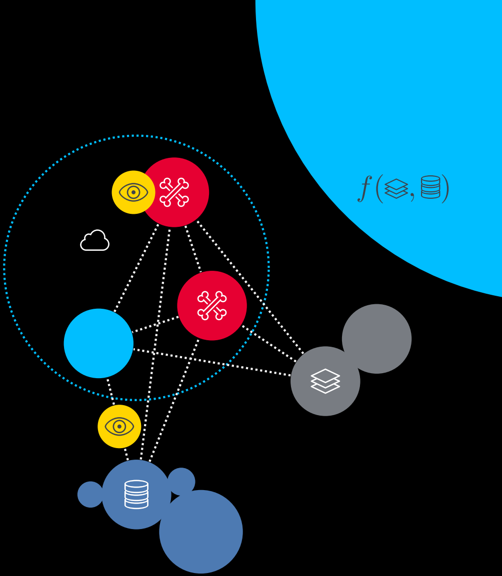
Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest



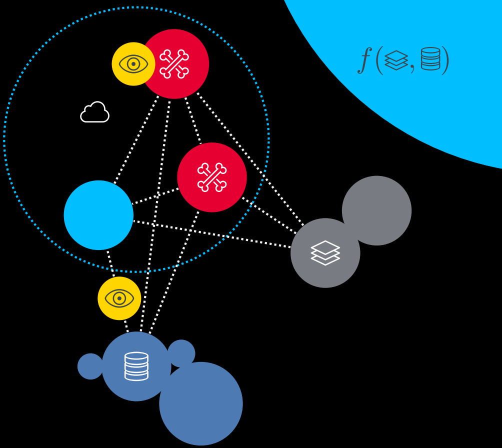
Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - malicious



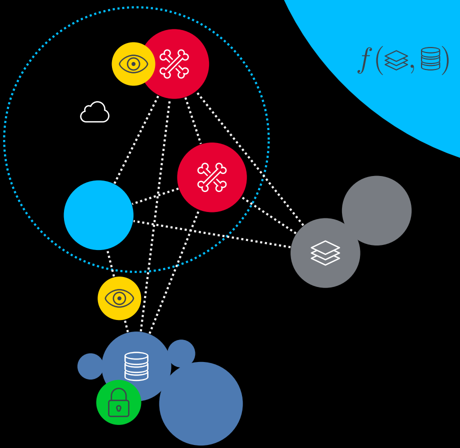
Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - malicious
- Security



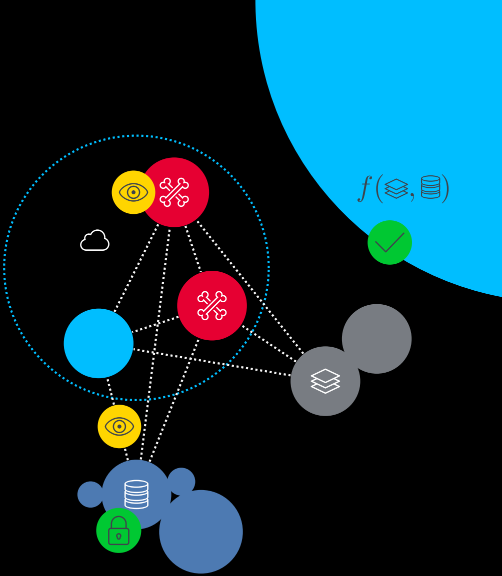
Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - malicious
- Security
  - keep inputs secret



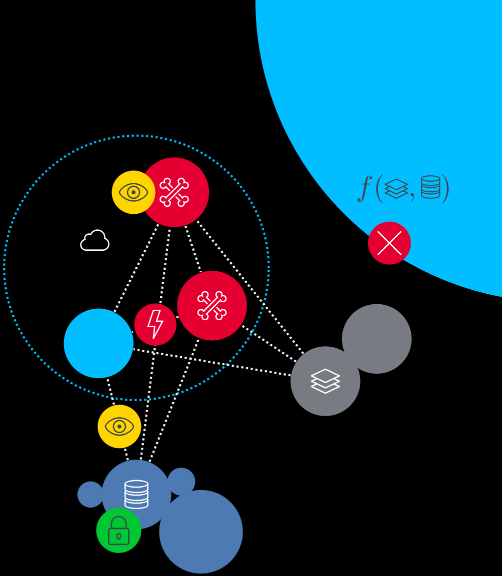
Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - malicious
- Security
  - keep inputs secret
  - correct result (or abort)



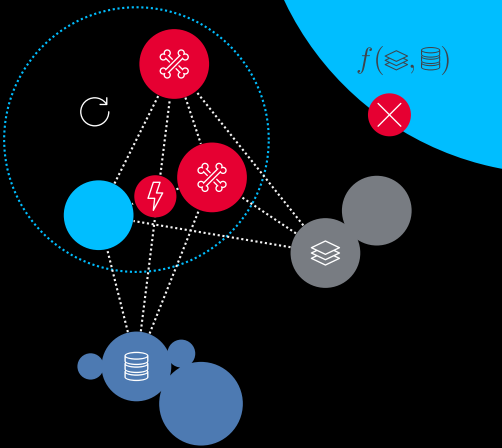
Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)



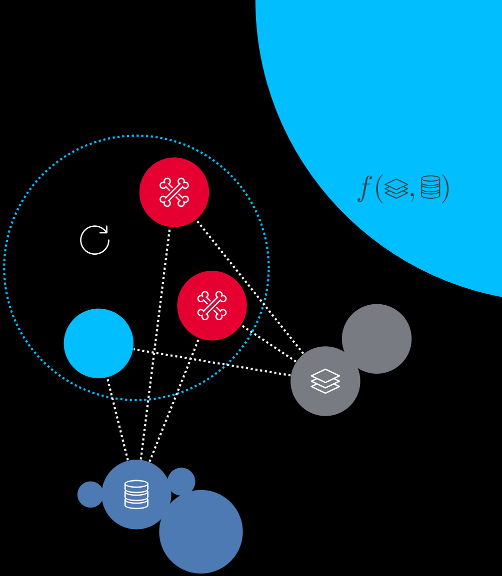
Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)
    → restart?



Multiparty Computation

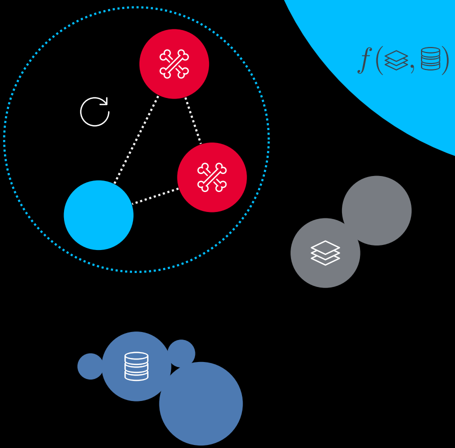
- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)
    → restart?



Multiparty Computation

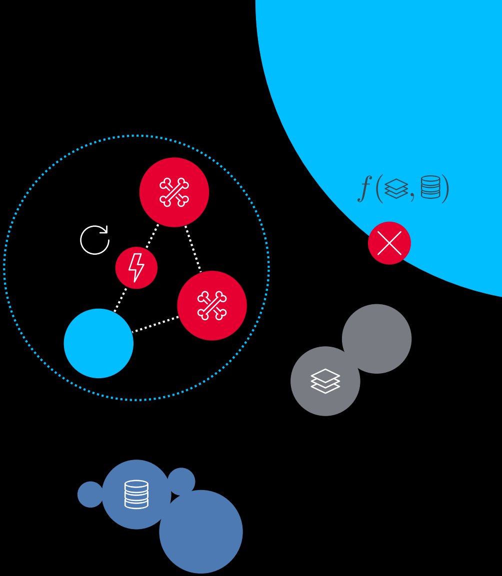
- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)

    → restart?



Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)
    → restart?

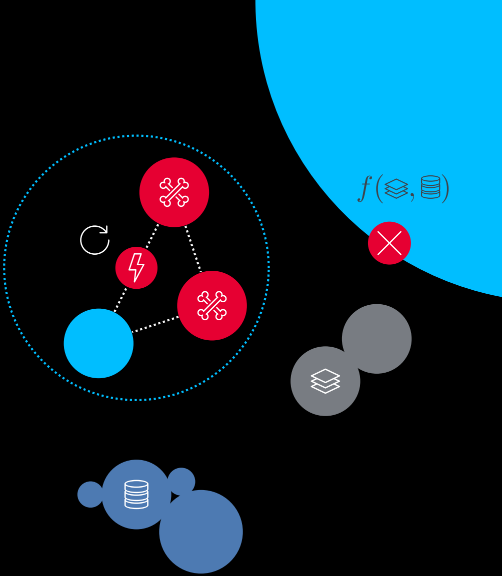


Multiparty Computation

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)

    → restart?

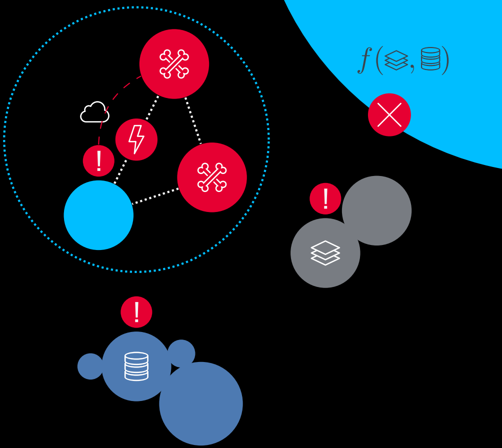
    → restart?



Multiparty Computation with Identifiable Abort

- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)

    → **identifiable abort**



Multiparty Computation with Identifiable Abort

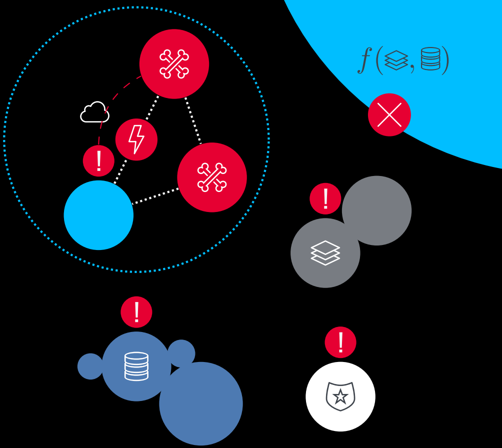
- Multiple parties want to compute a function
- Adversaries
  - semi-honest
  - **malicious**
- Security
  - keep inputs secret
  - correct result (or **abort**)

    → **identifiable abort**

    → **publicly identifiable abort**

    everyone agrees on audit

    {: ok, : !, : ok}



Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions

Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions
- Secret-sharing

Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions
- Secret-sharing
  - share $[x]_i$ of party P_i

Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions
- Secret-sharing
  - share $[x]_i$ of party P_i
  - secret $x = \sum_{i=0}^{n-1} [x]_i$

Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions
- Secret-sharing
  - share $[x]_i$ of party P_i
  - secret $x = \sum_{i=0}^{n-1} [x]_i$
  - linear operations **without interaction**:

    $[x+y]_i = [x]_i + [y]_i$

    $[c \cdot x]_i = c \cdot [x]_i$

    $[x+c]_i = [x]_i + \delta_i \cdot c$

Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions
- Secret-sharing
  - share $[x]_i$ of party P_i
  - secret $x = \sum_{i=0}^{n-1} [x]_i$
  - linear operations **without interaction**:

    $[x + y]_i = [x]_i + [y]_i$

    $[c \cdot x]_i = c \cdot [x]_i$

    $[x + c]_i = [x]_i + \delta_i \cdot c$
- **Offline phase** to generate **correlated** shares

Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions
- Secret-sharing
  - share $[x]_i$ of party P_i
  - secret $x = \sum_{i=0}^{n-1} [x]_i$
  - linear operations **without interaction**:
    $[x + y]_i = [x]_i + [y]_i$
    $[c \cdot x]_i = c \cdot [x]_i$
    $[x + c]_i = [x]_i + \delta_i \cdot c$
- **Offline phase** to generate **correlated** shares
  - random multiplication triples $([a]_i, [b]_i, [c]_i)$ with $c = a \cdot b$

Multiparty Computation: SPDZ [DPSZ12]

- **Efficient** maliciously secure MPC protocol for arithmetic functions
- Secret-sharing
  - share $[x]_i$ of party P_i
  - secret $x = \sum_{i=0}^{n-1} [x]_i$
  - linear operations **without interaction**:

    $[x+y]_i = [x]_i + [y]_i$

    $[c \cdot x]_i = c \cdot [x]_i$

    $[x+c]_i = [x]_i + \delta_i \cdot c$
- **Offline phase** to generate **correlated** shares
  - random multiplication triples $([a]_i, [b]_i, [c]_i)$ with $c = a \cdot b$

  $\rightarrow$ faster computation of the function in the **online phase**

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]

$$\sum_{i=0}^{n-1} [x]_i = x \mapsto \begin{cases} \mathtt{ok} \\ ! \end{cases}$$

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]

$$[x]_i \mapsto \begin{cases} \texttt{ok} \\ ! \end{cases}$$

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]
  → **identifiable abort**

$[x]_i$: ! $\Rightarrow P_i$: !

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]
  $\rightarrow$ **identifiable abort**
- Often not considered

$[x]_i$: ! $\Rightarrow P_i$: !

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]
  → **identifiable abort**

$[x]_i$: ! $\Rightarrow P_i$: !

- Often not considered
  - **publicly identifiable abort**

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]

  → **identifiable abort**

  $[x]_i$: ! $\Rightarrow P_i$: !

- Often not considered
  - **publicly identifiable abort**
  - **client-server setting**

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]
  $\rightarrow$ **identifiable abort**

$[x]_i$: ! $\Rightarrow P_i$: !

- Often not considered
  - **publicly identifiable abort**
  - **client-server setting**
- Online phase: use **linearity** of verification scheme (commitments/MACs) with **linearity of secret-sharing**

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]

  → **identifiable abort**

$$[x]_i\text{: } ! \Rightarrow P_i\text{: } !$$

- Often not considered
  - **publicly identifiable abort**
  - **client-server setting**
- Online phase: use **linearity** of verification scheme (commitments/MACs) with **linearity of secret-sharing**
- Offline phase: pre-generate **random authenticated shares**

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]

  → **identifiable abort**

$[x]_i$: ! $\Rightarrow P_i$: !

- Often not considered
  - **publicly identifiable abort**
  - **client-server setting**
- Online phase: use **linearity** of verification scheme (commitments/MACs) with **linearity of secret-sharing**
- Offline phase: pre-generate **random authenticated shares**
  - build other preprocessing from this

Identifiable Abort MPC

- Make values verifiable, e.g., with MACs [DPSZ12]
- Verification at share-level [BOS16; CFY17; CDKS24; BMRS24]
  $\rightarrow$ **identifiable abort**

$[x]_i$: ! $\Rightarrow P_i$: !

- Often not considered
  - **publicly identifiable abort**
  - **client-server setting**
- Online phase: use **linearity** of verification scheme (commitments/MACs) with **linearity of secret-sharing**
- Offline phase: pre-generate **random authenticated shares**
  - build other preprocessing from this
  - inputs, outputs, multiplications, ...

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

Public-Key [CFY17; RRRK22]

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties

Public-Key [CFY17; RRRK22]

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

Public-Key [CFY17; RRRK22]

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$
- \+ cheap to compute and verify

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$
- \+ cheap to compute and verify
- \+ peer-to-peer communication

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

+ cheap to compute and verify
+ peer-to-peer communication

− usually factor n overhead

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

+ cheap to compute and verify
+ peer-to-peer communication

− usually factor n overhead
− unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

\+ cheap to compute and verify

\+ peer-to-peer communication

– usually factor n overhead

– unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

\+ easy: publicly identifiable abort

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

\+ cheap to compute and verify

\+ peer-to-peer communication

– usually factor n overhead

– unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

\+ easy: publicly identifiable abort

\+ easy: for client-server setting

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

$+$ cheap to compute and verify

$+$ peer-to-peer communication

$-$ usually factor n overhead

$-$ unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

$+$ easy: publicly identifiable abort

$+$ easy: for client-server setting

$+$ good asymptotic complexity

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$
- \+ cheap to compute and verify
- \+ peer-to-peer communication
- – usually factor n overhead
- – unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**
- \+ easy: publicly identifiable abort
- \+ easy: for client-server setting
- \+ good asymptotic complexity
- – broadcast communication

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$
- \+ cheap to compute and verify
- \+ peer-to-peer communication
- – usually factor n overhead
- – unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**
- \+ easy: publicly identifiable abort
- \+ easy: for client-server setting
- \+ good asymptotic complexity
- – broadcast communication
- – usually extra ZKPs

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

\+ cheap to compute and verify

\+ peer-to-peer communication

– usually factor n overhead

– unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

\+ easy: publicly identifiable abort

\+ easy: for client-server setting

\+ good asymptotic complexity

– broadcast communication

– usually extra ZKPs

– expensive to compute and verify

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$

+ cheap to compute and verify
+ peer-to-peer communication

– usually factor n overhead
– unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**

+ easy: publicly identifiable abort
+ easy: for client-server setting
+ good asymptotic complexity

– broadcast communication
– usually extra ZKPs
– expensive to compute and verify

→ our work: global (non-pairwise) MACs

Identifiable Abort Techniques

MAC-based [BOS16; CDKS24; BMRS24]

- usually **pairwise** between parties
- usually **linear** $\alpha \cdot [x]_i + \rho$
- \+ cheap to compute and verify
- \+ peer-to-peer communication
- – usually factor n overhead
- – unclear: client-server setting

Public-Key [CFY17; RRRK22]

- usually **linear commitments**
- \+ easy: publicly identifiable abort
- \+ easy: for client-server setting
- \+ good asymptotic complexity
- – broadcast communication
- – usually extra ZKPs
- – expensive to compute and verify

→ our work: global (non-pairwise) MACs

→ combines advantages of both methods

Publicly Identifiable Authenticated Secret-Sharing

$$\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \ := \ \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

Publicly Identifiable Authenticated Secret-Sharing

$$\underbrace{}_{}\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) := \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

MAC key

share

PRF key

context, e.g., `counter++`

Publicly Identifiable Authenticated Secret-Sharing

$$\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \;:=\; \underbrace{\alpha}_{\text{MAC key}} \cdot \underbrace{[x]_i}_{\text{share}} + \mathsf{PRF}(\underbrace{\mathsf{fk}}_{\text{PRF key}}, \underbrace{\mathsf{ctx}}_{\text{context, e.g., counter++}})$$

$$\mathsf{MACCheck}(\alpha, \mathsf{fk}, [x]_i, \tau, \mathsf{ctx}) \;:=\; \begin{cases} \mathtt{ok} & \text{if } \tau = \mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \\ ! & \text{otherwise} \end{cases}$$

Publicly Identifiable Authenticated Secret-Sharing

$$\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \;:=\; \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

MAC key

share

PRF key

context, e.g.,
`counter++`

$$\mathsf{MACCheck}(\alpha, \mathsf{fk}, [x]_i, \tau, \mathsf{ctx}) \;:=\; \begin{cases} \texttt{ok} & \text{if } \tau = \mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \\ ! & \text{otherwise} \end{cases}$$

→ equivalent to the MAC of [CF13] and similar to what is used in [BMRS24]

Publicly Identifiable Authenticated Secret-Sharing

$$\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \;:=\; \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\Sigma$$

$$[\alpha]_0 \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_0, \mathsf{ctx}) \quad + \quad \ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots$$

Publicly Identifiable Authenticated Secret-Sharing

$$\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \;:=\; \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\Sigma$$

$$[\alpha]_0 \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_0, \mathsf{ctx}) \quad + \quad \dots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \dots$$

computed by P_0

computed by P_j

Publicly Identifiable Authenticated Secret-Sharing

$$\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \;:=\; \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

Σ

$$[\alpha]_0 \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_0, \mathsf{ctx}) \quad + \quad \ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots$$

Dec

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

Publicly Identifiable Authenticated Secret-Sharing

$$\mathsf{MAC}(\alpha, \mathsf{fk}, [x]_i, \mathsf{ctx}) \quad := \quad \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

Σ

$$[\alpha]_0 \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_0, \mathsf{ctx}) \quad + \quad \dots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \dots$$

Dec

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

encrypted by P_i encrypted by P_j

Verification

$$\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\Sigma$$

$$\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots$$

$$\mathsf{Dec}$$

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

Verification

- MAC check only requires

$$\begin{gathered}
\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx}) \\
\Big\uparrow \Sigma \\
\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots \\
\Big\uparrow \mathsf{Dec} \\
[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))
\end{gathered}$$

Verification

- MAC check only requires
  - MAC key α

$$\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\Big\uparrow \Sigma$$

$$\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots$$

$$\Big\uparrow \mathsf{Dec}$$

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

Verification

- MAC check only requires
  - MAC key α
  - PRF key fk

$$
\begin{tikzcd}
\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx}) \\
\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots \arrow[u, "\Sigma"] \\
[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx})) \arrow[u, "\mathsf{Dec}"]
\end{tikzcd}
$$

Verification

- MAC check only requires
  - MAC key α
  - PRF key fk
- MAC tag generation requires

$$\begin{tikzcd}
\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx}) \\
\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots \arrow[u, "\Sigma"] \\
[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx})) \arrow[u, "\mathsf{Dec}"]
\end{tikzcd}$$

Verification

- MAC check only requires
  - MAC key α
  - PRF key fk
- MAC tag generation requires
  - MAC key share $[\alpha]_j$

$$\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\uparrow \Sigma$$

$$\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots$$

$$\uparrow \mathsf{Dec}$$

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

Verification

- MAC check only requires
  - MAC key α
  - PRF key fk
- MAC tag generation requires
  - MAC key share $[\alpha]_j$
  - partial PRF key fk_j

$$\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots$$

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

Verification

- MAC check only requires
  - MAC key α
  - PRF key fk
- MAC tag generation requires
  - MAC key share $[\alpha]_j$
  - partial PRF key fk_j
  - encryption randomness that hides $[\alpha]_j$ and fk_j

$$\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\Sigma$$

$$\ldots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \ldots$$

$$\mathsf{Dec}$$

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

Verification

- MAC check only requires
  - MAC key α
  - PRF key fk
- MAC tag generation requires
  - MAC key share $[\alpha]_j$
  - partial PRF key fk_j
  - encryption randomness that hides $[\alpha]_j$ and fk_j

→ Verification only requires **3 field elements*** per party for any number of tags

$$\alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}, \mathsf{ctx})$$

$$\Big\uparrow \Sigma$$

$$\dots \quad + \quad [\alpha]_j \cdot [x]_i + \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}) \quad + \quad \dots$$

$$\Big\uparrow \mathsf{Dec}$$

$$[\alpha]_j \cdot \mathsf{Enc}(\mathsf{pk}_i, [x]_i) + \mathsf{EncDrown}(\mathsf{pk}_i, \mathsf{PRF}(\mathsf{fk}_j, \mathsf{ctx}))$$

*plus other information to decommit commitments

Protocol Overview

Setup

Protocol Overview

Setup

Offline phase

Protocol Overview

Generate tags with α

Setup

Offline phase

Protocol Overview

Generate tags with α

Setup

Offline phase

Online phase

Protocol Overview

Setup	Offline phase	Online phase
	Generate tags with α	Publish tagged values

Protocol Overview

Publish α, fk

Setup	Offline phase	Online phase
	Generate tags with α	Publish tagged values

Protocol Overview

Publish α, fk

Setup	Offline phase	Online phase	Verify offline phase
	Generate tags with α	Publish tagged values	

Protocol Overview

Publish α, fk

Generate tags with α

Publish tagged values

$c \stackrel{?}{=} [\alpha]_j \cdot \mathsf{Enc}([x]_i) + \mathsf{EncDrown}(\mathsf{PRF}(\mathsf{fk}_j))$

Setup

Offline phase

Online phase

Verify offline phase

Protocol Overview

Publish α, fk

Generate tags with α

Publish tagged values

$$c \stackrel{?}{=} [\alpha]_j \cdot \mathsf{Enc}([x]_i) + \mathsf{EncDrown}(\mathsf{PRF}(\mathsf{fk}_j))$$

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Protocol Overview

Publish α, fk

Generate tags with α

Publish tagged values

$\tau \stackrel{?}{=} \alpha \cdot [x]_i + \mathrm{PRF}(\mathrm{fk})$

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Protocol Overview

Publish α, fk

Generate tags with α

Publish tagged values

$\tau \stackrel{?}{=} \alpha \cdot [x]_i + \mathsf{PRF}(\mathsf{fk})$

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Output phase

Protocol Overview

Publish α, fk

Generate tags with α

Publish tagged values

Publish tagged output

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Output phase

Protocol Overview

Publish α, fk

Generate output tags with ω

Generate tags with α

Publish tagged values

Publish tagged output

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Output phase

Protocol Overview

Publish α, fk

Publish ω, fk$'$

Generate output tags with ω

Generate tags with α

Publish tagged values

Publish tagged output

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Output phase

Protocol Overview

Publish α, fk

Publish ω, fk$'$

Generate output tags with ω

Generate tags with α

Publish tagged values

Publish tagged output

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Output phase

Verify offline phase

Protocol Overview

Publish α, fk

Publish ω, fk$'$

Generate output tags with ω

$c \stackrel{?}{=} [\omega]_j \cdot \mathsf{Enc}([x]_i) + \mathsf{EncDrown}(\mathsf{PRF}(\mathsf{fk}'_j))$

Generate tags with α

Publish tagged values

Publish tagged output

Setup

Offline phase

Online phase

Verify offline phase

Verify online phase

Output phase

Verify offline phase

Protocol Overview

Publish α, fk

Publish ω, fk$'$

Generate output tags with ω

$$c \stackrel{?}{=} [\omega]_j \cdot \mathsf{Enc}([x]_i) + \mathsf{EncDrown}(\mathsf{PRF}(\mathsf{fk}'_j))$$

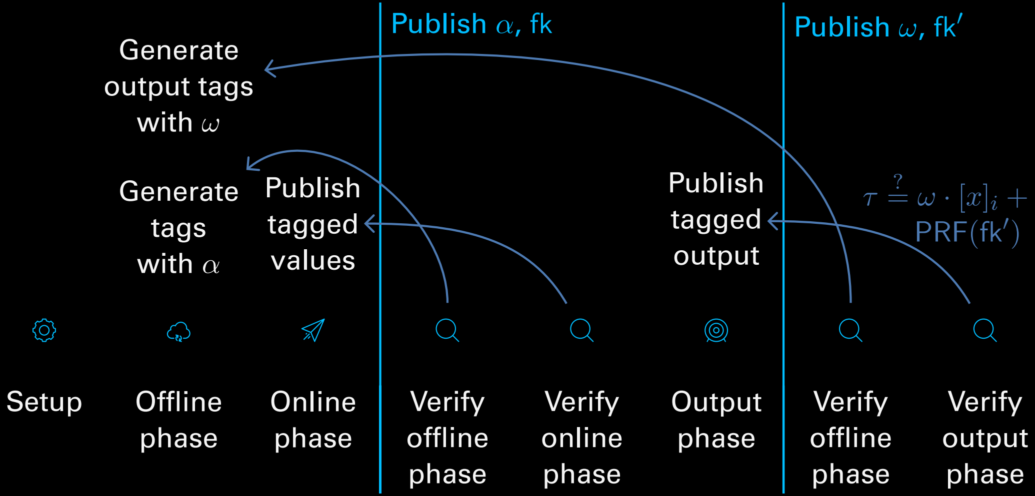
Generate tags with α

Publish tagged values

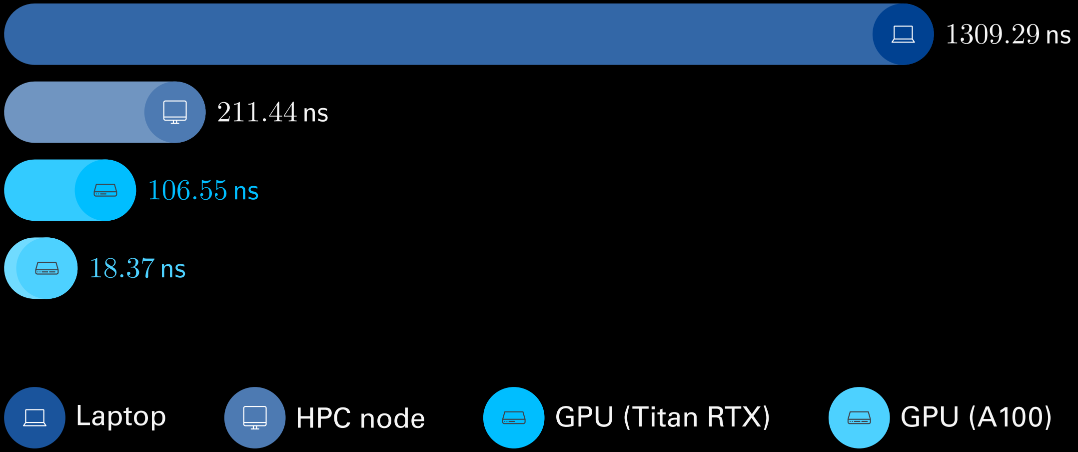
Publish tagged output

Setup | Offline phase | Online phase | Verify offline phase | Verify online phase | Output phase | Verify offline phase | Verify output phase

Protocol Overview



Evaluation: Homomorphic Encryption Verification

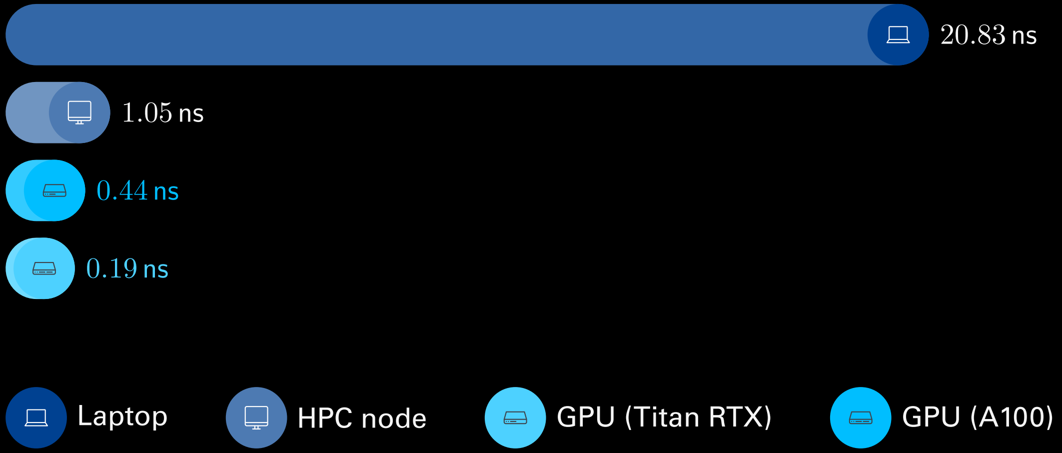


Evaluation: MAC Verification

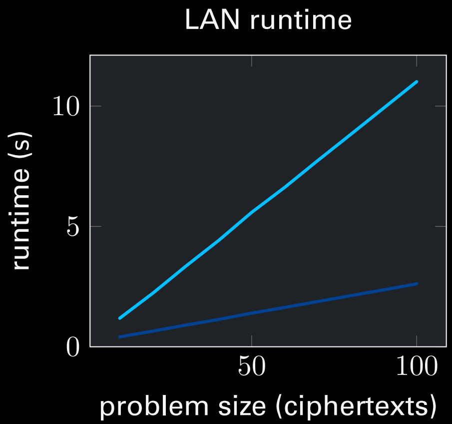
- 20.83 ns
- 1.05 ns
- 0.44 ns
- 0.19 ns

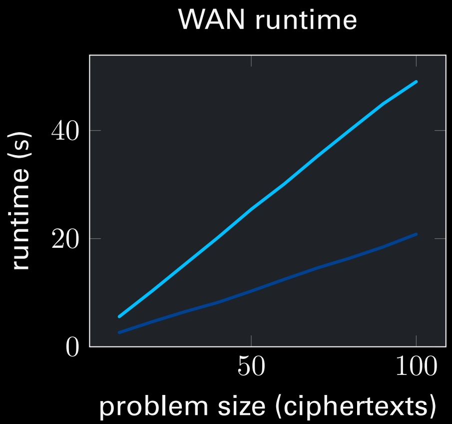
Laptop | HPC node | GPU (Titan RTX) | GPU (A100)

Evaluation: MAC Verification



Evaluation: Multiplication (Online)

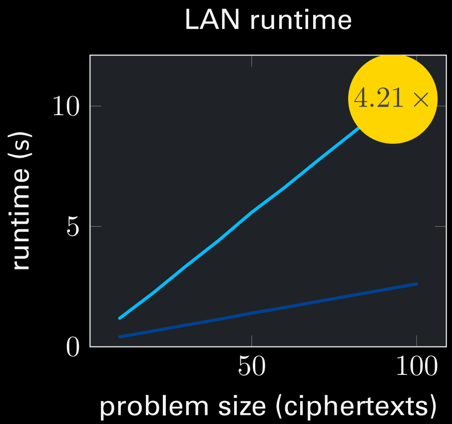


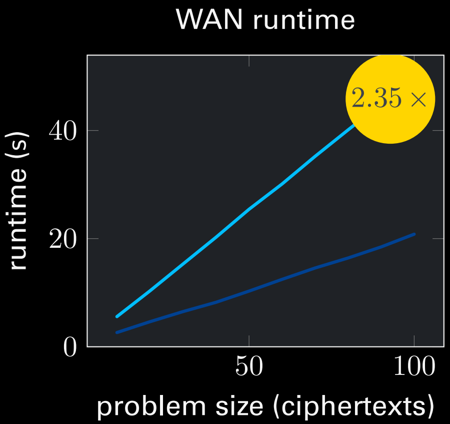


SPDZ (server)

Ours (server)

Evaluation: Multiplication (Online)



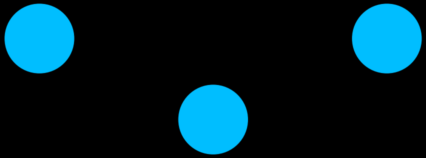


SPDZ (server)

Ours (server)

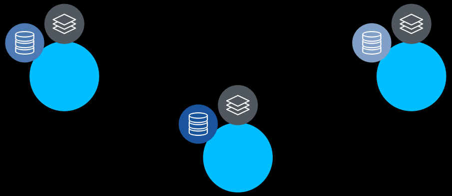
Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model



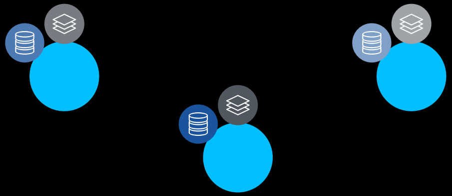
Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model



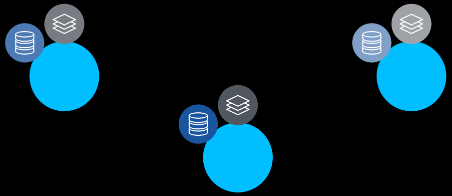
Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data



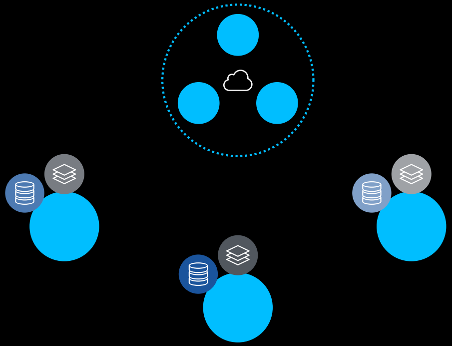
Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely



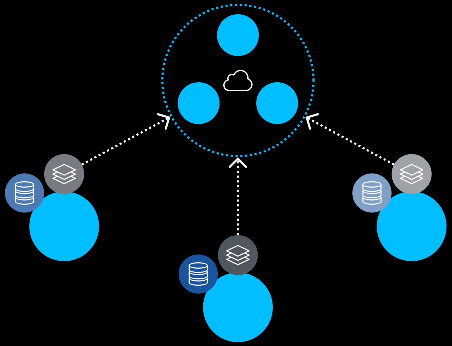
Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely
  - e.g., via MPC servers



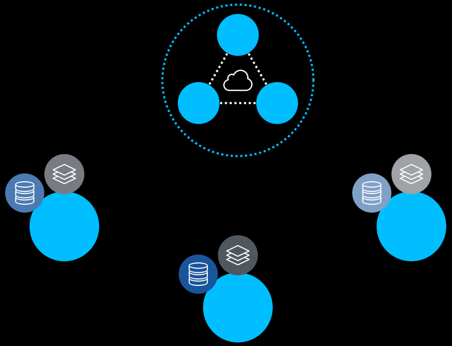
Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely
  - e.g., via MPC servers



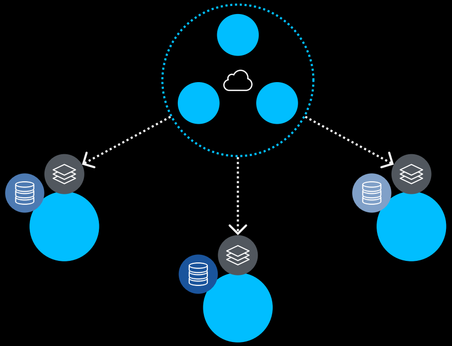
Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely
  - e.g., via MPC servers



Use-Case: Federated Learning with Secure Aggregation

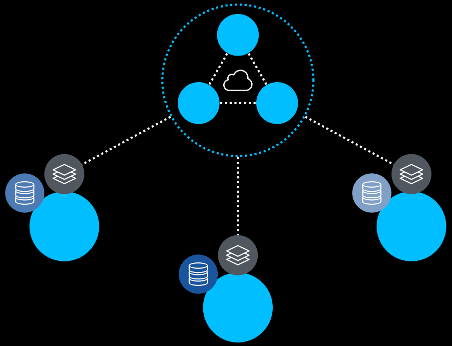
- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely
  - e.g., via MPC servers
- Obtain updated model



Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely
  - e.g., via MPC servers
- Obtain updated model

Repeat

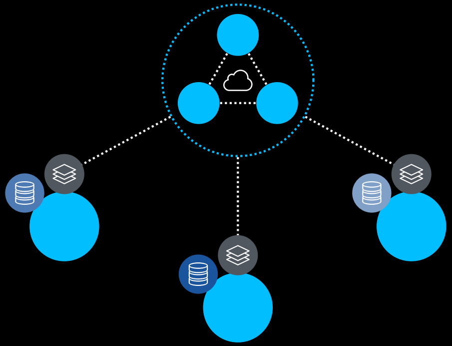


Use-Case: Federated Learning with Secure Aggregation

- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely
  - e.g., via MPC servers
- Obtain updated model

⟳ Repeat

→ **publicly identifiable abort** for clients

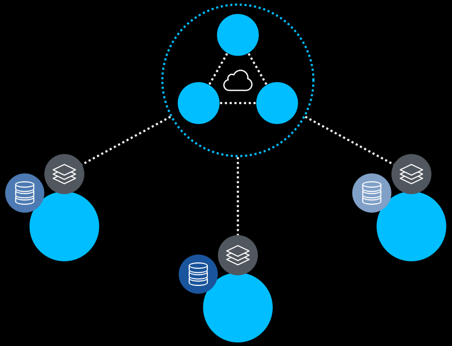


Use-Case: Federated Learning with Secure Aggregation

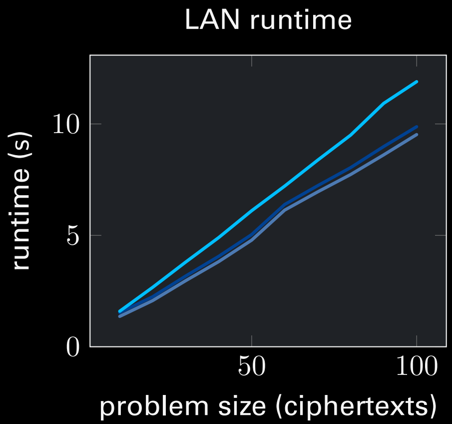
- Parties want to **train** an ML model
  - on private data
  - sharing the same model
- Parties train model with local data
- Aggregate model (gradients) securely
  - e.g., via MPC servers
- Obtain updated model
- Repeat

→ **publicly identifiable abort** for clients

→ **more efficient client I/O** protocols

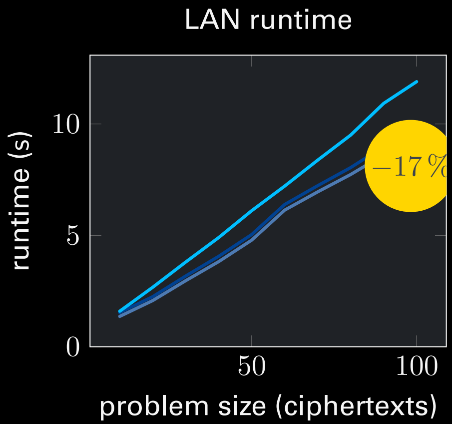


Evaluation: Secure Aggregation (Online)



SPDZ (server) SPDZ (client) Ours (client / server)

Evaluation: Secure Aggregation (Online)

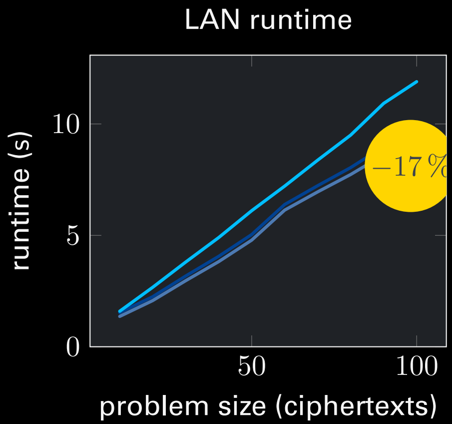


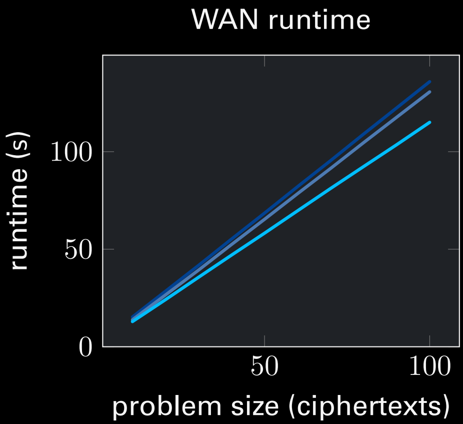
SPDZ (server)

SPDZ (client)

Ours (client / server)

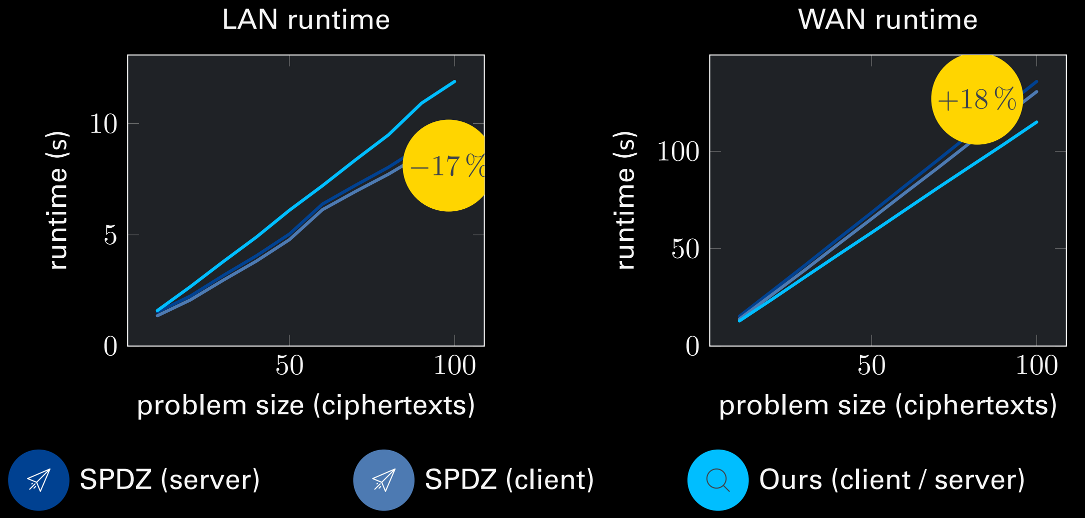
Evaluation: Secure Aggregation (Online)



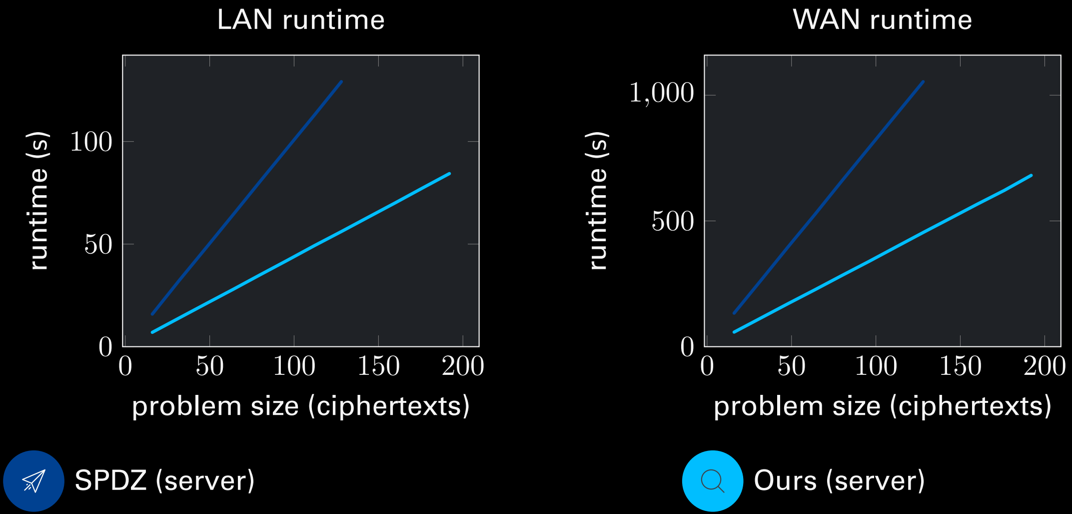


SPDZ (server) SPDZ (client) Ours (client / server)

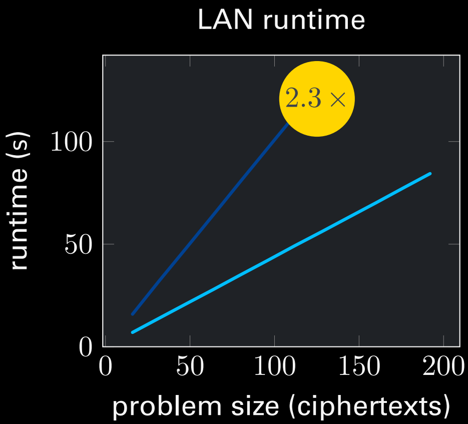
Evaluation: Secure Aggregation (Online)

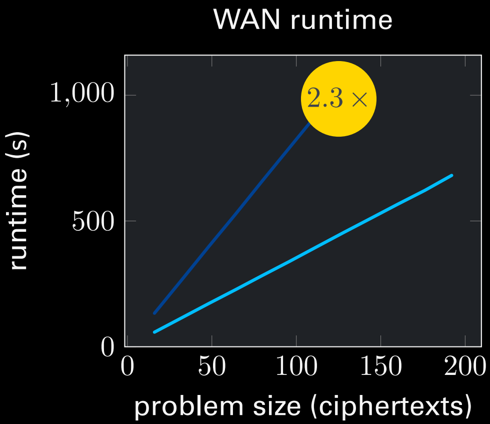


Evaluation: Secure Aggregation (Offline)



Evaluation: Secure Aggregation (Offline)





SPDZ (server)

Ours (server)

Conclusion

- First MPC protocol using global MACs for individual shares

Conclusion

- First MPC protocol using global MACs for individual shares
  - **publicly identifiable abort**

Conclusion

- First MPC protocol using global MACs for individual shares
  - **publicly identifiable abort**
  - **client-server** computations

Conclusion

- First MPC protocol using global MACs for individual shares
  - **publicly identifiable abort**
  - **client-server** computations
- **Same complexity** as HE-based SPDZ

Conclusion

- First MPC protocol using global MACs for individual shares
  - **publicly identifiable abort**
  - **client-server** computations
- **Same complexity** as HE-based SPDZ
- **Low concrete overhead** compared to SPDZ

Conclusion

- First MPC protocol using global MACs for individual shares
  - **publicly identifiable abort**
  - **client-server** computations
- **Same complexity** as HE-based SPDZ
- **Low concrete overhead** compared to SPDZ
- Efficient input/output for **outsourced computation**

References

[BMRS24] Carsten Baum, Nikolas Melissaris, Rahul Rachuri, and Peter Scholl. "Cheater Identification on a Budget: MPC with Identifiable Abort from Pairwise MACs". In: *Crypto 2024*. 2024, pp. 454–488. DOI: 10.1007/978-3-031-68397-8_14.

[BOS16] Carsten Baum, Emmanuela Orsini, and Peter Scholl. "Efficient Secure Multiparty Computation with Identifiable Abort". In: *TCC 2016-B*. 2016, pp. 461–490. DOI: 10.1007/978-3-662-53641-4_18.

[CDKS24] Ran Cohen, Jack Doerner, Yashvanth Kondi, and Abhi Shelat. "Secure Multiparty Computation with Identifiable Abort via Vindicating Release". In: *Crypto 2024*. 2024, pp. 36–73. DOI: 10.1007/978-3-031-68397-8_2.

[CF13] Dario Catalano and Dario Fiore. "Practical Homomorphic MACs for Arithmetic Circuits". In: *Eurocrypt 2013*. 2013, pp. 336–352. DOI: 10.1007/978-3-642-38348-9_21.

[CFY17] Robert K. Cunningham, Benjamin Fuller, and Sophia Yakoubov. "Catching MPC Cheaters: Identification and Openability". In: *ICITS 2017*. 2017, pp. 110–134. DOI: 10.1007/978-3-319-72089-0_7.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *Crypto 2012*. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[RRRK22] Marc Rivinius, Pascal Reisert, Daniel Rausch, and Ralf Küsters. "Publicly Accountable Robust Multi-Party Computation". In: *SP 2022*. 2022, pp. 2430–2449. DOI: 10.1109/SP46214.2022.9833608.

[Icons] from 1001FreeDownloads.com

University of Stuttgart
Germany

Thank you!

Marc Rivinius
Institute of Information Security (SEC)

E-Mail marc.rivinius@sec.uni-stuttgart.de

Paper eprint.iacr.org/2025/258
Code github.com/sec-stuttgart/pia-mpc

Appendix: Online Phase

- Linear operations: directly on shares and tags
- Multiplication of x and y using triple (a, b, c)
  - Open $[\![u]\!] = [\![x - a]\!]$ and $[\![v]\!] = [\![y - b]\!]$
  - Use $[\![x \cdot y]\!] = [\![c]\!] + [\![a]\!] \cdot v + u \cdot [\![b]\!] + u \cdot v \cdot \delta_i$
- Open
  - Broadcast $[x]$ and symmetric encryption to tag of $[x]$
  - Use $x = \sum_{i=0}^{n-1} [x]_i$

Appendix: Input/Output

- Input
  - Servers use random $[\![r]\!]$ from the offline phase
  - Servers open $[\![r]\!]$ to input party
  - Input party broadcasts $u = x - r$ for input x
  - Servers use $[\![x]\!] = [\![r]\!] + \delta_i \cdot u$
- Output
  - Servers use double authenticated random $[\![r]\!]$ from offline phase
  - Servers open $[\![u]\!] = [\![x - r]\!]$ for output x under α
  - Verification using α
  - Servers open $[\![r]\!]$ under ω
  - Verification using ω