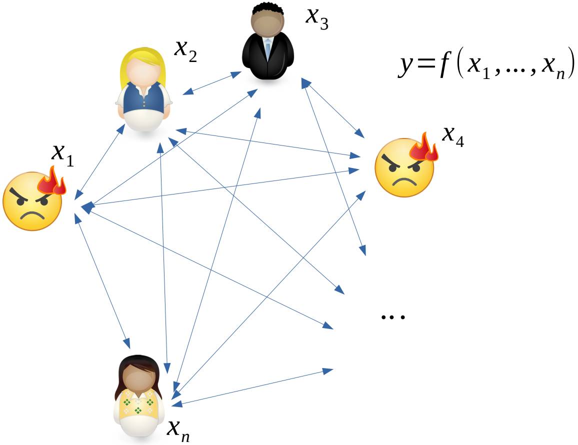
A Secret-Sharing Based MPC Protocol for Boolean Circuits with Good Amortized Complexity

Ignacio Cascudo [1] Jaron Skovsted Gundersen [2]

[1] IMDEA Software Institute [2] Aalborg University

TCC, 18 November 2020

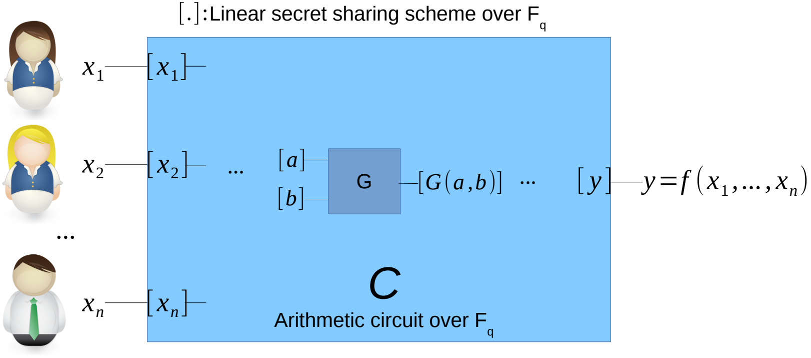
Secure multiparty computation (MPC)



- Private inputs $x_1, \dots, x_n$.
- Goal: compute $y = f(x_1, \dots, x_n)$.
- Secure channel between each pair.

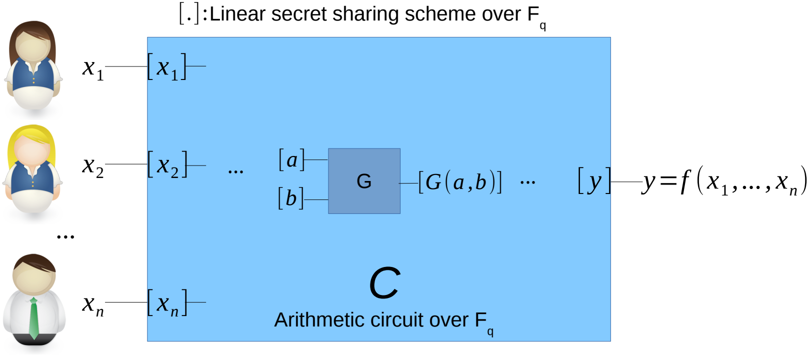
- Adversary corrupts a set of parties.
- Obtains no info. on honest x_i (beyond that implied by y and corrupted x_i).
- Can only alter computation of y by changing corrupted x_i.

Secret-sharing based MPC



- Function represented by arithmetic circuit over some finite field $\mathbb{F}_q$.
- Parties secret-share inputs.
- Gate-by-gate computation ($[a], [b] \rightarrow [G(a, b)]$)
  - Linear gates: using linearity of secret sharing.
  - Multiplication gates: Dedicated subprotocol.

Secret-sharing based MPC



- Function represented by arithmetic circuit over some finite field $\mathbb{F}_q$.
- Parties secret-share inputs.
- Gate-by-gate computation ($[a], [b] \to [G(a, b)]$)
  - Linear gates: using linearity of secret sharing.
  - Multiplication gates: Dedicated subprotocol.

Large finite fields

Many secret-sharing-based MPC protocols need large finite fields $\mathbb{F}_q$:

Large finite fields

Many secret-sharing-based MPC protocols need large finite fields $\mathbb{F}_q$:

- Honest majority, info th. security: BGW-style protocols use Shamir's secret sharing scheme (requires $q > n$).
- Dishonest majority, computational security (**this work**): Protocols such as SPDZ achieve active security via (linear homomorphic) message authentication codes (requires $q > 2^\lambda$, for sec. parameter λ).

SPDZ (Damgård et al., 2012)

- Over field $\mathbb{F}_q$, actively secure for dishonest majority ($(n-1)$ out of n)
- One additively shared global key $\alpha = \alpha_1 + \cdots + \alpha_n \in \mathbb{F}_q$
- Every data $x \in \mathbb{F}_q$ has MAC $m = \alpha \cdot x$.
- i-th party holds additive share α_i for α, x_i for every x, and m_i for every $\alpha \cdot x$.
- $[x] = \big((x_1, \ldots, x_n), (m_1, \ldots, m_n)\big)$, where $x = \sum x_i$, and $\alpha \cdot x = \sum m_i$.
- If adversary tries to output $x' = x + e$ instead of x, ($e \neq 0$), and introduces a MAC error Δ, then:

$$\alpha \cdot x' = m + \Delta \Leftrightarrow \alpha \cdot e = \Delta$$

So probability that adversary is not caught when opening is $1/|\mathbb{F}_q|$ (**too large for small** q**!**).

“SPDZ for small fields”

Our goal: A version of SPDZ for arithmetic circuit over $\mathbb{F}_q$, q small (from now on $q = 2$).

- Naive idea: Since $\mathbb{F}_2 \subseteq \mathbb{F}_{2^m}$ one could just use SPDZ over $\mathbb{F}_{2^m}$ for large enough m.
  - Problem: wasteful, m bits to represent one, + input ZK-validation
- Next idea: does bundling data in batches of k bits, $\mathbf{x} \in \mathbb{F}_2^k$, and MACing them together help?
- Using a coordinatewise MAC $\boldsymbol{\alpha} * \mathbf{x} = \mathbf{m}$ (where $\boldsymbol{\alpha} \in \mathbb{F}_2^k$) does **not** help.
  - Adversary can add error in one coordinate, succeed w.p. 1/2 (i.e. guessing one coordinate of $\boldsymbol{\alpha}$)

MiniMAC (Damgård/Zakarias, 2013)

A solution: MiniMAC (Damgård/Zakarias, 2013):

- Encode $\mathbf{x} \in \mathbb{F}_2^k$ with an error correcting code, $\mathbf{x} \to C(\mathbf{x}) \in \mathbb{F}_2^\ell$
- Use the MAC above on $C(\mathbf{x})$ (i.e. $\boldsymbol{\alpha} * C(\mathbf{x}) = \mathbf{m}$ where $\boldsymbol{\alpha} \in \mathbb{F}_2^\ell$)
- Cheating now requires to modify d coordinates (d min. distance of code)
- MAC fooled w.p. $1/2^d$.

MiniMAC (Damgård/Zakarias, 2013)

MiniMAC computes then on data-batches of k bits, i.e. can be seen as:

- Computing arithmetic circuits over the **ring** $\mathbb{F}_2^k$ with componentwise addition and multiplication, or
- Computing k evaluations of a circuit over $\mathbb{F}_2$ simultaneously.

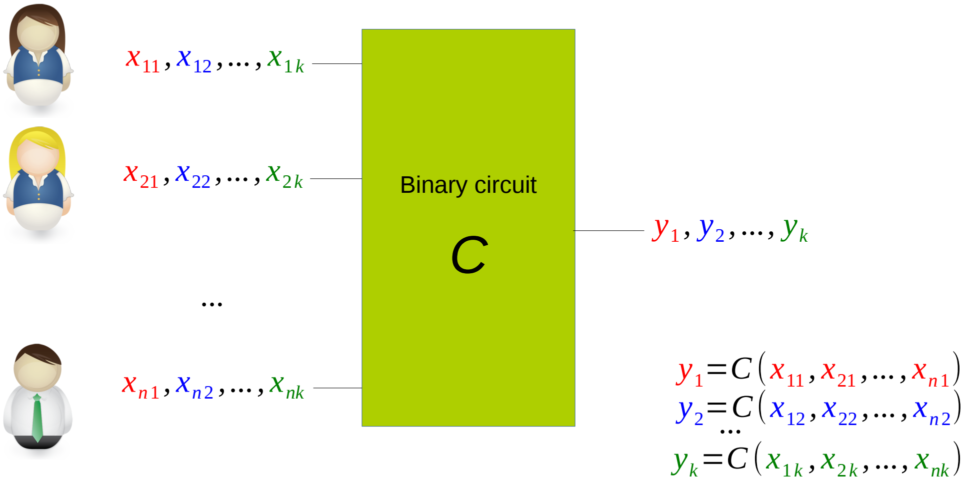
MiniMAC (Damgård/Zakarias, 2013)

MiniMAC computes then on data-batches of k bits, i.e. can be seen as:

- Computing arithmetic circuits over the **ring** $\mathbb{F}_2^k$ with componentwise addition and multiplication, or
- Computing k evaluations of a circuit over $\mathbb{F}_2$ simultaneously.

Note: This can be adapted for **single** evaluations of a well-formed boolean circuit (more on that later).

MiniMAC (Damgård/Zakarias, 2013)



MiniMAC (Damgård/Zakarias, 2013)

- Multiplication is done through Beaver's technique and involves the so-called Schur-square of the code C^{*2}
- Requirement: $d_{min}(C^{*2}) \geq \lambda$ (sec. parameter)
- Overhead depends on ℓ/k (ℓ length of the code, k dimension).
- For binary codes and $\lambda \sim 128$, best constructions (Cascudo, 2019) give $\ell/k \sim 10$
- Alternative (Damgård/Lauritsen/Toft, 2014): use Reed-Solomon over constant extension of $\mathbb{F}_2$ (requires much more preprocessing).

This paper: Alternative to MiniMAC with "better packing"

We present an alternative approach to compute simultaneously k instances of a boolean circuit,

- We use the notion of **Reverse Multiplication Friendly Embeddings (RMFE)**
- Previously used (Cascudo/Cramer/Xing/Yuan, 2018) in the case of **information-theoretically perfectly secure** MPC.
- More precisely: adapted Beerliova-Trubini/Hirt info th. secure protocol (see TCC Test of Time Award) for small fields, obtaining the same amortized communication.

Embedding via Reverse Multiplication Friendly Embeddings

Reverse Multiplication Friendly Embeddings: allows to embed the **ring** $\mathbb{F}_2^k$ into a **field** $\mathbb{F}_{2^m}$ while "reconciling" enough of their algebraic structures.

Embedding via Reverse Multiplication Friendly Embeddings

Reverse Multiplication Friendly Embeddings: allows to embed the **ring** $\mathbb{F}_2^k$ into a **field** $\mathbb{F}_{2^m}$ while “reconciling” enough of their algebraic structures.

A $(k, m)_2$-RMFE is a pair (ϕ,ψ) where

- $\phi : \mathbb{F}_2^k \to \mathbb{F}_{2^m}$ is $\mathbb{F}_2$-linear.
- $\psi : \mathbb{F}_{2^m} \to \mathbb{F}_2^k$ is $\mathbb{F}_2$-linear.
- For all $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^k$,

$$\mathbf{x} * \mathbf{y} = \psi(\phi(\mathbf{x}) \cdot \phi(\mathbf{y}))$$

(here $*$ denotes coordinate-wise product in $\mathbb{F}_2^k$, $\cdot$ field product in $\mathbb{F}_{2^m}$)

The point: m can be made to be “not much larger” than k

Constructions of RMFE

[Remember a $(k, m)_2$-RMFE embeds $\mathbb{F}_2^k$ into $\mathbb{F}_{2^m}$]

- **Asymptotical** (algebraic geometric constructions): There exist families of $(k, O(k))_2$-RMFE.

Constructions of RMFE

[Remember a $(k, m)_2$-RMFE embeds $\mathbb{F}_2^k$ into $\mathbb{F}_{2^m}$]

- **Asymptotical** (algebraic geometric constructions): There exist families of $(k, O(k))_2$-RMFE.
- **Non-asymptotical** (polynomial interpolation-based constructions): For all

  $r \leq 33$, there exists a $(3r, 10r - 5)_2$-RMFE. ($m \sim 3.3k$)
  E.g. we can embed $\mathbb{F}_2^{42}$ into $\mathbb{F}_{2^{135}}$.

  For all $r \leq 16$, there exists a $(2r, 8r)_2$-RMFE. ($m = 4k$, but "nicer" ext. fields)
  E.g. we can embed $\mathbb{F}_2^{32}$ into $\mathbb{F}_{2^{128}}$.

Our protocol (online phase)

Let (ϕ, ψ) be a $(k, m)_2$-RMFE.
Our online phase:

- Global additively-shared key $\alpha \in \mathbb{F}_{2^m}$
- Authenticated sharings of $\mathbf{x} \in \mathbb{F}_2^k$ are $\langle \mathbf{x} \rangle = ((\mathbf{x}_1, ..., \mathbf{x}_n), (m_1, ..., m_n))$ where:

$$\sum \mathbf{x}_i = \mathbf{x} \quad (\text{in } \mathbb{F}_2^k)$$

$$\sum m_i = \alpha \cdot \phi(\mathbf{x}) \quad (\text{in } \mathbb{F}_{2^m})$$

- Sums: $\langle \mathbf{x} + \mathbf{y} \rangle$ can be computed locally from $\langle \mathbf{x} \rangle$, $\langle \mathbf{y} \rangle$ (uses that ϕ is $\mathbb{F}_2$-linear).
- Products: $\langle \mathbf{x} * \mathbf{y} \rangle$?

Computing products

We need the following from preprocessing:

- A triple $(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle \mathbf{c} \rangle)$, for random $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^k$, where $\mathbf{c} = \mathbf{a} * \mathbf{b}$
- A reencoding pair $(\langle \psi(r) \rangle, [r])$ for random $r \in \mathbb{F}_{2^m}$, where $[\cdot]$ is the SPDZ authenticated sharing (same α as for $\langle \cdot \rangle$).

Multiplication of $\langle \mathbf{x} \rangle, \langle \mathbf{y} \rangle$:

- Open* $\langle \boldsymbol{\epsilon} \rangle = \langle \mathbf{x} \rangle - \langle \mathbf{a} \rangle, \langle \boldsymbol{\delta} \rangle = \langle \mathbf{y} \rangle - \langle \mathbf{b} \rangle$
- Compute $[\sigma] = \boldsymbol{\epsilon} * \langle \mathbf{y} \rangle + \boldsymbol{\delta} * \langle \mathbf{x} \rangle - \phi(\boldsymbol{\epsilon}) \cdot \phi(\boldsymbol{\delta}) - [r]$
- Open* σ
- Compute and output $\psi(\sigma) + \langle \mathbf{c} \rangle + \langle \psi(r) \rangle = \langle \mathbf{x} * \mathbf{y} \rangle$

*Partial open: only data shares are revealed, not MAC shares.

Comparison

- We compare to MiniMAC and Committed MPC (Frederiksen/Pinkas/Yanai, 2018).
- Committed MPC: uses UC homomorphic commitments, implemented from linear codes.
- The comparison essentially boils down to the "encoding expansion factor" (in our case m/k).

Sec. par.	Phase	MiniMAC	Committed MPC	Our protocol
$\lambda = 64$	Multiply	$20.14 \cdot (n-1)$	$29.89 \cdot (n-1)$	$10.2 \cdot (n-1)$
	Output	$19.5 \cdot n + 2(n-1)$	$19.5 \cdot (n-1)n$	$6.2 \cdot n + 2(n-1)$
$\lambda = 128$	Multiply	$23.48 \cdot (n-1)$	$35.58 \cdot (n-1)$	$10.42 \cdot (n-1)$
	Output	$24.22 \cdot n + 2(n-1)$	$24.22 \cdot (n-1)n$	$6.42 \cdot n + 2(n-1)$

Table: Total number of bits sent per instance at multiplication and output gates

The version of MiniMAC in Damgård/Lauritsen/Toft, 2014 needs only to communicate $8(n-1)$ per multiplication gate – But much more preprocessing.

Preprocessing

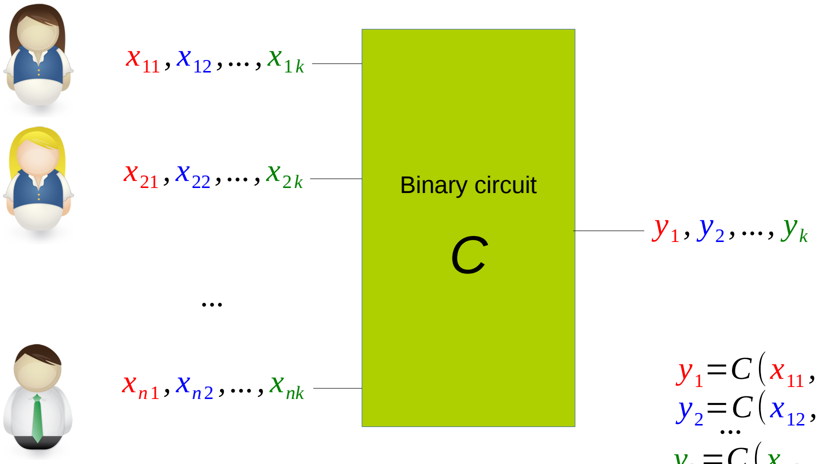
In the preprocessing we need to produce the following:

- Input pairs $(\mathbf{r}, \langle \mathbf{r} \rangle)$, where $\mathbf{r} \in \mathbb{F}_2^k$ is random and known by a single party.
- Multiplication triples $(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle \mathbf{a} * \mathbf{b} \rangle)$, where $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^k$ are random.
- Reencoding pairs $(\langle \psi(r) \rangle, [r])$, where $r \in \mathbb{F}_{2^m}$ is random.

We use techniques from MASCOT (Keller/Orsini/Scholl, 2016).
MASCOT is based on bit-OT's, and this fits well with the $\mathbb{F}_2$-linearity of ϕ and ψ.

From multiple instance to single instance evaluation

So far: We showed how to simultaneously compute k instances of a circuit over $\mathbb{F}_2$.



$$y_1 = C(x_{11}, x_{21}, \dots, x_{n1})$$
$$y_2 = C(x_{12}, x_{22}, \dots, x_{n2})$$
$$\dots$$
$$y_k = C(x_{1k}, x_{2k}, \dots, x_{nk})$$

From multiple instance to single instance evaluation

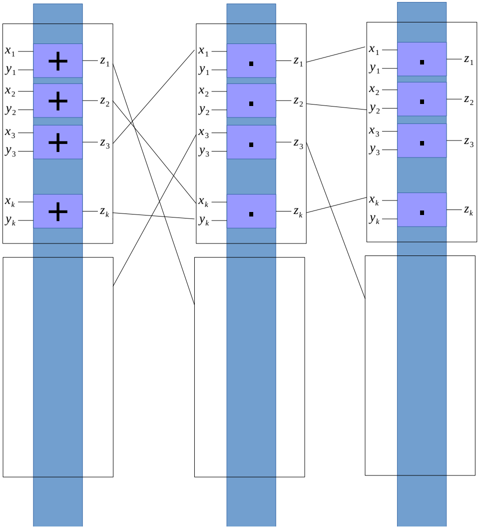
Damgård/Zakarias, 2013: Showed how to adapt MiniMAC to efficiently computing a **single** instance of a "**well-formed**" circuit:

- Layers of gates of the same type.
- Number of gates in most layers large (or multiple of k)
- Number of "direct wires" from layer i to j is large, or 0.

We can then:

- Group gates in a layer in batches of k, adding a small number of overhead dummy gates.
- Construct maps that reorganize all outputs of one layer into blocks of inputs of next layers (again without much overhead from additional dummy gates).

From multiple instance to single instance evaluation



From multiple instance to single instance evaluation

We simply follow approach from Damgård-Zakarias.

We compute block of gates in one layer with our protocol. Then we reorganize outputs to fit next layers:

- Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l)$ be all the output blocks from one layer.
- Reorganizing by $F_i(\mathbf{X}) = \mathbf{x}'_i \in \mathbb{F}_2^k$ s.t. $\mathbf{x}'_i$ are input blocks to a subsequent layer.
- Assume we have $\langle \mathbf{R} \rangle = (\langle \mathbf{r}_1 \rangle, \ldots, \langle \mathbf{r}_l \rangle)$, $\langle F_i(\mathbf{R}) \rangle$ from the preprocessing. Opening $\mathbf{X} - \mathbf{R}$ and computing $F_i(\mathbf{X} - \mathbf{R}) + \langle F_i(\mathbf{R}) \rangle$ yields $\langle F_i(\mathbf{X}) \rangle = \mathbf{x}'_i$.
- Preprocessing again easy ($\mathbb{F}_2$-linearity)

Conclusion

We presented a secret-sharing-based MPC protocol for computation of boolean circuits.

- Our approach applies the RMFE strategy from Cascudo et al., 2018 to the dishonest majority setting: RMFE+SPDZ
- Structurally similar to MiniMAC (Damgård/Zakarias, 2013)...
- ...but MACs over extension field allow for shorter encoding.
- In the paper we also present how to produce preprocessed data needed for the online phase.