Multi-Party Computation in the Head: Techniques and Applications

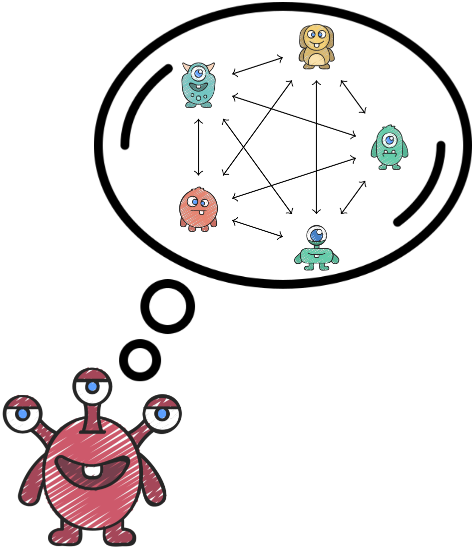
Damien Vergnaud
Sorbonne Université – LIP6

(with special thanks to Charles Bouillaguet, Thibauld Feneuil, Jules Maire, Matthieu Rivain and the CCA master students)

SORBONNE UNIVERSITÉ

Images by `juicy_fish` from `flaticon`

Outline



MPC-in-the-Head

MPC protocol

↓

ZK proof

Generic technique

Optimizations

Applications

Main Application: Digital Signatures

- consider some one-way function F
- picks uniformly at random sk in F's domain
- sets and publishes $\mathrm{pk} = F(\mathrm{sk})$

- to sign m, Alice proves
  - in zero-knowledge
  - **non-interactively** (using Fiat-Shamir heuristic using m)

  that she knows sk such that $\mathrm{pk} = F(\mathrm{sk})$

- F one-way against quantum computers
  $\rightsquigarrow$ **"post-quantum"** signatures
  (7-9 submission to the recent NIST call for signatures)

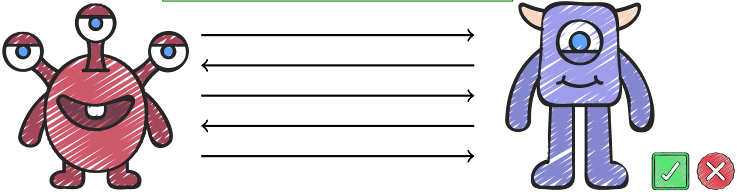
Zero-knowledge interactive proof

Goldwasser, Micali, Rackoff – STOC 1985 **(1993 Gödel Prize)**
Goldreich, Micali, Wigderson – FOCS 1986

x

$$\boxed{\mathcal{L} = \{y \mid \exists x, \mathcal{R}(y, x) = 1\}}$$

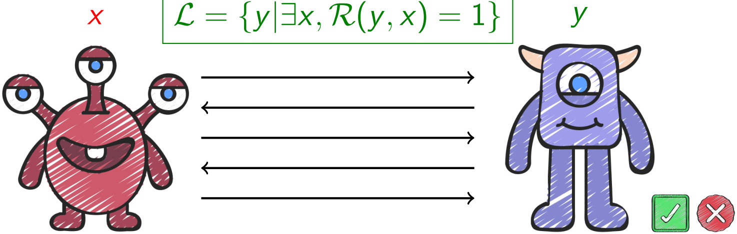
y



Zero-knowledge interactive proof

Goldwasser, Micali, Rackoff – STOC 1985 **(1993 Gödel Prize)**
Goldreich, Micali, Wigderson – FOCS 1986

$$\boxed{\mathcal{L} = \{y | \exists x, \mathcal{R}(y, x) = 1\}}$$

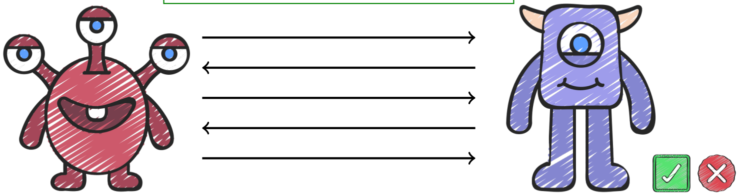


Completeness

Zero-knowledge interactive proof

Goldwasser, Micali, Rackoff – STOC 1985 **(1993 Gödel Prize)**
Goldreich, Micali, Wigderson – FOCS 1986

x $\boxed{\mathcal{L} = \{y | \exists x, \mathcal{R}(y, x) = 1\}}$ y



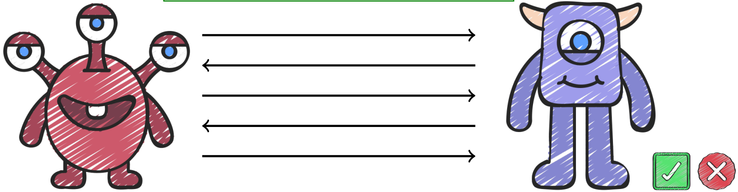
Completeness

(Knowledge) Soundness

Zero-knowledge interactive proof

Goldwasser, Micali, Rackoff – STOC 1985 **(1993 Gödel Prize)**
Goldreich, Micali, Wigderson – FOCS 1986

x $\boxed{\mathcal{L} = \{y \mid \exists x, \mathcal{R}(y, x) = 1\}}$ y



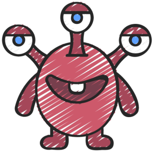
Completeness

(Knowledge) Soundness

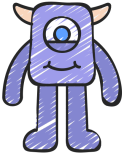
(Honest-Verifier) Zero-knowledge

Guillou-Quisquater Protocol (ZK for RSA – 1988)

$\boxed{(N, e) \quad y = x^e \bmod N}$

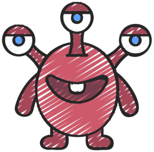


x y

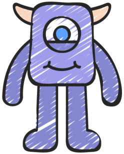


Guillou-Quisquater Protocol (ZK for RSA – 1988)

$$\boxed{(N, e) \quad y = x^e \bmod N}$$



x y

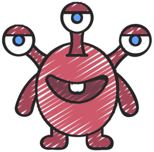


$r \xleftarrow{} (\mathbb{Z}/N\mathbb{Z})^*$

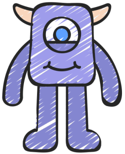
$k \leftarrow r^e \bmod N$

Guillou-Quisquater Protocol (ZK for RSA – 1988)

$\boxed{(N, e) \quad y = x^e \bmod N}$



x y



$r \xleftarrow{} (\mathbb{Z}/N\mathbb{Z})^*$

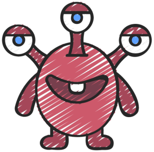
$k \leftarrow r^e \bmod N$

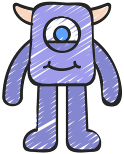
$\xrightarrow{k}$

$c \xleftarrow{} \{0, \ldots, e-1\}$

Guillou-Quisquater Protocol (ZK for RSA – 1988)

$\boxed{(N, e) \quad y = x^e \bmod N}$





x ——————————————— y

$r \leftarrow (\mathbb{Z}/N\mathbb{Z})^*$

$k \leftarrow r^e \bmod N$

$\xrightarrow{k}$

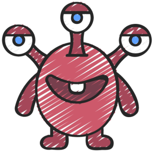
$c \leftarrow \{0, \ldots, e-1\}$

$\xleftarrow{c}$

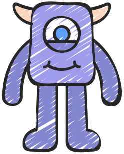
$s \leftarrow rx^c \bmod N$

Guillou-Quisquater Protocol (ZK for RSA – 1988)

$\boxed{(N, e) \quad y = x^e \bmod N}$



x ——————————————————— y



$r \longleftarrow (\mathbb{Z}/N\mathbb{Z})^*$

$k \leftarrow r^e \bmod N$

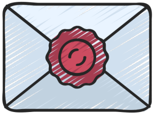
$\xrightarrow{\quad k \quad}$

$c \longleftarrow \{0, \ldots, e-1\}$

$\xleftarrow{\quad c \quad}$

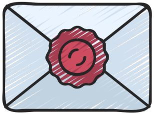
$s \leftarrow r x^c \bmod N \quad \xrightarrow{\quad s \quad} \quad s^e \stackrel{?}{\equiv} k y^c \bmod N$

Commitments



- digital analogue of a sealed envelope
  $\rightsquigarrow$ hide a value that cannot be changed
- $(\textsc{Commit}, \textsc{Open})$
  - $\textsc{Commit}(m; r) \rightsquigarrow (c, s)$
  - $\textsc{Open}(c, s) \rightsquigarrow m$ or $\perp$

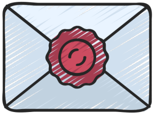
Commitments



- digital analogue of a sealed envelope
  $\rightsquigarrow$ hide a value that cannot be changed
- $(\textsc{Commit}, \textsc{Open})$
  - $\textsc{Commit}(m; r) \rightsquigarrow (c, s)$
  - $\textsc{Open}(c, s) \rightsquigarrow m$ or $\perp$

Hiding

Commitments

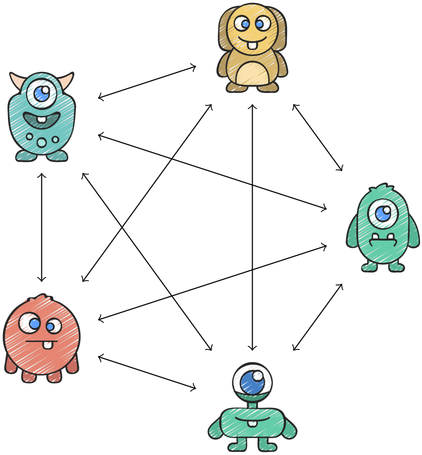


- digital analogue of a sealed envelope
  $\rightsquigarrow$ hide a value that cannot be changed
- $(\textsc{Commit}, \textsc{Open})$
  - $\textsc{Commit}(m; r) \rightsquigarrow (c, s)$
  - $\textsc{Open}(c, s) \rightsquigarrow m$ or $\perp$

Hiding

Binding

Multi-Party Computation



- computation between parties who do not trust each other
- preserve the privacy of each player's inputs
- guarantee the correctness of the computation

Multi-Party Computation

- Parties $P_1, \ldots, P_n$ with private input $x_1, \ldots, x_n$
  $\rightsquigarrow$ wish to compute a joint function $f(x_1, \ldots, x_n)$
- Some parties might be corrupted:
  - Semi-honest: follow the protocol specifications
  - Malicious: might act arbitrarily

Multi-Party Computation

- Parties $P_1, \ldots, P_n$ with private input $x_1, \ldots, x_n$
  $\rightsquigarrow$ wish to compute a joint function $f(x_1, \ldots, x_n)$
- Some parties might be corrupted:
  - Semi-honest: follow the protocol specifications
  - Malicious: might act arbitrarily

Multi-Party Computation

- Parties $P_1, \dots, P_n$ with private input $x_1, \dots, x_n$
  $\rightsquigarrow$ wish to compute a joint function $f(x_1, \dots, x_n)$
- Some parties might be corrupted:
  - Semi-honest: follow the protocol specifications
  - Malicious: might act arbitrarily

Perfect
Correctness

Multi-Party Computation

- Parties $P_1, \ldots, P_n$ with private input $x_1, \ldots, x_n$
  $\rightsquigarrow$ wish to compute a joint function $f(x_1, \ldots, x_n)$
- Some parties might be corrupted:
  - Semi-honest: follow the protocol specifications
  - Malicious: might act arbitrarily

Perfect Correctness

t-Privacy

Multi-Party Computation

- Parties $P_1, \ldots, P_n$ with private input $x_1, \ldots, x_n$
  $\rightsquigarrow$ wish to compute a joint function $f(x_1, \ldots, x_n)$
- Some parties might be corrupted:
  - Semi-honest: follow the protocol specifications
  - Malicious: might act arbitrarily

Perfect Correctness

t-Privacy

- For any f, there exist a t-private protocol (for $t < n/2$) with **unconditional** semi-honest security

Ben-Or, Goldwasser, Wigderson – STOC 1988

n-out-of-n Secret Sharing

- Let x be a secret from a group $(\mathbb{G}, \boxplus)$.
- Dealer chooses random $x_1, \ldots, x_{n-1}$ in $\mathbb{G}$ and computes

$$x_n = x \boxminus (x_1 \boxplus \cdots \boxplus x_{n-1})$$

  The **shares** are $(x_1, \ldots, x_n) \longleftarrow \textsc{Share}(x)$

- Given $(x_1, \ldots, x_n)$, one can successfully recover

$$x = x_1 \boxplus \cdots \boxplus x_n = \textsc{Reconstruct}(x_1, \ldots, x_n)$$

- Given all but one x_i's $\rightsquigarrow$ no information about x

MPC-in-the-Head

Ishai, Kushilevitz, Ostrovsky, Sahai – STOC 2007

- Given a public y, Alice wants to prove that she knows x s.t.

$$F(x) = y$$

- Alice uses a n-party secret-sharing $(\textsc{Share}, \textsc{Reconstruct})$:

$$(x_1, \ldots, x_n) \xleftarrow{\text{⚂⚃}} \textsc{Share}(x)$$

- Consider an n-party computation:

$$f(x_1, \ldots, x_n) := F(\textsc{Reconstruct}(x_1, \ldots, x_n))$$

- Alice simulates (in her head) a secure MPC protocol for f with
  - 2-privacy in the semi-honest model
  - perfect correctness

Views of Parties in MPC

- view of P_i denoted V_i is
  - its input w_i
  - its random coins r_i
  - all the messages received by P_i (in particular, $f(x_1, \ldots, x_n)$)
- Given V_i one can perform the same computation as P_i (using the description of the MPC protocol)
- V_i and V_j are consistent if the outgoing messages $P_i \rightarrow P_j$ are identical to the incoming messages $P_j \leftarrow P_i$ (and *vice versa*)
- Proposition 1: All pairs of views (V_i, V_j) are consistent iff there exists an execution of the protocol in which the view of P_i is V_i.

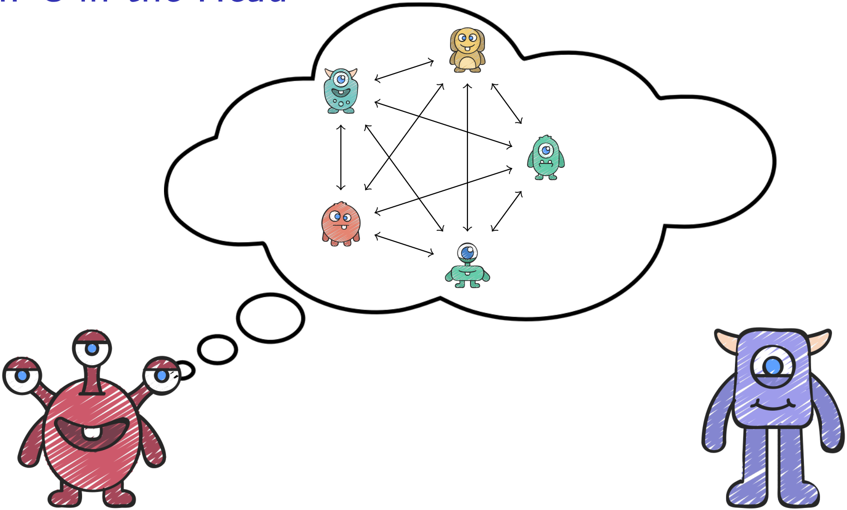
Views of Parties in MPC

- view of P_i denoted V_i is
  - its input w_i
  - its random coins r_i
  - all the messages received by P_i (in particular, $f(x_1, \ldots, x_n)$)
- Given V_i one can perform the same computation as P_i (using the description of the MPC protocol)
- V_i and V_j are consistent if the outgoing messages $P_i \rightarrow P_j$ are identical to the incoming messages $P_j \leftarrow P_i$ (and *vice versa*)
- Proposition 1: All pairs of views (V_i, V_j) are consistent iff there exists an execution of the protocol in which the view of P_i is V_i.

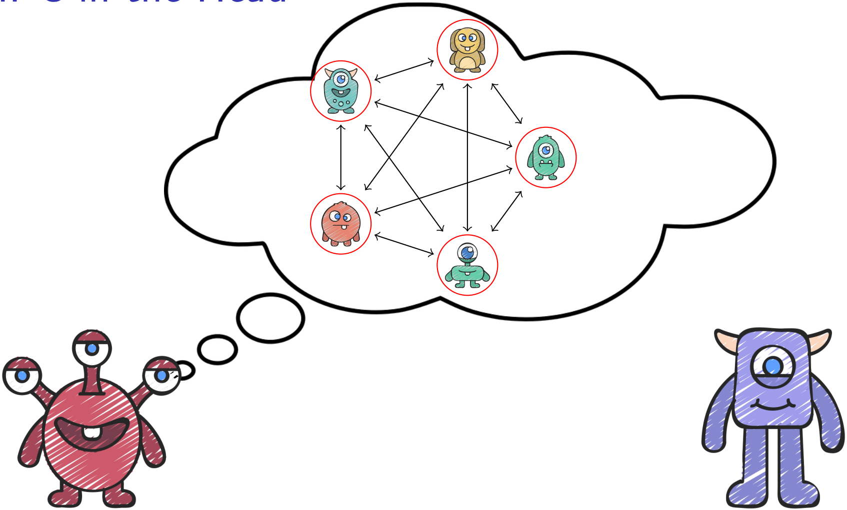
Views of Parties in MPC

- view of P_i denoted V_i is
  - its input w_i
  - its random coins r_i
  - all the messages received by P_i (in particular, $f(x_1, \ldots, x_n)$)
- Given V_i one can perform the same computation as P_i (using the description of the MPC protocol)
- V_i and V_j are consistent if the outgoing messages $P_i \rightarrow P_j$ are identical to the incoming messages $P_j \leftarrow P_i$ (and *vice versa*)
- Proposition 1: All pairs of views (V_i, V_j) are consistent iff there exists an execution of the protocol in which the view of P_i is V_i.

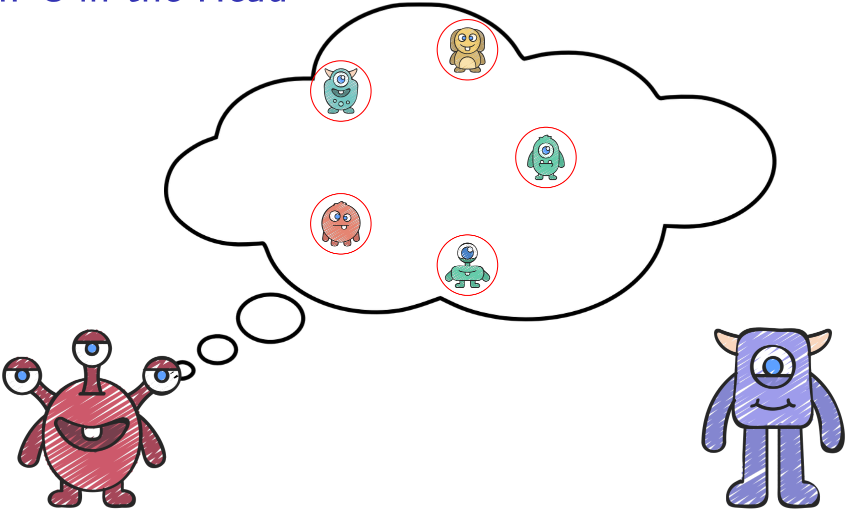
MPC-in-the-Head



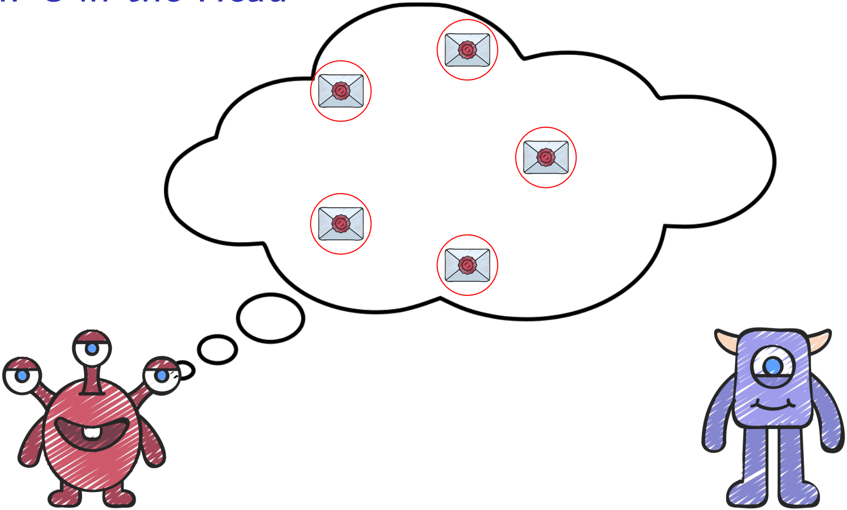
MPC-in-the-Head



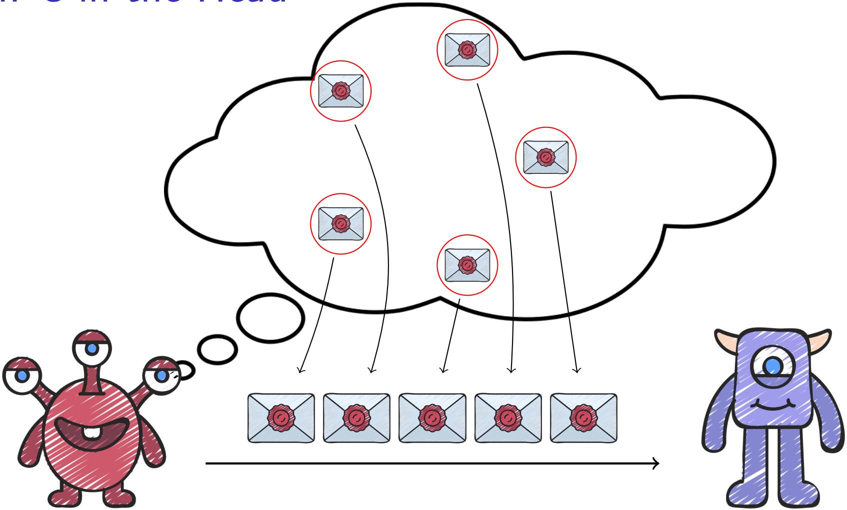
MPC-in-the-Head



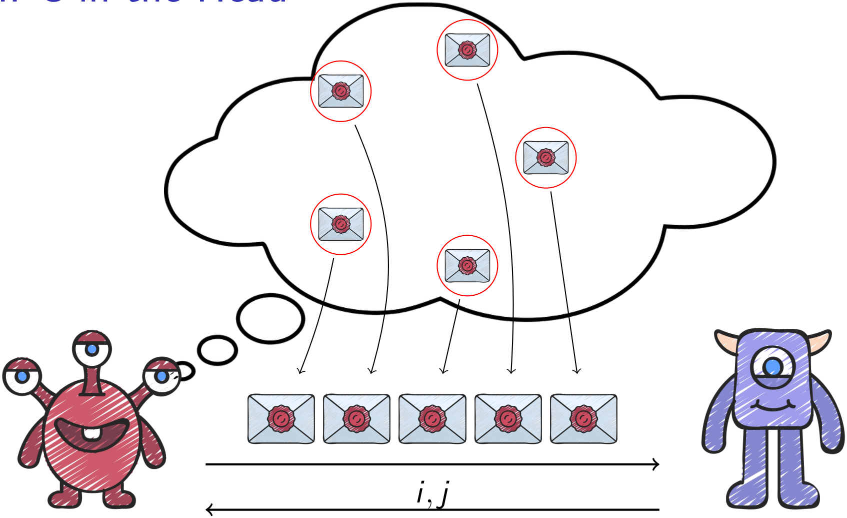
MPC-in-the-Head



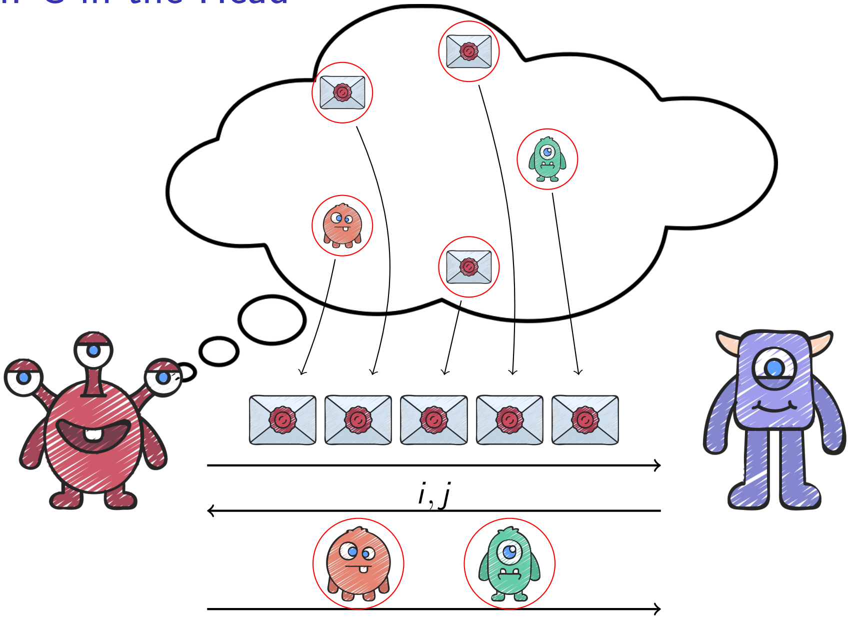
MPC-in-the-Head



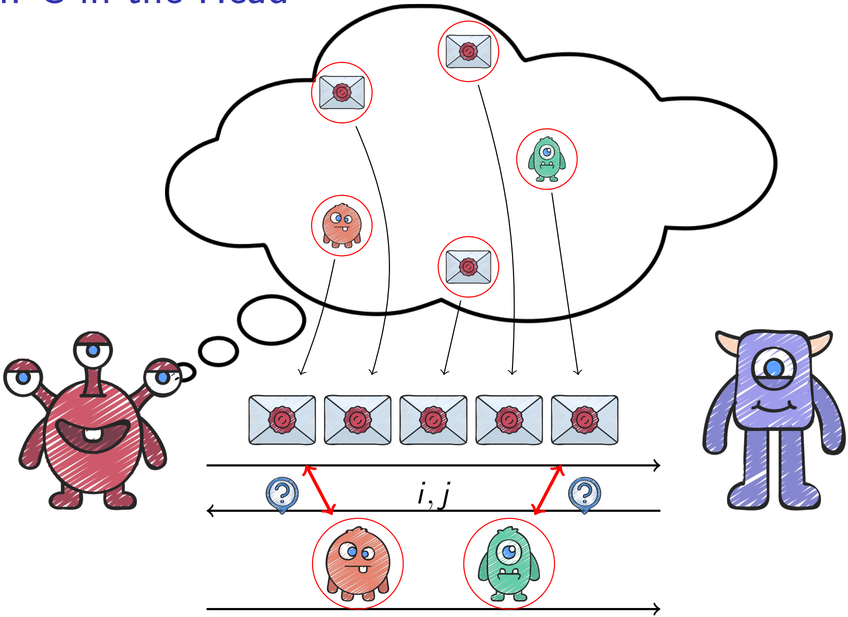
MPC-in-the-Head



MPC-in-the-Head



MPC-in-the-Head



MPC-in-the-Head – Security

1. **Completeness:** by inspection

2. **Soundness:** by Proposition 1, if all pairs of views are consistent and Π outputs 1 then

$$F(\textsc{Reconstruct}(x_1, \ldots, x_n)) = y$$

If ($F(x) \neq y$ or (at least) one pair of views is inconsistent), Bob detects it with probability

$$\geq \binom{n}{2}^{-1} = \frac{2}{n(n-1)}$$

3. **Zero-knowledge:** by the hiding property of the commitment scheme and the 2-privacy of Π

MPC-in-the-Head – Security

1. **Completeness:** by inspection

2. **Soundness:** by Proposition 1, if all pairs of views are consistent and Π outputs 1 then

$$F(\textsc{Reconstruct}(x_1, \ldots, x_n)) = y$$

   If ($F(x) \neq y$ or (at least) one pair of views is inconsistent), Bob detects it with probability

$$\geq \binom{n}{2}^{-1} = \frac{2}{n(n-1)}$$

3. **Zero-knowledge:** by the hiding property of the commitment scheme and the 2-privacy of Π

MPC-in-the-Head – Security

1. **Completeness:** by inspection

2. **Soundness:** by Proposition 1, if all pairs of views are consistent and Π outputs 1 then

$$F(\textsc{Reconstruct}(x_1, \ldots, x_n)) = y$$

If ($F(x) \neq y$ or (at least) one pair of views is inconsistent), Bob detects it with probability

$$\geq \binom{n}{2}^{-1} = \frac{2}{n(n-1)}$$

3. **Zero-knowledge:** by the hiding property of the commitment scheme and the 2-privacy of Π

MPC-in-the-Head using BGW

- the complexity **increases** with n
  the cheating probability **increases** with n
  $\rightsquigarrow$ pick the minimal n!
- for 2-privacy with BGW, we need at least $n \geq 5$ players
- with $n = 5$,
  - Alice has to commit 5 views of the protocol (and reveals 2)
  - If she cheats, Bob detects it with probability $\geq 1/10$
- with $n = 5$, a cheating Alice is not detected
  - in one run with probability $\leq 9/10$
  - in k independent runs with probability $\leq (9/10)^k$
    $\rightsquigarrow$ with $k \geq 842$, Alice is not detected with probability $\leq 2^{-128}$

MPC-in-the-Head using BGW

- the complexity **increases** with n
  the cheating probability **increases** with n
  $\rightsquigarrow$ pick the minimal n!
- for 2-privacy with BGW, we need at least $n \geq 5$ players
- with $n = 5$,
  - Alice has to commit 5 views of the protocol (and reveals 2)
  - If she cheats, Bob detects it with probability $\geq 1/10$
- with $n = 5$, a cheating Alice is not detected
  - in one run with probability $\leq 9/10$
  - in k independent runs with probability $\leq (9/10)^k$
    $\rightsquigarrow$ with $k \geq 842$, Alice is not detected with probability $\leq 2^{-128}$

MPC-in-the-Head using BGW

- the complexity **increases** with n
  the cheating probability **increases** with n
  $\rightsquigarrow$ pick the minimal n!
- for 2-privacy with BGW, we need at least $n \geq 5$ players
- with $n = 5$,
  - Alice has to commit 5 views of the protocol (and reveals 2)
  - If she cheats, Bob detects it with probability $\geq 1/10$
- with $n = 5$, a cheating Alice is not detected
  - in one run with probability $\leq 9/10$
  - in k independent runs with probability $\leq (9/10)^k$
    $\rightsquigarrow$ with $k \geq 842$, Alice is not detected with probability $\leq 2^{-128}$

MPC-in-the-Head using BGW

- the complexity **increases** with n
  the cheating probability **increases** with n
  $\rightsquigarrow$ pick the minimal n!
- for 2-privacy with BGW, we need at least $n \geq 5$ players
- with $n = 5$,
  - Alice has to commit 5 views of the protocol (and reveals 2)
  - If she cheats, Bob detects it with probability $\geq 1/10$
- with $n = 5$, a cheating Alice is not detected
  - in one run with probability $\leq 9/10$
  - in k independent runs with probability $\leq (9/10)^k$
    $\rightsquigarrow$ with $k \geq 842$, Alice is not detected with probability $\leq 2^{-128}$

n-out-of-n Secret Sharing in MPC

- Is it possible to reveal $n-1$ shares in the MPC and remain secure?
  $\rightsquigarrow$ would lead to better soundness!
- impossible classically for general functions (for IT security)
- **but**, possible for "linear" functions, e.g. for $a, b \in \mathbb{Z}$:

$$\begin{aligned} a \cdot x \boxplus b \cdot y &= a \cdot (x_1 \boxplus \cdots \boxplus x_n) \boxplus b \cdot (y_1 \boxplus \cdots \boxplus y_n) \\ &= (a \cdot x_1 \boxplus b \cdot y_1) \boxplus \cdots \boxplus (a \cdot x_n \boxplus b \cdot y_n) \end{aligned}$$

where $a \cdot x = \underbrace{x \boxplus \cdots \boxplus x}_{a \text{ times}}$ (for $a \geq 0$)

n-out-of-n Secret Sharing in MPC

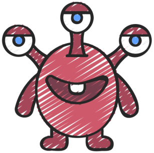
- Is it possible to reveal $n-1$ shares in the MPC and remain secure?
  $\rightsquigarrow$ would lead to better soundness!
- impossible classically for general functions (for IT security)
- **but**, possible for "linear" functions, e.g. for $a, b \in \mathbb{Z}$:

$$\begin{aligned} a \cdot x \boxplus b \cdot y &= a \cdot (x_1 \boxplus \cdots \boxplus x_n) \boxplus b \cdot (y_1 \boxplus \cdots \boxplus y_n) \\ &= (a \cdot x_1 \boxplus b \cdot y_1) \boxplus \cdots \boxplus (a \cdot x_n \boxplus b \cdot y_n) \end{aligned}$$

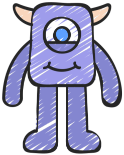
where $a \cdot x = \underbrace{x \boxplus \cdots \boxplus x}_{a \text{ times}}$ (for $a \geq 0$)

RSA-in-the-head (Maire-V. 2023)

$\boxed{(N, e) \quad y = x^e \bmod N}$

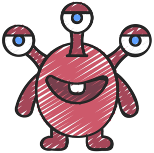


x y

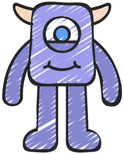


RSA-in-the-head (Maire-V. 2023)

$(N, e) \quad y = x^e \bmod N$



x y

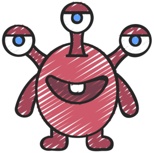


$[\![x]\!] \xleftarrow{\$} [(\mathbb{Z}/N\mathbb{Z})^*]^n$

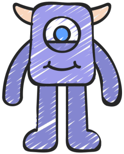
$(y_i \leftarrow x_i^e \bmod N)_i$

RSA-in-the-head (Maire-V. 2023)

$(N, e) \quad y = x^e \bmod N$



x y



$[\![x]\!] \xleftarrow{\text{⚂⚃}} [(\mathbb{Z}/N\mathbb{Z})^*]^n$

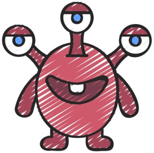
$(y_i \leftarrow x_i^e \bmod N)_i$

$\xrightarrow{\vec{y}}$

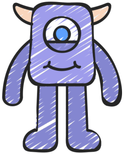
$i^\star \xleftarrow{\text{⚂⚃}} \{1, \ldots, n\}$

RSA-in-the-head (Maire-V. 2023)

$(N, e) \quad y = x^e \bmod N$



x ——— y



$[\![x]\!] \longleftarrow [(\mathbb{Z}/N\mathbb{Z})^*]^n$

$(y_i \leftarrow x_i^e \bmod N)_i$

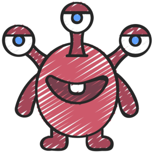
$\xrightarrow{\vec{y}}$

$i^\star \longleftarrow \{1, \ldots, n\}$

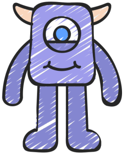
$\xleftarrow{i^\star}$

RSA-in-the-head (Maire-V. 2023)

$$\boxed{(N, e) \quad y = x^e \bmod N}$$



x ——— y



$$[\![x]\!] \longleftarrow [(\mathbb{Z}/N\mathbb{Z})^*]^n$$

$$(y_i \leftarrow x_i^e \bmod N)_i$$

$$\xrightarrow{\vec{y}}$$

$$i^\star \longleftarrow \{1, \ldots, n\}$$

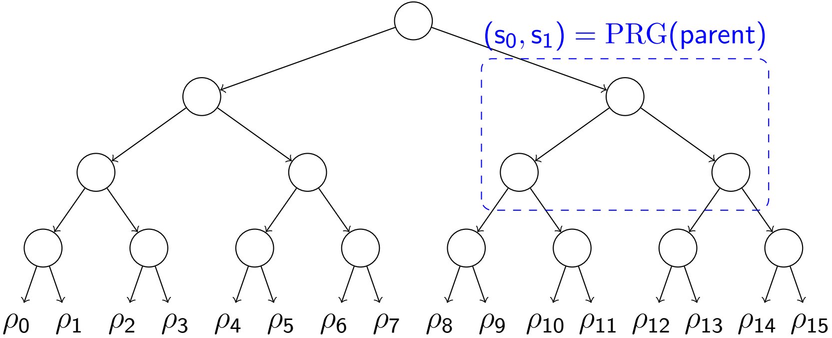
$$\xleftarrow{i^\star}$$

$$- \qquad \xrightarrow{(x_i)_{i \neq i^\star}}$$

$$x_i^e \stackrel{?}{\equiv} y_i \bmod N$$

$$y_1 \cdots y_n \stackrel{?}{\equiv} y \bmod N$$

Compressing the Proof - Tree PRG



Commitment

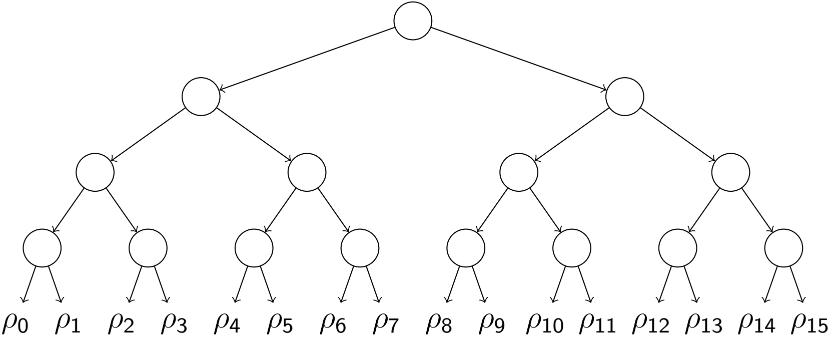
1. $(x_i, r_i) = \mathrm{PRG}(\rho_i)$
2. r_n, r picked at random
3. $c_i = H(y_i, r_i)$
4. $c = H(c_0, \ldots, c_n, r)$ and Δ_x

Response

Alice reveals r and

1. $\log_2(n)$ values in the tree (in blue)
2. c_{i^*}

Compressing the Proof - Tree PRG



Commitment

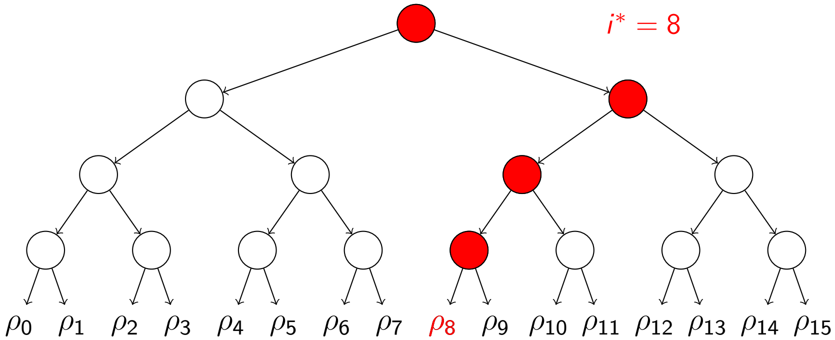
1. $(x_i, r_i) = \mathrm{PRG}(\rho_i)$
2. r_n, r picked at random
3. $c_i = H(y_i, r_i)$
4. $c = H(c_0, \ldots, c_n, r)$ and Δ_x

Response

Alice reveals r and

1. $\log_2(n)$ values in the tree (in blue)
2. c_{i^*}

Compressing the Proof - Tree PRG



Commitment

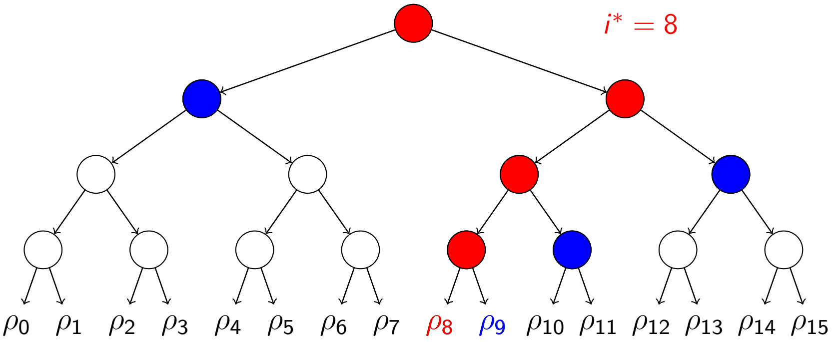
1. $(x_i, r_i) = \mathrm{PRG}(\rho_i)$
2. r_n, r picked at random
3. $c_i = H(y_i, r_i)$
4. $c = H(c_0, \ldots, c_n, r)$ and Δ_x

Response

Alice reveals r and

1. $\log_2(n)$ values in the tree (in blue)
2. c_{i^*}

Compressing the Proof - Tree PRG



Commitment

1. $(x_i, r_i) = \mathrm{PRG}(\rho_i)$
2. r_n, r picked at random
3. $c_i = H(y_i, r_i)$
4. $c = H(c_0, \ldots, c_n, r)$ and Δ_x

Response

Alice reveals r and

1. $\log_2(n)$ values in the tree (in blue)
2. c_{i^*}

RSA-in-the-head

$$\boxed{e = 3,\ N \simeq 2^{2048},\ \lambda = 128}$$

1. Guillou-Quisquater
   - Soundness error: $1/e = 1/3$
   - Iterations: 80
   - Proof size: $80 \times 2048 + 256 = 20.5$ KBytes

2. RSA-in-the-head
   - Soundness error: $1/n = 1/256$
   - Iterations: 16
   - Proof size: $16 \times (8 \times 128 + 2048) + 256 + 128 = 6.5$ KBytes

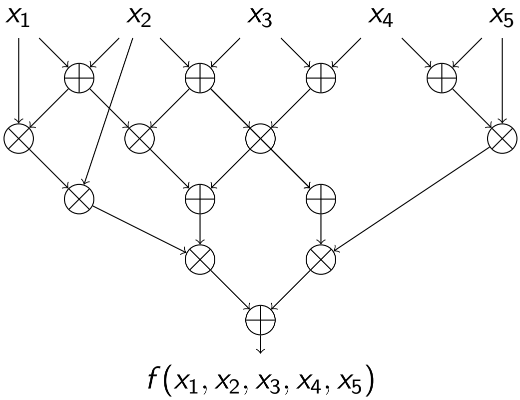
RSA-in-the-head

$\boxed{e = 17,\ N \simeq 2^{2048},\ \lambda = 128}$

1. Guillou-Quisquater
   - Soundness error: $1/e = 1/17$
   - Iterations: 32
   - Proof size:
     $32 \times 2048 + 256 = 8.2$ KBytes
2. RSA-in-the-head
   - Soundness error: $1/n = 1/256$
   - Iterations: 16
   - Proof size: $16 \times (8 \times 128 + 2048) + 256 + 128 = 6.5$ KBytes

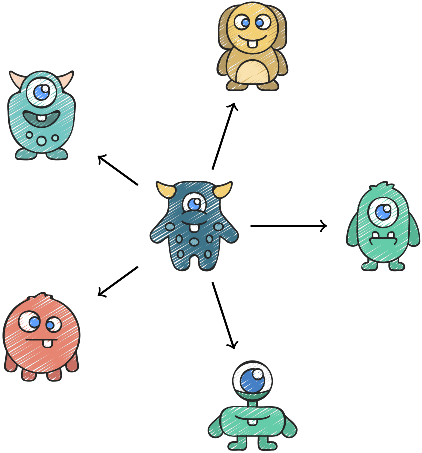
Beyond Linear functions?

- use **additive sharing** (n-out-of-n) in a finite field $\mathbb{F}$
- represent f using an arithmetic circuit over $\mathbb{F}$



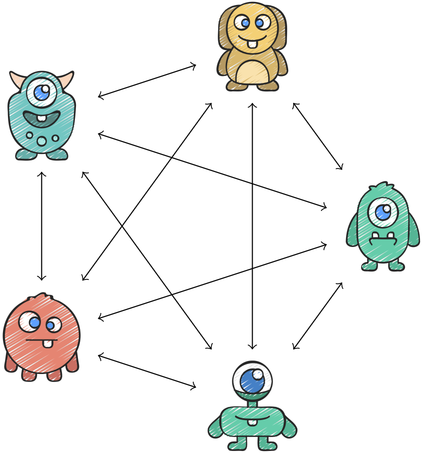
- linear gates are "easy"
- **How** to handle multiplication gates?

MPC with Pre-Processing



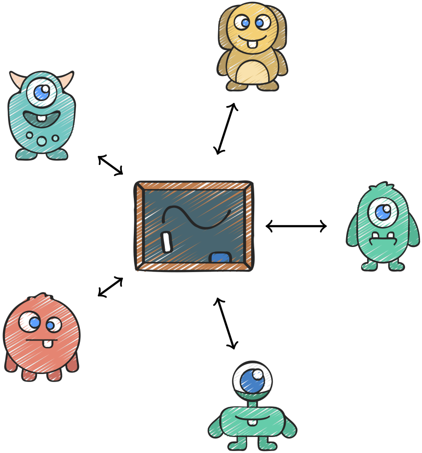
- parties obtain correlated secret inputs
- pre-processing is input independent

MPC with Pre-Processing



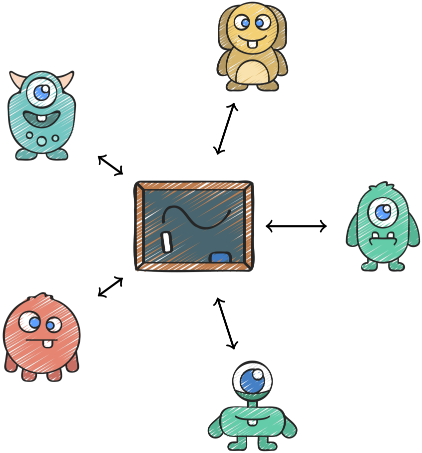
- parties obtain correlated secret inputs
- pre-processing is input independent
- parties then run the MPC protocol

MPC with Pre-Processing



- parties obtain correlated secret inputs
- pre-processing is input independent
- parties then run the MPC protocol
- lowers the cost (broadcast only)

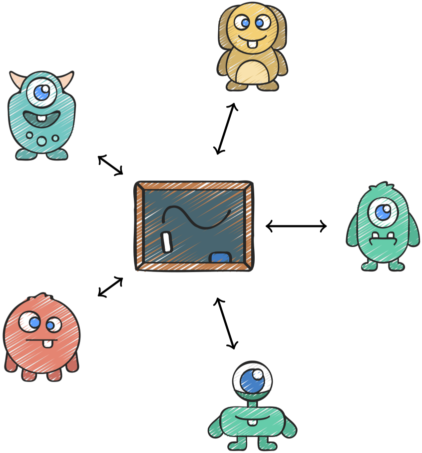
MPC with Pre-Processing



- parties obtain correlated secret inputs
- pre-processing is input independent
- parties then run the MPC protocol
- lowers the cost (broadcast only)

- **How to use it in MPC in the head?**

MPC with Pre-Processing



- parties obtain correlated secret inputs
- pre-processing is input independent
- parties then run the MPC protocol
- lowers the cost (broadcast only)

- **How to use it in MPC in the head?**
- …gives Alice more opportunities to cheat!

Beaver Triples

Beaver – Crypto 1991

$$[\![x]\!] = \textsc{Share}(x) = (x_1, \ldots, x_n) \quad [\![y]\!] = \textsc{Share}(y) = (y_1, \ldots, y_n)$$

- Given $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ where a and b are random and $c = a \cdot b$
- P_i computes $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$ and broadcasts them

$$\alpha = \alpha_1 + \cdots + \alpha_n = x - a \qquad \beta = \beta_1 + \cdots + \beta_n = y - b$$

- We have

$$\alpha \cdot \beta + \beta \cdot a + \alpha \cdot b + c = xy$$

- P_i computes $z_i = \alpha \cdot \beta + \beta \cdot a_i + \alpha \cdot b_i + c_i$

$$[\![z]\!] = (z_1, \ldots, z_n) \simeq \textsc{Share}(x \cdot y)$$

Beaver Triples

Beaver – Crypto 1991

$$[\![x]\!] = \textsc{Share}(x) = (x_1, \dots, x_n) \quad [\![y]\!] = \textsc{Share}(y) = (y_1, \dots, y_n)$$

- Given $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ where a and b are random and $c = a \cdot b$
- P_i computes $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$ and broadcasts them

$$\alpha = \alpha_1 + \cdots + \alpha_n = x - a \qquad \beta = \beta_1 + \cdots + \beta_n = y - b$$

- We have

$$\alpha \cdot \beta + \beta \cdot a + \alpha \cdot b + c = xy$$

- P_i computes $z_i = \alpha \cdot \beta + \beta \cdot a_i + \alpha \cdot b_i + c_i$

$$[\![z]\!] = (z_1, \dots, z_n) \simeq \textsc{Share}(x \cdot y)$$

Beaver Triples

Beaver – Crypto 1991

$$[\![x]\!] = \textsc{Share}(x) = (x_1, \ldots, x_n) \quad [\![y]\!] = \textsc{Share}(y) = (y_1, \ldots, y_n)$$

- Given $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ where a and b are random and $c = a \cdot b$
- P_i computes $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$ and broadcasts them

$$\alpha = \alpha_1 + \cdots + \alpha_n = x - a \qquad \beta = \beta_1 + \cdots + \beta_n = y - b$$

- We have

$$\alpha \cdot \beta + \beta \cdot a + \alpha \cdot b + c = xy$$

- P_i computes $z_i = \alpha \cdot \beta + \beta \cdot a_i + \alpha \cdot b_i + c_i$

$$[\![z]\!] = (z_1, \ldots, z_n) \simeq \textsc{Share}(x \cdot y)$$

Beaver Triples

Beaver – Crypto 1991

$$[\![x]\!] = \textsc{Share}(x) = (x_1, \ldots, x_n) \quad [\![y]\!] = \textsc{Share}(y) = (y_1, \ldots, y_n)$$

- Given $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ where a and b are random and $c = a \cdot b$
- P_i computes $\alpha_i = x_i - a_i$ and $\beta_i = y_i - b_i$ and broadcasts them

$$\alpha = \alpha_1 + \cdots + \alpha_n = x - a \qquad \beta = \beta_1 + \cdots + \beta_n = y - b$$

- We have

$$\alpha \cdot \beta + \beta \cdot a + \alpha \cdot b + c = xy$$

- P_i computes $z_i = \alpha \cdot \beta + \beta \cdot a_i + \alpha \cdot b_i + c_i$

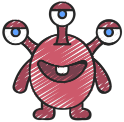
$$[\![z]\!] = (z_1, \ldots, z_n) \simeq \textsc{Share}(x \cdot y)$$

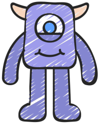
MPC in the Head with Beaver Triples

Katz, Kolesnikov, Wang – CCS 2018

$([\![a_i]\!], [\![b_i]\!], [\![c_i]\!]) \longleftarrow$ **Triples**

$$\xrightarrow{\{C_i = \textsc{Commit}([\![a_i]\!], [\![b_i]\!], [\![c_i]\!])\}_{i \in S}}$$





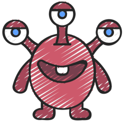
MPC in the Head with Beaver Triples

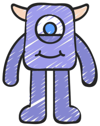
Katz, Kolesnikov, Wang – CCS 2018

$([\![a_i]\!], [\![b_i]\!], [\![c_i]\!]) \xleftarrow{\text{dice}}$ **Triples**

$$\xrightarrow{\{C_i = \textsc{Commit}([\![a_i]\!], [\![b_i]\!], [\![c_i]\!])\}_{i \in S}}$$

$$\xleftarrow{T \subset S}$$

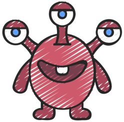




MPC in the Head with Beaver Triples

Katz, Kolesnikov, Wang – CCS 2018

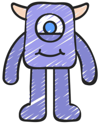
$(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket) \longleftarrow$ **Triples**



$$\xrightarrow{\{C_i = \textsc{Commit}(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket)\}_{i \in S}}$$

$$\xleftarrow{T \subset S}$$

$$\xrightarrow{\{\textsc{Open}(C_i)\}_{i \in T}}$$



MPC in the Head with Beaver Triples

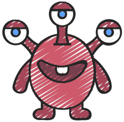
Katz, Kolesnikov, Wang – CCS 2018

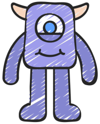
$(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket) \longleftarrow$ **Triples**

$\xrightarrow{\{C_i = \textsc{Commit}(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket)\}_{i \in S}}$

$\xleftarrow{T \subset S}$

$\xrightarrow{\{\textsc{Open}(C_i)\}_{i \in T}}$





Run MPC with triples in $S \setminus T$

$\downarrow$

$(\mathrm{View}_1, \ldots, \mathrm{View}_n)$

$\xrightarrow{\{C'_j = \textsc{Commit}(\mathrm{View}_j)\}_{1 \le j \le n}}$

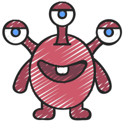
MPC in the Head with Beaver Triples

Katz, Kolesnikov, Wang – CCS 2018

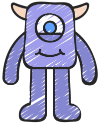
$([\![a_i]\!], [\![b_i]\!], [\![c_i]\!]) \longleftarrow$ **Triples**

$\xrightarrow{\{C_i = \textsc{Commit}([\![a_i]\!], [\![b_i]\!], [\![c_i]\!])\}_{i \in S}}$

$\xleftarrow{T \subset S}$



$\xrightarrow{\{\textsc{Open}(C_i)\}_{i \in T}}$



Run MPC with triples in $S \setminus T$

$\downarrow$

$(\mathrm{View}_1, \ldots, \mathrm{View}_n)$

$\xrightarrow{\{C'_j = \textsc{Commit}(\mathrm{View}_j)\}_{1 \le j \le n}}$

$\xleftarrow{j^\star}$

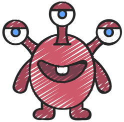
MPC in the Head with Beaver Triples

Katz, Kolesnikov, Wang – CCS 2018

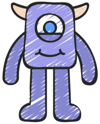
$(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket) \longleftarrow$ **Triples**

$\xrightarrow{\{C_i = \textsc{Commit}(\llbracket a_i \rrbracket, \llbracket b_i \rrbracket, \llbracket c_i \rrbracket)\}_{i \in S}}$

$\xleftarrow{T \subset S}$



$\xrightarrow{\{\textsc{Open}(C_i)\}_{i \in T}}$



Run MPC with triples in $S \setminus T$

$\downarrow$

$(\mathrm{View}_1, \ldots, \mathrm{View}_n)$

$\xrightarrow{\{C'_j = \textsc{Commit}(\mathrm{View}_j)\}_{1 \le j \le n}}$

$\xleftarrow{j^\star}$

$\xrightarrow{\{\textsc{Open}(C_j)_{j \neq j^\star}\}}$

Verifying is Cheaper than Computing . . .

- $|S \setminus T|$ = number of multiplication gates
- S has to be large enough to detect a cheating Alice with high probability
  (e.g. for $n = 128$, $\simeq \times 7$ overhead)

- **Idea:** Replace computing $x \cdot y$ from their sharing by committing $z = x \cdot y$ and checking that z is correct
- "sacrifice" a committed triple (a, b, c) with $c = a \cdot b$ (that is checked simultaneously)
  $\rightsquigarrow$ no need for **'cut and choose"**

Baum, Nof – PKC 2018

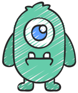
Verifying is Cheaper than Computing . . .

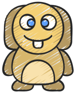
- $|S \setminus T|$ = number of multiplication gates
- S has to be large enough to detect a cheating Alice with high probability
  (e.g. for $n = 128$, $\simeq \times 7$ overhead)

- **Idea:** Replace computing $x \cdot y$ from their sharing by committing $z = x \cdot y$ and checking that z is correct
- "sacrifice" a committed triple (a, b, c) with $c = a \cdot b$ (that is checked simultaneously)
  $\leadsto$ no need for **'cut and choose"**

Baum, Nof – PKC 2018

Checking Products





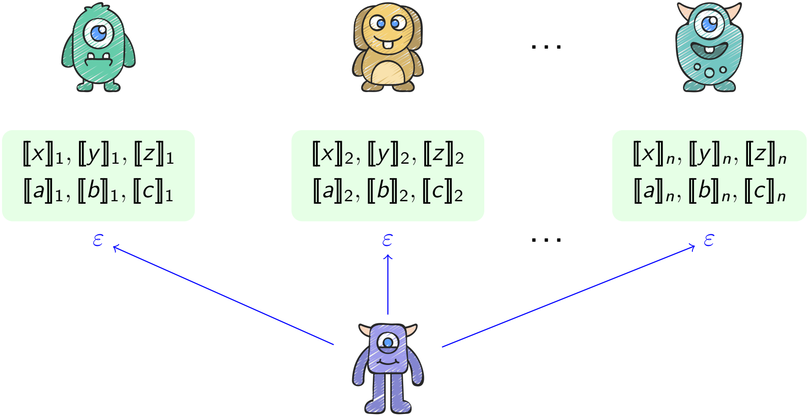
$\dots$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$
$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

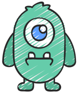
$[\![x]\!]_2, [\![y]\!]_2, [\![z]\!]_2$
$[\![a]\!]_2, [\![b]\!]_2, [\![c]\!]_2$

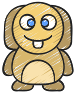
$[\![x]\!]_n, [\![y]\!]_n, [\![z]\!]_n$
$[\![a]\!]_n, [\![b]\!]_n, [\![c]\!]_n$

Checking Products



Checking Products





$\cdots$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$
$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$
$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$

$[\![x]\!]_2, [\![y]\!]_2, [\![z]\!]_2$
$[\![a]\!]_2, [\![b]\!]_2, [\![c]\!]_2$

ε

$[\![\alpha]\!]_2 = \varepsilon[\![x]\!]_2 + [\![a]\!]_2$
$[\![\beta]\!]_2 = [\![y]\!]_2 + [\![b]\!]_2$

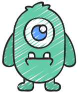
$\cdots$

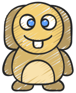
$[\![x]\!]_n, [\![y]\!]_n, [\![z]\!]_n$
$[\![a]\!]_n, [\![b]\!]_n, [\![c]\!]_n$

ε

$[\![\alpha]\!]_n = \varepsilon[\![x]\!]_n + [\![a]\!]_n$
$[\![\beta]\!]_n = [\![y]\!]_n + [\![b]\!]_n$

Checking Products





$\cdots$

$[\![x]\!]_1, [\![y]\!]_1, [\![z]\!]_1$
$[\![a]\!]_1, [\![b]\!]_1, [\![c]\!]_1$

$[\![x]\!]_2, [\![y]\!]_2, [\![z]\!]_2$
$[\![a]\!]_2, [\![b]\!]_2, [\![c]\!]_2$

$[\![x]\!]_n, [\![y]\!]_n, [\![z]\!]_n$
$[\![a]\!]_n, [\![b]\!]_n, [\![c]\!]_n$

ε ε $\cdots$ ε

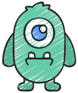
$$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1 + [\![a]\!]_1$$
$$[\![\beta]\!]_1 = [\![y]\!]_1 + [\![b]\!]_1$$

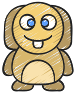
$$[\![\alpha]\!]_2 = \varepsilon[\![x]\!]_2 + [\![a]\!]_2$$
$$[\![\beta]\!]_2 = [\![y]\!]_2 + [\![b]\!]_2$$

$$[\![\alpha]\!]_n = \varepsilon[\![x]\!]_n + [\![a]\!]_n$$
$$[\![\beta]\!]_n = [\![y]\!]_n + [\![b]\!]_n$$

$[\![\alpha]\!]_1, [\![\beta]\!]_2$ $[\![\alpha]\!]_2, [\![\beta]\!]_2$ $\cdots$ $[\![\alpha]\!]_n, [\![\beta]\!]_n$

Checking Products





$\cdots$

$$\llbracket x\rrbracket_1, \llbracket y\rrbracket_1, \llbracket z\rrbracket_1 \quad \llbracket a\rrbracket_1, \llbracket b\rrbracket_1, \llbracket c\rrbracket_1$$

$$\llbracket x\rrbracket_2, \llbracket y\rrbracket_2, \llbracket z\rrbracket_2 \quad \llbracket a\rrbracket_2, \llbracket b\rrbracket_2, \llbracket c\rrbracket_2$$

$$\llbracket x\rrbracket_n, \llbracket y\rrbracket_n, \llbracket z\rrbracket_n \quad \llbracket a\rrbracket_n, \llbracket b\rrbracket_n, \llbracket c\rrbracket_n$$

ε $\quad$ ε $\quad \cdots \quad$ ε

$$\llbracket \alpha\rrbracket_1 = \varepsilon\llbracket x\rrbracket_1 + \llbracket a\rrbracket_1 \quad \llbracket \beta\rrbracket_1 = \llbracket y\rrbracket_1 + \llbracket b\rrbracket_1$$

$$\llbracket \alpha\rrbracket_2 = \varepsilon\llbracket x\rrbracket_2 + \llbracket a\rrbracket_2 \quad \llbracket \beta\rrbracket_2 = \llbracket y\rrbracket_2 + \llbracket b\rrbracket_2$$

$$\llbracket \alpha\rrbracket_n = \varepsilon\llbracket x\rrbracket_n + \llbracket a\rrbracket_n \quad \llbracket \beta\rrbracket_n = \llbracket y\rrbracket_n + \llbracket b\rrbracket_n$$

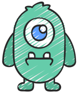
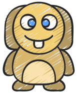
$\llbracket \alpha\rrbracket_1, \llbracket \beta\rrbracket_2$ $\quad$ $\llbracket \alpha\rrbracket_2, \llbracket \beta\rrbracket_2$ $\quad \cdots \quad$ $\llbracket \alpha\rrbracket_n, \llbracket \beta\rrbracket_n$

$$\llbracket v\rrbracket_1 = \varepsilon\llbracket z\rrbracket_1 + \alpha\llbracket b\rrbracket_1 - \llbracket c\rrbracket_1 + \beta\llbracket a\rrbracket_1 - \alpha\beta$$

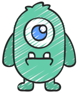
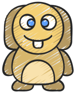
$$\llbracket v\rrbracket_2 = \varepsilon\llbracket z\rrbracket_2 + \alpha\llbracket b\rrbracket_2 - \llbracket c\rrbracket_2 + \beta\llbracket a\rrbracket_2 - \alpha\beta$$

$$\llbracket v\rrbracket_n = \varepsilon\llbracket z\rrbracket_n + \alpha\llbracket b\rrbracket_n - \llbracket c\rrbracket_n + \beta\llbracket a\rrbracket_n - \alpha\beta$$

Checking Products

		$\cdots$	
$\llbracket x\rrbracket_1, \llbracket y\rrbracket_1, \llbracket z\rrbracket_1$ $\llbracket a\rrbracket_1, \llbracket b\rrbracket_1, \llbracket c\rrbracket_1$	$\llbracket x\rrbracket_2, \llbracket y\rrbracket_2, \llbracket z\rrbracket_2$ $\llbracket a\rrbracket_2, \llbracket b\rrbracket_2, \llbracket c\rrbracket_2$		$\llbracket x\rrbracket_n, \llbracket y\rrbracket_n, \llbracket z\rrbracket_n$ $\llbracket a\rrbracket_n, \llbracket b\rrbracket_n, \llbracket c\rrbracket_n$
ε	ε	$\cdots$	ε
$\llbracket\alpha\rrbracket_1 = \varepsilon\llbracket x\rrbracket_1 + \llbracket a\rrbracket_1$ $\llbracket\beta\rrbracket_1 = \llbracket y\rrbracket_1 + \llbracket b\rrbracket_1$	$\llbracket\alpha\rrbracket_2 = \varepsilon\llbracket x\rrbracket_2 + \llbracket a\rrbracket_2$ $\llbracket\beta\rrbracket_2 = \llbracket y\rrbracket_2 + \llbracket b\rrbracket_2$		$\llbracket\alpha\rrbracket_n = \varepsilon\llbracket x\rrbracket_n + \llbracket a\rrbracket_n$ $\llbracket\beta\rrbracket_n = \llbracket y\rrbracket_n + \llbracket b\rrbracket_n$
$\llbracket\alpha\rrbracket_1, \llbracket\beta\rrbracket_2$	$\llbracket\alpha\rrbracket_2, \llbracket\beta\rrbracket_2$	$\cdots$	$\llbracket\alpha\rrbracket_n, \llbracket\beta\rrbracket_n$
$\llbracket v\rrbracket_1 = \varepsilon\llbracket z\rrbracket_1 + \alpha\llbracket b\rrbracket_1$ $-\llbracket c\rrbracket_1 + \beta\llbracket a\rrbracket_1 - \alpha\beta$	$\llbracket v\rrbracket_2 = \varepsilon\llbracket z\rrbracket_2 + \alpha\llbracket b\rrbracket_2$ $-\llbracket c\rrbracket_2 + \beta\llbracket a\rrbracket_2 - \alpha\beta$		$\llbracket v\rrbracket_n = \varepsilon\llbracket z\rrbracket_n + \alpha\llbracket b\rrbracket_n$ $-\llbracket c\rrbracket_n + \beta\llbracket a\rrbracket_n - \alpha\beta$
$\llbracket v\rrbracket_1$	$\llbracket v\rrbracket_2$	$\cdots$	$\llbracket v\rrbracket_n$

Checking Products

  … 

$[\![x]\!]_1, [\![y]\!]_1,$ … $[\![x]\!]_n, [\![y]\!]_n, [\![z]\!]_n$

$[\![a]\!]_1, [\![b]\!]_1,$ … $[\![a]\!]_n, [\![b]\!]_n, [\![c]\!]_n$

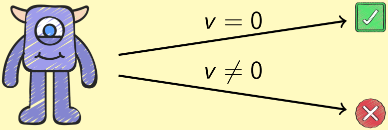
ε … ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1$ … $= \varepsilon[\![x]\!]_n + [\![a]\!]_n$

$[\![\beta]\!]_1 = [\![y]\!]_1$ … $= [\![y]\!]_n + [\![b]\!]_n$

$[\![\alpha]\!]_1, [\![\beta$ … $[\![\alpha]\!]_n, [\![\beta]\!]_n$

$$v = \varepsilon(xy - z) + (ab - c)$$

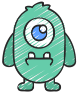


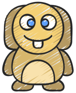
$[\![v]\!]_1 = \varepsilon[\![z]\!]_1$ … $= \varepsilon[\![z]\!]_n + \alpha[\![b]\!]_n$

$-[\![c]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$ $\quad -[\![c]\!]_2 + \beta[\![a]\!]_2 - \alpha\beta$ $\quad -[\![c]\!]_n + \beta[\![a]\!]_n - \alpha\beta$

$[\![v]\!]_1$ $\quad [\![v]\!]_2$ … $[\![v]\!]_n$

Checking Products



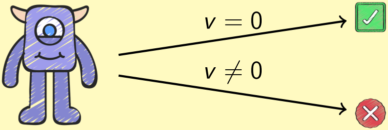


$\cdots$

$[\![x]\!]_1, [\![y]\!]_1,$
$[\![a]\!]_1, [\![b]\!]_1,$

$[\![x]\!]_n, [\![y]\!]_n, [\![z]\!]_n$
$[\![a]\!]_n, [\![b]\!]_n, [\![c]\!]_n$

$$v = \varepsilon(xy - z) + (ab - c)$$



Soundness error: $1/|\mathbb{F}|$

ε ε

$[\![\alpha]\!]_1 = \varepsilon[\![x]\!]_1$
$[\![\beta]\!]_1 = [\![y]\!]_1$

$= \varepsilon[\![x]\!]_n + [\![a]\!]_n$
$= [\![y]\!]_n + [\![b]\!]_n$

$[\![\alpha]\!]_1, [\![\beta$

$[\![\alpha]\!]_n, [\![\beta]\!]_n$

$[\![v]\!]_1 = \varepsilon[\![z]\!]_1$
$-[\![c]\!]_1 + \beta[\![a]\!]_1 - \alpha\beta$

$-[\![c]\!]_2 + \beta[\![a]\!]_2 - \alpha\beta$

$= \varepsilon[\![z]\!]_n + \alpha[\![b]\!]_n$
$-[\![c]\!]_n + \beta[\![a]\!]_n - \alpha\beta$

$[\![v]\!]_1$ $[\![v]\!]_2$ $\cdots$ $[\![v]\!]_n$

Example: Subset-Sum

Given $(w_1, \ldots, w_\ell, t) \in (\mathbb{Z}/p\mathbb{Z})^{\ell+1}$, find $(x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ s.t.

$$w_1 \cdot x_1 + \cdots + w_\ell \cdot x_\ell = t \bmod p$$

- **Linear relation:** $w_1 \cdot x_1 + \cdots + w_\ell \cdot x_\ell = t \bmod p$
- $x_i \in \{0,1\} \xrightarrow{\text{Arithmetization}} x_i(x_i - 1) = 0 \bmod p$
  $\rightsquigarrow \ell$ triples $\rightsquigarrow 2\ell$ auxiliary values + Tree PRG
- For $\ell = \lceil \log_2(p) \rceil = 256$, $n = 256 \rightsquigarrow 264$ KB!

Shamir – Unpublished, 1986	1186 KB
Ling, Nguyen, Stehlé, Wang – PKC 2013	2350 KB
Beullens – Eurocrypt 2020	122 KB
Feneuil, Maire, Rivain, V. – Asiacrypt 2022	16 KB

Example: Subset-Sum

Given $(w_1, \ldots, w_\ell, t) \in (\mathbb{Z}/p\mathbb{Z})^{\ell+1}$, find $(x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ s.t.

$$w_1 \cdot x_1 + \cdots + w_\ell \cdot x_\ell = t \bmod p$$

- **Linear relation:** $w_1 \cdot x_1 + \cdots + w_\ell \cdot x_\ell = t \bmod p$
- $x_i \in \{0,1\} \xrightarrow{\text{Arithmetization}} x_i(x_i - 1) = 0 \bmod p$
  $\rightsquigarrow \ell$ triples $\rightsquigarrow 2\ell$ auxiliary values + Tree PRG
- For $\ell = \lceil \log_2(p) \rceil = 256$, $n = 256 \rightsquigarrow 264$ KB!

Shamir – Unpublished, 1986	1186 KB
Ling, Nguyen, Stehlé, Wang – PKC 2013	2350 KB
Beullens – Eurocrypt 2020	122 KB
Feneuil, Maire, Rivain, V. – Asiacrypt 2022	16 KB

Conclusion

- MPC-in-the-Head is fun!
- Efficient and short ZK proofs for one-way functions
  - $\rightsquigarrow$ post-quantum signatures (but not only!)
- Many efficiency/communication improvements in the last 5 years (attend Thibauld's thesis on Monday!)
- Join the game:
  - pick your favorite OWF
  - find a cute, MPC-friendly arithmetization
  - get an efficient signature scheme!

Conclusion

- MPC-in-the-Head is fun!
- Efficient and short ZK proofs for one-way functions
  - $\rightsquigarrow$ post-quantum signatures (but not only!)
- Many efficiency/communication improvements in the last 5 years (attend Thibauld's thesis on Monday!)
- Join the game:
  - pick your favorite OWF
  - find a cute, MPC-friendly arithmetization
  - get an efficient signature scheme!