Multi-Party Computation in the Head: Techniques and Applications

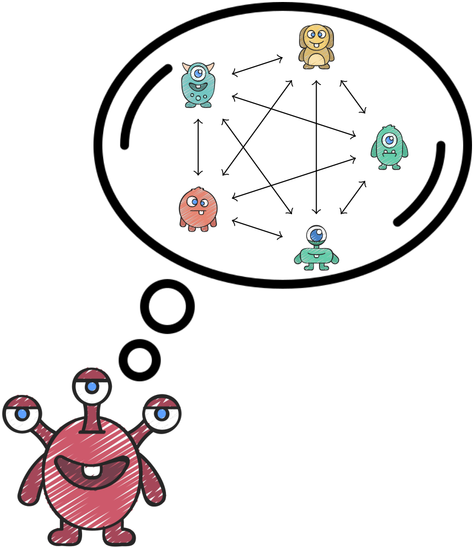
Damien Vergnaud

Sorbonne Université – LIP6

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

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)

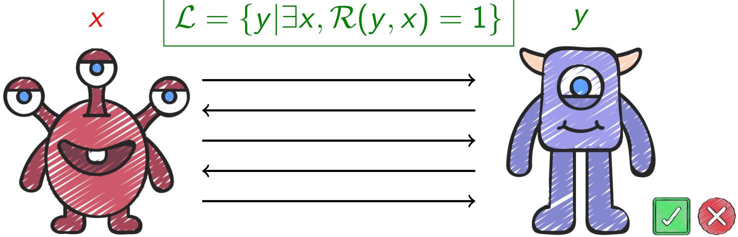
  that she knows sk such that $\mathrm{pk} = F(\mathrm{sk})$

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

Zero-knowledge interactive proof

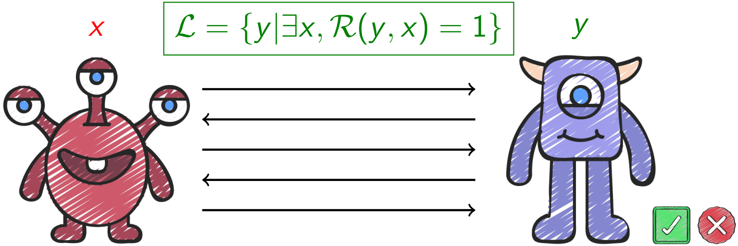
Goldwasser, Micali, Rackoff – STOC 1985 **(1993 Gödel Prize)**
Goldreich, Micali, Wigderson – FOCS 1986

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



Zero-knowledge interactive proof

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

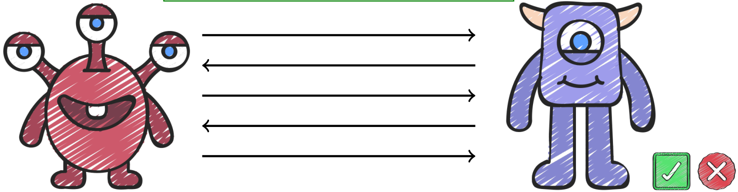


Completeness

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



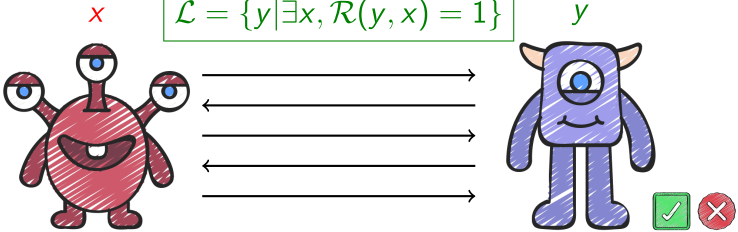
Completeness

(Knowledge) Soundness

Zero-knowledge interactive proof

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

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



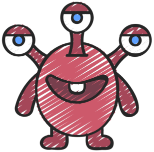
Completeness

(Knowledge)
Soundness

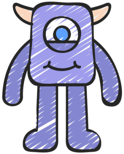
(Honest-Verifier)
Zero-knowledge

Guillou-Quisquater Protocol (ZK for RSA – 1988)

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

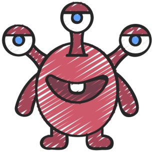


x ——————————————— y

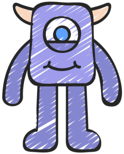


Guillou-Quisquater Protocol (ZK for RSA – 1988)

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



x y

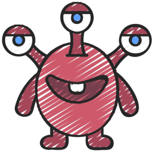


$r \xleftarrow{} (\mathbb{Z}/N\mathbb{Z})^*$

$k \leftarrow r^e \bmod N$

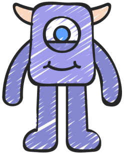
Guillou-Quisquater Protocol (ZK for RSA – 1988)

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



x

y



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

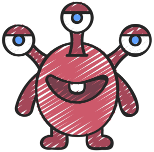
$k \leftarrow r^e \bmod N$

$\xrightarrow{k}$

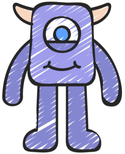
$c \xleftarrow{\text{⚂⚃}} \{0, \ldots, e-1\}$

Guillou-Quisquater Protocol (ZK for RSA – 1988)

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



x ———————————— y



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

$k \leftarrow r^e \bmod N$

$\xrightarrow{\quad k \quad}$

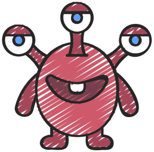
$c \xleftarrow{\$} \{0, \ldots, e-1\}$

$\xleftarrow{\quad c \quad}$

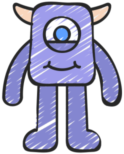
$s \leftarrow r x^c \bmod N$

Guillou-Quisquater Protocol (ZK for RSA – 1988)

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



x y



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

$k \leftarrow r^e \bmod N$

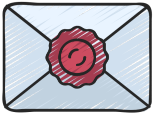
$\xrightarrow{\;k\;}$

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

$\xleftarrow{\;c\;}$

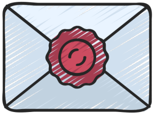
$s \leftarrow rx^c \bmod N \quad \xrightarrow{\;s\;} \quad s^e \stackrel{?}{\equiv} ky^c \bmod N$

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 $\perp$

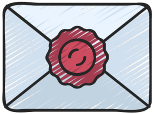
Commitments



- 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

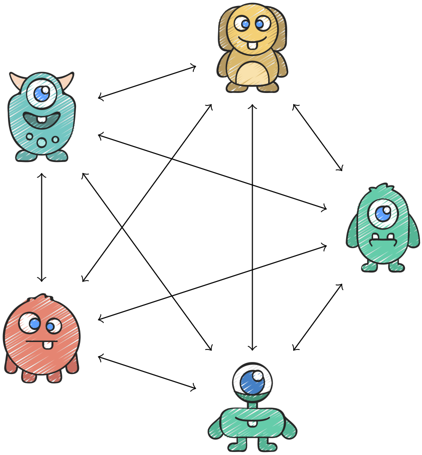


- digital analogue of a sealed envelope
  $\rightsquigarrow$ hide a value that cannot be changed
- $(\textsc{Commit}, \textsc{Open})$
  - $\textsc{Commit}(m; r) \rightsquigarrow (c, s)$
  - $\textsc{Open}(c, s) \rightsquigarrow m$ or $\perp$

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 $P_1, \ldots, P_n$ 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

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

- 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 $(\textsc{Share}, \textsc{Reconstruct})$:

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

- 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.

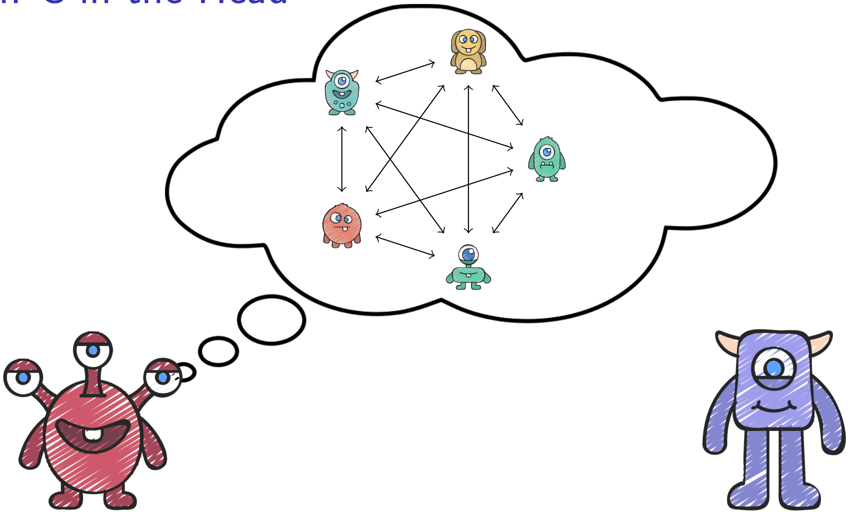
Views of Parties in MPC

- 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.

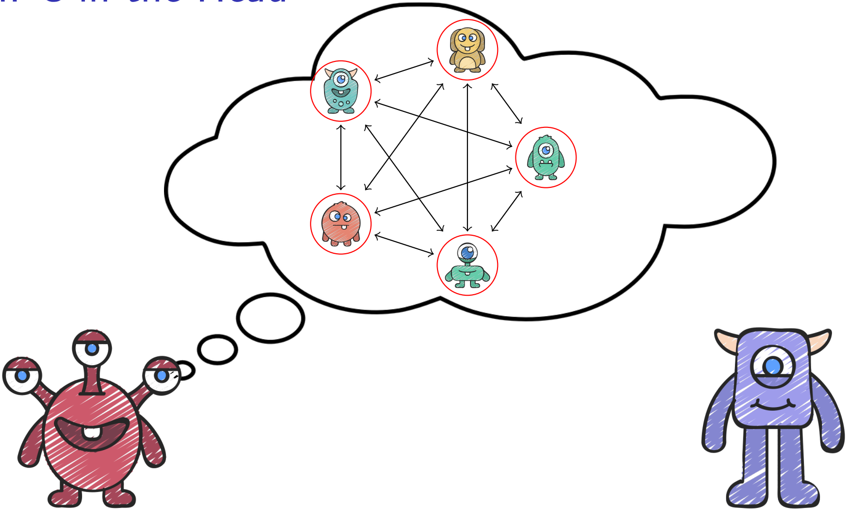
Views of Parties in MPC

- 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.

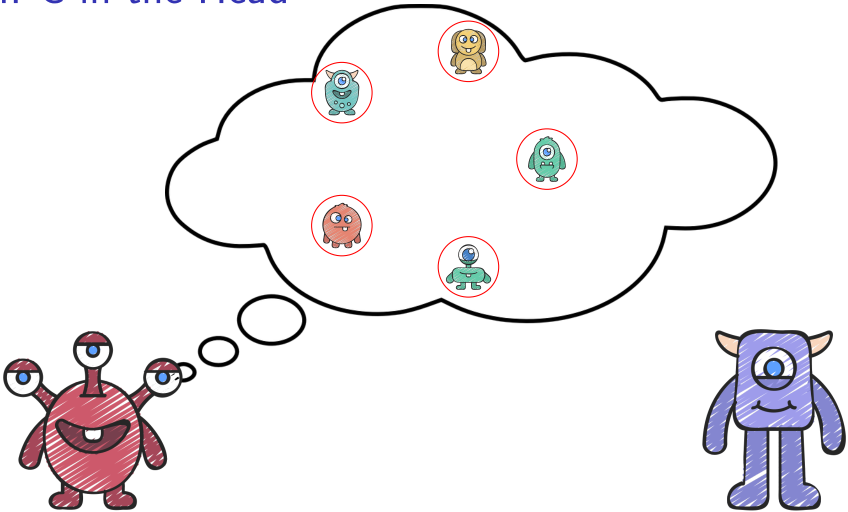
MPC-in-the-Head



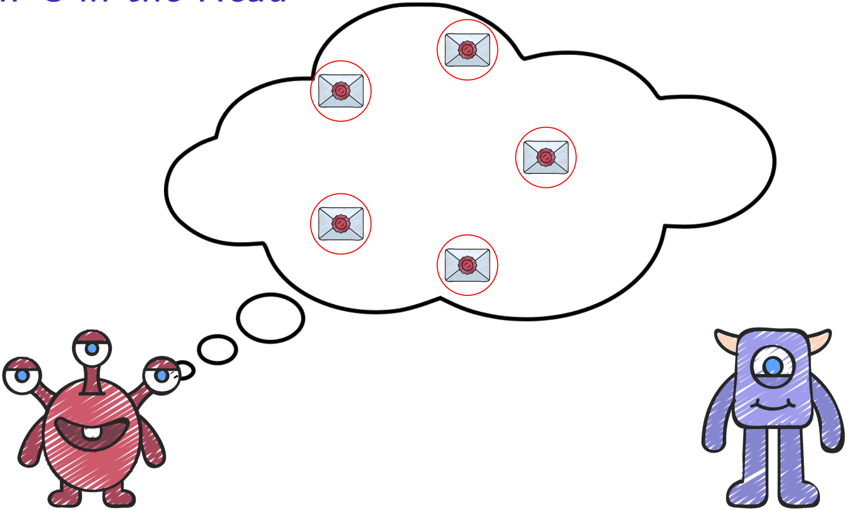
MPC-in-the-Head



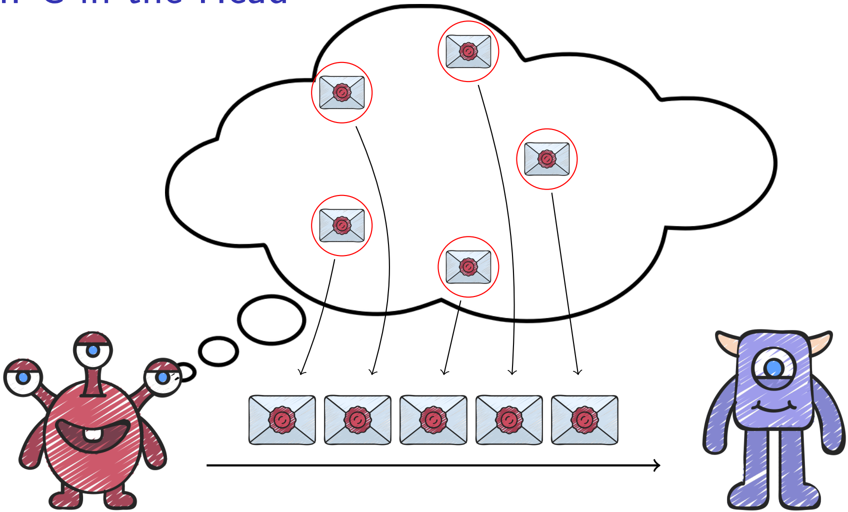
MPC-in-the-Head



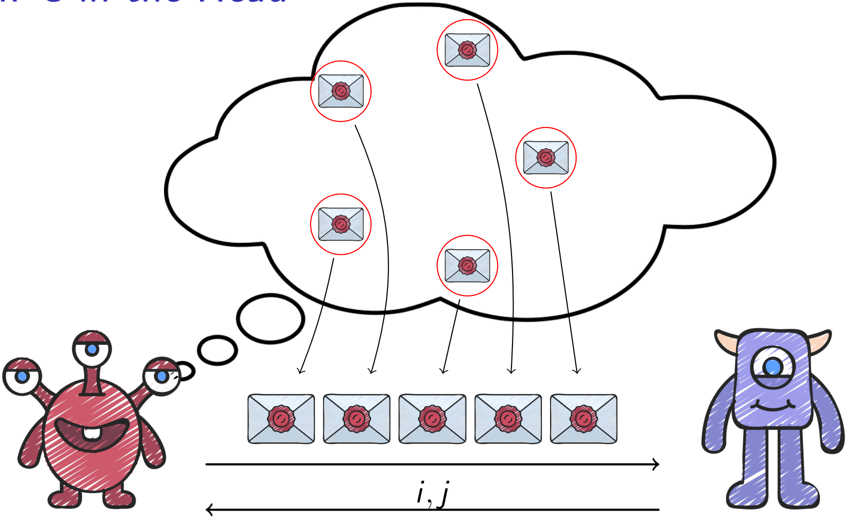
MPC-in-the-Head



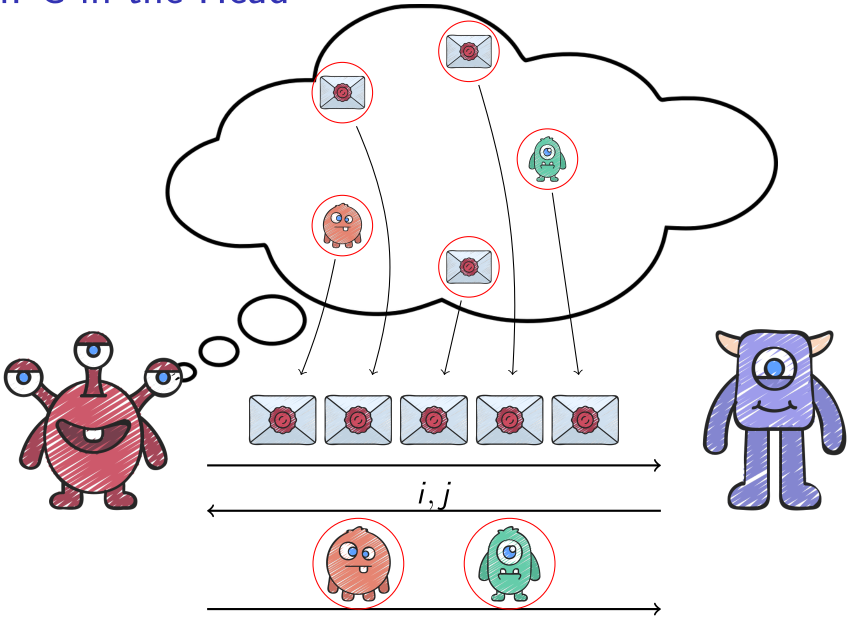
MPC-in-the-Head



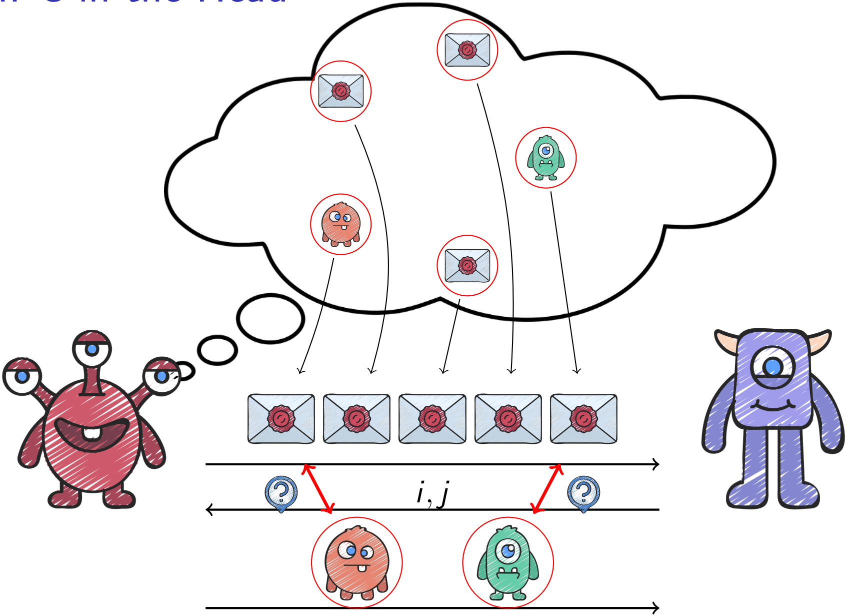
MPC-in-the-Head



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
  $\leadsto$ 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 $\leq (9/10)^k$
    $\leadsto$ 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)

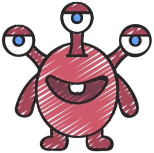
- **but**, possible for "linear" functions, e.g. for $a, b \in \mathbb{Z}$:

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

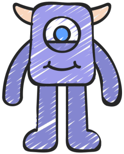
where $a \cdot x = \underbrace{x \boxplus \cdots \boxplus x}_{a \text{ times}}$ (for $a \geq 0$)

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

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

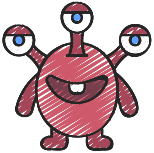


x $\qquad\qquad$ y

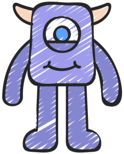


RSA-in-the-head (Maire-V. 2023)

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



x ____________________ y

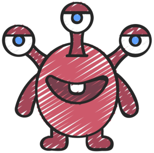


$\llbracket x \rrbracket \xleftarrow{\quad} [(\mathbb{Z}/N\mathbb{Z})^*]^n$

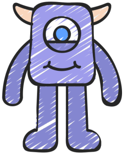
$(y_i \leftarrow x_i^e \bmod N)_i$

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

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



x y



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

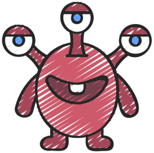
$(y_i \leftarrow x_i^e \bmod N)_i$

$\xrightarrow{\vec{y}}$

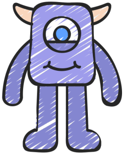
$i^\star \xleftarrow{} \{1, \ldots, n\}$

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

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



x ——————————— y



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

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

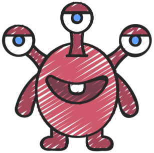
$\xrightarrow{\vec{y}}$

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

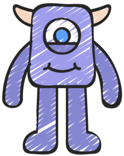
$\xleftarrow{i^\star}$

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

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



x $\qquad$ y



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

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

$\xrightarrow{\vec{y}}$

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

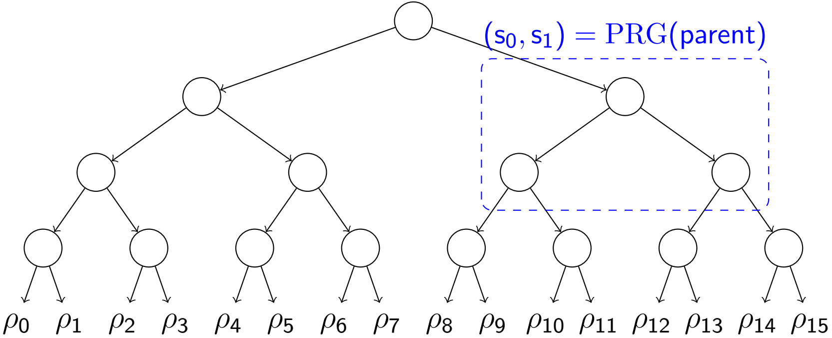
$\xleftarrow{i^\star}$

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

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

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

Compressing the Proof - Tree PRG



Commitment

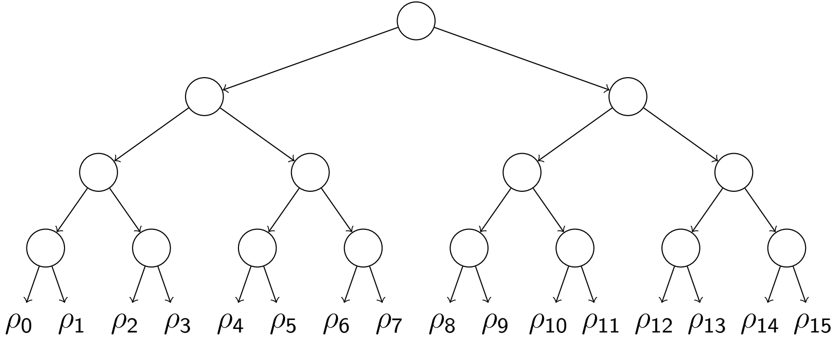
1. $(x_i, r_i) = \mathrm{PRG}(\rho_i)$
2. r_n, r picked at random
3. $c_i = H(y_i, r_i)$
4. $c = H(c_0, \ldots, c_n, r)$ and Δ_x

Response

Alice reveals r and

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

Compressing the Proof - Tree PRG



Commitment

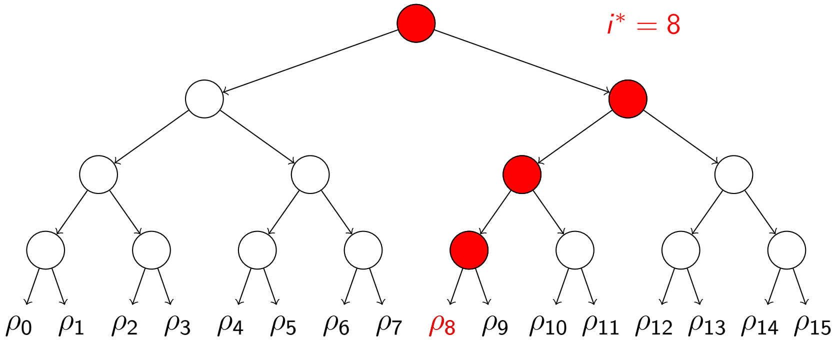
1. $(x_i, r_i) = \mathrm{PRG}(\rho_i)$
2. r_n, r picked at random
3. $c_i = H(y_i, r_i)$
4. $c = H(c_0, \ldots, c_n, r)$ and Δ_x

Response

Alice reveals r and

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

Compressing the Proof - Tree PRG



Commitment

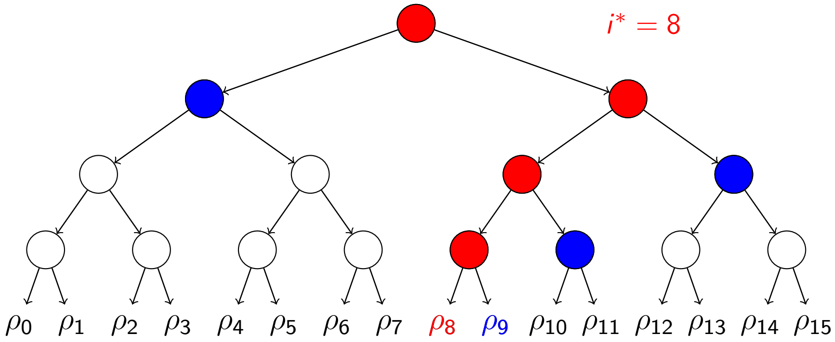
1. $(x_i, r_i) = \mathrm{PRG}(\rho_i)$
2. r_n, r picked at random
3. $c_i = H(y_i, r_i)$
4. $c = H(c_0, \ldots, c_n, r)$ and Δ_x

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

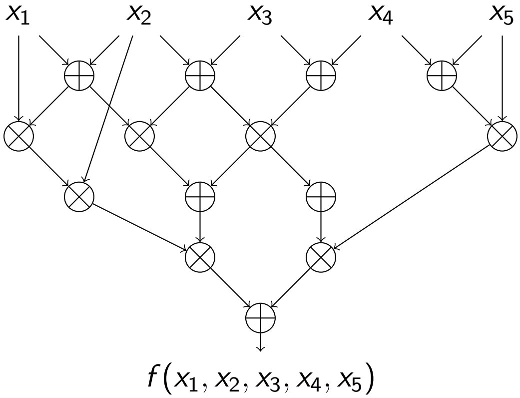
$$\boxed{e = 17,\ N \simeq 2^{2048},\ \lambda = 128}$$

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

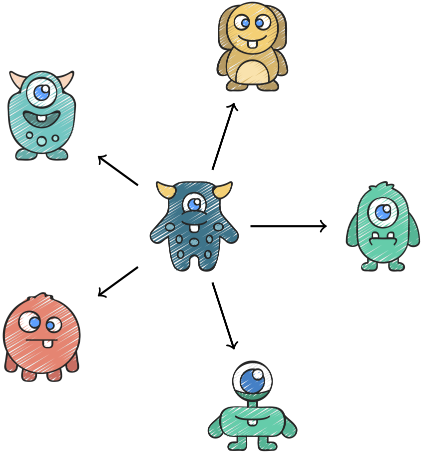
Beyond Linear functions?

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



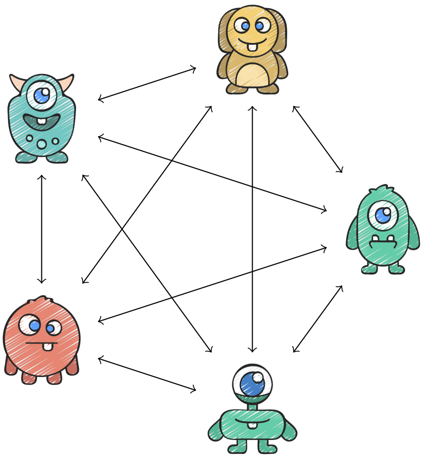
- linear gates are "easy"
- **How** to handle multiplication gates?

MPC with Pre-Processing



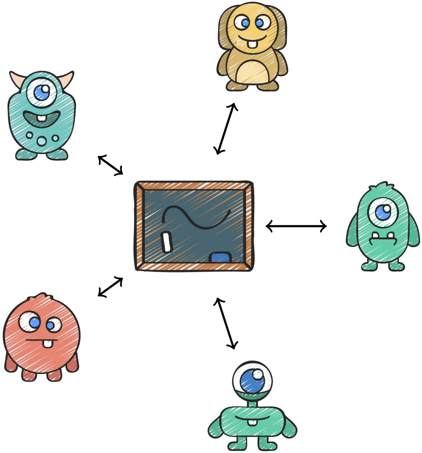
- parties obtain correlated secret inputs
- pre-processing is input independent

MPC with Pre-Processing



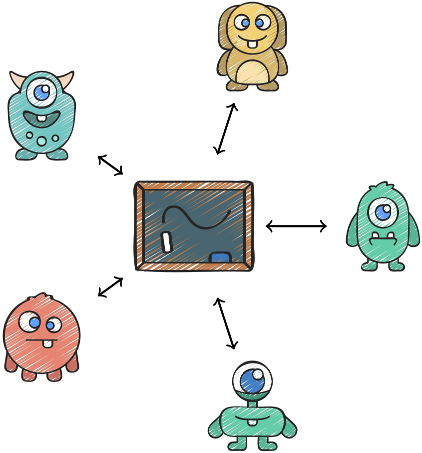
- parties obtain correlated secret inputs
- pre-processing is input independent
- parties then run the MPC protocol

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)

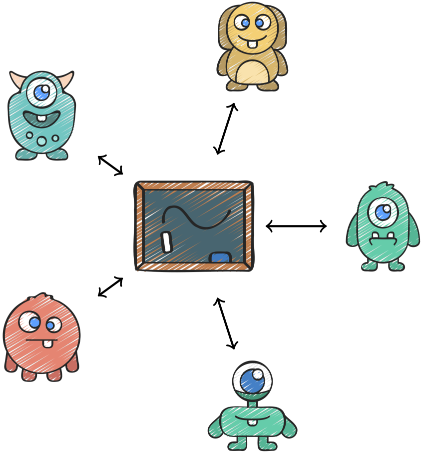
MPC with Pre-Processing



- 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!

Multiplication with Pre-Processing

$$[\![x]\!] = \textsc{Share}(x) \quad [\![y]\!] = \textsc{Share}(y) \quad [\![z]\!] = \textsc{Share}(x \cdot y)$$

- **Beaver (C 1991) / Katz, Kolesnikov, Wang (CCS 2018)**
  - given $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ where a and b are random and $c = a \cdot b$
  - compute $[\![\alpha]\!] = [\![x - a]\!]$, $[\![\beta]\!] = [\![y - b]\!]$ s.t.

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

$\rightsquigarrow$ need for **'cut and choose"**

- **Baum-Nof (PKC 2020)**
  - **Idea:** Replace computing $[\![z]\!]$ by committing it and checking that it is correct
  - "sacrifice" a triple $([\![a]\!], [\![b]\!], [\![c]\!])$ with $c = a \cdot b$
    $\rightsquigarrow$ checked simultaneously!

Multiplication with Pre-Processing

$$[\![x]\!] = \textsc{Share}(x) \quad [\![y]\!] = \textsc{Share}(y) \quad [\![z]\!] = \textsc{Share}(x \cdot y)$$

- **Beaver (C 1991) / Katz, Kolesnikov, Wang (CCS 2018)**
  - given $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ where a and b are random and $c = a \cdot b$
  - compute $[\![\alpha]\!] = [\![x - a]\!]$, $[\![\beta]\!] = [\![y - b]\!]$ s.t.

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

$\rightsquigarrow$ need for **'cut and choose"**

- **Baum-Nof (PKC 2020)**
  - **Idea:** Replace computing $[\![z]\!]$ by committing it and checking that it is correct
  - "sacrifice" a triple $([\![a]\!], [\![b]\!], [\![c]\!])$ with $c = a \cdot b$
  $\rightsquigarrow$ checked simultaneously!

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 (see presentations byThibauld, Antoine and Jules!)
- 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 (see presentations byThibauld, Antoine and Jules!)
- Join the game:
  - pick your favorite OWF
  - find a cute, MPC-friendly arithmetization
  - get an efficient signature scheme!

Bonne retraite Jean Claude !