# Secret sharing-based FHE

## Actively Secure MPC in the Dishonest Majority Setting: Achieving Constant Complexity in Online Communication, Computation Per Gate, Rounds, and Private Input Size

**Seunghwan Lee (Speaker)**[1,2], Jaesang Noh[2], Taejong Kim[2], Dohyuk Kim[1,2], and Dong-Joon Shin[1,2]

waLLLnut Co., Ltd.[1] and University of Hanyang[2]

*shlee@walllnut.com and kr3951@hanyang.ac.kr*

Presentation in Crypto25
August 18, 2025

# Presentation Overview

# Contribution Overview

# Secure Multi-Party Computation (MPC)

**MPC :**

- Allows multiple parties to jointly compute a function
- Inputs remain private throughout the computation

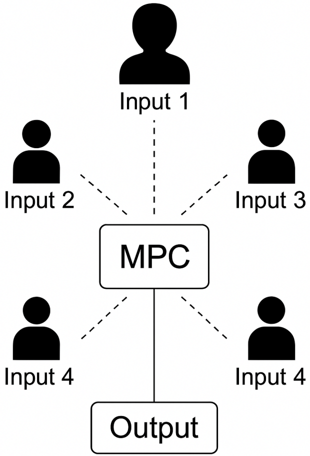
# Secure Multi-Party Computation (MPC)

**MPC :**

- Allows multiple parties to jointly compute a function
- Inputs remain private throughout the computation

**Security Models:**

- **Honest Majority Setting:**
  - Majority of parties follow the protocol
  - Enables **Guaranteed Output Delivery (GOD)**
- **Dishonest Majority Setting:**
  - Majority may be corrupted
  - At most **Security with Abort** can be guaranteed



# Dishonest Majority MPC: BMR vs SPDZ

**BMR Protocol (Circuit garbling)[3]:**

# Dishonest Majority MPC: BMR vs SPDZ

**BMR Protocol (Circuit garbling)[3]:**

- Constant-round MPC protocol
- Each gate must be **garbled and sent** to all parties
- the concrete communication cost of preprocessing/garbling is pretty larger than SPDZ

# Dishonest Majority MPC: BMR vs SPDZ

**BMR Protocol (Circuit garbling)[3]:**

- Constant-round MPC protocol
- Each gate must be **garbled and sent** to all parties
- the concrete communication cost of preprocessing/garbling is pretty larger than SPDZ

**SPDZ Protocol (secret sharing)[4]:**

- Uses information-theoretic MACs for integrity
- Small communication cost and very efficient in practice
- Online phase requires the number of non-constant communication round

# Dishonest Majority MPC: BMR vs SPDZ

**BMR Protocol (Circuit garbling)[3]:**

- Constant-round MPC protocol
- Each gate must be **garbled and sent** to all parties
- the concrete communication cost of preprocessing/garbling is pretty larger than SPDZ

**SPDZ Protocol (secret sharing)[4]:**

- Uses information-theoretic MACs for integrity
- Small communication cost and very efficient in practice
- Online phase requires the number of non-constant communication round

**Both protocols requires $\Omega(n|\mathcal{C}|)$ communications for a given circuit $\mathcal{C}$ with $|\mathcal{C}|$= $c_{\text{in}} + c_{\text{out}} + c_{\text{gate}}$**

# Dishonest Majority MPC: BMR vs SPDZ

**BMR Protocol (Circuit garbling)[3]:**

- Constant-round MPC protocol
- Each gate must be **garbled and sent** to all parties
- the concrete communication cost of preprocessing/garbling is pretty larger than SPDZ

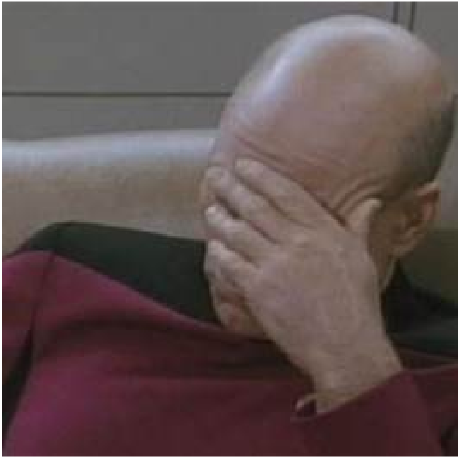
**SPDZ Protocol (secret sharing)[4]:**

- Uses information-theoretic MACs for integrity
- Small communication cost and very efficient in practice
- Online phase requires the number of non-constant communication round

**Both protocols requires $\Omega(n|\mathcal{C}|)$ communications for a given circuit $\mathcal{C}$ with $|\mathcal{C}|$= $c_{\text{in}} + c_{\text{out}} + c_{\text{gate}}$**

*How about FHE-based MPC? (Threshold FHE)*

# FHE-based MPC ...



# Practical Limitations of FHE-Based MPC

**Apparently, FHE-based MPC seems to reduce the cost from $\Omega(n(c_{\text{in}} + c_{\text{out}} + c_{\text{gate}}))$ to $\Omega(n(c_{\text{in}} + c_{\text{out}}))$, but...**

# Practical Limitations of FHE-Based MPC

**Apparently, FHE-based MPC seems to reduce the cost from $\Omega(n(c_{\text{in}} + c_{\text{out}} + c_{\text{gate}}))$ to $\Omega(n(c_{\text{in}} + c_{\text{out}}))$, but...**

- High Computational Overhead:
  - FHE operations are significantly slower than plaintext equivalents

# Practical Limitations of FHE-Based MPC

**Apparently, FHE-based MPC seems to reduce the cost from $\Omega(n(c_{\text{in}} + c_{\text{out}} + c_{\text{gate}})))$ to $\Omega(n(c_{\text{in}} + c_{\text{out}})))$, but...**

- High Computational Overhead:
  - FHE operations are significantly slower than plaintext equivalents

  Performance Degrades with Number of Parties:
  - Computation cost increases exponentially with the number of parties

# Practical Limitations of FHE-Based MPC

**Apparently, FHE-based MPC seems to reduce the cost from $\Omega(n(c_{\text{in}} + c_{\text{out}} + c_{\text{gate}}))$ to $\Omega(n(c_{\text{in}} + c_{\text{out}}))$, but...**

- High Computational Overhead:
  - FHE operations are significantly slower than plaintext equivalents
- Performance Degrades with Number of Parties:
  - Computation cost increases exponentially with the number of parties
- Security Challenges under Dishonest Majority:
  - Active security typically requires zero-knowledge proofs.
  - ZKPs introduce heavy overhead and complex protocol logic.

# Key Challenges in FHE-based MPC (PPT in NIST MPTS 2023)

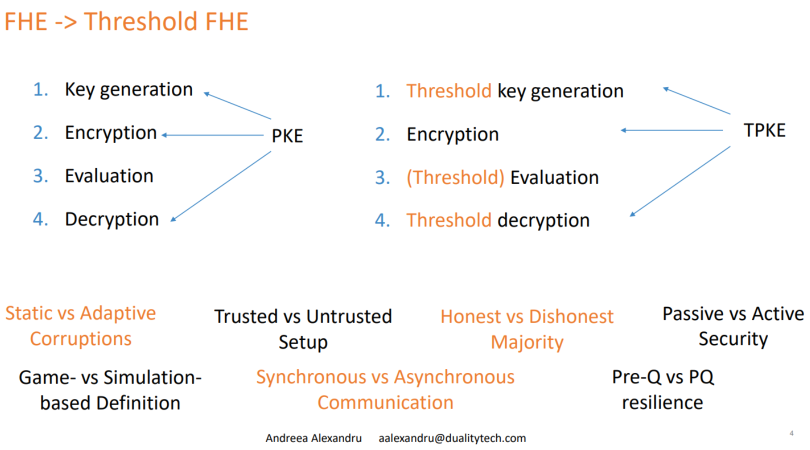



Figure: Content on Page 4 of the NIST MPTS 2023 PPT [13]

# Key Challenges in FHE-based MPC (PPT in NIST MPTS 2023)

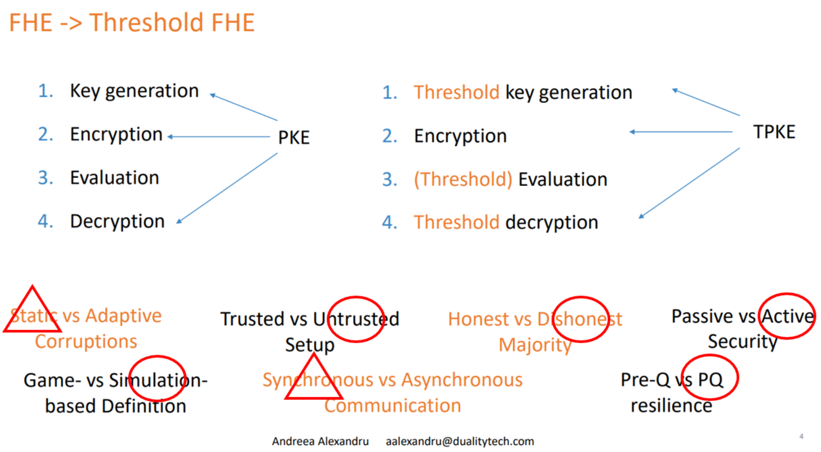



Figure: Today's FHE-based MPC properties

# Comparison of FHE-based MPC Protocols

- Our solution: secret sharing (SPDZ) + FHE + pre-processing model $\rightarrow$ SSFHE

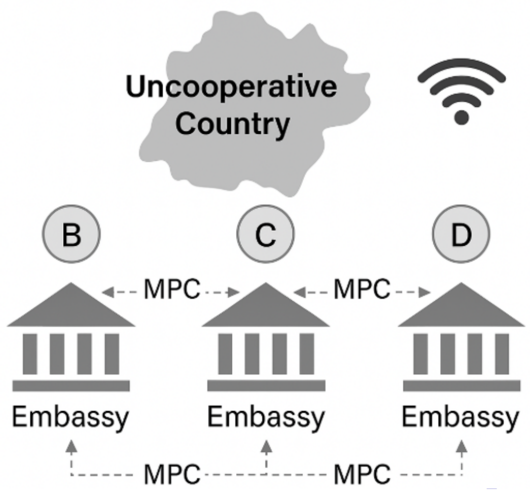
| | Types | $\lvert ct \rvert$ | $\lvert ev \rvert$ | $\lvert evGen \rvert$ |
|---|---|---|---|---|
| GLS15 [5] | ThHE | $O(nd_{\lambda,n}^2 \log d_{\lambda,n})$ | 0 | 0 |
| JS23 [6] | ThHE | $O(d_{\lambda,n})$ | $O(d_{\lambda,n}^2)$ | $O(nd_{\lambda,n}^2)$ |
| DDE+23 [7] | ThHE | $O(d_\lambda)$ | $O(d_\lambda^2)$ | unspecified |
| CCS19 [8] | MkHE | $O(nd_{\lambda,n})$ | $O(n^2 d_{\lambda,n}^2)$ | $O(n^2 d_{\lambda,n}^2)$ |
| TLX+21 [9] | MkHE | $O(d_{\lambda,n})$ | $O(d_{\lambda,n}^2)$ | $O(nd_{\lambda,n}^2)$ |
| Ours | ThHE | $O(d_\lambda)$ | $O(d_\lambda^2)$ | $O(n\kappa d_\lambda^2)$ |

| | Majority | Security | Gate speed | NTT-friendly |
|---|---|---|---|---|
| GLS15 [5] | Honest | Active | $O(\text{poly}(\lambda, n))$ | Yes |
| JS23 [6] | Honest | Passive | $O(\text{poly}(\lambda, n))$ | Yes |
| DDE+23 [7] | Honest | Active | $O(\text{poly}(\lambda))$ | No |
| CCS19 [8] | Honest | Passive | $O(\text{poly}(\lambda, n))$ | Yes |
| TLX+21 [9] | Honest | Passive | $O(\text{poly}(\lambda, n))$ | Yes |
| Ours | Dishonest | Active | $O(\text{poly}(\lambda))$ | Yes |

Table: Comparison of FHE-based MPC protocols. The parameter $\lambda$ is the computational security bit, $\kappa$ is the statistical security bit, and $d_{\lambda,n}$ is the dimension of LWE sample

# Circuit-Private MPC in Dishonest Majority Setting

- Another Goal: How to build circuit-private MPC under dishonest majority and eavesdropping environment?



# Circuit-Private MPC in Dishonest Majority Setting

- Anothor Goal: How to build circuit-private MPC under dishonest majority and eavesdropping environment?
- While the universal circuit approach might help [14],
  - the overall communication size nevertheless leaks information about the circuit.

# Circuit-Private MPC in Dishonest Majority Setting

- Another Goal: How to build circuit-private MPC under dishonest majority and eavesdropping environment?
- While the universal circuit approach might help [14],
  - the overall communication size nevertheless leaks information about the circuit.
- Problem in SPDZ, BMR:
  - Communication increases proportional to circuit size
  - Circuit structure is indirectly leaked through total traffic

# Circuit-Private MPC in Dishonest Majority Setting

- Another Goal: How to build circuit-private MPC under dishonest majority and eavesdropping environment?
- While the universal circuit approach might help [14],
  - the overall communication size nevertheless leaks information about the circuit.
- Problem in SPDZ, BMR:
  - Communication increases proportional to circuit size
  - Circuit structure is indirectly leaked through total traffic
- **FHE-based MPC is ready to be proven**
  - All evaluation is local
  - No circuit-dependent communication in the online phase
  - Circuit privacy is provably achieved

*Our protocol is the first to achieve circuit-private MPC with active security under dishonest majority.*

# Backgrounds

# LWE Sample and Encryption Review

**LWE Sample:**

- Fix modulus $q$, dimension $n$, and error distribution $\chi$ (e.g., discrete Gaussian)
- Let $s \in \mathbb{Z}_q^n$ be a secret vector
- Sample $A \in \mathbb{Z}_q^{n \times m}$ uniformly at random and noise $e \leftarrow \chi^m$
- Output the LWE sample: $(A, b = sA + e \mod q)$

# LWE Sample and Encryption Review

**LWE Sample:**

- Fix modulus $q$, dimension $n$, and error distribution $\chi$ (e.g., discrete Gaussian)
- Let $s \in \mathbb{Z}_q^n$ be a secret vector
- Sample $A \in \mathbb{Z}_q^{n \times m}$ uniformly at random and noise $e \leftarrow \chi^m$
- Output the LWE sample: $(A, b = sA + e \mod q)$

## Definition (Regev's Encryption [10])

Let $M = [A^t | b^t] \in \mathbb{Z}_q^{m \times (n+1)}$ be public, and $s \in \mathbb{Z}_q^n$ be a secret key.
To encrypt $x \in \{0, 1\}$:

- Sample $r \in \{0, 1\}^m$, $e_1 \leftarrow \chi^n$ $e_2 \leftarrow \chi$ and compute $c = (c_1, c_2)$:

$$c_1 = rA^t + e_1 \in \mathbb{Z}_q^n, \quad c_2 = rb^t + \left\lfloor \frac{q}{2} \right\rfloor m + e_2 = rA^t s^t + re^t + e_2 + \left\lfloor \frac{q}{2} \right\rfloor$$

# Addition and Multiplication in LWE ciphertext

**Addition (Linear):**

- Ciphertexts can be added.
- Example:
  $\mathrm{Dec}\big(\mathrm{Enc}(m_1) + \mathrm{Enc}(m_2)\big) = \big((A_1, b_2) + (A_2, b_2)\big) = m_1 + m_2$ (noise grows linearly).

# Addition and Multiplication in LWE ciphertext

**Addition (Linear):**

- Ciphertexts can be added.
- Example: $\mathrm{Dec}\big(\mathrm{Enc}(m_1) + \mathrm{Enc}(m_2)\big) = \big((A_1, b_2) + (A_2, b_2)\big) = m_1 + m_2$ (noise grows linearly).

**Multiplication (Nontrivial):**

- Multiplying ciphertexts is not straightforward and limited up to at most $L$ times.

# Bootstrapping: From Leveled HE to FHE

LHE ciphertext with message m after performing $L$ NAND gates

$c_1[m]$ →

$c[sk]$ →

**Bootstrapping ↔ Performing Decryption circuit $\mathcal{C}(c_1[m], sk) = m$ on the ciphertext domain with $|\mathcal{C}| = L' < L$**

Evaluation key
:= encrypting secret key $sk$

↓

LHE ciphertext with message m after performing $L' < L$ NAND gates

$$c_2[m] \leftarrow \mathrm{Eval}(\mathcal{C}, c[sk], c_1[m])$$

* $c_1[m]$ is no longer ciphertext, but plaintext values in terms of the circuit $\mathcal{C}$

# SPDZ Secret Sharing:

**(*n*-*out*-*of*-*n*)Linear Secret Sharing:** We denote the share of $m$ as

$$[m] = ([m]_1, [m]_2, \ldots, [m]_n) \in \mathbb{Z}_q^n$$

such that

$$m = \sum_{i=1}^{n} [m]_i \mod p$$

# SPDZ Secret Sharing:

**(*n*-*out*-*of*-*n*)Linear Secret Sharing:** We denote the share of $m$ as

$$[m] = ([m]_1, [m]_2, \ldots, [m]_n) \in \mathbb{Z}_q^n$$

such that

$$m = \sum_{i=1}^{n} [m]_i \mod p$$

## Definition (Ideal functionality $\mathcal{F}_{\text{SPDZ}}$)

- $\mathcal{F}^p_{\text{SPDZ}}$.Input($\cdot$): from input $x$, output $[x]_i$ to the $P_i$.
- $\mathcal{F}^p_{\text{SPDZ}}$.Rand(): output $[u]_i$ for some uniformly random $u \in \mathbb{Z}_p$.
- $\mathcal{F}^p_{\text{SPDZ}}$.RandBit(): output $[r]_i$ for some random bit $r \in \mathbb{Z}_p$.
- $\mathcal{F}^p_{\text{SPDZ}}$.MUL($[x], [y]$): output $[xy]_i$ to the $P_i$.

# Arithmetic on Secret Shares: SPDZ-style MPC

**Linear operations: Local and efficient**

- $[x + y]$ can be computed locally:

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

# Arithmetic on Secret Shares: SPDZ-style MPC

**Linear operations: Local and efficient**

- $[x + y]$ can be computed locally:

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

0.5em **Multiplication: Requires communication + preprocessing**

- $[x \cdot y]$ is computed using a **Beaver triple**:

$$\text{Preprocess: } \langle a \rangle,\ \langle b \rangle,\ \langle ab \rangle$$

- Parties use $x - a$ and $y - b$ to compute $xy$ via interactive protocol.
- Hence, multiplicative depth of the circuit $\mathcal{C}$ determines the number of communication round, and the number of multiplication in $\mathcal{C}$ determines the total communication.

# Threshold FHE: Idea and Challenges

- Remind LWE encryption:

$$\text{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum a_i s_i + e)$$

# Threshold FHE: Idea and Challenges

- Remind LWE encryption:

$$\mathrm{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum a_i s_i + e)$$

- If parties hold $[s_i]$ and $[e]$ as secret sharing, and $\vec{a}$ is a common reference string (CRS):

$$[b] = \sum a_i [s_i] + [e]$$

# Threshold FHE: Idea and Challenges

- Remind LWE encryption:

$$\mathrm{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum a_i s_i + e)$$

- If parties hold $[s_i]$ and $[e]$ as secret sharing, and $\vec{a}$ is a common reference string (CRS):

$$[b] = \sum a_i [s_i] + [e]$$

- Opening $[b]$ gives Regev encryption in threshold form

# Threshold FHE: Idea and Challenges

- Remind LWE encryption:

$$\mathrm{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum a_i s_i + e)$$

- If parties hold $[s_i]$ and $[e]$ as secret sharing, and $\vec{a}$ is a common reference string (CRS):

$$[b] = \sum a_i [s_i] + [e]$$

- Opening $[b]$ gives Regev encryption in threshold form

**Challenges:**

# Threshold FHE: Idea and Challenges

- Remind LWE encryption:

$$\mathrm{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum a_i s_i + e)$$

- If parties hold $[s_i]$ and $[e]$ as secret sharing, and $\vec{a}$ is a common reference string (CRS):

$$[b] = \sum a_i [s_i] + [e]$$

- Opening $[b]$ gives Regev encryption in threshold form

**Challenges:**

- the modulus $Q$ often not a prime

# Threshold FHE: Idea and Challenges

- Remind LWE encryption:

$$\mathrm{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum a_i s_i + e)$$

- If parties hold $[s_i]$ and $[e]$ as secret sharing, and $\vec{a}$ is a common reference string (CRS):

$$[b] = \sum a_i [s_i] + [e]$$

- Opening $[b]$ gives Regev encryption in threshold form

**Challenges:**

- the modulus $Q$ often not a prime
- **KDM security:** Must support

$$\mathrm{Enc}_{\vec{s}_1}(\vec{s}_2), \quad \mathrm{Enc}_{\vec{s}_1}(\vec{s}_1)$$

while $\mathrm{Enc}_{\vec{s}_1}$ and $\mathrm{Enc}_{\vec{s}_2}$ could use difference modulus, respectively.

# Random Bit Sampling over Composite-Modulus Secret Sharing

# Sampling on the secret sharing with Composite Modulus

- **Idea:** Sample secret key and error over a composite modulus

$$Q = p_1 \cdot p_2 \cdots p_\ell$$

where each $p_i$ is pairwise co-prime and $\mathcal{F}_{\mathsf{SPDZ}}^{p_i}$ are available.

# Sampling on the secret sharing with Composite Modulus

- **Idea:** Sample secret key and error over a composite modulus

$$Q = p_1 \cdot p_2 \cdots p_\ell$$

  where each $p_i$ is pairwise co-prime and $\mathcal{F}_{\text{SPDZ}}^{p_i}$ are available.
- However, directly invoking all $\mathcal{F}_{\text{SPDZ}}^{p_i}$.RandBit() is not a solution. This is because the elements $\{0, 1\} \in \mathbb{Z}_Q$ must correspond to the tuples $(0, \ldots, 0)$ and $(1, \ldots, 1)$ across all moduli.

# Sampling on the secret sharing with Composite Modulus

- **Idea:** Sample secret key and error over a composite modulus

$$Q = p_1 \cdot p_2 \cdots p_\ell$$

  where each $p_i$ is pairwise co-prime and $\mathcal{F}_{\text{SPDZ}}^{p_i}$ are available.
- However, directly invoking all $\mathcal{F}_{\text{SPDZ}}^{p_i}$.RandBit() is not a solution. This is because the elements $\{0, 1\} \in \mathbb{Z}_Q$ must correspond to the tuples $(0, \ldots, 0)$ and $(1, \ldots, 1)$ across all moduli.
- **Goal:** Construct arbitrary discrete distributions (e.g., Discrete Gaussian) from uniform bits over $Q$ **with active security**

# Sampling on the secret sharing with Composite Modulus

## Lemma (Inverse transform sampling)

*Let $\mathcal{D}$ be any discrete distribution with support size* $\mathrm{poly}(n)$ *and* $u_1, \ldots, u_\kappa$ *be* $\kappa$ *uniformly random samples. Then there exists a Boolean/arithmetic circuit such that:*

- ***Depth:*** *at most* $3\lceil \log \kappa \rceil + 2$
- ***Multiplications:*** *at most* $6 \cdot \mathrm{poly}(n) \cdot \kappa$
- ***Output:*** *a sample from distribution statistically close to* $\mathcal{D}$
- ***advantage of adversary:*** *statistical distance* $\leq 2^{-\kappa}$

# Sampling on the secret sharing with Composite Modulus

## Lemma (Inverse transform sampling)

*Let $\mathcal{D}$ be any discrete distribution with support size* $\mathrm{poly}(n)$ *and* $u_1, \ldots, u_\kappa$ *be* $\kappa$ *uniformly random samples. Then there exists a Boolean/arithmetic circuit such that:*

- ***Depth:*** *at most* $3\lceil \log \kappa \rceil + 2$
- ***Multiplications:*** *at most* $6 \cdot \mathrm{poly}(n) \cdot \kappa$
- ***Output:*** *a sample from distribution statistically close to* $\mathcal{D}$
- ***advantage of adversary:*** *statistical distance* $\leq 2^{-\kappa}$

- Actively secure **one-bit sampler** over $Q \Rightarrow$
- Actively secure sampler for **any** $\mathrm{poly}(n)$**-bounded distribution**, including discrete Gaussians with polynomial variance

# Core idea of samping one bit on the composite modulus [12]

- First, perform one-bit sampling of $[m]^p$ over an **auxiliary prime modulus** $p$ and obtaining $k$ such that $m = \sum_{i=1}^{n} [m]_i^p - kp$.

# Core idea of samping one bit on the composite modulus [12]

- First, perform one-bit sampling of $[m]^p$ over an **auxiliary prime modulus** $p$ and obtaining $k$ such that $m = \sum_{i=1}^{n} [m]_i^p - kp$.
- Then, **replicate the sampled share** into corresponding shares

$$[m]^{p_1},\ [m]^{p_2},\ \ldots,\ [m]^{p_\ell}$$

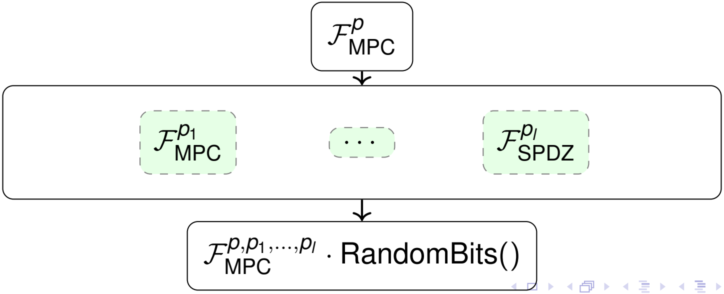
over the other prime moduli via $m = (\sum_{i=1}^{n} [m]_i^p - kp) \bmod p_i$.

# Core idea of samping one bit on the composite modulus [12]

- First, perform one-bit sampling of $[m]^p$ over an **auxiliary prime modulus** $p$ and obtaining $k$ such that $m = \sum_{i=1}^{n} [m]_i^p - kp$.
- Then, **replicate the sampled share** into corresponding shares

$$[m]^{p_1},\ [m]^{p_2},\ \ldots,\ [m]^{p_\ell}$$

over the other prime moduli via $m = (\sum_{i=1}^{n} [m]_i^p - kp) \bmod p_i$.



# Open Only High Bits to obtaining $k$ [12]

**$\Delta$-split (high bit only).** Let $p$ be prime and each party hold a share $x_i \in \{0, \ldots, p-1\}$. Fix $\Delta = \lceil p/n \rceil$ and write

$$[m]_i = \ell_i + \Delta h_i, \qquad 0 \leq \ell_i < \Delta.$$

Only the *high bits* $h_i$ are opened; the lows $\ell_i$ remain secret.

# Open Only High Bits to obtaining $k$ [12]

**Δ-split (high bit only).** Let $p$ be prime and each party hold a share $x_i \in \{0, \ldots, p-1\}$. Fix $\Delta = \lceil p/n \rceil$ and write

$$[m]_i = \ell_i + \Delta h_i, \qquad 0 \le \ell_i < \Delta.$$

Only the *high bits* $h_i$ are opened; the lows $\ell_i$ remain secret.

**Lemma [Correctness].** Let $[m]_1, \ldots, [m]_n$ be uniformly chosen subject to

$$[m]_1 + \cdots + [m]_n \in \{k \cdot p,\ k \cdot p + 1\}$$

for some integer $k$. Set $\Delta = \lceil p/n \rceil$ and decompose $x_i = \ell_i + \Delta h_i$ with $0 \le \ell_i < \Delta$. Then, with probability at least $1 - \frac{3}{p}$,

$$\boxed{k = \left\lfloor \frac{\Delta}{p} \sum_{i=1}^{n} h_i \right\rfloor}$$

# Open Only High Bits to obtaining $k$ [12]

**$\Delta$-split (high bit only).** Let $p$ be prime and each party hold a share $x_i \in \{0, \dots, p-1\}$. Fix $\Delta = \lceil p/n \rceil$ and write

$$[m]_i = \ell_i + \Delta h_i, \qquad 0 \le \ell_i < \Delta.$$

Only the *high bits* $h_i$ are opened; the lows $\ell_i$ remain secret.

**Lemma [Correctness].** Let $[m]_1, \dots, [m]_n$ be uniformly chosen subject to

$$[m]_1 + \cdots + [m]_n \in \{k \cdot p,\ k \cdot p + 1\}$$

for some integer $k$. Set $\Delta = \lceil p/n \rceil$ and decompose $x_i = \ell_i + \Delta h_i$ with $0 \le \ell_i < \Delta$. Then, with probability at least $1 - \frac{3}{p}$,

$$\boxed{k = \left\lfloor \frac{\Delta}{p} \sum_{i=1}^{n} h_i \right\rfloor}$$

**Lemma [Zero knowledge, Informal] The protocol has perfectly hiding property (no informational leakage).**

# Verification protocol [12]

**Verifying $t$ uniformly random bits consuming $\mathsf{sec}$ random bits**

1. Sample public $t + \mathsf{sec}$ values $r_{i,j}$ uniformly at random from $\{0, 2^{\mathsf{sec}} - 1\}$.

# Verification protocol [12]

**Verifying $t$ uniformly random bits consuming $\mathsf{sec}$ random bits**

1. Sample public $t + \mathsf{sec}$ values $r_{i,j}$ uniformly at random from $\{0, 2^{\mathsf{sec}} - 1\}$.
2. For each $v = 0, \ldots, t - 1$, compute linear shares

$$[S_v]^{p_v} = \sum_{i=1}^{t+\mathsf{sec}} r_{i,j} \cdot [m_i]^{p_v}.$$

# Verification protocol [12]

**Verifying $t$ uniformly random bits consuming sec random bits**

1. Sample public $t + \text{sec}$ values $r_{i,j}$ uniformly at random from $\{0, 2^{\text{sec}} - 1\}$.
2. For each $v = 0, \ldots, t-1$, compute linear shares

$$[S_v]^{p_v} = \sum_{i=1}^{t+\text{sec}} r_{i,j} \cdot [m_i]^{p_v}.$$

3. Open $[S_0]^{p_0}$ (auxiliary prime $p_0$) and each $S_v$.

# Verification protocol [12]

**Verifying $t$ uniformly random bits consuming $\mathsf{sec}$ random bits**

1. Sample public $t + \mathsf{sec}$ values $r_{i,j}$ uniformly at random from $\{0, 2^{\mathsf{sec}} - 1\}$.
2. For each $v = 0, \ldots, t-1$, compute linear shares

$$[S_v]^{p_v} = \sum_{i=1}^{t+\mathsf{sec}} r_{i,j} \cdot [m_i]^{p_v}.$$

3. Open $[S_0]^{p_0}$ (auxiliary prime $p_0$) and each $S_v$.
4. Abort if $S_0 \neq S_v \mod p_v$ for some $v$.

# Verification protocol [12]

**Verifying $t$ uniformly random bits consuming sec random bits**

1. Sample public $t + \text{sec}$ values $r_{i,j}$ uniformly at random from $\{0, 2^{\text{sec}} - 1\}$.
2. For each $v = 0, \ldots, t-1$, compute linear shares

$$[S_v]^{p_v} = \sum_{i=1}^{t+\text{sec}} r_{i,j} \cdot [m_i]^{p_v}.$$

3. Open $[S_0]^{p_0}$ (auxiliary prime $p_0$) and each $S_v$.
4. Abort if $S_0 \neq S_v \mod p_v$ for some $v$.
5. Otherwise output $[m_i]^{p_v}$.

# Verification protocol [12]

**Verifying $t$ uniformly random bits consuming sec random bits**

1. Sample public $t + \text{sec}$ values $r_{i,j}$ uniformly at random from $\{0, 2^{\text{sec}} - 1\}$.
2. For each $v = 0, \ldots, t-1$, compute linear shares
$$[S_v]^{p_v} = \sum_{i=1}^{t+\text{sec}} r_{i,j} \cdot [m_i]^{p_v}.$$
3. Open $[S_0]^{p_0}$ (auxiliary prime $p_0$) and each $S_v$.
4. Abort if $S_0 \neq S_v \mod p_v$ for some $v$.
5. Otherwise output $[m_i]^{p_v}$.

**Note.** The $r_{i,j}$ randomizers (size $2^{\text{sec}}$) guarantee hiding , while cross-checking ensures consistency across all moduli.

# Verification protocol [12]

**Verifying $t$ uniformly random bits consuming sec random bits**

1. Sample public $t + \text{sec}$ values $r_{i,j}$ uniformly at random from $\{0, 2^{\text{sec}} - 1\}$.
2. For each $v = 0, \ldots, t-1$, compute linear shares
$$[S_v]^{p_v} = \sum_{i=1}^{t+\text{sec}} r_{i,j} \cdot [m_i]^{p_v}.$$
3. Open $[S_0]^{p_0}$ (auxiliary prime $p_0$) and each $S_v$.
4. Abort if $S_0 \neq S_v \mod p_v$ for some $v$.
5. Otherwise output $[m_i]^{p_v}$.

**Note.** The $r_{i,j}$ randomizers (size $2^{\text{sec}}$) guarantee hiding , while cross-checking ensures consistency across all moduli.

**Drawback.** The opened value $[S_0]^{p_0}$ must satisfy $0 < S_0 < p_0$, which requires the auxiliary prime $p_0$ to be much larger than $2^{\text{sec}}$. Hence, a very large modulus $p_0$ is needed.

# [Our works] Opening High Bits: Counterexample & Lemma (in One Slide)

**Toy counterexample (not perfectly hiding).**

# [Our works] Opening High Bits: Counterexample & Lemma (in One Slide)

**Toy counterexample (not perfectly hiding).**

Let $p_0 = 7$, $\Delta = \lceil p_0/2 \rceil = 4$, and an adversary fixes its share $[m]_2^{(7)} = 1$.

# [Our works] Opening High Bits: Counterexample & Lemma (in One Slide)

**Toy counterexample (not perfectly hiding).**

Let $p_0 = 7$, $\Delta = \lceil p_0/2 \rceil = 4$, and an adversary fixes its share $[m]_2^{(7)} = 1$. Then $h_2 = 0$ and reconstruction $m \equiv [m]_1^{(7)} + [m]_2^{(7)} \pmod 7$ with $m \in \{0, 1\}$ forces $[m]_1^{(7)} \in \{0, 6\}$.

# [Our works] Opening High Bits: Counterexample & Lemma (in One Slide)

**Toy counterexample (not perfectly hiding).**

Let $p_0 = 7$, $\Delta = \lceil p_0/2 \rceil = 4$, and an adversary fixes its share $[m]_2^{(7)} = 1$. Then $h_2 = 0$ and reconstruction $m \equiv [m]_1^{(7)} + [m]_2^{(7)} \pmod 7$ with $m \in \{0, 1\}$ forces $[m]_1^{(7)} \in \{0, 6\}$. Opening only $h_1 = \lfloor [m]_1^{(7)} / \Delta \rfloor$ reveals $m$ ($h_1 = 0 \Rightarrow m = 1$, $h_1 = 1 \Rightarrow m = 0$). Hence perfect hiding can fail.

# [Our works] Opening High Bits: Counterexample & Lemma (in One Slide)

**Toy counterexample (not perfectly hiding).**

Let $p_0 = 7$, $\Delta = \lceil p_0/2 \rceil = 4$, and an adversary fixes its share $[m]_2^{(7)} = 1$. Then $h_2 = 0$ and reconstruction $m \equiv [m]_1^{(7)} + [m]_2^{(7)} \pmod 7$ with $m \in \{0, 1\}$ forces $[m]_1^{(7)} \in \{0, 6\}$. Opening only $h_1 = \lfloor [m]_1^{(7)}/\Delta \rfloor$ reveals $m$ ($h_1 = 0 \Rightarrow m = 1$, $h_1 = 1 \Rightarrow m = 0$). Hence perfect hiding can fail.

**Lemma (no leakage except boundary cases).** Write an honest share as $[m]_i = \ell_i + \Delta h_i$ with $0 \le \ell_i < \Delta$, $h_i \in \{0, 1\}$. If $[m]_i$ is *not* in any of the following boundary cases:

$$\text{(i) } \ell_i = 0, \qquad \text{(ii) } \ell_i = \Delta - 1, \qquad \text{(iii) } [m]_i = p - 1,$$

then revealing $h_i$ leaks no information about $m$ (i.e., $h_i \perp m$).

# [Our works] Opening High Bits: Counterexample & Lemma (in One Slide)

**Toy counterexample (not perfectly hiding).**

Let $p_0 = 7$, $\Delta = \lceil p_0/2 \rceil = 4$, and an adversary fixes its share $[m]_2^{(7)} = 1$. Then $h_2 = 0$ and reconstruction $m \equiv [m]_1^{(7)} + [m]_2^{(7)} \pmod 7$ with $m \in \{0, 1\}$ forces $[m]_1^{(7)} \in \{0, 6\}$. Opening only $h_1 = \lfloor [m]_1^{(7)}/\Delta \rfloor$ reveals $m$ ($h_1 = 0 \Rightarrow m = 1$, $h_1 = 1 \Rightarrow m = 0$). Hence perfect hiding can fail.

**Lemma (no leakage except boundary cases).** Write an honest share as $[m]_i = \ell_i + \Delta h_i$ with $0 \le \ell_i < \Delta$, $h_i \in \{0, 1\}$. If $[m]_i$ is *not* in any of the following boundary cases:

$$\text{(i) } \ell_i = 0, \qquad \text{(ii) } \ell_i = \Delta - 1, \qquad \text{(iii) } [m]_i = p - 1,$$

then revealing $h_i$ leaks no information about $m$ (i.e., $h_i \perp m$).

**Note (rejection sampling restores perfect hiding property).** Whenever a bad event is detected, discard that sample (rejection) and resample; the resulting message-bit shares $[m]$ are again uniformly random and thus perfectly hiding.

# [Our contribution] Protocol Fix: Boundary-Share Filter Before Opening $h_i$

**Idea.** Honest parties locally screen out boundary cases before any $h_i$ is opened.

# [Our contribution] Protocol Fix: Boundary-Share Filter Before Opening $h_i$

**Idea.** Honest parties locally screen out boundary cases before any $h_i$ is opened.

**Distribute $h_i$ phase (before high-bit opening).**

1. Each honest $P_i$ computes $\ell_i = [m]_i \bmod \Delta$ and checks:

$$\ell_i \in \{0, \Delta - 1\} \text{ or } [m]_i = p - 1.$$

# [Our contribution] Protocol Fix: Boundary-Share Filter Before Opening $h_i$

**Idea.** Honest parties locally screen out boundary cases before any $h_i$ is opened.

**Distribute $h_i$ phase (before high-bit opening).**

1. Each honest $P_i$ computes $\ell_i = [m]_i \bmod \Delta$ and checks:

$$\ell_i \in \{0, \Delta - 1\} \quad \text{or} \quad [m]_i = p - 1.$$

2. If a boundary case holds, $P_i$ broadcasts its *share index* to a public set $B$.

# [Our contribution] Protocol Fix: Boundary-Share Filter Before Opening $h_i$

**Idea.** Honest parties locally screen out boundary cases before any $h_i$ is opened.

**Distribute $h_i$ phase (before high-bit opening).**

1. Each honest $P_i$ computes $\ell_i = [m]_i \bmod \Delta$ and checks:

$$\ell_i \in \{0, \Delta - 1\} \text{ or } [m]_i = p - 1.$$

2. If a boundary case holds, $P_i$ broadcasts its *share index* to a public set $B$.
3. If $B \neq \emptyset$, resample/refresh those indices (or exclude them for this round) and restart the high-bit step.

# [Our contribution] Protocol Fix: Boundary-Share Filter Before Opening $h_i$

**Idea.** Honest parties locally screen out boundary cases before any $h_i$ is opened.

**Distribute $h_i$ phase (before high-bit opening).**

1. Each honest $P_i$ computes $\ell_i = [m]_i \bmod \Delta$ and checks:
$$\ell_i \in \{0, \Delta - 1\} \text{ or } [m]_i = p - 1.$$
2. If a boundary case holds, $P_i$ broadcasts its *share index* to a public set $B$.
3. If $B \neq \emptyset$, resample/refresh those indices (or exclude them for this round) and restart the high-bit step.
4. Otherwise (i.e., $B = \emptyset$) proceed to publicly open the $h_i$'s.

# [Our contribution] Protocol Fix: Boundary-Share Filter Before Opening $h_i$

**Idea.** Honest parties locally screen out boundary cases before any $h_i$ is opened.

**Distribute $h_i$ phase (before high-bit opening).**

1. Each honest $P_i$ computes $\ell_i = [m]_i \bmod \Delta$ and checks:
$$\ell_i \in \{0, \Delta - 1\} \text{ or } [m]_i = p - 1.$$
2. If a boundary case holds, $P_i$ broadcasts its *share index* to a public set $B$.
3. If $B \neq \emptyset$, resample/refresh those indices (or exclude them for this round) and restart the high-bit step.
4. Otherwise (i.e., $B = \emptyset$) proceed to publicly open the $h_i$'s.

**Note (probability & restart policy).** If all parties' shares are uniformly random, the probability that *some* honest share hits a boundary is at most $3np^{-1}$. Hence one can monitor the number of restarts and *abort*.

# [Our contribution] Verification with One-Bit $r$

- In the previous verification protocol, We reduce $r$ down to a *single bit*, leveraging the *Random Smudging Lemma*:
- By applying rejection sampling whenever a bad event is detected, we ensure the message shares remain uniformly random and perfectly hiding.
- This modification allows the auxiliary modulus to be reduced to only **32 bits**, while preserving security.

# [Our contribution] Verification with One-Bit $r$

- In the previous verification protocol, We reduce $r$ down to a *single bit*, leveraging the *Random Smudging Lemma*:
- By applying rejection sampling whenever a bad event is detected, we ensure the message shares remain uniformly random and perfectly hiding.
- This modification allows the auxiliary modulus to be reduced to only **32 bits**, while preserving security.

## Lemma (Random Smudging Lemma)

*Let $X \xleftarrow{\$} U_{2^\kappa}$ and $Y \xleftarrow{\$} D_B$ be independent. Let $\mathcal{E}$ be the event that $B-1 \leq X+Y < 2^\kappa$, and let $\mathcal{E}^c$ be its complement. Then:*

(i) $X+Y$ *and* $Y$ *are independent conditioned on* $\mathcal{E}$.

(ii) $\Pr[\mathcal{E}^c] \leq (B-1)\,2^{-\kappa}$.

(iii) $\Delta\big((X+Y),\ Y\big) \leq (B-1)\,2^{-\kappa}$.

# Improving Random Bit Sampling Overview

**Rotaru et al. [12]**

- Assumes **no leakage** during the sharing
- Requires **uniform randomness** from corrupted parties
- Requires $\lambda$**-bit auxiliary primes** (e.g., 128 bits)

# Improving Random Bit Sampling Overview

**Rotaru et al. [12]**

- Assumes **no leakage** during the sharing
- Requires **uniform randomness** from corrupted parties
- Requires $\lambda$**-bit auxiliary primes** (e.g., 128 bits)

**Our Improvements**

- Provides **full leakage analysis** of all possible cases
- Remains secure even when **corrupted parties choose shares arbitrarily**
- Requires using primes $p \approx 32$ bits via rejection procedure

# secret-shared FHE, SSFHE

# Informal Construction of SSFHE

**Step-by-step Construction:**

- Combine all primes $p_1, \ldots, p_l$ used in the given FHE into a single composite modulus $Q = p_1 \cdots p_l$.

# Informal Construction of SSFHE

**Step-by-step Construction:**

- Combine all primes $p_1, \ldots, p_l$ used in the given FHE into a single composite modulus $Q = p_1 \cdots p_l$.
- Use a multiplicative secret sharing scheme (e.g., SPDZ) to generate secret shares:
  - $[\vec{s}]^Q$: secret sharing of the secret key $\vec{s}$
  - $[\vec{e}]^Q$: secret sharing of the error vector $\vec{e}$

# Informal Construction of SSFHE

**Step-by-step Construction:**

- Combine all primes $p_1, \ldots, p_l$ used in the given FHE into a single composite modulus $Q = p_1 \cdots p_l$.
- Use a multiplicative secret sharing scheme (e.g., SPDZ) to generate secret shares:
  - $[\vec{s}]^Q$: secret sharing of the secret key $\vec{s}$
  - $[\vec{e}]^Q$: secret sharing of the error vector $\vec{e}$
  - $[\text{pk}]^Q$ and $[\text{ev}]^Q$ are deterministically generated from $[\vec{s}]$ and $[\vec{e}]$ so that it can computed via multiplicative secret sharing scheme

# Informal Construction of SSFHE

**Step-by-step Construction:**

- Combine all primes $p_1, \ldots, p_l$ used in the given FHE into a single composite modulus $Q = p_1 \cdots p_l$.
- Use a multiplicative secret sharing scheme (e.g., SPDZ) to generate secret shares:
  - $[\vec{s}]^Q$: secret sharing of the secret key $\vec{s}$
  - $[\vec{e}]^Q$: secret sharing of the error vector $\vec{e}$
  - $[\text{pk}]^Q$ and $[\text{ev}]^Q$ are deterministically generated from $[\vec{s}]$ and $[\vec{e}]$ so that it can computed via multiplicative secret sharing scheme
- Open (reveal) [pk] and [ev] to all parties.

**$\epsilon$-Correctness and $\delta$-IND-CPA Security Guarantee:**

# Informal Construction of SSFHE

**Step-by-step Construction:**

- Combine all primes $p_1, \ldots, p_l$ used in the given FHE into a single composite modulus $Q = p_1 \cdots p_l$.
- Use a multiplicative secret sharing scheme (e.g., SPDZ) to generate secret shares:
  - $[\vec{s}]^Q$: secret sharing of the secret key $\vec{s}$
  - $[\vec{e}]^Q$: secret sharing of the error vector $\vec{e}$
  - $[\text{pk}]^Q$ and $[\text{ev}]^Q$ are deterministically generated from $[\vec{s}]$ and $[\vec{e}]$ so that it can computed via multiplicative secret sharing scheme
- Open (reveal) [pk] and [ev] to all parties.

**$\epsilon$-Correctness and $\delta$-IND-CPA Security Guarantee:**

- Let FHE be $(\delta_F, \epsilon_F)$-secure
- Let the LSSS-based MPC have $\kappa$-soundness
- Then the resulting SSFHE satisfies:

$$\delta_S \leftarrow \delta_F, \quad \epsilon_S \leftarrow 2^{-O(\kappa)} + \epsilon_F \cdot \text{poly}(\lambda)$$

# Communication Cost of Gate Bootstrapping

**Key Insight:**

- Recall the structure of FHE (or LWE) encryption:

$$\mathrm{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum_{i=1}^{d} a_i s_i + e)$$

- If we can generate secret shares $[\vec{s}]$ and $[\vec{e}]$, then encryption can be computed locally.

# Communication Cost of Gate Bootstrapping

**Key Insight:**

- Recall the structure of FHE (or LWE) encryption:

$$\mathrm{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum_{i=1}^{d} a_i s_i + e)$$

- If we can generate secret shares $[\vec{s}]$ and $[\vec{e}]$, then encryption can be computed locally.

**Key Observation:**

- The circuit for generating pk and GINX-type ev consists of many parallel calls to:
  - random bit samplers or inverse transform samplers

# Communication Cost of Gate Bootstrapping

**Key Insight:**

- Recall the structure of FHE (or LWE) encryption:

$$\text{Enc}_{\vec{s}}(m) = (\vec{a},\ b = \sum_{i=1}^{d} a_i s_i + e)$$

- If we can generate secret shares $[\vec{s}]$ and $[\vec{e}]$, then encryption can be computed locally.

**Key Observation:**

- The circuit for generating pk and GINX-type ev consists of many parallel calls to:
  - random bit samplers or inverse transform samplers

**Communication Complexity:**

- For LWE dimension $d$:
  - random bit sampling: $O(1)$ rounds, $O(d^2)$ multiplications
  - Inverse transform: $O(\log \kappa)$ rounds, $O(d^2 \kappa)$ multiplications

# Circuit-private MPC

# Circuit privacy: Why we use SSFHE

## Previous MPC ideal functionality: $\mathcal{F}_{\text{ABB}}$

**Ideal functionality of $\mathcal{F}_{\text{ABB}}$**

**Initializ:** On input (init, $p^k$) for all parties, the functionality activates and store the modulus $p$.

**Input:** On input (input, $P_i$, varid, $x$) from $P_i$ and (input, $P_i$, varid, ?) from all other parties, with varid a fresh identifier, the functionality stores (varid, $x$)

**Add:** On command (add, varid$_1$, varid$_2$, varid$_3$) from all parties (if varid$_1$ and varid$_2$ are present in memory and varid$_3$ is not), the functionality retrieves (varid$_1$, $x$), (varid$_2$, $y$) and store (varid$_3$, $x + y \mod p$)

**Multiply:** On command (multiply, varid$_1$, varid$_2$, varid$_3$) from all parties (if varid$_1$ and varid$_2$ are present in memory and varid$_3$ is not), the functionality retrieves (varid$_1$, $x$), (varid$_2$, $y$) and store (varid$_3$, $xy \mod p$)

**Output:** On input (output, varid from all honest parties parties (if varid is present in memory), the functionality retrieves (varid, $x$), and outputs it to the environment. If the adversary inputs OK then $x$ is output to all parties. Otherwise $\perp$ is output to all parties.

Figure: The common ideal functionality for arithmetic black-box model.

# Circuit privacy: Why we use SSFHE

## Current circuit-private MPC ideal functionaliy: $\mathcal{F}_{\text{CPMPC}}$

**Ideal functionality of $\mathcal{F}_{\text{CPMPC}}$**

**Init:** On input (**init**, $t$) from all parties, store the threshold $t$. Initialize the register State to 0.

**CircuitInit:** On input (**CircInit**, $i, \mathcal{C}, I$) from all parties $P_i$, if State is 0, store the circuit $\mathcal{C}$ and leak the number of inputs $I$ and the input-to-party mapping to the adversary $\mathcal{A}$. Set State to 1.

**Input:** On input (**input**, $P_i$, varid, $x$) from all parties $P_i$ and (input, $P_i$, varid, ?) from all other parties in $\mathcal{P} \backslash P_i$, store (varid, $x$) if varid is a fresh identifier. Set State to 2.

**Output:** On input (**output**, $i$, varid) from all parties $P_i$, if varid exists in memory, TTP retrieves (varid, $x$) and outputs it to the adversary $\mathcal{A}$. If $\mathcal{A}$ inputs OK, $x$ is output to all parties, and State is reset to 0. If $\mathcal{A}$ inputs $\perp$, then $\perp$ is output to all parties, and the protocol terminates.

Figure: The ideal functionality for circuit-private MPC.

# Circuit privacy: Why we use SSFHE

## Current circuit-private MPC ideal functionaliy: $\mathcal{F}_{\text{CPMPC}}$

**Ideal functionality of $\mathcal{F}_{\text{CPMPC}}$**

**Init:** On input (**init**, $t$) from all parties, store the threshold $t$. Initialize the register State to 0.

**CircuitInit:** On input (**CircInit**, $i, \mathcal{C}, l$) from all parties $P_i$, if State is 0, store the circuit $\mathcal{C}$ and leak the number of inputs $l$ and the input-to-party mapping to the adversary $\mathcal{A}$. Set State to 1.

**Input:** On input (**input**, $P_i$, varid, $x$) from all parties $P_i$ and (input, $P_i$, varid, ?) from all other parties in $\mathcal{P} \backslash P_i$, store (varid, $x$) if varid is a fresh identifier. Set State to 2.

**Output:** On input (**output**, $i$, varid) from all parties $P_i$, if varid exists in memory, TTP retrieves (varid, $x$) and outputs it to the adversary $\mathcal{A}$. If $\mathcal{A}$ inputs OK, $x$ is output to all parties, and State is reset to 0. If $\mathcal{A}$ inputs $\perp$, then $\perp$ is output to all parties, and the protocol terminates.

Figure: The ideal functionality for circuit-private MPC.

### BMR vs SS vs FHE-based MPC

- Without corrupted parties, BMR or secret sharing simulators cannot easily generate views without circuit info.

# Circuit privacy: Why we use SSFHE

## Current circuit-private MPC ideal functionaliy: $\mathcal{F}_{\text{CPMPC}}$

**Ideal functionality of $\mathcal{F}_{\text{CPMPC}}$**

**Init:** On input (**init**, $t$) from all parties, store the threshold $t$. Initialize the register State to 0.

**CircuitInit:** On input (**CircInit**, $i, \mathcal{C}, I$) from all parties $P_i$, if State is 0, store the circuit $\mathcal{C}$ and leak the number of inputs $I$ and the input-to-party mapping to the adversary $\mathcal{A}$. Set State to 1.

**Input:** On input (**input**, $P_i$, varid, $x$) from all parties $P_i$ and (input, $P_i$, varid, ?) from all other parties in $\mathcal{P} \backslash P_i$, store (varid, $x$) if varid is a fresh identifier. Set State to 2.

**Output:** On input (**output**, $i$, varid) from all parties $P_i$, if varid exists in memory, TTP retrieves (varid, $x$) and outputs it to the adversary $\mathcal{A}$. If $\mathcal{A}$ inputs OK, $x$ is output to all parties, and State is reset to 0. If $\mathcal{A}$ inputs $\perp$, then $\perp$ is output to all parties, and the protocol terminates.

Figure: The ideal functionality for circuit-private MPC.

## BMR vs SS vs FHE-based MPC

- Without corrupted parties, BMR or secret sharing simulators cannot easily generate views without circuit info.
- (FHE-based MPC) Simulator need not construct output ciphertexts, thus security proof is feasible.

# Circuit-Private MPC

## Theorem (Informal)

*Let:*

- *An* ($\delta$-IND-CPA, $\epsilon$-correctness)*-secure SSFHE scheme be given,*
- *A* $\kappa$*-soundness secret sharing scheme be used,*
- *And all errors in the decrypted ciphertext are bounded by B..*

*Then, under the UC framework, the adversary's computational advantage is bounded by:*

$$\delta + (B - 1) \cdot 2^{-\kappa+2} + 2\epsilon$$

# Future Work (On Progress...)

- **SPDZ over 16-bit Primes:**
  - Explore implementation of SPDZ using small 16-bit NTT-friendly primes
  - Leverage vector Oblivious Linear Evaluation (vOLE) for efficient multiplication triples
  - Aim to reduce computation and memory cost while preserving active security

# Future Work (On Progress...)

- **SPDZ over 16-bit Primes:**
  - Explore implementation of SPDZ using small 16-bit NTT-friendly primes
  - Leverage vector Oblivious Linear Evaluation (vOLE) for efficient multiplication triples
  - Aim to reduce computation and memory cost while preserving active security
- **SSFHE with GOD in Honest Majority:**
  - Construct SSFHE protocol under honest majority assumption
  - Ensure **Guaranteed Output Delivery (GOD)** despite adversarial behavior
  - Implement and evaluate performance in real-world parameters

# Thank you

waLLLnut

# References I


Zama.
*TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data*, 2022.
https://github.com/zama-ai/tfhe-rs

Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, Vincent Zucca.
*OpenFHE: Open-Source Fully Homomorphic Encryption Library*, 2022.
Cryptology ePrint Archive, Paper 2022/915.
https://eprint.iacr.org/2022/915

Donald Beaver, Silvio Micali, and Phillip Rogaway,
"The Round Complexity of Secure Protocols,"
In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 503–513, 1990.
https://dl.acm.org/doi/10.1145/100216.100287

# References II


- Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias,
  "Multiparty Computation from Somewhat Homomorphic Encryption,"
  In *Advances in Cryptology – CRYPTO 2012*, Lecture Notes in Computer Science, vol. 7417, pp. 643–662, Springer, 2012.
  https://doi.org/10.1007/978-3-642-32009-5_38
- S. Dov Gordon, Feng-Hao Liu, and Elaine Shi,
  "Constant-Round MPC with Fairness and Guarantee of Output Delivery,"
  In *Advances in Cryptology – CRYPTO 2015*, Lecture Notes in Computer Science, vol. 9216, pp. 63–82, Springer, 2015.
  https://eprint.iacr.org/2015/371
- J. Park and S. Rovira,
  "Efficient TFHE Bootstrapping in the Multiparty Setting,"
  *Cryptology ePrint Archive*, Report 2023/759, 2023.
  https://eprint.iacr.org/2023/759

# References III


Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter,
"Noah's Ark: Efficient Threshold-FHE Using Noise Flooding,"
In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pp. 35–46, 2023.

Hao Chen, Ilaria Chillotti, and Yongsoo Song,
"Multi-key Homomorphic Encryption from TFHE,"
In *Advances in Cryptology – ASIACRYPT 2019, Part II*, 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, pp. 446–472, Springer, 2019.

Tanping Zhou, Long Chen, Xiaoliang Che, Wenchao Liu, Zhenfeng Zhang, and Xiaoyuan Yang,
"Multi-key Fully Homomorphic Encryption Scheme with Compact Ciphertexts,"
*Cryptology ePrint Archive*, Report 2021.
https://eprint.iacr.org/2021

# References IV


- Oded Regev,
  "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography,"
  In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 84–93, 2005.
  https://doi.org/10.1145/1060590.1060603

- Loris Bergerat, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap,
  "TFHE Gets Real: An Efficient and Flexible Homomorphic Floating-Point Arithmetic,"
  *Cryptology ePrint Archive*, Report 2025.
  https://eprint.iacr.org/2025

- Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood,
  "Actively Secure Setup for SPDZ,"
  *Journal of Cryptology*, vol. 35, no. 1, p. 5, Springer, 2022.

# References V

Andreea Alexandru, A. Al Badawi, N. Genise, D. Micciancio, Y. Polyakov, S. Ramanathapuram V. V., and V. Vaikuntanathan,
"Building Blocks for Threshold FHE,"
presented at *NIST MPTS 2023*.

Helger Lipmaa, Payman Mohassel, and Saeed Sadeghian.
Valiant's universal circuit: Improvements, implementation, and applications.
*Cryptology ePrint Archive*, 2016.