Multiparty Homomorphic Encryption: from Theory to Practice

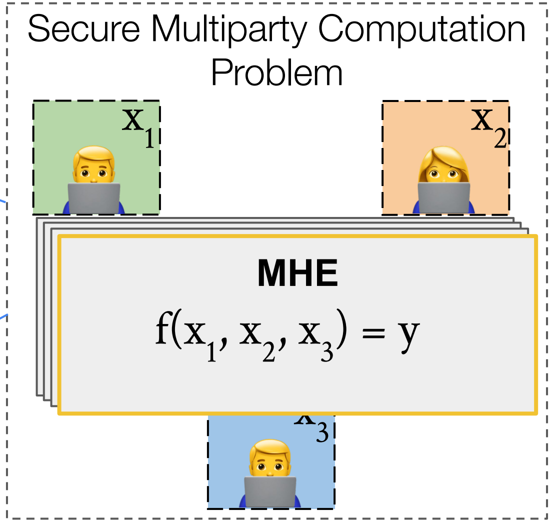
Christian Mouchet, HPI

@ FHE:IDEAs Workshop
25.05.2024

HPI Hasso Plattner Institut

Secure Multiparty Computation

Multiple parties want to **compute** a public function **without disclosing** their inputs.



[Functionality]
"Output y"

[Input privacy]
"Without revealing more information about the inputs than what y does"

$\mathcal{P}$

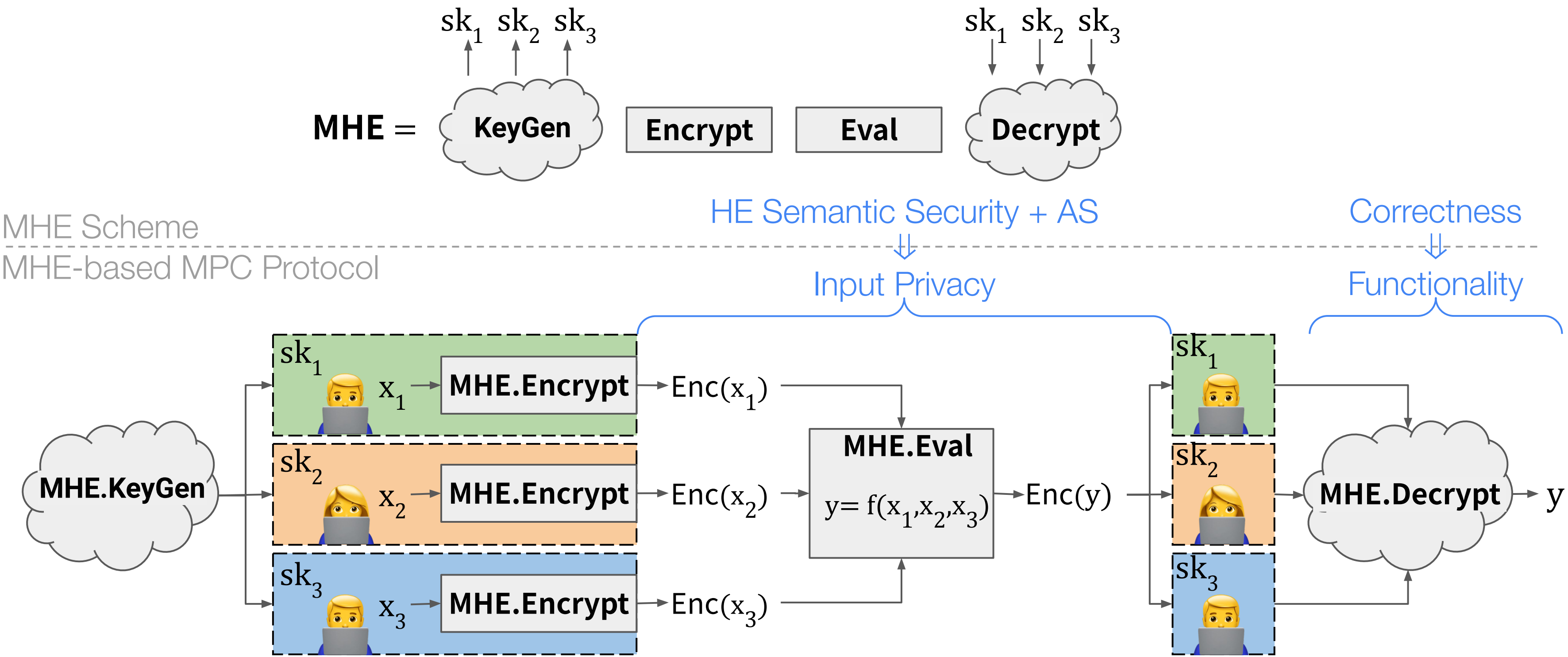
- N parties

$\mathscr{A} \subset \mathcal{P}$

- N-1 adversaries
- Passive and static

Multiparty Homomorphic Encryption – Intuition

Multiparty Homomorphic Encryption (MHE) extends Homomorphic Encryption (HE) with an **access-structure**.



sk_1 sk_2 sk_3 sk_1 sk_2 sk_3

MHE = **KeyGen** **Encrypt** **Eval** **Decrypt**

MHE Scheme

HE Semantic Security + AS

⇓

Correctness

⇓

MHE-based MPC Protocol

Input Privacy

Functionality

MHE.KeyGen

sk_1 x_1 → **MHE.Encrypt** → $Enc(x_1)$

sk_2 x_2 → **MHE.Encrypt** → $Enc(x_2)$

sk_3 x_3 → **MHE.Encrypt** → $Enc(x_3)$

MHE.Eval

$y= f(x_1,x_2,x_3)$

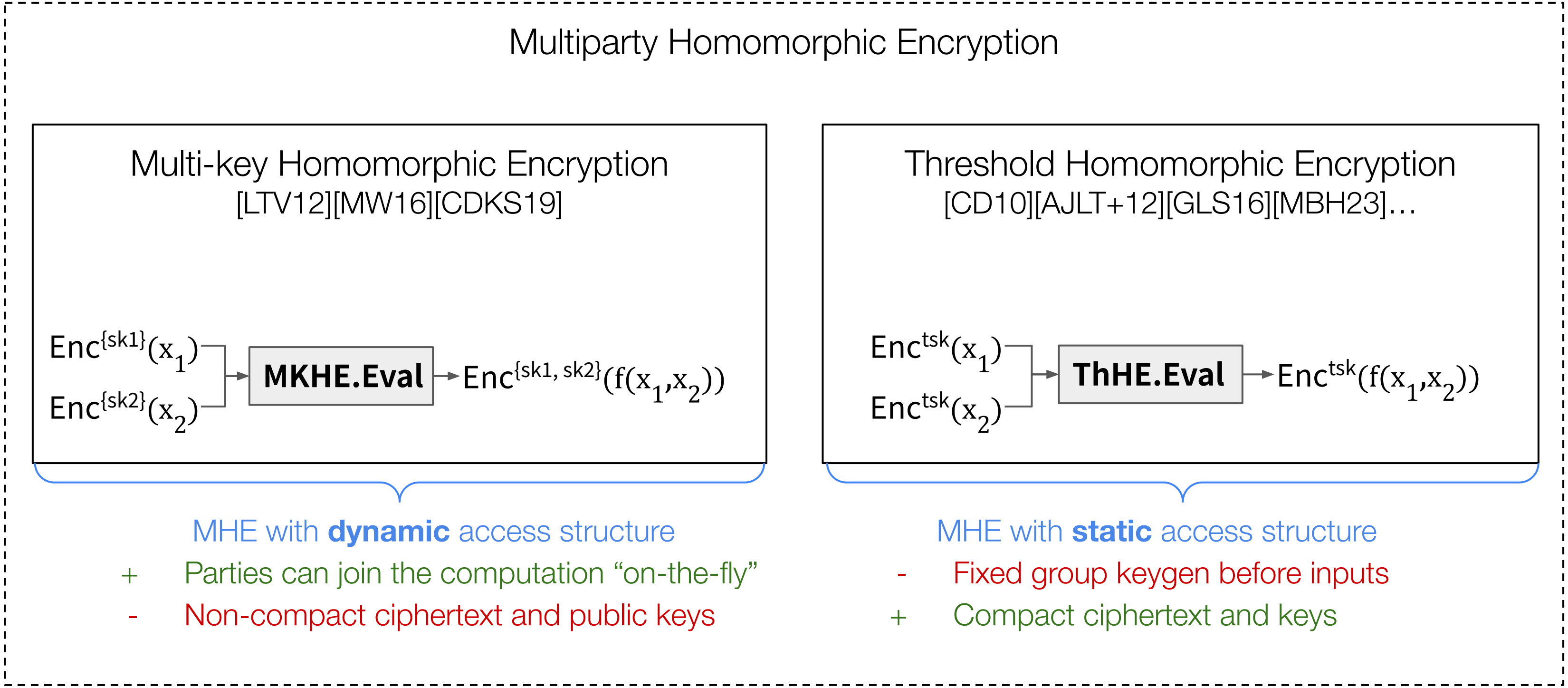
→ $Enc(y)$

sk_1 sk_2 sk_3

MHE.Decrypt → y

Multiparty Homomorphic Encryption – Two Main Families

There are two main families of MHE schemes.



Multiparty Homomorphic Encryption

Multi-key Homomorphic Encryption
[LTV12][MW16][CDKS19]

$\mathrm{Enc}^{\{sk1\}}(x_1)$, $\mathrm{Enc}^{\{sk2\}}(x_2)$ → **MKHE.Eval** → $\mathrm{Enc}^{\{sk1,\, sk2\}}(f(x_1,x_2))$

MHE with **dynamic** access structure

- \+ Parties can join the computation “on-the-fly”
- \- Non-compact ciphertext and public keys

Threshold Homomorphic Encryption
[CD10][AJLT+12][GLS16][MBH23]…

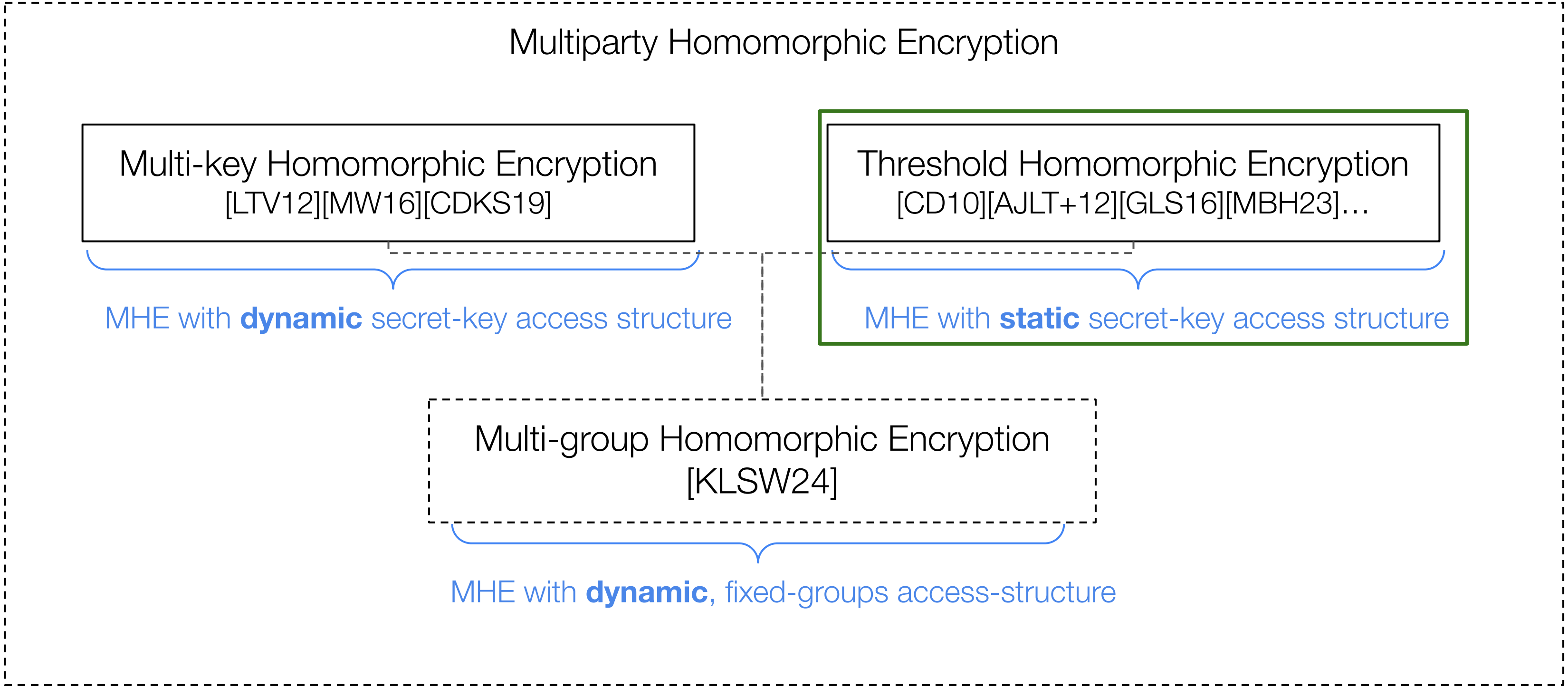
$\mathrm{Enc}^{tsk}(x_1)$, $\mathrm{Enc}^{tsk}(x_2)$ → **ThHE.Eval** → $\mathrm{Enc}^{tsk}(f(x_1,x_2))$

MHE with **static** access structure

- \- Fixed group keygen before inputs
- \+ Compact ciphertext and keys

Multiparty Homomorphic Encryption – Two Orthogonal Families

There are two main families of MHE schemes.



Multiparty Homomorphic Encryption

Multi-key Homomorphic Encryption
[LTV12][MW16][CDKS19]

MHE with **dynamic** secret-key access structure

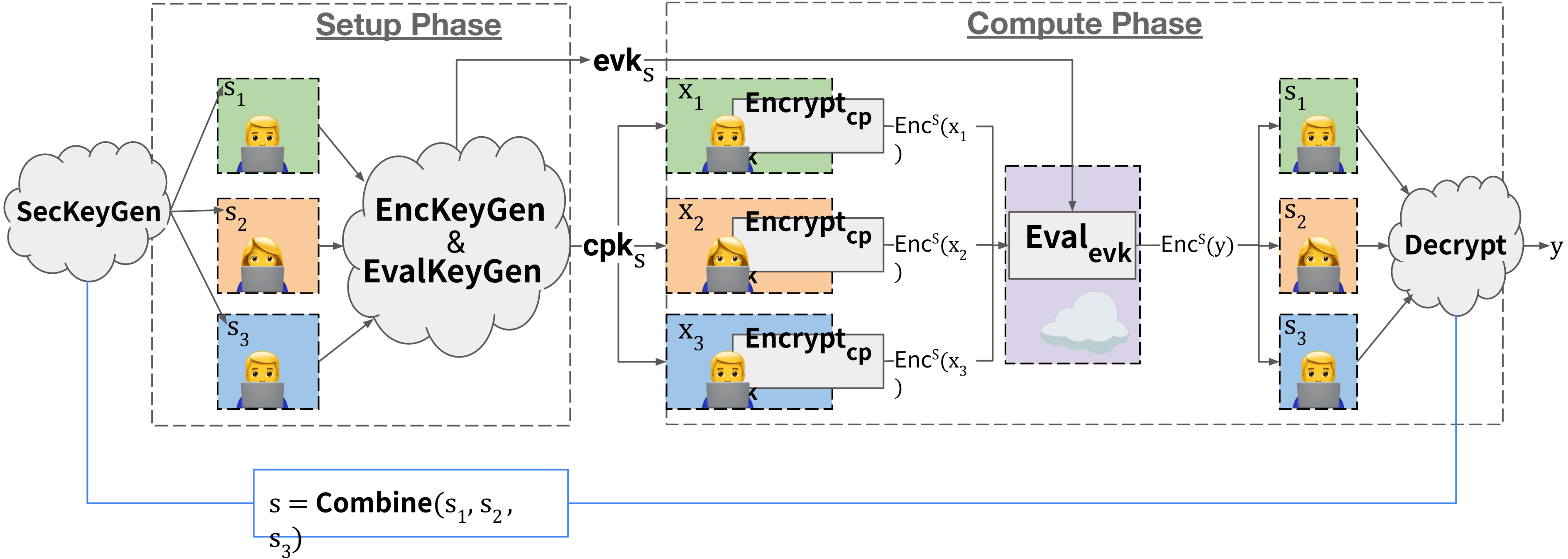
Threshold Homomorphic Encryption
[CD10][AJLT+12][GLS16][MBH23]…

MHE with **static** secret-key access structure

Multi-group Homomorphic Encryption
[KLSW24]

MHE with **dynamic**, fixed-groups access-structure

MHE-based MPC (Threshold-FHE case)



N-out-of-N-threshold: $s = s_1 + s_2 + s_3$

T-out-of-N-threshold: $s = \Delta_1 s_1 + \Delta_2 s_2$

Background: Ring learning-with-error [LPR10]

RLWE distribution:

Let:

$R = \mathbb{Z}_q[X]/(X^n+1)$	be a ring of degree n-1 polynomials with coefficients $\mathrm{mod}\ q$,
$\mathsf{U}(R)$	be the uniform distribution over R,
$\mathsf{Err}(R)$	be an error distribution over R $(\|\|e\|\| << q,\ e \leftarrow \mathsf{Err}(R))$,
$s \in R$	be a secret value in R

the *ring learning-with-error distribution* over s is defined as:

$$\mathsf{RLWE}_s := (sa + e,\ a) \qquad a \leftarrow \mathsf{U}(R) \qquad e \leftarrow \mathsf{Err}(R)$$

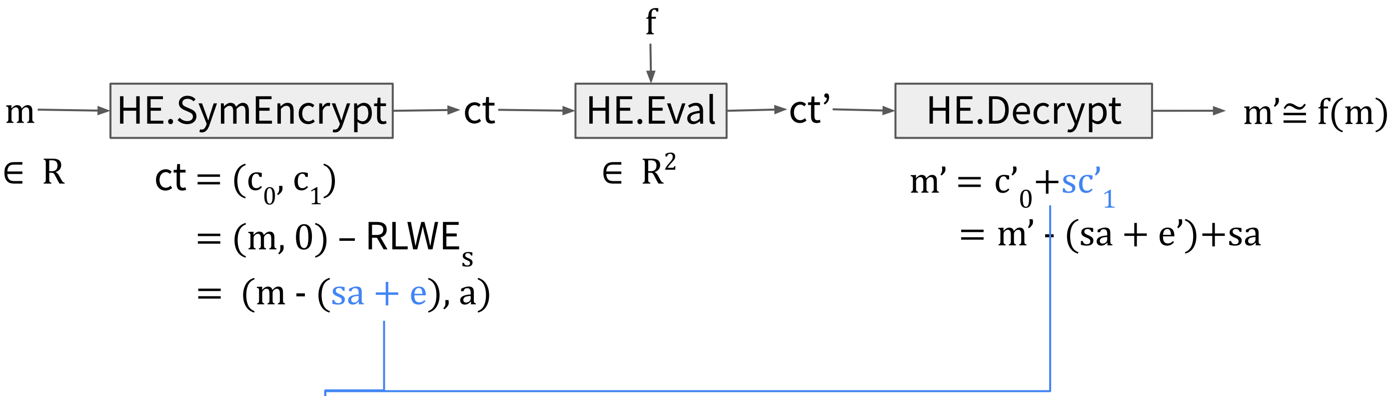
Given a polynomial number of <u>independent</u> samples from the RLWE_s distribution:

- **Search**: find s.
- **Decision**: distinguish from $\mathsf{U}(R^2)$

Background: (Symmetric) HE From RLWE

A simplified RLWE-based HE scheme.

Let $f: R \to R$, and $||s|| = 1$



$$\begin{aligned} ct &= (c_0, c_1) \\ &= (m, 0) - \mathrm{RLWE}_s \\ &= (m - (sa + e), a) \end{aligned}$$

$$\begin{aligned} m' &= c'_0 + sc'_1 \\ &= m' - (sa + e') + sa \end{aligned}$$

Scheme's operations are affine functions of the secret-key.

Secret-key operations are affine functions of the secret key

Other operations are also affine functions of the secret-key: $sa + e + x$

Setup phase:

Public Encryption Key Generation: $(sa + e, a)$

Public Rotation Key Generation for $\mathrm{rot}_k(\cdot)$: $(s\mathbf{b} + \mathbf{e} + \mathrm{rot}_{-k}(s)\mathbf{w}, \mathbf{b})$

Public Relinearization Key Generation: $(s\mathbf{d} + \mathbf{e} + s^2\mathbf{w}, \mathbf{d})$

Compute phase:

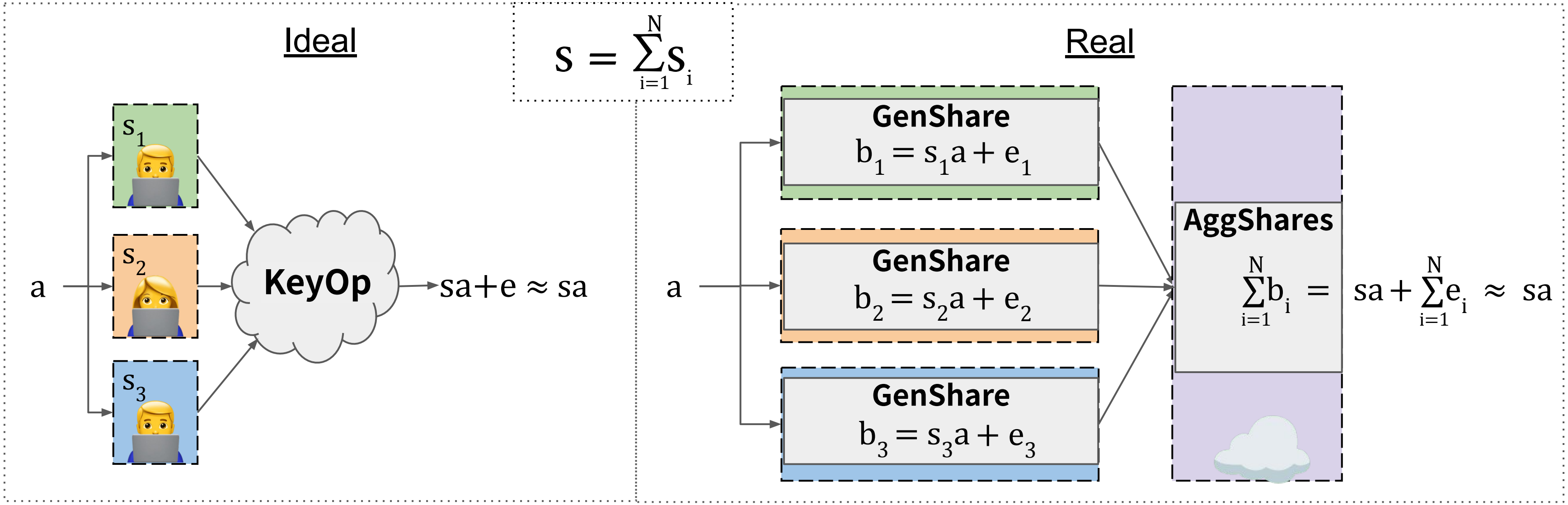
Decrypt: $sc_1 + e + c_0$

Re-encrypt: $((s\text{-}s')c_1 + e + c_0, c_1)$

MHE Scheme Construction – Secret-key Operations

Affine secret-key operations can be implemented as single-round protocols (Generalizing [AJLT+12][MTBH+21]).

→ We refer to these protocols as having **P**ublic **A**ggregatable **T**ranscripts (PAT)



Helper-Assisted, MHE-based MPC

The MHE-based MPC protocol has many practical advantages. [MTBH+21]

One-time setup

- ✓ Amortizable cost
- ✓ Session-like paradigm

Low communication complexity

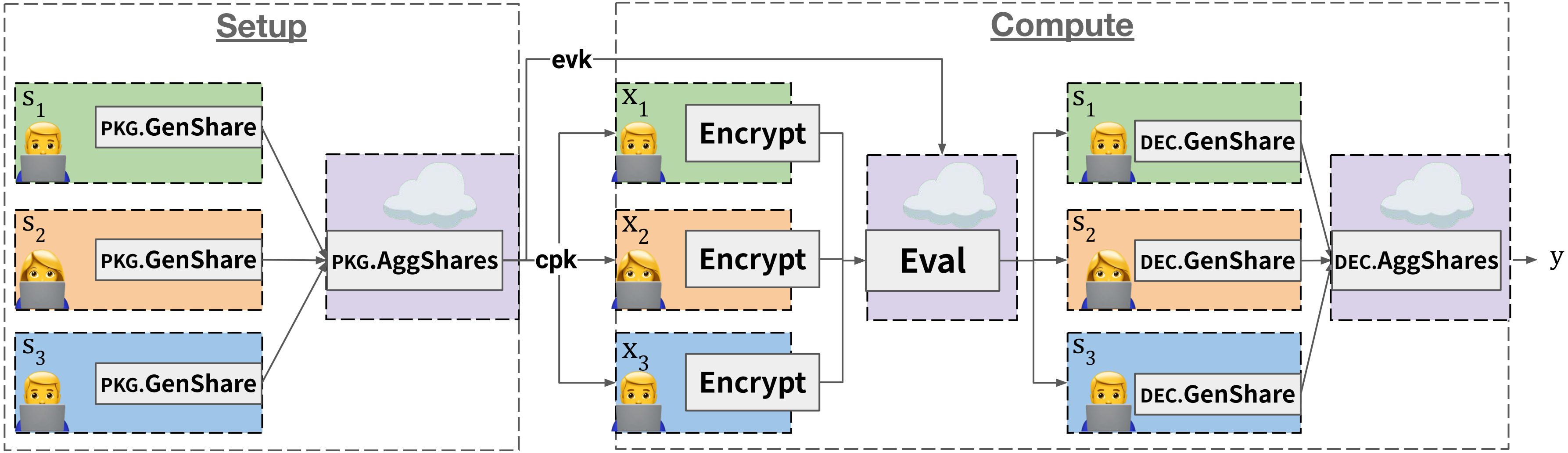
- ✓ 2+2 rounds
- ✓ Non-interactive Eval

Public Transcript

- ✓ Delegated public share aggregation
- ✓ Sublinear MPC

Delegated evaluation

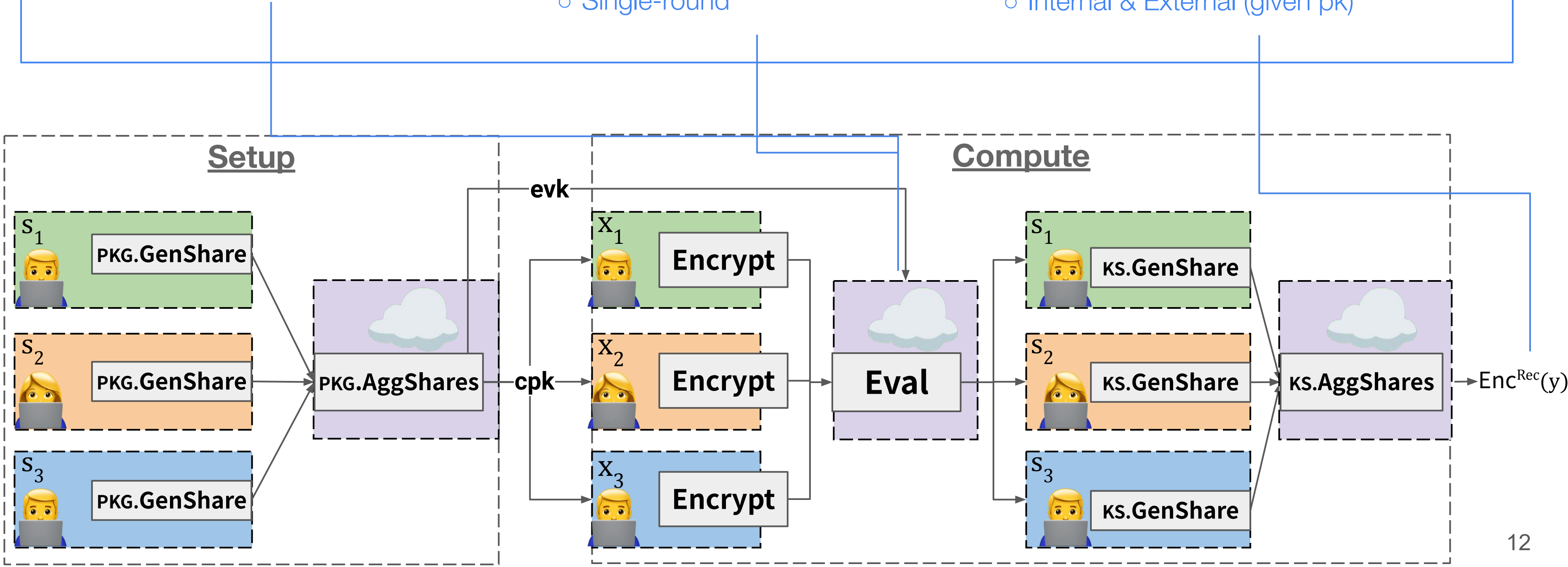
- ✓ In classic passive-adversary setting



Helper-Assisted, MHE-based MPC

Practicality Enhancements [**M**TBH21]

- ✓ Switching to/from data-level SS
  - ○ Single-round
- ✓ Refresh protocol
  - ○ Interactive “bootstrapping”
  - ○ Single-round
- ✓ Proxy-reencryption to design. receiver
  - ○ DEC → (P)KS
  - ○ Internal & External (given pk)



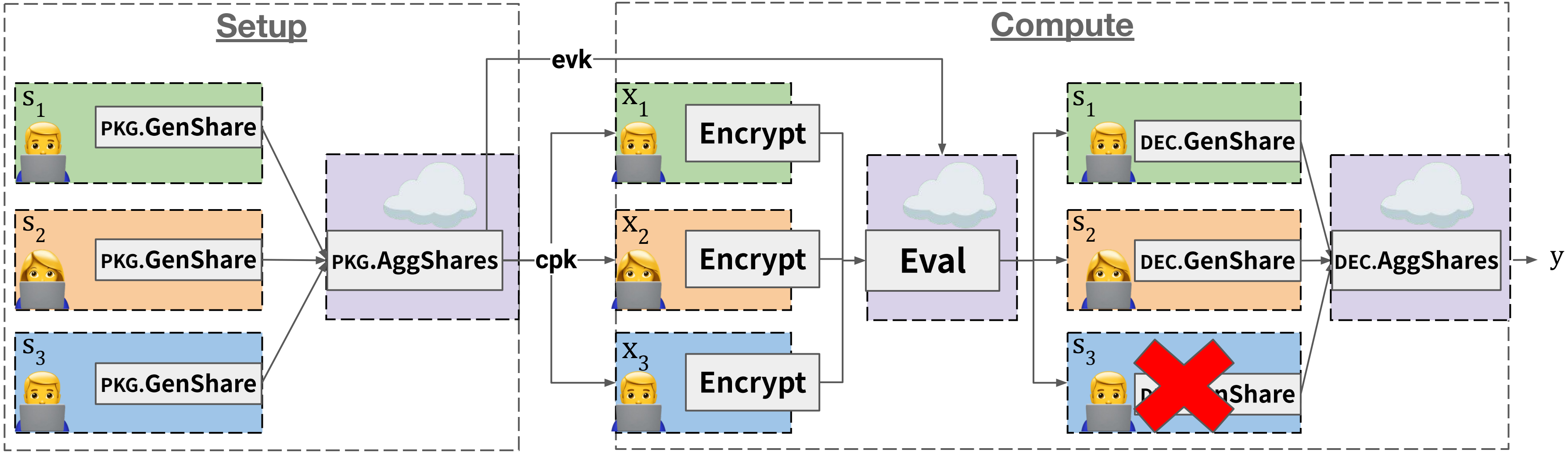
Helper-Assisted, MHE-based MPC

Fault-tolerance ?
- Temporary disconnects & Reboots
- Full crashes

"Not that bad"
- Most of the protocol state is public
- Delays the result

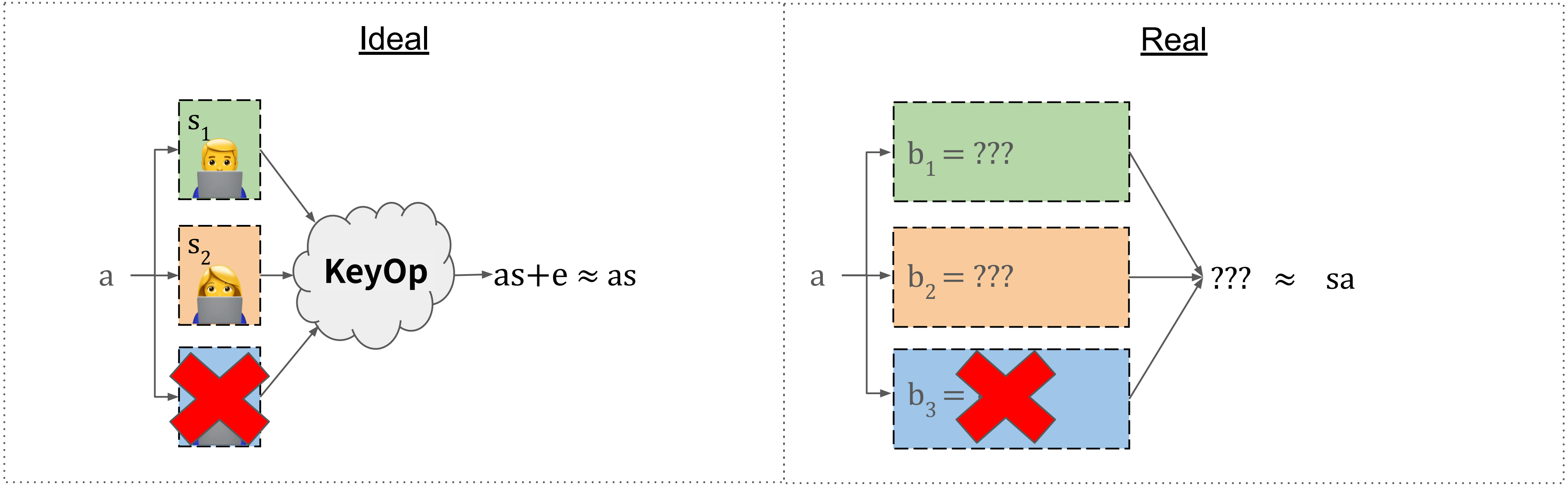
"Bad"
- … but "by design" when $|\mathcal{A}| = N\text{-}1$

→ What about $|\mathcal{A}| < T$?



T-out-of-N-Threshold Secret-Key Operations ?

Running PAT protocols among $\mathrm{T} < \mathrm{N}$ parties.



Previous Approaches

Previous approaches either require a trusted dealer or are leaky (and are all non-compact)

Compromises the secret-keys of offline parties
→ Compromises the “session”.

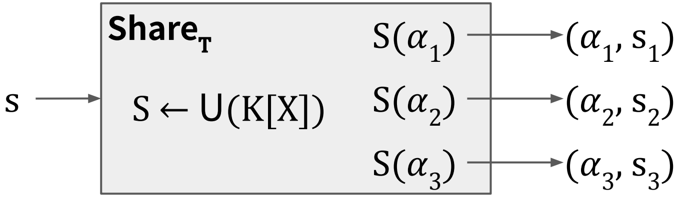
Requiring non-constant-size secrets
→ Leads to costly storage & ops.

Approach	**Trusted dealer**	**Leaky**	**Non-Compact**
1. Two-step [AJLT+12]	No	Yes	Yes
2. Single-step [BGGJ+18]	Yes	No	Yes
Ours [**M**BH23]	No	No	No

Shamir Secret-Sharing Scheme Reminder [Shamir 1979]

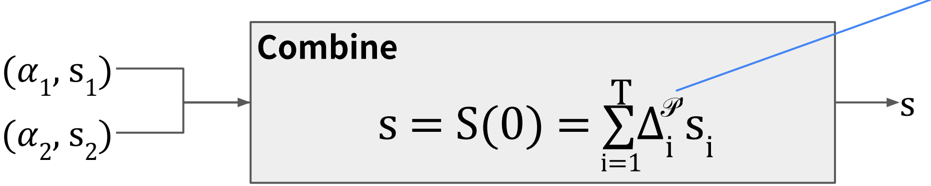
Shares are points on a uniformly random polynomial S over some finite field K where:

- S has degree-$(T\text{-}1)$ and
- $s = S(0)$.



The secret reconstruction is a linear combination of the shares with the **Lagrange interpolation coefficients:**

For $\mathcal{P}' \subset \mathcal{P}$ $|\mathcal{P}'| \geq T$ (w.l.o.g. assume $\mathcal{P}' = \{P_1, P_2, \ldots, P_T\}$):



$$s = S(0) = \sum_{i=1}^{T} \Delta_i^{\mathscr{P}} s_i$$

Lagrange coefficients depend on $\mathscr{P}$

$$\Delta_i^{\mathscr{P}} = \prod_{\substack{j=1 \\ j\neq i}}^{T} \alpha_j / (\alpha_j - \alpha_i)$$

Approach 1: Share Re-sharing + Two-steps Key-operations

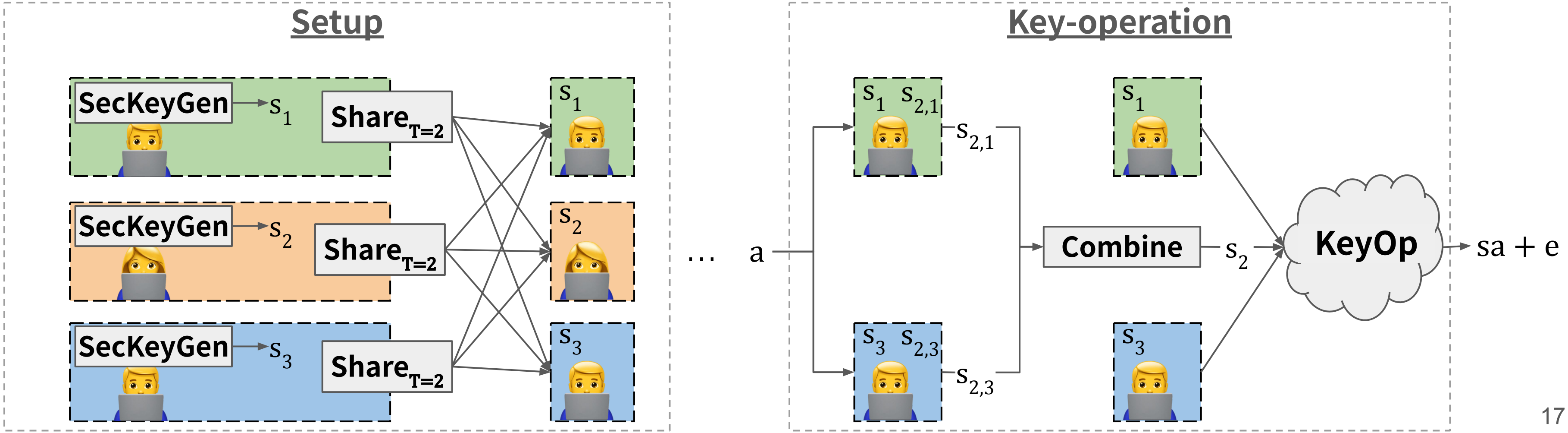
Asharov et al. proposed a share re-sharing scheme with two-steps key-operations [AJLT+12].

Pros

- ☑ No constraints on the field $\mathbb{K}$
- ☑ No need for trusted dealers

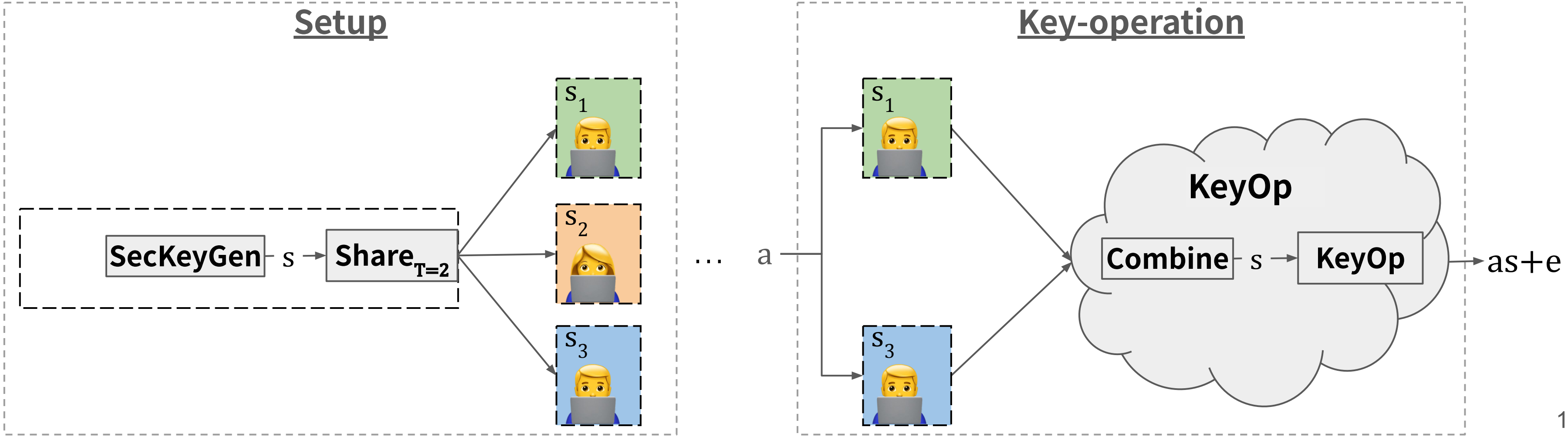
Cons

- ☒ Non-compact
- ☒ Leaks the failing parties' shares



Approach 2: Single-step Key-operation

Boneh et al. proposed a non-leaky approach based on a special sharing scheme ({0,1}-LSSS) [BGGJ+18].

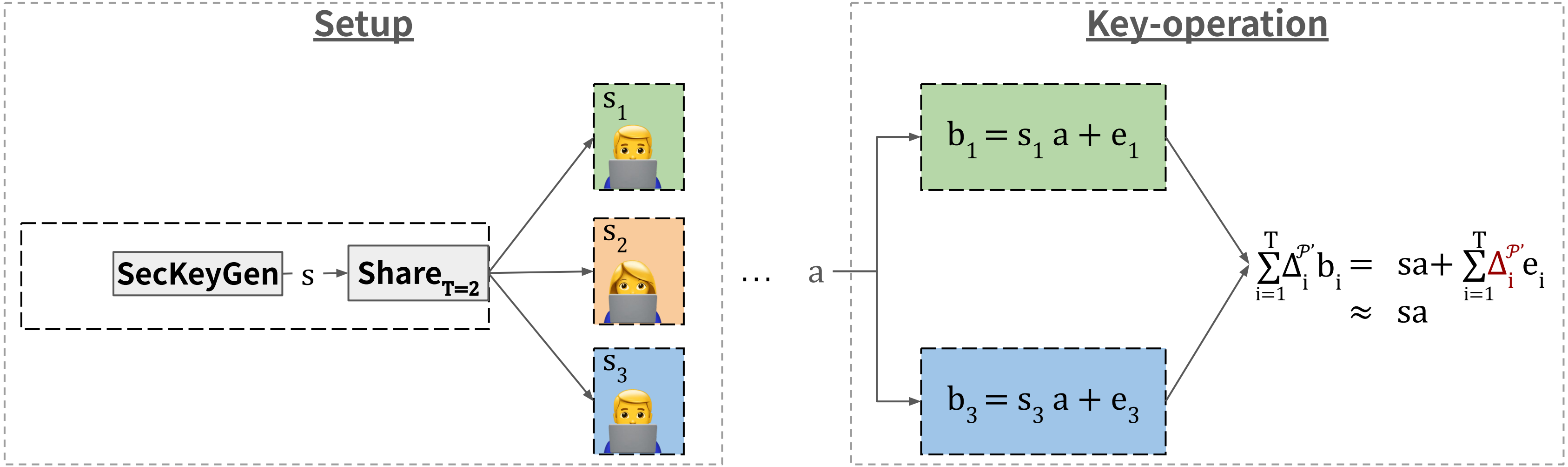


Approach 2: Single-step Key-operation

Boneh et al. proposed a non-leaky approach based on a special sharing scheme ({0,1}-LSSS) [BGGJ+18].

Pros
- ☑ Protects the failing parties' shares

Cons
- ☒ Non-compact ($O(N^{4.2})$)
- ☒ Requires a trusted dealer



Setup

SecKeyGen → s → **Share$_{T=2}$**

s_1

s_2

s_3

…

a

Key-operation

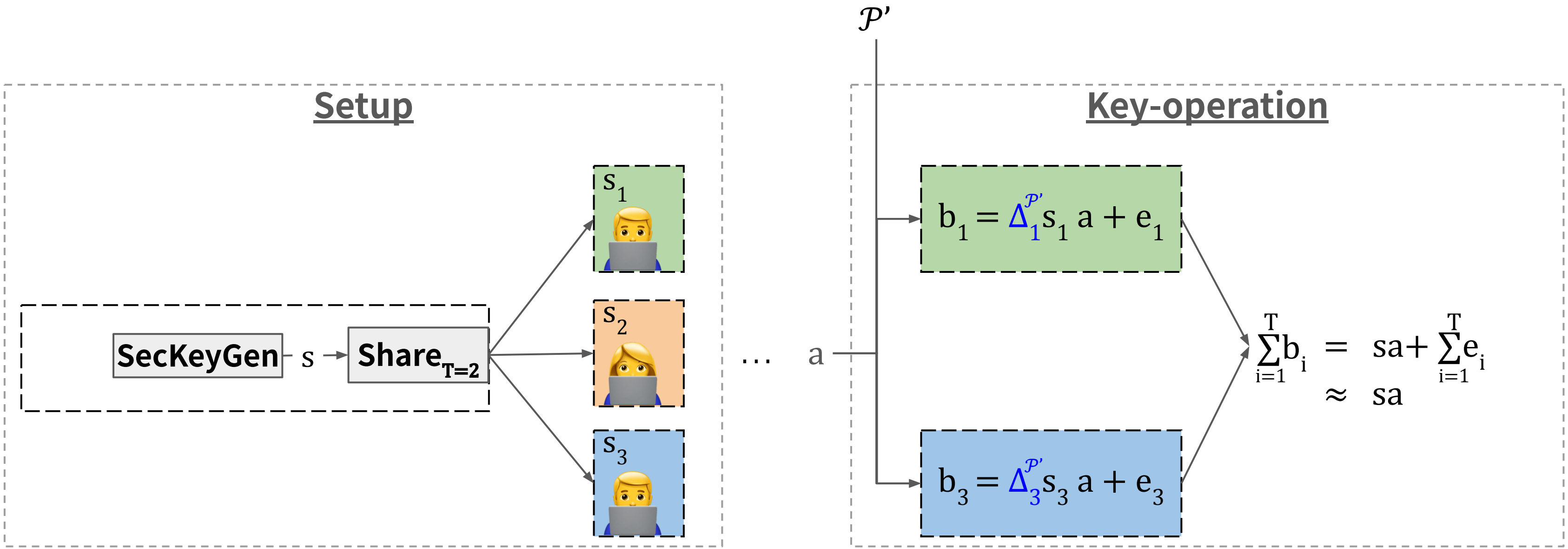
$b_1 = s_1 a + e_1$

$b_3 = s_3 a + e_3$

$$\sum_{i=1}^{T} \Delta_i^{\mathcal{P}'} b_i = sa + \sum_{i=1}^{T} \Delta_i^{\mathcal{P}'} e_i$$
$$\approx sa$$

Our Approach – Intuition

[MBH23]: If the parties know the set of online parties $\mathcal{P}'$ *before* computing their shares, there is a neat trick.



Setup

SecKeyGen s **Share$_{T=2}$**

s_1

s_2

s_3

…

a

$\mathcal{P}'$

Key-operation

$b_1 = \Delta_1^{\mathcal{P}'} s_1 a + e_1$

$b_3 = \Delta_3^{\mathcal{P}'} s_3 a + e_3$

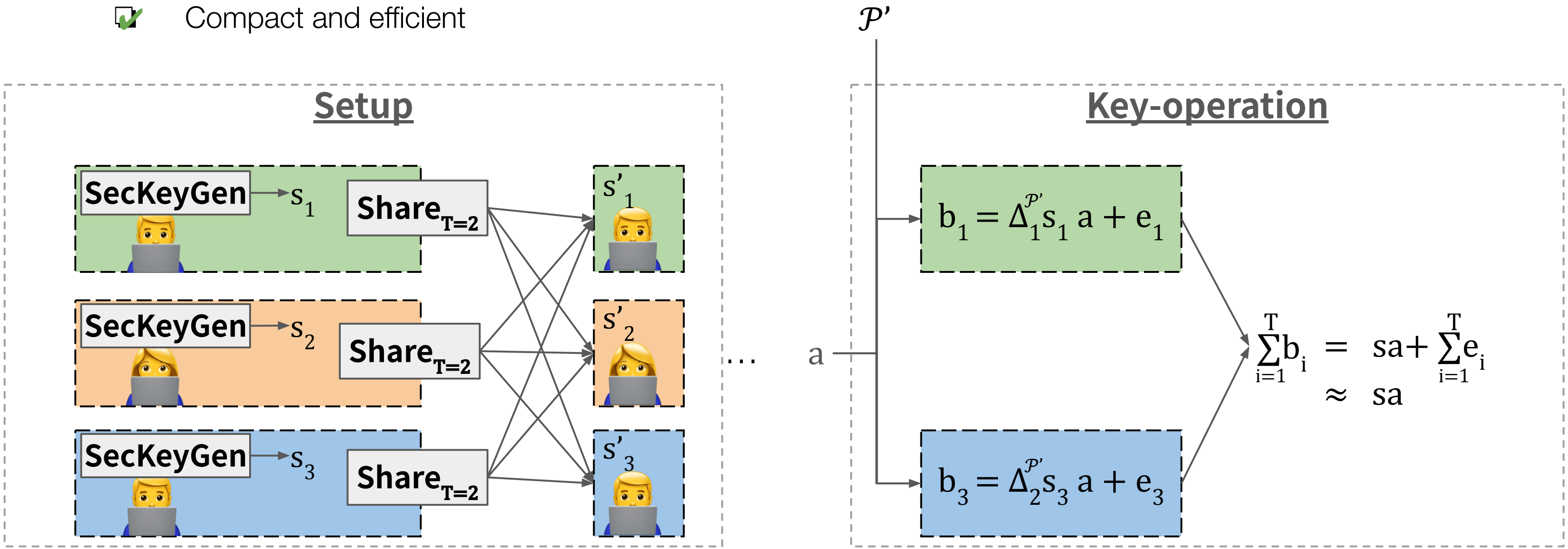
$$\sum_{i=1}^{T} b_i = sa + \sum_{i=1}^{T} e_i$$
$$\approx sa$$

Our Approach: Share Re-Sharing + Optimistic Key-operation

This trick combined with the share re-sharing approach yields an highly efficient solution. [MBH23]

Pros

- ☑ Protects the failing parties’ shares
- ☑ No trusted dealer
- ☑ Compact and efficient



Setup

SecKeyGen → s_1 **Share$_{T=2}$** → s'_1

SecKeyGen → s_2 **Share$_{T=2}$** → s'_2

SecKeyGen → s_3 **Share$_{T=2}$** → s'_3

…

Key-operation

$\mathcal{P}'$

a

$b_1 = \Delta_1^{\mathcal{P}'} s_1\ a + e_1$

$b_3 = \Delta_2^{\mathcal{P}'} s_3\ a + e_3$

$$\sum_{i=1}^{T} b_i = sa + \sum_{i=1}^{T} e_i \approx sa$$

Our Approach: Properties

Let:

$s = \sum^{N} s_i$ be the ideal secret-key in the N-out-of-N scheme

$s_{i,j} = S_i(\alpha_j)$ be the re-share of s_i held by party j in the T-out-of-N scheme

Then, for any $\mathcal{P}' \subset \mathcal{P}$, $|\mathcal{P}'| \geq T$, we can express S as:

$$s = \sum_{i=1}^{N} s_i = \sum_{i=1}^{N}\sum_{j=1}^{T}\Delta_j^{\mathcal{P}'} s_{i,j} = \sum_{j=1}^{T}\boxed{\sum_{i=1}^{N}\Delta_j^{\mathcal{P}'} s_{i,j}} = \sum_{j=1}^{T}\Delta_j^{\mathcal{P}'}\boxed{\sum_{i=1}^{N} s_{i,j}} = \boxed{\sum_{j=1}^{T} s_j^{\mathcal{P}'}}$$

Sum-over-T terms contains re-shares held by party j only.

Sum-over-N terms do not depend on the set of online parties $\mathcal{P}'$.

T-out-of-T additive sharing of the collective secret-key.

Non-leakyness / Correctness

The each party can compute its share locally given $\mathcal{P}'$.

Compactness

T-out-of-N re-shares can be aggregated in the setup phase.

Modularity

We can use the protocols of the N-out-of-N scheme for $N=T$.

Our Approach: Discussion

The dependence on the online parties' oracle introduces **two requirements**:

1. Implementation of the oracle

- Good $\mathcal{P}'$ requires accurate view over the network
- Requires consensus on the participant set $\mathcal{P}'$

2. A protocol-failure handling mechanism

- Parties can fail *after* $\mathcal{P}'$ was issued.
- Requires defining a (synchronous) "protocol failure" event.

Approach	**Trusted dealer**	**Leaky**	**Compact**	**Asynchronous**
Two-step [AJLT+12]	No	Yes	No	No
Single-step [BGGJ+18]	Yes	No	No	yes
Ours [**M**BH23]	No	No	Yes	No

Upside: Req. 1. + 2. can be realized in the passive, synchronous setting. So our scheme is highly relevant in this setting.

Downside: our method does not "directly" apply to stronger settings.

Implementation

Both the N-out-of-N- and the T-out-of-N-threshold scheme are implemented in Lattigo [**M**BTH20]

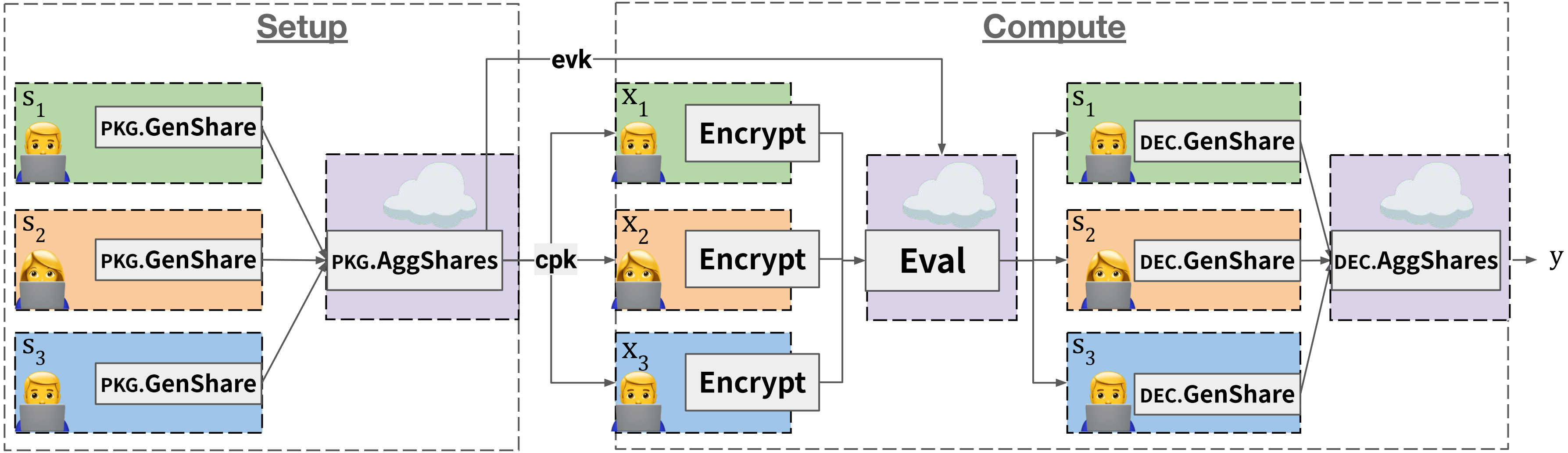
- Supports: (Threshold-) BGV, BFV, CKKS
- (Single-party) CKKS bootstrapping
- Standalone, modular math & crypto layers
- **Open-source** (Apache 2.0 licence)
- 100% written in Go
- Full suite of unit-test and benchmarks

LATTIGO

https://github.com/tuneinsight/lattigo

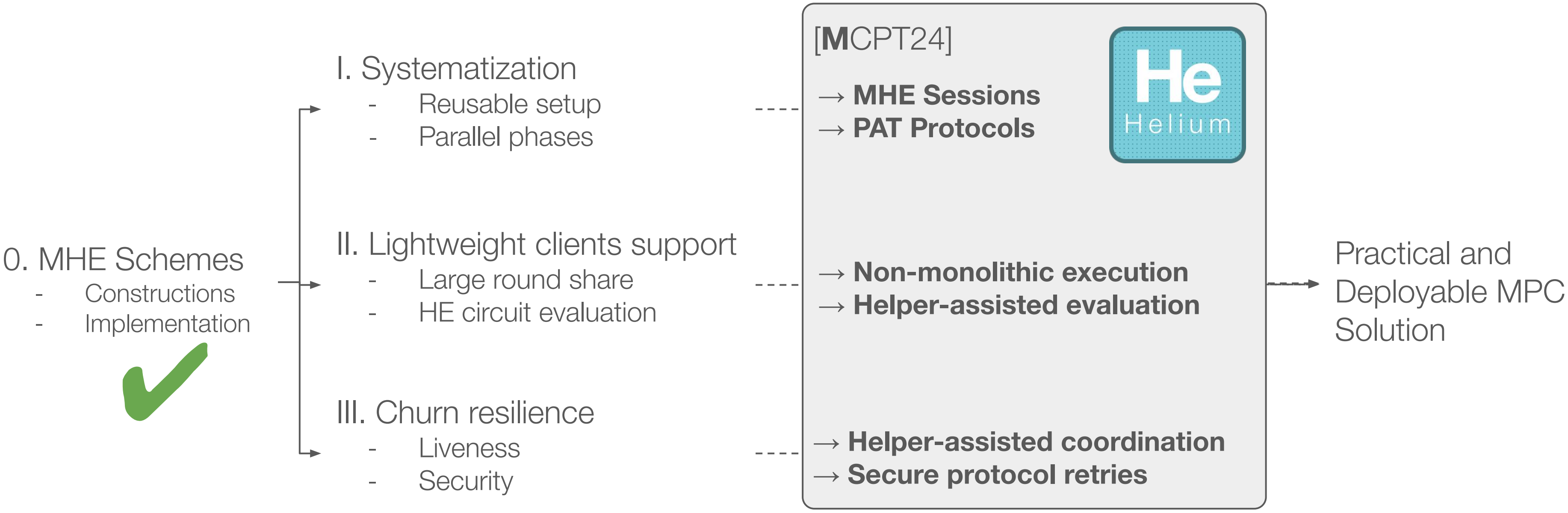
Fault-tolerant MHE-based MPC Protocol

Lattigo & OpenFHE provide the core element of the MHE-based MPC protocol…



Practical Challenges of MHE-based MPC

…, but the way to practice is full of challenges.



0. MHE Schemes
- Constructions
- Implementation

✓

I. Systematization
- Reusable setup
- Parallel phases

II. Lightweight clients support
- Large round share
- HE circuit evaluation

III. Churn resilience
- Liveness
- Security

[**M**CPT24]

He
Helium

→ **MHE Sessions**
→ **PAT Protocols**

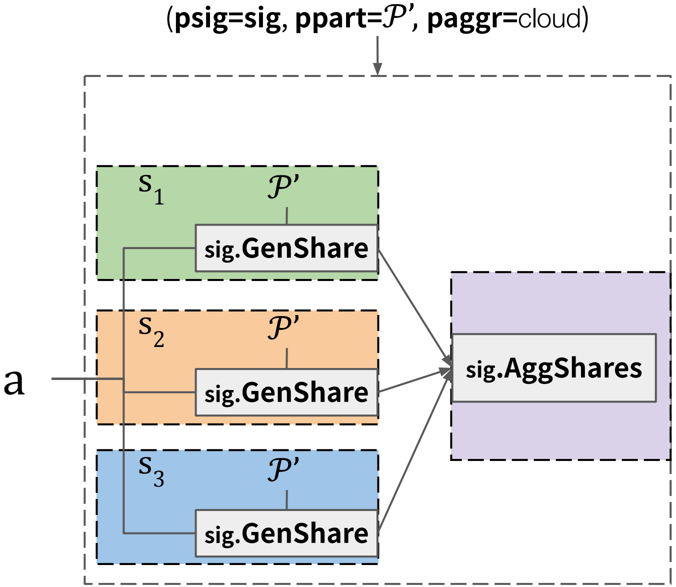
→ **Non-monolithic execution**
→ **Helper-assisted evaluation**

→ **Helper-assisted coordination**
→ **Secure protocol retries**

Practical and Deployable MPC Solution

Helium: Systematization

MHE-MPC reduces to running many **P**ublic **A**ggregatable **T**ranscript (PAT) **protocols** within a **session.**



PAT-protocol "mini-language":

psig := (**ptype**, **pargs**)

ptype ∈ {**CKG**, **EKG**, **DEC**, **KS**}

pargs: free protocols arguments
- operation identifier for **EKG**
- ciphertext identifier for **DEC/KS**

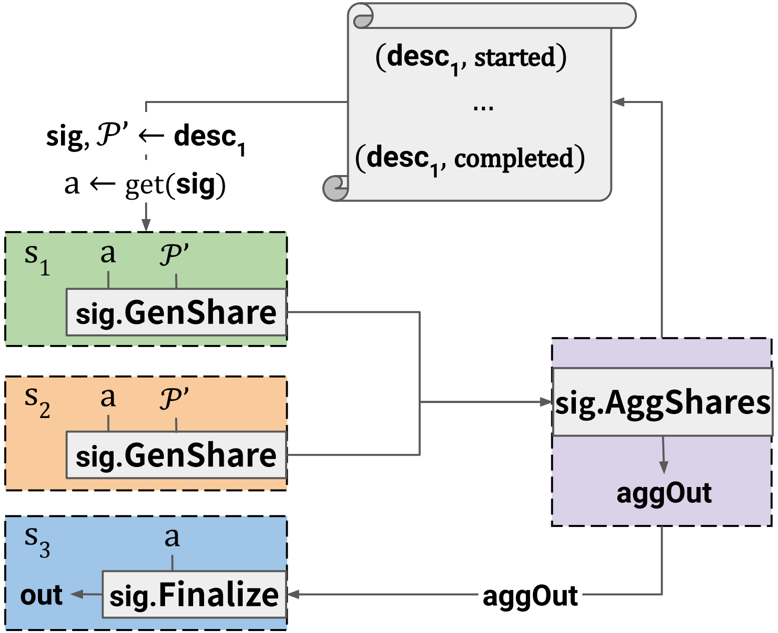
pdesc := (**psig**, **ppart, paggr**)

ppart $= \mathcal{P}' \subset \mathcal{P}, |\mathcal{P}'| = \mathrm{T}$

paggr: aggregator's network identity

Helium: Helper coordination

Helper orchestrates the execution via a **compact** public coordination log.

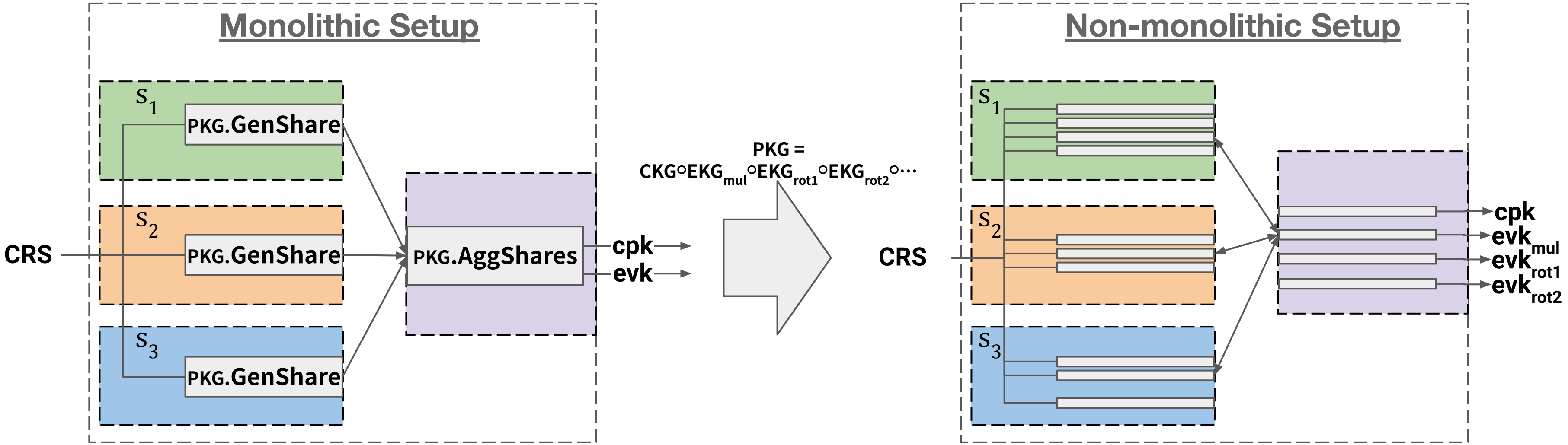


coordmsg := (**pdesc**, **pstatus**)

pstatus ∈ {**started**, **completed**}

coordlog := coordmsg || coordlog

Helium: Non-monolithic execution

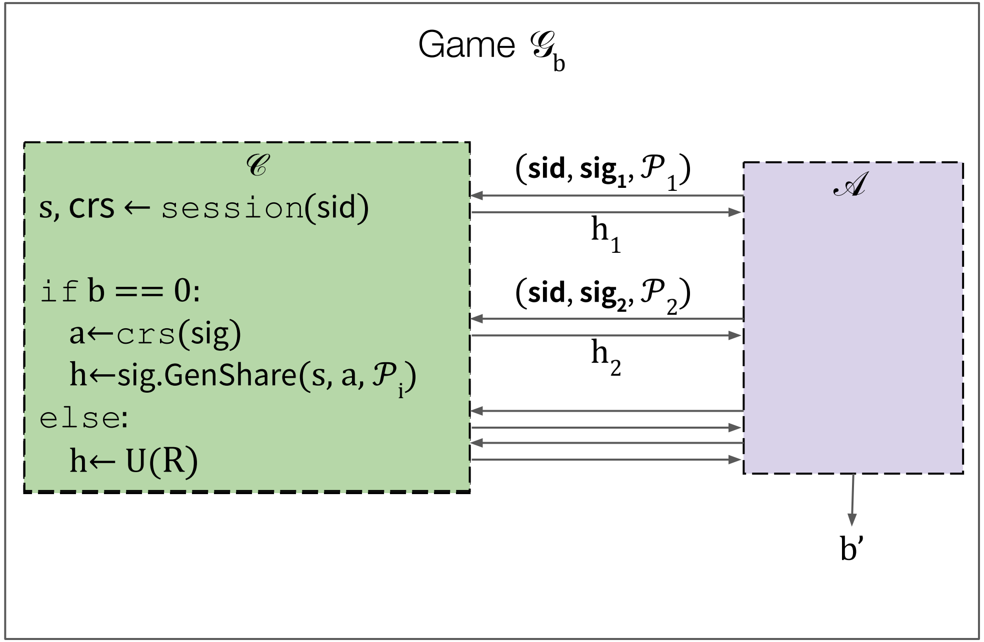


- Potentially **large round-1 message**
- Round can **fail** in the T-out-of-N Setting
- **Cannot distribute** the load if N_{online} > T
- No need for coordination: 1-2 round
- Matches the theoretical formulation

- **Small messages**: support weak nodes
- Dynamic Participant sets: **adapt** to network
- **Enables distribution** of the load
- Requires (finer grained) coordination
- Execution no longer match model of security proof

Helium: Modelling the Non-Monolithic Execution

Modelling the protocol execution mechanism as an interactive game (keygen case).



```
s, crs ← session(sid)

if b == 0:
  a←crs(sig)
  h←sig.GenShare(s, a, $\mathcal{P}_i$)
else:
  h← U(R)
```

- Security from RLWE assumption

$$\text{sig.GenShare}(s, a, \mathcal{P}) \sim \Delta^{\mathcal{P}} sa + e$$

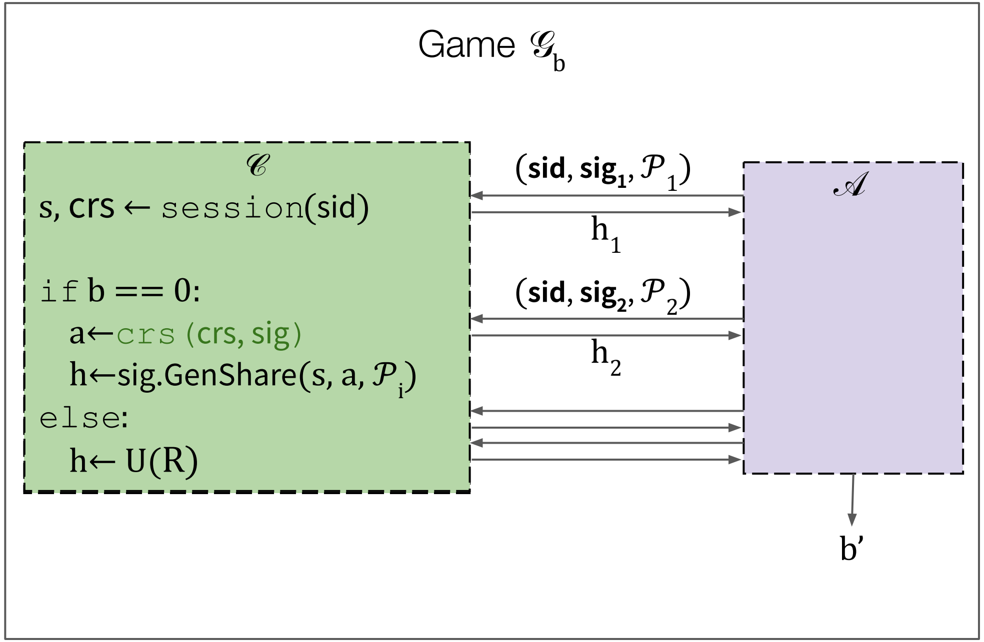
→ holds if $\mathscr{A}$ can only query a <u>poly. number</u> of <u>indep.</u> samples

→ a, e must be fresh

→ a is read from the CRS

Helium: Modelling the Non-Monolithic Execution

A non-monolithic, adaptive execution requires a random-access CRS



Game $\mathscr{G}_b$

$\mathscr{C}$

```
s, crs ← session(sid)

if b == 0:
    a ← crs(crs, sig)
    h ← sig.GenShare(s, a, $\mathcal{P}_i$)
else:
    h ← U(R)
```

$(\mathbf{sid}, \mathbf{sig_1}, \mathcal{P}_1)$

h_1

$(\mathbf{sid}, \mathbf{sig_2}, \mathcal{P}_2)$

h_2

$\mathscr{A}$

b’

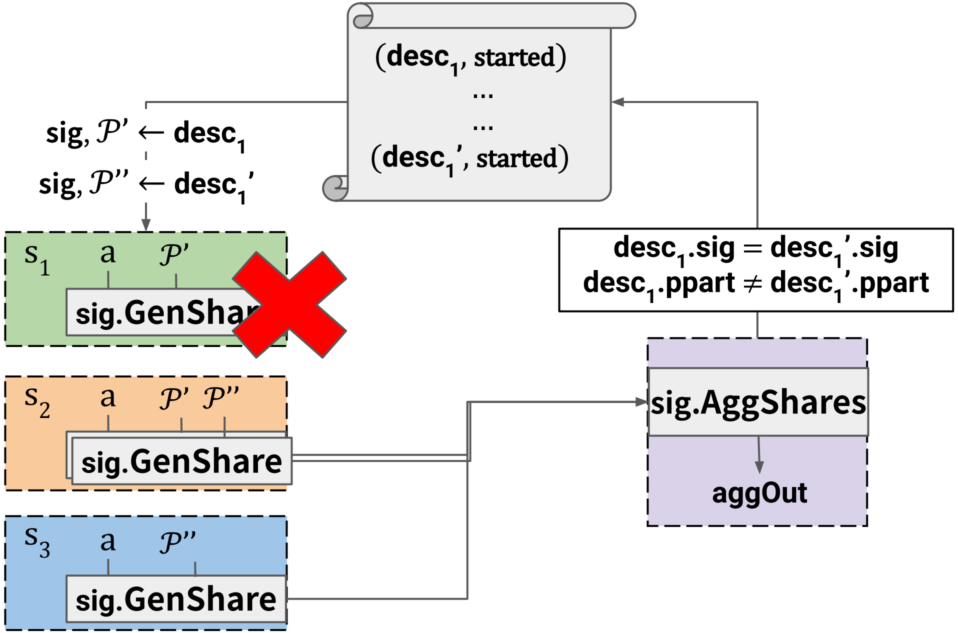
- a must be read fresh from the CRS for each sig.
- Not all parties participate to all protocols (or are even online) → Need a random-access CRS
- “Branching” the base CRS for each signature:

$$\texttt{crs}(\text{crs}, \text{sig}) \;:=\; \texttt{xof}(\text{crs}||\text{sig})$$

Unique signatures → fresh public polynomials

Helium: Helper coordination – Retries

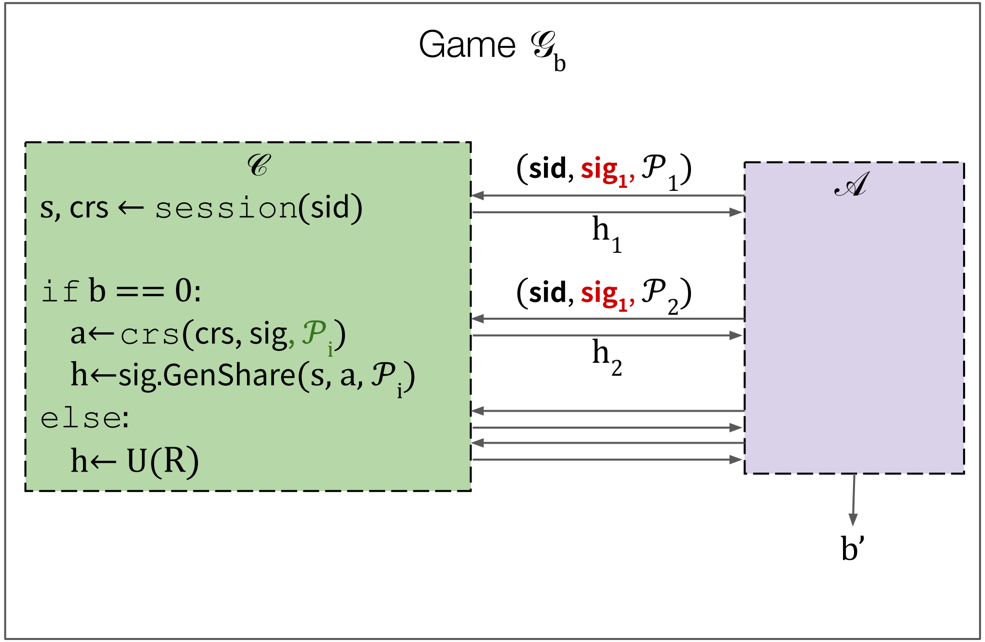
The PAT protocol semantic and non-monolithic execution provides a natural retry mechanism.



- \+ Minimal extra logic for retries
- \- Protocol failures require providing the challenger with more freedom.

Helium: Modelling the Non-Monolithic Execution

Retries allow repeated signatures with different participant sets.



Game $\mathscr{G}_b$

$\mathscr{C}$

```
s, crs ← session(sid)

if b == 0:
    a ← crs(crs, sig, $\mathcal{P}_i$)
    h ← sig.GenShare(s, a, $\mathcal{P}_i$)
else:
    h ← U(R)
```

- CRS-sampled polynomials are no longer fresh

$$(h_1, h_2, a)=(\Delta^{\mathcal{P}_1} sa + e_1 , \Delta^{\mathcal{P}_2} sa + e_2, a) \overset{c}{\not\equiv} U(R^3)$$

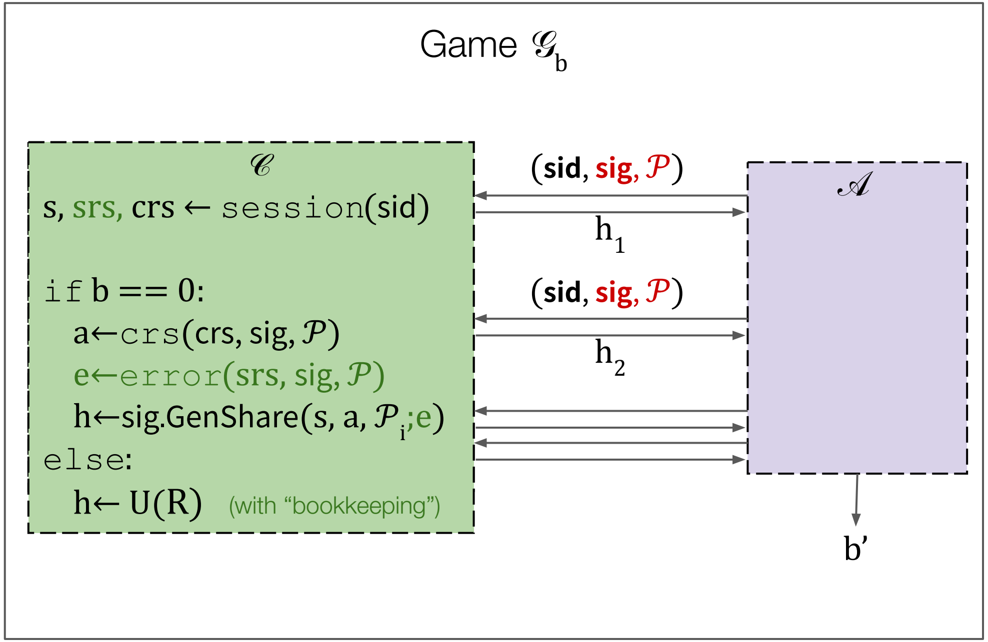
- “Branching” the base CRS for each protocols:

$$\texttt{crs}(\text{crs}, \text{sig}, \mathcal{P}) := \texttt{xof}(\text{crs} || \text{sig} || H(\mathcal{P}))$$

Unique protocol descriptor → fresh public polynomials

Helium: Modelling the Non-Monolithic Execution

Retries allow repeated signatures with <u>same participant sets</u>:



Game $\mathscr{G}_b$

$\mathscr{C}$

```
$s$, $srs$, $crs \leftarrow$ session(sid)

if $b == 0$:
    $a \leftarrow$ crs(crs, sig, $\mathcal{P}$)
    $e \leftarrow$ error(srs, sig, $\mathcal{P}$)
    $h \leftarrow$ sig.GenShare(s, a, $\mathcal{P}_i$;e)
else:
    $h \leftarrow U(R)$   (with "bookkeeping")
```

- Can happen in passive adv. setting:
    1. The network state at retry time.
    2. Stateless node restart.

$$(h_1, h_2, a) = (\Delta^{\mathcal{P}} sa + e_1 , \Delta^{\mathcal{P}} sa + e_2, a) \overset{c}{\not\approx} U(R^3)$$

- Bad solution: retry sequence numbers
  → Does not prevent case 2 failure.

- **Better solution**: resettable PAT protocols
  → By seeding the error distribution

→ Ensure $\mathscr{C}$ behaves like a random function

Practical Challenges of MHE-based MPC

0. MHE Schemes ✓
  - Construction
  - Implementation

I. Systematization ✓
  - Sessions
  - PAT protocol abstraction

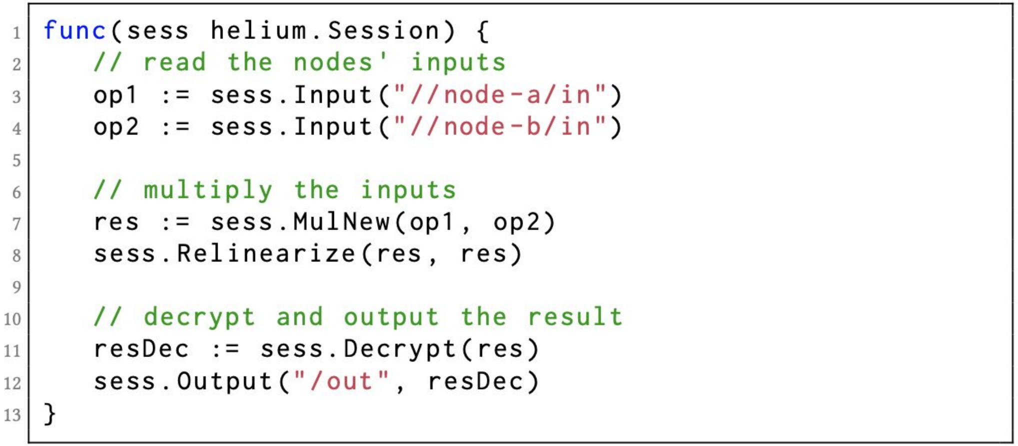
II. Lightweight clients support ✓
  - Non-Monolithic Execution
  - O(1) participant overhead

III. Churn resilience ✓
  - Retries
  - Resettability

Practical and Deployable MPC Solution

Implementation

We implemented Helium as an open-source library.

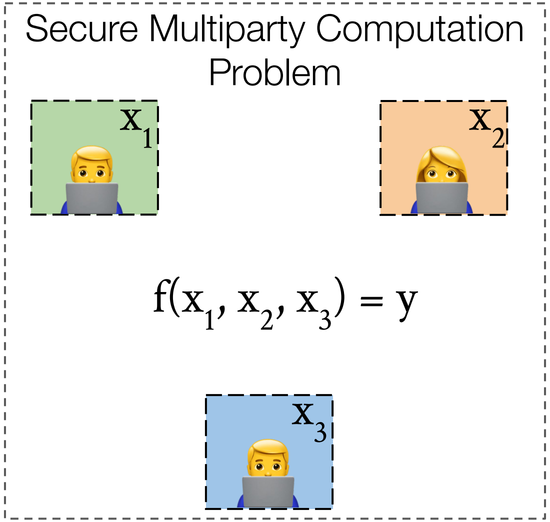


```
func(sess helium.Session) {
    // read the nodes' inputs
    op1 := sess.Input("//node-a/in")
    op2 := sess.Input("//node-b/in")

    // multiply the inputs
    res := sess.MulNew(op1, op2)
    sess.Relinearize(res, res)

    // decrypt and output the result
    resDec := sess.Decrypt(res)
    sess.Output("/out", resDec)
}
```

https://github.com/ChristianMct/helium

Conclusion: My “FHE:IDEA”

Hot take: In the short/medium term, future deployments of FHE will be solving SMPC problems.



Secure Multiparty Computation Problem

$$f(x_1, x_2, x_3) = y$$

References

[AJLT+12] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, “Multiparty computation with low communication, computation and interaction via threshold FHE,” in Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2012, pp. 483–501.

[Bea91] Beaver, Donald. "Efficient multiparty protocols using circuit randomization." *Advances in Cryptology—CRYPTO’91: Proceedings 11*. Springer Berlin Heidelberg, 1992.

[BGGJ+18] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. Rasmussen, and A. Sahai, “Threshold cryptosystems from threshold fully homomorphic encryption,” in Annual International Cryptology Conference, Springer, 2018, pp. 565–596.

[BMR90] Beaver, Donald, Micali, Silvio; Rogaway, Phillip (1990). "The round complexity of secure protocols". *Proceedings of the twenty-second annual ACM symposium on Theory of computing - STOC '90*. pp. 503–513.

[CDKS19] Chen, Hao, et al. "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference." *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019

[DSPZ12] Damgård, Ivan, et al. "Multiparty computation from somewhat homomorphic encryption." *Annual Cryptology Conference*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[GLS15] Dov Gordon, S., Feng-Hao Liu, and Elaine Shi. "Constant-round MPC with fairness and guarantee of output delivery." *Advances in Cryptology--CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II 35*. Springer Berlin Heidelberg, 2015.

[GMW87] Goldreich, Oded, Micali, Silvio; Wigderson, Avi (1987). "How to play ANY mental game". *Proceedings of the nineteenth annual ACM conference on Theory of computing - STOC '87*. pp. 218–229.

[KLSW24] Kwak, Hyesun, Dongwon Lee, Yongsoo Song, and Sameer Wagh. "A General Framework of Homomorphic Encryption for Multiple Parties with Non-interactive Key-Aggregation." In *International Conference on Applied Cryptography and Network Security*, pp. 403-430. Cham: Springer Nature Switzerland, 2024.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Crypto- graphic Techniques, French Riviera, May 30-June 3, 2010, Proceedings, Vol. 6110. Springer, 1.

[LTV12] Adriana Lòpez-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012, pages 1219–1234. ACM, 2012.

[MBH23] Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. 2023. An Efficient Threshold Access-Structure for RLWE-Based Multiparty Homomorphic Encryption. Journal of Cryptology 36 (2023).

[MCPT23] Christian Mouchet, Sylvain Chatel, Apostolos Pyrgelis, Carmela Troncoso. 2023. Helium: Scalable MPC among Lightweight Participants and under Churn, CCS2024 (To Appear)

[MTBH21] Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean- Pierre Hubaux. 2021. Multiparty Homomorphic Encryption from Ring-Learning- with-Errors. Proceedings on Privacy Enhancing Technologies 4 (2021), 291–311.

[MW16] Mukherjee, Pratyay, and Daniel Wichs. "Two round multiparty computation via multi-key FHE." *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer Berlin Heidelberg, 2016.

[Yao86] Yao, Andrew Chi-Chih (1986). "How to generate and exchange secrets". *27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*. pp. 162–167