Homomorphic Encryption in the SPDZ Protocol for MPC

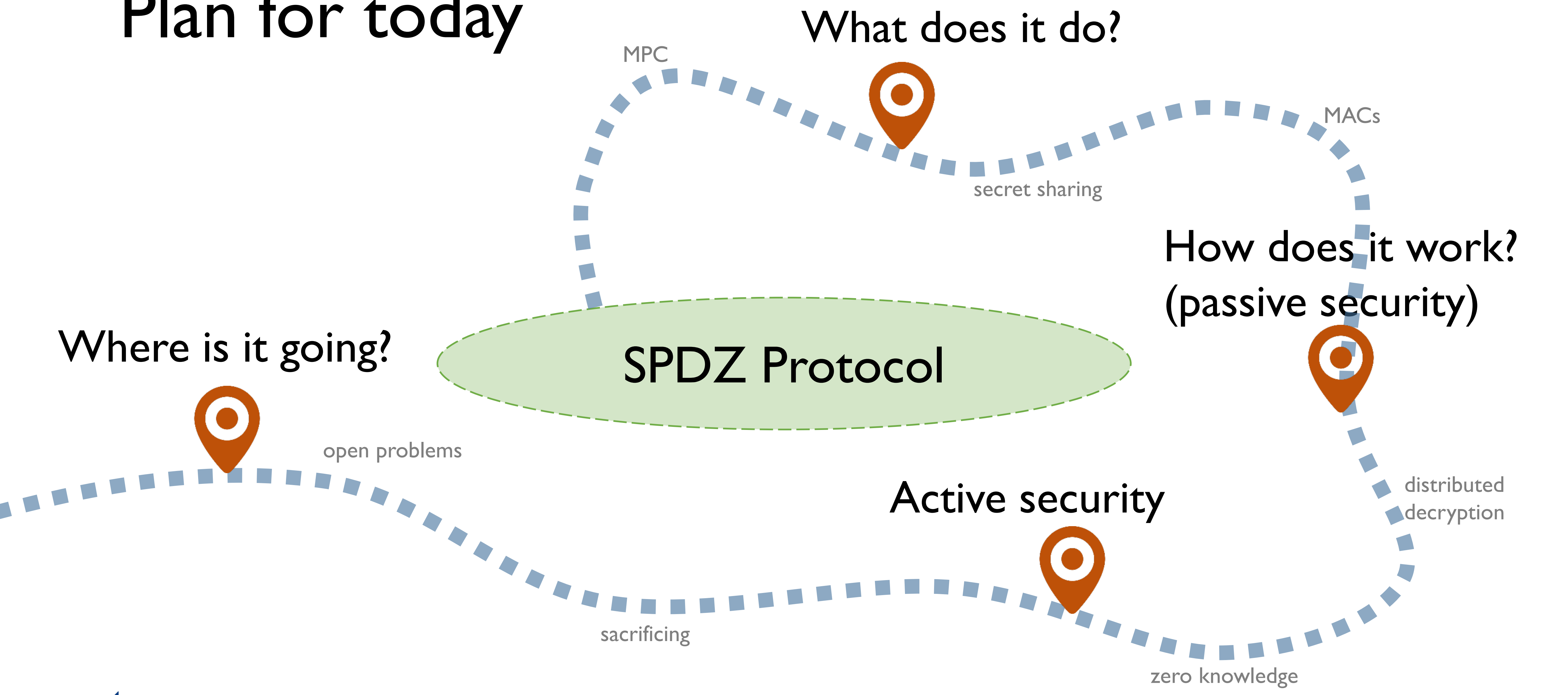
Peter Scholl

Lattices: From Theory to Practice, Simons Institute
1 May 2020

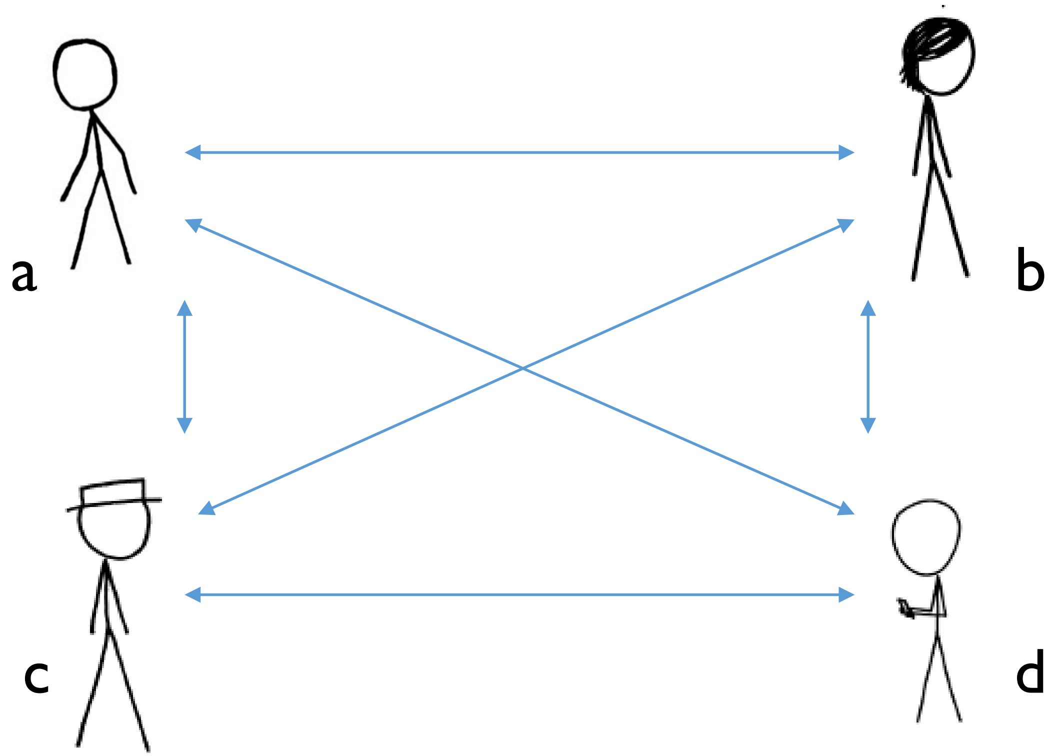
AARHUS UNIVERSITY

Thanks to: Carsten Baum and Marcel Keller

Plan for today



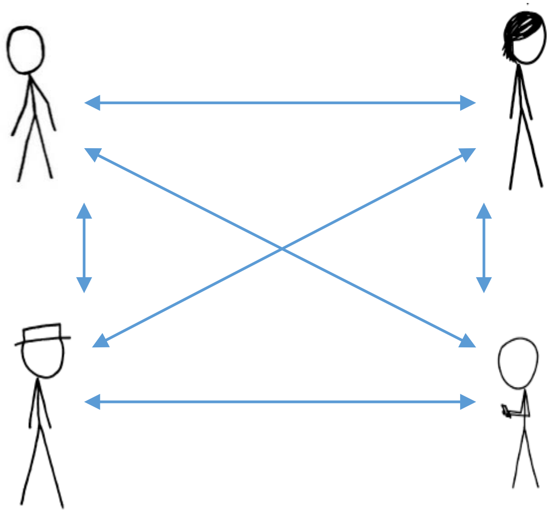
Secure Multi-Party Computation



Goal: Securely compute f(a,b,c,d)

The SPDZ setting

- Dishonest majority
  - ➢ Up to $t = n - 1$ parties may be corrupt
  - ➢ Requires computational assumptions
- Active security:
  - ➢ Security with abort
  - ➢ No fairness
- Arithmetic circuits
  - ➢ Typically in F_p, large prime p
  - ➢ Can also handle Boolean circuits, rings, …
- Originally: [**D**amgård **P**astro **S**mart **Z**akarias `12]
  - ➢ Building on ideas from [BDOZ 11]
  - ➢ Many subsequent improvements and variants [DKLPSS13], [KOS16], [KPR18], [CDESX18], [BCS19], …



MPC in the preprocessing model

- Preprocessing protocol can be done in advance
- Online phase:
  - After inputs are known
  - Lightweight: only ≈ 2x computational overhead on plaintext circuit evaluation

Additive secret sharing with MACs

[DPSZ12,DKLPSS13]

- Fixed MAC key $\alpha \leftarrow Z_p$

- Linear MAC scheme

$$MAC(x) = x \cdot \alpha \mod p$$

Secret share the MAC key, and $x, MAC(x)$:

$$\langle x \rangle, \langle \alpha \cdot x \rangle, \langle \alpha \rangle$$

Where $\langle x \rangle$ denotes $(x_1, \ldots, x_n)$, such that $x = \sum_i x_i$, and party P_i holds x_i

Reconstructed shared values

[DPSZ12,DKLPSS13]

$$\langle x \rangle, \langle \alpha \cdot x \rangle, \langle \alpha \rangle$$

where $x = \sum x_i, \quad \alpha x = \sum m_i, \ \alpha = \sum \alpha_i$

Challenge: how to check the MAC without revealing α?

- Parties open $x' = x + \Delta$
- P_i commits to $d_i = \alpha_i \cdot x' - m_i$
  - Note: $d_1 + \cdots + d_n = \alpha \cdot x' - \mathrm{MAC}(x) = \alpha \cdot \Delta$
- Open d_i and check they sum to 0

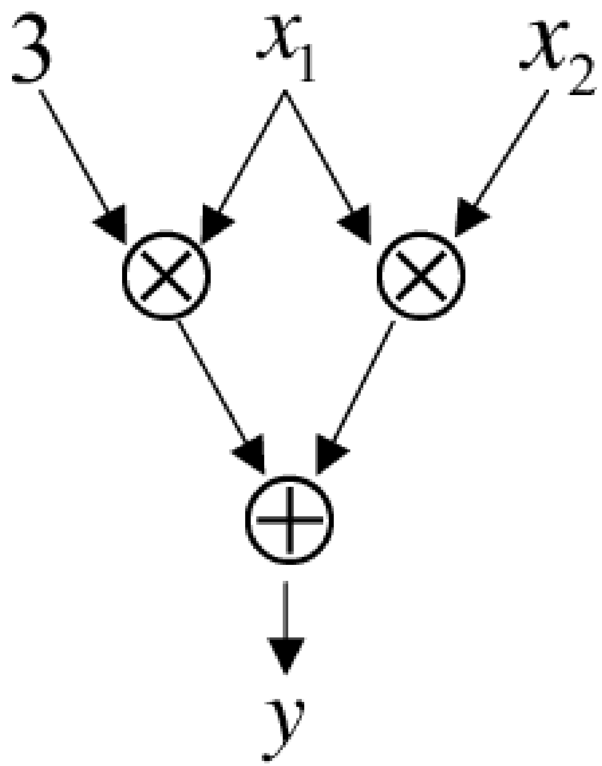
If $\Delta \neq 0$, have to guess α to pass

SPDZ online phase : securely computing arithmetic circuits

Main invariant:

- For every wire x, parties have $\langle x \rangle, \langle \alpha x \rangle$

Linear gates: local operations on shares

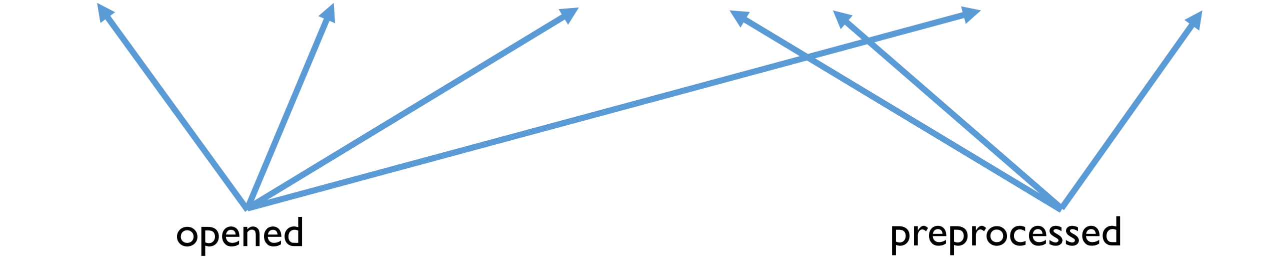


Multiplication of secret-shared values

- Have $\langle x \rangle, \langle y \rangle$, want $\langle x \cdot y \rangle$.
- Use random triple $\langle a \rangle, \langle b \rangle, \langle a \cdot b \rangle$
- Compute and open $\langle x + a \rangle, \langle y + b \rangle$

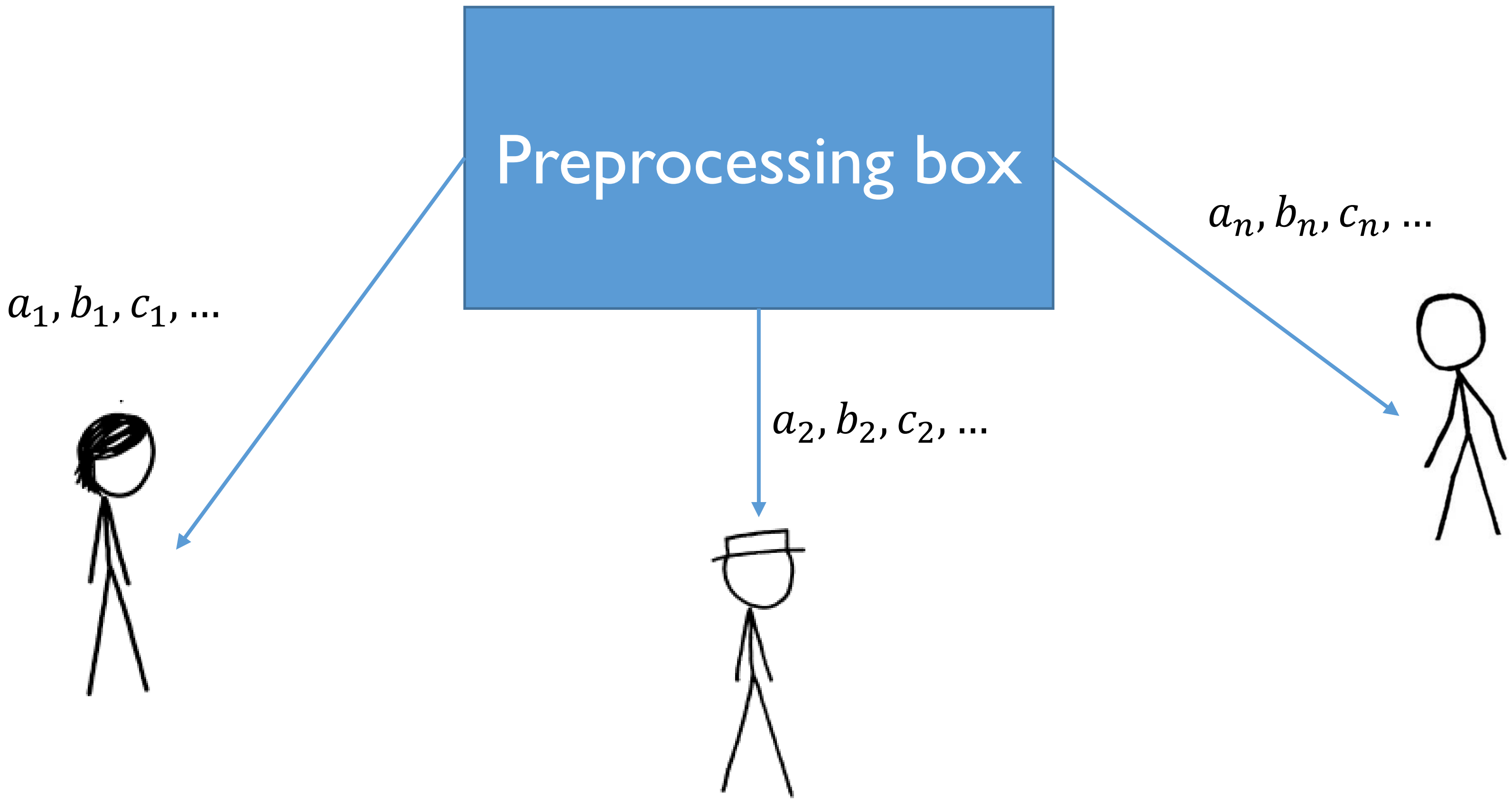
- Observe:

$$\begin{aligned} x \cdot y &= (x + a - a) \cdot (y + b - b) \\ &= (x + a) \cdot (y + b) - (x + a) \cdot b - a \cdot (y + b) + a \cdot b \end{aligned}$$

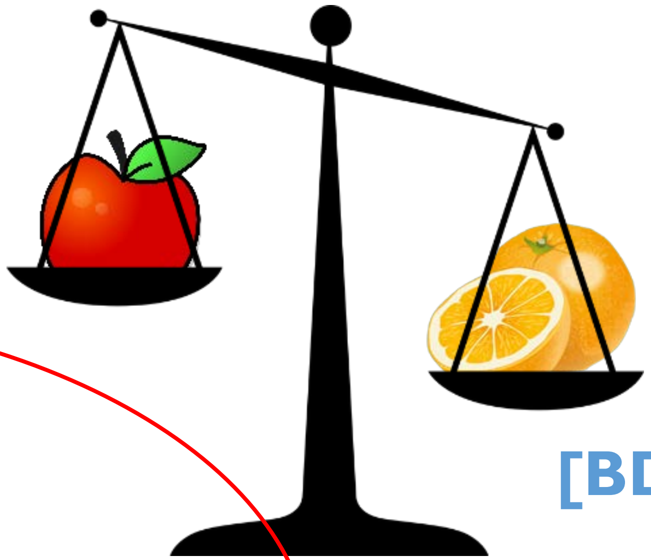


- SPDZ Basics: secret-sharing with MACs, online phase
- Passively secure preprocessing
- Active security
  - Zero knowledge proofs
  - Triple verification
- Open questions

How do we get $\langle a \rangle, \langle b \rangle, \langle a \cdot b \rangle$?



Triple generation: two main approaches



SPDZ-style

- Depth-1 HE
- Communication via broadcast
- Scales better with n parties

[BDOZ11]-style

- Linearly HE
- Pairwise communication channels
- Can be faster for 2 parties

Threshold homomorphic encryption

- Scheme $(KeyGen, Enc, DistDec)$, plaintext space Z_p.

Write $[a] := Enc_{pk}(a)$

- Homomorphism: $O(n)$ additions, 1 multiplication

- *KeyGen* setup:
  - Common pk, additive shares $\langle sk \rangle$

Not today

- Distributed decryption protocol:
  - $DistDec([m]) \rightarrow m$

Instantiating threshold homomorphic encryption

[BV11, BGV12]

Parameters: $R = Z[X]/(X^N + 1)$, N is a power of two.
Modulus $q > p$. “Small” distributions χ_{sk}, χ_{err}.
Plaintext space: $R_p \cong Z_p^N$ (via CRT)

$KeyGen$:

- $a \leftarrow R_q, s \leftarrow \chi_{sk}, e \leftarrow \chi_{err}, b = as + pe$
- $\boldsymbol{pk} = (b, -a), sk = (s)$

$Enc(\boldsymbol{pk}, m), (m \in R_p)$:

- $u \leftarrow \chi_{sk}, e_0, e_1 \leftarrow \chi_{err}$
- $\boldsymbol{c} = (c_0, c_1) = u \cdot \boldsymbol{pk} + p \cdot (e_0, e_1) + (m, 0)$

$Dec(sk, \boldsymbol{c})$, using:

- $c_0 + c_1 \cdot s = p \cdot e' + m$

Multiplicative homomorphism:

- View $\boldsymbol{c}$ as polynomial:
  $$c_0 + c_1(x)$$
- Decrypt with $c(s)$
- Multiply two polynomials ⇒ multiply ciphertexts!
  - Decryption requires s^2

Distributed decryption protocol

- Parties have (c_0, c_1) and shared $\langle s \rangle$
  - ➢Want to open: $\langle c_0 + c_1 \cdot s \rangle$
- **Problem:** $c_0 + c_1 \cdot s = p \cdot e' + m$
  - ➢Noise e' depends on the secret key!
- **Solution:** noise drowning
  - ➢Open

$$\langle c_0 + c_1 \cdot s \rangle + p \cdot \langle \tilde{e} \rangle$$

Superpolynomially larger than e'
e.g. $|\tilde{e}| \approx 2^{\kappa} \cdot |e'|$

Passive triple generation: basic protocol

- P_i samples a_i, b_i, c_i', broadcasts $[a_i], [b_i], [c_i']$
- All parties:
  - Compute $[a] = \sum_i [a_i], [b] = \sum_i [b_i]$ $[c'] = \sum_i [c_i']$
  - Compute $[\Delta] = Mult([a], [b]) - [c']$
  - $\Delta = DistDec([\Delta])$
- P_1 outputs $a_1, b_1, c_1' + \Delta$, P_i outputs a_i, b_i, c_i' $(i > 1)$

Adding MACs: essentially the same procedure

Directly gives $\langle a \rangle, \langle b \rangle, \langle a \cdot b \rangle$

- SPDZ Basics: secret-sharing with MACs, multiplication triples
- Passively secure SPDZ
- Active security
  - Zero knowledge proofs
  - Triple verification
- Open questions

Active security in two steps

- **I:** zero knowledge proof of plaintext knowledge
  - ➢ Ensure ciphertexts are correctly generated
  - ➢ Whenever P_i sends $[a_i]$, prove knowledge of a_i and randomness
- **II:** triple verification
  - ➢ Even with ZK proofs, may be additive errors in $\langle c \rangle$, due to $DistDec$
  - ➢ "sacrifice" one triple, to check another (soundness $1/p$)

Zero knowledge proofs in SPDZ

- Given ciphertext

$$\begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} -b \\ a \end{pmatrix} \cdot u + \begin{pmatrix} e_0 \\ e_1 \end{pmatrix} + \begin{pmatrix} m \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} -b & 1 & 0 & 1 \\ a & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} u \\ e_0 \\ e_1 \\ m \end{pmatrix}$$

- Prove knowledge of short pre-image satisfying linear relation

Proving knowledge of short preimages

$$\boldsymbol{c} = \boldsymbol{M} \cdot \boldsymbol{r}$$

Standard Σ-protocol:

$$\boldsymbol{c}' = \boldsymbol{M} \cdot \boldsymbol{r}'$$

Soundness ½

$$e \leftarrow \{0,1\}$$

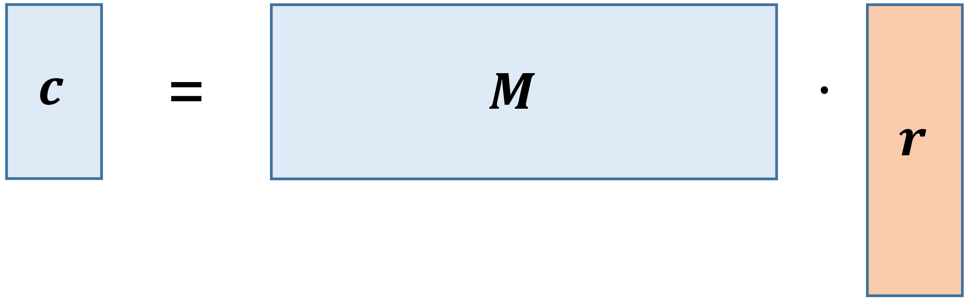
$$\boldsymbol{z} = \boldsymbol{r}' + e \cdot \boldsymbol{r}$$

Can leak on $\boldsymbol{r}$

Two options: (a) rejection sampling, or (b) noise drowning

Introduces large **soundness slack**, need bigger q

Proving knowledge of short preimages



Optimizations:

- Larger challenge space $\{X^i\}_i$ [BCS19]
  - ➢ Reduces # repetitions
  - ➢ Only proves that $2r$ is short
- Amortization
  - ➢ Batch many proofs together
  - ➢ Additive overhead of $O(\kappa)$ ciphertexts, instead of multiplicative

Variations on the basic SPDZ protocol

- [CKRRSW20]
  - ➢ Depth-2 instead of depth-1
  - ➢ Scale-invariant HE instead of BGV
  - ➢ Matrix triples via HE automorphisms

- Local distributed decryption (2 parties only)
  - ➢ "Local rounding" of $\langle c_0 + c_1 s \rangle$ gives shared $\langle m \rangle$
  - ➢ From homomorphic secret sharing [DHRW16, BKS19]

- Key switching, modulus switching [DPSZ 12]
  - ➢ Can reduce overhead of soundness slack [KPR18]

- SPDZ Basics: secret-sharing with MACs, multiplication triples
- Passively secure SPDZ and variants
- Active security
  - Zero knowledge proofs
  - Triple verification
- Open questions

Where can we hope to do better?

- **HE parameters:** ($\log q \approx 300$-600 bits)
  - ➢ Noise drowning in ZK proofs and distributed decryption
- **ZK proofs of plaintext knowledge:**
  - ➢ Need to run in large batches for efficiency
  - ➢ Computationally expensive (≈40%)
  - ➢ $O(n^2)$ communication complexity for n parties
    - Passive protocol can be $O(n)$

Improving zero knowledge proofs

- **Ideally:** want negligible soundness in one-shot, and tight bounds

- Possibly via proofs on committed values: [AELNS20]
  - Commit to randomness and prove shortness
  - Prove commitments satisfy linear relation given by $\boldsymbol{c}$ and $\boldsymbol{pk}$

- Questions:
  - How practical is this vs naïve methods?
  - Does it amortize well?

A step further: removing zero knowledge proofs?

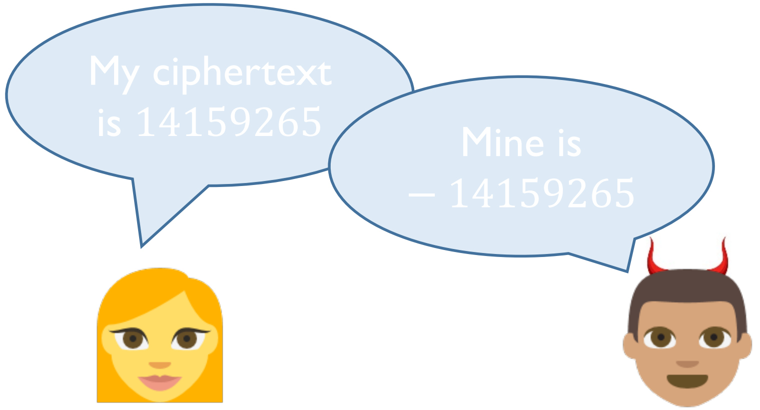
- Intuition: triple verification already guarantees correctness

- **Challenge:** ensure failure event is independent of sensitive information

- **Potential impact:** $O(n)$ complexity, better parameters, less computation

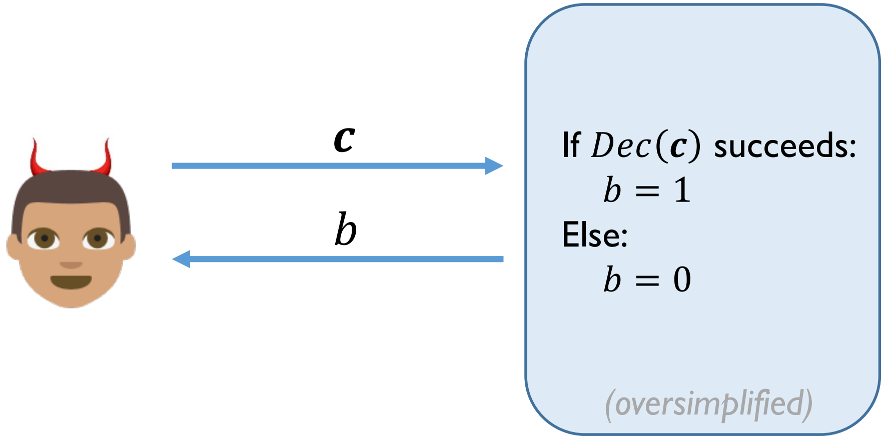
- Related: Overdrive [KPR18] removes proof of correct multiplication, security related to "linear-only encryption" assumption

A step further: removing zero knowledge proofs?



- Problem I: no independence of inputs
  - ➢ Solution: commit to ciphertexts

- Problem II: decryption failures can leak
  - ➢ In SPDZ, restricted form of leakage
  - ➢ Possible mitigations:
    - Abort/re-key on failure
    - Restrict number of executions
    - Increase sk entropy
    - Randomness extractor on triples



Noise drowning in distributed decryption

- Distributed decryption reveals values of the form:

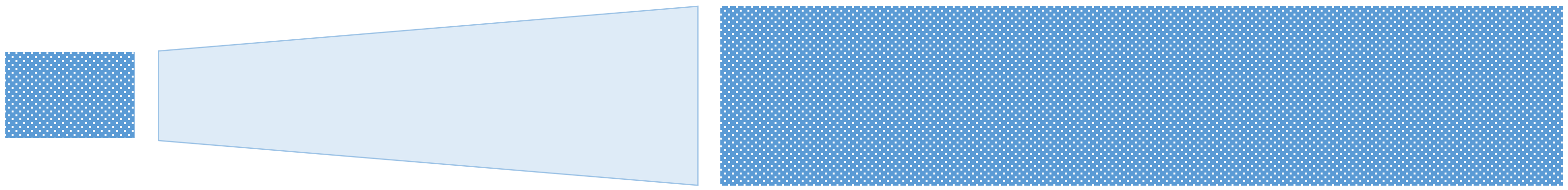
$$m + p \cdot (e + e' s) + \tilde{e}$$

Uniform in $\{\pm B \cdot 2^{\kappa}\}$

noise terms secret key

- **Q:** Is there an approach without noise flooding?
- **Q:** What goes wrong if we reduce size of $\tilde{e}$?

Alternative approach: non-interactive triple generation



- **Goal:** locally expand short seeds into large batch of triples

- [BCGIK**S**20]: candidate construction from low-noise ring-LPN in $Z_p[x]/(x^N+1)$
  - $+$ good concrete efficiency
  - $-$ Still requires many SPDZ triples to setup seeds
  - $-$ Assumption less studied when x^N+1 splits completely

Conclusion

- SPDZ Protocol
  - ➢Currently, most practical approach to dishonest majority MPC

- Lattices in SPDZ
  - ➢Low-depth SHE, large parameters
  - ➢Heavily reliant on ZK proofs of plaintext knowledge
  - ➢Noise drowning in distributed decryption

} room for improvement

References

Access **yyyy/zzz** at https://ia.cr/yyyy/zzz

- [AELNS20] *New Techniques for Practical Lattice-Based Zero-Knowledge* – Thomas Attema, Muhammed Esgin, Vadim Lyubashevsky, Khanh Ngoc, Gregor Seiler. *(Simons Institute Workshop)*
- [BCS19] *Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ* – Baum, Cozzo, Smart. **2019/035**
- [BDOZ11] *Semi-Homomorphic Encryption and Multiparty Computation* – Bendlin, Damgard, Orlandi, Zakarias. **2011/091**
- [BCGIKS20] *Pseudorandom Correlation Generators From Ring-LPN* – Boyle, Couteau, Gilboa, Ishai, Kohl, Scholl. *(coming soon)*
- [BKS19] *Homomorphic Secret Sharing From Lattices Without FHE* – Boyle, Kohl, Scholl. **2019/129**
- [CKRRSW20] *Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning* – Chen, Kim, Razenshteyn, Rotaru, Song, Wagh. **2020/451**
- [DKLPSS13] *Practical Covertly Secure MPC for Dishonest Majority – or: Breaking the SPDZ Limits* - Damgård, Keller, Larraia, Pastro, Scholl, Smart. **2012/642**
- [DPSZ12] *Multiparty Computation from Somewhat Homomorphic Encryption* – Damgård, Pastro, Smart, Zakarias. **2011/535**
- [KPR18] *Overdrive: Making SPDZ Great Again* – Keller, Pastro, Rotaru. **2017/1230**