UNIVERSITÄT BERN

Secure Multiparty Computation: Definitions and common approaches

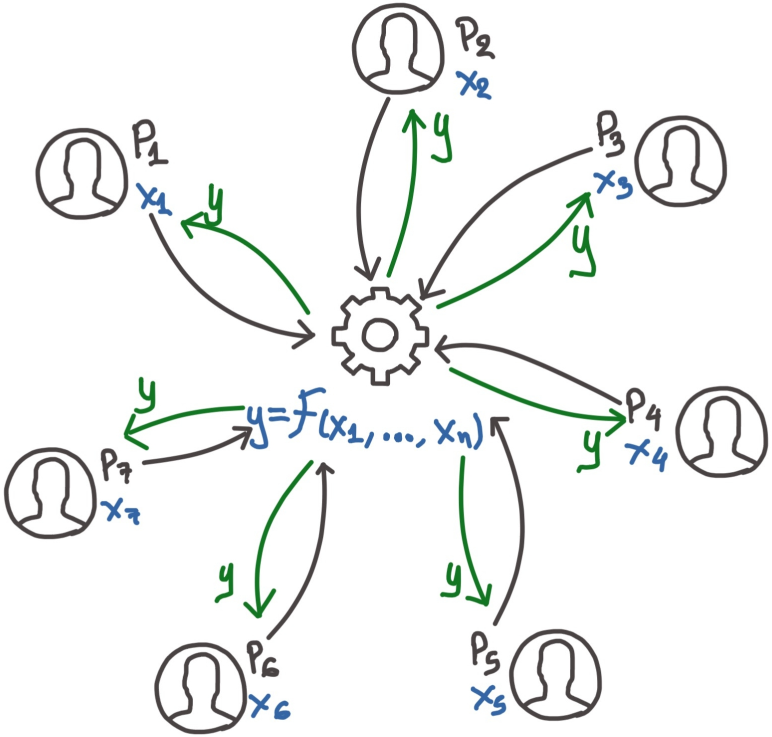
Orestis Alpos
oralpos@gmail.com

University of Bern

What is MPC

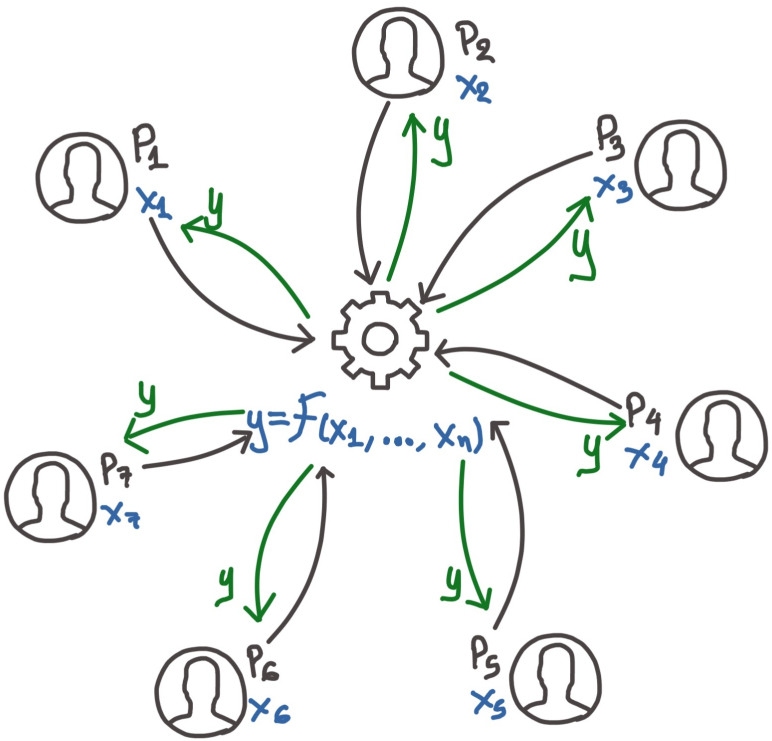
What is MPC

UNIVERSITÄT BERN



- Let $F()$ be a function of n inputs, $x_i , \ldots, x_n$
- Each party P_i holds input x_i
- Parties want to compute $F(x_1 , \ldots, x_n)$

Properties



- Privacy: Any information learnt by P_i can be derived by x_i and y
- Correctness: The output received by each player is correct

For example, in an **auction**:

- The output y is the highest bid.
- The party with highest bid will win
- All parties will know it
- Nothing should be learnt for the other bids. Of course, y reveals that all other bids are lower than that.

More properties

Not exhaustive
Each scheme satisfies different properties
Not all properties always guaranteed, there are trade-offs!

- Independence of inputs: Corrupt parties must choose inputs independent of honest parties

- Fairness: Corrupt parties receive output if and only if honest parties do

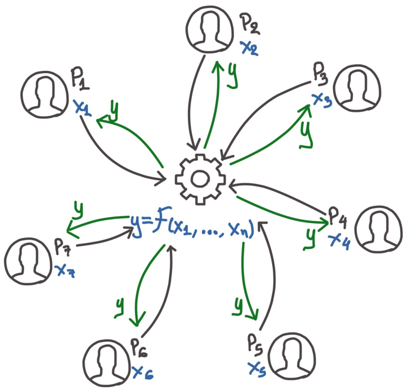
- Guaranteed output delivery (Robustness): Corrupt parties cannot prevent honest parties from receiving the output
  - Stronger than fairness

Formal definition

UNIVERSITÄT
BERN

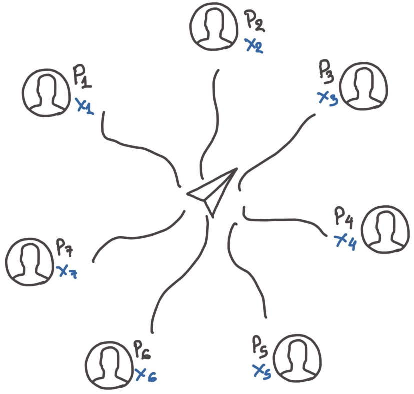
Ideal world

- An external trusted functionality does the computation
- Properties hold by definition



Real world

- No trusted party, parties run protocol
- Prove that the adversary cannot do any worse than in ideal world



Additional definition parameters

- Adversarial behavior
  - Passive (honest-but-curious, semi-honest)
  - Active (malicious)
  - Covert
- Corruption strategy
  - Static
  - Adaptive
  - Mobile (proactive security)
- Corruption thresholds
  - Honest supermajority ($t < n / 3$)
  - Honest majority ($t < n / 2$)
  - Dishonest majority (security with abort) ($t < n$)
- Type of security
  - Information theoretic
  - Computational
- Modular composition
  - Sequential (stand-alone setting)
  - Parallel (universal composability, UC)

Each scheme defined in one specific setting, for example *active adversary, static corruptions, honest majority*. There are security-efficiency trade-offs.

MPC approaches

First step

- Write *F* as an arithmetic circuit *C* of *add* and *multiply* gates.

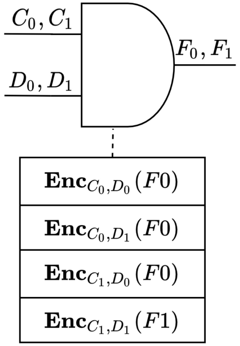
- Evaluate *C* gate by gate

- Addition and multiplication are universal over F_p

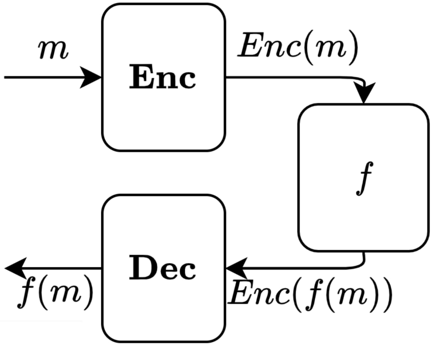
Whatever needs to be computed,
can be computed securely

Three approaches to evaluate the circuit

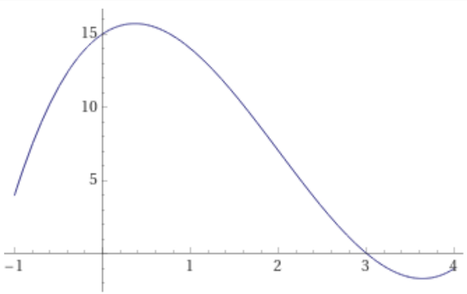
- Garbled circuits



- Homomorphic encryption

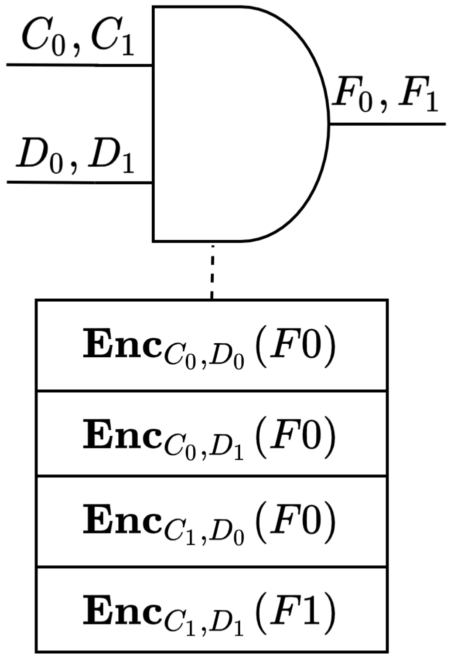


- Secret sharing

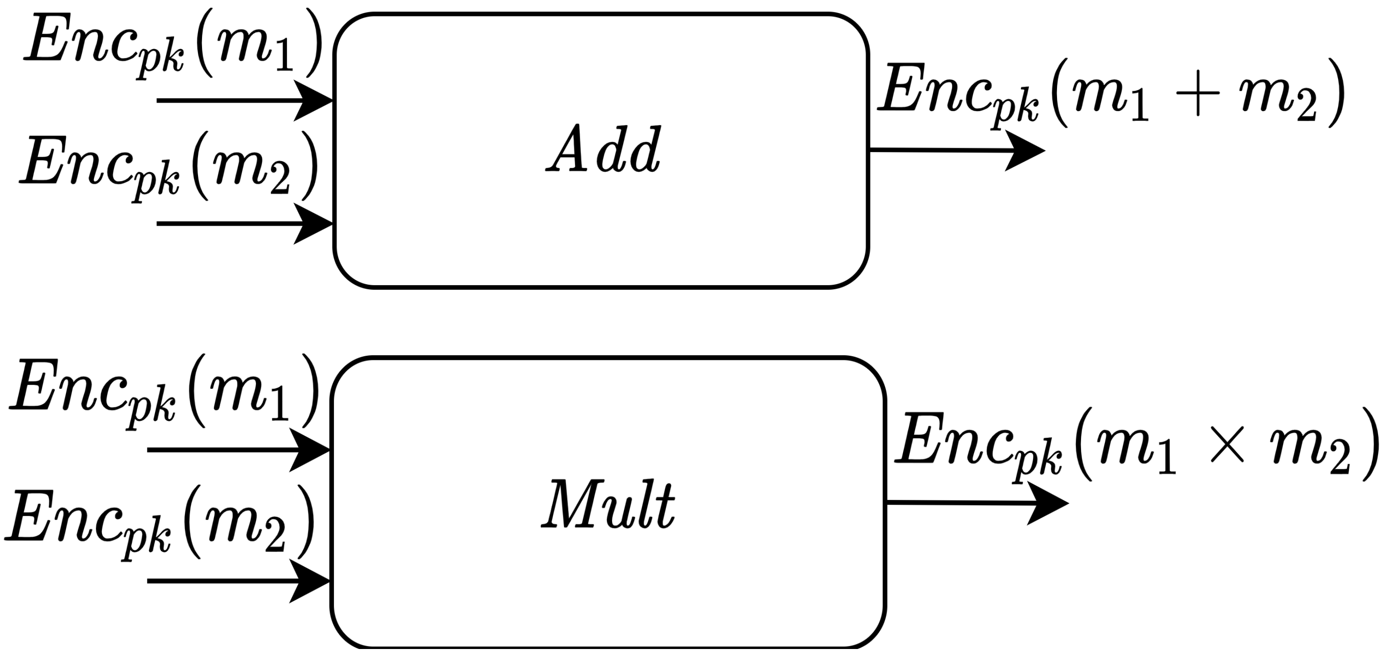


1. Garbled circuits

- Garbler and Evaluator [Yao82]
- Treat gate as matrix
  For example, AND gate has 4 rows, one for each possible input pair
- Encrypt each row, send only the keys that decrypt one input
- When output also encrypted, we can use it as input to the next gate



2. Fully homomorphic encryption

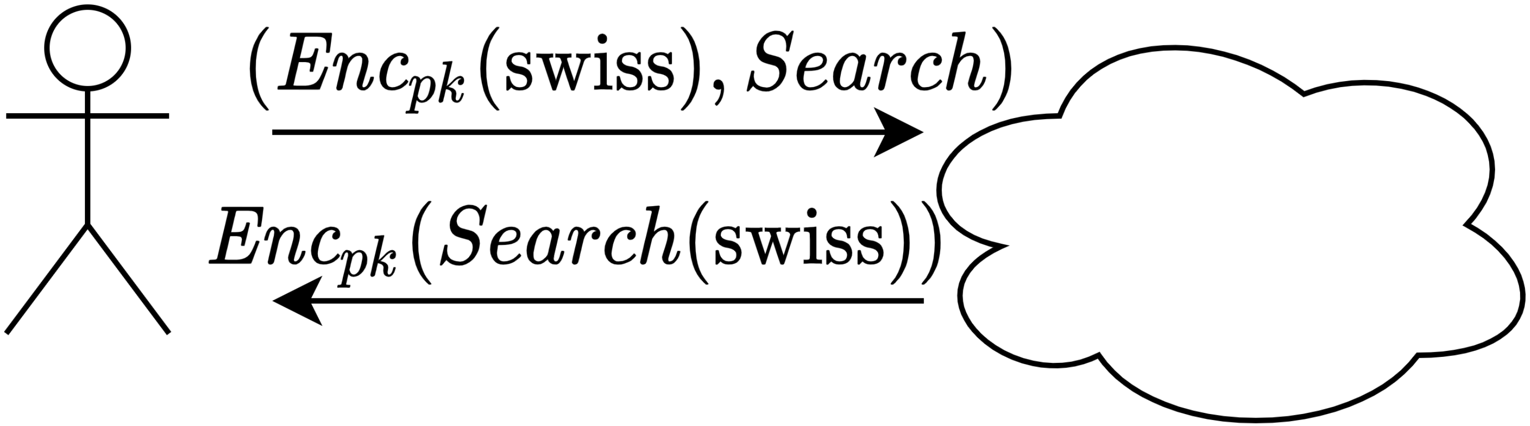


- *Add* and *Mult* are specific to the scheme

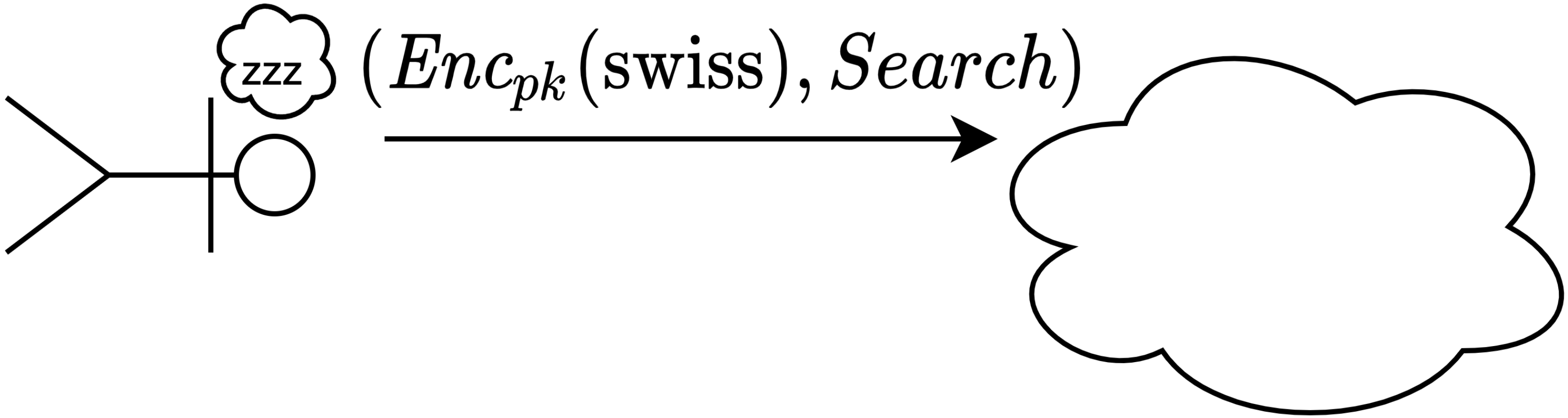
2. Fully homomorphic encryption

- For MPC, we also need partial decryption (*sk* is shared among parties)
- For passive, computational security with two rounds of communication:
- Each p_i encrypts its input and broadcasts
- Parties compute the circuit on ciphertexts
- Each p_i partially decrypts result and broadcasts
- Parties combine partial decryptions to obtain result

2. Fully homomorphic encryption is promising



2. Fully homomorphic encryption is slow



2. Fully homomorphic encryption vs (P/S) HE

- Partially homomorphic encryption

- Somewhat homomorphic encryption

- Examples:
  - ElGamal: $Enc_Y(m) = (g^r, mY^r)$
  - RSA: $Enc_e(m) = m^e$
  - Both partially homomorphic under multiplication

3. Secret sharing

UNIVERSITÄT BERN

- Share a value x among n participants, so that [Shamir79]
  - $t + 1$ can recover the secret
  - any t have no information about it

- Share
  - Degree-t random polynomial: $f(x) = k + a_1x + \dots + a_tx^t$
  - Give each party the share $s_i = f(i)$

- Reconstruct
  - $t + 1$ pairs (i, s_i) uniquely determine f
  - Lagrange interpolation

3. General secret sharing (LSSS)

- Share a value x among n parties, given access structure A, so that [CDM00]
  - An authorized set in A can recover the secret
  - Any other set has no information about it

- MSP (labeled 2D matrix M) is equivalent to LSSS

- Share
  - Random vector $\boldsymbol{r} = (k, a_1, \ldots, a_{d-1})$
  - Calculate shares as $\boldsymbol{s} = Mr$

- Reconstruct
  - For quorum A with shares $\boldsymbol{s}_A$ find recombination vector $\boldsymbol{\lambda}_A$ such that $\boldsymbol{\lambda}_A M_A = \boldsymbol{e}$
  - The value is $x = \boldsymbol{\lambda}_A \boldsymbol{s}_A$

3. Secret sharing - Add

b
UNIVERSITÄT
BERN

- Players hold sharings
  - *[x]* of *x,* made with *deg-t* polynomial
  - *[y]* of y*,* made with *deg-t* polynomial

- Obtain sharing *[x + y]* of *x* + *y* by locally adding shares

- No interaction

3. Secret sharing - Multiply

- Players hold sharings
  - *[x]* of *x,* made with *deg-t* polynomial f_1
  - *[y]* of y, made with *deg-t* polynomial f_2

- Obtain sharing *[xy]* of *xy* by locally multiplying shares

- But polynomial $g = f_1 f_2$ has degree *2t*

3. Secret sharing - Multiply

UNIVERSITÄT BERN

- Degree reduction
- Luckily, we have $2t + 1$ shares of g (we started with $t < n / 2$)
- These shares determine $g(0)$ as $g(0) = \sum_{i=1}^{2t+1} \lambda_i g(i)$
- Each p_i shares $g(i)$ with deg-t polynomial
- Parties now calculate $$[g(0)] = \sum_{i=1}^{2t+1} \lambda_i [g(i)]$$
- This is a sharing of $g(0)$ with the correct degree

3. Secret sharing - Multiply

UNIVERSITÄT BERN

- Similar idea for LSSS (Maurer)

- Requires the exchange of n^2 elements (each party send n elements)

3. Secret sharing - Multiply with Beaver trick

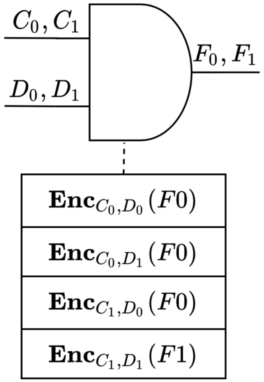
UNIVERSITÄT BERN

[Beaver91]

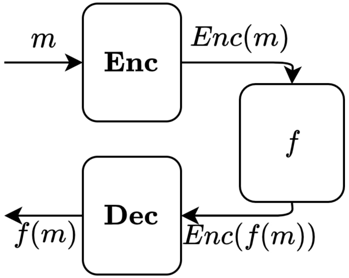
- Assume *[a], [b], [c]*, with *ab* = *c* and *a,b,c* unknown, are available
- Parties open *[ε]* = *[x]* – *[a]*. Reconstruct *ε*
- Parties open *[δ]* = *[y]* – *[b]*. Reconstruct *δ*
- Parties compute *[z]* = *[c]* + *ε[b]* + *δ[a]* + *εδ* locally
- Now *2n* elements are exchanged (each party send *2* elements)

Three approaches to evaluate the circuit - Summary

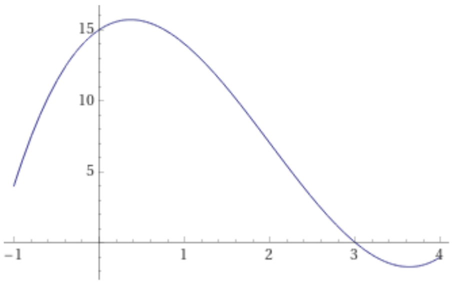
- Garbled circuits
  - 2PC
  - Low communication complexity
  - Practical and efficient for Boolean operations
  - Large circuit size for arithmetic operations



- Homomorphic encryption
  - Low communication complexity
  - Slow (computationally expensive operations)



- Secret sharing
  - No computationally expensive PK operations
  - High communication complexity
  - Number of rounds depends on multiplicative depth



Combine the three approaches: The preprocessing model

UNIVERSITÄT BERN

- Very fast online phase [DPSZ12]
  - Information theoretic primitives
  - No PK
  - Assume everything is given

- We saw how parties can add and multiply values, given sharings + Beaver triples

- Slow offline phase
  - Create everything for online phase
  - Heavy HE
  - Does not depend on circuit *C*
  - (it is not really offline)

- We saw how parties can create sharings (Beaver triples is similar)

From passive to active security

From passive to active security

- Adversary can send false shares

- We need a way to verify

- One solution: Verifiable secret sharing (Commitments)
  - Information-theoretic
  - Computational

Don’t slow me down!

From passive to active security

- Sacrifice security properties to gain efficiency
- Dishonest majority, security with abort
- We can detect cheating, not correct it

Thank you!

UNIVERSITÄT
BERN

References:

[Yao82] DBLP:conf/focs/Yao82b
[Beaver91] DBLP:conf/crypto/Beaver91a
[CDM00] DBLP:conf/eurocrypt/CramerDM00
[DPSZ12] DBLP:conf/crypto/DamgardPSZ12

Orestis Alpos
oralpos@gmail.com
orestisalpos.github.io