Introduction to Secure Multiparty Computation and SPDZ Protocol

Emmanuela Orsini

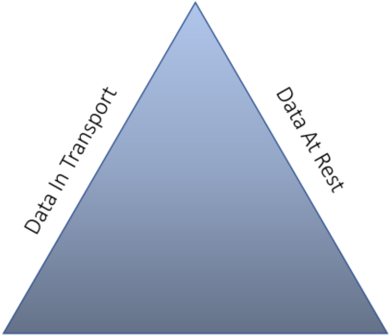
February 28th, 2023

Roadmap

1. Short Introduction to Secure MPC
2. Honest-majority LSSS-MPC
3. Dishonest-majority LSSS-MPC: The SPDZ protocol
4. 2-party Yao garbled circuit

Modern cryptography

TLS/SSL
IPSec



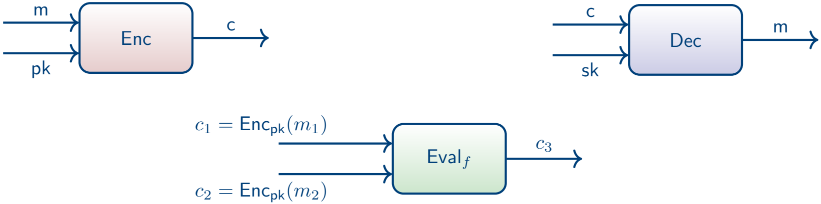
Hard disk encryption
Database encryption
HSM key storage

Data During Computation
?????????????????????????????????????

Modern cryptography

COED - Fully homomorphic encryption

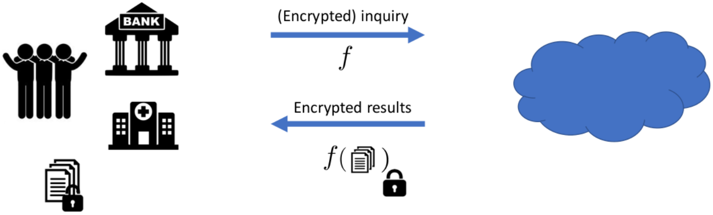
Homomorphic encryption scheme allows computation on ciphertexts.
It support three (main) operations



$$\mathsf{Dec}_{\mathsf{sk}}\big(\mathsf{Eval}_f(\mathsf{ek}, c_1, c_2)\big) = f(m_1, m_2)$$

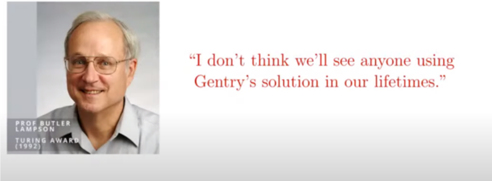
COED - Fully homomorphic encryption

In FHE the parties encrypt their data, a server computes the function in the encrypted domain, a designated party gets the output

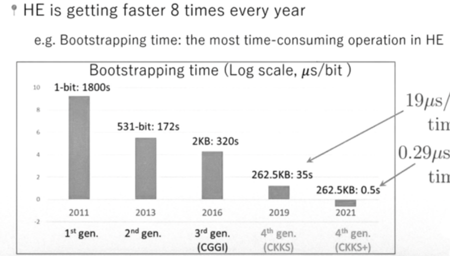


- Still rather slow in computation
- Relatively cheap in communication
- Only possible (currently) for simple functions

FHE - Recent developments



- Still slow in computation
- Relatively cheap in communication
- Only possible (currently) for simple functions



$19\mu s$/bit bootstrapping time! (amortized)

$0.29\mu s$/bit bootstrapping time! (amortized)

COED - Secure multiparty computation

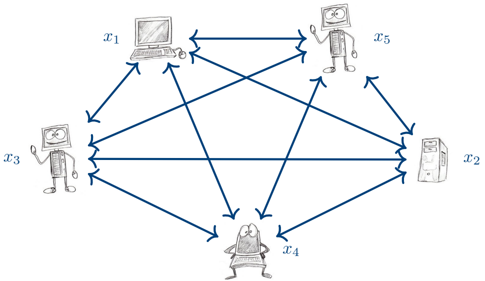
- While FHE allows computation to be performed on encrypted data held on a single server, MPC allows computation on data that is split across multiple servers

- MPC is well researched subfield of cryptography

    - Research began in the late 1980s
    - Thousands of research papers
    - MPC is now a very active applied area of research

COED - Secure multiparty computation

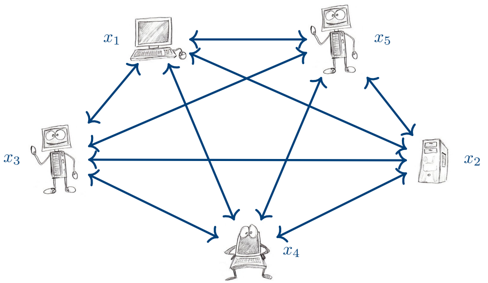
Secure function evaluation: $f(x_1, x_2, x_3, x_4, x_5)$



- **Correctness**: Parties obtain the correct output
- **Privacy**: Only the output is learned (and nothing else)

COED - Secure multiparty computation

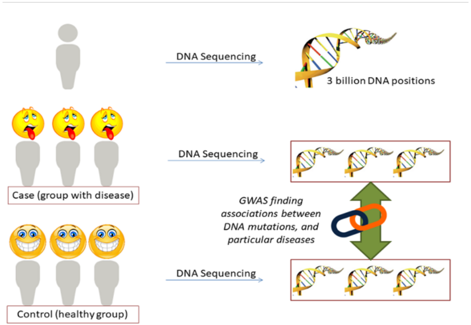
Secure function evaluation: $f(x_1, x_2, x_3, x_4, x_5)$



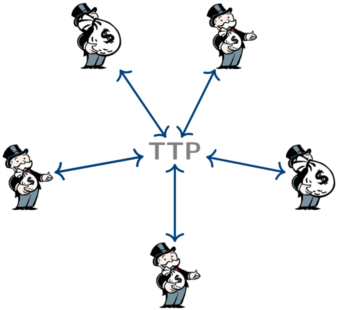
- Fast computation
- Expensive in communication
- Enables a number of applications

COED - Applications

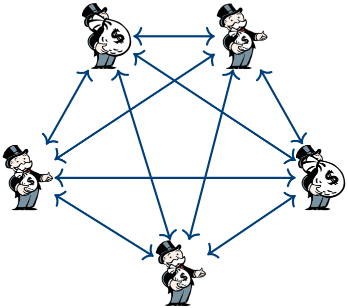
- The classic millionaires' problem
- Joint genome studies
- Studies on linked databases
- Outsourcing computation to the cloud
- Collaborative network anomaly detection
- Financial reporting in a consortium
- Securing cryptographic keys
- Statistics
- ...



Secure multiparty computation

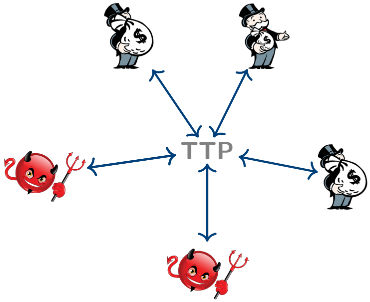


Ideal world

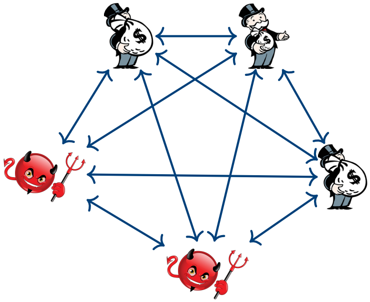


Real world

Secure multiparty computation

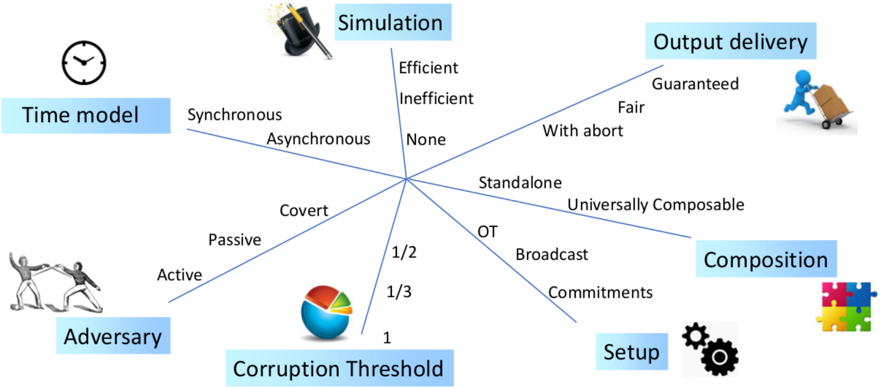


Ideal world



Real world

MPC dimensions



MPC dimensions

Computational model: Boolean/arithmetic circuit

Adversarial behaviour:

- *Passive (semi-honest)*, i.e. adversary correctly running the protocol cannot learn anything
- *Active (malicious)*, i.e. adversary arbitrary deviating from the protocol cannot learn anything

Number of corruptions: corruption threshold, honest/dishonest majority

Efficiency: round/communication/computation complexity

Security: information-theoretic, statistical, computational

MPC with a honest majority - Feasibility

Let n be the number of parties and t the number of parties that can be corrupt

- For $t < n/3$ secure multiparty protocols with guaranteed output delivery can be achieved for any function with **computational security** assuming a synchronous point-to-point network with authenticated channels and with **information-theoretic security** assuming the channels are also private.
- For $t < n/2$ secure multiparty protocols with guaranteed output delivery can be achieved for any function with **computational and information-theoretic security**, assuming that the parties also have access to a broadcast channel.

MPC with a dishonest majority - Feasibility

- For $t \geq n/2$ computationally secure multiparty protocols without guaranteed output delivery can be achieved

However, we can still have very efficient protocols

The two main paradigms for secure MPC

GMW

- Interaction at every gate (LSSS)
  - Support both arithmetic and Boolean computation
  - Very low bandwidth, good in the LAN setting
- Number of rounds depends on circuit depth

YAO

- Garbled circuit
  - Better suited for Boolean circuits
  - Requires significant bandwidth, faster on slower networks, like the Internet
- Small constant number of rounds, independent of circuit depth

LSSS

Reed–Solomon Codes

Consider the set of polynomials of degree less than or equal to t over $\mathbb{F}_q$

$$\mathbb{P} = \{f_0 + f_1 \cdot X + \cdots + f_t \cdot X^t : f_i \in \mathbb{F}_q\}.$$

This defines the set of code-words in our code, equal to q^{t+1}.

The actual code words are given by

$$\mathcal{C} = \{(f(1), f(2), \ldots, f(n)) : f \in \mathbb{P}\}.$$

Think of f as the message and $c \in \mathcal{C}$ as the codeword.

- There is redundancy in this representation
- $(t+1) \cdot \log_2 q$ bits of information are represented by $n \cdot \log_2 q$ bits.

Reed–Solomon Codes

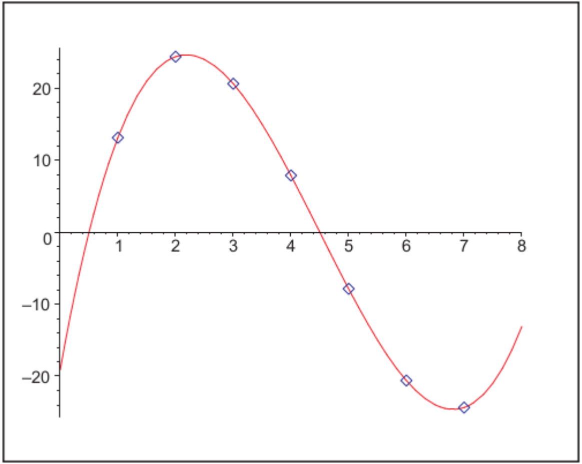


Figure: Cubic function evaluated at seven points

LSSS with an honest majority - SSS

We can use Reed–Solomon codes to define a secret sharing scheme.

A Reed–Solomon code is defined by two integers (n, t) with $t < n$.

We map secrets $s \in \mathbb{F}_q$ to the set $\mathbb{P}$ by associating a polynomial with the secret given by the constant term

For n parties we then distribute the shares as the elements of the code word

- So party i gets $s_i = f(i)$ for $1 \leq i \leq n$.

Secret reconstruction is via

$$s \leftarrow f(0) = \sum_{i=1}^{n} s_i \cdot \delta_i(0).$$

Actually any $t + 1$ parties can recover the secret.

Reed–Solomon Codes: Data Recovery

This can be done via Lagrange interpolation

Take the values

$$\delta_i(X) \leftarrow \prod_{1 \leq j \leq n, i \neq j} \left(\frac{X - j}{i - j} \right), \quad 1 \leq i \leq n.$$

Note that we have the following properties, for all i,

- $\delta_i(i) = 1$.
- $\delta_i(j) = 0$, if $i \neq j$.
- $\deg \delta_i(X) = n - 1$.

Lagrange interpolation takes the values s_i and computes

$$f(X) \leftarrow \sum_{i=1}^{n} s_i \cdot \delta_i(X).$$

Shamir secret sharing

A set of honest parties do not reveal their shares to anyone unless they want to.

A **passive adversary** controlling a subset A wants to learn the secret from the honest parties.

- This means $t \geq |A|$ to ensure privacy.
- Shamir is said to be a *threshold secret sharing scheme*
- If $|A| \leq t$ the adversary learns nothing at all about the secret.

The number of honest parties must be able to recover the secret, so we have

$$n - |A| > t \geq |A|$$

i.e.

$$|A| < n/2.$$

Shamir secret sharing

An active adversary is one which will lie about its shares

- In order for the honest parties to recover the wrong secret

To protect against this we use the error correcting property of Reed–Solomon codes.

Reed-Solomon code. The RS code is a linear $[n, t+1, n-t]$-code over $\mathbb{F}_q$.

- The code can always *detect* up to $n-t-1$ errors
- There exists an efficient decoding algorithm that *corrects* up to $\frac{n-t-1}{2}$ errors.

- If the adversary is of size $|A| \leq (n-t-1)/2$ we can recover the secret i.e.

$$t < n - 2 \cdot |A|$$

- To maintain security we require $|A| \leq t$, i.e.

$$|A| < n/3$$

Shamir secret sharing

An active adversary is one which will lie about its shares

- In order for the honest parties to recover the wrong secret

To protect against this we use the error correcting property of Reed–Solomon codes.

Reed-Solomon code. The RS code is a linear $[n, t+1, n-t]$-code over $\mathbb{F}_q$.

- The code can always *detect* up to $n-t-1$ errors
- There exists an efficient decoding algorithm that *corrects* up to $\frac{n-t-1}{2}$ errors.

- If the adversary is of size $|A| \leq (n-t-1)/2$ we can recover the secret i.e.

$$t < n - 2 \cdot |A|$$

- To maintain security we require $|A| \leq t$, i.e.

$$|A| < n/3$$

Shamir secret sharing

If we receive n shares and $t < n/2$ we know if someone is lying, and hence can abort.

- If we do not abort (we do not detect any errors), then we can recover the secret
- If we abort we do not know who cheated.

If we receive n shares and $t < n/3$ we can know if someone is lying, but we do not need to abort.

- We use the error-correction property to recover the correct shares for everyone, work out who is cheating, and recover the secret.

Shamir secret sharing

If we receive only $t+1$ shares we can reconstruct a secret, but not necessarily the correct one.

- We can also reconstruct the shares which are consistent for all parties who did not send us their shares.

In this case, if we had a lot of such openings to check,

- For each opening reconstruct the share vector
- Hash the share vector into a running hash function
- Compare the hash value with all other parties later on.

Thus if we are opening a lot of values, each party only needs to communicate with $t+1$ other parties, and not all n.

Honest-majority MPC with Shamir's secret sharing scheme

Input: The input data $(i, \langle r \rangle, r)$ is trivial:

- Party i generates an r value and distributed it to all parties
- If they distribute something invalid, then this will be detected later.
- If they distribute something not equal to r, then only they are affected in the end:
  - Either they will input an incorrect value into the MPC engine
  - Or they will not get the output they expect

Linear gate: Locally (Shamir's secret sharing is linear)

$$a \cdot \langle s \rangle + \langle r \rangle = \langle a \cdot s + r \rangle$$

Non-linear gate: ???

Schur Product

- Suppose each party i holds a vector of shares $\mathbf{s}_i$ for each secret s
  - In Shamir this a single value.

- The Schur product of two such sharings

$$(\mathbf{s}_1, \ldots, \mathbf{s}_n) \quad \text{and} \quad (\mathbf{s}'_1, \ldots, \mathbf{s}'_n)$$

  is the local tensor of each parties

$$\mathbf{s}_i \otimes \mathbf{s}'_i.$$

- $\mathbf{s}_i \otimes \mathbf{s}'_i$ is a vector of length $n \cdot (n+1)/2$

- In the case of Shamir this just means locally multiply the shares together to get one share.

Honest-majority MPC with Shamir's secret sharing scheme

- Given s and s' shared by polynomials f and f' of degree t.
- The Schur product held by party i is $f(i) \cdot f'(i)$.
- $s \cdot s'$ is shared by the polynomial $g = f \cdot f'$ of degree $2 \cdot t$
- The shares of g are $g(i) = f(i) \cdot f'(i)$.
- Since $2 \cdot t < n$ the Lagrange coefficients give us how to express $s \cdot s'$ in terms of a linear combination of the $g(i)$.

Multiplication Shamir

- We have $s_i = f(i)$ and $s'_i = f'(i)$ sharing s and s'.
- Parties form the Schur products locally $\widehat{s}_i = s_i \cdot s'_i$.

  We know, as $t < n/2$, that there exists λ_i such that

  $$s \cdot s' = \lambda_1 \cdot \widehat{s}_1 + \ldots + \lambda_n \cdot \widehat{s}_n.$$

- Parties now compute $u_i = \lambda_i \cdot \widehat{s}_i$, so we actually have a full threshold sharing of the product

  $$s \cdot s' = u_1 + \ldots + u_n.$$

Multiplication Shamir

- Party i now creates a sharing of u_i and sends the shares to each party.

  That is

  - Party i generates a polynomial $g_i(X)$ of degree t such that $g_i(0) = u_i$.
  - Party i sends party j the value $g_i(j)$.

  The resulting sharing of u_i we call $\langle u_i \rangle$.

- All parties can then compute a Shamir sharing of degree t of the product $s \cdot s'$ by computing the linear function

$$\langle s \cdot s' \rangle = \langle u_1 \rangle + \ldots + \langle u_n \rangle$$

  locally.

Passive Multiplication Protocol

Maurer's protocol gives a passive multiplication protocol:

Step 1: Form the Schur product of the parties shares.

Step 2: Express the product as a sum of the local Schur products.

Step 3: Reshare the resulting full threshold sharing.

Step 4: Recombine the resulting shares locally.

In Step 3 an adversarial party could lie, resulting in a potentially invalid sharing, or a sharing of the wrong value in the final output.

Why it is not active secure?

- Need to check that the multiplication gates are correctly evaluated

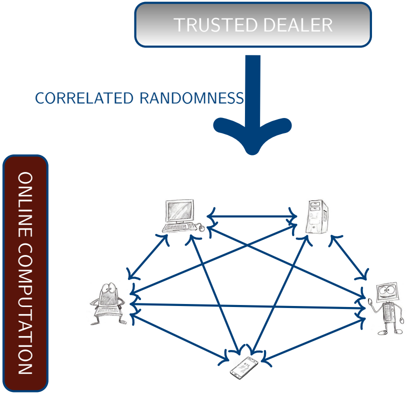
The dishonest-majority case

SPDZ setting:

- Dishonest majority: up to $n-1$ corruptions, requires computational assumption
- Active security: Security with abort, no fairness
- Arithmetic circuits: tipically $\mathbb{F}_p$, with large p, but can also handle Boolean circuits, rings etc

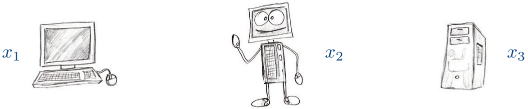
- What does 'SPDZ' stand for? [Damgärd, Pastro, Smart, Zakarias '12], there are many subsequent works with improvements and variants

MPC with preprocessing



LSSS MPC - Notation

- Every secret values $x \in \mathbb{F}$ in the computation is secret-shared among the parties.
- We consider an **additive-secret sharing scheme**



such that $x = \sum_i x_i$ and party P_i holds the share x_i.

- $\langle x \rangle$-representation
- Note the values x is unknown to the parties
- To reconstruct the value x **all** the shares are needed

LSSS - Linear computation

- The scheme is linear, so linear operations are local

$$\langle x \rangle + \langle y \rangle = \langle x + y \rangle$$

$$a \cdot \langle x \rangle = \langle a \cdot x \rangle$$

- We can compute any linear function on shared values

LSSS - Multiplication

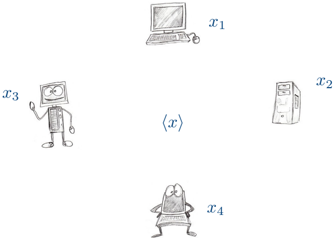
- Input multiplication gate: $\langle x \rangle$ and $\langle y \rangle$

random triple from preprocessing

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

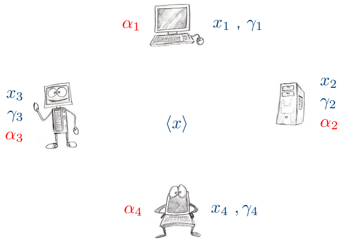
masked and opened

What if parties don't follow the protocol?

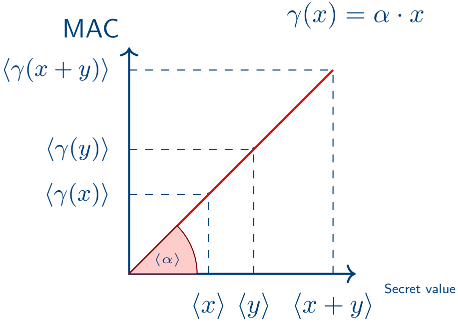


- $x = \sum_i x_i$
- $\gamma(x) = \sum_i \gamma_i(x) = \alpha \cdot x$

What if parties don't follow the protocol?



- ⋆ $x = \sum_i x_i$
- ⋆ $\gamma(x) = \sum_i \gamma_i(x) = \alpha \cdot x$



New online evaluation [DPSZ12, SPeeDZ])

- $\langle x \rangle = \{x_i\}_{i \in \mathcal{P}}$, such that $\sum_i x_i = x$
- $[x] = \{\langle x \rangle, \langle \alpha \rangle, \langle \gamma \rangle\}_{i \in \mathcal{P}}$, such that $\gamma = \alpha \cdot x$ in $\mathbb{F}$

1. Input values using $[x]$-representation
2. Evaluate the circuit gate by bate using the linearity of $[\cdot]$ and Beaver's trick for multiplication, with openings[1]
3. Do a batch check of MACs
4. If the check passes, reconstruct the output opening the output values

[1] Check MACs on opened values

Checking the openings

We need to check the MAC every time a value is opened
Check the MAC relation without revealing α

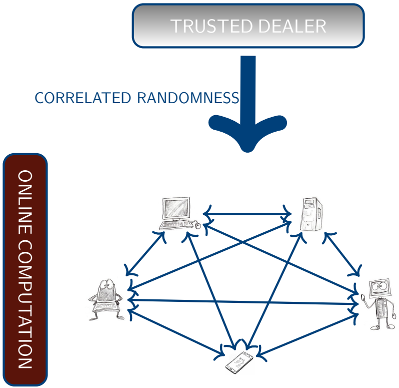
- A corrupt party P_i sends $x'_i = x_i + \delta$
- Each party reconstruct $x + \delta$

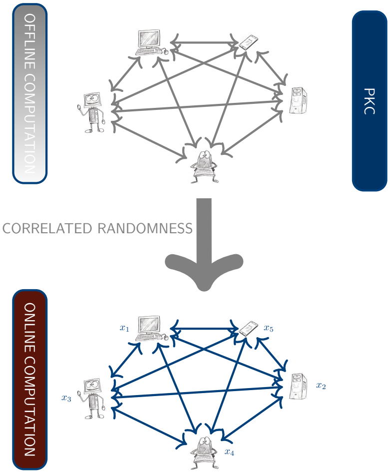
- Each party P_j, $s_j = \alpha_j \cdot (x + \delta) - \gamma_j$
- Parties compute $\sum_i s_i = \alpha(x + \delta) - \gamma(x) = \alpha \cdot \delta$

The check passes if $\sum_i s_i = 0$. If $\delta \neq 0$, the adversary has to guess α.

- Adversary wins with probability $\frac{1}{|\mathbb{F}|}$

Implementing the trusted dealer - Preprocessing





Preprocessing with homomorphic encryption

$\star$ Main goal of the preprocessing is to generate $[a], [b], [ab] = [c]$

We need a **threshold homomorphic encryption scheme**

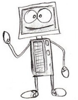
$\mathcal{E} = (\mathsf{KeyGen}(\cdot), \mathsf{Enc}_{\mathsf{pk}}(\cdot), \mathsf{DistDec}_{\mathsf{sk}}(\cdot), \mathsf{Eval}_{\mathsf{pk}}(\cdot))$ such that:

1. **Homomorphic Operations**: $O(n)$ additions and 1 multiplication
2. $\mathsf{KeyGen}(1^\lambda)$ returns a public key pk and a **secret-shared private key** $\langle \mathsf{sk} \rangle$
3. A **distributed decryption protocol** such that $\mathsf{DistDec}_{\mathsf{sk}}(\mathsf{Enc}(a))$ returns either a or $\langle a \rangle$

Preprocessing with homomorphic encryption [DPSZ12]

$a_1,\ b_1$ 

$\mathsf{Enc}_{\mathsf{pk}}(a_1),\ \mathsf{Enc}_{\mathsf{pk}}(b_1)$

$a_2,\ b_2$ 

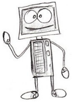
$\mathsf{Enc}_{\mathsf{pk}}(a_2),\ \mathsf{Enc}_{\mathsf{pk}}(b_2)$

 $a_3,\ b_3,$

$\mathsf{Enc}_{\mathsf{pk}}(a_3),\ \mathsf{Enc}_{\mathsf{pk}}(b_3)$

Preprocessing with homomorphic encryption [DPSZ12]

$a_1,\ b_1$ 

$a_2,\ b_2$ 

 $a_3,\ b_3,$

$\mathsf{Enc}_{\mathsf{pk}}(a_1),\ \mathsf{Enc}_{\mathsf{pk}}(b_1)$

$\mathsf{Enc}_{\mathsf{pk}}(a_2),\ \mathsf{Enc}_{\mathsf{pk}}(b_2)$

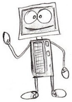
$\mathsf{Enc}_{\mathsf{pk}}(a_3),\ \mathsf{Enc}_{\mathsf{pk}}(b_3)$

ADDITIVE HOMOMORPHISM

$\mathsf{Enc}_{\mathsf{pk}}(a),\quad \mathsf{Enc}_{\mathsf{pk}}(b)$

Preprocessing with homomorphic encryption [DPSZ12]

$a_1,\ b_1$ 

 $a_2,\ b_2$

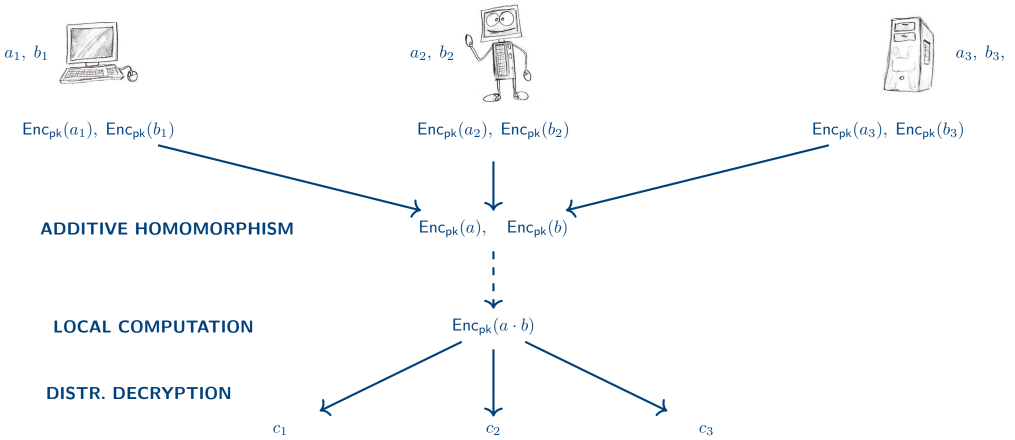
 $a_3,\ b_3,$

$\mathsf{Enc}_{\mathsf{pk}}(a_1),\ \mathsf{Enc}_{\mathsf{pk}}(b_1)$ $\quad$ $\mathsf{Enc}_{\mathsf{pk}}(a_2),\ \mathsf{Enc}_{\mathsf{pk}}(b_2)$ $\quad$ $\mathsf{Enc}_{\mathsf{pk}}(a_3),\ \mathsf{Enc}_{\mathsf{pk}}(b_3)$

ADDITIVE HOMOMORPHISM $\qquad \mathsf{Enc}_{\mathsf{pk}}(a),\quad \mathsf{Enc}_{\mathsf{pk}}(b)$

LOCAL COMPUTATION $\qquad \mathsf{Enc}_{\mathsf{pk}}(a \cdot b)$

Preprocessing with homomorphic encryption [DPSZ12]



Passive triple generation

1. P_i samples a_i, b_i, c_i' and broadcasts $Enc(a_i), Enc(b_i), Enc(c_i')$
2. All parties compute:
   - $Enc(a) = \sum_i Enc(a_i) \quad Enc(b) = \sum_i Enc(b_i) \quad Enc(c') = \sum_i Enc(c_i)$
   - $Enc(d) =$Mult$(Enc(a), Enc(b)) - Enc(c')$
   - $d =$ DistDec(d)
3. P_1 outputs $a_1, b_1, c_1' + d$ and each P_i outputs a_i, b_i, c_i', $i > 1$
4. Add MACs with the same procedure

Efficiency by batch computation [SV2011]

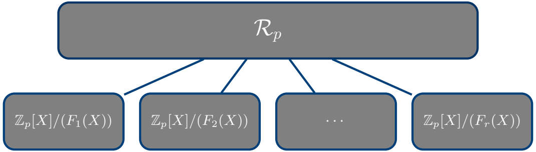
- Usually BGV (Brakerski et al. 2011) encryption scheme
- $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$, where $\deg(\Phi_m(X)) = \phi(m) = N$
- $\mathcal{R}_p = R/pR = \mathbb{Z}_p[X]/(\Phi_m(X))$, m and p coprime

$$\implies \quad \Phi_m(X) \equiv \prod_{i=1}^{r} F_i(X) \pmod{p}$$

- Each $F_i(X)$ has degree $d = \phi(m)/r = N/r$

$$\mathcal{R}_p \cong \mathbb{Z}_p[X]/(F_1(X)) \otimes \cdots \otimes \mathbb{Z}_p[X]/(F_r(X)) \cong \mathbb{F}_{p^d} \otimes \cdots \otimes \mathbb{F}_{p^d}$$

Batch computation



- We can have up to N isomorphisms

$$\psi_i : \mathbb{Z}_p[X]/F_i(X) \to \mathbb{F}_p$$

$\Rightarrow$ we can represent N plaintext elements of $\mathbb{F}_p$ as a single element in R_p.

Active security

- **Zero-knowledge proof of plaintext knowledge**
  - Ensure ciphertexts are correctly generated
  - Whenever P_i sends $Enc(a_i)$, prove knowledge of a_i and randomness

- **Triple verification**
  - Even with ZK proofs, may be additive errors in $\langle c \rangle$ due to DistDec
  - **Sacrifice** one triple, to check another

Improvements (this is not exaustive)

- ZK: Needs to run in large batches for efficiency and are computationally expensive ($\approx 40\%$)
  - Overdrive [KPR18] and TopGear [BCS19]

- Local distributed decryption: this works only for the 2-party case ("Local rounding" of $\langle c_0 + c_1 s \rangle$ gives a sharing of $\langle m \rangle$)

- Linear communication [GHM22]: this protocol is similar to SPDZ, except the step for computing a verified sum, where it is shown a mechanism to amortize the cost over multiple sums achieving linear communication when $|C| > n$.
  Match the $O(n)$ complexity of passive protocols.

Yao’s Garbled Circuits

Yao's garbled circuits

We consider the case of two party passively secure computation

We assume two parties who want to compute a function $(y_1, y_2) = f(x_1, x_2)$

- Party P_1 holds x_1 and wants to learn y_1
- Party P_2 holds x_2 and wants to learn y_2

Party P_1 does not want P_2 to learn x_1, and vice versa.

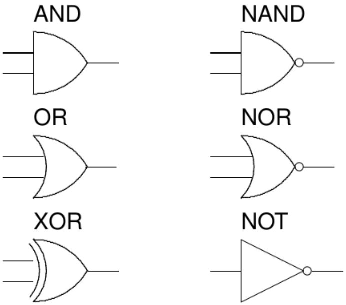
The oldest and simplest way of achieving this is via Yao's Garbled Circuits

- Which are surprisingly fast these days

We first describe the circuit construction mechanism, then we will build a protocol.

Garbled circuits: simple version

We take the function f are write it as a boolean circuit



Our aim is to "encrypt" each gate.

Wire values

- Each wire w_i in the circuit can have two values on it 0 or 1
- We assign two (symmetric) keys k_i^0 and k_i^1 to each wire value on each wire.
- Every gate G can be represented by a function with two input wires and one output wire

$$w_k = G(w_i, w_j)$$

- Note: "NOT" gates can be "folded" into the following output gate.

AND gate encryption

We go through an example of how to encrypt an AND gate

w_i	w_j	w_k
0	0	0
0	1	0
1	0	0
1	1	1

AND gate encryption

When someone evaluates the gate we want them to learn the wire key

w_i	w_j	w_k	m
0	0	0	k_k^0
0	1	0	k_k^0
1	0	0	k_k^0
1	1	1	k_k^1

AND gate encryption

Now we encrypt this message with the wire keys associated to w_i and w_j.

- We assume an IND-CCA two key symmetric encryption function $E_{\mathsf{k},\mathsf{k}'}(m)$.

w_i	w_j	w_k	c
0	0	0	$E_{\mathsf{k}_i^0,\mathsf{k}_j^0}(\mathsf{k}_k^0)$
0	1	0	$E_{\mathsf{k}_i^0,\mathsf{k}_j^1}(\mathsf{k}_k^0)$
1	0	0	$E_{\mathsf{k}_i^1,\mathsf{k}_j^0}(\mathsf{k}_k^0)$
1	1	1	$E_{\mathsf{k}_i^1,\mathsf{k}_j^1}(\mathsf{k}_k^1)$

AND gate encryption

We now create a random permutation of the table

w_i	w_j	w_k	c
1	1	1	$E_{\mathsf{k}_i^1,\mathsf{k}_j^1}(\mathsf{k}_k^1)$
0	1	0	$E_{\mathsf{k}_i^0,\mathsf{k}_j^1}(\mathsf{k}_k^0)$
0	0	0	$E_{\mathsf{k}_i^0,\mathsf{k}_j^0}(\mathsf{k}_k^0)$
1	0	0	$E_{\mathsf{k}_i^1,\mathsf{k}_j^0}(\mathsf{k}_k^0)$

AND gate encryption

We then just keep the ciphertext columns

- This table is called a **Garbled Gate**.

c
$E_{\mathsf{k}_i^1,\mathsf{k}_j^1}(\mathsf{k}_k^1)$
$E_{\mathsf{k}_i^0,\mathsf{k}_j^1}(\mathsf{k}_k^0)$
$E_{\mathsf{k}_i^0,\mathsf{k}_j^0}(\mathsf{k}_k^0)$
$E_{\mathsf{k}_i^1,\mathsf{k}_j^0}(\mathsf{k}_k^0)$

So each gate in the circuit has four ciphertexts associated to it.

Gate evaluation

- Gate evaluation occurs as follows:

- Suppose the party learns the wire label value for the zero value on wire i and the one value on wire j.
  - They learn k_i^0 and k_j^1.
  - Note they do not know wire i is zero and wire j is one.

- Using these values they can decrypt only one row of the table
  - They try all rows, but only one actually decrypts
  - This is why we needed an IND-CCA scheme, as it rejects invalid ciphertexts.

Gate evaluation

c
$E_{\mathsf{k}_i^1,\mathsf{k}_j^1}(\mathsf{k}_k^1)$
$E_{\mathsf{k}_i^0,\mathsf{k}_j^1}(\mathsf{k}_k^0)$
$E_{\mathsf{k}_i^0,\mathsf{k}_j^0}(\mathsf{k}_k^0)$
$E_{\mathsf{k}_i^1,\mathsf{k}_j^0}(\mathsf{k}_k^0)$

We can only decrypt the second row.

Hence, we learn k_k^0, but we have no idea it corresponds to the zero value on the output wire.

Garbled circuit

Given a function

$$y = F(x)$$

expressed as a boolean circuit for F the entire garbled circuit is the following values

- The garbled table for every gate in F.
- The "wire label table" for every possible input bit
- The "wire label table" for every possible output bit

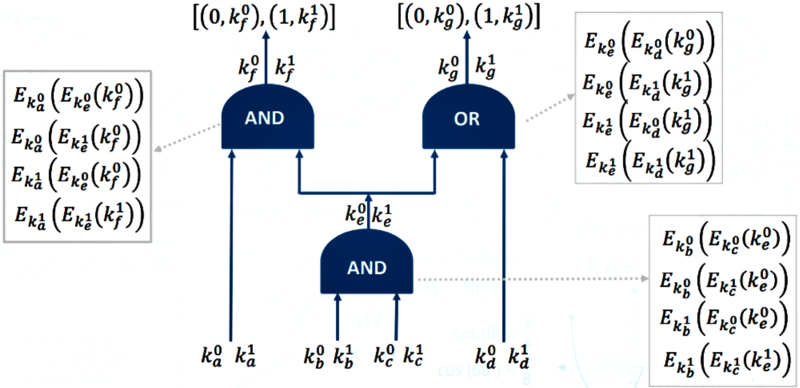
Suppose the input wires are wire numbers $0, \ldots, t$.

The input wire label table is then the values

$$(i, \mathsf{k}_i^0, \mathsf{k}_i^1).$$

Same for the output wire label table.

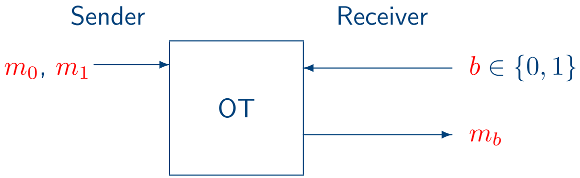
Garbled circuit



Input =0101

Oblivious transfer

Before giving Yao's MPC protocol we need another cryptographic tool



Yao's two party protocol

- We now have the building blocks for Yao's two party protocol.

- We first assume that the function is of the form

$$(y_1, y_2) = F(x_1, x_2)$$

  where
  - x_1 (resp. y_1) is party one's input (resp. output)
  - x_2 (resp. y_2) is party two's input (resp. output)

We now give a passively secure protocol (so having only a passively secure OT is OK).

Yao's two party protocol

Step 1: Party one (the circuit garbler) creates a garbled circuit for F

$$(G, (I_1, I_2), (O_1, O_2))$$

where

- G is the set of garbled gates
- I_1 is the input wire label table for party one.
- I_2 is the input wire label table for party two
- O_1 is the output wire label table for party one.
- O_2 is the output wire label table for party two.

Yao's two party protocol

Step 2:

The circuit garbler sends G to party two.

The circuit garbler also sends the values in I_1 corresponding to its input to the function

- So if the garbler wants to input bit b on wire w then it sends to party one the value k_w^b.
- This reveals nothing about the actual input, as k_w^b is a random key.

The circuit garbler also sends the table O_2 over to party two.

Yao's two party protocol

Step 3:

The parties now execute at OT protocol.

- One for each input wire w for player two.

Party two, P_2, acts as the receiver with input bit the input he wants for the function.

P_1 acts as the sender with the two "messages"

$$m_0 = \mathsf{k}_w^0 \quad \text{and} \quad m_1 = \mathsf{k}_w^1.$$

So if P_2 had input bit 0 he would learn k_w^0 but not k_w^1.

Yao's two party protocol

Step 4:

The receiver (the circuit evaluator) can now evaluate the garbled circuit to get the garbled output wire labels.

Using O_2 the receiver can now decode his output to the value y_2.

The receiver then sends the rest of the output wire labels back to P_1.

Step 5:
P_1 can decode his output value y_1 using this data and the table O_1.

More properties and variants

- Active Yao and improvements (Free-XOR, Half-gate, Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits [RR21], etc)
- Multiparty Yao ([BMR90])

- Honest majority protocols with active security with improved communication
- Different settings (Fluid MPC,)

- Different pre-processing with OT (TinyOT [NNOB12], Mascot [KOS2016] and subsequent work)
- Silent pre-processing

More properties and variants

- Active Yao and improvements (Free-XOR, Half-gate, Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits [RR21], etc)
- Multiparty Yao ([BMR90])

- Honest majority protocols with active security with improved communication
- Different settings (Fluid MPC,)

- Different pre-processing with OT (TinyOT [NNOB12], Mascot [KOS2016] and subsequent work)
- Silent pre-processing

More properties and variants

- Active Yao and improvements (Free-XOR, Half-gate, Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits [RR21], etc)
- Multiparty Yao ([BMR90])

- Honest majority protocols with active security with improved communication
- Different settings (Fluid MPC,)

- Different pre-processing with OT (TinyOT [NNOB12], Mascot [KOS2016] and subsequent work)
- Silent pre-processing