Zero-knowledge and multiparty computations

Daniel Bosk

School of Computer Science and Communication,
KTH Royal Institute of Technology, Stockholm

Department of Information and Communication Systems,
Mid Sweden University, Sundsvall

6th April 2020

1 More counter-intuitive things
- Secure multi-party computation
- Zero-knowledge proofs of knowledge

Example (Yao's Millionaires' Problem)

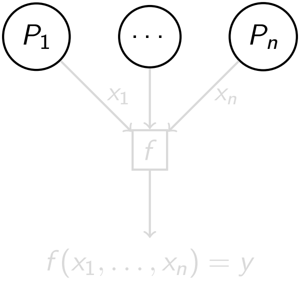
- Two millionaires meet in the street.
- They want to find out who is the richer.
- However, they don't want to reveal how many millions they each have.

Example (Yao's Millionaires' Problem)

- Two millionaires meet in the street.
- They want to find out who is the richer.
- However, they don't want to reveal how many millions they each have.

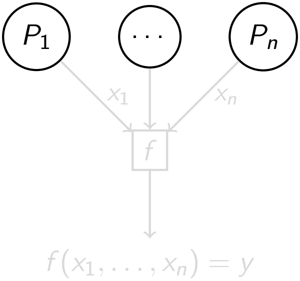
Idea

- We have n participants $P_1, \ldots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \ldots, x_n)$.



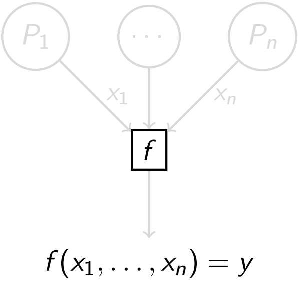
Idea

- We have n participants $P_1, \ldots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \ldots, x_n)$.



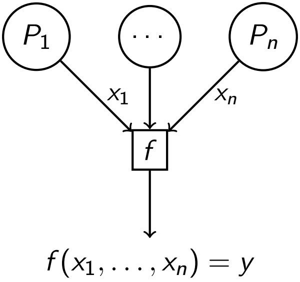
Idea

- We have n participants $P_1, \ldots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \ldots, x_n)$.



Idea

- We have n participants $P_1, \ldots, P_n$.
- Each person has a *secret* input value x_j for $1 \leq j \leq n$.
- But they desperately want to know $y = f(x_1, \ldots, x_n)$.



Example (Trivial solution)

- The n participants $P_1, \ldots, P_n$ agree on a trusted third-party (TTP).
- Each participant give their secret to the TTP.
- The TTP trusted third-party performs the computation.
- Every participant receives the result from the TTP.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j $(i \neq j)$ than what is revealed by y.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j ($i \neq j$) than what is revealed by y.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j $(i \neq j)$ than what is revealed by y.

Definition (Secure multiparty computation, MPC)

- n participants $P_1, \ldots, P_n$.
- n secret inputs $x_1, \ldots, x_n$.
- A protocol π is executed by the participants.
- At the end of the protocol each participant learns $y = f(x_1, \ldots, x_n)$.
- The participants executing π should be *equivalent* to giving $x_1, \ldots, x_n$ to a TTP T who computes $f(x_1, \ldots, x_n) = y$ and returns y to each participant.

Note

Each participant P_i learns no more about x_j $(i \neq j)$ than what is revealed by y.

- In general this problem is solved.
- We can construct protocols for arbitrary functions f.
- Efficiency varies though.
- However, there are practically feasible protocols.
- Sometimes we can use homomorphisms.
- But we can construct rather complex functions too.

- In general this problem is solved.
- We can construct protocols for arbitrary functions f.
- Efficiency varies though.
- However, there are practically feasible protocols.
- Sometimes we can use homomorphisms.
- But we can construct rather complex functions too.

- In general this problem is solved.
- We can construct protocols for arbitrary functions f.
- Efficiency varies though.
- However, there are practically feasible protocols.
- Sometimes we can use homomorphisms.
- But we can construct rather complex functions too.

Example (Sugar beet auctions[1])

- Several thousand farmers produce sugar beets.
- These are sold to the monopoly Danisco, the sugar producer.
- Contracts are allocated via a nation-wide exchange, a double auction.
- A double auction contains multiple sellers and multiple buyers.
- The purpose is to find the *market clearing price*.

[1] **MPCgoesLive**.

Example (Sugar beet auctions[1])

- Several thousand farmers produce sugar beets.
- These are sold to the monopoly Danisco, the sugar producer.
- Contracts are allocated via a nation-wide exchange, a double auction.
- A double auction contains multiple sellers and multiple buyers.
- The purpose is to find the *market clearing price*.

[1] **MPCgoesLive**.

Example (Sugar beet auctions, continued)

- Each buyer places a bid specifying how much he is willing to buy *at each potential price*.
- Each seller says how much they are willing to sell at each given price.
- The auctioneer computes the total supply and demand for each price.
- We want to find where supply equals demand.
- When done, anyone who specified non-zero for this price may trade at this price.

Example (Sugar beet auctions, continued)

- Each buyer places a bid specifying how much he is willing to buy *at each potential price*.
- Each seller says how much they are willing to sell at each given price.
- The auctioneer computes the total supply and demand for each price.
- We want to find where supply equals demand.
- When done, anyone who specified non-zero for this price may trade at this price.

Example (Sugar beet auctions, continued)

- Each buyer places a bid specifying how much he is willing to buy *at each potential price*.
- Each seller says how much they are willing to sell at each given price.
- The auctioneer computes the total supply and demand for each price.
- We want to find where supply equals demand.
- When done, anyone who specified non-zero for this price may trade at this price.

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message m, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message m, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message m, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Example

- Alice must prove her identity to Eve.
- Eve has Alice's public key, and knows it belongs to Alice.
- Alice wants to prove she is the owner of the private key belonging to the public key that Eve has.
- Eve asks Alice to sign the message m, if the signature verifies under the public key Eve believes Alice.

Gaaahh!

- Now Eve can show this message (chosen by Eve) with Alice's signature on it!
- What if Eve's chosen message was 'I give all my money to Eve'?

Idea

- Alice wants to prove that she knows the discrete logarithm x of a value g^x.
- She will do this without revealing x to Eve.

Definition (Schnorr's protocol[2])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[2] **Schnorr**.

Definition (Schnorr's protocol[2])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[2] **Schnorr**.

Definition (Schnorr's protocol[2])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[2] **Schnorr**.

Definition (Schnorr's protocol[2])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[2] **Schnorr**.

Definition (Schnorr's protocol[2])

- Prover wants to prove knowledge of x for $g^x = y$.
- Prover commits to randomness r, by sending $t = g^r$.
- Verifier replies with randomly chosen challenge c.
- After receiving c, prover replies with $s = r + cx$.
- Verifier accepts if $g^s = g^{r+cx} = g^r(g^x)^c = ty^c$.

[2] **Schnorr**.

Proof outline.

- We need to prove *completeness*: for all (most) statements the verifier will accept.
- We need to prove *soundness*: for all (most) false statements the verifier will reject.
- We need to prove that it is zero-knowledge.

Proof outline.

- We need to prove *completeness*: for all (most) statements the verifier will accept.
- We need to prove *soundness*: for all (most) false statements the verifier will reject.
- We need to prove that it is zero-knowledge.

Proof outline.

- We need to prove *completeness*: for all (most) statements the verifier will accept.
- We need to prove *soundness*: for all (most) false statements the verifier will reject.
- We need to prove that it is zero-knowledge.

□

Zero-knowledge

- Transcript for protocol: (t, c, s).
- Probability for transcript occurring: $\frac{1}{|R|} \cdot \frac{1}{|R'|}$.
- Simulate protocol: randomly choose c, randomly choose s, compute t by $g^s y^c$.
- We see that we get the same probability distribution.
- Thus the simulated transcripts are indistinguishable from the real ones.

Zero-knowledge

- Transcript for protocol: (t, c, s).
- Probability for transcript occurring: $\frac{1}{|R|} \cdot \frac{1}{|R'|}$.
- Simulate protocol: randomly choose c, randomly choose s, compute t by $g^s y^c$.
- We see that we get the same probability distribution.
- Thus the simulated transcripts are indistinguishable from the real ones.

Zero-knowledge

- Transcript for protocol: (t, c, s).
- Probability for transcript occurring: $\frac{1}{|R|} \cdot \frac{1}{|R'|}$.
- Simulate protocol: randomly choose c, randomly choose s, compute t by $g^s y^c$.
- We see that we get the same probability distribution.
- Thus the simulated transcripts are indistinguishable from the real ones.

Zero-knowledge proofs of knowledge