Overview

1. Interactive proofs and zero-knowledge protocols
2. Secure multi-party computation
[[ 3. Secure E-voting protocols ]]

Broadcast / Byzantine agreement

![Diagram showing a circle of seven nodes: P1, P2, P3, P4, P5, P6, and P7.]}
Broadcast / Byzantine agreement
Broadcast / Byzantine agreement

Diagram showing a network of nodes labeled P1, P2, P3, P4, P5, P6, and P7. The diagram illustrates the process of broadcast and Byzantine agreement with arrows indicating communication between nodes. Numbers 0 and 1 are used to represent the agreement status at different nodes.
Broadcast / Byzantine agreement
Theorem [LSP80]: Among \( n \) players, broadcast is achievable if and only if \( t < \frac{n}{3} \) players are corrupted.
Broadcast / Byzantine agreement
Generalization: Secure computation

\( \text{T} \) computes a function \( f(x_1, \ldots, x_7) \) of the inputs.

New operations of \( \text{T} \):  
- receive secret input
- keep secret state
- perform operations on state
Ideal solution: Involve a trusted party
Real solution: Simulation of trusted party
Some applications

- The millionaires’ problem
- Preventing software piracy
- On-line auctions
- E-voting
- Secure aggregation of databases
Secure MPC: Summary of known results

Adversary types:

- **passive**: plays correctly, but analyses transcript.
- **active**: cheats arbitrarily.

Types of security:

- **computational**: intractability assumptions
- **information-theoretic**: $\infty$ computing power

<table>
<thead>
<tr>
<th>type of security</th>
<th>adv. type</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>computational</td>
<td>passive</td>
<td>$t &lt; n$</td>
</tr>
<tr>
<td>computational</td>
<td>active</td>
<td>$t &lt; n/2$</td>
</tr>
<tr>
<td>information-theoretic</td>
<td>passive</td>
<td>$t &lt; n/2$</td>
</tr>
<tr>
<td>information-theoretic</td>
<td>active</td>
<td>$t &lt; n/3$</td>
</tr>
</tbody>
</table>
Formal (conventional) proof systems

- **Statements** are strings (finite sequences of symbols) over a finite alphabet, formed according to some **syntactic rules**.

  Example: $\forall u \exists v \forall x \forall y [(v > u) \land ((x > 1 \land y > 1) \rightarrow (xy \neq v \land xy \neq v + 2))]$

- The **semantics** defines which statements are **true**.

- A **proof** is a string.

- The **verification algorithm** takes as input a statement and a proof, and outputs “true” or “false”.
Formal (conventional) proof systems

- **Statements** are strings (finite sequences of symbols) over a finite alphabet, formed according to some **syntactic rules**.

  Example: $\forall u \exists v \forall x \forall y [(v > u) \land ((x > 1 \land y > 1) \rightarrow (xy \neq v \land xy \neq v + 2))]$

- The **semantics** defines which statements are **true**.

- A **proof** is a string.

- The **verification algorithm** takes as input a statement and a proof, and outputs “true” or “false”.

Requirements for a proof system:

- **Soundness**: Only true statements have proofs.

- **Completeness**: Every true statement has a proof.
Efficiency of the verification

Example: Let $n$ be a given (large) number, e.g. $n = 2^{247209813} - 1$.

- Statement: $n$ is a prime.
- Proof: A program that checks all odd numbers up to $\sqrt{n}$ as possible divisors of $n$, and outputs “true” if and only if no such divisor exists.
Efficiency of the verification

Example: Let $n$ be a given (large) number, e.g. $n = 2^{47209813} - 1$.

- Statement: $n$ is a prime.
- Proof: A program that checks all odd numbers up to $\sqrt{n}$ as possible divisors of $n$, and outputs “true” if and only if no such divisor exists.

For all practical matters, this proof is equivalent to: Believe that $n$ is prime!
Efficiency of the verification

Example: Let \( n \) be a given (large) number, e.g. \( n = 2^{47209813} - 1 \).

- **Statement:** \( n \) is a prime.
- **Proof:** A program that checks all odd numbers up to \( \sqrt{n} \) as possible divisors of \( n \), and outputs “true” if and only if no such divisor exists.

For all practical matters, this proof is equivalent to: **Believe that \( n \) is prime!**

Therefore, there is an additional requirement for a proof system:

- **Efficiency:** The verification algorithm must be efficient (e.g. run in time polynomial in the length of the input).
Efficiency of the verification

Example:  Let $n$ be a given (large) number, e.g. $n = 2^{47209813} - 1$.

- Statement: $n$ is a prime.
- Proof: A program that checks all odd numbers up to $\sqrt{n}$ as possible divisors of $n$, and outputs “true” if and only if no such divisor exists.

For all practical matters, this proof is equivalent to: **Believe that $n$ is prime!**

Therefore, there is an additional requirement for a proof system:

- **Efficiency:** The verification algorithm must be efficient (e.g. run in time polynomial in the length of the input).

Example (ctnd.): An efficient proof that $n$ is prime:

- The list of distinct prime factors $p_1, \ldots, p_k$ of $n - 1$. ($n - 1 = \prod_{i=1}^{k} p_i^{\alpha_i}$)
- Primality proofs for $p_1, \ldots, p_k$ (recursion!).
- $a$ such that $a^{n-1} \equiv 1 \pmod{n}$ and $a^{(n-1)/p_i} \not\equiv 1 \pmod{n}$ for $1 \leq i \leq k$. 
Interactive proofs

prover P \rightarrow \text{statement} \rightarrow \text{verifier V} \rightarrow \text{accept/reject}

\ldots
Interactive proofs

Motivations for interactive proofs:
Interactive proofs

Motivations for interactive proofs:

- Interactive proofs can be zero-knowledge.
Interactive proofs

Motivations for interactive proofs:

- Interactive proofs can be zero-knowledge.
- Interactive proofs are more powerful than static proofs
Interactive proofs

Motivations for interactive proofs:

- Interactive proofs can be zero-knowledge.
- Interactive proofs are more powerful than static proofs
- Applications:
  - Digital signature schemes
  - Entity authentication
  - Secure multi-party computation
Two types of interactive proofs
Two types of interactive proofs

Proofs of statements:

- The number 638634389........3427 has 3 prime factors.
- $z$ is a square modulo $n$ \((\exists x : z = x^2)\)
- The graphs $G$ and $H$ are isomorphic.
- $P=NP$
Two types of interactive proofs

Proofs of statements:

- The number 638634389........3427 has 3 prime factors.
- \( z \) is a square modulo \( n \) \( (\exists x : z = x^2) \)
- The graphs \( G \) and \( H \) are isomorphic.
- P=NP

Proofs of knowledge:

- I know the factors of the number 638634389........3427.
- I know a value \( x \) such that \( z = x^2 \pmod{n} \). (Fiat-Shamir)
- I know \( x \) such that \( z = g^x \).
- I know how to prove either P=NP or P\(\neq\)NP.
Hamiltonian cycles in a graph
The Graph Isomorphism (GI) problem

\[ \begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 & 0 \\
3 & 0 & 1 & 0 & 0 & 1 & 1 \\
4 & 1 & 1 & 0 & 0 & 1 & 0 \\
5 & 0 & 1 & 1 & 1 & 0 & 0 \\
6 & 1 & 0 & 1 & 0 & 0 & 0 
\end{array} \]

\[ \begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 \\
2 & 1 & 0 & 1 & 0 & 0 & 1 \\
3 & 0 & 1 & 0 & 0 & 1 & 1 \\
4 & 1 & 0 & 0 & 0 & 1 & 0 \\
5 & 0 & 0 & 1 & 1 & 0 & 1 \\
6 & 1 & 1 & 1 & 0 & 1 & 0 
\end{array} \]
## Zero-knowledge proof for GI: the protocol (one round)

**Prover Peggy**

- knows $\sigma$: $H = \sigma G \sigma^{-1}$
- choose a random permutation $\pi$ on the set $\{1, \ldots, n\}$
- compute $T = \pi G \pi^{-1}$

**Verifier Vic**

- choose challenge $c \in \{0, 1\}$ at random

\[ \rho = \pi \sigma^{-c} \]

\[ \text{check } T \overset{?}{=} \rho G \rho^{-1} \quad \text{if } c = 0 \]

\[ \text{check } T \overset{?}{=} \rho H \rho^{-1} \quad \text{if } c = 1 \]
Interactive proofs: Requirements
Interactive proofs: Requirements

- **Completeness**: If the statement is true [or the prover knows the claimed information], then the proof will be accepted by the verifier.
Interactive proofs: Requirements

- **Completeness:** If the statement is true [or the prover knows the claimed information], then the proof will be accepted by the verifier.

- **Soundness:** If the statement is false [or the prover does not know the claimed information], then for all prover strategies the proof will be accepted by the verifier only with negligible probability.
Interactive proofs: Requirements

- **Completeness:** If the statement is true [or the prover knows the claimed information], then the proof will be accepted by the verifier.

- **Soundness:** If the statement is false [or the prover does not know the claimed information], then for all prover strategies the proof will be accepted by the verifier only with negligible probability.

- **Zero-knowledge:** The prover leaks no information.
An int. proof for Graph Non-Isomorphism (one round)

Prover Peggy

Verifier Vic

Choose challenge $b \in \{0, 1\}$ and a permutation $\pi$ on the set $\{1, \ldots, n\}$ at random.

If $b = 0$, compute $K = \pi G \pi^{-1}$.

If $b = 1$, compute $K = \pi H \pi^{-1}$.

If $K \cong G$, let $c = 0$,
else let $c = 1$.

Accept if $c = b$
Fiat-Shamir protocol

Prover Peggy

knows $x \in \mathbb{Z}_m^*$

$k \in_R \mathbb{Z}_m^*$

$t = k^2$

$r = k \cdot x^c$

Verifier Vic

$z = x^2$

$c \in_R \{0, 1\}$

$r^2 \equiv t \cdot z^c$
Guillou-Quisquater protocol

Prover Peggy

knows \( x \in \mathbb{Z}_m^* \)

\( k \in_R \mathbb{Z}_m^* \)

\( t = k^e \)

\( r = k \cdot x^c \)

Verifier Vic

\( z = x^e \)

\( c \in_R [1, e - 1] \)

\( r^e \equiv t \cdot z^c \)
Schnorr protocol

Prover Peggy

knows $x \in \mathbb{Z}_q$

$k \in_R \mathbb{Z}_q$

t = $h^k$

$r = k + xc$

Verifier Vic

$z = h^x$

c $\in_R [0, q - 1]$

$h^r \overset{?}{=} t \cdot z^c$