Techniques in Cryptography and Cryptanalysis

Brice Minaud

e-mail: brice.minaud@ens.fr

MPRI, 2023
“Techniques in Cryptography and Cryptanalysis”: will cover (a choice of) important areas of cryptography.

- Zero-knowledge proofs
- Oblivious algorithms
- Lattices

2nd period

Teachers:

Phong Nguyễn
8 x 1.5h, 2nd period

Brice Minaud
8 x 1.5h, 1st period
Exams: 1.5h, same time slot as lectures. A few exercises.
See also website for additional material.

Interested in crypto? Have questions?
Don't hesitate to write to me and/or Phong.

Looking for an internship?
Both Phong and me are looking for interns. Again, just contact us and come discuss potential internships.

Contact: brice.minaud@ens.fr, phong.nguyen@ens.fr
What is security?

Historically, most basic goal = protecting the confidentiality of data exchanges.

Kerckhoff’s (first three) principles:
1. The system must be practically, if not mathematically, indecipherable.
2. It should not require secrecy, and it should not be a problem should it fall into enemy hands.
3. It must be possible to [...] change or modify [the key] at will.
One-Time Pad

Modern version: the algorithms are public. They are parametrized by a (secret) key.

Message space: $M \leftarrow \{0,1\}^n$  Key space: $K \leftarrow \{0,1\}^n$

Encryption($M$): $C = M \oplus K$

Decryption($C$): $M = C \oplus K$
One-Time Pad.
Message space: $M \leftarrow \{0,1\}^n$  
Key space: $K \leftarrow \{0,1\}^n$

Encryption($M$): $C = M \oplus K$

Decryption($C$): $M = C \oplus K$

**Naive security:** impossible for Eve to find $M$ from $C$.

Not great. Encryption could leak last bit of $M$ and still be secure by that definition.

We want to express that Eve learns *nothing* about $M$. 
Perfect secrecy


**Prior distribution:** distribution of $M$ known a priori to Eve.

**Posterior distribution:** distribution of $M$ known to Eve after seeing the encryption $\text{Enc}_K(M)$ of $M$ (for uniform $K$).

**Perfect secrecy:** posterior distribution = prior distribution.

Perfect secrecy, equivalent modern version, folklore, 20th century.

Let $M_0$ and $M_1$ be two arbitrary messages.

**Perfect secrecy:** $\text{Enc}_K(M_0) = \text{Enc}_K(M_1)$.

The equality is an equality of distributions. The randomness is over the uniform choice of $K$. 
**Proposition.** The One-Time Pad achieves perfect secrecy.

*Proof.* $\text{Enc}(M_0) = C$ iff $K = C \oplus M_0$.

So there is exactly one $K$ that yields each possible $C$. Since $K$ is uniform, so is $C$. Thus:

$$\text{Enc}(M_0) = \text{Unif}([0,1]^n) = \text{Enc}(M_1).$$

(Note: this would hold in any group.)

**Theorem (Shannon ’49).** If perfect secrecy holds, it must be the case that the two parties share some prior information (a key) with:

$$\text{length(key)} \geq \text{length(message)}$$

where length denotes the bit length.

So OTP is essentially the only perfectly secure scheme.
Measuring Security
Advantage

- Previous solution is infeasible in most cases.
  → we must be content with *imperfect* security.
- The relevant notion to formally express that Eve cannot learn anything is often about the *indistinguishability* of two distributions.

Roadmap of a security definition: the *adversary* is an algorithm attempting to infer secret information.

Often, this will be expressed as the adversary trying to distinguish two distributions.

**Advantage.**

Let $D_0$ and $D_1$ be two probability distributions. The advantage of an adversary $A$ (i.e. an algorithm, here with output in \{0,1\}) is:

$$
\text{Adv}_{D_0,D_1}(A) = \left| 2 \Pr_{b \in \{0,1\}} (A(D_b) = b) - 1 \right|
$$
Types of security

let $M_0$ and $M_1$ be two arbitrary messages...

**Perfect security:**

$\text{Enc}_K(M_0) = \text{Enc}_K(M_1)$ (as distributions, for uniform $K$).

*Equivalently:* $\text{Adv}^\text{Enc}_K(M_0, \text{Enc}_K(M_1))(A) = 0$, for every $A$.

**Statistical security:**

$\text{Adv}^\text{Enc}_K(M_0, \text{Enc}_K(M_1))(A)$ is negligible, for every $A$.

**Computational security:**

$\text{Adv}^\text{Enc}_K(M_0, \text{Enc}_K(M_1))(A)$ is negligible, for every efficient adversary $A$. 
Quantifying negligibility, efficiency

Security parameter, often denoted \( \lambda \): used to quantify security.

- “Asymptotic” security: used in more theoretical results. \( \lambda \) remains a variable.
- “Concrete” security: used in more practical results. Typically \( \lambda = 80, 128, \text{ or } 256. \text{ (e.g. “128-bit” security.)} \)

<table>
<thead>
<tr>
<th>Negligible (probability)</th>
<th>“Asymptotic” security</th>
<th>“Concrete” security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient (adversary)</td>
<td>( O(\lambda^{-c}) ) for all ( c )</td>
<td>usually ( \leq 2^{-\lambda/2} \text{ or } 2^{-\lambda} )</td>
</tr>
<tr>
<td></td>
<td>( \text{Poly}(\lambda) )</td>
<td>significantly less than ( 2^{\lambda} ) operations</td>
</tr>
</tbody>
</table>

Caveats: computation model (TM, RAM, circuits), “basic” operation, memory, etc.
## Concreteness of security

<table>
<thead>
<tr>
<th>Bits of security</th>
<th>Practical significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Your phone can do it, instantly.</td>
</tr>
<tr>
<td>66</td>
<td>Bitcoin hashes <em>per second</em> worldwide.</td>
</tr>
<tr>
<td>80</td>
<td>Bitcoin hashes per year worldwide.</td>
</tr>
<tr>
<td></td>
<td><em>(Some state actors could do it?)</em></td>
</tr>
<tr>
<td>128</td>
<td>Considered secure. Standard choice.</td>
</tr>
<tr>
<td></td>
<td><em>(Watch out for trade-offs, like time/data or multi-target)</em></td>
</tr>
<tr>
<td>256</td>
<td>Arguments for impossibility based on physics.</td>
</tr>
<tr>
<td></td>
<td><em>(Relevant for very long-term or quantum security.)</em></td>
</tr>
</tbody>
</table>

Types of security, again

let $M_0$ and $M_1$ be two arbitrary messages...

**Perfect security:**

$$\text{Enc}_K(M_0) = \text{Enc}_K(M_1)$$ (as distributions, for uniform $K$).

*Equivalently:* $\text{Adv}^{\text{Enc}_K(M_0), \text{Enc}_K(M_1)}(A) = 0$, for every $A$.

**Statistical security:**

$\text{Adv}^{\text{Enc}_K(M_0), \text{Enc}_K(M_1)}(A)$ is negligible, for every $A$.

**Computational security:**

$\text{Adv}^{\text{Enc}_K(M_0), \text{Enc}_K(M_1)}(A)$ is negligible, for every efficient adversary $A$. 
Statistical distance

Let $D_0$ and $D_1$ be two probability distributions over some set $X$.

$$\text{Dist}(D_0, D_1) = \frac{1}{2} \sum_{x \in X} |D_0(x) - D_1(x)|$$

**Proposition 1.** This is, in fact, a distance.

*Proof.* $x, y \mapsto |y - x|$ is a distance. So $\text{Dist}(\cdot, \cdot)$ is a sum of distances. (Can also write it out.)
**Proposition 2.** The statistical distance \( \text{Dist}(D_0, D_1) \) is equal to the advantage of the best adversary trying to distinguish \( D_0 \) from \( D_1 \).

**Proof.** Let \( A \) be the adversary such that, given \( x \leftarrow D_b \), \( A \) outputs 0 iff \( D_0(x) \geq D_1(x) \). \( A \) is clearly best possible.

\[
\text{Adv}_{D_0, D_1}^A = 2\Pr_{x \leftarrow D_b(x), b \leftarrow \{0,1\}}(A(x) = b) - 1
= 2 \sum_{x'} \sum_{b'} \Pr(A(x) = b|x = x', b = b') \cdot \Pr_{x \leftarrow D_b}(x = x'|b = b') \operatorname{Pr}_{b \leftarrow \{0,1\}}(b = b') - 1
= \sum_{x'} \sum_{b'} \mathbb{1}_{A(x') = b'} D_b(x') - 1
= \sum_{x'} \max(D_0(x'), D_1(x')) - 1
= \text{Dist}(D_0, D_1) \quad \text{using: } \max(a, b) = \frac{1}{2}(a + b + |b - a|).
\]
**Corollary.** Let $A$ be any algorithm. Then:

$$\text{Dist}(A(D_0), A(D_1)) \leq \text{Dist}(D_0, D_1)$$

**Proof.** Let $B$ be the best adversary distinguishing $D_0$ from $D_1$, and $C$ be the best adversary distinguishing $A(D_0)$ from $A(D_1)$.

$$\text{Dist}(A(D_0), A(D_1)) = \text{Adv}^{A(D_0), A(D_1)}(C) = \text{Adv}^{D_0, D_1}(C \circ A)$$
$$\leq \text{Adv}^{D_0, D_1}(B) = \text{Dist}(D_0, D_1).$$

**Proposition 3.** For all $n$, $\text{Dist}(D_0^n, D_1^n) \leq n \text{Dist}(D_0, D_1)$.

**Proof.**

$$\text{Dist}(A^n, B^n) \leq \text{Dist}(A^n, A^{n-1}B) + \text{Dist}(A^{n-1}B, A^{n-2}B^2) + \ldots + \text{Dist}(AB^{n-1}, B^n).$$

Sometimes called the “hybrid” argument, although the same term is also used in more general settings.
Advantage of the best adversary = statistical distance.
By extension:

**Advantage of a class of adversaries.**

Let $D_0$ and $D_1$ be two probability distributions, and $A$ a set of adversaries. Define:

$$\text{Adv}_{D_0,D_1}(A) = \sup\{\text{Adv}_{D_0,D_1}(A) : A \in A\}$$

Define $A(t)$ the set of adversaries that terminate in time $t$. Let:

$$\text{Adv}_{D_0,D_1}(t) = \text{Adv}_{D_0,D_1}(A(t))$$

**NB** For asymptotic security, what matters usually is to distinguish two *families* of distributions. We want (abuse of notation):

$$\text{Adv}_{D_0,D_1}(\text{Poly}(\lambda)) = \text{Negl}(\lambda)$$

with $D_0, D_1$ (implicitly) parametrized by $\lambda$. 
Types of security, revisited

let $M_0$ and $M_1$ be two arbitrary messages...

**Perfect security:**

$\text{Enc}_K(M_0) = \text{Enc}_K(M_1)$ (as distributions, for uniform $K$).

*Equivalently:* $\text{Dist}(\text{Enc}_K(M_1), \text{Enc}_K(M_2)) = 0$.

*Equivalently:* $\text{Adv}^{\text{Enc}_K(M_0), \text{Enc}_K(M_1)}(\{\text{all } A\}) = 0$.

**Statistical security:**

$\text{Dist}(\text{Enc}_K(M_1), \text{Enc}_K(M_2))$ is negligible.

*Equivalently:* $\text{Adv}^{\text{Enc}_K(M_0), \text{Enc}_K(M_1)}(\{\text{all } A\})$ is negligible.

**Computational security:**

$\text{Adv}^{\text{Enc}_K(M_0), \text{Enc}_K(M_1)}(\{\text{efficient } A\})$ is negligible.
A simple example

Consider a Bernoulli (coin flip) distribution $B$ with $B(0) = 1/2 - \varepsilon$ and $B(1) = 1/2 + \varepsilon$. Let $U$ be the uniform distribution on \{0,1\}. Observe:

$$\text{Dist}(B,U) = \varepsilon.$$ 

Assume we are doing a one-time pad with an imperfect randomness source, where the key bits are drawn according to $B$:

$$K \leftarrow B^n \text{ (instead of } U^n)$$

Say $\varepsilon$ is negligible (asymptotic sense).

*Is this still secure?*

Perfect security? Statistical? Computational?
A simple example, cont'd

Let's encrypt a message $M \in \{0,1\}^n$.

$$\text{Dist}(\text{Enc}_K(M), U^n) = \text{Dist}(K \oplus M, U^n) \leq \sum_{i<n} \text{Dist}((K \oplus M)_i, U)$$

$$= n\varepsilon$$

$i$-th bit of $K \oplus M$

For $M_0, M_1 \in \{0,1\}^n$.

$$\text{Dist}(\text{Enc}_K(M_0), \text{Enc}_K(M_1)) \leq \text{Dist}(\text{Enc}_K(M_0), U^n) + \text{Dist}(\text{Enc}_K(M_1), U^n) \leq 2n\varepsilon$$

Note that $n \cdot \text{Negl}(n) = \text{Negl}(n)$ so this is (statistically) secure!

(A more refined analysis shows this grows in $\sqrt{n\varepsilon}$. The hybrid argument is a little crude here.)
Theorem (Shannon ’49). If perfect secrecy holds, it must be the case that the two parties share some prior information (a key) with:

\[ \text{length(key)} \geq \text{length(message)} \]

where length denotes the bit length.

Saying perfect security is impossible.

What about *statistical* security?
Modern definition

Statistical secrecy.
Let $M_0$ and $M_1$ be two arbitrary messages.

**Statistical secrecy:** $\text{Dist}(\text{Enc}_K(M_0), \text{Enc}_K(M_1))$ is negligible.

More formally:

Statistical secrecy.
∀ $p \in \text{Poly}(\lambda)$, ∀ $M_0, M_1$ of size $p$, $\text{Dist}(\text{Enc}_K(M_0), \text{Enc}_K(M_1)) = O(\lambda^{-c})$, (where the distributions are induced by $K \leftarrow_\$ \{0,1\}^\lambda$).

Is this possible?

No. Hint: for $p > 3\lambda$, ∃ $M_0, M_1$ such that the set of possible encryptions are disjoint.
Conclusion

For many goals of cryptography (even simply symmetric encryption), can't have statistical security.

→ **Strategy:**

1) Reduce security of crypto **protocols** to the security of their basic bricks (known as **primitives**: encryption, signature etc).

2) Reduce security of **primitives** to *known* mathematical hard problem (e.g. discrete logarithm).

*How?* use arguments based on advantage, statistical distance, etc: see MPRI course 2.30 “Proofs of security protocols”!
**Conclusion**

**Corollary:** crypto requires problems that are **computationally** hard, but not information theoretically hard (= against unbounded adversaries).

Proving hardness of a problem:

Remark. Information-theoretical arguments don’t even really care what an algorithm is. (Turing machine? RAM? Quantum? Family of circuits? …)
Conclusion

Crypto requires problems that are **computationally hard**, but not information theoretically hard. *Which we don't know how to do.*

**Corollary 1.** Cryptography requires hardness assumptions.

**Corollary 2.** Entire modern world relies on statements we don't know how to prove (or if they are provable).

...and it would be catastrophic if they were wrong. (Payment ecosystem, secure Internet, private messaging, etc)

Remark. Crypto is rather unique within Computer Science in requiring that some problems should be *hard*. 
Hard problem zoo

Hard problems relate to cryptographic **primitives**. Higher-level constructions can be proved secure assuming secure underlying primitives.

**Hard problems for asymmetric primitives.**
- The RSA problem ($\neq$ factorization).
- Discrete Logarithm over certain groups.
- Hard problems in lattices.
- Syndrome decoding for random codes.
- Etc...

**Hard problems for symmetric primitives.**
- Ad-hoc assumptions: the primitive (AES, DES etc) is secure.

See MPRI 2.12.2 “Arithmetic algorithms for cryptology”
See MPRI 2.12.1 = us :)
See MPRI 2.13.2 = right after :)
Wait, no proofs for symmetric primitives?

**Functionality:** no special structure (roughly, only need to “obfuscate”).

**Speed:** handles massive data: needs to be extremely fast.

**Strategy:** take extremely efficient operations (XOR, add, bit shift etc), combine them in carefully designed, but algebraically “incoherent” ways (on purpose).

⇒ No proof (for primitives) but well-studied, and works very well.

---

vs asymmetric crypto:

**Functionality:** special structure (trapdoor, morphism etc).

**Speed:** critical but punctual operations.

**Strategy:** take mathematical objects with special properties, use those properties.

⇒ Proofs reducing to some hard problem in underlying math object.
SHA-2 hash:

RSA encryption:

Public parameter $N = pq$ with prime $p$, $q$.

Encryption of $m = m^3 \mod N$. 

In practice
What is *this* course about then?

In combination, existing MPRI crypto courses cover all major types of primitives (cf. earlier slide), both in symmetric and asymmetric crypto + proofs built on top (MPRI 2.30).

**This course:**
- second half: lattices.
- *this* half: **advanced primitives**.

**Key point:** cryptography can do *much more* than simple encryptions, signatures, key exchanges, etc.

*Think: electronic voting, cryptocurrencies, delegated computation, homomorphic encryption...*

→ We will see some important constructions beyond basic primitives.
Zero knowledge proofs
A zero-knowledge course would be a very bad course.

Image credit Oded Goldreich [www.wisdom.weizmann.ac.il/~oded/PS/zk-tut10.ps]
Expressivity

Zero-knowledge (ZK) proofs are very powerful and versatile.

On an intuitive level (for now), statements you may want to prove:

• “I followed the protocol honestly.” (but want to hide the secret values involved.) *E.g. prove election result is correct, without revealing votes.*

• “I know this secret information.” (but don't want to reveal it.) *For identification purposes.*

• “The amount of money going into this transaction is equal to the amount of money coming out.” (but want to hide the amount, and how it was divided.)
What do we want to prove?

Want to prove a statement on some $x$: $P(x)$ is true.

Exemple: $x = \text{list } V$ of encryptions of all votes + election result $R$
$P(V,R) = \text{result } R$ is the majority vote among encrypted votes $V$.

In general, can regard $x$ as a bit string.

*Equivalently:* want to prove $x \in \mathcal{L}$. (set $\mathcal{L} = \{y : P(y)\}$.)
What is a proof?

For a language $\mathcal{L}$:

Prover $P$ \hspace{1cm} Verifier $V$

- $x$ \hspace{1cm} Proof $\pi$ for $x \in \mathcal{L}$

Expected properties of proof system:

- **Completeness.** If $x \in \mathcal{L}$, then $\exists$ proof $\pi$, $V(\pi) = \text{accept}$.

- **Soundness.** If $x \notin \mathcal{L}$, then $\forall$ proof $\pi$, $V(\pi) = \text{reject}$.

- **Efficiency.** $V$ is PPT (Probabilistic Polynomial Time).

Without the last condition, definition is vacuous (prover is useless).
Zero knowledge

*Intuitively:* Verifier learns *nothing* from $\pi$ other than $x \in \mathcal{L}$.

...this is impossible for previous notion of proof.

(only possible languages are those in BPP, i.e. when the proof is useless...)

→ going to generalize/relax notion of proofs in a few ways:

- Interactive proof, probabilistic prover, imperfect (statistical) soundness...
Brief interlude: crypto magic

Challenge:

Define an injective mapping $F: \{0,1\}^\ast \rightarrow \{0,1\}^\lambda$.

How about if injectivity is only computational?

i.e. computationally hard to find $x \neq y$ s.t. $F(x) = F(y)$.

Then it's fine! It's a (cryptographic) hash function.

(Story for another time: hardness as sketched above is ill-defined.)
An Interactive Proof \((P,V)\) for \(\mathcal{L}\) must satisfy:

- **(Perfect) Completeness.** If \(x \in \mathcal{L}\), then \(P \leftrightarrow V\) accepts.

- **(Statistical) Soundness.** If \(x \notin \mathcal{L}\), then \(\forall\) prover \(P^*\), \(\Pr[P^* \leftrightarrow V \text{ rejects}] = \text{non-negl}(|x|)\). (i.e. \(\geq 1/p(|x|)\) for some fixed polynomial \(p\).)

- **Efficiency.** \(V\) is PPT.

Caveat: prover is unbounded.
Interactive Proofs

Concept discovered independently in cryptography (Goldwasser, Micali, Rackoff), and complexity theory (Babai).

Complexity theory view:

**NP** = languages that have a (non-interactive) proof with (deterministic) verifier in **P**.

**IP** = languages that have an interactive proof with probabilistic verifier.

Remark:
- “interactive proofs with deterministic verifier” = NP.
- “non-interactive proofs with probabilistic verifier” = AM.

Connection with Probabilistically Checkable Proofs (rich theory).
**IP**

**IP**: complexity class of languages that admit an interactive proof.

Public-coin proof: verifier gives its randomness to prover (AM).
Private-coin proof: no such restriction (IP). No more expressive.

**Theorem.** Shamir, LKFN at FOCS '90.

\[ \textbf{IP} = \text{PSPACE}. \]

Very powerful but in crypto, for usability, we want **efficient** (PPT) prover.

→ **argument** of knowledge.
Interactive argument of knowledge

An Interactive Proof $(P, V)$ for $\mathcal{L}$ must satisfy:

- **(Perfect) Completeness.** If $x \in \mathcal{L}$, then $P \leftrightarrow V$ accepts.

- **(Statistical) Soundness.** If $x \notin \mathcal{L}$, then $\forall$ PPT prover $P^*$, $\Pr[P^* \leftrightarrow V \text{ rejects}] = \text{non-negl}(|x|)$. (i.e. $\geq 1/p(|x|)$ for some fixed polynomial $p$.)

- **Efficiency.** $V$ is PPT.
Preliminary examples
Pepsi vs Coke is in IP

Prosper ($P$) wants to prove to Véronique ($V$) that she can distinguish Pepsi from Coke. Let $(X_0, X_1) = (\text{Pepsi, Coke})$.

**Prover $P$ (Prosper)**
- **Tasting (or chemistry?)**

**Verifier $V$ (Véronique)**
- $b \leftarrow \$ \{0, 1\}
- guess $b'$
- accept iff $b' = b$

This interactive proof is **complete** and **sound**.

Soundness error = $1/2$. Reduce to $2^{-\lambda}$: iterate the protocol $\lambda$ times.
Graph isomorphism (unbounded prover)

• Suppose two graphs $G_0, G_1$ are isomorphic: $\exists \sigma, \sigma(G_0) = G_1$.
• Prover wants to prove $G_0 \sim G_1$ without revealing anything about the isomorphism.

Formally: $\mathcal{L} = \{(G,G') : G \sim G'\}$, want to prove $(G_0, G_1) \in \mathcal{L}$.

Prover $P$

Verifier $V$

$\theta \leftarrow$ random isom. on $G_0$

$H = \theta(G_0)$

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

$\rho$ s.t. $H = \rho(G_b)$

accept iff $H = \rho(G_b)$

$(H = \rho(G_b))$
Analysis

‣ (Perfect) Completeness.
“If $x \in \mathcal{L}$, then $P \leftrightarrow V$ accepts”.

Clearly true.

‣ (Statistical) Soundness.
“If $x \notin \mathcal{L}$, then $\forall$ prover $P^*$, $\Pr[P^* \leftrightarrow V \text{ rejects}] = \text{non-negl}(|x|)$”.

True: $V$ will reject with probability $\geq 1/2$.

‣ Efficiency. $V$ is PPT.

We want to actually use this $\rightarrow$ want a bounded prover (PPT).
Graph isomorphism (bounded prover)

• Prover knows an isomorphism $\sigma$ between $G_0$, $G_1$: $\sigma(G_0) = G_1$.
• Prover wants to prove $G_0 \sim G_1$ without revealing anything about the isomorphism.

Formally: $\mathcal{L} = \{(G,G') : G \sim G'\}$, want to prove $(G_0,G_1) \in \mathcal{L}$.

Prover $P$

Verifier $V$

$\theta \leftarrow \text{random isom. on } G_0$

$H = \theta(G_0)$

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

$\rho = \theta \circ \sigma^b$

$\text{accept iff } H = \rho(G_b)$

Bounded prover who knows a witness. Public coin. Perfect ZK.
Proofs of knowledge
Motivation

We want a bounded prover (PPT). Corollary: secret prover knowledge is necessary.

In Graph isomorphism, intuitively, proof “shows” that not only statement is true, but prover knows a witness: the permutation $\sigma$.

This difference is meaningful!

Example: cyclic group $G = \langle g \rangle$. For this language...

$\mathcal{L} = \{h \mid \exists s, h = g^s\}$

Interactive Proof: trivial ($\mathcal{L} = G$ !)

Knowing a witness: hard (Discrete Log problem)
Defining knowledge

To generalize, we want witnesses.

NP languages are great: \( \mathcal{L} = \{ x \mid \exists w, R(x, w) \} \) for efficient \( R \).

→ Want to formalize: the prover \textbf{knows} a witness.

\[ \textbf{What does it mean for an algorithm to know something?} \]
Defining knowledge

Attempt 1.
Algorithm $\mathcal{A}$ knows a secret $s$ if it outputs $s$.

Attempt 2.
Algorithm $\mathcal{A}$ knows a secret $s$ if $s$ appears somewhere in the code.

Attempt 3.
Algorithm $\mathcal{A}$ knows a secret $s$ if there exists an efficient algorithm $\mathcal{E}$ that can extract $s$ from $\mathcal{A}$. 
Soundness of a knowledge proof

Knowledge soundness.

∃ efficient extractor $E$ that, given access to $P$ and $x$, can compute $w$ such that $R(x,w)$. 

can control completely, including random tape 

witness $w$ s.t. $R(x,w)$
More formally

NP Language $\mathcal{L} = \{x \mid \exists w, R(x,w)\}$.

Knowledge soundness (attempt 1, right idea, not yet perfect).

A proof system is **knowledge-sound** if and only if:

∃ efficient extractor $E^P$ with oracle access to a prover $P$ such that ∀ $x$, ∀ $P$ that convinces $V$ (with probability 1), $E^P$ outputs $w$ such that $R(x,w)$ (with probability $\geq 1/2$).

*Remark*: this is a property of the verifier.
Modern definition

NP Language $\mathcal{L} = \{x \mid \exists w, R(x,w)\}$.

Knowledge soundness (simple).
A proof system is **knowledge-sound** if and only if:

- $\exists$ efficient extractor $E^P$ with oracle access to a prover $P$ such that
- $\forall x, \forall P$ that convinces $V$ with non-negligible probability, $E^P$ outputs $w$ such that $R(x,w)$ with non-negligible probability.

Knowledge soundness (better).
A proof system is **knowledge-sound** with soundess $\kappa$ iff:

$\exists$ efficient extractor $E^P$ such that if $\epsilon = \Pr[P \leftrightarrow V \text{ accepts}] > \kappa$, then $E^P$ succeeds with probability at least $\epsilon - \kappa$.

https://www.wisdom.weizmann.ac.il/~oded/pok.html
Knowledge soundness for Graph Isomorphism

Extractor:
- calls $P$, gets $H = \theta(G_0)$.
- asks $b = 0$, and $b = 1$. This is legitimate due to randomness control!
  Gets back $\rho_0$, $\rho_1$ with $H = \rho_0(G_0) = \rho_1(G_1)$.
- $G_1 = \rho_1^{-1} \circ \rho_0(G_0) \rightarrow$ witness $\sigma = \rho_1^{-1} \circ \rho_0$.

Special soundness: two challenges reveals witness, cf. next section.
Zero knowledge
Towards zero knowledge

For language in NP, witness itself is a proof of knowledge...

- **Zero-knowledge**: prove membership or knowledge while revealing *nothing else*. 
Towards zero knowledge

Need to formalize: *the verifier learns nothing.*
Honest-verifier zero-knowledge.

The (interactive) proof system $(P,V)$ is zero-knowledge iff:

$\exists$ efficient (PPT) simulator $S$ s.t. $\forall x \in \mathcal{L}$, transcript of $P$ interacting with $V$ on input $x$ is indistinguishable from the output of $S(x)$. 
Point of definition:
- anything V could learn from interacting (honestly) with P, could also learn by just running S.
- S is efficient and knows no secret information.
⇒ Anything V can compute with access to P, can compute without P.

That expresses formally: “V learns nothing from P”.

- Is the Graph Isomorphism proof ZK?

Yes. Simulator: choose b in {0,1}, and random permutation π of Gb.
Publish as simulated transcript: (π(Gb), b, π). This is identically distributed to a real transcript → perfect zero-knowledge.

Key argument: π(Gb) for uniform π does not depend on b.
Types of zero knowledge

Let $\rho$ be the distribution of real transcripts, $\sigma$ simulated transcript.

- **Perfect ZK:** $\rho = \sigma$.
- **Statistical ZK:** $\text{dist}(\rho, \sigma)$ is negligible. ($\text{dist} = \text{statistical distance}$)
- **Computational ZK:** advantage of efficient adversary trying to distinguish $\rho$ from $\sigma$ is negligible.

Likewise: completeness, soundness can be perfect/statistical/computational.

What if the prover is **malicious** (does not follow the protocol?)
Honest-verifier Zero-knowledge

The (interactive) proof system \((P,V)\) is zero-knowledge iff:
\[\forall \text{ prover } P^*, \exists \text{ PPT simulator } S \text{ s.t. } \forall x \in \mathcal{L}, \text{ transcript of } P^* \text{ interacting with } V \text{ on input } x \text{ is indistinguishable from output of } S(x).\]
A ZK proof is (perfectly/statistically/computationally):
1. Complete
2. Sound

A ZK proof of knowledge is (perfectly/statistically/computationally):
1. Complete
2. Knowledge-Sound
Is a zero-knowledge proof of knowledge possible?

Yes. Graph Isomorphism is an example.

Subtlety: Knowledge Extractor can control the prover’s random tape, Verifier cannot.
Examples
Graph isomorphism

- I know an isomorphism $\sigma$ between two graphs $G_0$, $G_1$: $\sigma(G_0) = G_1$.
- I want to prove $G_0 \sim G_1$ without revealing anything about the isomorphism.

Formally: $\mathcal{L} = \{(G, G') : G \sim G'\}$, want to prove $(G_0, G_1) \in \mathcal{L}$.

Prover $P$

Verifier $V$

$\theta \leftarrow$ random isom. on $G_0$

$H = \theta(G_0)$

$b \leftarrow \mathbb{0,1}$

$\rho = \theta \circ \sigma^b$

$b$ accept iff $H = \rho(G_b)$

Bounded prover who knows a witness. Public coin. Perfect ZK.
• I am an unbounded prover who knows $G_0 \not\sim G_1$.
• I want to prove $G_0 \not\sim G_1$ without revealing anything else.

Formally: $\mathcal{L} = \{(G, G') : G \not\sim G'\}$, want to prove $(G_0, G_1) \in \mathcal{L}$.

Knowledge of a square root

- Public $N = pq$ for large primes $p, q$, public $x$ in $\mathbb{Z}_N$.
- I am a bounded prover who knows $w$ such that $x = w^2 \mod N$.
- I want to prove that knowledge without revealing anything else.

<table>
<thead>
<tr>
<th>Prover $P$</th>
<th>Verifier $V$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r \leftarrow \mathbb{Z}_N$</td>
<td>$b \leftarrow {0,1}$</td>
</tr>
<tr>
<td>$y = r^2$</td>
<td>accept iff $z^2 = x^by$</td>
</tr>
<tr>
<td>$z = w^br$</td>
<td></td>
</tr>
</tbody>
</table>
Knowledge of a discrete log

- Let \( G = \langle g \rangle \sim \mathbb{Z}_p \) and \( y \in G \). I know \( x \in \mathbb{Z}_p \) such that \( y = g^x \).
- Corresponding language is trivial! \( \forall y \exists x, y = g^x \). But proof of knowledge still makes sense.

Known as Schnorr protocol.
Analysis of Schnorr protocol

- **(Perfect) Completeness.**
  
  Clear.

- **(Special) Knowledge soundness.**
  
  **Extractor:** gets \( r = g^k \), asks two challenges \( e \neq e' \), gets back \( s, s' \) with \( r = g^s y^e = g^{s'} y^{e'} \). Yields \( y = g^{(s-s')/(e'-e)} \).

- **(Perfect) Honest-verifier zero knowledge.**
  
  **Simulator:** draw \( e \leftarrow \mathbb{Z}_p \), \( s \leftarrow \mathbb{Z}_p \), then \( r = g^s y^e \). Return transcript \((r,e,s)\). Note \( r, e \) still uniform and independent \( \rightarrow \) distribution is identical to real transcript.

We will use this for a signature!
Equality of exponents = DH language

• Let $\mathbb{G} \sim \mathbb{Z}_p$, $g, h \in \mathbb{G}$. I know $x \in \mathbb{Z}_p$ such that $(y, z) = (g^x, h^x)$.
• Corresponding language is Diffie-Hellman language (for fixed $g, h$)!
  $\mathcal{L} = \{(g, g^a, g^b, g^{ab}) : a, b \in \mathbb{Z}_p\} \leftrightarrow \mathcal{L}' = \{(g^a, h^a) : a \in \mathbb{Z}_p\}$ for $h = g^b$

![Diagram](image)

This is two ‘simultaneous’ executions of Schnorr protocol, with same $(k, e)$. Soundness and ZK proofs are the same.

We will use this in a voting protocol!
Sigma protocols and NIZK
**Sigma protocol**

**Schnorr protocol:**

- **Prover** $P$
- **Verifier** $V$

1. $k \leftarrow \mathbb{Z}_p$
2. $r = g^k$
3. $e \leftarrow \mathbb{Z}_p$
4. $s = k - xe$

Accept if $r = g^s y^e$

Public-coin ZK protocols following this pattern = **Sigma Protocols**.

**Fiat-Shamir transform:**

By setting $\text{Challenge} = \text{Hash(Commit)}$, can be made non-interactive

$\rightarrow$ Non-Interactive Zero-Knowledge (NIZK)
Special soundness: “answering two distinct uniform challenges for the same commit $\Rightarrow$ knowing witness”

Special soundness $\Rightarrow$ Knowledge soundness $\Rightarrow$ Soundness
Sigma protocol $\rightarrow$ signature

**NIZK knowledge proof**: “I know a witness $w$ for $R(x,w)$” and can prove it non-interactively without revealing anything about $w$.

This is an identification scheme.

**Sigma protocol $\rightarrow$** can integrate message into challenge randomness.

This yields a signature scheme!

- **Public key**: $x$
- **Secret key**: $w$
- **Sign$(m)$**: signature = NIZK proof with challenge = hash(commit,$m$)
- **Verify** signature = verify proof.

That is the Fiat-Shamir transform.
Example: Schnorr signature

Schnorr protocol:

\[ k \leftarrow \mathbb{Z}_p \]
\[ r = g^k \]
\[ e \leftarrow \mathbb{Z}_p \]
\[ s = k - xe \]
\[ \text{accept iff } r = g^s y^e \]

Schnorr signature:

**Public key:** \( y = g^x \)

**Secret key:** \( x \)

**Sign**(\( m \)): signature \( \sigma = (r, s) \) with \( r = g^k \) for \( k \leftarrow \mathbb{Z}_p \), \( s = k - xH(r,m) \).

**Verify**(\( \sigma, m \)): accept iff \( r = g^s y^{H(r,m)} \).

Security reduces to Discrete Log, in the Random Oracle Model.
ZK proofs for arbitrary circuits
Reductions

Suppose there exists an efficient (polynomial) reduction from $\mathcal{L}'$ to $\mathcal{L}$:

$\exists$ efficient $f$ such that $x \in \mathcal{L}'$ iff $f(x) \in \mathcal{L}$. (Karp reduction.)

If I can do ZK proofs for $\mathcal{L}$, I can do ZK proofs for $\mathcal{L}'$!

To prove $x \in \mathcal{L}'$, do a ZK proof of $f(x) \in \mathcal{L}$.

Also works for knowledge proofs (via everything being constructive).

⇒ The dream: if we can do ZK proof for an NP-complete language, we can prove everything we ever want!

Notably circuit-SAT.
A commitment scheme is a (family of) functions $C: X \times A \rightarrow V$ s.t.:

- **Binding**: it is hard to find $x \neq x'$ and $a, a'$ s.t. $C(x,a) = C(x',a')$.
- **Hiding**: for all $x, x'$, the distributions $C(x,a)$ for $a \leftarrow A$ and $C(x',a)$ for $a \leftarrow A$ are indistinguishable.

Usage:

- Alice **commits** to a value $x$ by drawing $a \leftarrow A$ and sending $C(x,a)$.
- Later, Alice **opens** the commitment by revealing the inputs $x, a$.

Instantiation: pick a hash function.
The dream: ZK proof for 3-coloring

- I know an 3-coloring $c$ of a graph $G$ (into $\mathbb{Z}_3$).
- I want to prove that such a coloring exists, without revealing anything about the coloring.

Formally: $\mathcal{L} = \{(G): G \text{ admits a 3-coloring}\}$

Bounded prover with a witness. Public coin. Computational ZK.
...this is incredibly inefficient.

- transform circuit-SAT instance into 3-coloring instance.

- run previous protocol many times (roughly circuit size × security parameter) → gigantic proofs, verification times...
SNARKs

SNARK(?) tile by William Morris.
Most of what follows is going to happen in a finite field.

For a short presentation of finite fields, see:


A **key idea** we will use:

If $P \neq Q$ are two degree-$d$ polynomials over $\mathbb{F}_q$, then for $\alpha \leftarrow \mathbb{F}_q$ drawn uniformly at random, $\Pr[ P(\alpha) \neq Q(\alpha) ] \geq 1 - d/q$.

→ to check if two bounded-degree polynomials are equal, it is enough to check at a random point!

*Proof:* $P-Q$ is a non-zero polynomial of degree at most $d$, so it can be zero on at most $d$ points.
Véronique wants to compute the 1000\(^{th}\) Fibonacci number in \(\mathbb{Z}_p\).

She doesn't have time, so she asks Prosper to do it. But she wants a **proof** that the computation was correct.

**“Solution”:** agree on whole computation circuit → encode as SAT problem → transform into 3-coloring problem → include NIZK proof of that 3-coloring problem with the result.

Remark: size of proof is linear in the size of the circuit Véronique doesn't want to compute.

(P & V hate closed formulas and fast exponentiation.)
We would like to achieve zero-knowledge proofs that are succinct and non-interactive.

Succinct Non-interactive Argument of Knowledge: SNARK.

Also a fantastical beast by Lewis Caroll:
A new approach

Prosper computes the Fibonacci sequence $f_1, ..., f_{1000}$ in $\mathbb{Z}_p$. He sends $f_1$, $f_2$, and $f_{1000}$ to Véronique.

Now V. wants to check $f_{i+2} = f_i + f_{i+1}$ for all $i$'s.

**Magic claim:** she will be able to check that this computation was correct, for all $i$, with 99% certainty, by asking Prosper for only 4 values in $\mathbb{Z}_p$.

**Disclaimers:**
- we assume Prosper answers queries honestly (for now).
- from now on, assume $|\mathbb{Z}_p|$ is “large enough”, say $|\mathbb{Z}_p| > 100000$. (Otherwise, just go to a field extension.)

This line of presentation is loosely borrowed from Eli Ben-Sasson:
https://www.youtube.com/watch?v=9VuZvdxFZQo
A new approach

**Setup:** Prosper interpolates a degree-999 polynomial $P$ in $\mathbb{Z}_p$ such that $P(i) = f_i$ for $i = 1, \ldots, 1000$.

Let $D = (X-1) \cdot (X-2) \cdot \ldots \cdot (X-998)$.

\[
P(i+2) - P(i+1) - P(i) = 0 \text{ for } i = 1, \ldots, 998
\]
\[
\Leftrightarrow \quad D \text{ divides } P(X+2) - P(X+1) - P(X)
\]
\[
\Leftrightarrow \quad P(X+2) - P(X+1) - P(X) = D \cdot H \text{ for some } H
\]

**How Véronique checks that the computation was correct:**

- Véronique draws $\alpha \leftarrow \mathbb{Z}_p$ uniformly, computes $D(\alpha)$.
- She asks Prosper for $P(\alpha)$, $P(\alpha+1)$, $P(\alpha+2)$, $H(\alpha)$.
- She accepts computation was correct iff:
  \[
P(\alpha+2) - P(\alpha+1) - P(\alpha) = D(\alpha) \cdot H(\alpha)
\]
Why the approach works

**Completeness:** if Prosper computed the $f_i$'s **correctly**, then he can compute $H(\alpha)$ as required.

**Soundness:** The only requirements for soundness to hold are

- The same polynomial $P$ was used to compute $P(\alpha)$, $P(\alpha+1)$, $P(\alpha+2)$ (as well as $P(1)$, $P(2)$, $P(1000)$);
- $P$ and $H$ have the correct degree (resp. 1000 and 1).

If Prosper computed the $P(i) = f_i$'s **incorrectly**, then as long as the previous requirements hold, we have:

$$\Pr[ P(\alpha+2) - P(\alpha+1) - P(\alpha) = D(\alpha) \cdot H(\alpha) ] \leq \frac{1000}{\rho} < 0.01$$

so Véronique will detect the issue with > 99% probability.

(An implicit assumption here is: $H$ does not depend on $\alpha$.)
It remains to force Prosper to answer queries honestly.

In particular, soundness argument crucially relies on $P, H$ being bounded-degree polys.

→ need to limit Prosper to computing polys of degree $< 1000$.

→ A new ingredient: **pairings**.
Quick “reminder”

Fix cyclic group $\mathbb{G} = \langle g \rangle$.

**Discrete Logarithm Problem**: given $g^a$ for uniform $a$, compute $a$.

**Computational Diffie-Hellman Problem**: given $(g^a, g^b)$ for uniform $a$, $b$, compute $g^{ab}$.

In crypto, it is often assumed that these problems are difficult (in the relevant group).

Example of group used in practice: prime subgroup of $\mathbb{Z}_p^*$.
Pairings. Let $\mathbb{G} = \langle g \rangle$, $\mathbb{T} = \langle t \rangle$ be two cyclic groups of order $p$. A map $e: \mathbb{G} \times \mathbb{G} \to \mathbb{T}$ is a \textit{pairing} iff for all $a, b$ in $\mathbb{Z}_p$,

$$e(g^a, g^b) = t^{ab}.$$ 

Remarks:

- Definition doesn't depend on choice of generators, as long as $t = e(g, g)$.

- Assume Discrete Log is hard in $\mathbb{G}$, otherwise this is useless. On the other hand, $e$ implies DDH cannot be hard (why?).

- First two groups need not be equal in general.

- Can be realized with $\mathbb{G}$ an elliptic curve, $\mathbb{T} = \mathbb{F}_q^*$. 
Encodings

Fix $G = \langle g \rangle$ of order $p$.

**Encode** a value $a \in \mathbb{Z}_p$ as $g^a$. We will write $[a] = g^a$.

We assume DL is hard $\rightarrow$ decoding a *random* value is hard. But encoding is deterministic $\rightarrow$ checking if $h \in G$ encodes a given value is easy.

**Additive homomorphism:** given encodings $[a],[b]$ of $a$ and $b$, can compute encoding of $a+b$: $[a+b] = [a][b]$.

$\rightarrow$ can compute $\mathbb{Z}_p$-linear functions over encodings.

**Idea:** a pairing $e: \langle g \rangle \times \langle g \rangle \rightarrow \langle t \rangle$ allows computing quadratic functions over encodings (at the cost of moving to $\mathbb{T}$).
Keeping Prosper honest, using encodings

First: want to ensure $P$ computed by Prosper is degree $\leq 1000$.

**Approach:**

- Véronique draws evaluation point $\alpha \leftarrow \mathbb{Z}_p$ uniformly at random.

- V. publishes encodings $[\alpha], [\alpha^2], \ldots, [\alpha^{1000}]$.

$\rightarrow$ Prosper can compute $[P(\alpha)]$, because it is a linear combination of the $[\alpha^i]$'s, $i \leq 1000$. But only for $\deg(P) \leq 1000$.

E.g. cannot compute $[\alpha^{1001}]$.

Prosper can compute in the same way $[P(\alpha)], [P(\alpha+1)], [P(\alpha+2)], [H(\alpha)]$.

**Remark:** Prosper can compute $[(\alpha+1)^i]$ from the $[\alpha^j]$'s for $j \leq i$. 
Remaining issues:

1) ensure value “\([P(\alpha)]\)” returned by Prosper is in fact a linear combination of \([\alpha^i]\)'s.

2) ensure \(\deg(H) \leq 1\), not 1000.

3) ensure \([P(\alpha)], [P(\alpha+1)], [P(\alpha+2)]\) etc. are from same polynomial.

4) last issue: how does Véronique check the result? Cannot decode encodings.
Dealing with issues (1) and (2)

1) ensure \([P(\alpha)]\) is in fact a linear combination of \(\alpha^i\)'s.
2) ensure \(\text{deg}(H) \leq 1\), not 1000.

Solution:
V. publishes encodings \(\alpha\), \(\alpha^2\), ..., \(\alpha^{1000}\)... 
...and also encodings \(\gamma\), \(\gamma\alpha\), \(\gamma\alpha^2\), ..., \(\gamma\alpha^{1000}\) for a uniform \(\gamma\).

→ Prosper can compute \([P(\alpha)]\) and \([\gamma P(\alpha)]\), and send them to V.
V. can now use the pairing \(e\) to check: \(e([P(\alpha)], \gamma) = e([\gamma P(\alpha)], [1])\).

The point: if Prosper did not compute \([P(\alpha)]\) as linear combination of \(\alpha^i\)'s, he cannot compute \([\gamma P(\alpha)]\). (Note this is quadratic.)

This is an ad-hoc knowledge assumption (true in a generic model).
Goal

1) ensure \([P(\alpha)]\) is in fact a linear combination of \([\alpha^i]\)'s.

2) ensure \(\text{deg}(H) \leq 1\), not 1000.

Solution:

V. publishes encodings \([\alpha], [\alpha^2], \ldots, [\alpha^{1000}]\)...

...and also encodings \([\eta], [\eta\alpha]\), for a uniform \(\eta\).

\[\rightarrow\] Prosper can compute \([H(\alpha)]\), and \([\eta H(\alpha)]\).

V. can check: \(e([H(\alpha)],[\eta]) = e([\eta H(\alpha)],[1])\).

The point: if Prosper did not compute \([H(\alpha)]\) as linear combination of \([\alpha^i]\)'s, \(i \leq 1\), he cannot compute \([\eta H(\alpha)]\).
Dealing with issue (3)

Goal

3) ensure \([P(\alpha)], [P(\alpha+1)], [P(\alpha+2)]\) etc. are from same polynomial.

Solution:
Let's deal with \([P(\alpha)], [P(\alpha+1)]\).

V. publishes \([\theta], [\theta((\alpha+1)^2-\alpha^2)], \ldots, [\theta((\alpha+1)^{1000}-\alpha^{1000})]\) for a uniform \(\theta\).

\[\rightarrow\] Prosper can compute \([\theta(P(\alpha+1)-P(\alpha))]\).

V. can check: \(e([\theta(P(\alpha+1)-P(\alpha))],[1]) = e([P(\alpha+1)-P(\alpha)],[\theta]).\)

The point: if Prosper did not compute \([P(\alpha)], [P(\alpha+1)]\) with same coefficients, he cannot compute \([\theta(P(\alpha+1)-P(\alpha))]\).
Checking divisibility

Summary of 3 previous slides: we have forced Prosper to compute $[P(\alpha)], [H(\alpha)], ...$ as polys of correct degree.

Remains to check $P(\alpha+2)-P(\alpha+1)-P(\alpha) = D(\alpha) \cdot H(\alpha)$, using the encodings.

**No problem!** this is a quadratic equation. Check:

$e([P(\alpha+2)-P(\alpha+1)-P(\alpha)], [1]) = e([D(\alpha)], [H(\alpha)])$

**Conclusion.** Since $P(\alpha), H(\alpha)$ etc are polys of right degree, original argument applies: checking equality at random $\alpha$ ensures with $\geq 1 - 1000/|\mathbb{Z}_p| > 99\%$ probability the equality is true on the whole polys $\rightarrow D$ divides $P(\alpha+2)-P(\alpha+1)-P(\alpha) \rightarrow$ computation was correct.
Efficiency

Prosper proves correct computation by providing a constant number of encodings: \([P(\alpha)], [\gamma P(\alpha)], [H(\alpha)], [\eta H(\alpha)]\) etc.

#encodings is absolute constant, independent of circuit size.

Pre-processing by Véronique was still linear in circuit size: publishes \([\alpha^i], i \leq 1000\), etc. But...

- Can be amortized over many circuits.
- Exist “fully succinct” SNARKs, with O(log(circ. size)) verifier pre-processing.
Working with circuits directly

In essence: we have seen how to do a succinct proof of polynomial divisibility.

Can in principle encode valid machine state transitions as polynomial constraints $\rightarrow$ succinct proofs for circuit-SAT.

Now: want to do that more concretely = get SNARKs for circuit-SAT (directly).
We are going to encode a circuit as polynomials.

For simplicity, forget about negations. Write circuit with \(\oplus\) (XOR), \(\otimes\) (AND) gates. Then:

1) Associate an integer \(i\) to each input; and to each output of a mult gate \(\times\).

2) Associate an element \(r_i \in \mathbb{F}_q\) to mult gate \(i\).

Now circuit can be encoded as polys. For each \(i = 1, \ldots, 6\), define polynomials \(v_i, w_i, y_i\):

\[
\begin{align*}
\cdot v_i(r_j) &= 1 \text{ if value } i \text{ is left input to gate } j, \ 0 \text{ if not.} \\
\cdot w_i(r_j) &= 1 \text{ if value } i \text{ is right input to gate } j, \ 0 \text{ if not.} \\
\cdot y_i(r_j) &= 1 \text{ if value } i \text{ is output of gate } j, \ 0 \text{ if not.}
\end{align*}
\]
In this case, \( v_i, w_i, y_i \) are degree 2.

Encoding mult gate 5:
\[ v_3(r_5) = 1, \quad v_i(r_5) = 0 \text{ otherwise.} \]
\[ w_4(r_5) = 1, \quad w_i(r_5) = 0 \text{ otherwise.} \]
\[ y_5(r_5) = 1, \quad y_i(r_5) = 0 \text{ otherwise.} \]

Encoding mult gate 6:
\[ v_1(r_6) = v_2(r_6) = 1, \quad v_i(r_6) = 0 \text{ otherwise.} \]
\[ w_5(r_6) = 1, \quad w_i(r_6) = 0 \text{ otherwise.} \]
\[ y_6(r_6) = 1, \quad y_i(r_6) = 0 \text{ otherwise.} \]

**The point:** an assignment of variables \( c_1, ..., c_6 \) satisfies the circuit iff:
\[
(\Sigma c_i v_i(r_5)) \cdot (\Sigma c_i w_i(r_5)) = \Sigma c_i y_i(r_5) \quad \text{and} \quad (\Sigma c_i v_i(r_6)) \cdot (\Sigma c_i w_i(r_6)) = \Sigma c_i y_i(r_6)
\]

Equivalently:
\[
(X\!-\!r_5)(X\!-\!r_6) \text{ divides } (\Sigma c_i v_i) \cdot (\Sigma c_i w_i) - \Sigma c_i y_i
\]
we have reduced:

“Prosper wants to prove he knows inputs satisfying a circuit.”

into:

“Prosper wants to prove he knows linear combinations $V = \sum c_i v_i$, $W = \sum c_i w_i$, $Y = \sum c_i y_i$, such that $T = (X - r_5)(X - r_6)$ divides $VW - Y$.”

\[ \iff \exists H, \ T \cdot H = V \cdot W - Y \]

We know how to do that!

V. publishes $[\alpha^i]$, plus auxiliary $[\gamma \alpha^i]$ etc... (at setup, indep. of circuit)
P.'s proof is $[V(\alpha)]$, $[W(\alpha)]$, $[Y(\alpha)]$, $[H(\alpha)]$, plus auxiliary $[\gamma V(\alpha)]$ etc...
V. checks $e(T(\alpha), H(\alpha)) = e([V(\alpha)], [W(\alpha)]) e([Y(\alpha)], [1])^{-1}$ and auxiliary stuff.

Constant-size proof. Construction works for any circuit.
Succinct Zero Knowledge?

Proof: $([V(\alpha)], [W(\alpha)], [Y(\alpha)], [H(\alpha)])$ (+ auxiliary values)

*Is it ZK?*

- ZK not needed for “delegation of computation” application.
- But needed for other applications.
Zero Knowledge

**Proof:** \([V(\alpha), W(\alpha), Y(\alpha), H(\alpha)]\) (+ auxiliary values)

**Validity check:** \(T \cdot H = V \cdot W - Y\) at point \(\alpha\) (+ auxiliary tests)

(That is, concretely we check: \(e([T(\alpha), H(\alpha)]) = e([V(\alpha), W(\alpha)])e([Y(\alpha), 1])^{-1}\)

Natural idea to build ZK simulator:
Pick proof \((a,b,c,d)\) uniformly among values that verify the validity check.

*Problem:* cannot argue indistinguishability.
*Solution:* modify proof so that the above simulator is indistinguishable.
zk-SNARK

\[ [V'] = [V + d_V T] \quad d_V \leftarrow \mathbb{Z}_p \]

\[ [W'] = [V + d_W T] \quad d_W \leftarrow \mathbb{Z}_p \]

\[ [Y'] = [Y + d_Y T] \quad d_Y \leftarrow \mathbb{Z}_p \]

\[ [H'] = [(V' \cdot W' - Y') / T] \quad \text{well-defined: } T \mid V' \cdot W' - Y' , \text{ because } T \mid V \cdot W - Y. \]

**Old proof:** \( ([V(\alpha)], [W(\alpha)], [Y(\alpha)], [H(\alpha)]) \)

**New proof:** \( ([V'(\alpha)], [W'(\alpha)], [Y'(\alpha)], [H'(\alpha)]) \)

**Validity check:** unchanged!

New proof is ZK because it is a uniform among 4-tuples that satisfy validity*.

⚠ Above statement assumes \([\alpha]'s + auxiliary values were built honestly.\)

**Remark:** also, need to adapt auxiliary checks.

* assuming \(T(\alpha) \neq 0\), which holds except with negligible probability. Building simulator left as exercise.
**Subversion of preprocessing**

**Q:** How to enforce the $[\alpha^i]'s$ + auxiliary values were built honestly?

**A:** check $[\alpha^i][\alpha] = [\alpha^i]$ for all $i$.

Same idea for auxiliary values: check $[\alpha^i][\gamma] = [\gamma \alpha^i]$ for all $i$, etc.

**Q:** What if the pairing is asymmetric ($G_1 \times G_2 \to T$ instead of $G \times G \to T$)?

**A:** Publish $[\alpha^i]'s$ in both groups. Can then check as above.

*Historical note:* for fully succinct SNARKs subversion of preprocessing is a non-trivial issue. Initial setup required “trusted setup” → “ceremonies”.
Recap: the scheme

Prover $P$

- Check coherence

Verifier $V$

$[\alpha], [\alpha^2], \ldots, [\alpha^n]$

aux. values:

$[\gamma\alpha^i]'s$, $[\eta\alpha^i]'s$, etc.

Preprocessing

Proof(s)

Prover $P$

Verifier $V$

$([V'(\alpha)], [W'(\alpha)], [Y'(\alpha)], [H'(\alpha)])$

aux. values:

$[\gamma V'(\alpha)]'s$, $[\eta V'(\alpha)]'s$, etc.

- Check validity including coherence of aux. values.
Recap: the ideas

**Key step:** convert target computation/language into polynomials.

*Why? Polynomials...*
- ...enable "key lemma" about checking equality at single point ("minimal distance" property). Key in "succint proof"-style results.
- ...enable enforcing honesty efficiently, here via encodings + pairings.

**Even better:** arrange that checking proof = checking quadratic equation.

Enables use of pairings. More specific to this approach.
In practice

Construction was proposed in Pinocchio scheme (Parno et al. S&P 2013).

Practical: proofs ~ 300kB, verification time ~ 10 ms.
- Introduced for verifiable outsourced computation.
- Zero-knowledge variant (built as seen earlier).
- Further improvements since.
Applications

Real-World Crypto
Application #1: e-Voting
e-Voting

Are going to see (more or less) Helios voting system.
https://heliosvoting.org/

Used for many small- to medium-scale elections. Including IACR (International Association for Cryptologic Research).

We will focus on yes/no referendum.

Nice description of Belenios variant: https://hal.inria.fr/hal-02066930/document
Goals

We want:

‣ Vote privacy
‣ Full verifiability:
  • Voter can check their vote was counted
  • Everyone can check election result is correct
    Every voter cast $\leq 1$ vote, result $=$ number of yes votes

We do not try to protect against:

‣ Coercion/vote buying
Quick reminder: Diffie-Hellman key exchange

\[ G = \langle g \rangle, \quad |G| = p \]

\[ a \leftarrow_{\$} \mathbb{Z}_p \]

\[ g^a \]

\[ b \leftarrow_{\$} \mathbb{Z}_p \]

\[ g^b \]

Afterwards: Alice & Bob can both compute \( g^{ab} \).
But computing \( g^{ab} \) is hard for external observer (DDH problem).
Quick reminder: ElGamal Encryption

\[ G = \langle g \rangle, \quad |G| = p \]

**Secret key:** \( x \in \mathbb{Z}_p \)

**Public key:** \( h = g^x \)

\[ \text{Enc}(m) = (g^r, m \cdot h^r), \text{ where } r \leftarrow \mathbb{Z}_p \]

\[ \text{Dec}(k,c) = c/k^x \]

Security reduces to DDH assumption over \( G \).

**Multiplicatively homomorphic:** given \((k,c) = \text{Enc}(m), (k',c') = \text{Enc}(m')\), \((kk',cc')\) is an encryption of \(mm'\).
Election = want to add up encrypted votes...
→ just use **additively homomorphic** encryption!

Helios: use ElGamal. **Multiplicatively** homomorphic.
To make it additive: vote for $v$ is $g^v$.
Recovering $v$ from $g^v$ is discrete log, but brute force OK ($v$ small).

In addition: voters sign their votes.
Helios: Schnorr signatures.

Who decrypts the result?
Public bulletin board

Voter $i$
- own voter secret sig. key $sk_i$
- want to vote $v_i \in \{0,1\}$

Anobody
- checks encrypted result $c = \sum c_i$

Decryption trustee
- generates ElGamal master key pair $(mpk=g^x, msk=x)$

Problem: how to verify final result.
Making election result verifiable

ElGamal encryption:

Master keys: \((\text{mpk} = g^x, \text{msk} = x)\)

Encrypted election result \(c = (c_L = g^k, c_R = m \cdot g^{xk})\)

Election result = \(\text{Dec}(c) = m = c_R / c_L^x\)

\(\rightarrow\) giving decryption is same as giving \(c_L^x\)

\(\rightarrow\) to prove decryption is correct, prove:

\(\text{discrete log of } (c_L)^x \text{ in base } c_L = \text{discrete log of } \text{mpk} = g^x \text{ in base } g\)

\(\iff (g, g^x, c_L, c_L^x) \in \text{Diffie-Hellman language}\)

\(\rightarrow\) to make election result verifiable: decryption trustee just provides NIZK proof of DH language for \((g, g^x, c_L, c_L^x)\)!

Take ZK proof of DH language from earlier + Fiat-Shamir \(\rightarrow\) NIZK

Note ZK property is crucial.
Now with verifiable election result

Voter $i$
owns voter secret sig. key $sk_i$
wants to vote $v_i \in \{0,1\}
generates
- votes: $c_i = \text{enc}_{mpk}(v_i)$
- signatures: $\text{sig}_{sk}(c_i)$

Anobody
checks
- encrypted result: $c = \sum c_i$
- result: $\text{dec}_{msk}(c) + \text{DH proof}$

Decryption trustee
generates ElGamal master key pair ($mpk=g^x, msk=x$)

Public bulletin board
- Voter public sig. keys: $pk_i$
- Master public key: $mpk=g^x$

Problem 2: how about I vote $\text{enc}_{mpk}(1000)$?
Proving individual vote correctness

In addition to vote $\text{enc}_{\text{mpk}}(v_i)$ and signature $\text{sig}_{\text{sk}_i}(c_i)$, voter provides NIZK proof that $v_i \in \{0,1\}$.

Helios doesn't use SNARK here, but more tailored proof of disjunction.

Note ZK property is crucial again.

To prevent “weeding attack” (vote replication):
NIZK proof includes $g^k$, $\text{pk}_i$ in challenge randomness (hash input of sigma protocol), where $g^k$ is the randomness used in $\text{enc}_{\text{mpk}}(v_i)$.
$\rightarrow$ proof (hence vote) cannot be duplicated without knowing $\text{sk}_i$. 
Voter $i$
  owns voter secret sig. key $sk_i$
  wants to vote $v_i \in \{0,1\}$

Generated:
  - votes: $c_i = \text{enc}_{mpk}(v_i) + \text{proof} \leq 1$
  - signatures: $\text{sig}_{sk}(c_i)$

Anobody
  checks
  - encrypted result: $c = \sum c_i$
  - result: $\text{dec}_{msk}(c) + \text{DH proof}$

Decryption trustee
  generates ElGamal master key pair $(mpk=g^x, msk=x)$

Public bulletin board
  - Voter public sig. keys: $pk_i$
  - Master public key: $mpk=g^x$

Bonus problem: replace decryption trustee by threshold scheme.
Application #2: Anonymous Cryptocurrencies
Cryptocurrencies

Electronic Money: credit cards etc. *Traceable* + *Central authority* (bank).

≠ Electronic Cash: not traceable.

≠ Cryptocurrencies: no central authority*.

*in principle.*
Normal life

Client → Bank → Shop
Problem: double spending

Fundamental problem with electronic money.
Central Authority solution

Central authority: keeps ledger of who has spent what.

Check not spent

same coin

coin $i$

coin $i$

coin $i$

coin $i$

coin $i$
Cryptocurrency solution: public ledger

No bank → who checks validity of transactions? (no double spending)

Idea: just publish all transactions! Everybody can check.

Public ledger:
How to prevent people from writing any transaction they want?

An account is a (public key, secret key) pair for signature scheme.

**Pseudo-anonymity:** account is just a key.

Alice: \((pk_A, sk_A)\).
Bob: \((pk_B, sk_B)\).

Ledger: 
\[ pk_A \rightarrow pk_B \]
\[ + \text{sign}_{sk_A}(pk_A \rightarrow pk_B) \]
Accounts

Ledger: \( pk_A \rightarrow pk_B \)

\[ + \text{sign}_{skA}(pk_A \rightarrow pk_B) \]

How do you know \( pk_A \) has the money?

Comes from previous transaction (tx) in the ledger (chain).

\( pk_C \rightarrow pk_A \)

\[ + \text{sign}_{skA}(pk_C \rightarrow pk_A) \]

\( pk_A \rightarrow pk_B \)

\[ + \text{sign}_{skA}(pk_A \rightarrow pk_B) \]
Fungibility

One transaction:

\[
\begin{align*}
3 \cdot \text{pk}_A & \quad \text{pk}_D & \quad 2 \\
1 \cdot \text{pk}_B & \quad \text{pk}_E & \quad 1.5 \\
1 \cdot \text{pk}_C & \quad \text{pk}_A & \quad 1.5
\end{align*}
\]

\[\Sigma = 5 \quad \sum \quad \Sigma = 5\]

+ signatures with \(\text{sk}_A, \text{sk}_B, \text{sk}_C\).

Payback: \(\text{pk}_A\) is giving the change back to itself.
Ledger is a chain of transactions.

<table>
<thead>
<tr>
<th></th>
<th>pkA</th>
<th>pkD</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pkB</td>
<td>pkE</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>pkC</td>
<td>pkA</td>
<td>1.5</td>
</tr>
<tr>
<td>+ sign</td>
<td>skA, skB, skC.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>pkA</th>
<th>pkH</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pkF</td>
<td>pkI</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>pkG</td>
<td>pkA</td>
<td>1</td>
</tr>
<tr>
<td>+ sign</td>
<td>skA, skF, skG.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No real notion of account: every tx input links to previous unspent tx output (utxo).

To receive money, user can create new “account” (pk, sk) as destination, for every tx.
The blockchain

Transactions are arranged into blocks.

One block

<table>
<thead>
<tr>
<th>3</th>
<th>pk_A</th>
<th>pk_D</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pk_B</td>
<td>pk_E</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>pk_C</td>
<td>pk_A</td>
<td>1.5</td>
</tr>
</tbody>
</table>

+ sign sk_A, sk_B, sk_C.

| 1.5 | pk_A | pk_H | 2 |
| 2  | pk_F | pk_I | 1 |
| 0.5 | pk_G | pk_A | 1 |

+ sign sk_A, sk_F, sk_G.

tx 1  tx 2...
The blockchain

Blocks are arranged into a chain.

Each new block contains hash(previous block).
Bitcoin and anonymity

Whole transaction graph is public!

Can trace transactions. See e.g. Ron and Shamir 2012.

\[
\begin{array}{ccc}
3 & pk_A & pk_D & 2 \\
1 & pk_B & pk_E & 1.5 \\
1 & pk_C & pk_A & 1.5 \\
+ sign & sk_A, sk_B, sk_C.
\end{array}
\]

Probably same person
Bitcoin and anonymity

Suspicious activity.
Normal transaction

<table>
<thead>
<tr>
<th>Amount</th>
<th>Public Key 1</th>
<th>Public Key 2</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$p_{ka}$</td>
<td>$p_{kd}$</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>$p_{kb}$</td>
<td>$p_{ke}$</td>
<td>1.5</td>
</tr>
<tr>
<td>1</td>
<td>$p_{kc}$</td>
<td>$p_{ka}$</td>
<td>1.5</td>
</tr>
</tbody>
</table>

+ sign $s_{ka}$, $s_{kb}$, $s_{kc}$.

Encrypt is public: identities (public keys), amounts.
Hidden transaction (Zcash)

Everything is encrypted: identities, amounts.

Need to check:

- Sum of inputs = sum of outputs. ➡️ ZK proof (w/ homomorphism)
- Quantities are positive. ➡️ ZK proof (“range proof”)
- Spender knows secret key for each spender ID. ➡️ zk-SNARK
- Spender IDs correspond to previous unspent tx's. ➡️ ?
Interlude: Merkle trees

Leaves of the Merkle tree are (all) commits $c_1, \ldots, c_n$.
Each internal node of the Merkle tree is a hash of its children.
To prove \( c \) is a leaf, without revealing location, prove:

“The exist \( x_1, \ldots, x_h \) such that \( R = H(x_h, \ldots, H(x_2, H(x_1, c))\ldots) \).”

where \( H \) is collision-resistant (with symmetric inputs).
Linking transaction to unspent tx's

Scheme (simplified):

‣ Every tx contains \texttt{commit} = \text{Hash}(recipient, tx-ID, randomness).
‣ Every tx publishes \texttt{nullifier} = \text{Hash}(spending key, tx-ID) for the tx-ID of previous tx it spends from.

Blockchain stores:

- \textbf{T}: Merkle tree of \texttt{commits}
- \textbf{L}: List of \texttt{nullifiers}

When a new tx is issued:

‣ Check ZK proof (included in tx) that \(\exists\) \texttt{commit} for spender in \textbf{T}.
‣ Check \texttt{nullifier} is new (not in \textbf{L}), then add it to \textbf{L}.
‣ Check ZK proof (included in tx) that \texttt{commit} and \texttt{nullifier} are for same tx-ID.
Non-application: Annoying mathematicians

Please don't.
Theorem proving vs NP

Let's encode theorem statements in some way: Encoding of $T$ is $[T]$.

$$\mathcal{L} = \{(T, 1^B) : \text{there exists a proof of } T \text{ of length at most } B\}$$

This language is NP-complete. (Exercise)

→ If you prove the Riemann hypothesis, can in theory publish a succinct ZK proof of knowledge. Only information revealed:

- You know a (correct) proof.
- An upper bound on the size of your proof.
- No information about the proof whatsoever. 👍

Tip: To maximize annoyance, publish ZK proof of knowledge for “Riemann Hypothesis is true OR Syracuse conjecture is true”.
Bonus question: If you read a proof of “P = NP” using the zk-SNARK from this course, can you deduce that P = NP?