ZK proofs for arbitrary circuits
Reductions

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

\[ \exists \text{ efficient } f \text{ such that } x \in \mathcal{L}' \text{ iff } f(x) \in \mathcal{L}. \]  
(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.
Commitment scheme

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

Prover $P$

$\theta \leftarrow$ $\$ permutation on $\mathbb{Z}_3$.

Verifier $V$

$\nu, \omega \leftarrow$ $\$ vertex set

commit on $\theta \circ c(x)$ for each vertex $x$.

accept iff $\theta \circ c(\nu) \neq \theta \circ c(\omega)$

open commit on $\theta \circ c(\nu), \theta \circ c(\omega)$

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 \times security parameter) \rightarrow gigantic proofs, verification times...
SNARKs

SNARK(?) tile by William Morris.
Finite Fields

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, $\Pr[ P(\alpha) \neq Q(\alpha) ] \geq 1 - \frac{d}{q}$.

→ 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\textsuperscript{th} Fibonacci number in $\mathbb{Z}_p$.

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

\textbf{“Solution”:} agree on whole computation circuit $\rightarrow$ encode as SAT problem $\rightarrow$ transform into 3-coloring problem $\rightarrow$ 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 **succint** and non-interactive.

**Succint Non-interactive Argument of Knowledge: SNARK.**

Also a fantastical beast by Lewis Caroll:
A new approach

Prosper computes the Fibonacci sequence $f_1, \ldots, f_{1000}$ 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 $i = 1, ..., 1000$.

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

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

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:

$$\Pr[ P(\alpha+2) - P(\alpha+1) - P(\alpha) = D(\alpha) \cdot H(\alpha) ] \leq 1000/p < 0.01$$

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 $e: \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{T}$ is a 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 $\mathbb{G} = \langle g \rangle$ 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 $h \in \mathbb{G}$ 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 $[\alpha], [\alpha^2], ..., [\alpha^{1000}]$.

→ 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.

Goal

**Solution:**

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

...and also encodings \([\gamma], [\gamma\alpha], [\gamma\alpha^2], \ldots, [\gamma\alpha^{1000}]\) for a uniform \(\gamma\).

\(\rightarrow\) Prosper can compute \([P(\alpha)], [\gamma P(\alpha)]\), 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 $[\alpha]$, $[\alpha^2]$, ..., $[\alpha^{1000}]$...

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

→ 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 \([\theta], [\theta((\alpha+1)^2-\alpha^2)], ..., [\theta((\alpha+1)^{1000}-\alpha^{1000})]\) 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 \([P(\alpha)], [H(\alpha)], \ldots\) 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 $O(\log(\text{circ. size}))$ 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 \(\otimes\).

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 \(v_i, w_i, y_i\):

\[v_i(r_j) = \begin{cases} 1 & \text{if value } i \text{ is left input to gate } j, \\ 0 & \text{if not.} \end{cases}\]

\[w_i(r_j) = \begin{cases} 1 & \text{if value } i \text{ is right input to gate } j, \\ 0 & \text{if not.} \end{cases}\]

\[y_i(r_j) = \begin{cases} 1 & \text{if value } i \text{ is output of gate } j, \\ 0 & \text{if not.} \end{cases}\]
Exemple.

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 \( c_1, \ldots, c_6 \) 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 $V = \Sigma c_i v_i$, $W = \Sigma c_i w_i$, $Y = \Sigma c_i y_i$, such that $T = (X-r_5)(X-r_6)$ divides $VW-Y$."

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

1. quadratic!
2. polynomial equality!

We know how to do that!

V. publishes $[\alpha^i]$, plus auxiliary $[\gamma^i\alpha]$ 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.
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.
- Further improvements since.

Can be made zero-knowledge at negligible additional cost.
A ZK application: e-Voting
e-Voting

Are going to see (more or less) **Helios** voting system.

[https://heliosvoting.org/](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 Belenos variant: [https://hal.inria.fr/hal-02066930/document](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

Nice description of Belenios variant: https://hal.inria.fr/hal-02066930/document
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?
Voter $i$
- owns voter secret sig. key $sk_i$
- wants to vote $v_i \in \{0,1\}$

Anobody generates
- votes: $c_i = \text{enc}_{mpk}(v_i)$
- signatures: $\text{sig}_{sk_i}(c_i)$

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

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

First attempt: 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 = Dec\( (c) = m = c_R / c_L^x \)

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

→ to prove decryption is correct, prove:

\[ \text{discrete log of } (c_L)^x \text{ in base } c_L = \text{discrete log of } \text{mpk}=g^x \text{ in base } g \]
\[ \iff (g, g^x, c_L, c_L^x) \in \text{Diffie-Hellman language} \]

→ 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 → NIZK

Note ZK property is crucial.
Voter $i$
owns voter secret sig. key $sk_i$
wants to vote $v_i \in \{0,1\}$

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

Anobody
checks

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

• votes: $c_i = enc_{mpk}(v_i)$
• signatures: $sig_{sk_i}(c_i)$

• encrypted result: $c = \sum c_i$
• result: $dec_{msk}(c) +$ DH proof

Now with verifiable election result

Problem 2: how about I vote $enc_{mpk}(1000)$?
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 $\text{enc}_{\text{mpk}}(v_i)$.
→ proof (hence vote) cannot be duplicated without knowing $\text{sk}_i$. 
Now with full verifiability

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

Anobody
  
  checks

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

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

votes: $c_i = \text{enc}_{mpk}(v_i) + \text{proof} \leq 1$

signatures: $\text{sig}_{sk_i}(c_i)$

encrypted result: $c = \sum c_i$

result: $\text{dec}_{msk}(c) + \text{DH proof}$

Bonus problem: replace decryption trustee by threshold scheme.