Overview of Modern SNARK Constructions

From this time we focus on the non-interactive proofs.

Definition:
SNARK: a succint proof that a certain statement is true.

Example statement: "I know an `m` s.t. `SHA256(m) = 0`"

The proof is **short** and **fast** to verify.

zk-SNARK: the proof reveals nothing about `m`.

The power ZKPs:

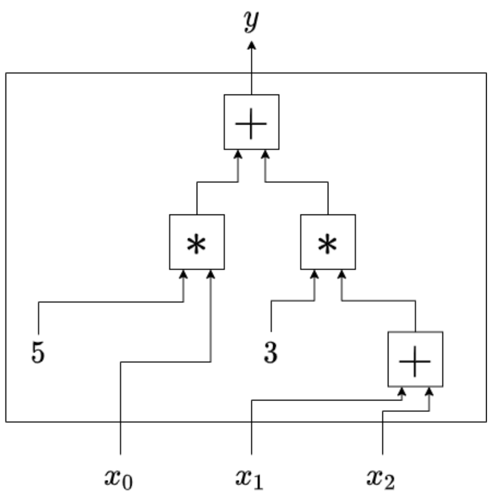
"*A single reliable machine (e.g. Blockchain) can monitor and verify the computations of a set of powerful machines working with unreliable software.*"

SNARK Components

Arithemtic circuits

We denote finite field $\mathbb{F} = \{0, ..., p-1\}$ as a finite field for some prime $p > 2$.

Definition:
Arithemtic circuit: $C : \mathbb{F}^n \to \mathbb{F}$ - it takes n elements in $\mathbb{F}$ and produces one element in $\mathbb{F}$.
It can be reason as:
- Directed asyclic graph where internal nodes are labeles as maths operations and inputs are labeleld as constants and variables.
- It defines an n-variate polynomial with an evaluation recipe

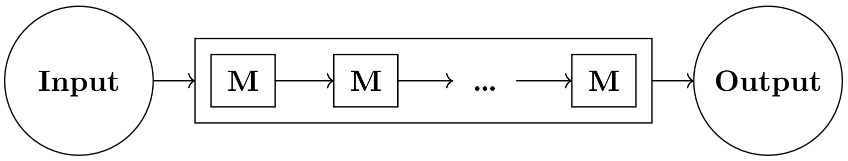


The above circuit can be represented as a n-variate polynomial $5x_0 + 3(x_1 + x_2)$

We denote $|C|$ = number of gates in a circuit C

We have two different circuit types:

- **Unstructured**: a circuit with arbitraty wires
- **Structured**: a circuit is built in layers of the same circuit layer that is repeated



M is often called a virtual machine (VM), you can think about it as one step of computation.

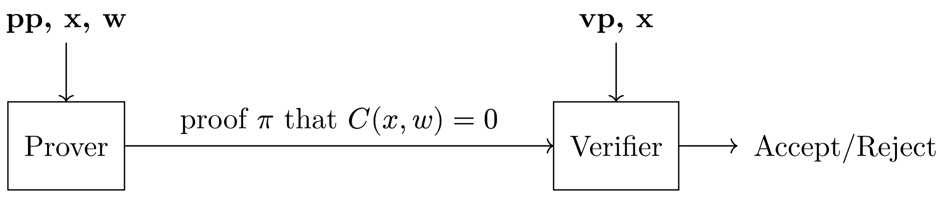
NARK: Non-Interactive ARgumebt of Knowledge

Given some arithemtic circuit: $C(x, w) \to \mathbb{F}$ where

- $x \in \mathbb{F}^n$ - a pubic statement
- $w \in \mathbb{F}^M$ - a secret wintess

Before executing the circuit there is a preprocessing setup: $S(C) \to$ public params (***pp***, ***pv***)

It takes a circuit description and produces public params.



Definition:
A **preprocessing NARK** is a triple (S, P, V):

- $S(C) \to$ public params *(pp, vp)* for prover and verifier
- $P(\text{pp}, x, w) \to$ proof π
- $V(\text{vp}, x, \pi) \to$ accept/reject

Side note: *All algorithms and adversary have access to a random oracle*

Definition:
Random oracle: is an oracle (black box) that responds to any unique queary with a uniformly distributed random response from its output domain. If the input is repeated, the same output is returned.

SNARK: Succinct Non-Interactive ARgumebt of Knowledge

We now impose additional requirements on the NARK

- **Completeness:** $\forall x, w \; C(x, w) = 0 \Rightarrow P[V(\text{vp}, x, P(\text{pp}, x, w)) = \text{accept}] = 1$
- **Knowledge soundness**: V accepts $\Rightarrow P$ knows w s.t. $C(x, w) = 0$.
  An extractor E can extract a valid w from P.
- (Optional) **Zero-Knowledge**: $(C, \text{pp}, \text{vp}, x\pi)$ reveal nothing new about w.

Definition:
A **succint preprocessing NARK** is a triple (S, P, V):

- $S(C) \to$ public params *(pp, vp)* for prover and verifier
- $P(\text{pp}, x, w) \to$ short proof π; $\text{len}(\pi) = \text{sublinear}(|w|)$
- $V(\text{vp}, x, \pi) \to$ accept/reject; **fast to verify**: $\text{time}(V) = O_\lambda(|x|, \text{sublinear}(|C|))$

In practice we have a stronger constraints:

Definition:
A **strongly succint preprocessing NARK** is a triple (S, P, V):

- $S(C) \to$ public params *(pp, vp)* for prover and verifier
- $P(\text{pp}, x, w) \to$ short proof π; $\text{len}(\pi) = \log(|w|)$
- $V(\text{vp}, x, \pi) \to$ accept/reject; **fast to verify**: $\text{time}(V) = O_\lambda(|x|, \log(|C|))$

We have a *Big O* notation with λ symbol, λ usually refers to some secret parameter that represents the level of security (e.g. length of keys, etc.). Therefore, the complexity is analyzed with repsect to the secret parameter.

You can notice that because the verifier need to verify the proof in shorter time than the circuit size, it does not have the time to read the circuit. This is the reason why we have the preprocessing step S. It reads the circuit C and generates a *summary* of it. Therefore, $|\text{vp}| \leq \log(|C|)$

Types of preprocessing Setup

Suppose we have a setup for some circuit C: $S(C; r) \to$ *public params (pp, vp)*, where r - random bits.

We have the following types of setup:

- **Trusted setup per circuit**: $S(C; r)$, r random bits must be kept private from the prover, otherwise it can prove false statements.
- **Trsuted universal (updatable) setup:** secret r is independent of C
  $S = (S_{\text{init}}, S_{\text{index}}) : S_{\text{init}}(\lambda; r) \to \text{gp}, S_{\text{index}}(\text{gp}; C) \to (\text{pp, vp})$ where
  - S_{init} - one time setup
  - S_{index} - deterministic algorith
  - *gp* - global params

The benefit of the universal setup that we can generate params for as many circuits as we want.

- **Transperent setup:** no secret data, $S(C)$

Overview of Proving Systems

	Size of proof	**Verifier time**	**Setup**	**Post-Quantum**
Groth'16	~ 200 bytes $O_{\lambda(1)}$	~ 1.5 ms $O_{\lambda(1)}$	trusted setup per circuit	no
Plonk & Marlin	~ 400 bytes $O_{\lambda(1)}$	~ 3 ms $O_{\lambda(1)}$	universal trusted setup	no
Bulletproofs	~ 1.5 KB $O_{\lambda(\log\lvert C\rvert)}$	~ 3 sec $O_{\lambda(\lvert C\rvert)}$	transperent	no
Bulletproofs	~ 100 KB $O_{\lambda(\log^2\lvert C\rvert)}$	~ 3 sec $O_{\lambda(\log^2\lvert C\rvert)}$	transperent	yes

Knowledge Soundness

If V accepts then P knows w s.t. $C(x, w) = 0$.

It means than we can w from P.

Definition:
(S, P, V) is (adaptively) **knowledge sound** for a circuit C if for every polynomial time adversary $A = (A_0, A_1)$ s.t.

$gp \leftarrow S_{\text{init}}()$, $(C, x, \text{st}) \leftarrow A_0(\text{gp})$, *(pp, vp)* $\leftarrow S_{\text{index}}(C)$, $\pi \leftarrow A_1(\text{pp}, x, \text{st})$:

$P[V(\text{vp}, x, \pi) = \text{accept}] > 1/10^6$ (non-negligible).

A acts as a malicious prover that tries to prove a statement without a knowledge of w. It is split into two algorithms A_0 and A_1

Given global parameters to A_0, the malicous prover generates a circuit, and a statement for which it tries to forge a proof for, the malicous prover also generates some internal state *st.*

Then, public params are generates from the circuit. Then malicious A_1 generates a forged proof π from prover params, a statements and an internal state.

If a malicious prover convinces a verifier with a probability grater than $1/10^6$, then there is an efficient **extractor** E (that uses A) s.t.

Definition:
$gp \leftarrow S_{\text{init}}()$, $(C, x, \text{st}) \leftarrow A_0(\text{gp})$, $w \leftarrow E(\text{gp}, C, x)$ (using A_1):

$P(C(x, w) = 0) > 1/10^6 - \varepsilon$ (for a negligible ε)

Building Efficent SNARKs

There are a general paradigm: two steps

- A functional commitment scheme. Requires Cryptographic assumptions
- A compatible interactive oracle proof. Does not require any assumptions

$$\begin{tikzcd}
\text{Functional commitment scheme} \arrow[dr] & & \\
& \text{Proving process} \arrow[r] & \text{SNARK for general circuits} \\
\text{IOP} \arrow[ur] & &
\end{tikzcd}$$

Functional Commitments

There are two algorithms:

- `commit(m, r) -> com` (r is chosen at random)
- `verify(m, com, r) -> accept/reject`

There are two informal properties:

- **binding**: cannnot produce `com` and two valid commitment openings for `com`
- **hiding**: `com` reveals nothing about commited data

Here we gave a standard hash construction:

Given some fixed hash function: $H : M \times R \to T$, we the two algorithms become:

- `commit(m, r): com := H(m, r)`
- `verify(m, com, r): accept if com = H(m, r)`

Then we can construct a functional commitment scheme.

Describing commitment to a function

Given some family of functions: $F = \{f : X \to Y\}$

The commiter acts as a prover. The prover chooeses some randonmness r and commits a descirption of a function f with r to a verifier. The function can decribed as a circuit, or as a binary code, etc. The verifier then sends $x \in X$, and the prover will respond with $y \in Y$ alongside a proof π, such that $f(x) = y$ and $f \in F$.

We can describe a commitment to a function family F using the following procedure (syntax):

- $\text{setup}(1^\lambda) \to \text{gp}$ - outputs global public parameteres *gp*.
- $\text{commit}(\text{gp}, f, r) \to \text{com}_f$ - produces a commitment to $f \in F$ with $r \in R$. It involves a **binding** (and optionall **hiding**, for ZK) committment scheme for F.
- eval(Prover P, verifier V) - an evaluation protocol between a prover and a verifier where for a given com_f and $x \in X, y \in Y$:
  - $P(\text{gp}, f, x, y, r) \to$ short proof π
  - $V(\text{gp}, \text{com}_f, x, y, \pi) \to$ accept/reject.

This evaluation protocol is a SNARK itself for the **relation**:
$f(x) = y, f \in F, \text{commit}(\text{gp}, f, r) = \text{com}_f$

For the setup, the public statements are com_f, x, y that are known to verifier. The prover is proving that it knows the description of f (a witness), and r s.t. the **relation** is true.

Commitment schemes

- **Polynomial**: a commit to a univariate $f(X) \in \mathbb{F}_p^{\leq d}[X]$. The family of functions is the set of all univarate polynomial function with degree of at most d
- **Multilinear**: a commit to a multilienar $f \in \mathbb{F}_p^{\leq 1}[X_1, ..., X_k]$. We a commiting to polynomial with multiple variables $X_1, ..., X_k$ but in each polynomial the degree is at most 1. e.g. $f(x_1, ..., x_k) = x_1 x_2 + x_1 x_4 x_5 + x_7$
- **Vector (e.g. Merke trees)**: a commit to $\vec{u} = (u_1, ..., u_d) \in F_p^d$. In the future, we would like to "open" any particular cell in the vector s.t. $f_{\vec{u}}(i) = u_i$. We can reason as we are commit to a function that is identified by a vector. Therefore, we would like to prove that given index i it evaluated to a cell u_i. Merkle trees are used for implementation of a vector commitmement.
- **Inner product** (aka inner product arguments - IPA): a commit to $\vec{u} \in F_p^d$. It commits to a function $f_{\vec{u}}(\vec{v}) = (\vec{u}, \vec{v})$ (inner product of u and v). We later prove that given some vector v for a function identified by a vector u, it results in an expected inner product value.

Polynomial Commitments

Suppose a prover commits to a polynomial $f(X) \in \mathbb{F}_p^{\leq d}[X]$.

Then the evaluation scheme **eval** looks as following:

For public $u, v \in \mathbb{F}_p$ (in finite field), prover can convince the verifier that the committed polynomial satisfies

$$\boxed{f(u) = v \text{ and } \deg(f) \leq d}$$

Note that the verifier knows (d, com_f, u, v). To make this proof a SNARK, the proof size and the verifier time should be $O_\lambda(\log d)$

Also note that trivial commitmement schemes are not a polynomial commitment. An example of a trivial commitment is as follows:

- *commit*$\Big(f = \sum_{i=0}^{d} a_i X^i, r\Big)$: outputs $\text{com}_f \leftarrow H((a_0, ..., a_d), r)$. We simply output a commitment to all coefficients of a polynomial (just a hash of them).
- *eval*: prover sends $\pi = ((a_0, ..., a_d), r)$ to verifier; and verifier accepts if $f(u) = v$ and $H((a_0, ..., a_d), r) = \text{com}_f$

The problem with this commitment scheme is that the proof π is not succinct. Specifically, the proof size and verification time are <u>linear</u> in d (but should be of $\leq \log d$).

Now let's look of the usage of the polynomial commitments. Let's start with an interesting observation.

For a non-zero $f \in \mathbb{F}_{p[X]}^{\leq d}$ and for $r \leftarrow \mathbb{F}_p$:

$$\boxed{(*)\ P[f(r) = 0] \leq d/p}$$

So, the proability that a randomly samples r in the finite field $\mathbb{F}_p$ is one of the roots of the degrees in a polynomial as at most the number of roots divided by the number of values in the field. Therefore for $r \leftarrow \mathbb{F}_p$: if $f(r) = 0$ then f is most likely identically zero.

Another useuful observation is:

SZDL lemma: (*) also holds for <u>multvariate</u> polynomial (where d is the total degree of f)

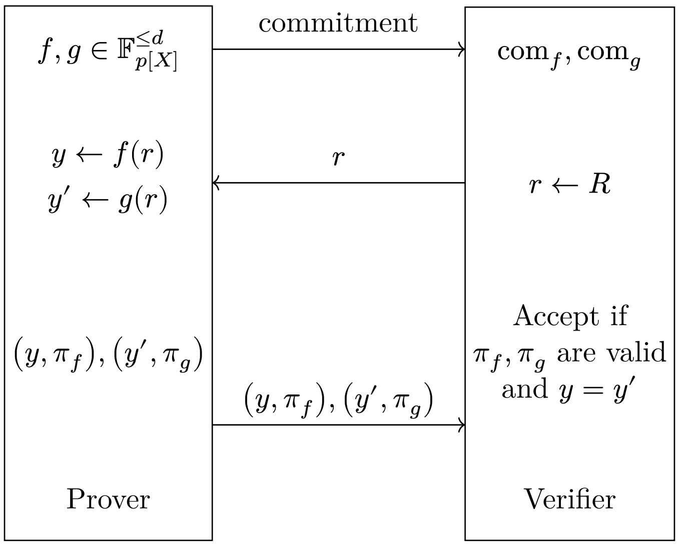
Proof: TODO

Based on the observaton aboce we can prove that two functions are identical.

Suppose p $\cong 2^{256}$ and $d \leq 2^{40}$ so that $\frac{d}{p}$ is negligible. Consequently, let $f, g \in \mathbb{F}_{p[X]}^{\leq d}$.
Then for $r \leftarrow \mathbb{F}_p$ if $f(r) = g(r)$ then $f = g$ with high proability. This holds because

$f(r) = g(r) \Rightarrow f(r) - g(r) = 0 \Rightarrow f - g = 0 \Rightarrow f = g$. This gives a simple equality test protocol.

Now let's look at the protocol of the two committed polynomials.


commitment
$f, g \in \mathbb{F}_{p[X]}^{\leq d}$
$\text{com}_f, \text{com}_g$
r
$y \leftarrow f(r)$
$y' \leftarrow g(r)$
$r \leftarrow R$
$(y, \pi_f), (y', \pi_g)$
Accept if π_f, π_g are valid and $y = y'$
$(y, \pi_f), (y', \pi_g)$
Prover
Verifier

Where π_f and π_g are the proves that the $y = f(r)$ and $y' = g(r)$ respectively.

Fiat-Shamir Transform

This allows us to make a protocol non-interactive. However, it isn't secure for every protocol.

We are going to start by using a cryptographic hash function $H : M \to R$. The prover will then use this function to generate verifier's random bits on its own using H.

The protocol becomes as follows:

- Let's $x = \left(\mathrm{com}_f, \mathrm{com}_g\right)$, and $w = (f, g)$
- The prover computes r, such that $r \leftarrow H(x)$
- The prover then computes $y \leftarrow f(r), y' \leftarrow g(r)$ and generates π_f, π_g
- The prover sends y, y', π_f, π_g to verifier.
- The verifier can now also compute $r \leftarrow H(x)$ from $\left(\mathrm{com}_f, \mathrm{com}_g\right)$ and verify the proof.

To prove knowledge soundness, we need to solve a theorem that the given protocol is a SNARK if
1. $d \,/\, p$ is negligible (where $f, g \in \mathbb{F}_p^{\leq d}[X]$)
2. H is modelled as a random oracle.

In practice, H is described as SHA256.

Internative Oracle Proofs (F-IOP)

Functional Commitment Schemes allows us to commit to a function whereas **Interactive Oracle Proofs** allows us to boost the commitment into a SNARK for general circuits.

As an example we can take a polynomial scheme for $\mathbb{F}_p^{\leq d}[X]$ and, using Poly-IOP, boost into a SNARK for any circuit C where $|C| < d$

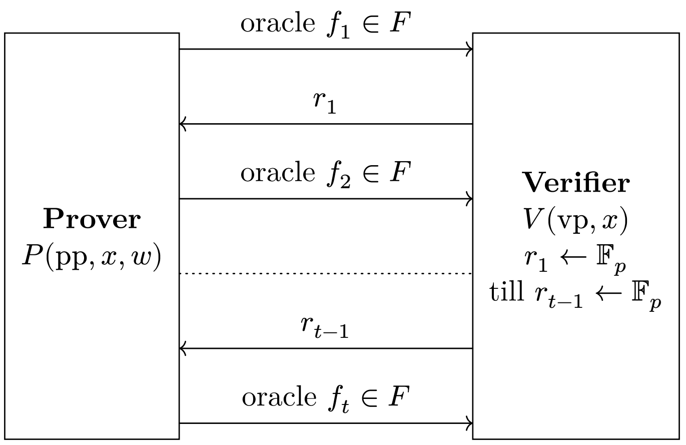
Let's define what F-IOP is.

Let $C(x, w)$ be some arithemtic circuit and let $x \in \mathbb{F}_p^n$.
F-IOP is a proof system that proves $\exists w : C(x, w) = 0$ as follows

Setup(C) $\to$ public params pp and vp. But public params for a verifier (vp) with contain a set of functions that will be replaced with function committments using Functional Commitment scheme. You can think of them as *oracles for functions in F*

The set of oracles generated by the verifier can be quearied by it at any time. Remember, that in real SNARK, the oracles are commitmements to functions.

From the prover side, the interaction looks as following:



The verifier then proceed the verifcation by computing: $\mathbf{verify}^{f_{-s},\dots,f_t}(\boldsymbol{x}, \boldsymbol{r_1}, ..., \boldsymbol{r_{t-1}})$. ($-s$ is offset index to account for additional functions before f_1 that was sent by the prover.)
It takes a statement x and all randomness that the verifier has sent to the prover., and it's given access to oracle functions that the verifier has and all the oracle functions that the prover sent as part of a proof. The verifier can evaluate any of the functions at any point and can decide whether to accept the proof or not.

The IOP flavour

Poly-IOP

- Sonic
- Marlin
- Plonk
- etc

Multilinear-IOP

- Spartan
- Clover
- Hyperplonk
- etc

Vector-IOP

- STARK
- Breakdown
- Orion
- etc

(**Poly-IOP** + Poly-comit || **Multilinear-IOP** + Multilinear-Commit || **Vector-IOP** + Merkle) + **Fiat-Shamir Transform** = **SNARK**

Reading

- a16z reading list