ZERO KNOWLEDGE PROOFS

by Marija Mikić

Types of preprocessing setups

- ➢ **Trusted setup per circuit**: $S(C; r)$, random r must be kept secret from the prover;
- ➢ **Trusted universal (updatable) setup**: secret r is independent from C;

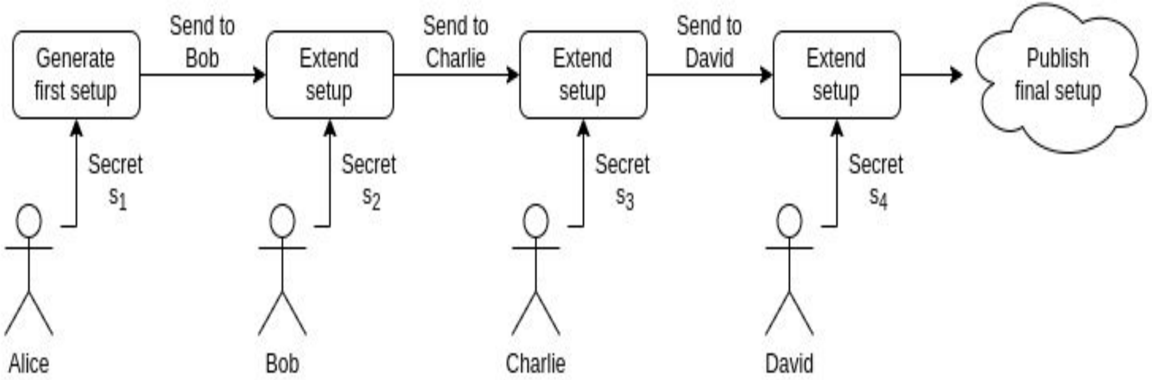
  $S=(S\_init, S\_index)$: $S\_init=S(\lambda; r) \rightarrow pp$ one time, $S\_index(pp, C) \rightarrow (Sp, Sv)$
- ➢ **Transparent setup**: $S(C)$ does not use secret data.

A trusted setup ceremony is a procedure that is done once to generate a piece of data that must then be used every time some cryptographic protocol is run.

There's More 👉

Trusted setups

Zcash



*

You can find more information on these links:

[Link 1 >]

[Link 2 >]

Trusted setups

Example:

Cyclic group $\mathbb{G} = \{0, G, 2*G, \ldots,(p-1)G\}$ of order p.

Setup(λ) $\rightarrow$ pp:

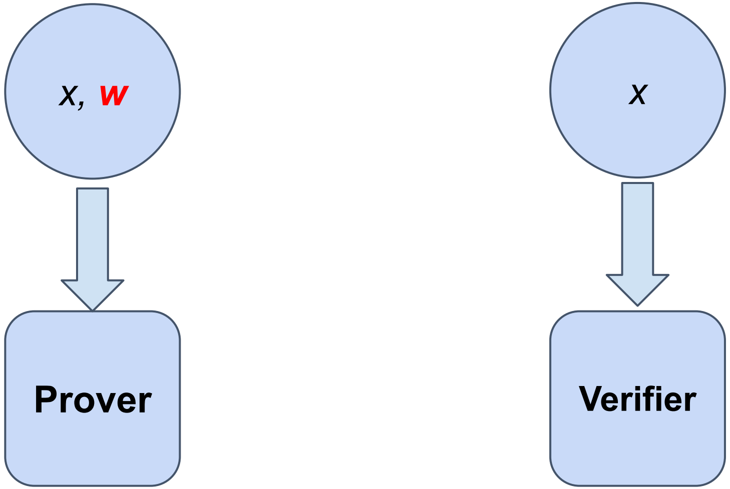
- ➢ Sample random $s \in \boldsymbol{F}$;
- ➢ pp=$(G, sG, s\hat{}2*G, \ldots,s\hat{}d*G) \in G\hat{}\{d+1\}$;
- ➢ Delete s (trusted setup).

There's More 👉

Argument system

Public arithmetic circuits: *C(x,w)* $\rightarrow \boldsymbol{F}$
x-**public** statement in $\boldsymbol{F}$^*n*
w-**secret** witness in $\boldsymbol{F}$^*m*

Prover wants to convince Verifier that he knows *w* such that *C(x,w)*=0

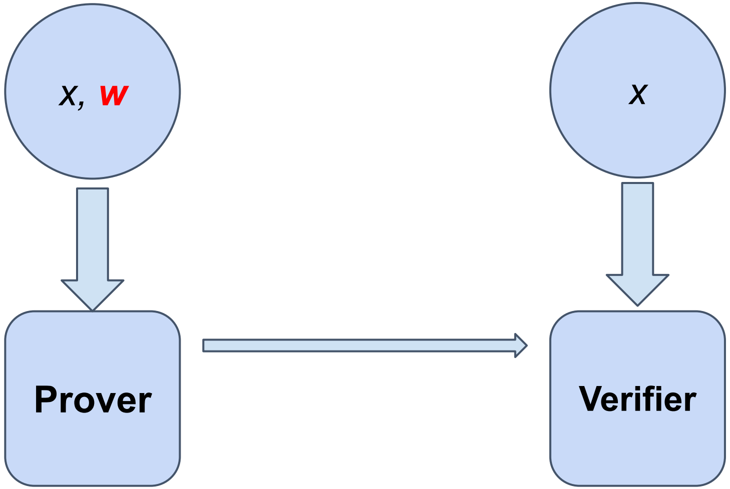


There's More 👉

Argument system

Public arithmetic circuits: $C(x,w) \to \boldsymbol{F}$
x-**public** statement in $\boldsymbol{F}$^n
w-**secret** witness in $\boldsymbol{F}$^m

Prover wants to convince Verifier that he knows *w* such that *C(x,w)*=0

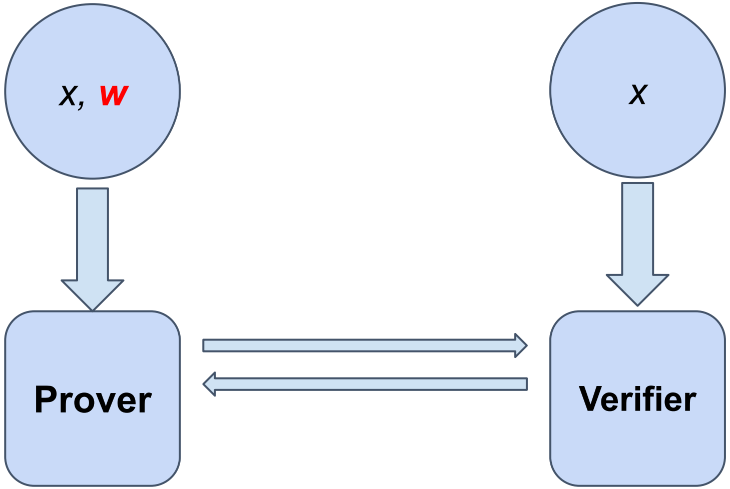


There's More 👉

Argument system

Public arithmetic circuits: $C(x,w) \to \boldsymbol{F}$
x-**public** statement in $\boldsymbol{F}$^n
w-**secret** witness in $\boldsymbol{F}$^m

Prover wants to convince Verifier that he knows *w* such that *C(x,w)*=0

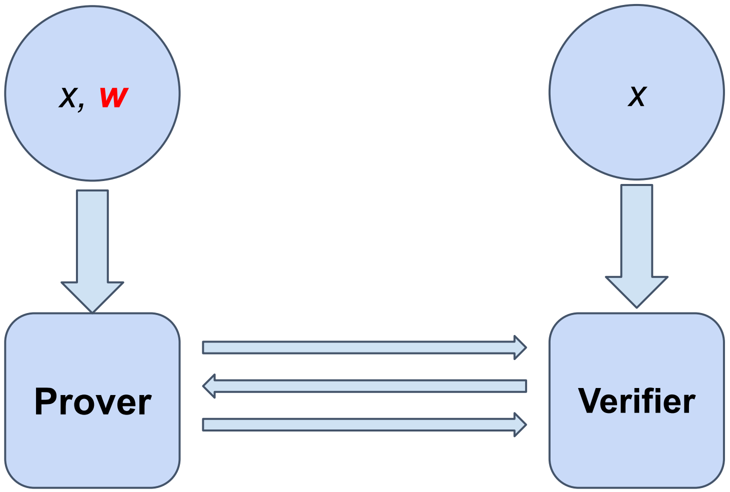


There's More 👉

Argument system

Public arithmetic circuits: $C(x,w) \to \mathbb{F}$
x-**public** statement in $\mathbb{F}^n$
w-**secret** witness in $\mathbb{F}^m$

Prover wants to convince Verifier that he knows *w* such that *C(x,w)*=0

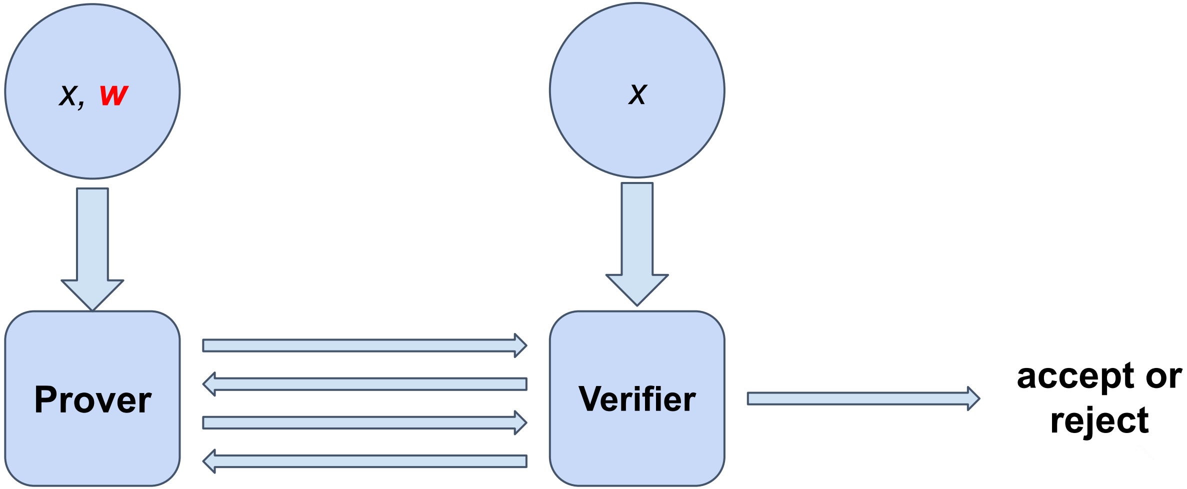


There's More 👉

Argument system

Public arithmetic circuits: $C(x,w) \to \mathbb{F}$
x-**public** statement in $\mathbb{F}^n$
w-**secret** witness in $\mathbb{F}^m$

Prover wants to convince Verifier that he knows
w* such that *C(x,w)=0

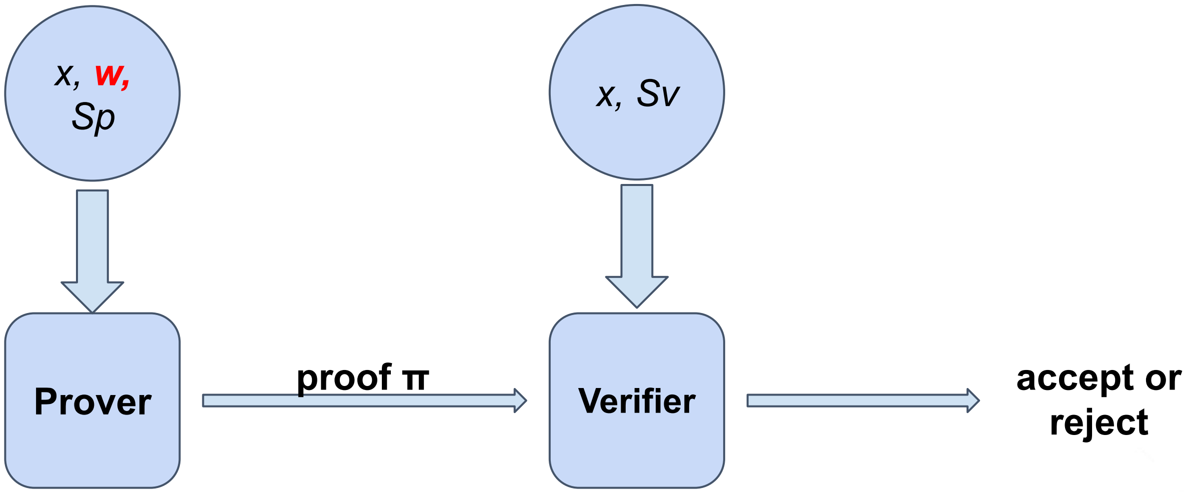


There's More 👉

Non-Interactive Preprocessing argument system

Public arithmetic circuits: $C(x,w) \rightarrow \mathbb{F}$
x-**public** statement in $\mathbb{F}^n$
w-**secret** witness in $\mathbb{F}^m$

Preprocessing setup: $S(C;r) \rightarrow (Sp, Sv)$



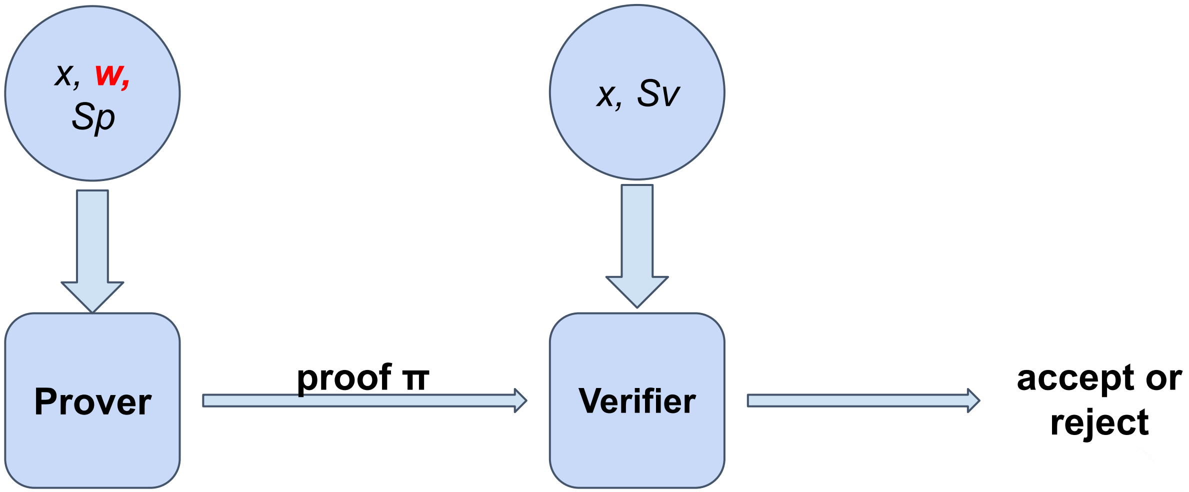
There's More 👉

Non-Interactive Preprocessing argument system

Public arithmetic circuits: $C(x,w) \rightarrow \mathbb{F}$
x-**public** statement in $\mathbb{F}$^n
w-**secret** witness in $\mathbb{F}$^m

Preprocessing setup: $S(C;r) \rightarrow (Sp, Sv)$

Completeness:
$\forall$ x,w: $C(x,w)=0$ => $\Pr[V(x,Sv,P(x,w,Sp))=\text{accept}]=1$



There's More 👉

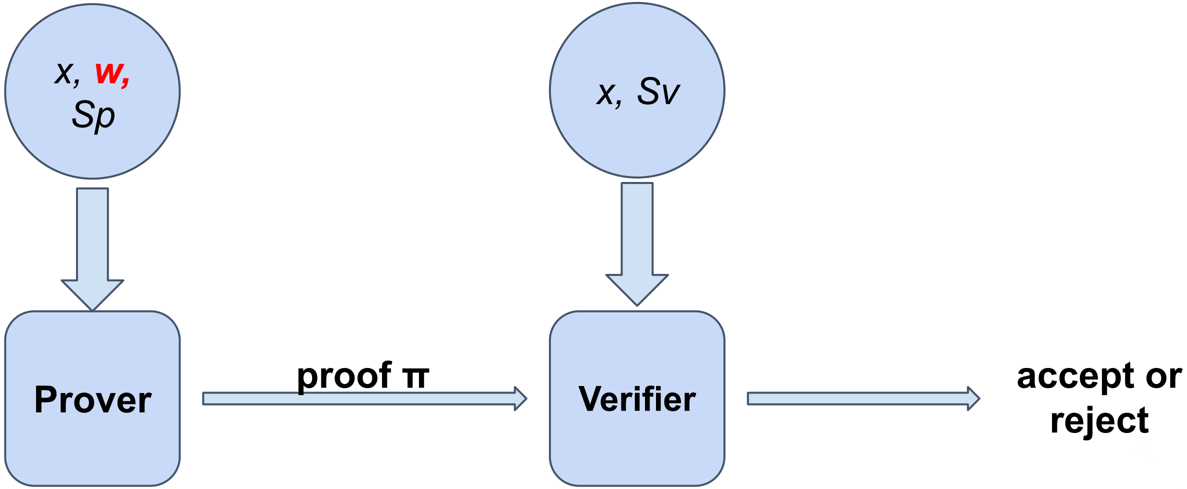
Non-Interactive Preprocessing argument system

Public arithmetic circuits: $C(x,w) \rightarrow F$
x-**public** statement in Fˆ*n*
w-**secret** witness in Fˆ*m*

Preprocessing setup: ***S(C;r)*** **→** ***(Sp, Sv)***

Completeness:
∀ *x,w*: *C(x,w)=0* => Pr[V(*x,Sv,*π(*x,w,Sp*))=accept]=1

Soundness:
V accept proof => Prover "knows" *w* such that *C(x,w)*=0



There's More 👉

Non-Interactive Preprocessing argument system

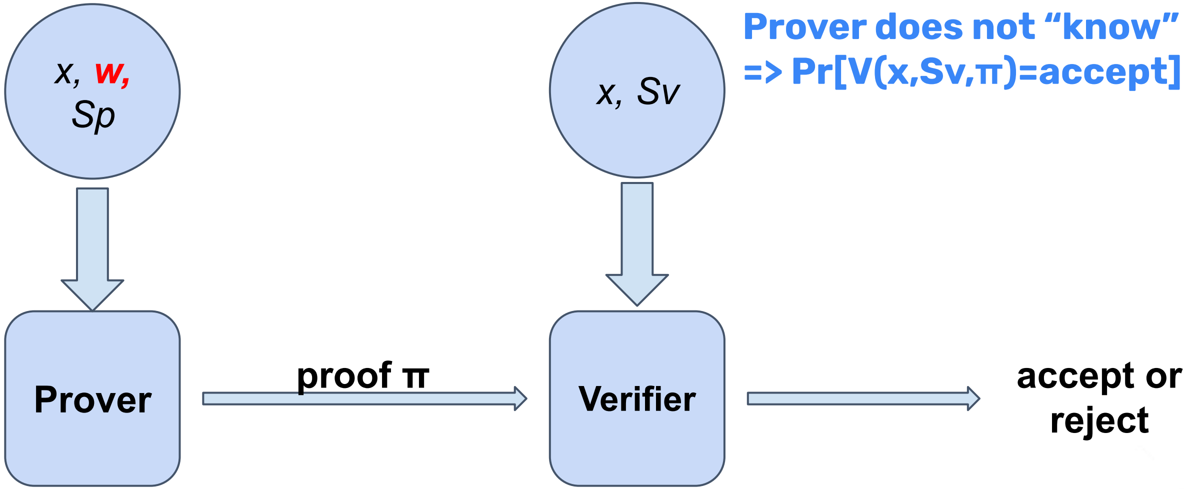
Public arithmetic circuits: *C(x,w)* $\rightarrow \mathbb{F}$
x-**public** statement in $\mathbb{F}$^*n*
w-**secret** witness in $\mathbb{F}$^*m*

Preprocessing setup: *S(C;r)* $\rightarrow$ *(Sp, Sv)*

Completeness:
∀ *x,w*: *C(x,w)=0* => Pr[V(*x,Sv*,π(*x,w,Sp*))=accept]=1

Soundness:
V accept proof => Prover "knows" *w* such that *C(x,w)*=0

Prover does not "know" *w* such that *C(x,w)*=0
=> Pr[V(x,Sv,π)=accept] < negligible



There's More 👉

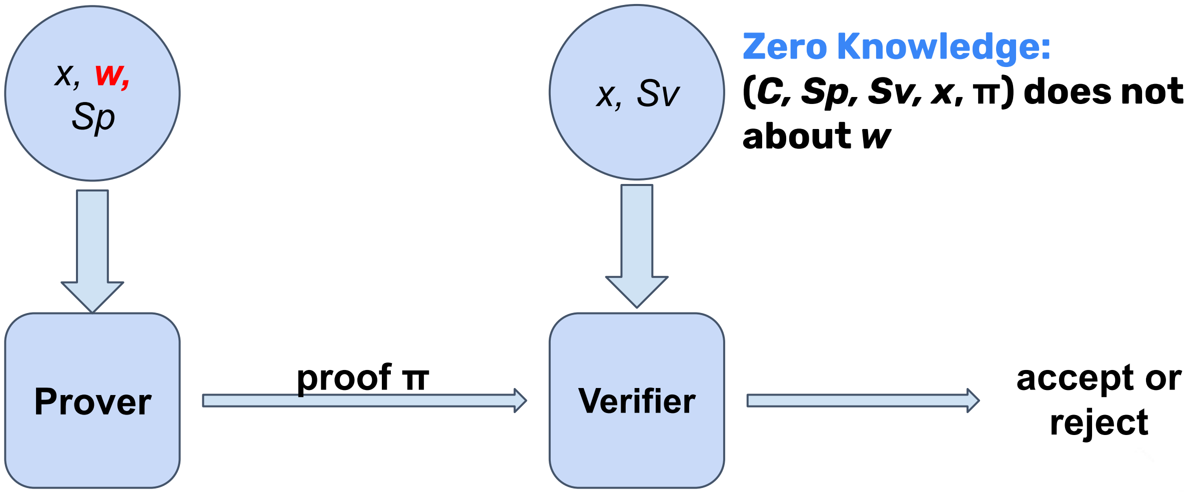
Non-Interactive Preprocessing argument system

Public arithmetic circuits: $C(x,w) \rightarrow F$
x-**public** statement in $F\hat{}n$
w-**secret** witness in $F\hat{}m$

Preprocessing setup: $S(C;r) \rightarrow (Sp, Sv)$

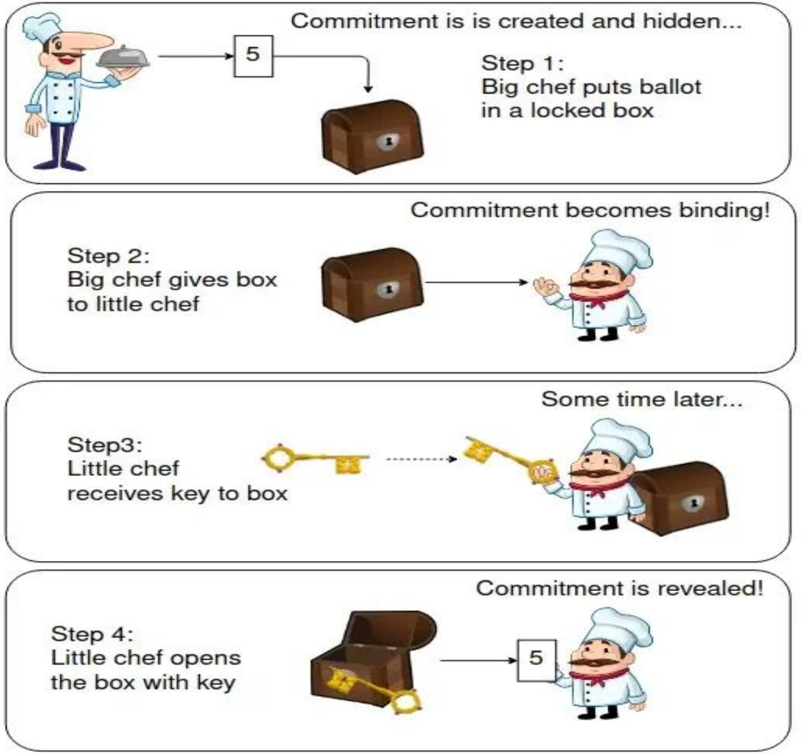
Completeness:
$\forall$ ***x,w***: ***C(x,w)=0*** **=> Pr[V(*x,Sv*,π(*x,w,Sp*)=accept]=1**

Soundness:
V accept proof => Prover "knows" *w* such that *C(x,w)*=0



Zero Knowledge:
(*C, Sp, Sv, x,* π) does not reveal nothing about *w*

Commitments



A **commitment scheme** is a cryptographic primitive that allows one to commit to a chosen value (or chosen statement) while keeping it hidden to others, with the ability to reveal the committed value later.

There's More 👉

Commitments

Example 1: Exam

Example 2: Aviator

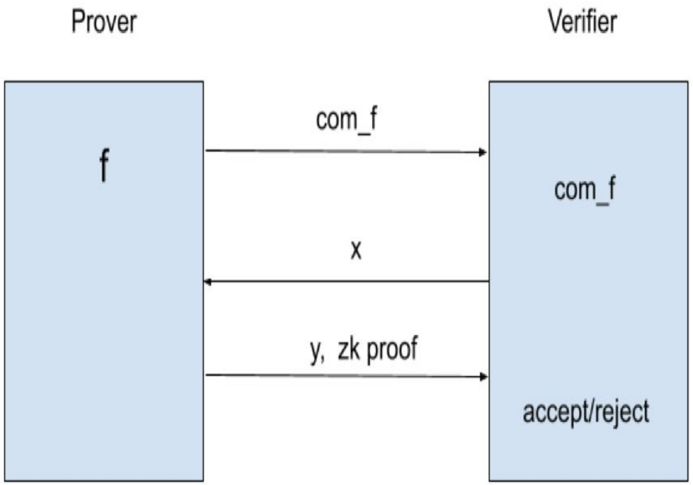
*

You can find more information on these links:

[Link 1 >]

[Link 2 >]

Functional commitments



Binding: cannot produce two valid openings for com.

Hiding: com reveals nothing about committed function.

There's More 👉

Functional commitments

A functional commitment scheme FC for $\boldsymbol{F}$ is a tuple (Setup, Commit, Eval):

- ➔ **Setup(λ) –> pp**
- ➔ **Commit(pp, f) –> com_f**

  Binding (and optionally hiding) commitment scheme for $\boldsymbol{F}$

- ➔ **Eval (prover P, verifier V)**: for given com_f, x and y:

  P(pp, f, x, y) –> proof

  V(pp, com_f, x, y, proof) –> accept or reject

There's More 👉

Polynomial commitments

Polynomial commitments schemes:

- ➔ **KZG (trusted setup) -** Kate Zaverucha Goldberg 2010;
- ➔ Dory'20;
- ➔ Dark'20;
- ➔ FRI (uses hash function) - Fast Reed-Solomon IOP of Proximity.

*

You can find more information on this link:

Link 1 >

KZG

Group $\mathbb{G}$={0, *G, 2G, 3G, …, (p-1)G*} of order *p*.

- <u>Setup(λ) –> *pp*:</u>

  - Sample random *s* from F;

  - *pp=(H0=G, H1=s*G, H2=s*s*G, …, Hd= s*…*s*G*);

  - Delete *s*.

- Commit (*pp, f*) –> com_f where com_*f:=f(s)*G*

KZG

Group $\mathbb{G}$={0, *G, 2G, 3G, …, (p-1)G*} of order *p*.

➔ <u>Setup(λ) —> *pp*:</u>

- Sample random *s* from $\boldsymbol{F}$;

- *pp*=(*H0=G, H1=s*G, H2=s*s*G, …, Hd= s*…*s*G*);

- Delete *s*.

➔ Commit (*pp, f*) —> com_f where com_*f*:=*f(s)*G*

f(x)=f0+f1x+…+fdx…*x* => com_f= *f0*H_0+ f1*H1+…+fd*Hd*

KZG

Group $\mathbb{G}$={0, *G, 2G, 3G, …, (p-1)G*} of order *p*.

➔ <u>Setup(λ) —> *pp*:</u>

- Sample random *s* from $\boldsymbol{F}$;

- *pp=(H0=G, H1=s*G, H2=s*s*G, …, Hd= s*…*s*G*);

- Delete *s*.

➔ Commit (*pp, f*) —> com_f where com_*f:=f(s)*G*

f(x)=f0+f1x+…+fdx…*x* => com_f=*f0*H_0+ f1*H1+…+fd*Hd=f0*G+f1*s*G+…+fd*s*…*s*G*

KZG

Group $\mathbb{G}$={0, *G, 2G, 3G, …, (p-1)G*} of order *p*.

➔ Setup(λ) —> *pp*:

- Sample random *s* from $\boldsymbol{F}$;

- *pp=(H0=G, H1=s*G, H2=s*s*G, …, Hd= s*…*s*G)*;

- Delete *s*.

➔ Commit (*pp, f*) —> com_f where com_*f:=f(s)*G*

f(x)=f0+f1x+…+fdx…*x* => com_f=*f0*H_0+ f1*H1+…+fd*Hd=f0*G+f1*s*G+…+fd*s*…*s*G*

*=(f0+f1*s+…+fd*s*…*s)*G=f(s)*G*

Schwartz - Zippel lemma

For $0 \neq f \in \mathbb{F}$^(≤d)[x] and random $r \in \mathbb{F}$ than **Pr[*f(r)*=0]≤*d/p***.

Suppose *p*≈2^256 and *d*≤2^40 then *d/p* is negligible.

For different *f,g* $\in \mathbb{F}$^(≤d)[x] and random $r \in \mathbb{F}$ than Pr[*f(r)*=*g*(*r*)] ≤*d/p*.

So if ***f(r)-g(r)*=0 w.h.p. *f(x)=g(x)***.

KZG

➔ Eval (Prover P, Verifier V):

f(x0)=y ⇔ x0 is a root of f-y ⇔ (x-x0) divides f-y ⇔ exist q such that

q(x)*(x-x0) = f(x)-y

Prover		**Verifier**
Compute q(x)		accept if
Compute com_q	**y, π:=com_q** ------------------------------>	**(s -x0)*com_q=com_f-y*G**

KZG

➔ Eval (Prover P, Verifier V):

f(x0)=y ⇔ x0 is a root of f-y ⇔ (x-x0) divides f-y ⇔ exist q such that

q(x)*(x-x0) = f(x)-y

Prover		**Verifier**
Compute q(x)		accept if
Compute com_q	**y, π:=com_q** ------------------------------->	**(s -x0)*com_q=com_f-y*G**
		((s-x0)*q(s))*G= (s-x0)*q(s)*G=(f(s)-y)*G

KZG

- Eval (Prover P, Verifier V):

f(x0)=y ⇔ x0 is a root of f-y ⇔ (x-x0) divides f-y ⇔ exist q such that

q(x)*(x-x0) = f(x)-y

Prover		**Verifier**
Compute q(x)		accept if
Compute com_q	**y, π:=com_q** ------------------------------->	**(s -x0)*com_q=com_f-y*G**
		But verifier does not know s!!!!

KZG

→ Eval (Prover P, Verifier V):

$f(x0)=y$ ⇔ $x0$ is a root of $f-y$ ⇔ $(x-x0)$ divides $f-y$ ⇔ exist q such that

$q(x)*(x-x0) = f(x)-y$

Prover		**Verifier**
Compute q(x)		accept if
Compute com_q	**y, π:=com_q** ------------------------------>	**(s -x0)*com_q=com_f-y*G**

answer: Elliptic curve pairing!!!!

Thank you!