Uvod u Circom

Uvodni pojmovi

Trusted setup

Ceremonija generisanja javnih parametara, koji se koriste za generisanje i verifikaciju dokaza

Ograničenja (Constraints)

Algebarske jednačine nad konačnim poljem koje svaka validna witness tabela mora da zadovolji.

ZK-friendly heš funkcije

Poseidon heš funkcija je mnogo pogodnija za aritmetička kola od Keccak heš funkcije, jer generiše znatno manje ograničenja, što utiče na veličinu dokaza

Circom

Šta je Circom?

Circom je HDL (Hardware Description Language) za pisanje aritmetičkih kola i ograničenja.

Postoje i moderniji jezici, npr. Noir. U takvim jezicima kod često izgleda bliže običnom programiranju, pa je lakše pisati kompleksnije aplikacije.
Circom je, sa druge strane, bliži samom kolu i ograničenjima.

Pošto direktnije kontrolišemo strukturu kola, Circom je često pogodniji kada želimo da optimizujemo broj ograničenja i napravimo efikasnije dokaze.

Korak po korak

1 **Pišemo kolo**

Definišemo signale, pomoćne vrednosti i ograničenja.

2 **Kompajliramo**

Kolo se prevodi u R1CS (Rank-1 Constraint System) format

3 **Trusted setup**

Kod Groth16 se setup generiše za svako kolo, a kod PLONK-a imamo univerzalni setup

4 **Witness**

Za privatne i javne inpute računa se kompletna witness tabela

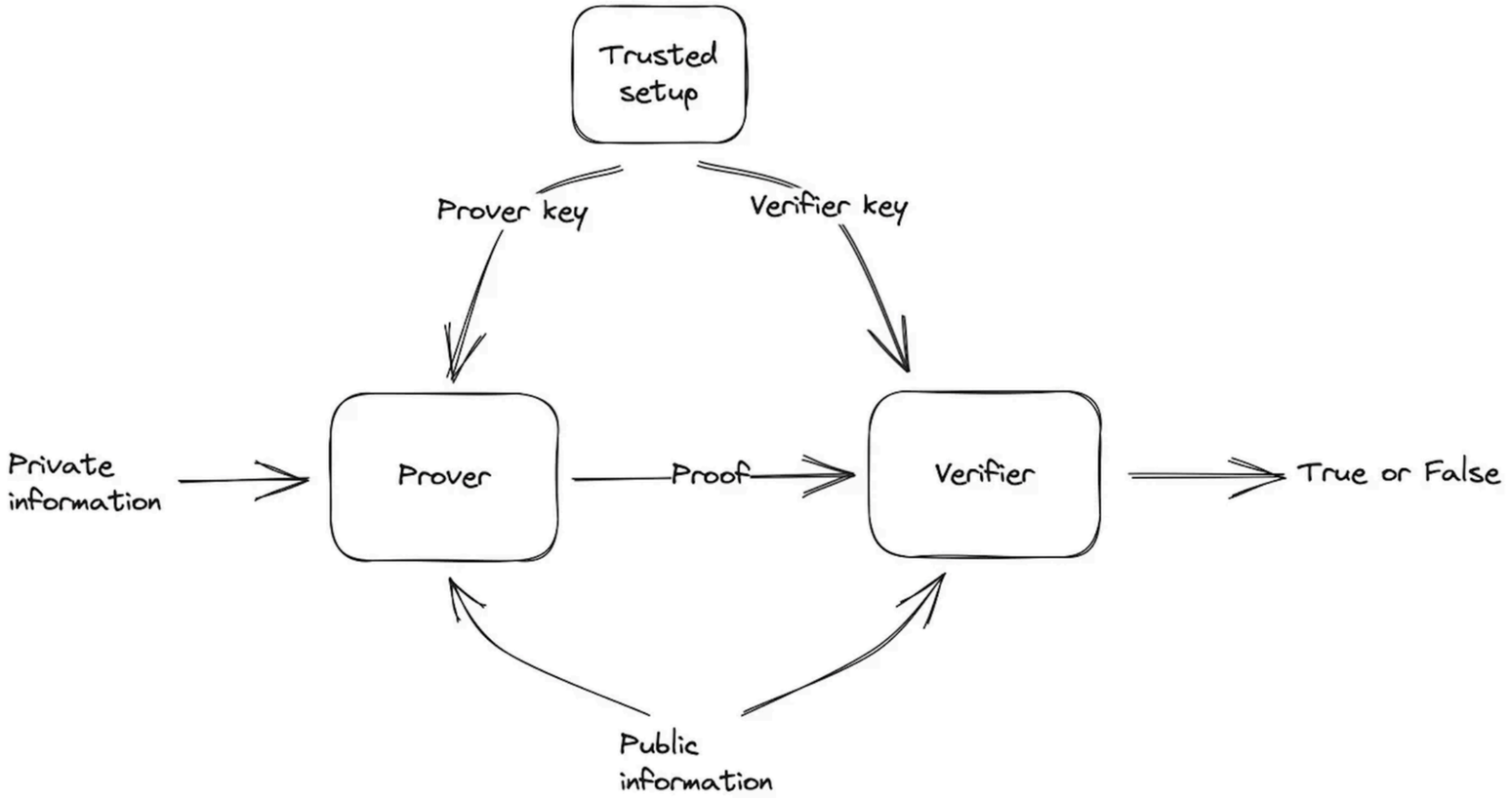
5 **Dokaz**

Prover koristi witness i trusted setup da napravi ZK dokaz

6 **Verifikacija**

Verifier proverava dokaz koristeći trusted setup i izazov koji je poslao prover-u

Skica protokola



Signali i promenljive

Circom dokumentacija je dostupna ovde: LINK

Circom radi nad krivom nad poljem F_p, gde je
p = 21888242871839275222246405745257275088548364400416034343698204186575808495617

- signal je deo kola i witness-a; može da učestvuje u constraint proveri. Nad signalima se mogu pisati aritmetički izrazi u kojima se pojavljuju +, - i *. Constraintovi moraju biti kvadratnog oblika, pa se složenije operacije grade pomoću dodatnih signala i gotovih template-a.
- var je pomoćna promenljiva.

Signals:

```
signal input a;
signal output b;
signal c;
```

Variables:

```
var x = 5;
```

Aritmetički operatori

Operator	
+	sabiranje po modulu p
-	oduzimanje po modulu p
*	množenje po modulu p
/	množenje inverzom od b, ako postoji
\	celobrojno deljenje
%	ostatak pri deljenju
**	stepenovanje

Logički i bitovski operatori

Operator	
==	jednakost
!=	različitost
<, >	poređenje
<=, >=	poređenje
&&, ||, !	logički operatori

Operator	
&	bitovsko AND
|	bitovsko OR
^	bitovsko XOR
~	komplement
<<, >>	pomeranje bitova

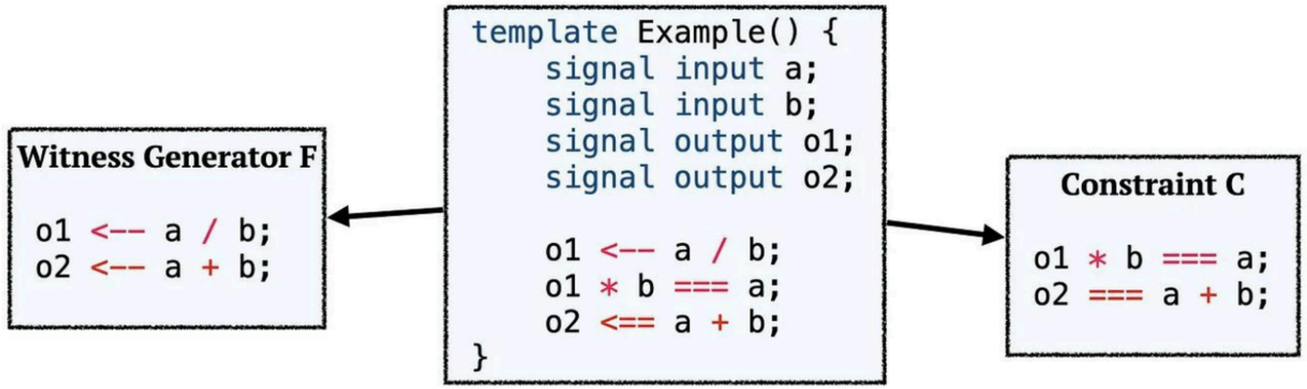
Naredbe kontrola toka

• Broj constraintova u kolu mora biti fiksiran pre računanja witness-a.
• Zato if/else grananja koja generišu constraintove ne smeju zavisiti od nepoznatih input signala.
• Isto važi za for i while petlje: ako petlja generiše constraintove, broj iteracija mora biti poznat u trenutku kompajliranja kola (tj. u trenutku kada private i public inputi nisu poznati).

• **Zamena za if/else grananje su multiplekseri, koji omogućavaju da se obe mogućnosti predstave u kolu, a zatim se odgovarajuća vrednost bira na osnovu ulaznih signala**

Konstrukcija ZKP-a u Circom-u

Operator	
<--	operator dodele
===	dodaje constraint
<==	dodela + constraint



Pravila za pisanje constraintova

Dozvoljeno

- konstantni izraz: a <== 2;
- linearni izraz: a <== b + 3;
- kvadratni izraz: a <== b * c + 5;

Nije dozvoljeno direktno

```
a <== b * c * d; (non-quadratic constraint)
```

Mora se razložiti na pomoćne signale:

```
tmp <== b * c;
a   <== tmp * d;
```

Circomlib

Circomlib je biblioteka koja sadrži veliki broj gotovih kola, kao što su komparatori, heš funkcije, operacije na eliptičkim krivama itd.

- Templejti koje ćemo najčešće koristiti:
- comparators — komparatori;
- mux1, mux2 — multiplekseri;
- poseidon — Poseidon heš funkcija;
- escalarmulfix, escalarmulany — skalarno množenje tačke na eliptičkoj krivoj.

Circomlib je dostupan ovde: LINK

zkREPL

Interaktivno okruženje za eksperimentisanje sa Circom kolima, dostupno je ovde: LINK

Čemu služi?

- pisanje i testiranje Circom kola u browseru;
- brza provera witness-a i grešaka u constraint-ima;
- nije zamena za ozbiljan lokalni development setup.

Inpute pišemo u komentaru, u sledećem formatu:

```
/* INPUT = {
  "a": "5",
  "b": "8"
} */
```

Primeri koje ćemo raditi

Membership proof

Dokaz pripadnosti skupu bez otkrivanja identiteta

Privatni ključ → javni ključ

Dokaz da znamo privatni ključ za zadati javni ključ

Pogađanje broja

Dokazujemo da znamo tajni broj bez njegovog otkrivanja

Range proof

Dokaz da je tajna vrednost u zadatom opsegu

Pogađanje broja

Posmatramo igru između dva igrača:

- Prvi igrač tajno bira broj **x**.
- Drugi igrač pokušava da pogodi taj broj i javno kaže svoj pokušaj **y**.

Želimo da napravimo ZK kolo koje proverava da li je drugi igrač pogodio broj, a da se pri tome, tajni broj **x** ne otkriva javno. Dakle, prvi igrač je prover, a drugi igrač je verifier.

Naivno rešenje: kolo koje proverava samo jednakost tajnog broja i javne pretpostavke.
U ovom slučaju:
• privatni input je tajni broj **x**;
• javni input je pretpostavka drugog igrača **y**;
• kolo proverava uslov: **x=y**.
Ako je uslov ispunjen, dokaz prolazi i verifier zna da je pokušaj tačan.
Ako uslov nije ispunjen, dokaz ne prolazi.

Da li ovo rešenje ima neki nedostatak?

Pogađanje broja

Prvi igrač nije unapred javno „zaključao” svoj broj. To znači da bi mogao da menja **x** prilikom svakog pokušaja drugog igrača. Zato nam treba mehanizam kojim se prvi igrač unapred obavezuje na izabrani broj, ali bez njegovog otkrivanja.

Da bi prvi igrač dokazao da nije promenio broj nakon pogađanja, uvodimo commitment.
Prvi igrač tajno bira broj **x** i nasumičnu vrednost **salt**. Zatim javno objavljuje:
C = H(**x, salt**), gde je H heš funkcija.
Vrednost **C** je javni commitment. **Ona „zaključava” prvog igrača na izabrani broj, ali ne otkriva sam broj.**

Uloga salt vrednosti je veoma važna. Ako bi commitment bio samo H(**x**), a broj **x** dolazi iz malog skupa, drugi igrač bi mogao unapred da izračuna heševe svih mogućih brojeva i otkrije **x**.

Pogađanje broja

Kada drugi igrač javno kaže svoju pretpostavku **y**, prvi igrač pravi ZK dokaz da zna vrednosti **x** i **salt** takve da važi:
C = H(**x, salt**)
i vrši se provera da li je pokušaj tačan: **x** = **y**.

U ovom kolu:
• privatni inputi su: **x, salt**
• javni inputi su: **C, y**
• kolo proverava: **C** = H(**x, salt**) i **x** = **y**

Ako dokaz prođe, verifier zna dve stvari:
• prvi igrač zaista zna broj i salt koji odgovaraju ranije objavljenom commitment-u;
• pokušaj drugog igrača je tačan.

Range proof

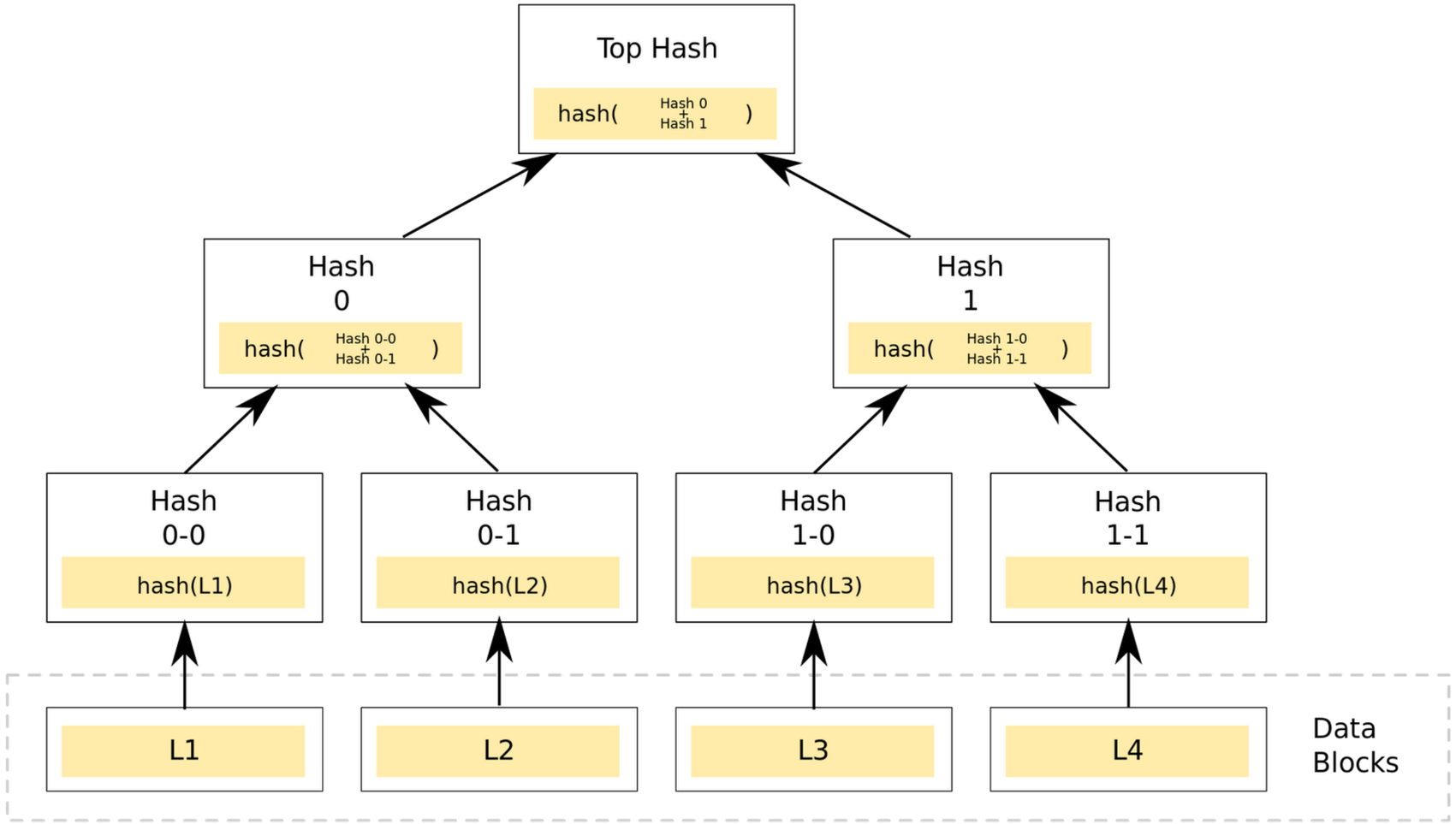
Ideja

Range proof nam omogućava da dokažemo da neki broj **x** pripada nekom zadatom intervalu, pri čemu ne otkrivamo **x**, a granice intervala mogu biti javne.

Primeri

- dokaz da osoba ima bar 18 godina bez otkrivanja godine rođenja
- dokaz da je lokacija u dozvoljenom regionu bez otkrivanja tačne lokacije
- dokaz da račun ima dovoljno sredstava bez otkrivanja celog stanja

Merkle stablo



Merkle stablo

• Merkle stablo je binarno heš stablo u kome se vrednost svakog unutrašnjeg čvora računa kao heš vrednosti njegovih neposrednih potomaka.

• Listovi stabla predstavljaju pojedinačne elemente skupa, odnosno heširane podatke koje želimo da organizujemo i proveravamo.

• Koren stabla, odnosno root hash, predstavlja kratak „otisak” celog skupa podataka. Ako se promeni makar jedan element u listu, menja se heš tog lista, zatim heševi čvorova iznad njega, pa se na kraju menja i koren stabla.

• Praktično je nemoguće izmeniti podatke u stablu tako da korenski heš ostane isti.

Zašto je Merkle stablo važno u ZK?

- Možemo javno objaviti samo root
- Prover privatno čuva list i putanju do korena
- ZK dokaz proverava da putanja zaista vodi do javnog root-a
- Tako dokazujemo pripadnost skupu bez otkrivanja kog tačno elementa

Membership proof

Dokaz da tajna vrednost pripada nekom skupu.

Šta dokazujemo?

Prover dokazuje da zna vrednost **x** takvu da se hash(**x**) nalazi kao list u Merkle stablu, kao i putanju kojom se od hash(**x**) dobija javno poznati root.

private inputi: vrednost **x**, putanja do root-a stabla, niz suseda na toj putanji

public input: root stabla

Primeri

- pravo glasa bez otkrivanja identiteta
- ostavljanje anonimnih recenzija
- dozvola za pristup servisu bez otkrivanja dodatnih detalja naloga