Zero Knowledge Proofs and ZCash

Shubham Sahai Srivastava

CS-731: Lecture 17

Department of Computer Science and Engineering, IIT Kanpur.

In the last class …

- Zero Knowledge proof introduction
- All languages in NP have a zero knowledge proofs
- ZKP of Graph 3-coloring
- Issues with using interactive ZKP in blockchains
- zk-SNARKS: introduction
- zk-SNARKS: an example

zkSNARKS: Example

$$x^3 + x + 5 == 35$$

Function:

```
def qeval(x):
    y = x**3
    return x + y + 5
```

Flattened Code:

$sym_1 = x * x$

$y = sym_1 * x$

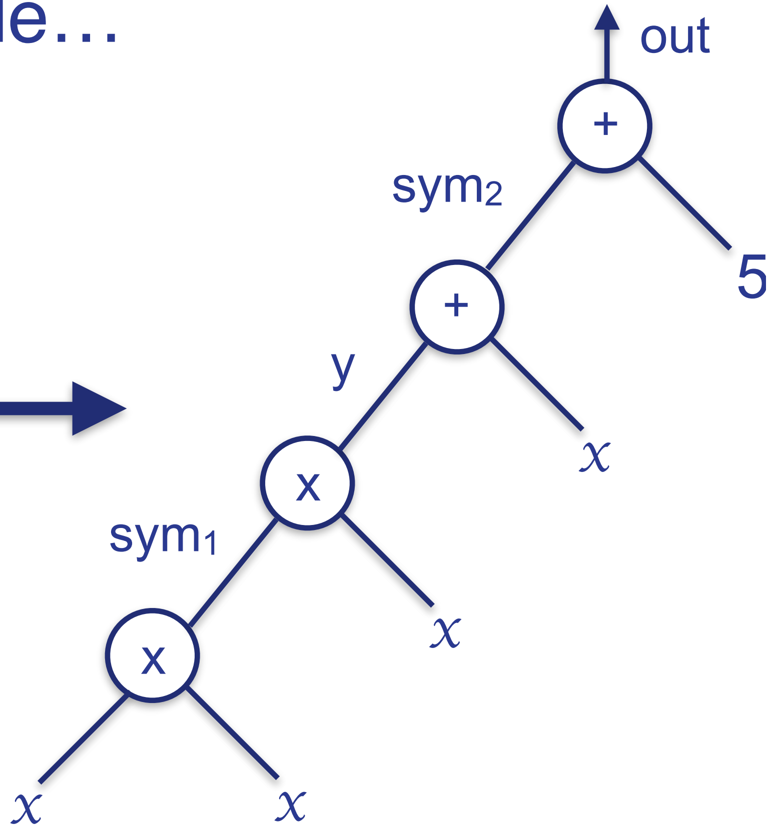
$sym_2 = y + x$

$\sim out = sym_2 + 5$

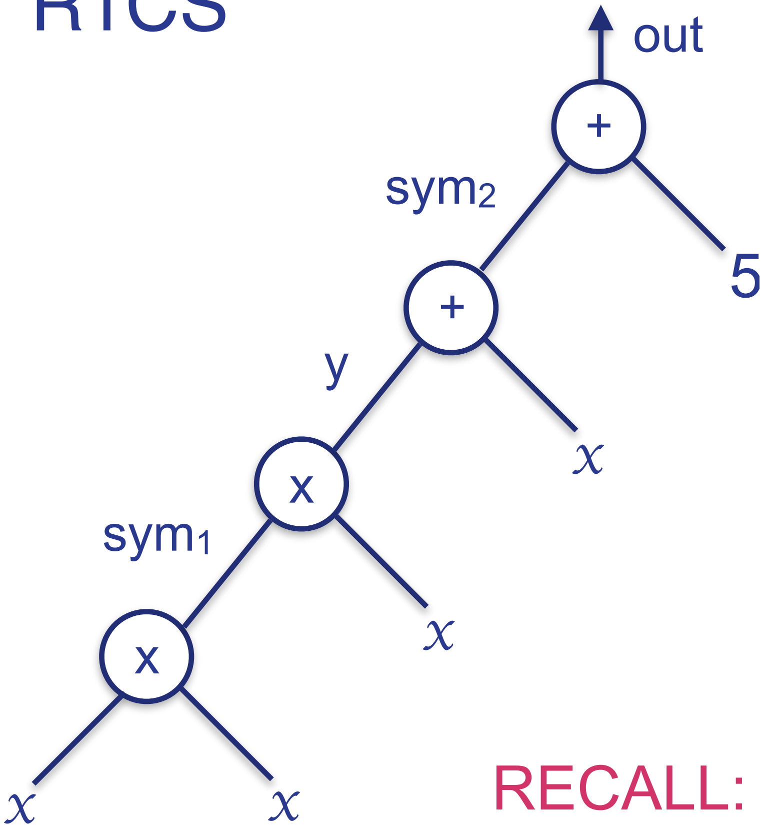
Arithmetic Circuit from code…

Flattened Code:

$$
\begin{aligned}
sym_1 &= x * x \\
y &= sym_1 * x \\
sym_2 &= y + x \\
\sim out &= sym_2 + 5
\end{aligned}
$$



R1CS



Fix a variable ordering amongst <u>all</u> the variables of the circuit

[ONE, x, sym_1, y, sym_2, out]

RECALL: Every gate corresponds to a constraint

R1CS Constraints …

[ONE, x, OUT, sym_1, y, sym_2]

A	B	C
[0, 1, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0]	[0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0]	[0, 1, 0, 0, 0, 0]	[0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 1, 0]	[1, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 1]
[5, 0, 0, 0, 0, 1]	[1, 0, 0, 0, 0, 0]	[0, 0, 1, 0, 0, 0]

$$\boxed{\mathbf{S} = [1, 3, 35, 9, 27, 30]}$$

R1CS

- Goal is to come up with s, that satisfies all the R1CS simultaneously.

- To verify: Solve each R1CS equation corresponding to a solution vector.

- QAP: Implement the same logic as R1CS, but using polynomials instead of dot products.

$$\boxed{A(x) * B(x) = C(x)}$$

R1CS to QAP

A

[0, 1, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0]
[0, 1, 0, 0, 1, 0]
[5, 0, 0, 0, 0, 1]

$A_1(x)$

B

[0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]

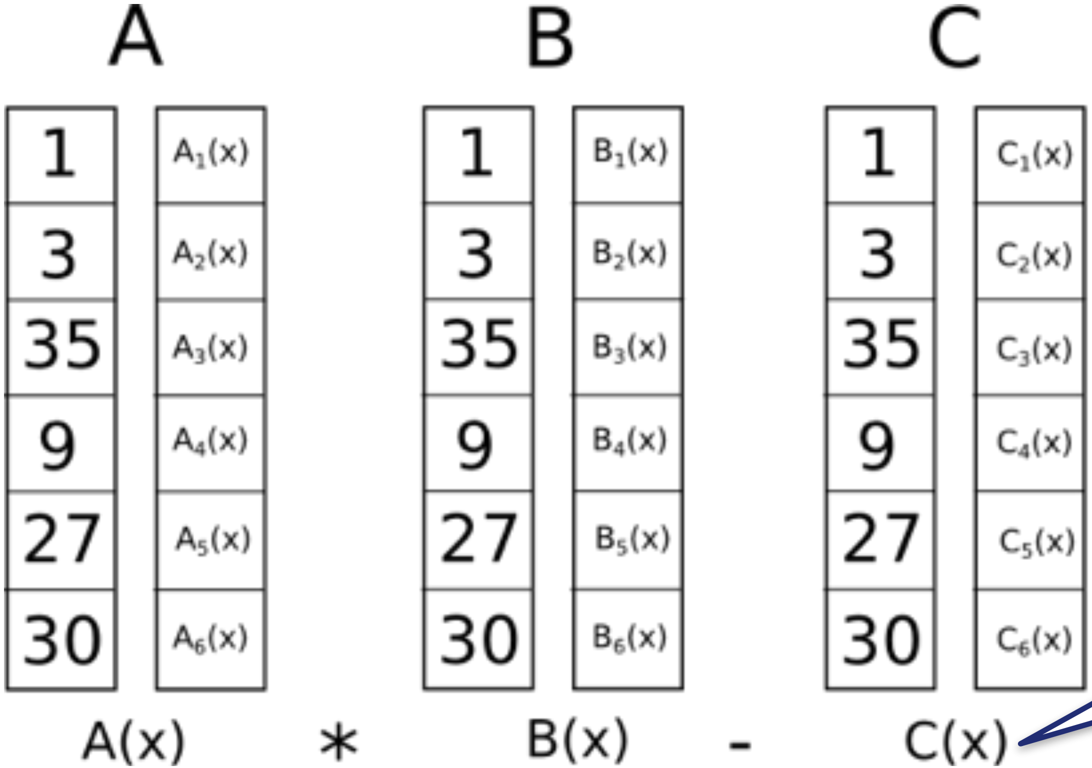
$B_2(x)$

C

[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 0, 0]

$C_4(x)$

Quadratic Arithmetic Programs (QAPs)



$A(x) = \Sigma\, s_i A_i(x)$

$B(x) = \Sigma\, s_i B_i(x)$

$C(x) = \Sigma\, s_i C_i(x)$

Should be 0 for all x corresponding to circuit gates.

QAP …

$$A(x) * B(x) - C(x) = 0, \quad \forall\, x \in [1, |Gates|\,]$$

$$\Updownarrow$$

$A(x) * B(x) - C(x)$ is divisible by $\prod_{i=1}^{|Gates|} (x-i)$

Target Polynomial (T(x))

QAP …

$$T(x) = \prod_{i=1}^{|Gates|} (x-i)$$

$$\boxed{A(x) * B(x) - C(x) = H(x) * T(x)}$$

$$A(x) = \Sigma\, s_i\, A_i(x)$$

$$B(x) = \Sigma\, s_i\, B_i(x)$$

$$C(x) = \Sigma\, s_i\, C_i(x)$$

If we know the solution vector (**s**), then:

- We know: s_i, $\forall i \in [1, |Vars|\,]$
- For a given **e**, <u>can compute</u> : A(e), B(e), C(e)
- T(x) is <u>public</u>, so we can compute H(e)

Proof

- The evaluation point e is chosen randomly by the *Verifier* and sent to the *Prover*.
- A(e), B(e), C(e), H(e) is the desired proof.
- The point e is send in an “hidden” form.
- Polynomials have to be evaluated “blindly”.

Road Ahead …

- Homomorphic Hiding Scheme
- Blind evaluation of polynomial
- Knowledge of Coefficient Assumption
- Making blind evaluation verifiable

Homomorphic Hiding (HH)

- A Homomorphic Hiding can be defined as E(.):
    - Given E(x), its hard to find x
    - Given $x \neq y$, $E(x) \neq E(y)$ … different inputs lead to different outputs
    - If one knows E(x) and E(y), she can compute arithmetic expressions in x and y … given E(x) and E(y) compute E(x+y)

HH: an example

- Consider the group Z_p^*:
  - Elements of the group: {1, 2, 3, … p-1}
  - Its a cyclic group … ∃ a generator *g*
  - For large p, discrete log problem is believed to be hard … given $h \in Z_p^*$, its difficult to find $a \in \{0, \ldots p\text{-}2\}$, st, $g^a = h \ (mod\ p)$
  - Exponents add up when elements are multiplied …
    $g^a \times g^b = g^{a+b\ (mod\ p-1)}$

HH: an example

- Homomorphic Hiding can be defined as $E(x) = g^x$:
    - Given $E(x)$, its hard to find x … discrete log is hard
    - Given $x \neq y$, $E(x) \neq E(y)$ … different inputs lead to different outputs
    - If one knows $E(x)$ and $E(y)$, she can compute arithmetic expressions in x and y … given $E(x)$ and $E(y)$ compute $E(x+y)$

Blind Evaluation of Polynomials

$$P(x) = a_0 + a_1.x + a_2.x^2 + a_3.x^3 \ \dots + a_d.x^d$$

Evaluating P at a point $s \in F_p$,

$$P(s) = a_0 + a_1.s + a_2.s^2 + a_3.s^3 \ \dots + a_d.s^d$$

P is a linear combination of 1, s, s^2, … s^d

Given $E(x) = g^x$, $E(y) = g^y$, a, b

$$E(ax + by) = g^{(ax + by)} = E(x)^a . E(x)^b$$

Blind Evaluation of Polynomials

$P(x) = a_0 + a_1.x + a_2.x^2 + a_3.x^3 \;\ldots + a_d.x^d$

$s \in F_p$

$$\xrightarrow{P(x)}$$

Bob computes: P(s)

Blind Evaluation of Polynomials

$P(x) = a_0 + a_1.x + a_2.x^2 + a_3.x^3 \; \ldots + a_d.x^d$

$s \in F_p$

$\xleftarrow{\quad s \quad}$

$\xrightarrow{\quad E(P(s)) \quad}$

Alice computes: E(P(s))

Blind Evaluation of Polynomials

Given $E(x) = g^x$, $E(y) = g^y$, a, b

$$E(ax + by) = g^{(ax + by)} = E(x)^a . E(x)^b$$

$$P(x) = a_0 + a_1.x + a_2.x^2 + a_3.x^3 \ \dots + a_d.x^d$$

$s \in F_p$

$E(1), E(s), E(s^2), \dots E(s^d)$

$E(P(s))$

Alice computes: $E(P(s))$

Ensuring Alice is honest…

How do we ensure that Alice sent $E(P(s))$?

$P(x)$

$s \in F_p$

$E(1), E(s), E(s^2), \ldots E(s^d)$

$E(P(s))$

Alice computes: $E(P(s))$

Knowledge of Coefficient Assumption

$|G| = p$,
discrete log is hard

For $\alpha \in F_p$; $a,b \in G$ is an α-pair if $b = \alpha.a$

$\alpha \in F_p$
$a \in G$
Computes:
$b = \alpha.a$

(a,b)

Alice has to compute (a',b') an α-pair

$(a',b') = (\beta.a\ ,\ \beta.b)$; $\beta \in F_p$

Knowledge of Coefficient Assumption

$|G| = p$,
discrete log is hard

For $\alpha \in F_p$; $a, b \in G$ is an α-pair if $b = \alpha . a$

$\boxed{P \text{ is a linear combination of } 1, s, s^2, \ldots s^d}$

$\alpha \in F_p$
$a \in G$
Computes:
$b = \alpha . a$

$(a_0,b_0), (a_1,b_1), \ldots (a_d,b_d)$

(a', b')

$a' = c_0.a_0 + c_1.a_1 \ldots c_d.a_d$
$b' = c_0.b_0 + c_1.b_1 \ldots c_d.b_d$
where, $c_0, c_1, \ldots c_d \in F_p$

Stitching everything together…

$P(x) = a_0 + a_1.x + a_2.x^2 + a_3.x^3 \ \dots + a_d.x^d$

$\alpha \in F_p$

$s \in G$

$E(1), E(s), E(s^2), \dots E(s^d),$

$E(\alpha.1), E(\alpha.s), E(\alpha.s^2), \dots E(\alpha.s^d)$

$(a',b') = \big(E(P(s)), \alpha.E(P(s))\big)$

Alice computes: $E(P(s)), \alpha.E(P(s))$

Finally,

$$T(x) = \prod_{i=1}^{|Gates|} (x-i)$$

$$\boxed{A(x) * B(x) - C(x) = H(x) * T(x)}$$

$A(x) = \Sigma\, s_i\, A_i(x)$

$B(x) = \Sigma\, s_i\, B_i(x)$

$C(x) = \Sigma\, s_i\, C_i(x)$

If we know the solution vector (**s**), then:

- We know: s_i, $\forall i \in [1, |Vars|\,]$
- For a given **e**, can compute : $A(e)$, $B(e)$, $C(e)$
- $T(x)$ is public, so we can compute $H(e)$

ZCash

Decentralized Anonymous Payments

ZeroCash: Decentralized anonymous payment from Bitcoin: Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, Madars Virza, *IEEE S&P 2014*

- Proposed in 2014, improving on ZeroCoin.
- Construct decentralized anonymous payment scheme, that hides sender and receiver address as well as the amount.

Goals

- Issues with bitcoin:
    - De-anonymization
    - Purchase/Cash Flow visible
    - Amount being transacted
    - Can leverage competitors
    - Transaction Graph analysis

Designing a anonymous payment scheme, which hides the sender/ receiver details as well as the amount.

User Anonymity with fixed value coins

$c := \mathrm{COMM}_r\,(m)$

Each coin has a same value, lets say 1 BTC

- Picks a random serial number sn and nonce r
- $cm := \mathrm{COMM}_r(sn)$
- **c** $:= (\, r, sn, cm)$
- tx_{MINT} containing *cm* is sent to the ledger
- Its added to the ledger if Bob has payed 1 BTC to a backing escrow pool

$c := COMM_r\,(sn)$

User Anonymity with fixed value coins

Each coin has a same value, lets say 1 BTC

cm_1	cm_2	cm_3	.	.	.	.	.	.	cm_α

CMList

- tx_{SPEND}
  - Serial number *sn*
  - *zkSNARK* proof for the NP statement: *I know a **r** such that $COMM_r(sn)$ appears in CMList.*
- If sn does not appear in the ledger, then its a valid txn.

$c := COMM_r(sn)$

User Anonymity with fixed value coins

cm_1	cm_2	cm_3	.	.	.	.	.	.	cm_α

CMList

- tx_{MINT} : add a new commitment to CMList
- tx_{SPEND} : redeem a coin in CMList
  - Does not reveal anything about **r**
  - Finding which **cm** is spent is difficult.
- Can represent CMList in the form of Merkle tree with root rt.
- tx_{SPEND} includes *zkSNARK* proof for the NP statement: *I know a* ***r*** *such that* $COMM_r(sn)$ *appears in the leaf of a Merkle tree with root rt.*

Issues…

- Coin commitment **cm** of a coin **c** is a commitment to **sn**
- u_A created **c** and sends to u_B, then
  - its not anonymous, as u_A can see when the coin is spent
  - its not safe, as u_A can spend the coin again
- Coins are created of fixed values
  - It reveals the amount being transferred
  - Transferring amount not in multiples of 1 BTC not supported.

Extending coins for DAP

(a_{pk}, a_{sk})

address keypair

- To mint a coin with value **v** a user u:
  - Randomly choses a serial number **sn** using a random nonce β: $sn := PRF_{a_{sk}}(\beta)$.
- u commits to the tuple: (a_{pk}, sn, β) in two phases:
  - u computes $k := COMM_r\ (a_{pk} \,||\, \beta)$, for a random r
  - u computes $cm := COMM_s\ (v \,||\, k)$, for a random s
- The minting results in a coin: $(a_{pk}, v, \beta, r, s, cm)$
- $tx_{MINT} := (v, k, s, cm)$

Any user can verify the tx_{MINT} by computing $COMM_s(v \,||\, k)$. But it reveals nothing about the owner or serial number of the coin.

Spending coins : *Pour* Operation

Pour takes a set of input coins to be spent and "pours" their value into a set of freshly output coins, st, output value = input value.

(a_{pk}, a_{sk})
address keypair

- Alice wants to spend coin: $(a_{pk}, v, \beta, r, s, cm)$ to produce two coins
  - c_1 and c_2 with values v_1 and v_2, st $v_1 + v_2 = v$
  - targeted to address: $b_{pk,1}$ and $b_{pk,2}$
- Then Alice, for each $i \in \{1,2\}$ does:
  - computes $k_i := COMM_{ri}(b_{pk,i} \,||\, \beta_i)$, for a random ri
  - u computes $cm_i := COMM_{si}(v_i \,||\, k_i)$, for a random si
- This yields two new coins:
  - $c_1 = (b_{pk,1}, v_1, \beta_1, r1, s1, cm_1)$
  - $c_2 = (b_{pk,2}, v_2, \beta_2, r2, s2, cm_2)$

Spending coins : *Pour* Operation

(a_{pk}, a_{sk})
address keypair

Next Alice produces zkSNARK proof for the following NP statement:

- Given MT root rt, serial number **sn**, and coin commitments cm_1 and cm_2, I know coins c, c_1, c_2 and a_{sk} such that:
  - The coins are well formed.
  - a_{sk} matches a_{pk}.
  - **sn** is computed correctly
  - cm appears as a leaf of MT with root rt
  - $v_1 + v_2 = v$
- $tx_{POUR} := (rt, sn, cm_1, cm_2, Proof_{POUR})$ is appended to the ledger

ZCash

- Algorithms:
  - Setup
  - Create Address
  - Mint
  - Pour
  - VerifyTransaction
  - Receive

- Security
  - Anonymity
  - Ledger Indistingushability … nothing revealed beside public information
  - Balance … cant own more money that received or minted

Network simulation
third-scale Bitcoin network on EC2

Bitcoind + Zerocash hybrid currency

`libzerocash`
provides DAP interface

Statement for zkSNARK
Hand-optimized

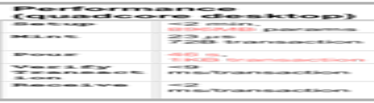
`libsnark`
zkSNARK

SCIPR Lab

Instantiate Zerocash primitives and parameters

`bitcoind`
.

Performance (quadcore desktop)

`Setup`	<2 min, 896MB params
`Mint`	
`Pour`	46 s, 1KB transaction
`Verify Transaction`	<9 ms/ transaction
`Receive`	<2 ms/ transaction

Questions?

Thank You!