zk-SNARKs

Panagiotis Grontas

NTUA-advTCS

01/06/2017

From theory to practice...

zkSnark

Zero **K**nowledge **S**uccinct **N**on **I**nteractive **A**rguments Of **K**nowledge

Use

Efficiently verify the correctness of computations without executing them

Applications

- Verify cloud computations (centralised, decentralised)
- Anonymous bitcoin (ZCash)

Application Model

- A client owns input u (e.g query)
- A server owns a private input w (e.g. private DB)
- The client wishes to learn $z = f(u, w)$ for a function f known to both
- Client: computation correctness (integrity)
- Server: private input confidentiality

Client: its computing power should be confined to the bare minimum of sending u and receiving z

What zk-Snarks offer

- **Z**ero **K**nowledge: The client (verifier $\mathcal{V}$) learns nothing but the validity of the computation
- **S**uccinct: The proof is tiny compared to the computation
  - the proof size is constant $O_\lambda(1)$ (depends only on the security parameter λ)
  - verification time is $O_\lambda(|f| + |u| + |z|)$ and does not depend on the running time of f
- **N**on **I**nteractive: The proofs are created without interaction with the verifier and are publicly verifiable strings
- **A**rguments: Soundness is guaranteed only against a computationally bounded server (prover $\mathcal{P}$)
- of **K**nowledge: The proof cannot be constructed without access to a witness

Position in the complexity landscape...

- $NP = PCP[O(logn), O(1)]$
- One-Way Functions $\Rightarrow NP \subseteq ZK$ (Goldreich, Micali, Wigderson) (ZKP for 3-COL)
- We can use PCP to construct ZK proofs (in theory)
- The proofs are hugely inefficient
- Can we construct SNARKs without using PCPs?
- Yes, using QSPs and QAP - a better characterisation of NP and cryptographic assumptions

Main idea

1. Transform the verification of the computation to checking a relation between secret polynomials:

$$\text{computation validity} \leftrightarrow p(x)q(x) = s(x)r(x)$$

2. The verifier chooses a random evaluation point that must be kept secret:

$$p(x_0)q(x_0) = s(x_0)r(x_0)$$

3. Homomorphic Encryption to compute the evaluation of the polynomials at x_0 by using $\mathtt{Enc}(x_0)$:

$$\mathtt{Enc}(p(x_0))\mathtt{Enc}(q(x_0)) = \mathtt{Enc}(s(x_0))\mathtt{Enc}(r(x_0))$$

4. Randomise for ZK:

$$\mathtt{Enc}(k + p(x_0))\mathtt{Enc}(k + q(x_0)) = \mathtt{Enc}(k + s(x_0))\mathtt{Enc}(k_r(x_0))$$

ZK Proofs

- Shaffi Goldwasser, Silvio Micali and Charles Rackoff, 1985
- Interactive proof systems
  - Computation as a dialogue
  - Prover ($\mathcal{P}$): wants to prove that a string belongs to a language
  - Verifier ($\mathcal{V}$): wants to check the proof st:
    - A correct proof convinces $\mathcal{V}$ with overwhelming probability
    - A wrong proof convinces $\mathcal{V}$ with negligible probability
- Zero Knowledge Proofs
  - $\mathcal{V}$ is convinced without learning anything else

A breakthrough with many theoretical and practical applications

An easy example

- $\mathcal{V}$ is color blind
- O $\mathcal{P}$ holds two identical balls of different color
- Can the $\mathcal{V}$ be convinced of the different colors?
- Yes
  - $\mathcal{P}$ hands the balls to $\mathcal{V}$ (commit)
  - $\mathcal{V}$ hides the balls behind his back, one in each hand
  - He randomly decides to switch hands or not
  - $\mathcal{V}$ presents the balls to $\mathcal{P}$ (challenge)
  - $\mathcal{P}$ responds if the balls have switched hands (response)
  - $\mathcal{V}$ accepts or not
  - Malicious $\mathcal{P}$: Cheating Probability 50%
  - Repeat to reduce

Definitions: Notation

- Language $\mathcal{L} \in \mathtt{NP}$
- Polynomial Turing Machine $\mathcal{M}$
- $x \in \mathcal{L} \Leftrightarrow \exists w \in \{0,1\}^{p(|x|)} : M(x, w) = 1$
- 2 PPT TM $\mathcal{P}$, $\mathcal{V}$
- $< \mathcal{P}(x, w), \mathcal{V}(x) >$ is the interaction between $\mathcal{P}$, $\mathcal{V}$ with common public input x and private $\mathcal{P}$ input w.
- $out_{\mathcal{V}} < \mathcal{P}(x, w), \mathcal{V}(x) >$ is the output of $\mathcal{V}$ at the end of the protocol

Properties: Completeness and Soundness

Completeness

An honest $\mathcal{P}$, convinces an honest $\mathcal{V}$ with certainty: If $x \in \mathcal{L}$ and $M(x, w) = 1$ then: $Pr[out_{\mathcal{V}} < \mathcal{P}(x, w), \mathcal{V}(x) > (x) = 1] = 1$

Properties:Soundness

A malicious $\mathcal{P}$ ($\mathcal{P}^*$), only convinces an honest $\mathcal{V}$, with negligible probability. If $x \notin \mathcal{L}$ $\quad \forall(\mathcal{P}^*, w)$:
$Pr[out_{\mathcal{V}} < \mathcal{P}^*(x, w), \mathcal{V}(x) > (x) = 1] = negl(\lambda)$

Note:
Proof of Knowledge: $\mathcal{P}^*$ is not PPT.
Argument of Knowledge: O $\mathcal{P}^*$ is PPT.

Properties:(Perfect) Zero Knowledge

$\mathcal{V}$ does not gain any more knowledge than the validity of the $\mathcal{P}$'s claim.

For each $\mathcal{V}^*$ there is a PPT $\mathcal{S}$:

If $x \in \mathcal{L}$ and $M(x, w) = 1$ the random variables:

$out_{\mathcal{V}^*} < \mathcal{P}(x, w), \mathcal{V}^*(x) > (x)$ and

$out_{\mathcal{V}^*} < \mathcal{S}(x), \mathcal{V}^*(x) > (x)$

follow the same distribution: We allow a malicious verifier that does not follow the protocol and cheats in order to learn w

Intuition

What ever the $\mathcal{V}$ can learn after interacting with the $\mathcal{P}$, can be learnt by interacting with $\mathcal{S}$ (disregarding $\mathcal{P}$)

Constructing the simulator

A theoretical construction with practical applications

Reminder: $\mathcal{S}$ does not have access to the witness

- $\mathcal{S}$ take $\mathcal{P}$'s place during the interaction with $\mathcal{V}$
- We cannot distinguish between $<\mathcal{S}, \mathcal{V}>$ and $<\mathcal{P}, \mathcal{V}>$
- We allow rewinds:
- when $\mathcal{V}$ sets a challenge that cannot be answered by $\mathcal{S}$ then we stop and rewind it
- ZK if despite the rewind $\mathcal{V}$ accepts at some point
- Why? Because he cannot distinguish between $\mathcal{P}$ (with the witness) and $\mathcal{S}$ (without the witness)
- As long as $\mathcal{S}$ is PPT
- As a result $\mathcal{V}$ extracts the same information from $\mathcal{P}$ and $\mathcal{S}$ (nothing to extract)

Cryptographic Applications

- Authentication without passwords
  - Proof that the user know the password
  - Transmission and processing is not needed
- Proof that a ciphertext contains a particular message
- Digital signatures
- Anti-Malleability
- In general: Proof that a player follows a protocol without releasing any private input

Σ - protocols

A 3 round protocol with an honest verifier and special soundness

1. **Commit** $\mathcal{P}$ commits to a value
2. **Challenge** $\mathcal{V}$ selects a random challenge uniformly from a challenge space (honest)
3. **Response** $\mathcal{P}$ responds using the commitment, the witness and the random challenge.

Special Soundness

Two execution of the protocol with the same commitment reveal the witness

Knowledge of DLOG:Schnorr's protocol I

Protocol input

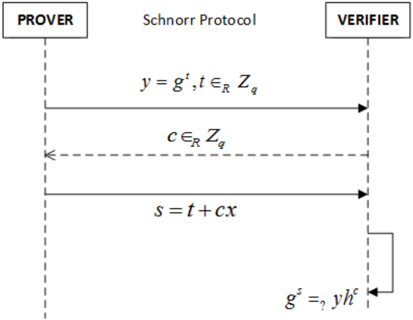
- **Public:** g is a generator of an order q subgroup of $\mathbb{Z}_p^*$ with hard DLP and a random $h \in \mathbb{Z}_p^*$
- **Private:** $\mathcal{P}$ knows a witness $x \in \mathbb{Z}_q^*$ st: $h = g^x \pmod{p}$

Goal

Proof of knowledge of x without releasing any more information

Knowledge of DLOG:Schnorr's protocol II

- **Commit** $(\mathcal{P} \to \mathcal{V})$**:**
  - Randomly Select $t \in_R \mathbb{Z}_q^*$
  - Compute $y = g^t \bmod p$.
  - Send y to $\mathcal{V}$.
- **Challenge** $(\mathcal{V} \to \mathcal{P})$**:** Select and challenge with $c \in_R \mathbb{Z}_q^*$
- **Response** $(\mathcal{P} \to \mathcal{V})$**:** $\mathcal{P}$ computes $s = t + cx \bmod q$ and sends it to $\mathcal{V}$
- $\mathcal{V}$ accepts iff $g^s = yh^c \pmod{p}$



Properties I

- **Completeness**
$$g^s = g^{t+cx} = g^t g^{cx} = yh^c \pmod{p}$$
- **Soundness** Probability that $\mathcal{P}^*$ cheats an honest verifier: $\frac{1}{q}$ - negligible - repeat to decrease
- **Special soundness** Let (y, c, s) nad (y, c', s') be two successful protocol transcripts

$$g^s = yh^c \quad g^{s'} = yh^{c'} \Rightarrow g^s h^{-c} = g^{s'} h^{-c'} \Rightarrow$$
$$g^{s-xc} = g^{s'-xc'} \Rightarrow s - xc = s' - xc' \Rightarrow x = \frac{c' - c}{s - s}$$

Since $\mathcal{P}$ can answer these 2 questions he knows DLOG of h

Properties II

Zero knowledge: no

- A cheating verifier does not choose randomly
- but bases each challenge to the commitment received before $\mathcal{S}$
- In the simulated execution it will switch challenge
- $\mathcal{S}$ will not be able to respond

How to add ZK:

- $\mathcal{V}$ commits to randomness before the first message by $\mathcal{P}$ or
- Challenge space $\{0, 1\}$
  - In this case $\mathcal{V}$ has only two options.
  - As a result the $\mathcal{S}$ can prepare for both.

Properties III

It provides Honest Verifier Zero Knowledge. Let $\mathcal{S}$ without knowledge of the witness x and an honest $\mathcal{V}$

- $\mathcal{S}$ follows the protocol and commits to $y = g^t, t \in_R \mathbb{Z}_q^*$
- $\mathcal{V}$ selects $c \in_R \mathbb{Z}_q^*$
- If $\mathcal{S}$ can answer (which occurs with negligible probability) the protocol resumes normally
- Else the $\mathcal{V}$ is rewound (with the same random tape)
- $\mathcal{V}$ selects the same $c \in_R \mathbb{Z}_q^*$ (because the random tape has not changed)
- $\mathcal{S}$ sends $s = t$. $\mathcal{V}$ will accept since $yh^c = g^t h^{-c} h^c = g^t = g^s$

The conversations $(t \in_R \mathbb{Z}_q; g^t h^{-c}, c \in_R \mathbb{Z}_q, t)$
$(t, c \in_R \mathbb{Z}_q; g^t, c, t + xc)$ follow the same distribution

Removing interactivity

Question

Can we do away with $\mathcal{V}$?

$\mathcal{P}$ generates the proof by himself
The proof is verifiable by anyone

Fiat Shamir Transform

Replace the challenge with the output of a pseudorandom function on the commitment
In practice we use a hash function $\mathcal{H}$

Non-interactive Schnorr with the Fiat Shamir

Input

- **Public:** g is a generator of an order q subgroup of ($\mathbb{Z}_p^*$ with hard DLP and $h \in \mathbb{Z}_p^*$
- **Private:** $\mathcal{P}$ has a witness $x \in \mathbb{Z}_q^*$ st: $h = g^x \mod p$

The Prover:

- Randomly select $t \in_R \mathbb{Z}_q$,
- Compute $y = g^t \mod p$
- **Compute** $c = \mathcal{H}(y)$ **where** $\mathcal{H}$ **is a hash function in** $\mathbb{Z}_q$
- Compute $s = t + cx \mod q$
- **Release** (h, c, s)
- **Anyone can verify that** $c = \mathcal{H}(g^s h^{-c})$

The common reference string

- Both parties have access to a string of (random) data
- This is created in a trusted way (e.g. through a secure multiparty computation protocol)
- The prover simulates the verifier challenge by selecting data from the CRS

Homomorphic Encryption Schemes

Applying a function on the ciphertexts yields the encryption of a function on the plaintext

$$\mathtt{Enc}(m_1) \otimes \mathtt{Enc}(m_2) = \mathtt{Enc}(m_1 \oplus m_2)$$

Multiplicative Homomorphism in El Gamal:

$$\begin{aligned}\mathtt{Enc}(m_1) \cdot \mathtt{Enc}(m_2) &= (g^{r_1}, m_1 h^{r_1}) \cdot (g^{r_2}, m_2 h^{r_2}) \\ &= (g^{r_1+r_2}, (m_1 \cdot m_2) h^{r_1+r_2})\end{aligned}$$

Additive Homomorphism in El Gamal:

$$\begin{aligned}\mathtt{Enc}(m_1) \cdot \mathtt{Enc}(m_2) &= (g^{r_1}, g^{m_1} h^{r_1}) \cdot (g^{r_2}, g^{m_2} h^{r_2}) \\ &= (g^{r_1+r_2}, g^{m_1+m_2} h^{r_1+r_2})\end{aligned}$$

Application - polynomials

Task

Let $\texttt{Enc}(x) = g^x$ where g is a suitable group generator and $p(x) = \sum_{i=0}^{d} a_i x^i$ a polynomial
Two parties with knowledge of x_0 and $p(x)$ respectively can compute $\texttt{Enc}(p(x_0))$

- The $\mathcal{V}$ (the party that knows x_0) releases

$$\texttt{Enc}(x_0^0), \texttt{Enc}(x_0^1), \cdots, \texttt{Enc}(x_0^d)$$

into the common reference string
- The $\mathcal{P}$ (the party that knows the coefficients) computes:

$$\prod_{i=0}^{d} \texttt{Enc}(x_0^i)^{a_i} = \texttt{Enc}(\sum_{i=0}^{d} a_i x_0^i) = \texttt{Enc}(p(x_0))$$

Pairings I

In general

Functions that map elements from source groups $\mathcal{G}_1, \mathcal{G}_2$ or $\mathcal{G}^2$ to a destination group $\mathcal{G}_T$.

What is interesting: They transform difficult problems in $\mathcal{G}$ to easy problems in $\mathcal{G}_T$.

Definition

A pairing is an efficiently calculable function $e : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}_T$ st:

- Bilinear: $e(g^a, g^b) = e(g, g)^{ab}$ where $g \in \mathcal{G}$ $a, b \in \mathbb{Z}$
- Non-Degenerate:If $\mathcal{G} =< g >$ then $\mathcal{G}_T =< e(g, g) >$

Pairings II

In practice: $G = \mathcal{E}(\mathbb{F}_p)$ and $G_T = \mathbb{F}_{p^a}$

How to easily solve DDH

Input: (g, g^a, g^b, g^c)
Check if $g^c = g^{ab}$
Easily compute $e(g^a, g^b) = e(g, g)^{ab}$
Compare with $e(g, g^c) = e(g, g)^c$
but the CDH remains hard

Observation

The pairing allows us to do a multiplication between 'encrypted' values

Application - check the correct evaluation of polynomials I

- The $\mathcal{V}$ that knows x_0:
  - computes and publishes into the CRS:

$$\texttt{Enc}(x_0^0), \texttt{Enc}(x_0^1), \cdots, \texttt{Enc}(x_0^d)$$

  - selects a scaling factor b
  - computes and publishes into the CRS:

$$\texttt{Enc}(bx_0^0), \texttt{Enc}(bx_0^1), \cdots, \texttt{Enc}(bx_0^d)$$

- The $\mathcal{P}$ that knows $p(x)$:
  - computes and publishes $\texttt{Enc}(p(x_0)), \texttt{Enc}(bp(x_0))$
- The secrets b, x_0 should be destroyed

Application - check the correct evaluation of polynomials II

Check:

- Use a pairing function e to compute:
  - $e(\mathtt{Enc}(p(x_0)), \mathtt{Enc}(b)) = e(g, g)^{bp(x_0)}$
  - $e(\mathtt{Enc}(bp(x_0)), \mathtt{Enc}(1)) = e(g, g)^{bp(x_0)}$

Observation

- The homomorphic combination of encrypted polynomials allows us to do additions
- plus the multiplication from the pairing

A 'new' security assumption I

Let $\mathbb{G}$ a group of order q generated by g and $x \in_R \mathbb{Z}_q$. Let $h = g^x$

Knowledge of exponents (Damgard 1991)

For any adversary $\mathcal{A}(q, g, h)$ that outputs a value (c, y) such that $y = c^x$, there exists an extractor $\mathcal{B}$ who on input $\mathcal{B}(q, g, h)$ outputs s: $c = g^s$

A 'new' security assumption II

Intuition

- The exponent in question is s
- Since $y = c^x$ and we do not know x the only way to have come up with (c, y) is through s
- That is: $c = g^s$ and $y = h^s$
- Between ZKP of DLOG equality and double DLOG knowledge
- Non standard, but cannot be derived from standard assumptions such as the DDH.

KoE Relation to zk-SNARKs

There is no need to know x in order to validate knowledge of exponent:

$$e(h, c) = e(g, y) = e(g, g)^{sx}$$

The correspondence

$C = \mathtt{Enc}(p(x_0)) = g^{p(x_0)}$ and
$Y = \mathtt{Enc}(bp(x_0)) = g^{bp(x_0)}$

If it does not hold then a cheating prover might come up with Y without knowing $p(x_0)$

Remarks

- Is it sound?
- Answer: No - the prover can cheat by replacing p with any polynomial
- Is it zero knowledge?
- Answer: No - it allows the verifier to learn $\mathtt{Enc}(p(x_0))$

Evaluate polynomials and check in ZK

ZK: $\mathcal{V}$ must not even learn $\mathtt{Enc}(p(x_0))$

- $\mathcal{V}$ selects b, x_0 and computes:

$$\mathtt{Enc}(x_0^0), \mathtt{Enc}(x_0^1), \cdots \mathtt{Enc}(x_0^d)$$
$$\mathtt{Enc}(bx_0^0), \mathtt{Enc}(bx_0^1), \cdots \mathtt{Enc}(bx_0^d)$$

- $\mathcal{P}$ selects a and computes:

$$\mathtt{Enc}(a)\mathtt{Enc}(p(x_0)) = \mathtt{Enc}(a + p(x_0))$$
$$\mathtt{Enc}(b)^a\mathtt{Enc}(bp(x_0)) = \mathtt{Enc}(ba)\mathtt{Enc}(bp(x_0)) = \mathtt{Enc}(b(a + p(x_0)))$$

- Check the pairing step as before:

$$e(\mathtt{Enc}(a + p(x_0)), \mathtt{Enc}(b)) = e(g, g)^{b(a+p(x_0))}$$
$$e(\mathtt{Enc}(b(a + p(x_0))), \mathtt{Enc}(1)) = e(g, g)^{b(a+p(x_0))}$$

R1CS

Definition

A system of rank-1 quadratic equations over $\mathbb{F}$ is a set of constraints $\{(\boldsymbol{v}_j, \boldsymbol{w}_j, \boldsymbol{y}_j)\}_{i=1}^{N_c}$ and $n \in \mathbb{N}$ where:

- $\boldsymbol{v}_j, \boldsymbol{w}_j, \boldsymbol{y}_j \in \mathbb{F}^{1+N_v}$
- $n \leq N_v$

Satisfiability

A R1 system C is satisfiable on input $\boldsymbol{c} \in \mathbb{F}^n$ if there is a witness $s \in \mathbb{F}^{N_v}$:

- $\boldsymbol{c} = (c_1, \cdots, c_n)$
- $\forall j \in N_c : \boldsymbol{v}_j \cdot (1, \boldsymbol{c}) \times \boldsymbol{w}_j \cdot (1, \boldsymbol{c}) = \boldsymbol{y}_j \cdot (1, \boldsymbol{c})$

Facts

BC to R1CS

Boolean circuit $C : \{0,1\}^n \times \{0,1\}^h \times \{0,1\}$ with α wires and β (bilinear) gates $\rightarrow$ R1CS with with $N_v = \alpha$ and $N_c = \beta + h + 1$

AC to R1CS

Arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \times \mathbb{F}^l$ with α wires and β (bilinear) gates $\rightarrow$ R1CS with with $N_v = \alpha$ and $N_c = \beta + l$

Quadratic Span Programs - QSP I

Definition

A QSP over a field $\mathbb{F}$ for inputs of length n consists of

- 2 sets of source polynomials:
  $\mathcal{V} = \{v_0, \cdots, v_m\}, \mathcal{W} = \{w_0, \cdots, w_m\}$
- the target polynomial: t
- an injective function $f\colon [n] \times \{0,1\} \to [m]$

Quadratic Span Programs - QSP II

QSP Verification

An input $u \in \{0,1\}^n$ is accepted by a QSP iff $\exists$ tuples $a = (a_1, \cdots, a_m),\ b = (b_1, \cdots, b_m) \in \mathbb{F}^m$:

- $a_k \wedge b_k = 1$, if $\exists i : k = f(i, u_i)$
- $a_k \wedge b_k = 0$, if $\exists i : k = f(i, 1 - u_i)$
- t divides the linear combination $v_a \cdot w_b$ where
  $v_a = v_0 + \sum_{i=1}^{m} a_i v_i$,
  $w_b = w_0 + \sum_{i=1}^{m} b_i w_i$

Quadratic Span Programs - QSP III

Remarks:

- Check if a target polynomial divides a linear combination of some given polynomials
- f restricts which polynomials can be used in the linear combination
- The NP witness is the pair a, b
- QSP Verification is NP-Complete
- In practice:
  - Find $h : th = v_a \cdot w_b \Leftrightarrow th - v_a \cdot w_b = \mathbf{0}$
  - Check that it is a zero polynomial
  - Evaluate at a single point $t(x_0)h(x_0) - v_a(x_0) \cdot w_b(x_0) = 0$ (The number of roots is tiny compared to the number of field elements)

Quadratic Arithmetic Programs I

Definition

A QAP $\mathcal{Q}$ over a field $\mathbb{F}$ is:

- 3 sets of source polynomials $\mathcal{V} = \{v_0, \cdots, v_m\}$, $\mathcal{W} = \{w_0, \cdots, w_m\}$, $\mathcal{Y} = \{y_0, \cdots, y_m\}$
- the target polynomial t
- a function $f\colon \{0,1\}^n \to \{0,1\}^{n'}$

Quadratic Arithmetic Programs II

$\mathcal{Q}$ computes f if: $(c_1, \cdots, c_{n+n'}) \in \mathbb{F}^{n+n'}$ is a valid assignment of f's inputs and outputs and there exist coefficients $(c^{N+1}, \cdots, c^m)$ such that $t(x)$ divides $p(x)$ where:

$$p(x) = (v_0(x) + \sum_{k=1}^{m} c_k v_k(x)) \cdot (w_0(x) + \sum_{k=1}^{m} c_k w_k(x)) - (y_0(x) + \sum_{k=1}^{m} c_k y_k(x))$$

For simplicity: $v(x) = v_0(x) + \sum_{k=1}^{m} c_k v_k(x)$ etc.

From Code to QAP

Process

Code → Algebraic Circuit → R1CS → QAP → ZKSnark

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

Task

Prove that you executed f with input $= 3$

Convert to circuit - Flattening

Convert code into a format that contains only commands of the form:

- `x=y`
- `x=y op z`

As a result the function *f* becomes:

```
def f(x):
    sym_1 = x * x
    y = sym_1 * x
    sym_2 = y + x
    out = sym_2 + 5
```

Convert to R1CS

Rules

- Each command can be considered as a logic gate and represented as a relation between vectors
- The vectors have as many elements as the total number of variables in the command plus one (for constants)
- Mapping vector $[one, x, out, sym_1, y, sym_2]$
- Vector $\boldsymbol{y}$ is the left hand side
- Vector $\boldsymbol{v}$, $\boldsymbol{w}$ are the right hand sides

Application to example commands

Command

sym$_1$ $= x * x$

Command

y = sym1 * x

$$\begin{array}{rlllll} & [one, & x, out, & sym_1, y, & sym_2] \\ \boldsymbol{v} = & [0, & 1, 0, & 0, 0, & 0] \\ \boldsymbol{w} = & [0, & 1, 0, & 0, 0, & 0] \\ \boldsymbol{y} = & [0, & 0, 0, & 1, 0, & 0] \end{array}$$

Indeed $c = [1, 3, 0, 9, 0, 0]$
satisfies: $\boldsymbol{cv} \cdot \boldsymbol{cw} - \boldsymbol{cy} = 0$

$$\begin{array}{rlllll} & [one, & x, out, & sym_1, y, & sym_2] \\ \boldsymbol{v} = & [0, & 0, 0, & 1, 0, & 0] \\ \boldsymbol{w} = & [0, & 1, 0, & 0, 0, & 0] \\ \boldsymbol{y} = & [0, & 0, 0, & 0, 1, & 0] \end{array}$$

$c = [1, 3, 0, 9, 27, 0]$

Application to commands

Command

sym2 = y+x

Command

out = sym2+5

$$\begin{array}{rcccc} & [one, & x,out, & sym_1,y, & sym_2] \\ \boldsymbol{v} = [& 0,1, & 0,0, & 1,0] \\ \boldsymbol{w} = [& 1,0, & 0,0, & 0,0] \\ \boldsymbol{y} = [& 0,0, & 0,0, & 0,1] \end{array}$$

$$\begin{array}{rcccc} & [one, & x,out, & sym_1,y, & sym_2] \\ \boldsymbol{v} = [5, & 0,0, & 0,0, & 1] \\ \boldsymbol{w} = [1, & 0,0, & 0,0, & 0] \\ \boldsymbol{y} = [1, & 0,0, & 0,0, & 0] \end{array}$$

Remark: addition is implied in the dot product

$c = [1, 3, 0, 9, 27, 30]$

The final R1CS

$$\boldsymbol{V} = \{[0,1,0,0,0,0],[0,0,0,1,0,0],[0,1,0,0,1,0],[5,0,0,0,0,1]\}$$
$$\boldsymbol{W} = \{[0,1,0,0,0,0],[0,1,0,0,0,0],[1,0,0,0,0,0],[1,0,0,0,0,0]\}$$
$$\boldsymbol{Y} = \{[0,0,0,1,0,0],[0,0,0,0,1,0],[0,0,0,0,0,1],[1,0,0,0,0,0]\}$$

The solution is the vector $\boldsymbol{c} = [1,3,35,9,27,30]$

From Vectors To Polynomials

- Use Lagrange interpolation to transform the sets of m vectors with n elements into n polynomials of degree $m-1$
- Construct polynomial v_j with values $v_j(i) = V[i][j]$ (value element of vector i in position j)
- For instance: $v_1(1) = 0, v_1(2) = 0, v_1(3) = 0, v_1(4) = 5$
- $v_1(x) = \frac{5}{6}x^3 - 5x^2 + \frac{55}{6}x - 5$
- $v_2(1) = 1, v_2(2) = 0, v_2(3) = 1, v_2(4) = 0$
- $v_2(x) = -\frac{2}{3}x^3 + 5x^2 + \frac{34}{3}x + 8$
- Repeat for $\boldsymbol{w}, \boldsymbol{y}$
- Finally add the polynomials together to obtain v, w, y

From Vectors To Polynomials - Why?

- Why? Because we can check all the constraints simultaneously!
- $cv(x) \cdot cw(X) = cy(x)$
- Define $t(x) = cv(x) \cdot cw(X) - cy(x)$
- This polynomial must be zero to all the points that correspond to the logic gates
- A multiple of the base polynomial $(x-1)(x-2)$...

Setup Phase I

- Non interactiveness - Public verifiability
- Fix the homomorphic encryption scheme, verifier, polynomials
- $\mathcal{V}$ selects random field elements $x_0, b \in \mathbb{F}$
- computes and publishes in the CRS:
  - $\{\mathtt{Enc}(x_0^k)\}_{k=0}^{d}$ (in reality: d $= 2 \cdot 10^6$)
  - $\{\mathtt{Enc}(bx_0^k)\}_{k=0}^{d}$
  - $\{\mathtt{Enc}(v_k(x_0)), \mathtt{Enc}(bv_k(x_0))\}_{k=1}^{m}$
  - $\{\mathtt{Enc}(w_k(x_0)), \mathtt{Enc}(bw_k(x_0))\}_{k=1}^{m}$
  - $\{\mathtt{Enc}(y_k(x_0)), \mathtt{Enc}(by_k(x_0))\}_{k=1}^{m}$
  - $\mathtt{Enc}(t(x_0)), \mathtt{Enc}(bt(x_0))$

Setup Phase II

- selects random field values $\gamma, \beta_v, \beta_w, \beta_y$ in order to ensure soundness (i.e. that the correct polynomials were evaluated)
- computes and publishes in the CRS:
  - $\mathtt{Enc}(\gamma), \mathtt{Enc}(\beta_v\gamma), \mathtt{Enc}(\beta_w\gamma), \mathtt{Enc}(\beta_y\gamma)$
  - $\{\mathtt{Enc}(\beta_v v_k(x_0))\}_{k=1}^m$
  - $\{\mathtt{Enc}(\beta_w w_k(x_0))\}_{k=1}^m$
  - $\{\mathtt{Enc}(\beta_y y_k(x_0))\}_{k=1}^m$
  - $\mathtt{Enc}(\beta_v t(x_0)), \mathtt{Enc}(\beta_w t(x_0)), \mathtt{Enc}(\beta_y t(x_0))$

All computations in the proof must use only these elements

Performance: $O(|C|)$

The prover

- Evaluates the circuit for the function and obtains the output
- As a result the $\mathcal{P}$ knows the values of c_i
- Solves for h
- Define:
  - I_{mid}: the indices that are not in IO of $f\left(\{N+1\cdots m\}\right)$
  - $v_{mid}(x) = \sum_{k\in I_{mid}} c_k v_k(x)$
- Generate the proof (9 encrypted values):
  - $V_{mid} = \mathtt{Enc}(v_{mid}(x_0)),\ W = \mathtt{Enc}(w(x_0)),\ Y = \mathtt{Enc}(y(x_0)),$ $H = \mathtt{Enc}(h(x_0))$
  - $V'_{mid} = \mathtt{Enc}(bv_{mid}(x_0)),\ W' = \mathtt{Enc}(bw(x_0)),\ Y' = \mathtt{Enc}(by(x_0)),$ $H' = \mathtt{Enc}(bh(x_0))$
  - $K = \mathtt{Enc}(\beta_v v_{mid}(x_0) + \beta_w w(x_0) + \beta_y y(x_0))$
- All these values can be computed by leveraging the homomorphic properties of the underlying cryptosystem from what is on the CRS
- Performance: $O(|C|) + O(|C| log^2(|C|)$

The verifier

- Retrieves the values of c_i from the input u and the output
- Computes $\mathtt{Enc}(v_{io}(x_0)) = \mathtt{Enc}(\sum_{k \notin I_{mid}} c_k v_k(x_0))$
- Verifies the following equations using the pairing function:
  - $e(V'_{mid}, \mathtt{Enc}(1)) = e(V_{mid}, \mathtt{Enc}(b))$
  - $e(W', \mathtt{Enc}(1)) = e(W, \mathtt{Enc}(b))$,
  - $e(H', \mathtt{Enc}(1)) = e(H, \mathtt{Enc}(b))$
  - $e(Y', \mathtt{Enc}(1)) = e(Y, \mathtt{Enc}(b))$
  - For soundness check:
    $e(\mathtt{Enc}(\gamma), K) =$
    $e(\mathtt{Enc}(\beta_v\gamma), V_{mid}) \cdot e(\mathtt{Enc}(\beta_w\gamma), W) \cdot e(\mathtt{Enc}(\beta_y\gamma), Y)$
  - Check the QAP relation:
    $\frac{e(\mathtt{Enc}(v_0(x_0)) \cdot \mathtt{Enc}(v_{io}(x_0)) \cdot V_{mid}, \mathtt{Enc}(w_0(x_0)W))}{e(y_0(x_0)Y, \mathtt{Enc}(1))} = e(H, \mathtt{Enc}(t(x_0)))$

Completeness

$$
\begin{aligned}
e(\texttt{Enc}(\gamma), K) &= \\
e(\texttt{Enc}(\gamma), \texttt{Enc}(\beta_v v_{mid}(x_0) + \beta_w w(x_0) + \beta_y y(x_0))) &= \\
e(g^{\gamma}, g^{\beta_v v_{mid}(x_0) + \beta_w w(x_0) + \beta_y y(x_0)}) &= \\
e(g, g)^{\gamma \cdot (\beta_v v_{mid}(x_0) + \beta_w w(x_0) + \beta_y y(x_0))}
\end{aligned}
$$

$$
\begin{aligned}
e(\texttt{Enc}(\beta_v\gamma), V_{mid}) \cdot e(\texttt{Enc}(\beta_w\gamma), W) \cdot e(\texttt{Enc}(\beta_y\gamma), Y) &= \\
e(\texttt{Enc}(\beta_v\gamma, \texttt{Enc}(v_{mid}(x_0)))e(\texttt{Enc}(\beta_w\gamma), \texttt{Enc}(w(x_0)))e(\texttt{Enc}(\beta_y\gamma), \texttt{Enc}(y(x_0))) &= \\
e(g, g)^{\beta_v\gamma v_{mid}(x_0)} \cdot e(g, g)^{\beta_w\gamma w(x_0)} \cdot e(g, g)^{\beta_y\gamma y(x_0)} &= \\
e(g, g)^{\beta_v\gamma v_{mid}(x_0) + \beta_w\gamma w(x_0) + \beta_y\gamma y(x_0)}
\end{aligned}
$$

Completeness for the QAP Relation I

The parts of the left hand pairings:

$$\mathtt{Enc}(v_0(x_0))\mathtt{Enc}(v_{io}(x_0))V_{mid} = \mathtt{Enc}(v_0(x_0))\mathtt{Enc}(v_{io}(x_0))\mathtt{Enc}(v_{mid}(x_0)) =$$

$$\mathtt{Enc}(v_0(x_0) + v_{io}(x_0) + v_{mid}(x_0)) = \mathtt{Enc}(v_0(x_0) + \sum_{i=1}^{m} c_i v_i(x_0)) = \mathtt{Enc}(v(x_0))$$

$$\mathtt{Enc}(w_0(x_0))W = \mathtt{Enc}(w_0(x_0))\mathtt{Enc}(w(x_0)) =$$

$$\mathtt{Enc}(w_0(x_0) + \sum_{i=1}^{m}(c_i w_i(x_0))) = \mathtt{Enc}(w(x_0))$$

Completeness for the QAP Relation II

$$\mathtt{Enc}(y_0(x_0))Y = \mathtt{Enc}(y_0(x_0))\mathtt{Enc}(y(x_0)) =$$

$$\mathtt{Enc}(y_0(x_0) + \sum_{i=1}^{m}(c_i y_i(x_0))) = \mathtt{Enc}(y(x_0))$$

Left hand side: $e(\mathtt{Enc}(v(x_0)), \mathtt{Enc}(w(x_0))) = e(g,g)^{v(x_0)\cdot w(x_0) - y(x_0)}$
Right hand side:
$e(H, \mathtt{Enc}(t(x_0))) = e(g^h(x_0), g^t(x_0)) = e(g,g)^{h(x_0)t(x_0)}$

Intuition between soundness

The relation
$e(\mathtt{Enc}(\gamma), K) = e(\mathtt{Enc}(\beta_v\gamma), V_{mid}) \cdot e(\mathtt{Enc}(\beta_w\gamma), W) \cdot e(\mathtt{Enc}(\beta_y\gamma), Y)$
protects from a prover that tries to cheat by using another polynomial.

- The values $\beta_v, \beta_w, \beta_y$ do not appear in the CRS in isolation
- The expression $\beta_v v_{mid}(x_0) + \beta_w w(x_0)) + \beta_y y(x_0)$ can only be encrypted from the respected values in the CRS in encrypted form mixed with γ

Shifting for Zero Knowledge

The $\mathcal{P}$ chooses $\delta_{mid}, \delta_w, \delta_y$.
Define

- $V_{\delta mid} = \texttt{Enc}(v_{mid}(x_0) + \delta_{mid} t(x_0))$
- $w_\delta(x_0) = w(x_0) + \delta_w t(x_0)$
- $y_\delta(x_0) = y(x_0) + \delta_y t(x_0)$
- As a result V_{mid}, W, Y are randomised

The equation $v(x_0)w(x_0) - y(x_0) = h(x_0)t(x_0)$ must still hold
To achieve this we replace $H = \texttt{Enc}(h(x_0))$ in the CRS accordingly

vnTinyRAM

- zk-SNARKs for a general purpose CPU
- Circuit generator: Translate program execution into sequence of circuits
- Compose zk-SNARKs for these circuits
- Bound on the running time

Pinnochio: A cloud based lie detector I

- General purpose computation validator
- Client: represents functions as a public evaluation key
- Client: provides input or ZKPoK of some property of the input
- Server: evaluates the computation and provides proof (signature)
- Compiler toolchain to use with C-programs
- Transforms to QAP, QSP
- Use:
  - Protect against malicious servers
  - Extra server feature (at a higher price)
- Performance
  - Setup: Linear in the size of the computation

Pinnochio: A cloud based lie detector II

- Proof Size: constant (288 bytes)
  - Does not depend on function
  - Does not depend on input/output size
- Verification: Linear in the size of the input and output typically 10ms (5 - 7 orders of magnitude gain)
- Proof generation: up to 60 times fewer work

Bitcoin's problem I

Bitcoin is not anonymous

- All transactions are recorded in the blockchain
- Users use pseudonyms
- Deanonymization
  - The structure of the transaction graph
  - Real world information (value, dates, blockchain exit points)

Bitcoins are not fully fungible(?)

- In the protocol itself all coins have the same value

but...

Bitcoin's problem II

- Each coin has a history than can be traced
- This might have an effect on the ability to spend the coins or on their value (e.g. Wannacry ransomware)

A first solutions: mixes

- Users entrust their coins to a 'trusted' entity
- They receive coins with the same value but different origins
- Many problems (fees, delays, trust)

ZeroCoin

- A decentralised mix
- Two kinds of coins: base and anonymous
- Each anonymous transaction is accompanied by a ZK proof that the coin spent can be linked to a valid base coin
  - The base coin comes from a valid transaction
  - The base coin has not been spent
- Problems:
  - Performance bottleneck for ZK proofs
  - Functionality: Does not support all denominations etc.
  - Anonymity: Does not hide metadata

Transactions occur using the base coin and are periodically washed in the distributed mix

zCash=Zerocoin+SNARKs

- Performance
    - 288 byte proof
    - 895MB CRS
    - transaction $<$ 1KB (vs 45KB in Zerocoin)
    - 6ms verification (vs 450ms in Zerocoin)
    - 40sec to make a transaction

zCash CRS generation ceremony I

Goal

- Generate x_0 in CRS: $g^{x_0^1}, \cdots, g^{x_0^d}$
- No participant must learn the entire x_0
- All shares of x_0 must be later destroyed
- A single honest participant is required

zCash CRS generation ceremony II

The protocol

- Each participant generates a random s_i
- The first participant computes and publishes $g^{s_1}, \cdots, g^{s_1^d}$ e
- The second partipant computes $g^{s_1 s_2}, \cdots, g^{{s_1^d s_2}^d}$
- $\cdots$
- The last participant computes $g^{s_1 s_2 \cdots s_n}, \cdots, g^{{s_1^d s_2^d \cdots s_n}^d}$
- $x_0 = s_1 s_2 \cdots s_n$

zCash CRS generation ceremony III

Validation

A partipant might cheat by computing $g^{s_p \cdot s_i^{`}}$. validation can be done using pairings.

- $e(g^{s_i}, g^{s_i}) = e(g,g)^{s_i^2}$
- $e(g, g^{s_i^2}) = e(g,g)^{s_i^2}$

This check is repeated for all powers

Bibliography-References

1. Rosario Gennaro Craig Gentry Bryan Parno Mariana Raykova "Quadratic span programs and succinct NIZKs without PCPs." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg, 2013.
2. Parno, B., Howell, J., Gentry, C., Raykova, M. (2013, May). Pinocchio: Nearly practical verifiable computation. In Security and Privacy (SP), 2013 IEEE Symposium on (pp. 238-252). IEEE.
3. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP '14). IEEE Computer Society, Washington, DC, USA, 459-474.
4. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture." In USENIX Security, vol. 2014. 2014.
5. I. Damgård. Towards practical public-key cryptosystems provably-secure against chosen-ciphertext attacks. Advances in Cryptology – CRYPTO '91, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.
6. Succinct Computational Integrity and Privacy Research
7. Christian Reitwiessner zkSNARKs in a nutshell
8. Vitalik Buterin zkSNARKs: under the hood
9. Alfred Menezes An introduction to pairing based crypto
10. Zerocash parameter generation