zkSNARKs from scratch, a technical explanation

iden3.io
github.com/iden3
twitter.com/identhree

arnaucube.com
github.com/arnaucube
twitter.com/arnaucube

2019-08-20

Warning

- I'm not a mathematician, this talk is not for mathematicians
- In free time, have been studying zkSNARKS & implementing it in Go
- Talk about a technical explaination from an engineer point of view
- The idea is to try to transmit the learnings from long night study hours during last winter
- Also at the end will briefly overview how we use zkSNARKs in iden3
- This slides will be combined with
  - parts of the code from https://github.com/arnaucube/go-snark
  - whiteboard draws and writtings
- Don't use your own crypto. But it's fun to implement it (only for learning purposes)

Contents

- zkSNARK (Pinocchio)
    - Circuit compiler
    - R1CS
    - QAP
    - Lagrange Interpolation
    - Trusted Setup
    - Proofs generation
    - Proofs verification
- Groth16
- How we use zkSNARKs in iden3
- libraries
- Circuit languages
- utilities (Elliptic curve & Hash functions) inside the zkSNARK libraries
    - BabyJubJub
    - Mimc
    - Poseidon
- References

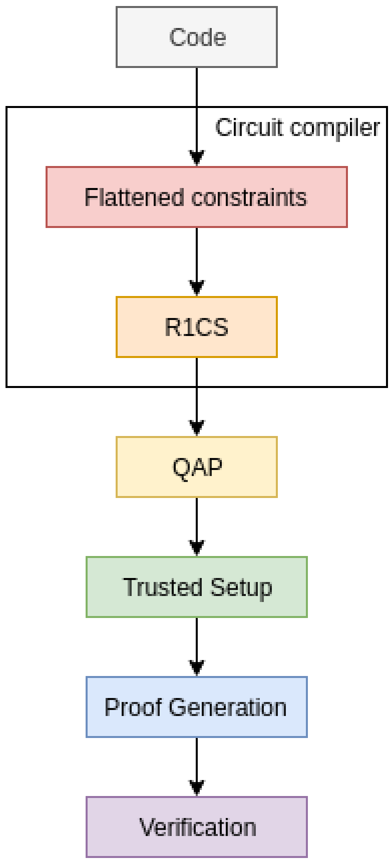
Introduction

- zero knowledge concept
- examples
- some concept explanations
  - https://en.wikipedia.org/wiki/Zero-knowledge_proof
  - https://hackernoon.com/wtf-is-zero-knowledge-proof-be5b49735f27

zkSNARK overview

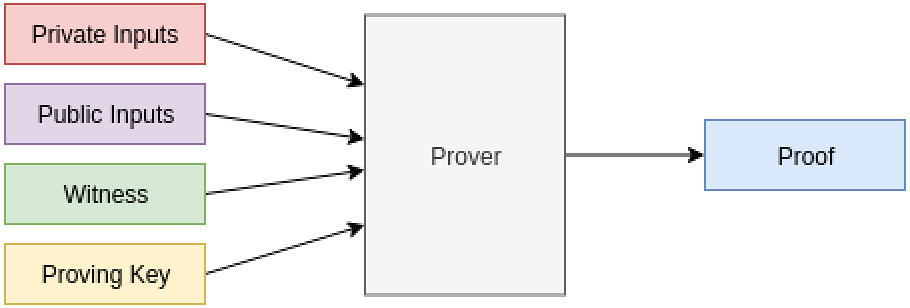
- protocol to prove the correctness of a computation
- useful for
  - scalability
  - privacy
  - interoperability
- examples:
  - Alice can prove to Brenna that knows x such as $f(x) = y$
  - Brenna can prove to Alice that knows a certain input which $Hash$ results in a certain known value
  - Carol can proof that is a member of an organization without revealing their identity
  - etc

zkSNARK flow

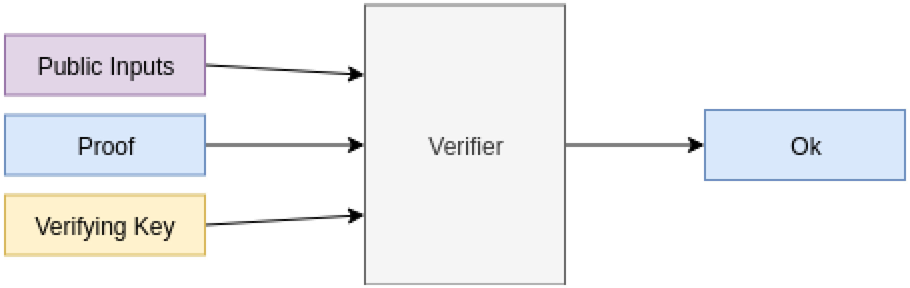


Generating and verifying proofs

Generating a proof:



Verifying a proof:

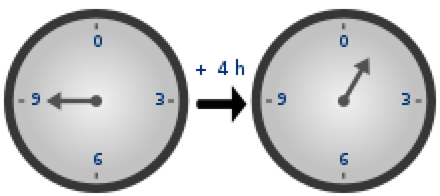


Foundations

- Modular aritmetic
- Groups
- Finite fields
- Elliptic Curve Cryptography

Basics of modular arithmetic

- Modulus, `mod`, `%`
- Remainder after division of two numbers




```
5 mod 12 = 5
14 mod 12 = 2
83 mod 10 = 3
```

```
5 + 3 mod 6 = 8 mod 6 = 2
```

Groups

- a **set** with an **operation**
  - **operation** must be *associative*
- neutral element ($identity$): adding the neutral element to any element gives the element
- inverse: e + $e_{inverse}$ = $identity$
- cyclic groups
  - finite group with a generator element
  - any element must be writable by a multiple of the generator element
- abelian group
  - group with *commutative* operation

Finite fields

- algebraic structure like Groups, but with **two operations**
- extended fields concept (https://en.wikipedia.org/wiki/Field_extension)

Elliptic curve

- point addition

$$(x_1, y_1) + (x_2, y_2) = (\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2})$$

- G1
- G2

(whiteboard explanation)

Pairings

- 3 typical types used for SNARKS:
  - BN (Barreto Naehrig) - used in Ethereum
  - BLS (Barreto Lynn Scott) - used in ZCash & Ethereum 2.0
  - MNT (Miyaji- Nakabayashi - Takano) - used in CodaProtocol
- $y^2 = x^3 + b$ with embedding degree 12
- function that maps (pairs) two points from sets `S1` and `S2` into another set `S3`
- is a bilinear function
- $e(G_1, G_2) -> G_T$
- the groups must be
  - cyclic
  - same prime order (r)

- F_q, where $q =$ `21888242871839275222246405745257275088696311157297823662689037894645226208583`
- F_r, where $r =$ `21888242871839275222246405745257275088548364400416034343698204186575808495617`

Bilinear Pairings

$$e(P_1 + P_2, Q_1) == e(P_1, Q_1) \cdot e(P_2, Q_1)$$

$$e(P_1, Q_1 + Q_2) == e(P_1, Q_1) \cdot e(P_1, Q_2)$$

$$e(aP, bQ) == e(P, Q)^{ab} == e(bP, aQ)$$

$$e(g_1, g_2)^6 == e(g_1, 6 \cdot g_2)$$

$$e(g_1, g_2)^6 == e(6 \cdot g_1, g_2)$$

$$e(g_1, g_2)^6 == e(3 \cdot g_1, 2g_2)$$

$$e(g_1, g_2)^6 == e(2 \cdot g_1, 3g_2)$$

BLS signatures

(small overview, is offtopic here, but is interesting)

- key generation
  - random private key x in $[0, r-1]$
  - public key g^x
- signature
  - $h = Hash(m)$ (over G2)
  - signature $\sigma = h^x$
- verification
  - check that: $e(g, \sigma) == e(g^x, Hash(m))$
    $e(g, h^x) == e(g^x, h)$

- aggregate signatures
    - $s = s0 + s1 + s2...$
- verify aggregated signatures

$$e(G, S) == e(P, H(m))$$

$$e(G, s0 + s1 + s2...) == e(p0, H(m)) \cdot e(p1, H(m)) \cdot e(p2, H(m))...$$

More info:

https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html

Circuit compiler

- not a software compiler -> a constraint prover
  - what this means
- constraint concept
  - `value0` == `value1` `<operation>` `value2`
- want to proof that a certain computation has been done correctly
- graphic of circuit with gates (whiteboard)
- about high level programing languages for zkSNARKS, by *Harry Roberts*: https://www.youtube.com/watch?v=nKrBJo3E3FY

Circuit code example:

$$f(x) = x^5 + 2 \cdot x + 6$$

```
func exp5(private a):
        b = a * a
        c = a * b
        d = a * c
        e = a * d
        return e

func main(private s0, public s1):
        s2 = exp5(s0)
        s3 = s0 * 2
        s4 = s3 + s2
        s5 = s4 + 6
        equals(s1, s5)
        out = 1 * 1
```

Inputs and Witness

For a certain circuit, with the inputs that we calculate the Witness for the circuit signals

- private inputs: `[8]`
  - in this case the private input is the 'secret' x value that computed into the equation gives the expected $f(x)$
- public inputs: `[32790]`
  - in this case the public input is the result of the equation
- signals: `[one s1 s0 b0 c0 d0 s2 s3 s4 s5 out]`
- witness: `[1 32790 8 64 512 4096 32768 16 32784 32790 1]`

R1CS

- Rank 1 Constraint System
- way to write down the constraints by 3 linear combinations
- 1 constraint per operation
- $(A, B, C) = A.s \cdot B.s - C.s = 0$
- from flat code constraints we can generate the R1CS

R1CS

$$(a_{11}s_1 + a_{12}s_2 + ... + a_{1n}s_n) \cdot (b_{11}s_1 + b_{12}s_2 + ... + b_{1n}s_n) - (c_{11}s_1 + c_{12}s_2 + ... + c_{1n}s_n) = 0$$

$$(a_{21}s_1 + a_{22}s_2 + ... + a_{2n}s_n) \cdot (b_{21}s_1 + b_{22}s_2 + ... + b_{2n}s_n) - (c_{21}s_1 + c_{22}s_2 + ... + c_{2n}s_n) = 0$$
$$(a_{31}s_1 + a_{32}s_2 + ... + a_{3n}s_n) \cdot (b_{31}s_1 + b_{32}s_2 + ... + b_{3n}s_n) - (c_{31}s_1 + c_{32}s_2 + ... + c_{3n}s_n) = 0$$
[...]
$$(a_{m1}s_1 + a_{m2}s_2 + ... + a_{mn}s_n) \cdot (b_{m1}s_1 + b_{m2}s_2 + ... + b_{mn}s_n) - (c_{m1}s_1 + c_{m2}s_2 + ... + c_{mn}s_n) = 0$$

*where s are the signals of the circuit, and we need to find a, b, c that satisfies the equations

R1CS constraint example:

- signals: `[one s1 s0 b0 c0 d0 s2 s3 s4 s5 out]`
- witness: `[1 32790 8 64 512 4096 32768 16 32784 32790 1]`
- First constraint flat code: `b0 == s0 * s0`
- R1CS first constraint:
  $A_1 = [00100000000]$
  $B_1 = [00100000000]$
  $C_1 = [00010000000]$

R1CS example:

A	B	C:
$[00100000000]$	$[00100000000]$	$[00010000000]$
$[00100000000]$	$[00010000000]$	$[00001000000]$
$[00100000000]$	$[00001000000]$	$[00000100000]$
$[00100000000]$	$[00000100000]$	$[00000010000]$
$[00100000000]$	$[20000000000]$	$[00000001000]$
$[00000011000]$	$[10000000000]$	$[00000000100]$
$[60000000100]$	$[10000000000]$	$[00000000010]$
$[00000000010]$	$[10000000000]$	$[01000000000]$
$[01000000000]$	$[10000000000]$	$[00000000010]$
$[10000000000]$	$[10000000000]$	$[00000000001]$

QAP

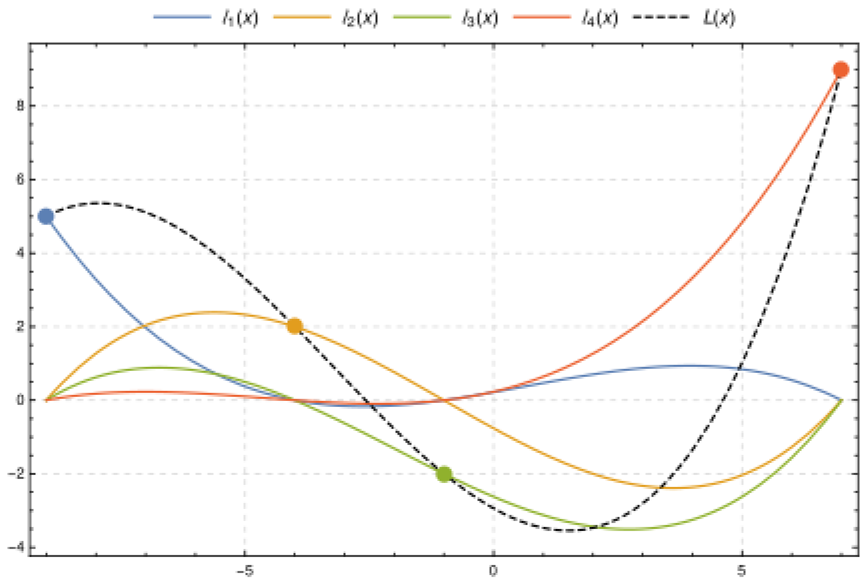
- Quadratic Arithmetic Programs
- 3 polynomials, linear combinations of R1CS
- very good article about QAP by Vitalik Buterin https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649

qap
[[719 1242070924870244586024300008560234260977149249706147980614620316936643099137 2256139319825496721816866623940705239781919703574628936815717673596454866025 18695724907992523792123340921389944136479669980813907288629700683524980694293 32946365572716409058485475314475794
27390873599854288502775406776000212841752 7218813433253010965948047820730394139446593097104802067403707503431454433717 4035644779495366369101681059281810094451104686326706332119356396899914691395 17770806708596625847188579056041137213508438114312641110757488759481347958998
 12301822215246771225479310530587180326924466410660410571987723340079226099186 20737312640805420475454949429050115515233242759310604583689095296352999297246] [10 7035506637376909892864916132404124135604831414419439610474422774256509873563 1594280547272261494858463402596020
9692472429705144298030896053089071387497534 16010120505364184148176860075375187423399512968677323533341950344010261511705 7957288294033236513087495421973738547816019974734579152031951313661413713518 17898198598326907343191071364611417650531735473256861416461552381731260071
92 3328836936758889773383307540424543919716730419229938556437435220041737542042 10497887913086009529955381803119672263451097985497154566925492871031820056752 5603433604244171598314171629527420472817366501515235776181866657089173990371 19387272263938393750299749215783062133
862295085120482006857489238967803789781] [127 15174111229009211830660504300382722055434124050605877777159032822995213270226 228002529914992450231733393179763282172378795837667024413522960276831338877 4061822847744865502276435634054301434256081511033994027514983107153921252
242 8455093817680970029426779997082888380559047012313485488668143110265828802639 4131912514348363181421746269735487924701664622347054631316177202350132367616 13414148843332055821966981298742739767808285821782743269662267496286910414851 3380227983184173785975222051268236596
015901512894936520670324522199372542147 21679240552750532142867316801509158746557017170898172904652474806322046435329 17028411169391936144158903235073802166688957103950205730735890718452977558415] [0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0] [218882428
71839275222246405745257275088548364400416034343698204186575808495407 20064222632519335620392538599819168831169334033714698148390020504361157788229 16932987888353439303876733333483753089335331904210737679777638516559340738329 138929541561535399674536214244202426603702812930
41844020930665712868256226070 18491005176105887713793578186878802184179920342434795679936712078451021551892 13246946988061061358463710143744246694215208038168454118425683992083900766648 16735385695760445847009231059394624911452603614484759591952585284319420245603 533525920
0010823335422561400406460802833663822601408371276437270477853320807 13505349855298052802059674656014645080677237340117810079427676680397642950247 13125345638773065385006785667381706277056606013721698372071805079936257386087] [21888242871839275222246405745257275088548364400
416034343698204186575808495407 20064222632519335620392538599819168831169334033714698148390020504361157788229 16932987888353439303876733333483753089335331904210737679777638516559340738329 13892954156153539967453621424420242660370281293041844020930665712868256226070 18491005
176105887713793578186878802184179920342434795679936712078451021551892 13246946988061061358463710143744246694215208038168454118425683992083900766648 16735385695760445847009231059394624911452603614484759591952585284319420245603 53352592000108233354225614004064608028336638226
01408371276437270477853320807 13505349855298052802059674656014645080677237340117810079427676680397642950247 13125345638773065385006785667381706277056606013721698372071805079936257386087] [120 7296080957279758407415468581752425029516121466805344781232734728858602831538 4742
452622231842964820054578139076269185478953423474107801277573758091841091 20408759788835324211853824616179700013118706436313839429726010755446147365596 19729818921977346693385996289822182683983178466486119845916853495955138491269 14728963432508512284969977199412708028335670
211113289777113583233883304466824 4225646887757860077628125553598279496261420349524762185797292197130607473462 13740952469543545000632465828967067138922028762483399337988317072683702000026 4134445875791863097535432196326374183392468831189695376031883013019874938061 2043409
3403270323372990683882088562600026748524740246876883068862143573070098] [21888242871839275222246405745257275088548364400416034343698204186575808495572 14852736234462365329381489612853150952943532985996594733223781412319298622152 10240570772181946621836711259388225130713699
058766073210801659815862253260307 5606690783242957204746053535144274233991257627170059590816345556521700152436 16674585021116447860280768821213354702873302602261381718775645828245598555310 3480838623368884740204463135877719441164982949788383239379783860226291767713 5502461
055281817798925832555404953876426741606215697522513020774680862969036 9832337671000817282374179088742553730252677976694821776613637753652307625809 11358868895098338163449498854508016467082985533588916235306081954362902206406 100038824316035258877866261178968518187442772611
82209347362859790432018823344] [0 0 0 0 0 0 0 0 0 0]]
[[378 11091780217197918436035182593926008050823913229893363435278812200895756923139 16188179623964463966453070915763193034238894504474358733360130179655025034332 12050355932914600980766057483982303839258686728161142364373972752408826667274 172331912194081793633485156345037
74744195630652063665928588777080923835334879 7253013812740259833482807829707358631772449916480452121065735947472979134553 109061210142671388869417913973765343663912119067568406001113514933241765691399 1288395248408528766388842666222154419894711925685864455416097997754792960
3901 21337236757878043467519716711739513823298448977141672368032190365906799427585 497383296740483530320336920677335456294559669438466212516907494826124660645] [0 0 0 0 0 0 0 0 0 0] [10 9467533623136829362003405659654932478776871903354554537552001017209377484196 1003645422
1591381952105445174065389430483188517730447493707648784757280443145 14547043953613375451663387804219542404253158452317944870861946924837028457072 10997322026233135849510607331037248976781070585903472810878924117352498226793 7038944770764516921876347032777469772399494269277
309192588594945879731933457 7736885848448743811196819808566634041716053805424834361765545785393810086298 20296929976543685057692545856667720879481777788831555127275568541596082471225 15005823647405288831679939176845277728116415891773314021044289685648028235016 143142762914
59418873503530882452159730733790787466739302816501130205205140874] [21888242871839275222246405745257275088548364400416034343698204186575808495572 12116705875482455926600688894695991566874987435944590440261505888997322560183 3498210244695741307841166632500939500759068953280
776917430337990533097964821 3339694200088175128156247225814056456963034171412684605219031551483491415356 6699474337335528162642432869598711107831730284363449400684016316134227496125 10218313382356911644552184904339724429358776366791443810799387336406657820266 1291254327751
9072431457167833747260547029052470939875815952516983677881470157 15600801677850126273237033936190564769213861511387797494275768648656140489758 20666257884128208542552044154667639021476877092247966553329537082806410060011 2500970607900881471946656529474212954686069315295552
336840714947608004705836] [120 4169189118445576232808839189572814302580640838174482732132991273633487332187 15512857844882914899576222484535513217518801118707554499144838363977934116656 8225462698266584775979105651094698217799722653648406556937380700272734382903 3921643514
537870143985814362691928453364915288407872819912594916761499022187 19471416054740355249790031777551784297521149164536763884914860807641396307548 7569683993177749347693548653568140968122976021810545210528962281190800438069 260574319902848514550552449348300893911290052385905
17075831195460209295828 16772300391080016053237225989718967538090036371906096157810012811221383414697 11884360440235749334459779627360423269804087139233824996335344396143789672273] [21888242871839275222246405745257275088548364400416034343698204186575808495407 9120101196599
698009269335727190531286895151833506680976540918411073253540403 17692996321403414137982511310749630696576594557002961094489381717482111866693 13670018349125547349449259884422251895579510914889458485948554373930910028387 17768997164708411621393089108476218457300720822282183
435960556037574388980009 16431382322540455913366919868488273868556098553367870226067888003950311794303 16811386539065443330419808857121212672176729879763981933423759604411697358436 3100834406843897323151574147244780637544351623392271532023912259764906203546 133913485903405
56576943807959424763439591047942198976567220915200259227280999 1454149468568951849255721863168712488521615875675787466815135324432235425519] [0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0]]
[[0 0 0 0 0 0 0 0 0 0] [21888242871839275222246405745257275088548364400416034343698204186575808495572 14852736234462365329381489612853150952943532985996594733223781412319298622152 10240570772181946621836711259388225130713699058766073210801659815862253260307 560669078324295
7204746053535144274233991257627170059590816345556521700152436 16674585021116447860280768821213354702873302602261381718775645828245598555310 3480838623368884740204463135877719441164982949788383239379783860226291767713 55024610552818177989258325554049538764267416062156975225
13020774680862969036 9832337671000817282374179088742553730252677976694821776613637753652307625809 11358868895098338163449498854508016467082985533588916235306081954362902206406 10003882431603525887786626117896851818744277261182209347362859790432018823344] [0 0 0 0 0 0 0 0 0
 0] [10 9467533623136829362003405659654932478776871903354554537552001017209377484196 10036454221591381952105445174065389430483188517730447493707648784757280443145 14547043953613375451663387804219542404253158452317944870861946924837028457072 10997322026233135849510607331037
248976781070585903472810878924117352498226793 7038944770764516921876347032777469772399494269277309192588594945879731933457 7736885848448743811196819808566634041716053805424834361765545785393810086298 2029692997654368505769254585666772087948177778883155512727556854159608247
1225 15005823647405288831679939176845277728116415891773314021044289685648028235016 14314276291459418873503530882452159730733790787466739302816501130205205140874] [21888242871839275222246405745257275088548364400416034343698204186575808495572 12116705875482455926600688894695
991566874987435944590440261505888997322560183 3498210244695741307841166632500939500759068953280776917430337990533097964821 3339694200088175128156247225814056456963034171412684605219031551483491415356 6699474337335528162642432869598711107831730284363449400684016316134227496
125 10218313382356911644552184904339724429358776366791443810799387336406657820266 12912543277519072431457167833747260547029052470939875815952516983677881470157 15600801677850126273237033936190564769213861511387797494275768648656140489758 20666257884128208542552044154667639
021476877092247966553329537082806410060011 2500970607900881471946656529474212954686069315295552336840714947608004705836] [120 4169189118445576232808839189572814302580640838174482732132991273633487332187 1551285784488291489957622248453551321751880111870755449914483836397793
4116656 8225462698266584775979105651094698217799722653648406556937380700272734382903 3921643514537870143985814362691928453364915288407872819912594916761499022187 19471416054740355249790031777551784297521149164536763884914860807641396307548 756968399317774934769354865356814
0968122976021810545210528962281190800438069 26057431990284851455055244934830089391129005238590517075831195460209295828 16772300391080016053237225989718967538090036371906096157810012811221383414697 1188436044023574933445977962736042326980408713923382499633534439614378967227
3] [21888242871839275222246405745257275088548364400416034343698204186575808495407 9120101196599698009269335727190531286895151833506680976540918411073253540403 17692996321403414137982511310749630696576594557002961094489381717482111866693 136700183491255473494492598844222518
95579510914889458485948554373930910028387 17768997164708411621393089108476218457300720822282183435960556037574388980009 16431382322540455913366919868488273868556098553367870226067888003950311794303 168113865390654433304198088571212126721767298797639819334237596044116973584
36 3100834406843897323151574147244780637544351623392271532023912259764906203546 13391348590340556576943807959424763439591047942198976567220915200259227280999 1454149468568951849255721863168712488521615875675787466815135324432235425519] [252 21888242871839275222246405745257
27508854836440041603434369820418657580848874 19152212512859365819465605027100115702479818850364030050735928663253832434413 8056089390329733241521246559018302636757384119597568195944477929781373959758 1284414251854457469638764781579333156237733883218857570862846009559483206
8759 16636584599463949118575479922350060822511239469621770548040058668199459998893 12160134928799597345692447636254041715860202444675574635387891214764338053125 8132090233634730724931824356744890397481510384876790537415652249873651073024 21508238655314287805193516756624336
284927733074019922636342332586114422931457 8762897233066209837239620077875568811491758386694335971626399106639551109530] [21888242871839275222246405745257275088548364400416034343698204186575808495407 2006422263251933562039253859981916883116933403371469814839002050436115778
8229 16932987888353439303876733333483753089335331904210737679777638516559340738329 13892954156153539967453621424420242660370281293041844020930665712868256226070 18491005176105887713793578186878802184179920342434795679936712078451021551892 1324694698806106135846371014374424
6694215208038168454118425683992083900766648 16735385695760445847009231059394624911452603614484759591952585284319420245603 5335259200010823335422561400406460802833663822601408371276437270477853320807 13505349855298052802059674656014645080677237340117810079427676680397642950
247 13125345638773065385006785667381706277056606013721698372071805079936257386087] [130 14331587594656668300280384714156549165120952881224784391707157503115112705101 20685258094954457913404688604099285961657908658567772138697330662829479338625 14530637422360233137784278946
297612347969855004575128619369756912880600381684 5798864344171307984227085966538646143250834040804664654250600623040743709170 16518783292341203019289084335873849793388843758438975918759738472056430474016 755448382451674985101143309402282341597815076875470074223472741717234
5015504 2350597510790279308341441886829464313824762347564519561215605757139713561161 9737879480036034695849603825853794656209835332704931152213749670109048928432 17933122795369441901044027352614349645340679209444694540042353914535568364262] [2188824287183927522224640574525
7275088548364400416034343698204186575808495616 1242070924870244586024300008560234260977149249706147980614620316936643101143 17577909330112989377389350645620797801765774783865852977404460604199520810713 56843805341769546322324219505981195005092533650110424287646570837271389
78802 16245180256443212079011004264058133854781989203433775489463510919724232867840 6398004325559038145113807605283246323626029432089200779515024846434861615241 570006324787481125579333482949408205430946989594167561033807400692078346240 989820506853181510133001318010559645
621358740660348113922198883106561445693 9383389832334867862513123097886210315118017823938272659494629448535784966437 7573966580379856348742874862805115357814573612949295040881703056370603354743]]

Lagrange Interpolation

(Polynomial Interpolation)

- for a group of points, we can find the smallest degree polynomial that goees through all that points
  - this polynomial is unique for each group of points



$$L(x) = \sum_{j=0}^{n} y_j l_j(x)$$

$$\ell_j(x) := \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)},$$

Shamir's Secret Sharing

(small overview, is offtopic here, but is interesting)

- from a secret to be shared, we generate 5 parts, but we can specify a number of parts that are needed to recover the secret
- so for example, we generate 5 parts, where we will need only 3 of that 5 parts to recover the secret, and the order doesn't matter
- we have the ability to define the thresholds of M parts to be created, and N parts to be able the recover

Shamir's Secret Sharing - Secret generation

- we want that are necessary n parts of m to recover s
  - where $n < m$
- need to create a polynomial of degree $n - 1$
  $f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + ... + +\alpha_{n-1} x^{n-1}$
- where α_0 is the secret s
- α_i are random values that build the polynomial
  *where α_0 is the secret to share, and α_i are the random values inside the $FiniteField$

$$f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + ... + +\alpha_{n-1} x^{n-1}$$

- the packets that we will generate are $P = (x, f(x))$
  - where x is each one of the values between 1 and m
    - $P_1 = (1, f(1))$
    - $P_2 = (2, f(2))$
    - $P_3 = (3, f(3))$
    - ...
    - $P_m = (m, f(m))$

Shamir's Secret Sharing - Secret recovery

- in order to recover the secret s, we will need a minimum of n points of the polynomial
  - the order doesn't matter
- with that n parts, we do Lagrange Interpolation/Polynomial Interpolation, recovering the original polynomial

QAP

$(\alpha_1(x)s_1 + \alpha_2(x)s_2 + ... + \alpha_n(x)s_n) \cdot (\beta_1(x)s_1 + \beta_2(x)s_2 + ... + \beta_n(x)s_n) - (\gamma_1(x)s_1 + \gamma_2(x)s_2 + ... + \gamma_n(x)s_n) = P(x)$

|----------------------- $A(x)$ -----------------------|------------------------ $B(x)$ -----------------------|------------------------ $C(x)$ ------------------------|

- $P(x) = A(x)B(x) - C(x)$
- $P(x) = Z(x)h(x)$
- $Z(x)$: divisor polynomial
  - $Z(x) = (x - x_1)(x - x_2)...(x - x_m) => ... => (x_1, 0), (x_2, 0), ..., (x_m, 0)$
    - optimizations with FFT
- $h(x) = P(x)/Z(x)$

The following explanation is for the Pinocchio protocol, all the examples will be for this protocol. The Groth16 is explained also in the end of this slides.

Trusted Setup

- concept
- τ (Tau)
- "Toxic waste"
- Proving Key
- Verification Key

$$g_1 t^0, g_1 t^1, g_1 t^2, g_1 t^3, g_1 t^4, \ldots$$
$$g_2 t^0, g_2 t^1, g_2 t^2, g_2 t^3, g_2 t^4, \ldots$$

Proving Key:

$pk = (C, pk_A, pk'_A, pk_B, pk'_B, pk_C, pk'_C, pk_H)$ where:

- $pk_A = \{A_i(\tau)\rho_A P_1\}_{i=0}^{m+3}$
- $pk'_A = \{A_i(\tau)\alpha_A \rho_A P_1\}_{i=n+1}^{m+3}$
- $pk_B = \{B_i(\tau)\rho_B P_2\}_{i=0}^{m+3}$
- $pk'_B = \{B_i(\tau)\alpha_B \rho_B P_1\}_{i=0}^{m+3}$
- $pk_C = \{C_i(\tau)\rho_C P_1\}_{i=0}^{m+3} = \{C_i(\tau)\rho_A \rho_B P_1\}_{i=0}^{m+3}$
- $pk'_C = \{C_i(\tau)\alpha_C \rho_C P_1\}_{i=0}^{m+3} = \{C_i(\tau)\alpha_C \rho_A \rho_B P_1\}_{i=0}^{m+3}$
- $pk_K = \{\beta(A_i(\tau)\rho_A + B_i(\tau)\rho_B C_i(\tau)\rho_A \rho_B)P_1\}_{i=0}^{m+3}$
- $pk_H = \{\tau^i P_1\}_{i=0}^{d}$

where:

- d: degree of polynomial $Z(x)$
- m: number of circuit signals

Verification Key:

$vk = (vk_A, vk_B, vk_C, vk_\gamma, vk^1_{\beta\gamma}, vk^2_{\beta\gamma}, vk_Z, vk_{IC})$

- $vk_A = \alpha_A P_2, vk_B = \alpha_B P_1, vk_C = \alpha_C P_2$
- $vk_{\beta\gamma} = \gamma P_2, vk^1_{\beta\gamma} = \beta\gamma P_1, vk^2_{\beta\gamma} = \beta\gamma P_2$
- $vk_Z = Z(\tau)\rho_A\rho_B P_2, vk_{IC} = (A_i(\tau)\rho_A P_1)_{i=0}^n$

```go
type Pk struct { // Proving Key pk:=(pkA, pkB, pkC, pkH)
        G1T [][3]*big.Int // t encrypted in G1 curve, G1T
        A   [][3]*big.Int
        B   [][3][2]*big.Int
        C   [][3]*big.Int
        Kp  [][3]*big.Int
        Ap  [][3]*big.Int
        Bp  [][3]*big.Int
        Cp  [][3]*big.Int
        Z   []*big.Int
}

type Vk struct {
        Vka   [3][2]*big.Int
        Vkb   [3]*big.Int
        Vkc   [3][2]*big.Int
        IC    [][3]*big.Int
        G1Kbg [3]*big.Int    // g1 * Kbeta * Kgamma
        G2Kbg [3][2]*big.Int // g2 * Kbeta * Kgamma
        G2Kg  [3][2]*big.Int // g2 * Kgamma
        Vkz   [3][2]*big.Int
}
```

```go
// Setup is the data structure holding the Trusted Setup d
type Setup struct {
	Toxic struct {
		T      *big.Int // trusted setup secret
		Ka     *big.Int
		Kb     *big.Int
		Kc     *big.Int
		Kbeta  *big.Int
		Kgamma *big.Int
		RhoA   *big.Int
		RhoB   *big.Int
		RhoC   *big.Int
	}
	Pk Pk
	Vk Vk
}
```

Proofs generation

- A, B, C, Z (from the QAP)
- random $\delta_1, \delta_2, \delta_3$
- $H(z) = \dfrac{A(z)B(z) - C(z)}{Z(z)}$
  - $A(z) = A_0(z) + \sum_{i=1}^{m} s_i A_i(x) + \delta_1 Z(z)$
  - $B(z) = B_0(z) + \sum_{i=1}^{m} s_i B_i(x) + \delta_2 Z(z)$
  - $C(z) = C_0(z) + \sum_{i=1}^{m} s_i B_i(x) + \delta_2 Z(z)$

    (where m is the number of public inputs)

- $\pi_A =< c, pk_A >$
- $\pi'_A =< c, pk'_A >$
- $\pi_B =< c, pk_B >$
  - example:

```
for i := 0; i < circuit.NVars; i++ {
        proof.PiB = Utils.Bn.G2.Add(proof.PiB, Utils.E
        proof.PiBp = Utils.Bn.G1.Add(proof.PiBp, Utils
}
```

$(c = 1 + witness + \delta_1 + \delta_2 + \delta_3$

- $\pi'_B =< c, pk'_B >$
- $\pi_C =< c, pk_C >$
- $\pi'_C =< c, pk'_C >$
- $\pi_K =< c, pk_K >$
- $\pi_H =< h, pk_K H >$
- proof: $\pi = (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi_H$

Proofs verification

- $vk_{kx} = vk_{IC,0} + \sum_{i=1}^{n} x_i vk_{IC,i}$

Verification:

- $e(\pi_A, vk_a) == e(\pi_{A'}, g_2)$
- $e(vk_b, \pi_B) == e(\pi_{B'}, g_2)$
- $e(\pi_C, vk_c) == e(\pi_{C'}, g_2)$
- $e(vk_{kx} + \pi_A, \pi_B) == e(\pi_H, vk_{kz}) \cdot e(\pi_C, g_2)$
- $e(vk_{kx} + \pi_A + \pi_C, V^2_{\beta\gamma}) \cdot e(vk^1_{\beta\gamma}, \pi_B) == e(\pi_k, vk^1_{\gamma})$

Example (whiteboard):

$$\frac{e(\pi_A, \pi_B)}{e(\pi_C, g_2)} = e(g_1 h(t), g_2 z(t))$$

$$\frac{e(A_1 + A_2 + ... + A_n, B_1 + B_2 + ... + B_n)}{e(C_1 + C_2 + ... + C_n, g_2)} = e(g_1 h(t), g_2 z(t))$$

$$\frac{e(g_1\alpha_1(t)s_1 + g_1\alpha_2(t)s_2 + ... + g_1\alpha_n(t)s_n, g_2\beta_1(t)s_1 + g_2\beta_2(t)s_2 + ... + g_2\beta_n(t)s_n)}{e(g_1\gamma_1(t)s_1 + g_1\gamma_2(t)s_2 + ... + g_1\gamma_n(t)s_n, g_2)} = e(g_1 h(t), g_2 z(t))$$

$$\begin{aligned}&e(g_1\alpha_1(t)s_1 + g_1\alpha_2(t)s_2 + ... + g_1\alpha_n(t)s_n, g_2\beta_1(t)s_1 + g_2\beta_2(t)s_2 + ... + g_2\beta_n(t)s_n)\\ &= e(g_1 h(t), g_2 z(t)) \cdot e(g_1\gamma_1(t)s_1 + g_1\gamma_2(t)s_2 + ... + g_1\gamma_n(t)s_n, g_2)\end{aligned}$$

Groth16

Trusted Setup

$\tau = \alpha, \beta, \gamma, \delta, x$

$\sigma_1 =$

- $\alpha, \beta, \delta, \{x^i\}_{i=0}^{n-1}$
- $\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\}_{i=0}^{l}$
- $\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\}_{i=l+1}^{m}$
- $\{\frac{x^i t(x)}{\delta}\}_{i=0}^{n-2}$

$\sigma_2 = (\beta, \gamma, \delta, \{x^i\}_{i=0}^{n-1})$

(where $u_i(x), v_i(x), w_i(x)$ are the QAP)

```
type Pk struct { // Proving Key
	BACDelta [][3]*big.Int // {( βui(x)+αvi(x)+wi(x) )
	Z        []*big.Int
	G1        struct {
		Alpha    [3]*big.Int
		Beta     [3]*big.Int
		Delta    [3]*big.Int
		At       [][3]*big.Int // {a(τ)} from 0 to
		BACGamma [][3]*big.Int // {( βui(x)+αvi(x)
	}
	G2 struct {
		Beta     [3][2]*big.Int
		Gamma    [3][2]*big.Int
		Delta    [3][2]*big.Int
		BACGamma [][3][2]*big.Int // {( βui(x)+αvi
	}
	PowersTauDelta [][3]*big.Int // powers of τ encryp
}
```

```go
type Vk struct {
        IC [][3]*big.Int
        G1 struct {
                Alpha [3]*big.Int
        }
        G2 struct {
                Beta  [3][2]*big.Int
                Gamma [3][2]*big.Int
                Delta [3][2]*big.Int
        }
}
```

```go
// Setup is the data structure holding the Trusted Setup d
type Setup struct {
	Toxic struct {
		T      *big.Int // trusted setup secret
		Kalpha *big.Int
		Kbeta  *big.Int
		Kgamma *big.Int
		Kdelta *big.Int
	}
	Pk Pk
	Vk Vk
}
```

Proofs Generation

$$\pi_A = \alpha + \textstyle\sum_{i=0}^{m} \alpha_i u_i(x) + r\delta$$
$$\pi_B = \beta + \textstyle\sum_{i=0}^{m} \alpha_i v_i(x) + s\delta$$

$$\pi_C = \frac{\sum_{i=l+1}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta} + \pi_A s + \pi_B r - rs\delta$$

$$\pi = \pi_A^1, \pi_B^1, \pi_C^2$$

Proof Verification

$$[\pi_A]_1 \cdot [\pi_B]_2 = [\alpha]_1 \cdot [\beta]_2 + \sum_{i=0}^{l} a_i [\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}]_1 \cdot [\gamma]_2 + [\pi_C]_1 \cdot [\delta]_2$$

$$e(\pi_A, \pi_B) = e(\alpha, \beta) \cdot e(pub, \gamma) \cdot e(\pi_C, \delta)$$

How we use zkSNARKs in iden3

- proving a credentials without revealing it's content
- proving that an identity has a claim issued by another identity, without revealing all the data
- proving any property of an identity
- ITF (Identity Transition Function), a way to prove with a zkSNARK that an identity has been updated following the defined protocol
    - identities can not cheat when issuing claims
- etc

Other ideas for free time side project

- Zendermint (Tendermint + zkSNARKs)

zkSNARK libraries

- bellman (rust)
- libsnark (c++)
- snarkjs (javascript)
- websnark (wasm)
- go-snark (golang) [do not use in production]

Circuit languages

language	snark library with which plugs in
Zokrates	libsnark, bellman
Snarky	libsnark
circom	snarkjs, websnark, bellman
go-snark-circuit	go-snark

Utilities (Elliptic curve & Hash functions) inside the zkSNARK

- we work over F_r, where $r =$ `21888242871839275222246405745257275088548364400416034343698204186575808495617`
- BabyJubJub
- Mimc
- Poseidon

Utilities (Elliptic curve & Hash functions) inside the zkSNARK

BabyJubJub

- explaination: https://medium.com/zokrates/efficient-ecc-in-zksnarks-using-zokrates-bd9ae37b8186
- implementations:
  - go: https://github.com/iden3/go-iden3-crypto
  - javascript & circom: https://github.com/iden3/circomlib
  - rust: https://github.com/arnaucube/babyjubjub-rs
  - c++: https://github.com/barryWhiteHat/baby_jubjub_ecc

Utilities (Elliptic curve & Hash functions) inside the zkSNARK

Mimc7

- explaination: https://eprint.iacr.org/2016/492.pdf
- implementations in:
  - go: https://github.com/iden3/go-iden3-crypto
  - javascript & circom: https://github.com/iden3/circomlib
  - rust: https://github.com/arnaucube/mimc-rs

Utilities (Elliptic curve & Hash functions) inside the zkSNARK

Poseidon

- explaination: https://eprint.iacr.org/2019/458.pdf
- implementations in:
  - go: https://github.com/iden3/go-iden3-crypto
  - javascript & circom: https://github.com/iden3/circomlib

References

- `Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture`, Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, Madars Virza https://eprint.iacr.org/2013/879.pdf
- `Pinocchio: Nearly practical verifiable computation`, Bryan Parno, Craig Gentry, Jon Howell, Mariana Raykova https://eprint.iacr.org/2013/279.pdf
- `On the Size of Pairing-based Non-interactive Arguments`, Jens Groth https://eprint.iacr.org/2016/260.pdf
- (also all the links through the slides)

Thank you very much

iden3.io
github.com/iden3
twitter.com/identhree

arnaucube.com
github.com/arnaucube
twitter.com/arnaucube

2019-08-20