Lecture 12:
Verified computation and its applications, course conclusion

Eran Tromer
Verified computation using computational proofs
Motivation 1:
cloud computing
Integrity of data: digital signatures / message authentication codes
SNARKs for Clouds

\[ x = 314152; \]
\[ v = DB[x]; \]
\[ w = \text{func}(v, y); \]
\[ \text{return } w; \]
SNARK motivation 2: IT supply chain
IT supply chain threats

Can you trust the hardware and software you bought?

The New York Times

“F.B.I. Says the Military Had Bogus Computer Gear”

ars technica

“Chinese counterfeit chips causing military hardware crashes”

The New York Times

“A Saudi man was sentenced […] to four years in prison for selling counterfeit computer parts to the Marine Corps for use in Iraq and Afghanistan.”
Supply chain for the F-35 Joint Strike Fighter

73 chips

Made by the cheapest bidder
SNARK motivation 3: Privacy for Bitcoin

Zerocash
zerocash-project.org

[Ben-Sasson Chiesa Garman Gree Miers Tromer Virza 2014]
Bitcoin’s privacy problem

Bitcoin: decentralized **digital** currency. What’s to prevent double-spending?
Bitcoin’s privacy problem

Bitcoin: decentralized digital currency. What’s to prevent double-spending?

Solution: broadcast every transaction into a public ledger (blockchain):

The cost: privacy.

- **Consumer purchases** (timing, amounts, merchant) seen by friends, neighbors, and co-workers.
- **Account balance** revealed in every transaction.
- **Merchant’s cash flow** exposed to competitors.
Bitcoin’s privacy problem (cont.)

- Pseudonymous, but:
  - Most users use a single or few addresses
  - Transaction graph can be analyzed.

[Reid Martin 11] [Barber Boyen Shi Uzun 12] [Ron Shamir 12] [Ron Shamir 13] [Meiklejohn Pomarole Jordan Levchenko McCoy Voelker Savage 13] [Ron Shamir 14]

- Also: threat to the currency’s fungibility.
- Centralized: reveal to the bank.
- Decentralized: reveal to everyone?!
Zerocash: divisible anonymous payments

- Zerocash is a new privacy-preserving protocol for digital currency designed to sit on top of Bitcoin (or similar ledger-based currencies).

- Zerocash enables users to pay one another directly via payment transactions of variable denomination that reveal neither the origin, destination, or amount.
More about Zerocash

• Efficiency:
  – 288 byte proof per transactions (128-bit security)
  – <6 ms to verify a proof
  – <1 min to create for $2^{64}$ coins; asymptotically: $\log(#\text{coins})$
  – 896MB “system parameters” (fixed throughout system lifetime).

• Trust in initial generation of system parameters (once).

• Crypto assumptions:
  – Pairing-based elliptic-curve crypto
  – Less common: Knowledge of Exponent
    [Boneh Boyen 04] [Gennaro 04] [Groth 10] …
  – Properties of SHA256, encryption and signature schemes
I got the money from last night, and I haven’t spent it in any of my prior transactions.

Intuition: “virtual accountant” using cryptographic proofs.
Requisite proof properties

- zero knowledge
- succinct
- noninteractive
- argument of knowledge

 zkSNARK

NP witness

NP decision algorithm

NP statement

proof

Prover

Verifier

(blockchain size, verification by everyone) (even 2 rounds are too much)
Basic anonymous e-cash (#1)

Minting:

I hereby spend 1 BTC to create sn

Spending:

I’m using up a coin with (unique) sn

Legend:

In public ledger
Basic anonymous e-cash (#2) [Sander Ta-Shma 1999]

Minting:

I hereby spend 1 BTC to create cm

Spending:

I’m using up a coin with (unique) sn, and here are its cm and r.

Legend:

- In private wallet
- In public ledger
Basic anonymous e-cash (#3) [Sander Ta-Shma 1999]

Minting:
I hereby spend 1 BTC to create cm

Spending:
I’m using up a coin with (unique) sn, and I know r, and a cm in the tree with root, that match sn.

Legend:
- In private wallet
- In public ledger
- Proved to be known
Basic anonymous e-cash – requisite proofs

Spending:
I’m using up a coin with (unique) sn, and I know a cm in the tree, and r, that match sn.

Requires:
- zero knowledge
- succinct
- noninteractive argument of knowledge
- zkSNARK
zkSNARK

with great power comes great functionality

- cm (coin commitment)
- sn (serial number)
- $r$
I hereby spend $v$ BTC to create $cm$, and here is $k, r'$ to prove consistency.

I’m using up a coin with value $v$ (unique) $sn$, and I know $r', r''$ that are consistent with $cm$.

Adding variable denomination (#4)

Minting:

Spending:
Adding direct anonymous payments (#5)

CreateAddress: payee creates $a_{pk}, a_{sk}$

Minting, spending analogous to above.

Sending?

<table>
<thead>
<tr>
<th>cm</th>
<th>sn</th>
</tr>
</thead>
<tbody>
<tr>
<td>(coin commitment)</td>
<td>(serial number)</td>
</tr>
</tbody>
</table>

I’m using up a coin with value $v$ (unique) $sn$, and I know $r', r'', \rho, a_{pk}$ that are consistent with $cm$.

Unknown to payer
Sending direct anonymous payments

1. Create coin using $a_{pk}$ of payee.
2. Send coin secrets $(v, \rho, r', r'')$ to payee out of band, or encrypted to payee’s public key.
Pouring Zerocash coins (#6)

Single transaction type capturing:

- Sending payments
- Making change
- Exchanging into bitcoins
- Transaction fees

\[ \text{Pouring: old Zerocash coins} \rightarrow \text{new Zerocash coins} \rightarrow \text{public bitcoins} \]

- \( v_1 \) to \( \text{dest}_1 \)
- \( v_2 \) to \( \text{dest}_2 \)
- \( v_{\text{pub}} \)

\[ \text{proof of value } v_{\text{pub}} \]

\[ \text{the old coins were valid, and values of old coins } = v_1 + v_2 + v_{\text{pub}} \]
Pouring Zerocash coins

Single transaction type capturing:

Sending payments
Making change
Exchanging into bitcoins
Transaction fees

old Zerocash coin ↘

\[v_1\]

dest₁

\[v_2\]

dest₂

\[v_{\text{pub}}\]

old Zerocash coin ↗

new Zerocash coin ↗

value \[v_1\] to dest₁

new Zerocash coin ↗

value \[v_2\] to dest₂

public bitcoins of value \[v_{\text{pub}}\]

\[s_{n_1}\ s_{n_2}\ c_{m_1}\ c_{m_2}\ \ldots\ \text{proof}\]

the old coins were valid, and values of old coins = \[v_1 + v_2 + v_{\text{pub}}\]
Example of a Zerocash Pour transaction

| root | 1c4ac4a110e863deeca050dc5e5153f2b7010af9 |
| sn_1 | a365e7006656f1432df9096b46cc7f1fd2b9949367180fd8de40909e30b7fd |
| sn_2 | 6937031dce13facdebe79e8e2712ffad2e980c9114ececa9a9b25fc88df73b52 |
| cm_1 | a4d015440f9caea0c3ca3a83cf04058262d74b50cb14ed063e047694580103 |
| cm_2 | 2ca1f833b63ac827ba6ae69b53edc855e66e2c2d0a24f6ed5b04f5a0542dc772 |
| v_pub | 0000000000000042 |
| pubkeyHash info | 8f9a43f0fe285f052ec209724bb0e502ff5427 |
| SigPK | 2dd489d97daa8ce006c6049e1699ba16a610d8d43 |
| Sig | f1d2d2f924e98ac86f6f7b36c94bdcf32beec15a3835982f3d2bb3a324b6bd7b8ce116bad69e |
| MAC_1 | b8a5917e1a587a907bc9e3ec5e395240ce1bf700276ec0fa92d1835cb7f629 |
| MAC_2 | ade6218b3a17d60993656947b2bb446f12698d4bcafa85fc6f39f5b46603a |
| ciphertext_1 | 048070fe125bda93ae6a70c8b65ad2b2a4384868d7243c74e80abc5b74dfe3524a987a2e3ed075d54a753a866973eaa5070c4e08954ff5d80caae214ce5752f42d6676f0e59d5bded68ad33b0c73cf9ace671d8f0126d86e667b319d255d7002d0a028effd8648057afa823a25dd3e5f255c6ed55e229db556186e46967baf4d2303af7fe09d24b8e30277336cb7d8c81d3c786f1547fe0d00c29b63bd9272aad87b3f1a2b667f7a575e |
| ciphertext_2 | 0493110814319b0b5cabb9a9225062354987c88bf60496985ca52c71a7705554979a50099e5c5a359bdf0411983388fa5de840a0d6481e6fd9f3864d2d127986af981767f420ca19ad2c1879abc14bf9d78642ea08ac272063e66f78bc426e0edffddedbe60bea586eacec46b017069c8be2ebe8e8a2fa5e0f6780a4e246672b2c343e873820b2d2e4b954e9216b566c140de79351af47254d122a35f17f840156bd7b1feeb942792dc |
| zkSNARKproof | a4c3cad60e2e55c1d8a37ebc51885cf6c5da40bb1c1c0bf3ed97b778277fbaadce2b40c4a0cc3f285436df1e1adcecfcc533bioaafefef9d3975726f2aca8292286ca88d49ada2b13f99c81faa213b8254485b14ce7a0ce5f7592eed233da3e276b9deb5a365947869f17002b095f7058d611e206c2087618c5820e36585f0cc0084340f355139d01803e2d94322876997e6ed7832a5fc5dc30874ff09829d688e7e15123e0f0a1a5b649edf2cf58acf05d8c7563741298025806d6be9ce58c40d1b4fe87dadcb1146769e6154fb623d3fba9e7c8ad17088b17992715df431c9451e0b597d6506adad84e98475d4be530eb501925df2d2981a2970a37995253b99a98e50dd00eab5306c10be5 |

~1KB total. Less without direct payments and public outputs.
Decentralized Anonymous Payment (DAP) system

Algorithms:

Setup  CreateAddress

Mint  Pour

VerifyTransaction  Receive

Security:

1. **Ledger indistinguishability**
   Nothing revealed beside public information, even by chosen-transaction adversary.

2. **Balance**
   Can’t own more money than received or minted.

3. **Transaction non-malleability**
   Cannot manipulate transactions en route to ledger.

*(Requires further changes to the construction.)*
### Zerocash Implementation

**Network simulation**
- third-scale Bitcoin network on EC2

**Bitcoind + Zerocash hybrid currency**
- **libzerocash** provides DAP interface
- **Statement for zkSNARK**
  - Hand-optimized
- **libsnaerk**
  - zkSNARK
  - Instantiate Zerocash primitives and parameters
  - **SCIPR LAB**

**Performance (quadcore desktop)**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time/Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>&lt;2 min, 896MB params</td>
</tr>
<tr>
<td>Mint</td>
<td>23 $\mu$s, 72B transaction</td>
</tr>
<tr>
<td>Pour</td>
<td>46 s, 1KB transaction</td>
</tr>
<tr>
<td>Verify Transaction</td>
<td>&lt;9 ms/transaction</td>
</tr>
<tr>
<td>Receive</td>
<td>&lt;2 ms/transaction</td>
</tr>
</tbody>
</table>
Trusted setup

- **Setup** generate fixed keys used by all provers and verifiers.
- If **Setup** is compromised at the dawn of the currency, attacker could later forge coins.
- Ran once. Once done and intermediate results erased, no further trust (beyond underlying cryptographic assumptions)
- Anonymity is unaffected by corrupted setup
- Can be done by an MPC protocol, secure if even one of the participants is honest.

[Ben-Sasson Chiesa Green Tromer Virza 2015]
Other applications of zk-SNARK for Bitcoin

• Lightweight clients
  – Proof of transaction validity:
    “This transaction is valid with respect to block chain head $H$.”
  – Blockchain compression
    “Here’s a summary of the 24GB blockchain with head $H$.

• Turing-complete scripts/contracts with cheap verification (e.g., Ethereum)

• Proof of reserve
  “I own $N$ bitcoins.”

• ... and many other amazing ideas on the Bitcoin forums
Building SNARKs
zkSNARK for NP: setting

Prover $P$  

Verifier $V$

function $f(x, w) = y$

proof $\pi$

witness $w$, internal values $z$
zkSNARK for NP: setting

**zero knowledge**

**succinct** (\(V\) and \(\pi\))

**noninteractive argument of knowledge**

\[ f(x, w) = y \]

Prover \(P\)

Verifier \(V\)

**proof \(\pi\)**
Preprocessing zkSNARK for NP: setting

Variants:
- Dependence on $f$
- Cheap / expensive
- Secret / public randomness
- Publicly-verifiable / designated-verifier

Prover $P$

Generator $G$

Proving key $pk$

Verification key $vk$

\[ f(x, w) = y \]

\[ \text{proof } \pi \]
SNARK constructions for general NP statement

- Preprocessing zkSNARK
  - Theory
    - [Groth 10] [Lipmaa 12] [Gennaro Gentry Parno Raykova 13]
      - [Bitansky Chiesa Ishai Ostrovsky Paneth 13]
      - [Danezis Fournet Jens Groth Kohlweiss 14]
  - Implementations
    - "SNARKs for C"
      Execution of C programs can be proved in 288 bytes and verified in 6 ms.
    - [Ben-Sasson Chiesa Genkin Tromer Virza 13]
      - [BFRSVW13] [BCGGMTV14] [FL14]
    - Trusted generation of proving+verification keys

- PCP-based SNARKs
  - Theory
    - [BFLS 91] [Kilian 92] [Micali 94]
    - [... PCP literature ...]
    - [Bitansky Canetti Chiesa Tromer 11]
    - [Ben-Sasson Chiesa Genkin Tromer 13]
    - [Bitansky Canetti Chiesa Goldwasser Lin Rubinstein Tromer 14]
  - No trust assumption
Which SNARK?

**“Long” keys**
- Lipmaa14
- ZPK14
- BBFR15
- Lipmaa12
- BCGTV13
- Lipmaa13
- DFGK14
- Groth10
- BCIOP13

**“Short” keys**
- Kilian92
- Micali94
- Valiant08
- BC12
- GLR11
- DL08
- BCCT13
- BCCGLRT14
- BCTV14
- CTV15
- CFHKKNPZ14

**Used by Zerocash (libsnark implementation)**
- BCTV14
- Groth10
- FLZ13
- GGPR13
- KPPSST14
- WSRBW15
- PGHR13
- CTV15
- CFHKKNPZ14
## Preprocessing SNARKs for NP

<table>
<thead>
<tr>
<th>Source</th>
<th>Proof size (field elements)</th>
<th>CRS size</th>
<th>Prover runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Groth10]</td>
<td>42</td>
<td>$O(s^2)$</td>
<td>$O(s^2)$</td>
</tr>
<tr>
<td>[Lipmaa12]</td>
<td>39</td>
<td>$\tilde{O}(s)$</td>
<td>$O(s^2)$</td>
</tr>
<tr>
<td>QAP-based</td>
<td>7–8</td>
<td>$O(s)$</td>
<td>$\tilde{O}(s)$</td>
</tr>
<tr>
<td>[GGPR13]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Preprocessing is</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>private-coin and costs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\tilde{O}(s)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[DFGK14]</td>
<td>4</td>
<td>$O(s)$</td>
<td>$\tilde{O}(s)$</td>
</tr>
</tbody>
</table>
zkSNARK construction via QAP and Linear PCPs

- Computation
- Algebraic Circuit
- R1CS
- QAP
- Linear PCP
- Linear Interactive Proof
- zkSNARK
Efficient computation $f(\cdot)$.

- Deterministic $f(x) \rightarrow y$
- Nondeterministic: $\exists w: f(x, w) \rightarrow y$

Arithmetic circuit $C(\cdot; \cdot)$ over $\mathbb{F}$.

$\exists z: C(x, y)$ accepts with internal values $z \in \mathbb{F}^n$
Arithmetic Circuit ⇒ R1CS (Rank-1 Quadratic System) [GGPR13]

Arithmetic circuit $C(\cdot,\cdot)$ over $\mathbb{F}$.

$\exists z: C(x, y)$ accepts with internal values $z \in \mathbb{F}^n$

Completeness

Soundness, PoK

$R1CS \left( a_j, b_j, c_j \right)_{j=1}^m$ vectors in $\mathbb{F}^k$.

$\exists z \in \mathbb{F}^n$:

$\forall j \in \{1, \ldots, m\}$:

$\begin{array}{c}
1 \\
x \\
y \\
z
\end{array} \cdot
\begin{array}{c}
\tau \\
a_j \\
y \\
z
\end{array} =
\begin{array}{c}
1 \\
x \\
y \\
z
\end{array} \cdot
\begin{array}{c}
\tau \\
b_j \\
y \\
z
\end{array} =
\begin{array}{c}
1 \\
x \\
y \\
z
\end{array} \cdot
\begin{array}{c}
\tau \\
c_j
\end{array}$
Expressing gates as constraints:

Multiplication gate in $C$ converted into a constraint:

Addition gate in $C$ converted into a constraint:

Generally, any bilinear gate.
R1CS (Rank-1 Quadratic Constraint System) $\Rightarrow$ QAP (Quadratic Arithmetic Program) [GGPR13]

**R1CS** $(a_j, b_j, c_j)_{j=1}^m$ vectors in $\mathbb{F}_k^m$.

$\exists z \in \mathbb{F}_n$:

$\forall j \in \{1, \ldots, m\}$:

$1 \begin{bmatrix} x \\ y \\ z \end{bmatrix} a_j \begin{bmatrix} x \\ y \\ z \end{bmatrix} = b_j \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 1 \begin{bmatrix} x \\ y \\ z \end{bmatrix} c_j$

**Matrices** $(A, B, C)$ in $\mathbb{F}_k^{k \times m}$.

$\exists z \in \mathbb{F}_n$:

$1 \begin{bmatrix} x \\ y \\ z \end{bmatrix} A \begin{bmatrix} x \\ y \\ z \end{bmatrix} - 1 \begin{bmatrix} x \\ y \\ z \end{bmatrix} C = 0^m$
R1CS $\Rightarrow$ QAP (cont.)

Matrices $(A, B, C)$ in $\mathbb{F}^{k \times m}$.

$\exists z \in \mathbb{F}^m: \tau A_1 \cdot \tau B \cdot \tau C = 0^m$

Intuition: multiples of $V(\alpha)$ are the polynomials with all of $\alpha_i, \ldots, \alpha_m$ as roots.

Fix $S = \{\alpha_1, \ldots, \alpha_m\} \subset \mathbb{F}$. For $i = 1, \ldots, k$ and $j = 1, \ldots, m$:
Let $A_i(\alpha)$ be the degree-$(m - 1)$ polynomial such that $A_i(\alpha_j) = A_{i,j}$.
Likewise $B_i(\alpha), C_i(\alpha)$.

Let $V(\alpha) = \prod_{j=1}^{m} (\alpha - \alpha_j)$, vanishing on $S$.

QAP: $(A_i(\alpha), B_i(\alpha), C_i(\alpha))_{i=1}^k$ and $V$ polynomials in $\mathbb{F}[\alpha]$.

Let $P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha)$

$\exists z \in \mathbb{F}^n: V(\alpha)$ divides $P_{x,y,z}(\alpha)$

i.e.,

$\exists H(\alpha): P_{x,y,z}(\alpha) = H(\alpha)V(\alpha)$
QAP ⇒ Linear PCP

QAP: \((A_i(\alpha), B_i(\alpha), C_i(\alpha))\)\(^k\)\(_{i=1}\) and \(V\) polynomials in \(\mathbb{F}[\alpha]\).

Let \(P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha)\)

\[\exists z \in \mathbb{F}^n:\]
\[\exists H(\alpha): P_{x,y,z}(\alpha) = H(\alpha)V(\alpha)\]

Probabilistic check: \(\tau \leftarrow_R \mathbb{F}\) and check \(P_{x,y,z}(\tau) = H(\tau) \cdot V(\tau)\).

Soundness: polynomial identity testing with degree \(< 2m \ll |\mathbb{F}|\)

Probabilistic check via linear queries

Let \(\pi = (1, x, y, z, h)\) where \(h\) is the coefficient vector of \(H\).

This check can be done by 4 linear queries to \(\pi\)

\(+ 5\)th for checking \(x, y\) via random linear combination.\)

• Any \(\tilde{\pi}\) still commits to some low-degree \(\tilde{H}(\tau) \cdot \tilde{P}_{x,y,z}(\tau)\).
**QAP \Rightarrow Linear PCP: the algorithms**

Let \( P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha) \)

\[
\begin{align*}
\text{Prover: compute } H & \text{ and its coefficient vector } h; \\
\text{Output } \pi = (1, x, y, z, h) & \text{ where } h \text{ is the coefficient vector of } H. \\
\text{Complexity: Dominated by computing the } m & \text{ coefficients of } H. \text{ With suitable FFT: } \sim m \log m + \#\text{"nonzero entries in } A, B, C \text{" field operations.}
\end{align*}
\]

\[
\begin{align*}
\text{Query: Verify: draw } \tau & \leftarrow \mathbb{F}, \text{ make linear queries to } \pi \text{ according to } \tau. \\
\text{Complexity: } \sim 4m + 2(\#\text{nonzero entries in } A, B, C) & \text{ field operations.}
\end{align*}
\]

\[
\begin{align*}
\text{Decision: check a simple quadratic equation in the answers.}
\end{align*}
\]

*Later: important for public verifiability (will use of pairings).*
QAP $\Rightarrow$ Linear PCP: adding ZK

Let $P_{x,y,z}(\alpha) = A_{x,y,z}(\alpha) \cdot B_{x,y,z}(\alpha) - C_{x,y,z}(\alpha)$

\[
\begin{array}{c}
1 \tau \\
\tau \\
\tau \\
\end{array}
\begin{array}{c}
A_1(\alpha) \\
A_2(\alpha) \\
A_k(\alpha) \\
\end{array}
\begin{array}{c}
1 \tau \\
\tau \\
\tau \\
\end{array}
\begin{array}{c}
B_1(\alpha) \\
B_2(\alpha) \\
B_k(\alpha) \\
\end{array}
\begin{array}{c}
1 \tau \\
\tau \\
\tau \\
\end{array}
\begin{array}{c}
C_1(\alpha) \\
C_2(\alpha) \\
C_k(\alpha) \\
\end{array}
\]

$\delta_1, \delta_2, \delta_3 \leftarrow_R \mathbb{F}$

$+ \delta_1 V(\alpha) + \delta_2 V(\alpha) + \delta_3 V(\alpha)$

Honest-Verifier Zero Knowledge:
Prover adds random multiple of $V(\alpha)$ to $A, B, C(\alpha)$.

- ZK: The queries to $A_{x,y,z}, B_{x,y,z}, C_{x,y,z}$ return random independent $\mathbb{F}$ elements. The query to $H$ follows from them. The $x,y$-consistency query is predictable.
Linear PCP $\Rightarrow$ SNARK

**Linear PCP**

Prove$(x, y, w) \rightarrow \pi \in \mathbb{F}^m$

$q_1, \ldots, q_k \in \mathbb{F}^m$

$a_1, \ldots, a_k \in \mathbb{F}$

$(a_i = \pi^\tau q_i)$

Verify$(x, y)$

Intuition: send $q_i$ in special encrypted form that restricts the prover to just linear functions.
zkSNARK construction via QAP and Linear PCPs

- Computation
- Algebraic Circuit
- R1CS
- QAP
- Linear PCP
- Linear Interactive Proof
- zkSNARK
Full [PGHR13] protocol ([BCTV14USENIX] variant)

(a) Key generator $G$

- **INPUTS:** circuit $C: \mathbb{F}_p^n \times \mathbb{F}_p^m \rightarrow \mathbb{F}_p^l$
- **OUTPUTS:** proving key pk and verification key vk

1. Compute $(\bar{A}, \bar{B}, \bar{C}, Z) := \text{QAPInst}(C)$; extend $\bar{A}, \bar{B}, \bar{C}$ via
   \[A_{m+1} = B_{m+2} = C_{m+3} = Z,\]
   \[A_{m+2} = A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0.\]

2. Randomly sample $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \alpha, \beta, \gamma \in \mathbb{F}_p$.

3. Set $pk := (C, pk_A, pk'_A, pk_B, pk'_B, pk_C, pk'_C, pk_K, pk'_K)$ where for $i = 0, 1, \ldots, n + 3$:
   \[pk_{A,i} := A_i(\tau) \rho_A P_i, \quad pk'_{A,i} := A_i(\tau) \alpha_A \rho_A P_i,\]
   \[pk_{B,i} := B_i(\tau) \rho_B P_i, \quad pk'_{B,i} := B_i(\tau) \alpha_B \rho_B P_i,\]
   \[pk_{C,i} := C_i(\tau) \rho_A \rho_B P_i, \quad pk'_{C,i} := C_i(\tau) \alpha_A \rho_B P_i,\]
   \[pk_{K,i} := \beta(A_i(\tau) \rho_A + B_i(\tau) \rho_B + C_i(\tau) \rho_A \rho_B) P_i,\]
   and for $i = 0, 1, \ldots, d$, $pk_{H,i} := \tau^i P_i$.

4. Set $vk := (vk_A, vk_B, vk_C, vk_\gamma, vk_{1,\gamma}, vk_{2,\gamma}, vk_2, vk_{1,2})$ where
   \[vk_A := \alpha_A \rho_B P_1, \quad vk_B := \alpha_B P_1, \quad vk_C := \alpha_C P_1,\]
   \[vk_\gamma := \gamma P_2, \quad vk_{1,\gamma} := \gamma \beta P_1, \quad vk_{2,\gamma} := \gamma \beta P_2,\]
   \[vk_2 := Z(\tau) \rho_A \rho_B P_2, \quad (vk_{2,i})_i = 0 := (A_i(\tau) \rho_A P_i)_{i=0}^n.\]

5. Output $(pk, vk)$.

(b) Prover $P$

- **INPUTS:** proving key pk, input $\vec{x} \in \mathbb{F}_p^n$, and witness $\vec{a} \in \mathbb{F}_r^n$
- **OUTPUTS:** proof $\pi$

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPInst}(C)$.
2. Compute $\vec{s} := \text{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_p^m$.
3. Randomly sample $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_p$.
4. Compute $\vec{h} = (h_0, h_1, \ldots, h_d) \in \mathbb{F}_r^{d+1}$, which are the coefficients of $H(z) := \frac{A(z)B(z)}{C(z)} - \vec{s}$ where $A, B, C \in \mathbb{F}_r[z]$ are as follows:
   \[A(z) := A_0(z) + \sum_{i=1}^n s_i A_i(z) + \delta_1 Z_i(z),\]
   \[B(z) := B_0(z) + \sum_{i=1}^n s_i B_i(z) + \delta_2 Z_i(z),\]
   \[C(z) := C_0(z) + \sum_{i=1}^n s_i C_i(z) + \delta_3 Z_i(z).\]

5. Set $pk'_{A,i} := "\text{same as } pk_A,"$ but with $pk'_{A,i} = 0$ for $i = 0, 1, \ldots, n$.

6. Letting $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{d+m}$, compute
   \[\pi_A := \langle \vec{c}, pk_A \rangle, \quad \pi'_{A} := \langle \vec{c}, pk'_A \rangle, \quad \pi_B := \langle \vec{c}, pk_B \rangle, \quad \pi'_{B} := \langle \vec{c}, pk'_B \rangle,\]
   \[\pi_C := \langle \vec{c}, pk_C \rangle, \quad \pi'_{C} := \langle \vec{c}, pk'_C \rangle, \quad \pi_K := \langle \vec{c}, pk_K \rangle, \quad \pi'_{K} := \langle \vec{h}, pk'_K \rangle.\]

7. Output $\pi := (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi'_K)$.

(c) Verifier $V$

- **INPUTS:** verification key vk, input $\vec{x} \in \mathbb{F}_p^n$, and proof $\pi$
- **OUTPUTS:** decision bit

1. Compute $vk_{\vec{x}} := vk_{C,0} + \sum_{i=1}^n \pi_i vk_{C,i} \in \mathbb{G}_1$.
2. Check validity of knowledge commitments for $A, B, C$:
   \[e(\pi_A, vk_A) = e(\pi'_A, P_2), \quad e(vk_B, \pi_B) = e(\pi'_B, P_2),\]
   \[e(\pi_C, vk_C) = e(\pi'_C, P_2).\]
3. Check same coefficients were used:
   \[e(\pi_K, vk_\gamma) = e(vk_{\vec{x}} + \pi_A + \pi_C, vk_{2,\gamma}) \cdot e(vk_{2,\gamma}, \pi_B).\]
4. Check QAP divisibility:
   \[e(vk_{\vec{x}} + \pi_A, \pi_B) = e(\pi_H, vk_2) \cdot e(\pi_C, P_2).\]
5. Accept if and only if all the above checks succeeded.

Key sizes. When invoked on a circuit $C: \mathbb{F}_p^n \times \mathbb{F}_p^m \rightarrow \mathbb{F}_p^l$ with $a$ wires and $b$ (bilinear) gates, the key generator outputs:
- $pk$ with $(6a + b + n + l + 26) \mathbb{G}_1$-elements and $(a + 4) \mathbb{G}_2$-elements;
- $vk$ with $(n + 3) \mathbb{G}_1$-elements and 5 $\mathbb{G}_2$-elements.

Proof size. The proof always has 7 $\mathbb{G}_1$-elements and 1 $\mathbb{G}_2$-element.
[PGHR13] assumptions

- $q$-power Diffie-Hellman
- $q$-strong Diffie-Hellman
- $q$-power Knowledge of Exponent

$q = \text{poly}(\text{circuit size})$

**Assumption 2 (q-PKE [21])** The $q$-power knowledge of exponent assumption holds for $G$ if for all $\mathcal{A}$ there exists a non-uniform probabilistic polynomial time extractor $\chi_{\mathcal{A}}$ such that

$$\Pr\left[ (p, G, G_T, e) \leftarrow G(1^K); g \leftarrow G \setminus \{1\}; \alpha, s \leftarrow \mathbb{Z}_p^*; \right. \right.$$

$$\left. \sigma \leftarrow (p, G, G_T, e, g, g_s, \ldots, g_s^q, g_{\alpha}, g_{\alpha s}, \ldots, g_{\alpha s^q}); \right. \right.$$

$$(c, \hat{c}; a_0, \ldots, a_q) \leftarrow (\mathcal{A} \parallel \chi_{\mathcal{A}})(\sigma, z);$$

$$\hat{c} = c^\alpha \land c \neq \prod_{i=0}^q g_{a_i s^i} = \text{negl}(K)$$

for any auxiliary information $z \in \{0, 1\}^\text{poly}(K)$ that is generated independently of $\alpha$. Note that $(y; z) \leftarrow (\mathcal{A} \parallel \chi_{\mathcal{A}})(x)$ signifies that on input $x$, $\mathcal{A}$ outputs $y$, and that $\chi_{\mathcal{A}}$, given the same input $x$ and $\mathcal{A}$’s random tape, produces $z$. 
SNARKs for C: a peek under the hood

**Setup**
- preprocessing SNARKs: $T \cdot \text{polylog } T$
- Public proving key is a “template” of a correct computation.
- Scalable / PCP-based SNARK: $\text{poly}(S)$

**Legend**
- $T$ – running time
- $S$ – program size

**Diagram**
1. C program
2. Compiler (new gcc backend)
3. TinyRAM assembly code (new machine spec.)
4. ACSP Generator
5. Algebraic Constraint Satisfaction Problem
6. TinyRAM interpreter
7. Prover
   - Input
   - Auxiliary input
   - Cost $\approx T \cdot \text{polylog } T$
8. Verifier
   - Output
   - Proof
   - Cost $\approx \text{poly}(S) + \text{poly log } T$
zkSNARK backend implementations

- **Pinocchio/Geppetto**
  [https://vc.codeplex.com](https://vc.codeplex.com)  
  [PGHR13] [CFHKKNPZ15]

- **libsnark**
  [github.com/scipr-lab/libsnark](https://github.com/scipr-lab/libsnark)  
  [BCGTV13a] [BCTV14crypto] [BCTV14usenix] …

- **snarklib**
  [github.com/jancarlsson/snarklib](https://github.com/jancarlsson/snarklib)  
  (clone of libsnark with different C++ style by “Jan Carlsson”)

Numerous frontends (some included in the above), to be discussed tomorrow.
Example: libsnark backends

- [PGHR13] backend with [BCTV14USENIX] improvements
  - speed of verifier by merging parts of the pairing computation
  - reduced verification key size to ~1/3 (when #inputs ≪ #gates)
- Square Span Programs [DFGK14] backend
- ADSNARK backend, [BBFR15] backend
- Tailored libraries for finite fields, ECC, pairings

<table>
<thead>
<tr>
<th>1M arithmetic gates, 1000-bit input, desktop PC</th>
<th>80-bit security</th>
<th>128-bit security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generator</td>
<td>97 s</td>
<td>117 s</td>
</tr>
<tr>
<td>Prover</td>
<td>115 s</td>
<td>147 s</td>
</tr>
<tr>
<td>Verifier</td>
<td>4.9 ms</td>
<td>5.1 ms</td>
</tr>
<tr>
<td></td>
<td>(4.7 + 0.0004</td>
<td>x</td>
</tr>
<tr>
<td>Proof size</td>
<td>230 B</td>
<td>288 B</td>
</tr>
</tbody>
</table>

- Full code, MIT license  
  
github.com/scipr-lab/libsnark
### SNARKs for C general programs

<table>
<thead>
<tr>
<th>Feasibility</th>
<th>Network</th>
<th>C program size</th>
<th>Program running time</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory (poly)</td>
<td>Fast verify</td>
<td>Fast prove</td>
<td>1 hop</td>
<td>Any</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>[Kilian 92] [Micali 94] [Groth 2010]</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>[Chiesa Tromer 2010] [Valiant 08]</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>[Ben-Sasson Chiesa Genkin Tromer Virza 2013]</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>[Parno Gentry Howell Raykova 2013]</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>[Ben-Sasson Chiesa Tromer Virza 2014 USENIX Security]</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>[Ben-Sasson Chiesa Tromer Virza 2014 CRYPTO]</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Tighter frontends from high level (Geppetto, Buffet…) at cost in universality, supporting random accesses and general control flow, and scalability.
Proof-Carrying Data

• Diverse network, containing untrustworthy parties and unreliable components.
• Impractical to verify internals of each node, so give up.
• Enforce only correctness of the messages and ultimate results.
Proof-Carrying Data (cont.)

- Every message is augmented with a proof attesting to its *compliance* with a prescribed policy.
- Compliance can express any property that can be verified by locally checking every node.
- Proofs can be verified efficiently and retroactively.
System designer specifies his notion of correctness via a compliance predicate $C(\text{incoming, local inputs, outgoing})$ that must be locally fulfilled at every node.
Examples of C-compliance

**Correctness** is a **compliance predicate** $C$(in,code,out) that must be locally fulfilled at every node

- $C_a = “the output is the result of correctly computing a prescribed program”$
- $C_b = “the output is the result of correctly executing some program signed by the sysadmin”$
- $C_c = “the output is a well-traced object of a given class (in an object-oriented language), and thus respects the class invariants”$  
  [Chong Tromer Vaughan 13]
SNARKs and Proof-Carrying Data: prospective applications

- Bitcoin
  (Zerocash, compression)
- Platform integrity
  (supply chain, BYOD, cloud)
- Information provenance
- Safe deserialization in distributed programs
  [Chong Tromer Vaughan 2013]
- Software whitelists
- MMO virtual worlds
- “Compliance engineering”
Conclusion
<table>
<thead>
<tr>
<th>Primitive</th>
<th>Attacks</th>
<th>Guarantees</th>
<th>Functional</th>
<th>Output form</th>
<th>Communication</th>
<th>Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FHE</td>
<td>ANY</td>
<td>none</td>
<td>yes</td>
<td>YES</td>
<td>Circuits</td>
<td>Encrypted</td>
</tr>
<tr>
<td>Obfuscation (VBB)</td>
<td>ANY</td>
<td>ANY</td>
<td>YES</td>
<td>YES</td>
<td>Plaintext</td>
<td>Minimal</td>
</tr>
<tr>
<td>Leakage resilience</td>
<td>Varies</td>
<td>none</td>
<td>yes</td>
<td>YES</td>
<td>Varies</td>
<td>Plaintext</td>
</tr>
<tr>
<td>Tamper resilience</td>
<td>Varies</td>
<td>Varies</td>
<td>Varies</td>
<td>Varies</td>
<td>Plaintext</td>
<td>Minimal</td>
</tr>
<tr>
<td>TPM, SGX</td>
<td>Some</td>
<td>Some</td>
<td>Yes</td>
<td>ANY</td>
<td>Plaintext</td>
<td>Minimal</td>
</tr>
<tr>
<td>Computational proofs (SNARK/PCD)</td>
<td>ANY</td>
<td>ANY</td>
<td>YES</td>
<td>no</td>
<td>RAM, distributed</td>
<td>Plaintext + proof</td>
</tr>
<tr>
<td>Multiparty computation</td>
<td>ANY</td>
<td>ANY</td>
<td>YES</td>
<td>YES</td>
<td>ANY</td>
<td>Plaintext</td>
</tr>
<tr>
<td>Garbled circuits</td>
<td>ANY</td>
<td>none</td>
<td>yes</td>
<td>YES</td>
<td>Circuits</td>
<td>Plaintext</td>
</tr>
</tbody>
</table>

- **Leakage**: ANY
- **Tampering**: none, no
- **Correctness**: yes
- **Secrecy**: YES
- **Function class**: Circuits
- **Output form**: Encrypted
- **Communication**: Minimal
- **Assumptions**: Computational