TEL AVIV UNIVERSITY

Information Security – Theory vs. Reality

0368-4474-01, Winter 2015-2016

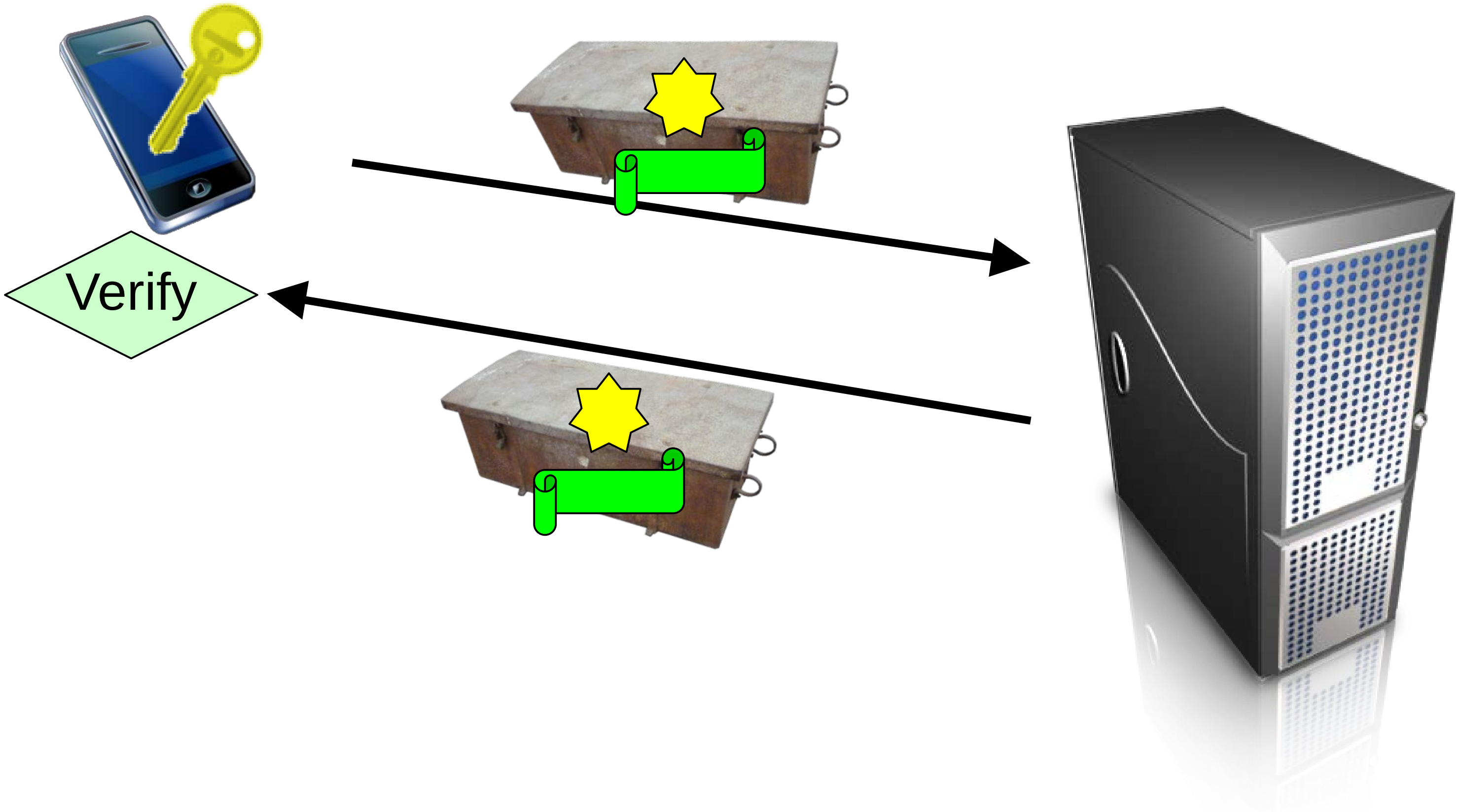
Lecture 12:
Verified computation and its applications, course conclusion

Eran Tromer

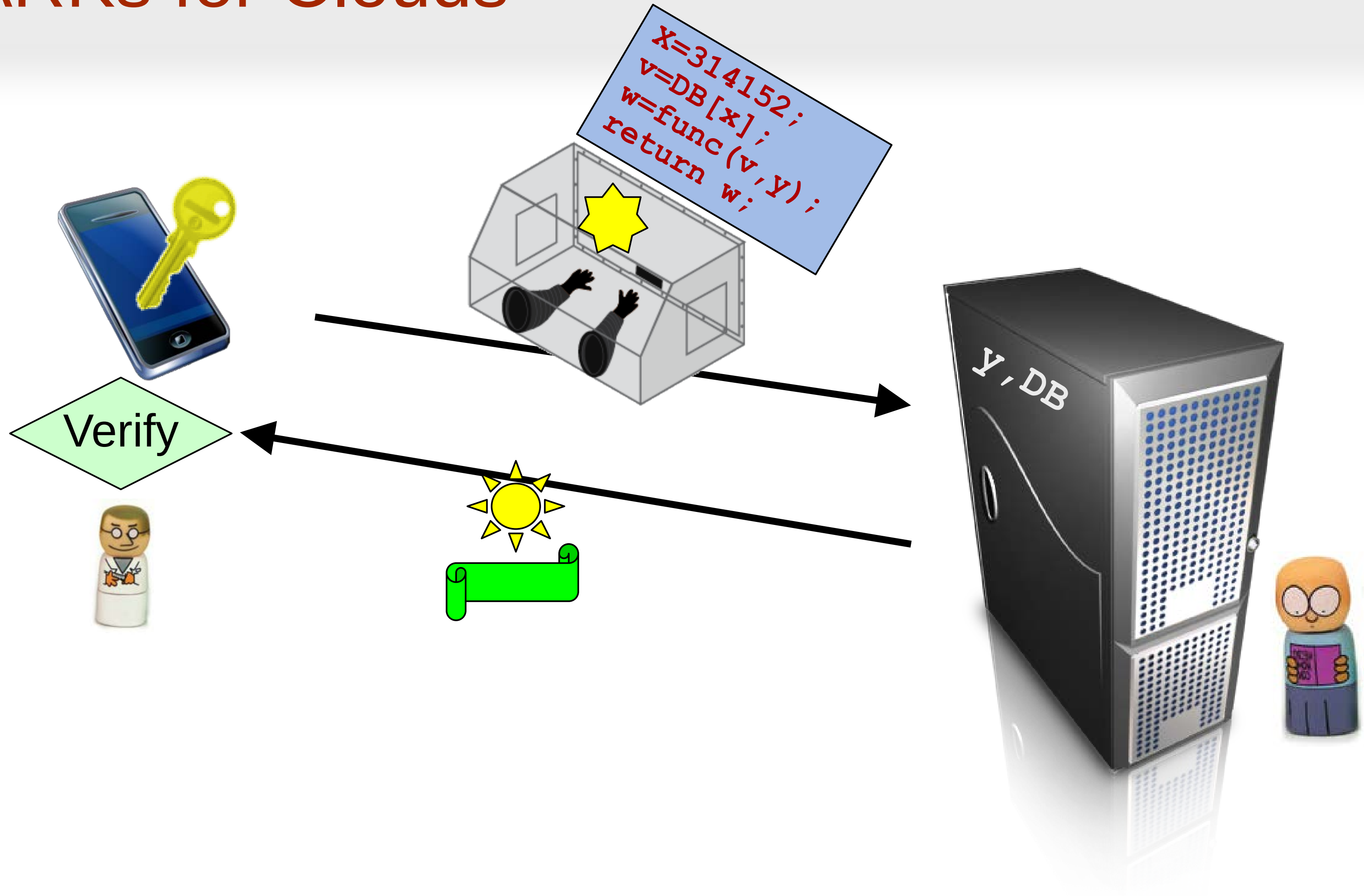
Verified computation using computational proofs

Motivation 1:
cloud computing

Integrity of data:
digital signatures / message authentication codes



SNARKs for Clouds



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

SNARK motivation 3: Privacy for Bitcoin

Zerocash

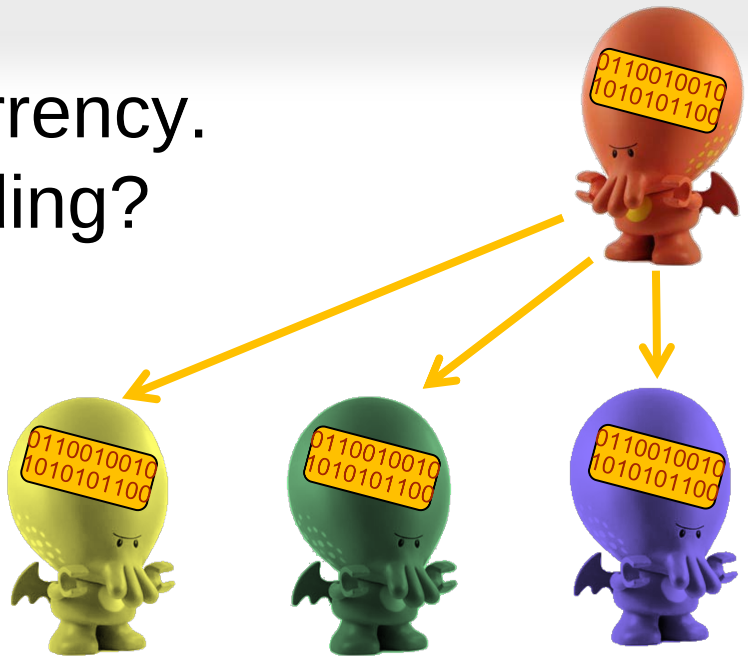
`zerocash-project.org`

[Ben-Sasson Chiesa Garman Gree Miers Tromer Virza 2014]

Bitcoin's privacy problem

Bitcoin: decentralized <u>digital</u> currency.

What's to prevent double-spending?

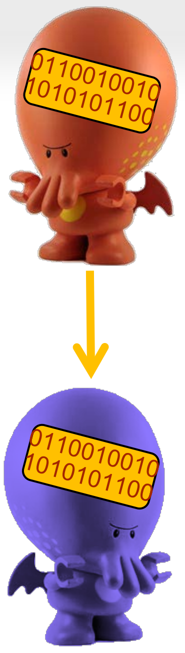


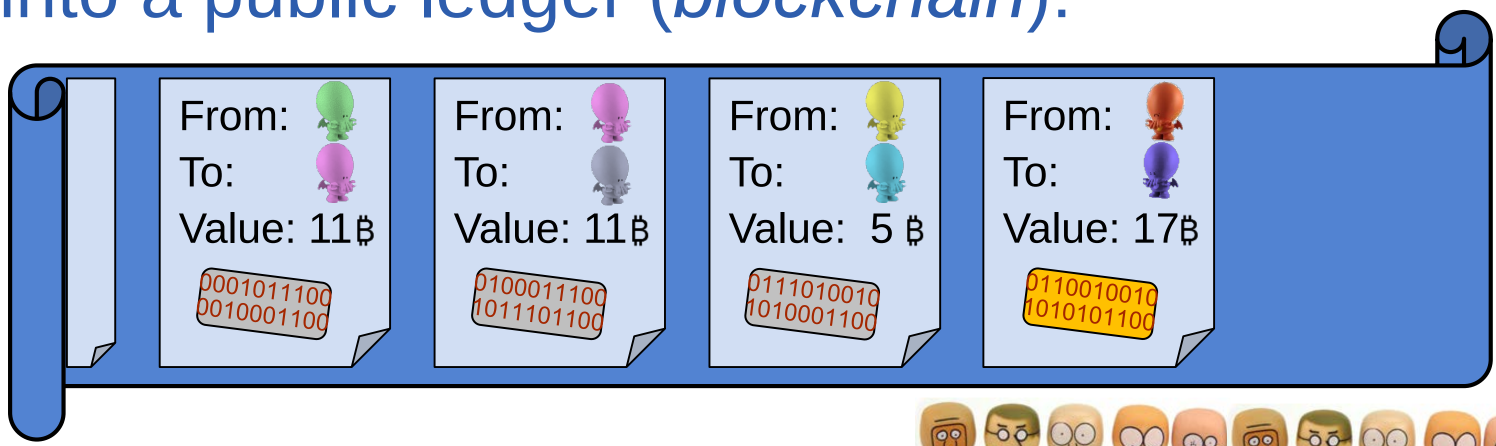
Bitcoin’s privacy problem

Bitcoin: decentralized <u>digital</u> currency.

What’s to prevent double-spending?

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

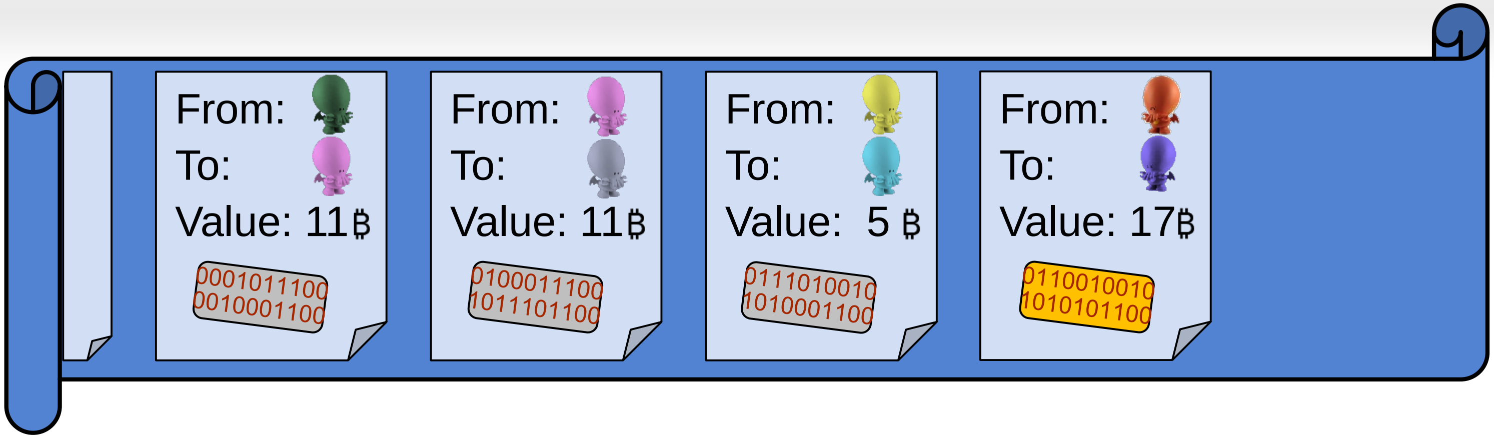




The cost: privacy.

- 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.)



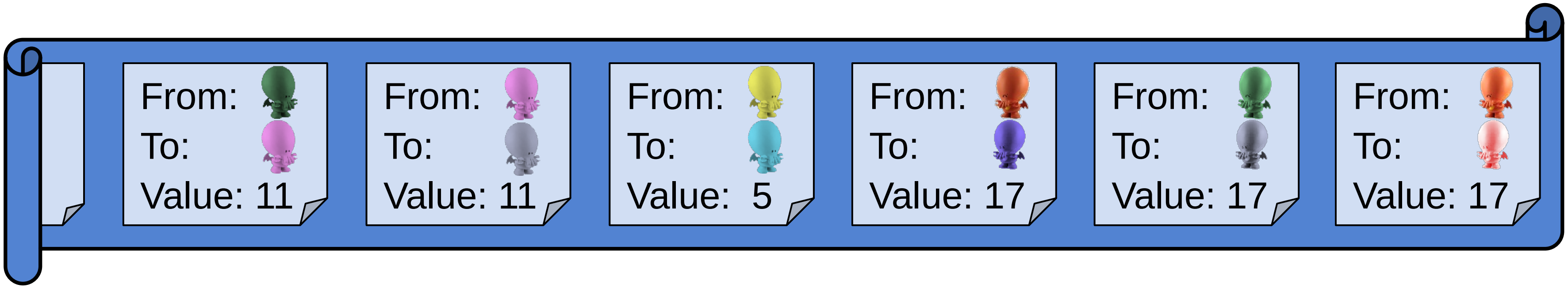
- Pseudonymous, but:
  - Most users use a single or few addresses
  - Transaction graph can be analyzed.

[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).



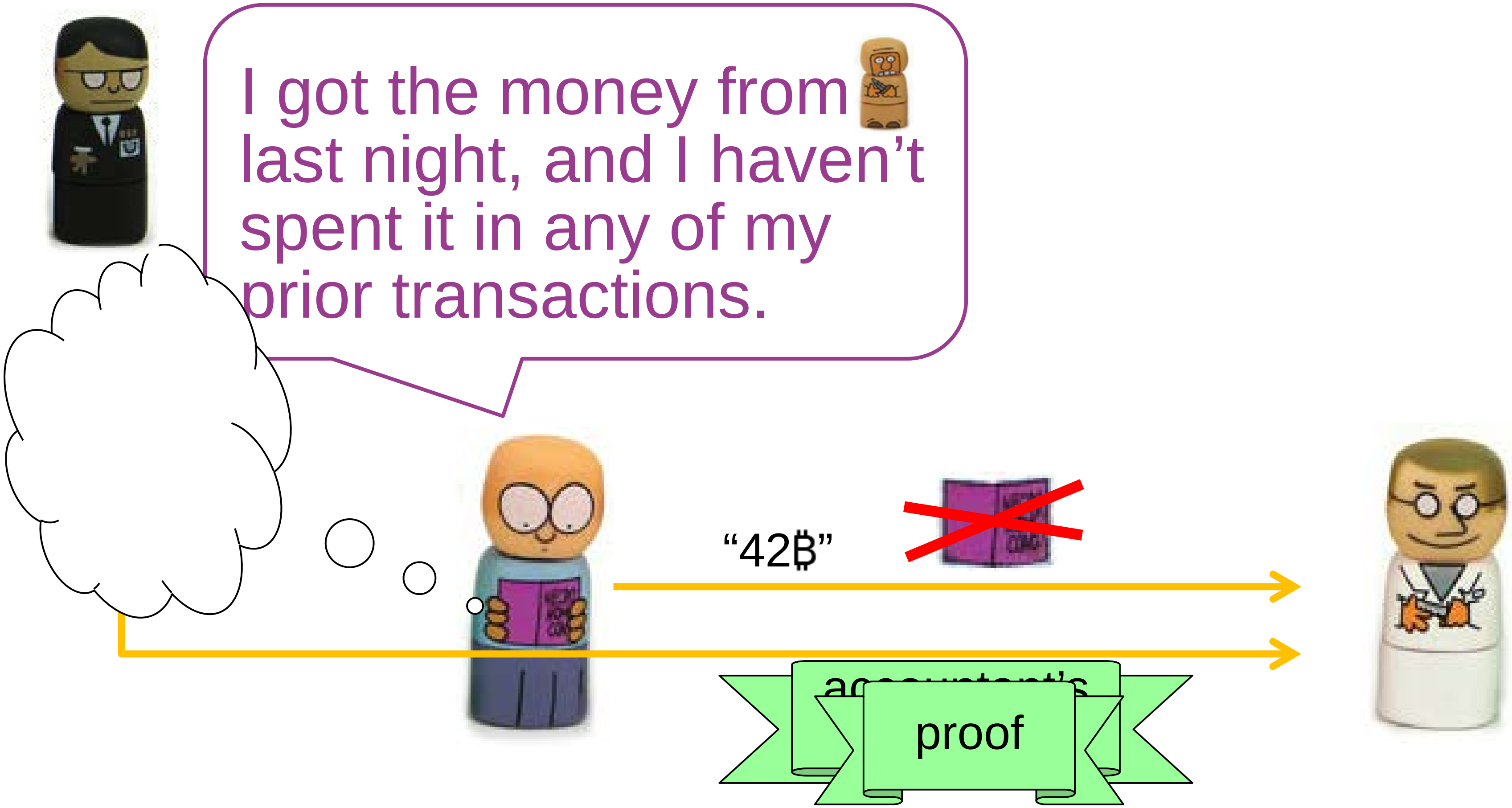
- Zerocash enables users to pay one another <u>directly</u> via payment transactions of <u>variable denomination</u> that <u>reveal neither the origin, destination, or amount.</u>

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(#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

Zerocash: in *proofs* we trust



Intuition: “virtual accountant” using cryptographic proofs.

Requisite proof properties

zero **k**nowledge

succinct (blockchain size, verification by everyone)

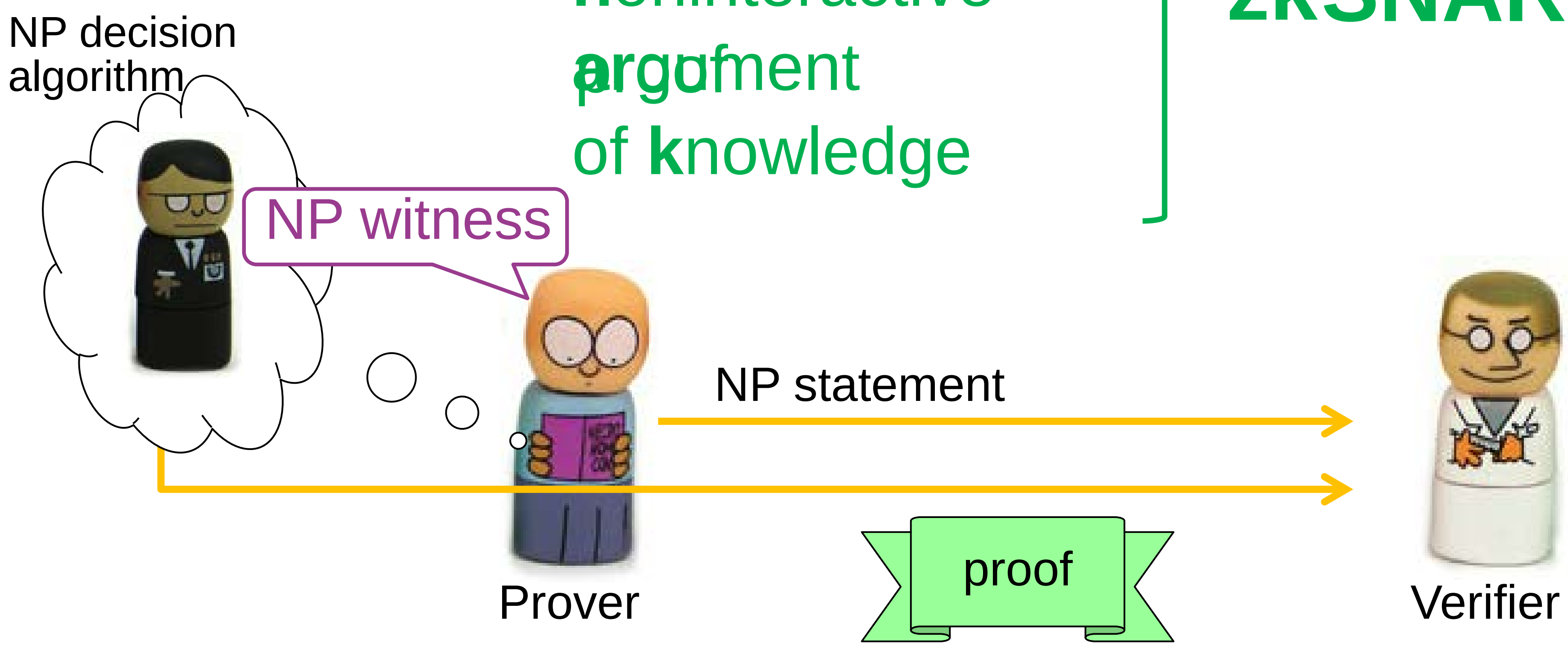
noninteractive (even 2 rounds are too much)

argument

proof

of **k**nowledge

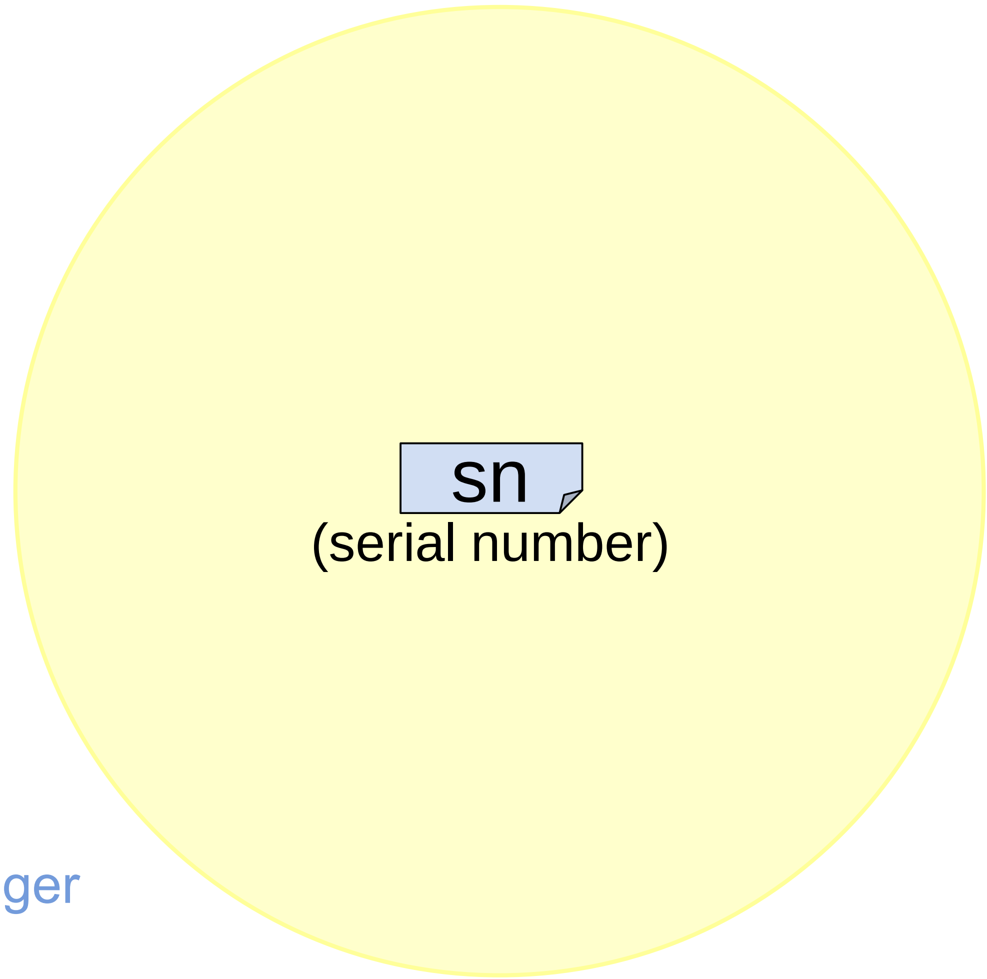
zkSNARK

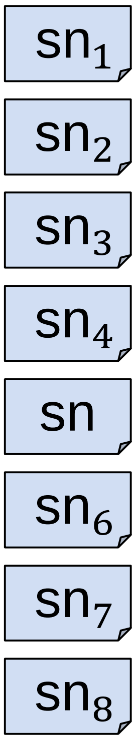


Basic anonymous e-cash (#1)

Minting: I hereby spend 1 BTC to create sn

Spending: ~~I'm using up a coin with (unique) sn~~





Legend:

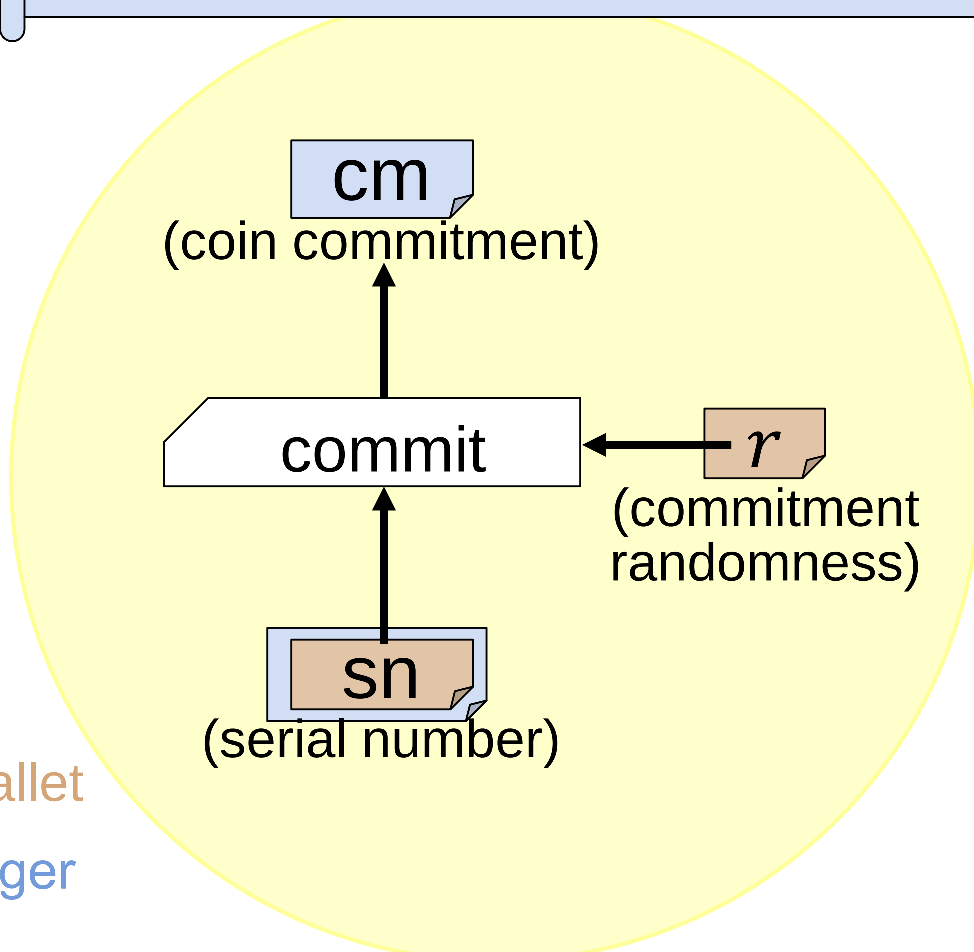
In public ledger

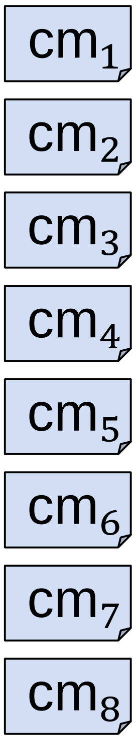
Basic anonymous e-cash (#2)

[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:

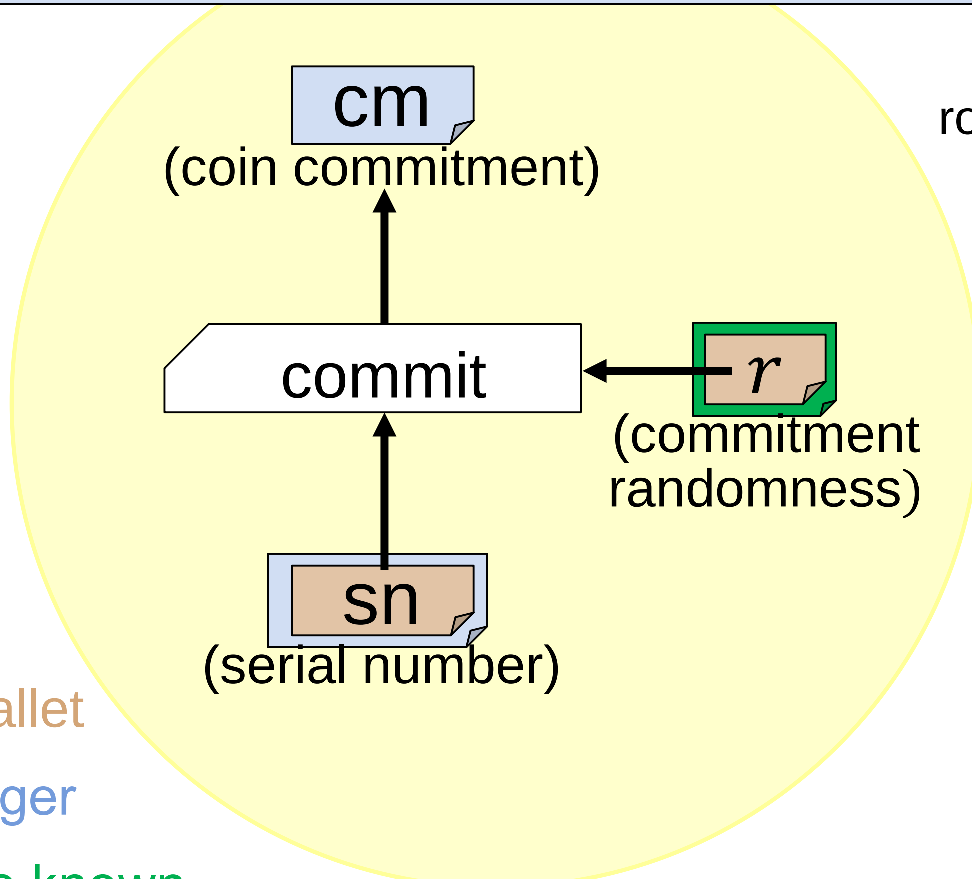
In private wallet

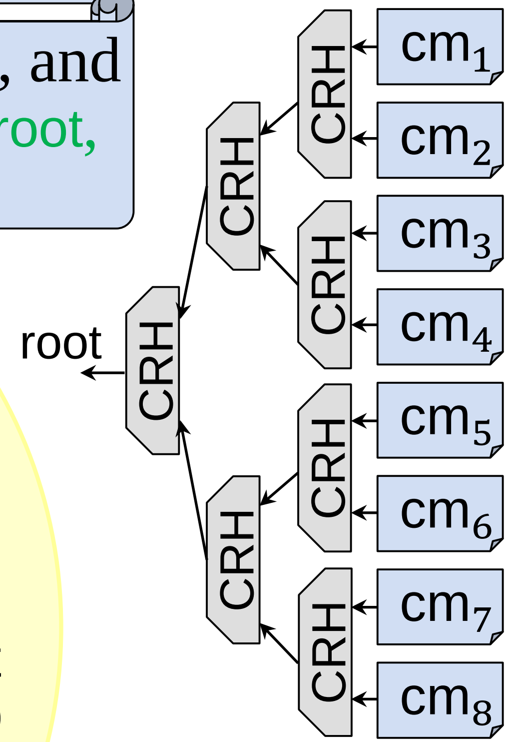
In public ledger

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

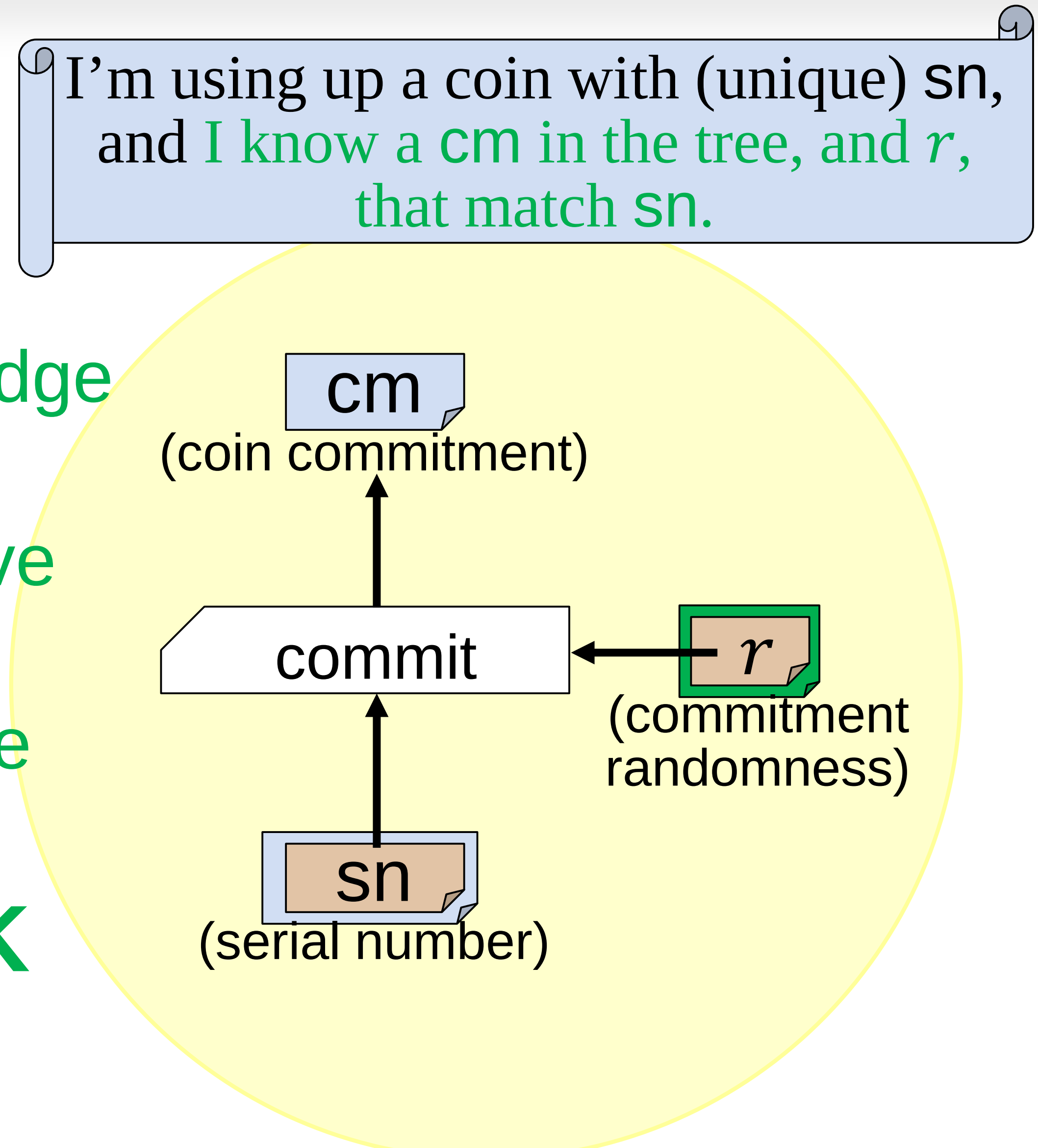
Basic anonymous e-cash – requisite proofs

Spending:

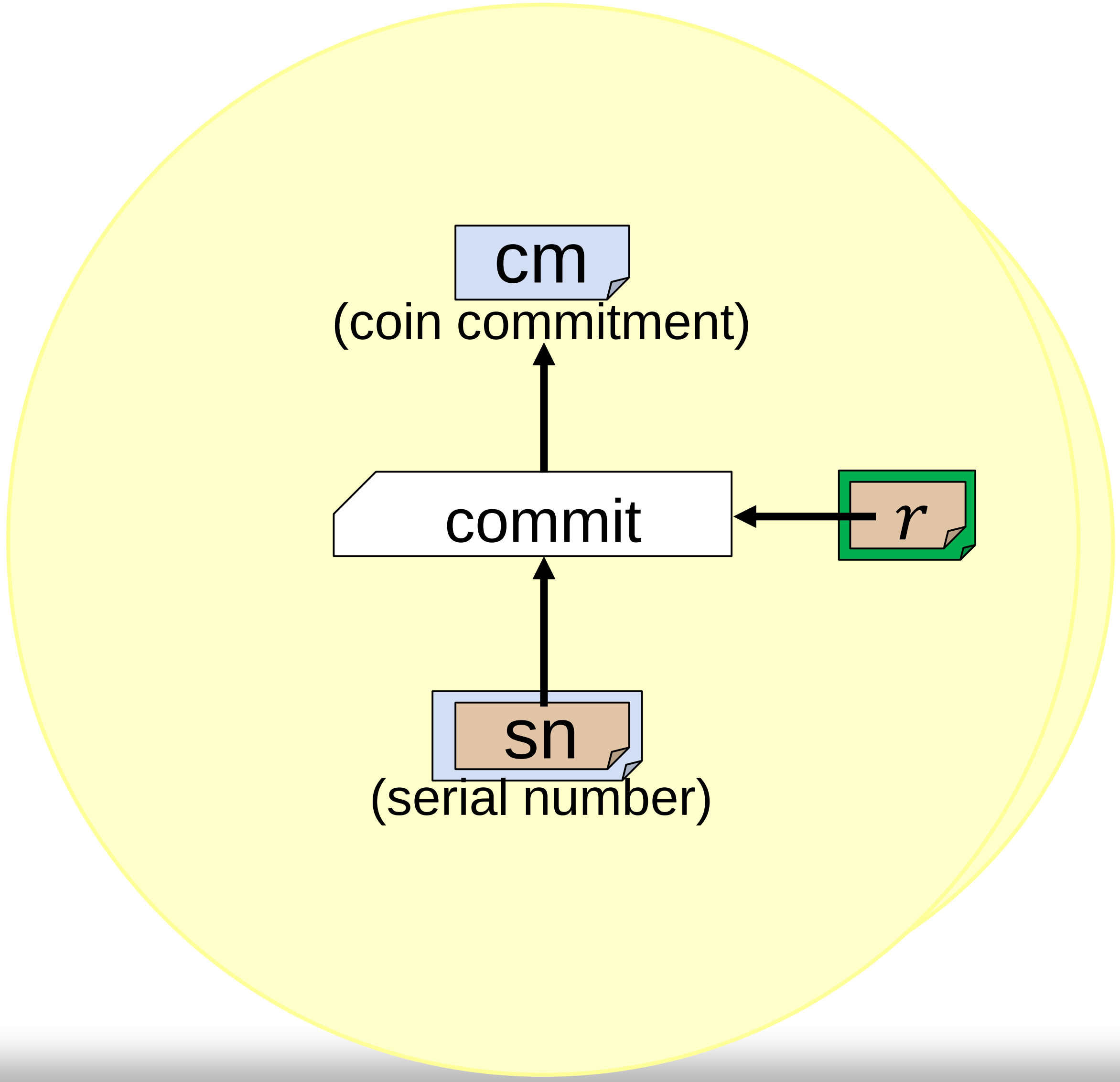
Requires:

zero **k**nowledge
succinct
noninteractive
argument
proof
of **k**nowledge

zkSNARK



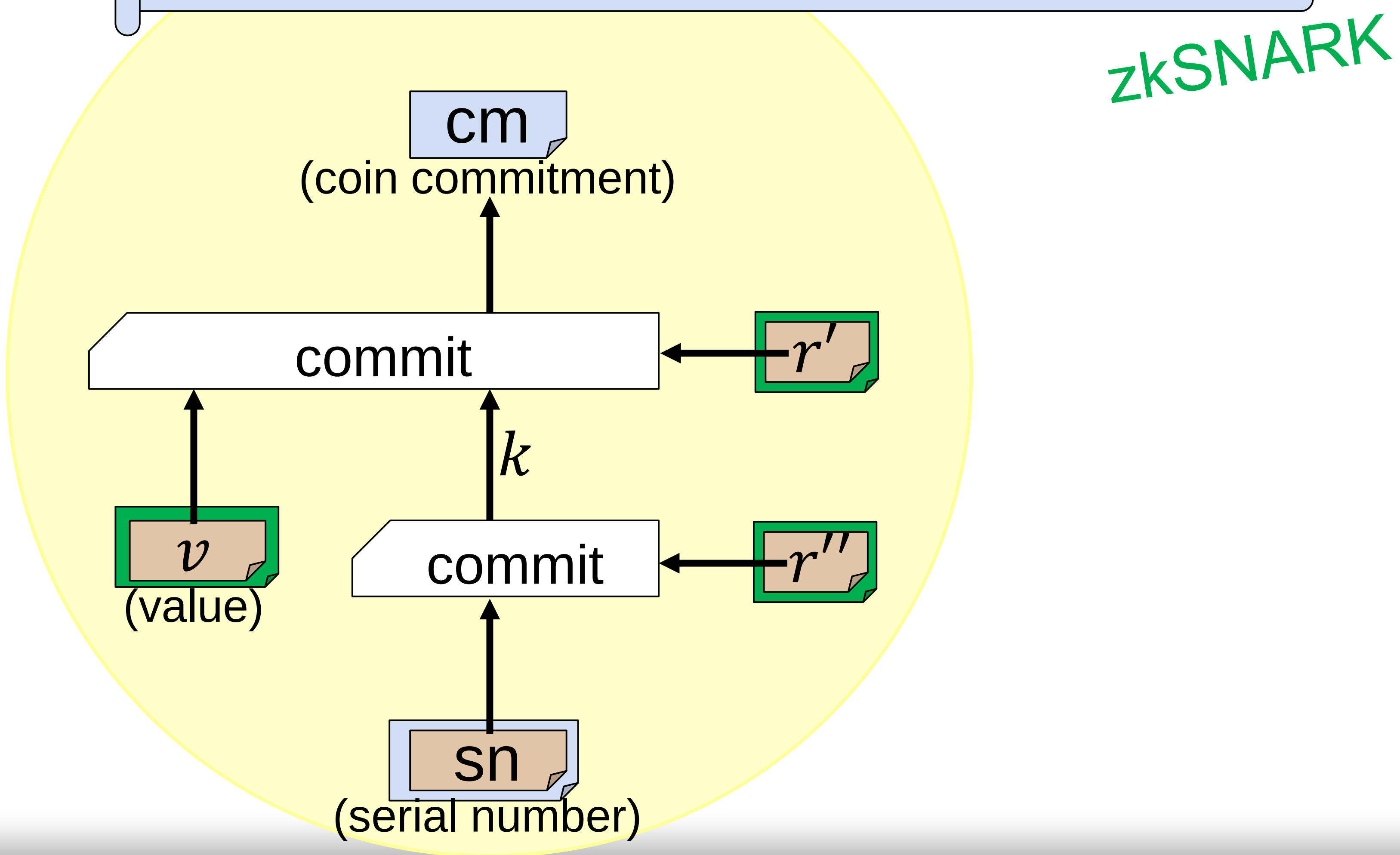
zkSNARK
with great power comes great functionality



Adding variable denomination (#4)

Minting: I hereby spend v BTC to create cm, and here is k, r' to prove consistency.

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

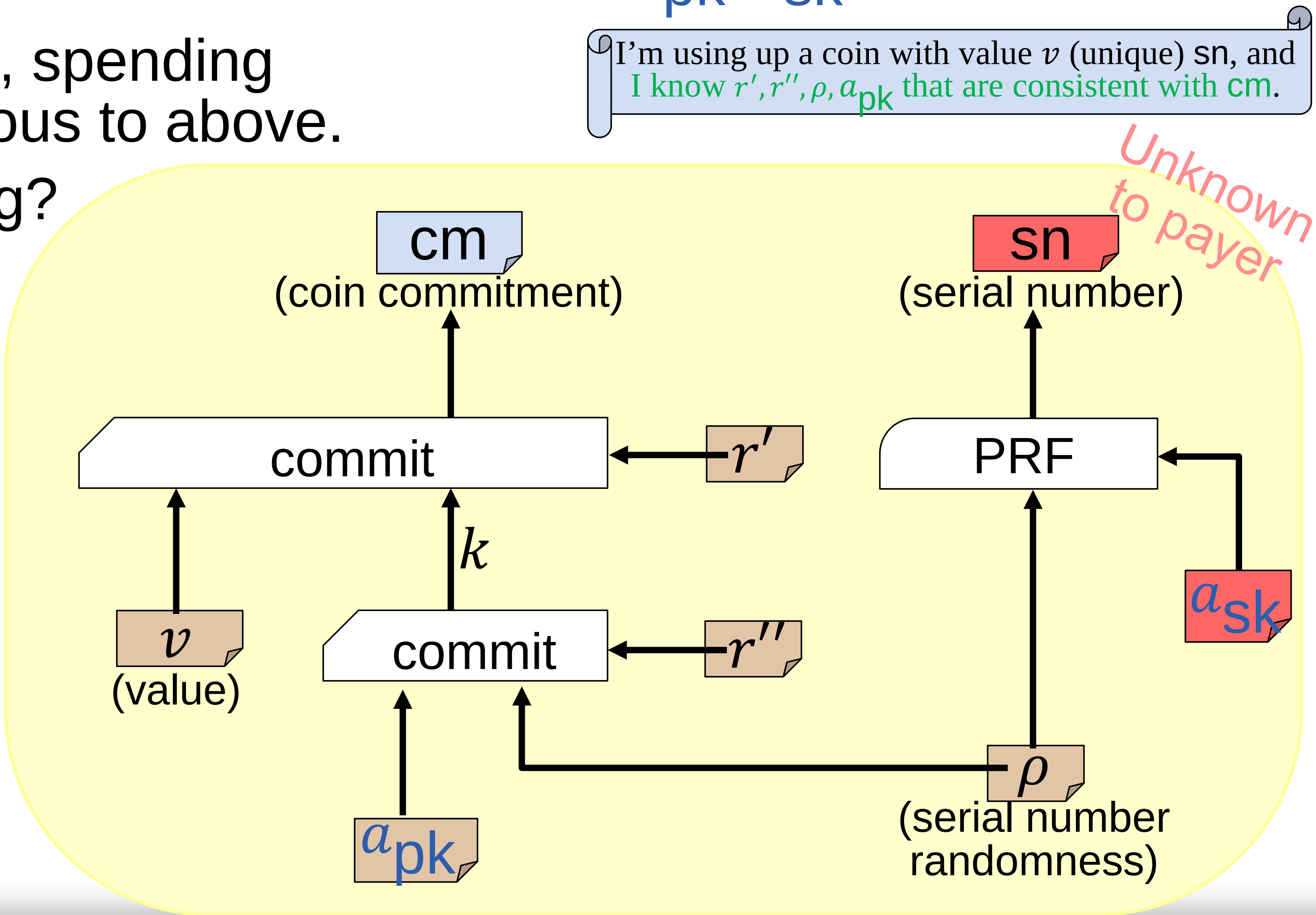


Adding direct anonymous payments (#5)

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

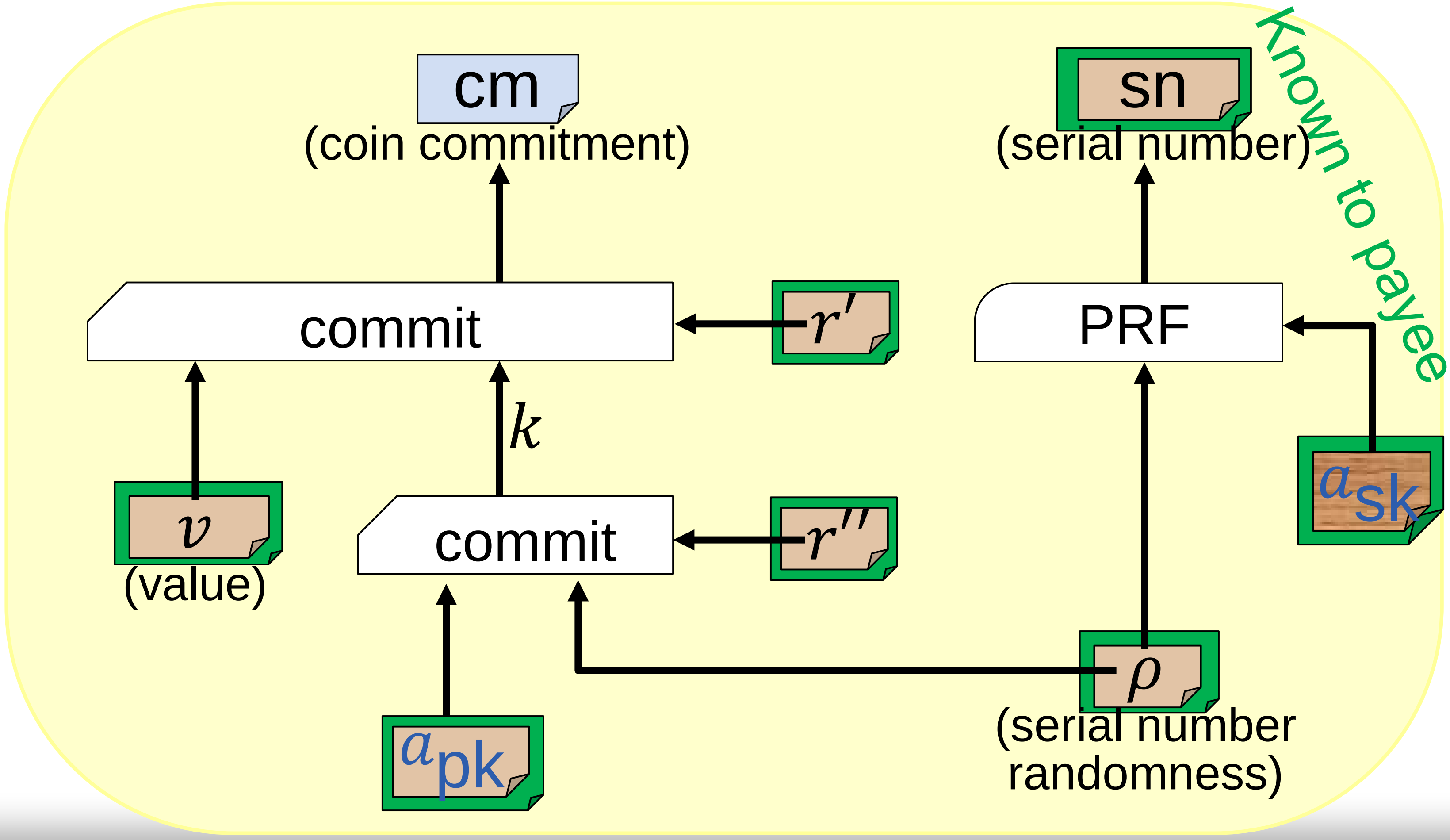
Minting, spending analogous to above.

Sending?



Sending direct anonymous payments

1. Create coin using a_{pk} of payee.
2. Send coin secrets (v, ρ, r', r'') to payee
   out of band, or encrypted to payee's public key.

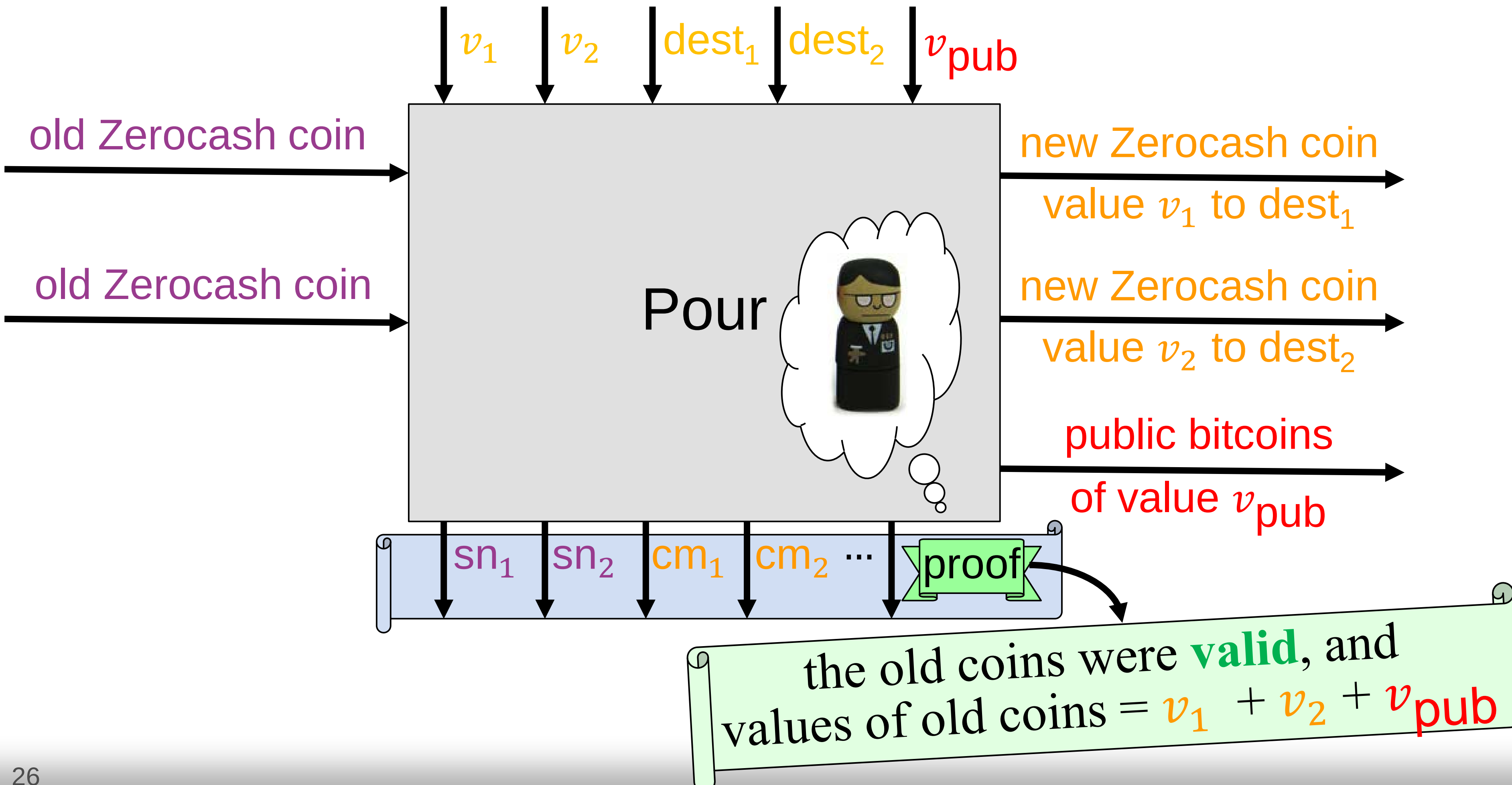


Pouring Zerocash coins (#6)

Simplified

Single transaction type capturing:

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

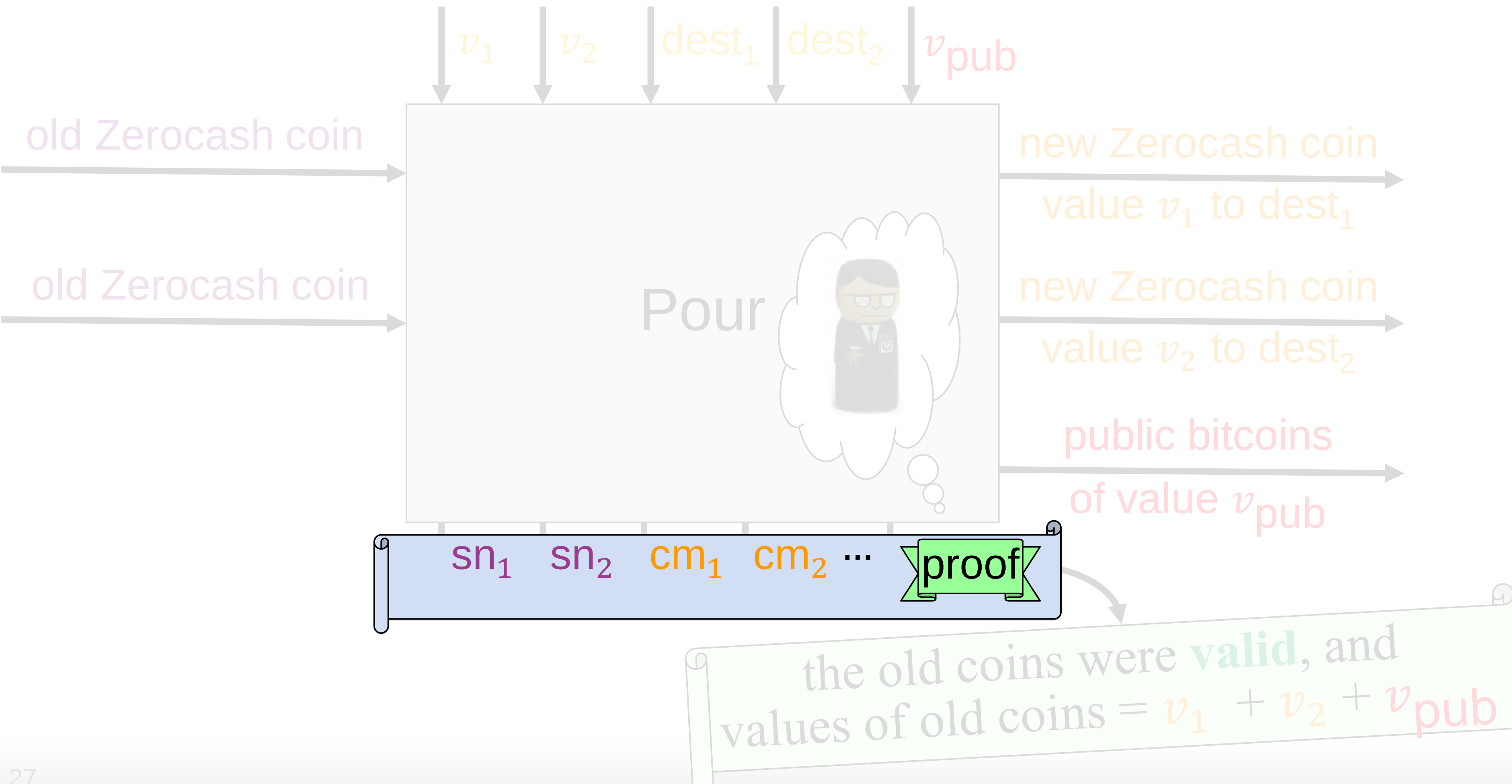


Pouring Zerocash coins

Simplified

Single transaction type capturing:

Sending payments
Making change
Exchanging into bitcoins
Transaction fees



Example of a Zerocash Pour transaction

root	1c4ac4a110e863deeca050dc5e5153f2b7010af9
sn_1	a365e7006565f14342df9096b46cc7f1d2b9949367180fdd8de4090eee30bfdc
sn_2	6937031dce13facdebe79e8e2712ffad2e980c911e4cec8ca9b25fc88df73b52
cm_1	a4d015440f9cfae0c3ca3a38cf04058262d74b60cb14ecd6063e047694580103
cm_2	2ca1f833b63ac827ba6ae69b53edc855e66e2c2d0a24f8ed5b04fa50d42dc772
v_pub	0000000000000042
pubkeyHash info	8f9a43f0fe28bef052ec209724bb0e502ffb5427
SigPK	2dd489d97daa8ceb006cb6049e1699b16a6d108d43
Sig	f1d2d2f924e986ac86fdf7b36c94bcdf32beec15a38359c82f32dbb3342cb4bedcb78ce116bac69e
MAC_1	b8a5917eca1587a970bc9e3ec5e395240ceb1ef700276ec0fa92d1835cb7f629
MAC_2	ade6218b3a17d609936ec6894b7b2bb446f12698d4bcafa85fcbf39fb546603a
ciphertext_1	048070fe125bdaf93ae6a7c08b65adbb2a438468d7243c74e80abc5b74dfe3524a987a2e3ed075d54ae7a53866973eaa5070c4e0895 4ff5d80caae214ce572f42dc6676f0e59d5b1ed68ad33b0c73cf9eac671d8f0126d86b667b319d255d7002d0a02d82efc47fd8fd648 057fa823a25dd3f52e86ed65ce229db56816e646967baf4d2303af7fe09d24b8e30277336cb7d8c81d3c786f1547fe0d00c029b63bd 9272aad87b3f1a2b667fa575e
ciphertext_2	0493110814319b0b5cabb9a9225062354987c8b8f604d96985ca52c71a77055b4979a50099cefc5a359bdf0411983388fa5de840a0d 64816f1d9f38641d217986af98176f420caf19a2dc18c79abcf14b9d78624e80ac272063e6b6f78bc42c6ee01edfbcddbeb60eba586 eaecd6cb017069c8be2ebe8ae8a2fa5e0f6780a4e2466d72bc3243e873820b2d2e4b954e9216b566c140de79351abf47254d122a35f 17f840156bd7b1feb942729dc
zkSNARKproof	a4c3cad6e02eec51dc8a37ebc51885cf86c5da04bb1c1c0bf3ed97b778277fb8adceb240c40a0cc3f2854ce3df1eafdcefccc532bc5afaefefe9d3975726f2ca829228 6ca8dd4f8da21b3f98c61fac2a13f0b82544855b1c4ce7a0c9e57592ee1d233d43a2e76b9bdeb5a365947896f117002b095f7058bdf611e20b6c2087618c58208e3 658cfcc00846413f8f355139d0180ac11182095cdee6d9432287699e76ed7832a5fc5dc30874ff0982d9658b8e7c51523e0fa1a5b649e3df2c9ff58dc05dac7563741 298025f806dfbe9cfe5c8c40d1bf4e87dacb11467b9e6154fb9623d3fba9e7c8ad17f08b17992715dfd431c9451e0b59d7dc506dad84aef98475d4be530eb501925 dfd22981a2970a3799523b99a98e50d00eaab5306c10be5

~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

`libsnark`
zkSNARK

SCIPR LAB

Instantiate Zerocash primitives and parameters

`bitcoind`

Performance (quadcore desktop)

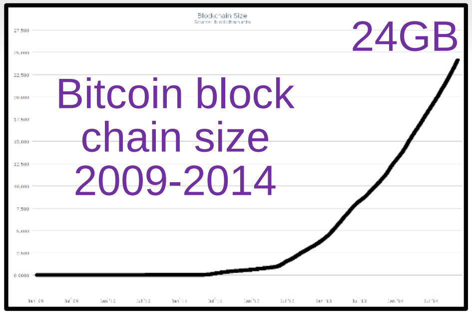
`Setup`	<2 min, 896MB params
`Mint`	23 μs 72B transaction
`Pour`	46 s, 1KB transaction
`Verify Transaction`	<9 ms/transaction
`Receive`	<2 ms/transaction

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:

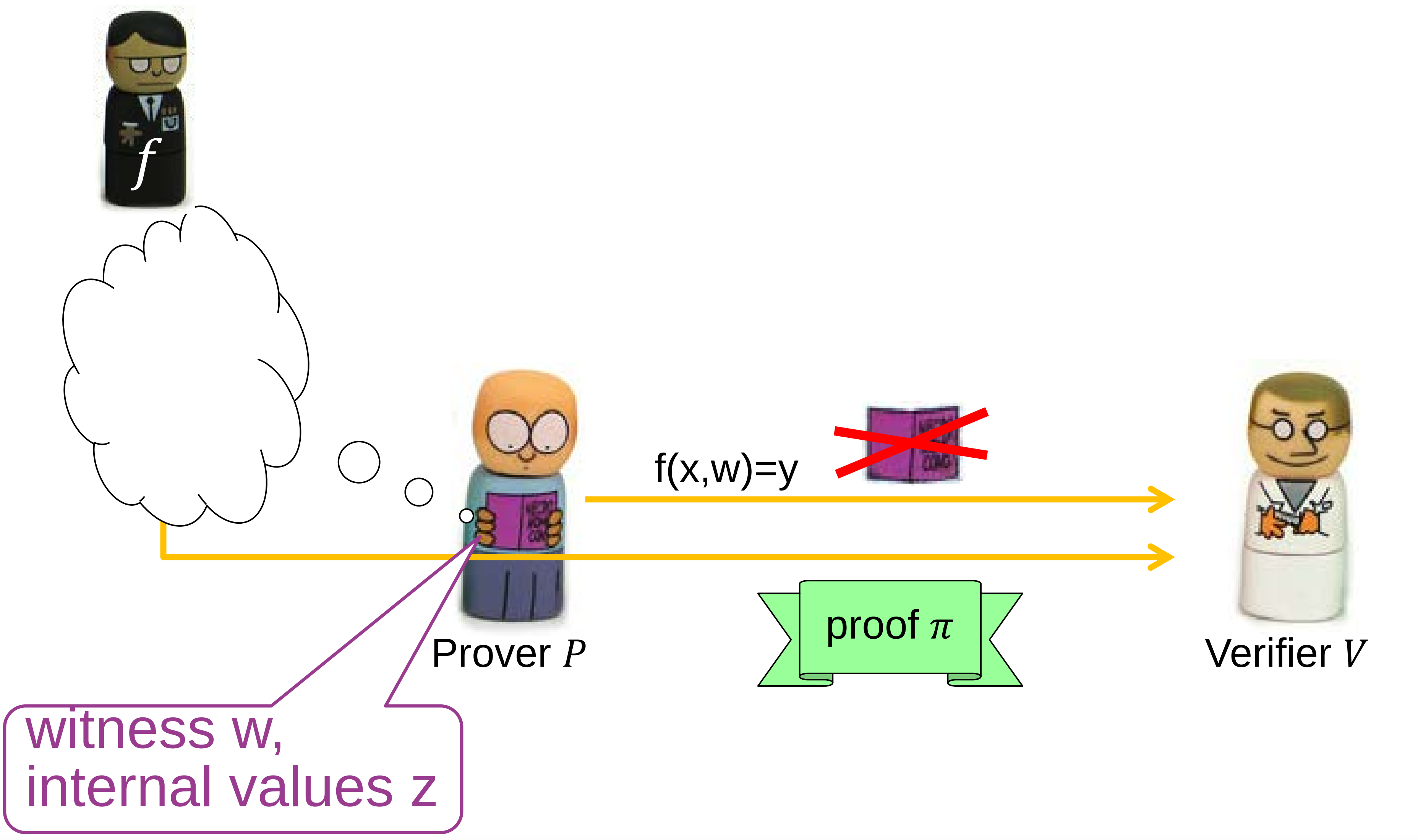
    "This transaction is valid with respect to block chain head *H*."
  - Blockchain compression

    "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

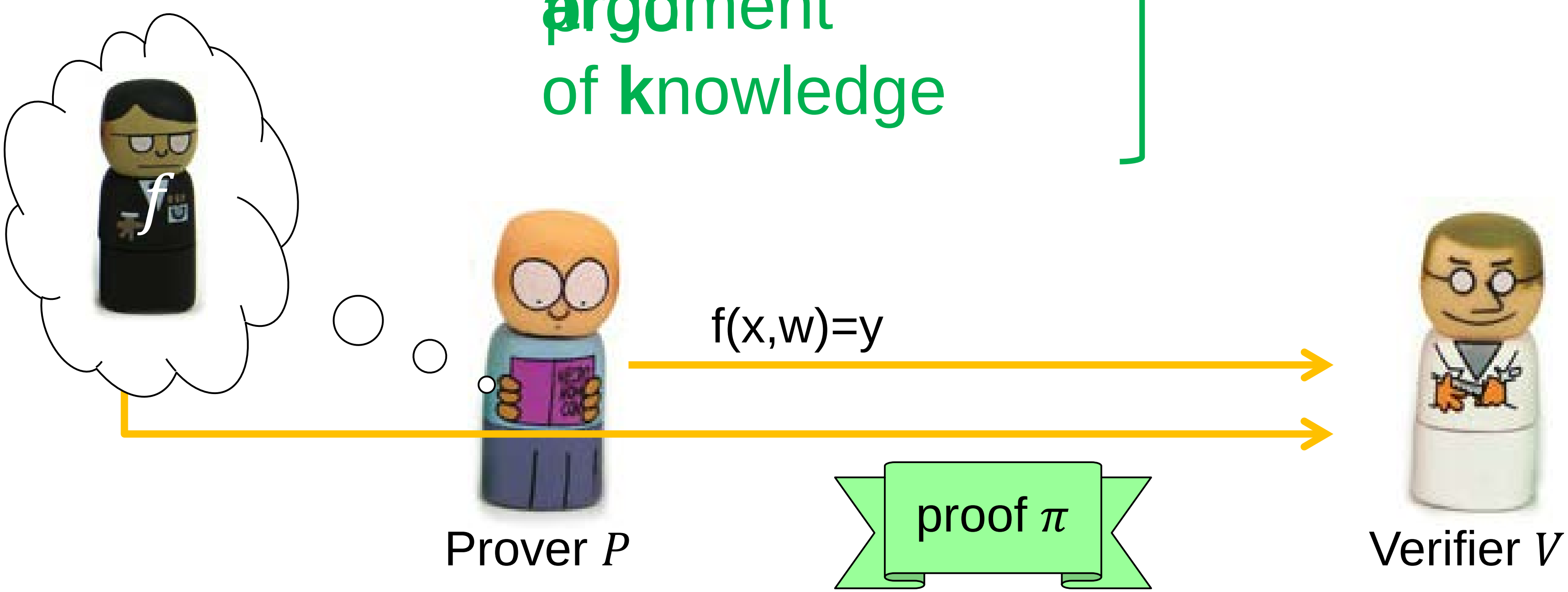
zkSNARK for NP: setting



zkSNARK for NP: setting

zero **k**nowledge
succinct **(*V*** and π)
noninteractive
argument / **p**roof
of **k**nowledge

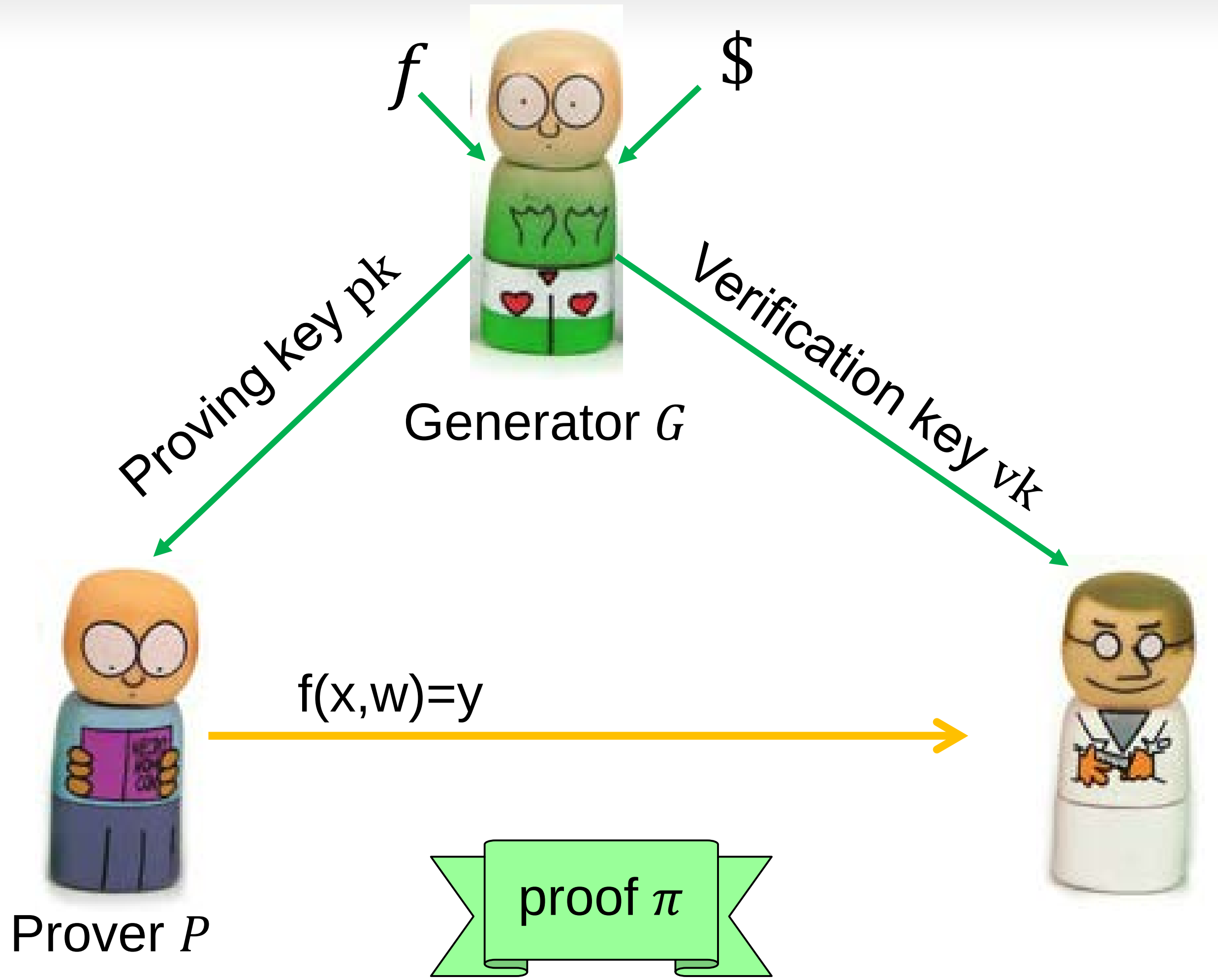
zkSNARK



Preprocessing zkSNARK for NP: setting

Variants:

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



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
    [Parno Gentry Howell Raykova 13]
    [Ben-Sasson Chiesa Genkin Tromer Virza 13]
    [Braun Feldman Ren Setty Blumberg Walfish 2013]
    [Ben-Sasson Chiesa Tromer Virza 14 @ CRYPTO]
    [Ben-Sasson Chiesa Tromer Virza 14 @ USENIX Security]
    [BFRSVW13] [BCGGMTV14] [FL14]

    "SNARKs for C"
    Execution of C programs can be proved in 288 bytes and verified in 6 ms.
  - 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

SCIPR Lab
scipr-lab.org

Which SNARK?

"Long" keys

[BCGTV13] [FLZ13]
[Lipmaa14]
[KPPSST14]
[ZPK14]* [Lipmaa13]
[WSRBW15]
[BBFR15] [DFGK14] [GGPR13]
[Groth10] [PGHR13]
[Lipmaa12] [BCIOP13] [BCTV14$_{\text{USENIX}}$]

[CTV15]
[CFHKKNPZ14]

"Short" keys

[Kilian92] [GLR11]
[Micali94]
[BC12]
[Valiant08]
[BCCGLRT14] [DL08]
[DFH12] [BCCT13]
[BCTV14$_{\text{CRYPTO}}$] [BCCT12]

Used by Zerocash (libsnark implementation)

Preprocessing SNARKs for NP

	Proof size (field elements)	CRS size	Prover runtime
[Groth10]	42	$O(s^2)$	$O(s^2)$
[Lipmaa12]	39	$\tilde{O}(s)$	$O(s^2)$
QAP-based [GGPR13] Reinterpreted as linear PCPs: [BCIOP13] [SBVBBW13] Improvements: [PGHR13] [BCTV14usenix] [BBFR15] [CFHKKNPX15] ...	7—8	$O(s)$	$\tilde{O}(s)$
[DFGK14]	4	$O(s)$	$\tilde{O}(s)$

Preprocessing is private-coin and costs $\tilde{O}(s)$

zkSNARK construction via QAP and Linear PCPs

- Computation
- Algebraic Circuit
- R1CS
- QAP
- Linear PCP
- Linear Interactive Proof
- zkSNARK

Computation ⇒ Arithmetic Circuit

Efficient computation $f(\cdot)$.

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

completeness ⇓

soundness, PoK ⇑

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 $\mathbb{F}^k$.

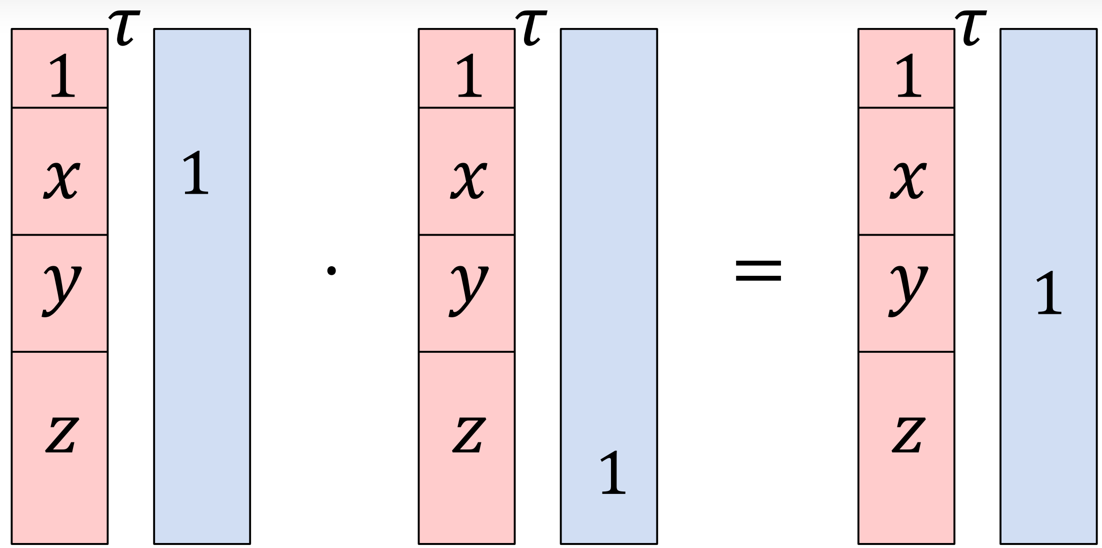
$\exists z \in \mathbb{F}^n$:

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

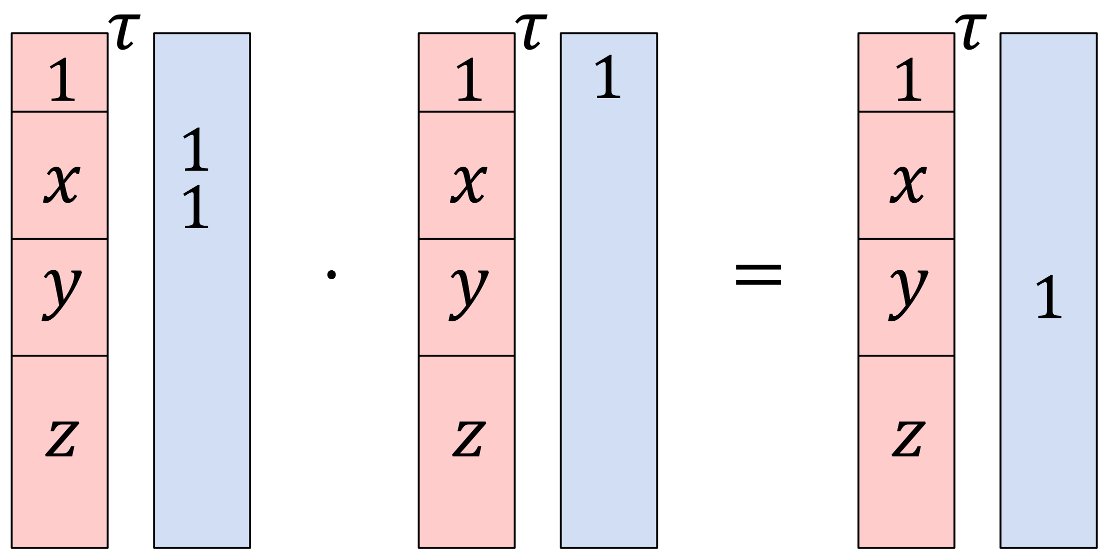
$$\begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} a_j \cdot \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} b_j = \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} c_j$$

Expressing gates as constraints:

Multiplication gate in C converted into a constraint:



Addition gate in C converted into a constraint:



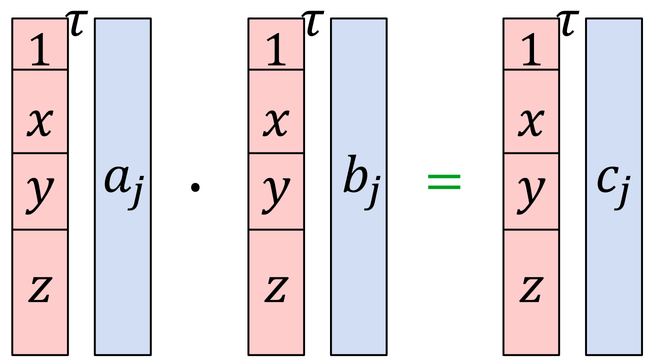
Generally, any bilinear gate.

R1CS (Rank-1 Quadratic Constraint System) ⇒ QAP (Quadratic Arithmetic Program) [GGPR13]

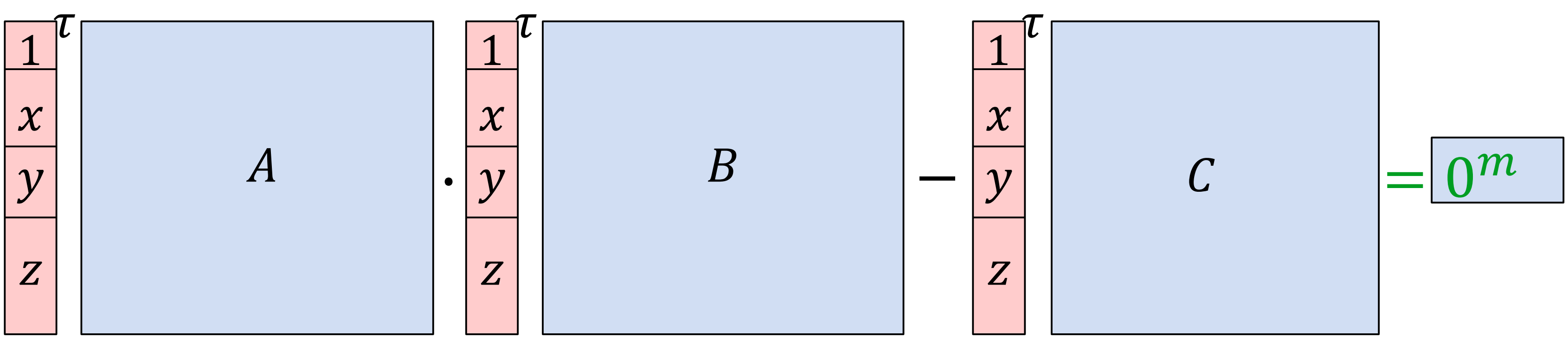
R1CS $(a_j, b_j, c_j)_{j=1}^{m}$ vectors in $\mathbb{F}^k$.

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

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



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



R1CS ⇒ QAP (cont.)

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

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

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

$$\begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} A \cdot \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} B - \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} C = 0^m$$

interpolate to $A_i(\alpha)$ — interpolate to $B_i(\alpha)$ — interpolate to $C_i(\alpha)$

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: $\left(A_i(\alpha), B_i(\alpha), C_i(\alpha)\right)_{i=1}^{k}$ 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.,
$\exists H(\alpha)$: $P_{x,y,z}(\alpha) = H(\alpha)V(\alpha)$

$$\begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} \begin{pmatrix}A_1(\alpha)\\A_1(\alpha)\\\vdots\\A_k(\alpha)\end{pmatrix} \quad \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} \begin{pmatrix}B_1(\alpha)\\B_2(\alpha)\\\vdots\\B_k(\alpha)\end{pmatrix} \quad \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau} \begin{pmatrix}C_1(\alpha)\\C_2(\alpha)\\\vdots\\C_k(\alpha)\end{pmatrix}$$

QAP ⇒ Linear PCP [GGPR13] [BCIOP13] [BCGTV13]

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

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

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

$\exists H(\alpha)$:

$P_{x,y,z}(\alpha) = H(\alpha)V(\alpha)$

$$\begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau}\begin{pmatrix}A_1(\alpha)\\A_2(\alpha)\\\vdots\\A_k(\alpha)\end{pmatrix} \quad \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau}\begin{pmatrix}B_1(\alpha)\\B_2(\alpha)\\\vdots\\B_k(\alpha)\end{pmatrix} \quad \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau}\begin{pmatrix}C_1(\alpha)\\C_2(\alpha)\\\vdots\\C_k(\alpha)\end{pmatrix}$$

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

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

- Any $\tilde{\pi}$ still commits to some low-degree $\tilde{H}(\tau)$ $\widetilde{P_{x,y,z}}(\tau)$.

QAP ⇒ Linear PCP: the algorithms

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

$$\begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau}\begin{pmatrix}A_1(\alpha)\\A_2(\alpha)\\\vdots\\A_k(\alpha)\end{pmatrix} \cdot \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau}\begin{pmatrix}B_1(\alpha)\\B_2(\alpha)\\\vdots\\B_k(\alpha)\end{pmatrix} - \begin{pmatrix}1\\x\\y\\z\end{pmatrix}^{\tau}\begin{pmatrix}C_1(\alpha)\\C_2(\alpha)\\\vdots\\C_k(\alpha)\end{pmatrix}$$

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

- Prover: compute H and its coefficient vector h;
  Output $\pi = (1, x, y, z, h)$ where h is the coefficient vector of H.
  Complexity: Dominated by computing the m coefficients of H. With suitable FFT: $\sim m \log m + (\#\text{"nonzero entries in "} A, B, C)$ field operations.
- Query: Verify: draw $\tau \leftarrow_R \mathbb{F}$, make linear queries to π according to τ.
  Complexity: $\sim 4m + 2(\#$nonzero entries in $A, B, C)$ field operations.
- Decision: check a simple quadratic equation in the answers.

Later: important for public verifiability (will use of pairings).

QAP ⇒ Linear PCP: adding ZK

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

$$\begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} \begin{pmatrix} A_1(\alpha) \\ A_2(\alpha) \\ \vdots \\ A_k(\alpha) \end{pmatrix} + \delta_1 V(\alpha) \cdot \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} \begin{pmatrix} B_1(\alpha) \\ B_2(\alpha) \\ \vdots \\ B_k(\alpha) \end{pmatrix} + \delta_2 V(\alpha) - \begin{pmatrix} 1 \\ x \\ y \\ z \end{pmatrix}^{\tau} \begin{pmatrix} C_1(\alpha) \\ C_2(\alpha) \\ \vdots \\ C_k(\alpha) \end{pmatrix} + \delta_3 V(\alpha)$$

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

Honest-Verifier Zero Knowledge:

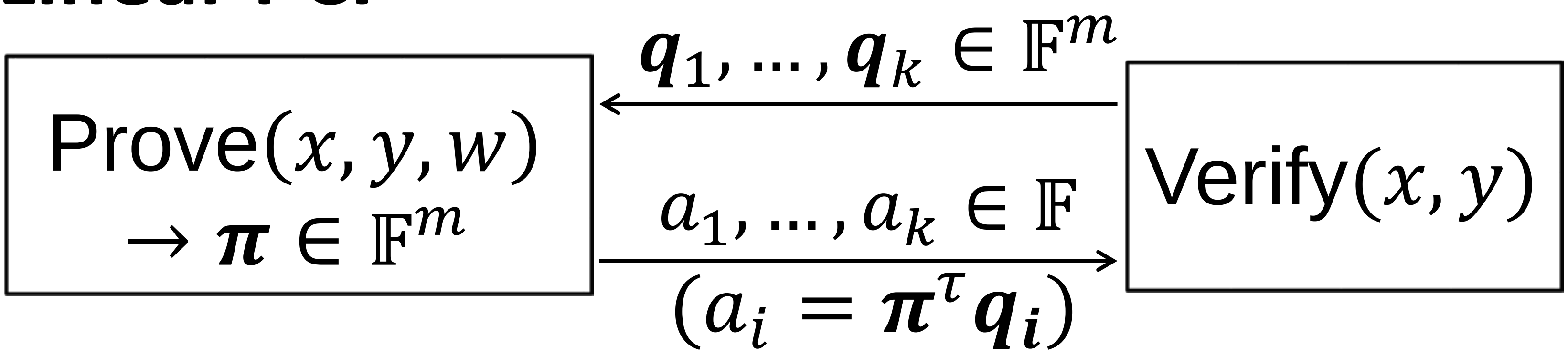
Prover adds random multiple of $V(\alpha)$ to $A, B, C(\alpha)$.

- 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 ⇒ SNARK

[BCIOP13]

Linear PCP



Intuition: send $\boldsymbol{q}_i$ 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)

Public parameters. A prime r, two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order r with generators $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively, and a pairing $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ (where $\mathbb{G}_T$ is also cyclic of order r).

(a) Key generator G

- INPUTS: circuit $C\colon \mathbb{F}_r^n \times \mathbb{F}_r^h \to \mathbb{F}_r^l$
- OUTPUTS: proving key pk and verification key vk

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \mathsf{QAPinst}(C)$; extend $\vec{A}, \vec{B}, \vec{C}$ 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 $\tau, \rho_{\mathsf{A}}, \rho_{\mathsf{B}}, \alpha_{\mathsf{A}}, \alpha_{\mathsf{B}}, \alpha_{\mathsf{C}}, \beta, \gamma \in \mathbb{F}_r$.
3. Set $\mathsf{pk} := (C, \mathsf{pk}_{\mathsf{A}}, \mathsf{pk}'_{\mathsf{A}}, \mathsf{pk}_{\mathsf{B}}, \mathsf{pk}'_{\mathsf{B}}, \mathsf{pk}_{\mathsf{C}}, \mathsf{pk}'_{\mathsf{C}}, \mathsf{pk}_{\mathsf{K}}, \mathsf{pk}_{\mathsf{H}})$ where for $i = 0, 1, \ldots, m+3$:
   $$\mathsf{pk}_{\mathsf{A},i} := A_i(\tau)\rho_{\mathsf{A}}\mathcal{P}_1\,, \quad \mathsf{pk}'_{\mathsf{A},i} := A_i(\tau)\alpha_{\mathsf{A}}\rho_{\mathsf{A}}\mathcal{P}_1\,,$$
   $$\mathsf{pk}_{\mathsf{B},i} := B_i(\tau)\rho_{\mathsf{B}}\mathcal{P}_2\,, \quad \mathsf{pk}'_{\mathsf{B},i} := B_i(\tau)\alpha_{\mathsf{B}}\rho_{\mathsf{B}}\mathcal{P}_1\,,$$
   $$\mathsf{pk}_{\mathsf{C},i} := C_i(\tau)\rho_{\mathsf{A}}\rho_{\mathsf{B}}\mathcal{P}_1\,, \mathsf{pk}'_{\mathsf{C},i} := C_i(\tau)\alpha_{\mathsf{C}}\rho_{\mathsf{A}}\rho_{\mathsf{B}}\mathcal{P}_1\,,$$
   $$\mathsf{pk}_{\mathsf{K},i} := \beta\big(A_i(\tau)\rho_{\mathsf{A}} + B_i(\tau)\rho_{\mathsf{B}} + C_i(\tau)\rho_{\mathsf{A}}\rho_{\mathsf{B}}\big)\mathcal{P}_1\,,$$
   and for $i = 0, 1, \ldots, d$, $\mathsf{pk}_{\mathsf{H},i} := \tau^i\mathcal{P}_1$.
4. Set $\mathsf{vk} := (\mathsf{vk}_{\mathsf{A}}, \mathsf{vk}_{\mathsf{B}}, \mathsf{vk}_{\mathsf{C}}, \mathsf{vk}_{\gamma}, \mathsf{vk}^1_{\beta\gamma}, \mathsf{vk}^2_{\beta\gamma}, \mathsf{vk}_{\mathsf{Z}}, \mathsf{vk}_{\mathsf{IC}})$ where
   $$\mathsf{vk}_{\mathsf{A}} := \alpha_{\mathsf{A}}\mathcal{P}_2\,, \mathsf{vk}_{\mathsf{B}} := \alpha_{\mathsf{B}}\mathcal{P}_1\,, \mathsf{vk}_{\mathsf{C}} := \alpha_{\mathsf{C}}\mathcal{P}_2$$
   $$\mathsf{vk}_{\gamma} := \gamma\mathcal{P}_2\,, \quad \mathsf{vk}^1_{\beta\gamma} := \gamma\beta\mathcal{P}_1\,, \quad \mathsf{vk}^2_{\beta\gamma} := \gamma\beta\mathcal{P}_2\,,$$
   $$\mathsf{vk}_{\mathsf{Z}} := Z(\tau)\rho_{\mathsf{A}}\rho_{\mathsf{B}}\mathcal{P}_2\,, (\mathsf{vk}_{\mathsf{IC},i})_{i=0}^{n} := (A_i(\tau)\rho_{\mathsf{A}}\mathcal{P}_1)_{i=0}^{n}\,.$$
5. Output $(\mathsf{pk}, \mathsf{vk})$.

Key sizes. When invoked on a circuit $C\colon \mathbb{F}_r^n \times \mathbb{F}_r^h \to \mathbb{F}_r^l$ 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.

(b) Prover P

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

1. Compute $(\vec{A}, \vec{B}, \vec{C}, Z) := \mathsf{QAPinst}(C)$.
2. Compute $\vec{s} := \mathsf{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_r^m$.
3. Randomly sample $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$.
4. Compute $\vec{h} = (h_0, h_1, \ldots, h_d) \in \mathbb{F}_r^{d+1}$, which are the coefficients of $H(z) := \frac{A(z)B(z)-C(z)}{Z(z)}$ where $A, B, C \in \mathbb{F}_r[z]$ are as follows:
   $$A(z) := A_0(z) + \textstyle\sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z)\,,$$
   $$B(z) := B_0(z) + \textstyle\sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z)\,,$$
   $$C(z) := C_0(z) + \textstyle\sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z)\,.$$
5. Set $\tilde{\mathsf{pk}}_{\mathsf{A}}$:= "same as pk_{A}, but with $\mathsf{pk}_{\mathsf{A},i} = 0$ for $i = 0, 1, \ldots, n$".
   Set $\tilde{\mathsf{pk}}'_{\mathsf{A}}$:= "same as $\mathsf{pk}'_{\mathsf{A}}$, but with $\mathsf{pk}'_{\mathsf{A},i} = 0$ for $i = 0, 1, \ldots, n$".
6. Letting $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$, compute
   $$\pi_{\mathsf{A}} := \langle \vec{c}, \tilde{\mathsf{pk}}_{\mathsf{A}} \rangle,\ \pi'_{\mathsf{A}} := \langle \vec{c}, \tilde{\mathsf{pk}}'_{\mathsf{A}} \rangle,\ \pi_{\mathsf{B}} := \langle \vec{c}, \mathsf{pk}_{\mathsf{B}} \rangle,\ \pi'_{\mathsf{B}} := \langle \vec{c}, \mathsf{pk}'_{\mathsf{B}} \rangle,$$
   $$\pi_{\mathsf{C}} := \langle \vec{c}, \mathsf{pk}_{\mathsf{C}} \rangle,\ \pi'_{\mathsf{C}} := \langle \vec{c}, \mathsf{pk}'_{\mathsf{C}} \rangle,\ \pi_{\mathsf{K}} := \langle \vec{c}, \mathsf{pk}_{\mathsf{K}} \rangle,\ \pi_{\mathsf{H}} := \langle \vec{h}, \mathsf{pk}_{\mathsf{H}} \rangle.$$
7. Output $\pi := (\pi_{\mathsf{A}}, \pi'_{\mathsf{A}}, \pi_{\mathsf{B}}, \pi'_{\mathsf{B}}, \pi_{\mathsf{C}}, \pi'_{\mathsf{C}}, \pi_{\mathsf{K}}, \pi_{\mathsf{H}})$.

(c) Verifier V

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

1. Compute $\mathsf{vk}_{\vec{x}} := \mathsf{vk}_{\mathsf{IC},0} + \sum_{i=1}^n x_i \mathsf{vk}_{\mathsf{IC},i} \in \mathbb{G}_1$.
2. Check validity of knowledge commitments for A, B, C:
   $$e(\pi_{\mathsf{A}}, \mathsf{vk}_{\mathsf{A}}) = e(\pi'_{\mathsf{A}}, \mathcal{P}_2)\,, e(\mathsf{vk}_{\mathsf{B}}, \pi_{\mathsf{B}}) = e(\pi'_{\mathsf{B}}, \mathcal{P}_2)\,,$$
   $$e(\pi_{\mathsf{C}}, \mathsf{vk}_{\mathsf{C}}) = e(\pi'_{\mathsf{C}}, \mathcal{P}_2)\,.$$
3. Check same coefficients were used:
   $$e(\pi_{\mathsf{K}}, \mathsf{vk}_{\gamma}) = e(\mathsf{vk}_{\vec{x}} + \pi_{\mathsf{A}} + \pi_{\mathsf{C}}, \mathsf{vk}^2_{\beta\gamma}) \cdot e(\mathsf{vk}^1_{\beta\gamma}, \pi_{\mathsf{B}})\,.$$
4. Check QAP divisibility:
   $$e(\mathsf{vk}_{\vec{x}} + \pi_{\mathsf{A}}, \pi_{\mathsf{B}}) = e(\pi_{\mathsf{H}}, \mathsf{vk}_{\mathsf{Z}}) \cdot e(\pi_{\mathsf{C}}, \mathcal{P}_2)\,.$$
5. Accept if and only if all the above checks succeeded.

[PGHR13] assumptions

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

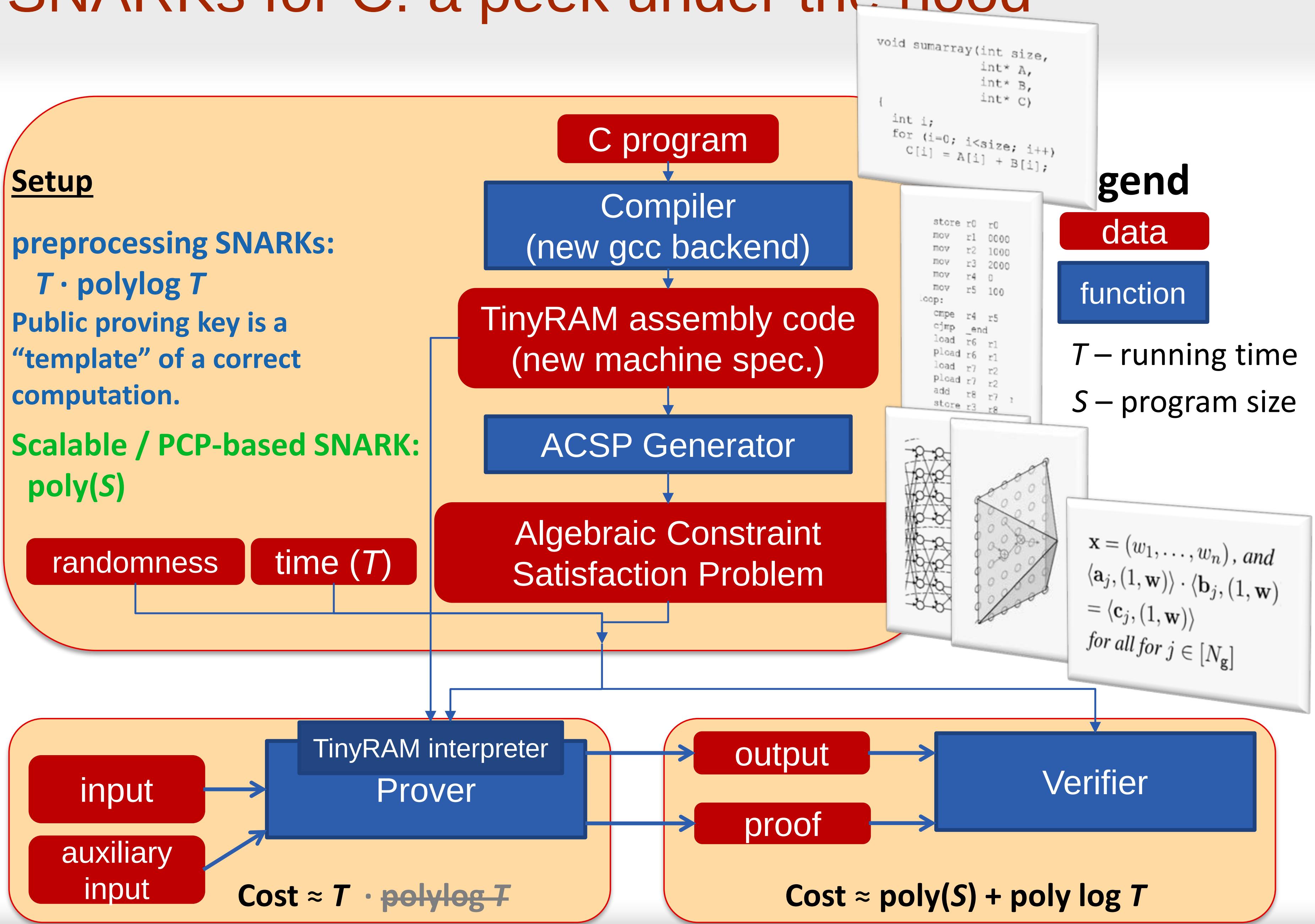
- q-power Diffie-Hellman
- q-strong Diffie-Hellman
- q-power Knowledge of Exponent

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

$$Pr[\quad (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\kappa) \,;\, g \leftarrow \mathbb{G}\setminus\{1\} \,;\, \alpha, s \leftarrow \mathbb{Z}_p^* \,;$$
$$\sigma \leftarrow (p, \mathbb{G}, \mathbb{G}_T, e, g, g^s, \ldots, g^{s^q}, g^\alpha, g^{\alpha s}, \ldots, g^{\alpha s^q}) \,;$$
$$(c, \hat{c} \,;\, a_0, \ldots, a_q) \leftarrow (\mathcal{A} \,\|\, \chi_{\mathcal{A}})(\sigma, z) \quad :$$
$$\hat{c} = c^\alpha \wedge c \neq \textstyle\prod_{i=0}^{q} g^{a_i s^i}] = \mathtt{negl}(\kappa)$$

for any auxiliary information $z \in \{0,1\}^{\text{poly}(\kappa)}$ that is generated independently of α. Note that $(y; z) \leftarrow (\mathcal{A} \,\|\, \chi_{\mathcal{A}})(x)$ 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



zkSNARK backend implementations

- Pinocchio/Geppetto
  `https://vc.codeplex.com`
  [PGHR13] [CFHKKNPZ15]
- libsnark
  `github.com/scipr-lab/libsnark`
  [BCGTV13a] [BCTV14crypto] [BCTV14usenix] …
- snarklib
  `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
  - speedof 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

1M arithmetic gates, 1000-bit input, desktop PC	80-bit security	128-bit security
Generator	97 s	117 s
Prover	115 s	147 s
Verifier	4.9 ms ($4.7 + 0.0004\|x\|$ ms)	5.1 ms ($4.8 + 0.0005\|x\|$ ms)
Proof size	230 B	288 B

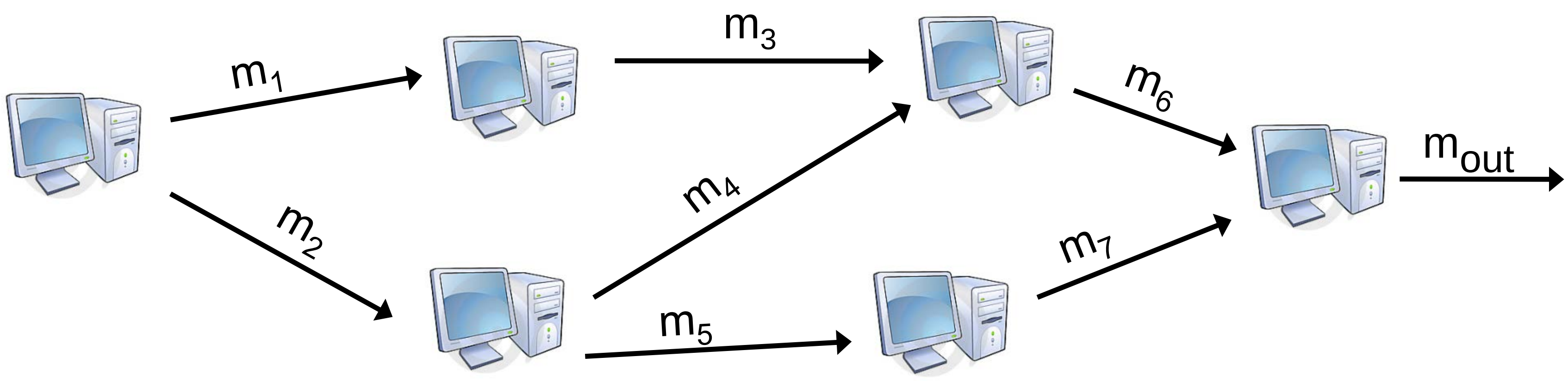
- Full code, MIT license `github.com/scipr-lab/libsnark`

SNARKs for C general programs

Feasibility			Network		C program size		Program running time		Papers
Theory (*poly*)	Fast verify	Fast prove	1 hop	Any	Small	Any	Short	Any	
✓			✓						[Kilian 92] [Micali 94] [Groth 2010]
✓			✓	✓					[Chiesa Tromer 2010] [Valiant 08]
✓	✓		✓		✓		✓		[Ben-Sasson Chiesa Genkin Tromer Virza 2013] [Parno Gentry Howell Raykova 2013]
✓	✓		✓		✓	✓	✓		[Ben-Sasson Chiesa Tromer Virza 2014 USENIX Security]
✓	✓	?	✓	✓	✓	✓	✓	✓	[Ben-Sasson Chiesa Tromer Virza 2014 CRYPTO]

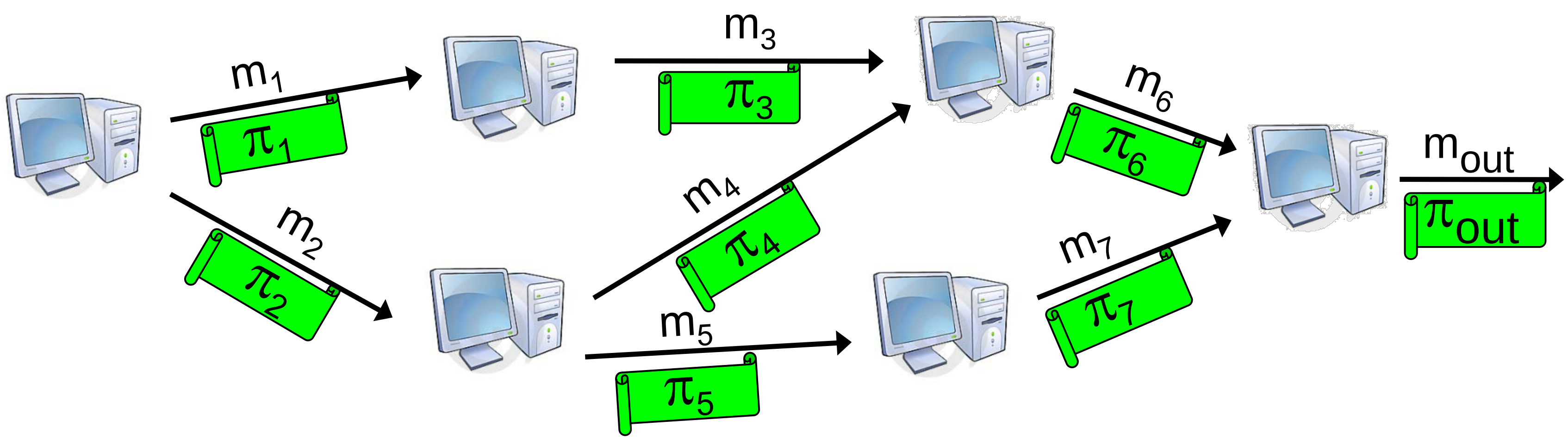
Tighter frontends from high level (Geppetto, Buffet…) at cost in universality, supporting random accesses and generan control flow, and scalability.

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.

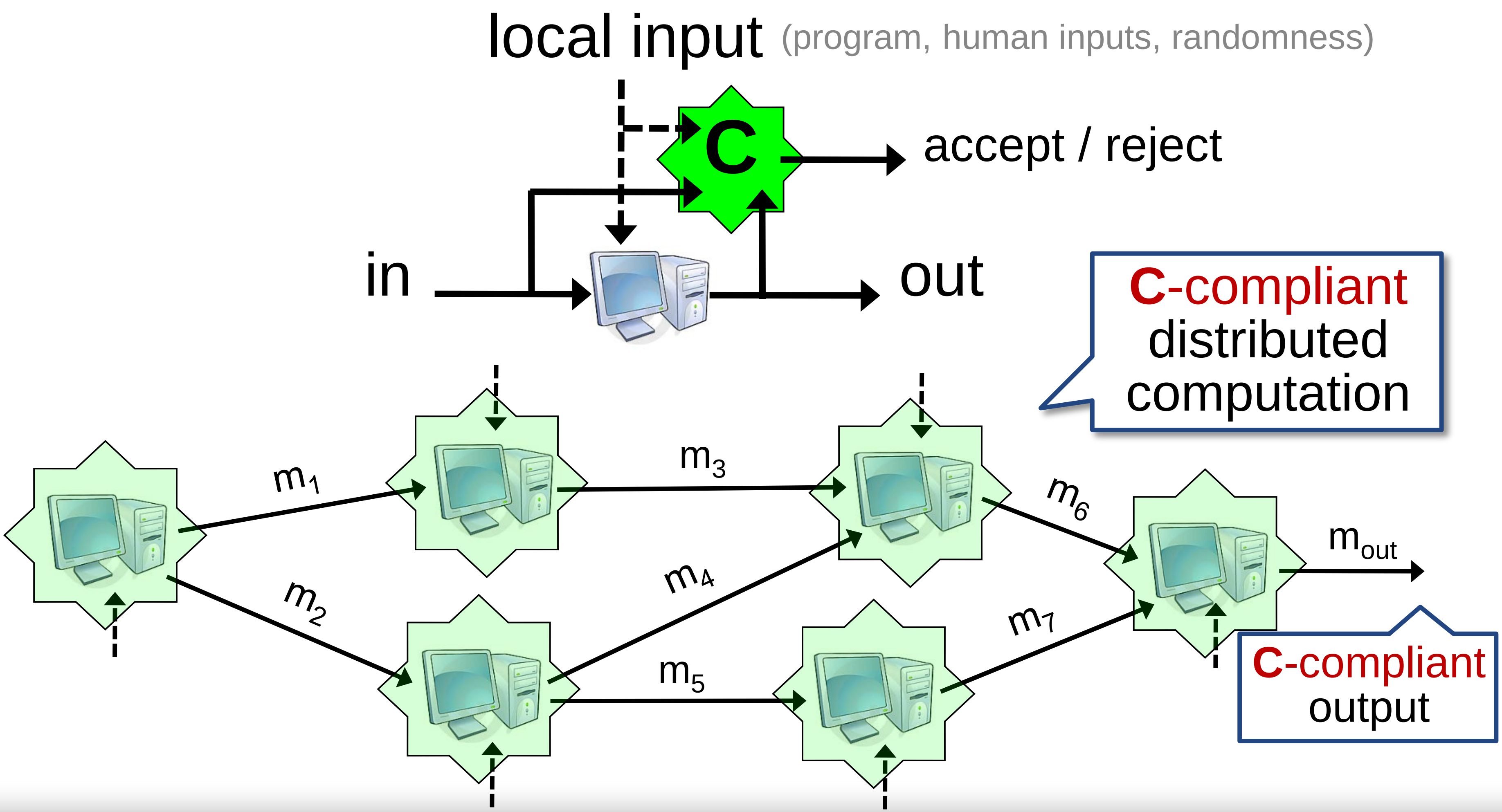
Proof-Carrying Data (cont.)



- 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.

C-compliance

System designer specifies his notion of **correctness** via a **compliance predicate** **C**(incoming, local inputs, outgoing) that must be locally fulfilled at every node.



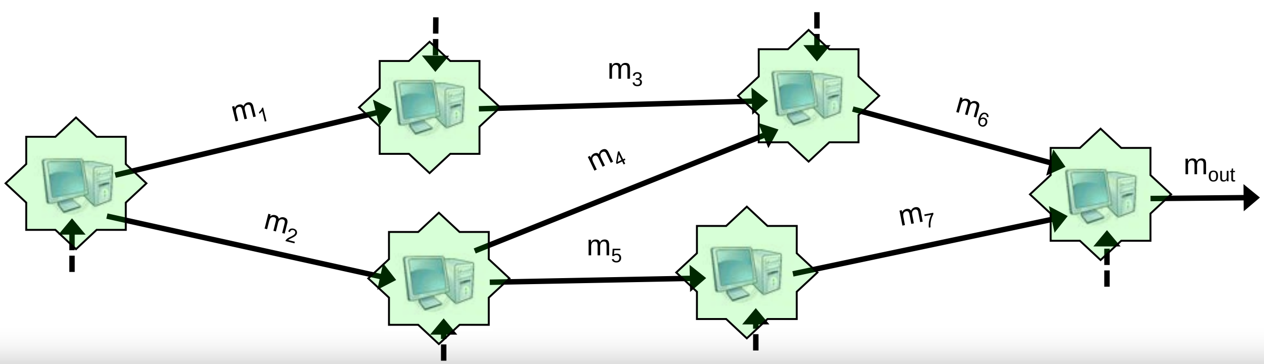
Examples of **C**-compliance

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

C = “the output is the result of correctly computing a <u>prescribed</u> program”

C = “the output is the result of correctly executing some program <u>signed by the sysadmin</u>”

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

Primitive	Attacks		Guarantees		Functionality		Communication	Assumptions
	Leakage	Tampering	Correctness	Secrecy	Function class	Output form		
FHE	ANY	none	yes	YES	Circuits	Encrypted	Minimal	Computational
		ANY	no					
Obfuscation (VBB)	ANY	ANY	YES	YES	YES	Plaintext	Minimal	Impossible. Special cases/heuristic
Leakage resilience	Varies	none	yes	YES	Varies	Plaintext	Minimal	Varies
Tamper resilience	Varies	Varies	Varies	Varies	Varies	Plaintext	Minimal	Varies
TPM, SGX	Some	Some	Yes	Yes	ANY	Plaintext	Minimal	Secure hardware
Computational proofs (SNARK/ PCD)	ANY	ANY	YES	no	RAM, distributed	Plaintext + proof	Minimal	Exotic computational / oracle
Multiparty computation	ANY	ANY	YES	YES	ANY	Plaintext	Heavy interaction	Mild computational
Garbled circuits	ANY	none	yes	YES	Circuits	Plaintext	Preprocessing + minimal	Mild computational
		ANY	no					