Zero-knowledge proofs for Bitcoin scalability and beyond

Madars Virza

(based on joint works with Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Daniel Genkin, Matthew Green, Shaul Kfir, Ian Miers and Eran Tromer)

Outline

1. A very brief intro to zero-knowledge proofs

2. The power of ZK for Bitcoin scalability

3. Zero-knowledge in practice …

Can zero-knowledge proofs be implemented?

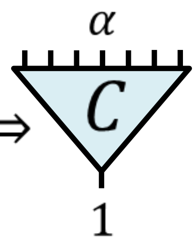
How to “program” zero-knowledge proofs?

How to deploy them in real systems?

[GMR89]

Zero Knowledge

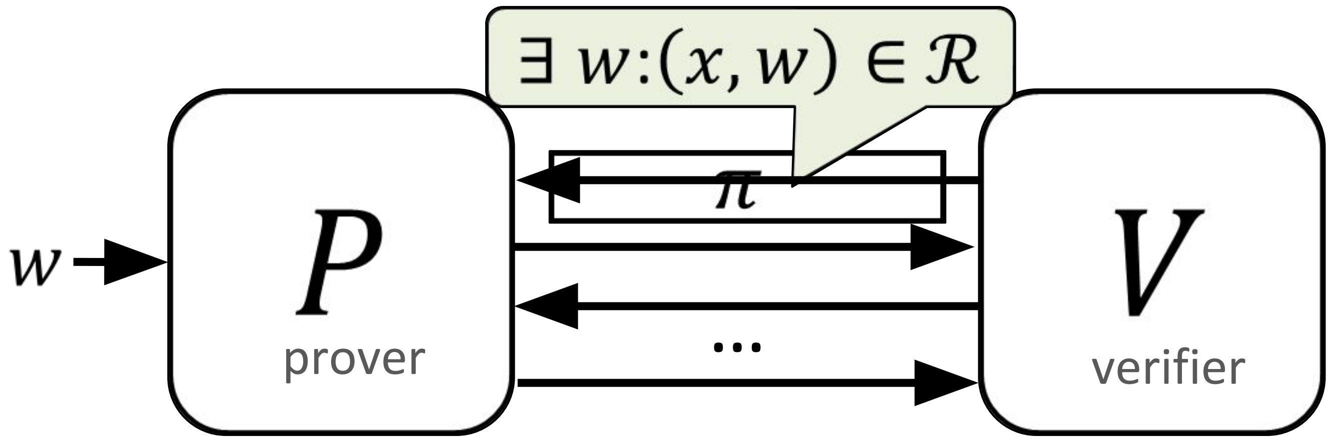
E.g.
Circuit-SAT

$(C, \alpha) \in \mathcal{R}_{\text{CSAT}} \Leftrightarrow$ 

$\mathcal{R}$ — A relation: a set of pairs (x, w)

instance x, *witness* w

x

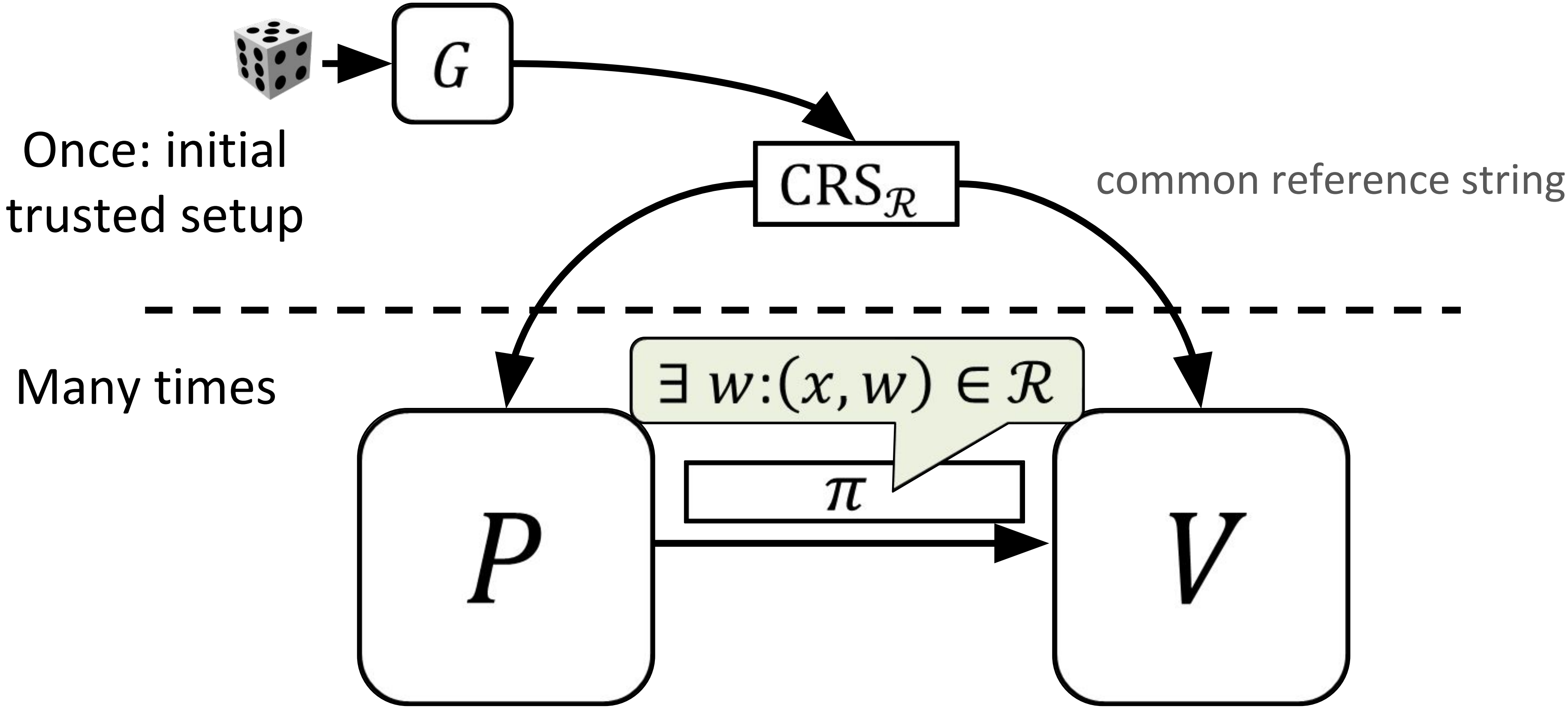


Completeness: $(x, w) \in \mathcal{R} \Rightarrow P$ can make V accept

Soundness: $(x, w) \notin \mathcal{R} \Rightarrow P$ can't make V accept

Zero knowledge: V learns nothing else other than $\exists w$

NIZKs



Thm: **Impossible** for NP (without any help) [GMR85, GO94]

Thm: **Possible** for NP with help of CRS. [BFM88, NY90, BDMP91]

Many scalability problems can be traced back to questions about privacy

Fungibility: if all transactions are public, receiving "wrong" change for coffee could **taint & devalue your coins**

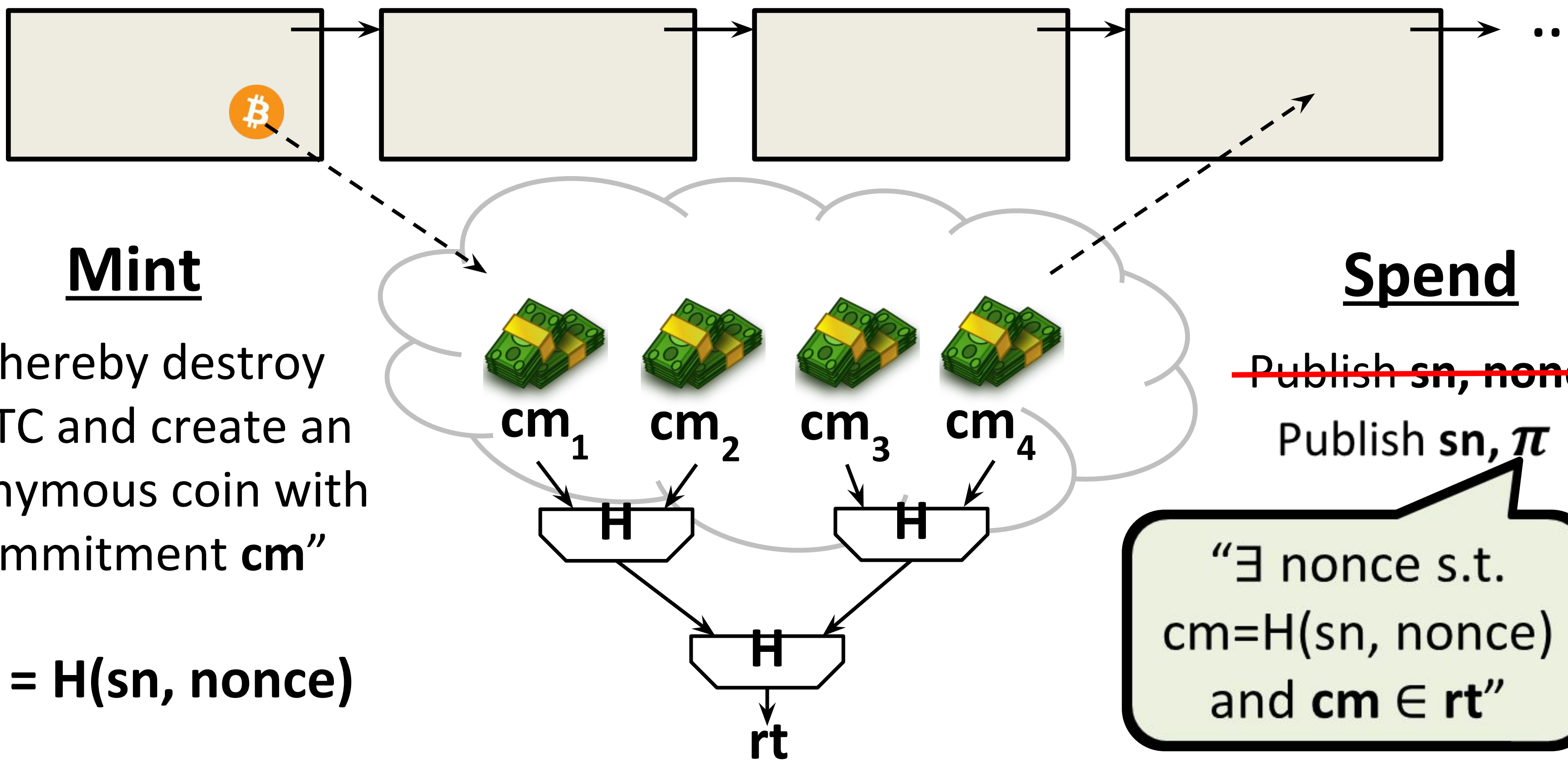
Solvency: if proving solvency is privacy liability (thus not done) you get **distrust in traditional service providers**

Decentralization: if miners can't covertly repurpose their work, you get strong incentives for **pooling and miner centralization**

Claim: zero-knowledge proofs helpful for all of above!
"Proof": by example…

ZK ⇒ privacy and fungibility

(based on a scheme by Sanders and Ta-Shma)



Mint

"I hereby destroy 1 BTC and create an anonymous coin with commitment **cm**"

cm = H(sn, nonce)

Spend

~~Publish **sn, nonce**~~

Publish **sn, π**

"∃ nonce s.t. cm=H(sn, nonce) and **cm** ∈ **rt**"

(1) π is **zero-knowledge** and **unlinkable** to **cm**, yet ensures **integrity**

(2) Publishing **sn** ensures **no double-spending**

Zerocash builds upon this adding direct payments, divisibility, …

ZK ⇒ privacy-preserving proofs of solvency

solvency = **"assets > liabilities"** (Provisions [DBBCB15])

privacy-preserving = **"reveal nothing about keys & balances"**

Two hiding commitments: $\mathbf{cm}_{\mathbf{asset}}$=H($\mathbf{v}_{\mathbf{asset}}$, r) and $\mathbf{cm}_{\mathbf{liab}}$ = H($\mathbf{v}_{\mathbf{liab}}$, r')

Three kinds of statements:

1. Exchange is **solvent**: π

"I can open $\mathbf{cm}_{\mathbf{asset}}$ and $\mathbf{cm}_{\mathbf{liab}}$ to $\mathbf{v}_{\mathbf{asset}}$ and $\mathbf{v}_{\mathbf{liab}}$ where $\mathbf{v}_{\mathbf{asset}} > \mathbf{v}_{\mathbf{liab}}$"

2. Each account balance is included in $\mathbf{v}_{\mathbf{liab}}$:

User	Balance	
Alice	v_1	→ $\mathbf{cm}_1$
Bob	v_2	→ $\mathbf{cm}_2$
Charlie	v_3	→ $\mathbf{cm}_3$
…	…	

a) Publish commitments to all balances

b) Prove to user i that $\mathbf{cm}_{\mathbf{i}}$ opens to $\mathbf{v}_{\mathbf{i}}$

c) Prove that $\mathbf{cm}_{\mathbf{liab}}$ sums values of all $\mathbf{cm}_{\mathbf{i}}$

3. Exchange controls at least $\mathbf{v}_{\mathbf{asset}}$ BTC:

Fix a large anonymity set of public keys and their balances.

Prove knowledge of private keys for a subset that controls $\mathbf{v}_{\mathbf{asset}}$ BTC.

Outline

1. A very brief intro to zero-knowledge proofs

2. The power of ZK for Bitcoin scalability

3. Zero-knowledge in practice …

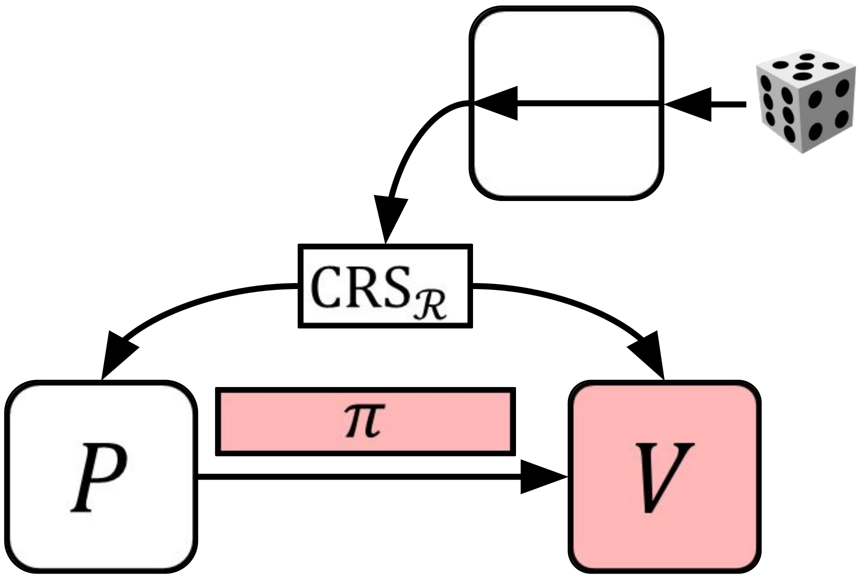
Can zero-knowledge proofs be implemented?

How to "program" zero-knowledge proofs?

How to deploy them in real systems?

NIZKs vs SNARKs

E.g. Schnorr proofs, CT range proofs



Efficiency:

$$|\pi| = \tilde{O}_\lambda(T_\mathcal{R})$$
$$\text{time}(V) = \tilde{O}_\lambda(T_\mathcal{R})$$

Sufficient assumptions:

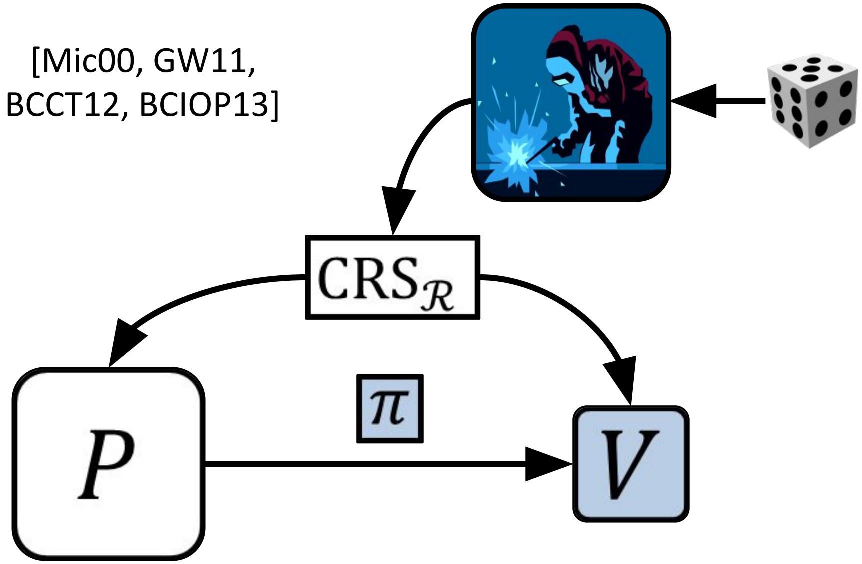
- trapdoor permutations
- decision linear assumption (DLIN)

"Simple" CRS

SNARKs

(Succinct Non-Interactive Arguments of Knowledge)

[Mic00, GW11, BCCT12, BCIOP13]



Efficiency:

$$\boxed{\begin{aligned}|\pi| &= O_\lambda(1)\\ \text{time}(V) &= O_\lambda(|x|)\end{aligned}}$$

Sufficient assumptions:

- random oracle
- knowledge-of-exponent [D92, HT98]

"Complex" CRS

Finding a SNARK

(i) Theoretical constructions

[Killian92, Micali94, Valiant08, Mie08, DL08, Groth10, GLR11, BCCT12, DFH12, BC12, Lipmaa12, BCIOP13, GGPR13, PGHR13, BCGTV13, Lipmaa13, FLZ13, BCCT13, BCTV14a,BCCGLRT14, BCTV14b, Lipmaa14, KPPSST14, ZPK14, DFGK14, WSRBW15, BBFR15, CFHKKNPZ15]

(ii) Working prototypes

Buffet & Pantry	www.pepper-project.org	[BFRSBW14, WSRBW15]
libsnark	libsnark.org	[BCGTV13,BCGTV14]
Pinocchio & Geppetto	vc.codeplex.com	[PGHR13,CFHKKNPZ15]

Most have full **source code available**!

(iii) Implemented systems

SNARKs are **feasible** for certain applications!

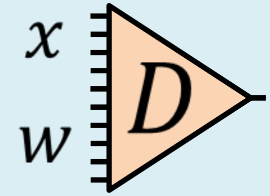
E.g.: Zerocash [BCGGTV14], **Hawk** [MSKK15], ...

How to program SNARKs

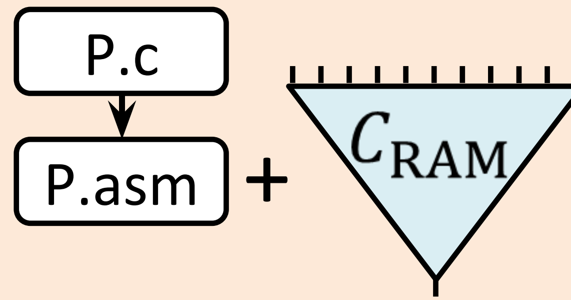
Relation I have in mind:
Hashes, Merkle trees,
digital signatures,…

$\overset{?}{\Longrightarrow}$

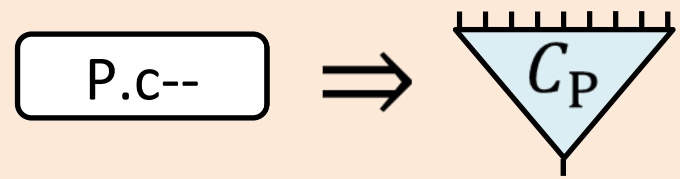
Relations SNARKs understand:
Circuit
satisfiability



The "SNARKS for C" approach:
1. Pick a CPU and write universal circuit C_{RAM} for it
2. Write a C program P that decides $\mathcal{R}$
3. Compile P to assembly & plug into C_{RAM}

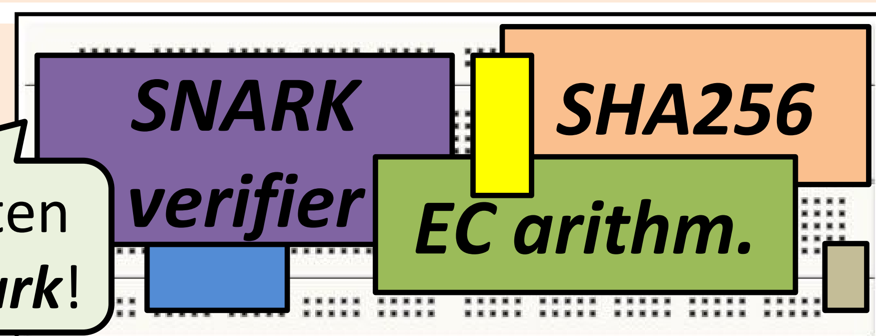


The program analysis approach:
1. Write P in restricted subset* of C
2. Use a circuit generator for that subset
(* - all memory accesses & bounds on loops must be known at compile time)



The "gadget DSL" approach:
Write subcircuits for each
component o
compose the the

Lots of pre-written gadgets in *libsnark*!



SNARK performance in practice

Verification time: only depends on $|x|$, usually ms in practice.

Prover performance = base SNARK performance(size of circuit)

[MKKS15] prover benchmarks for **470k** gate circuit:

Pinocchio: 1242s *libsnark*: 33s

Concretely: implementing **SHA256** compression function

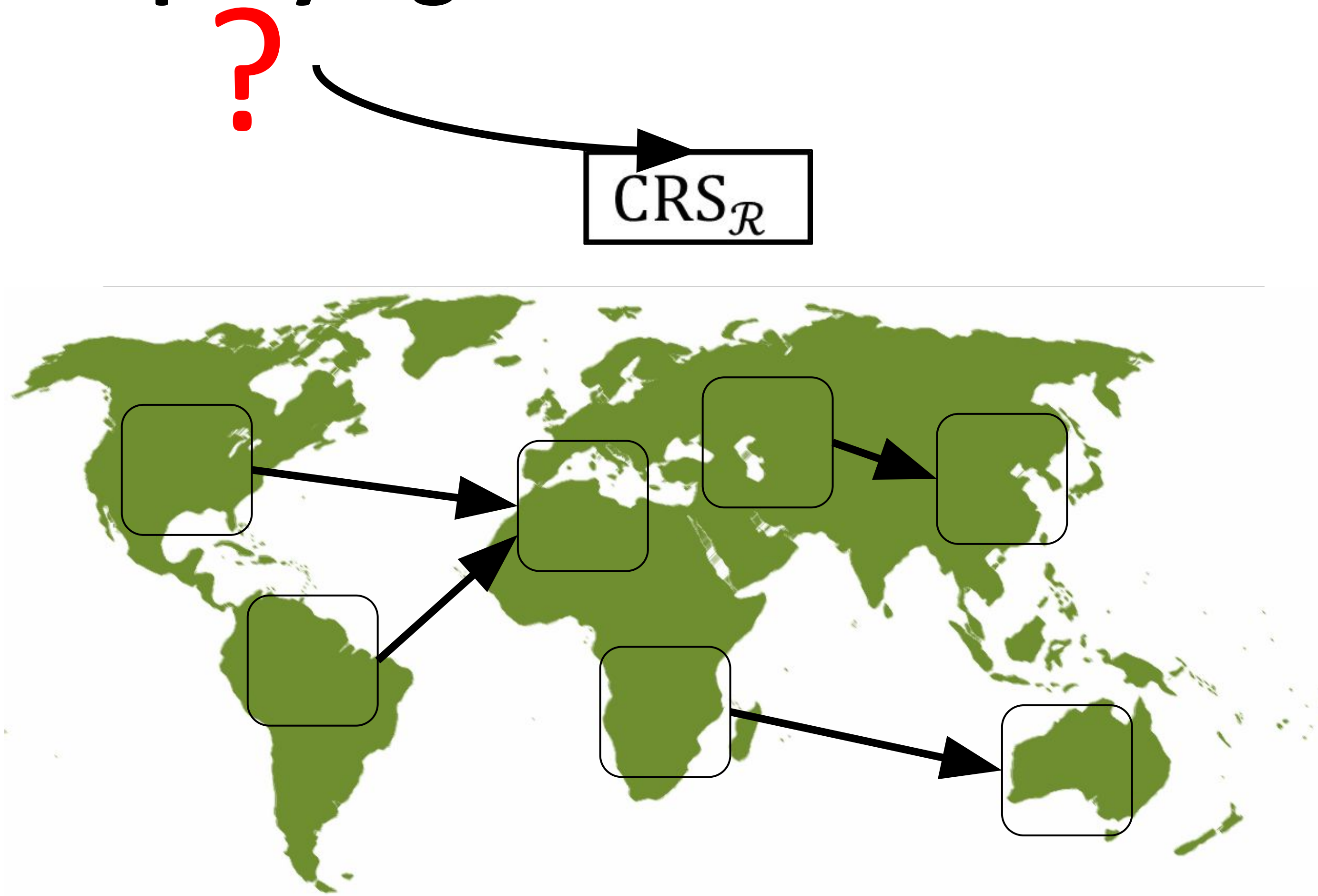
Approach	Compression function size	"Capacity" of 470k gate circuit
SNARKS for C / TinyRAM		
Pinocchio circuit generator*		
libsnark gadgets		

(* invoked on a SH256 C implementation from

Crucial for efficient use: **relation engineering**

- Try to check local properties ("is the tx OK?" not "is the chain OK?")
- Understand and leverage non-determinism
- Consider SNARK-friendly crypto [BCTV14b, KZMQCPPSS15]

Deploying NIZKs and SNARKs



Q: In practice, who generates the CRS?

Consequences of a “bad” CRS

1) Zero-knowledge still holds:
e.g. Zerocash remains private even with a bad CRS)

2) Soundness breaks:
adversary can prove false statements

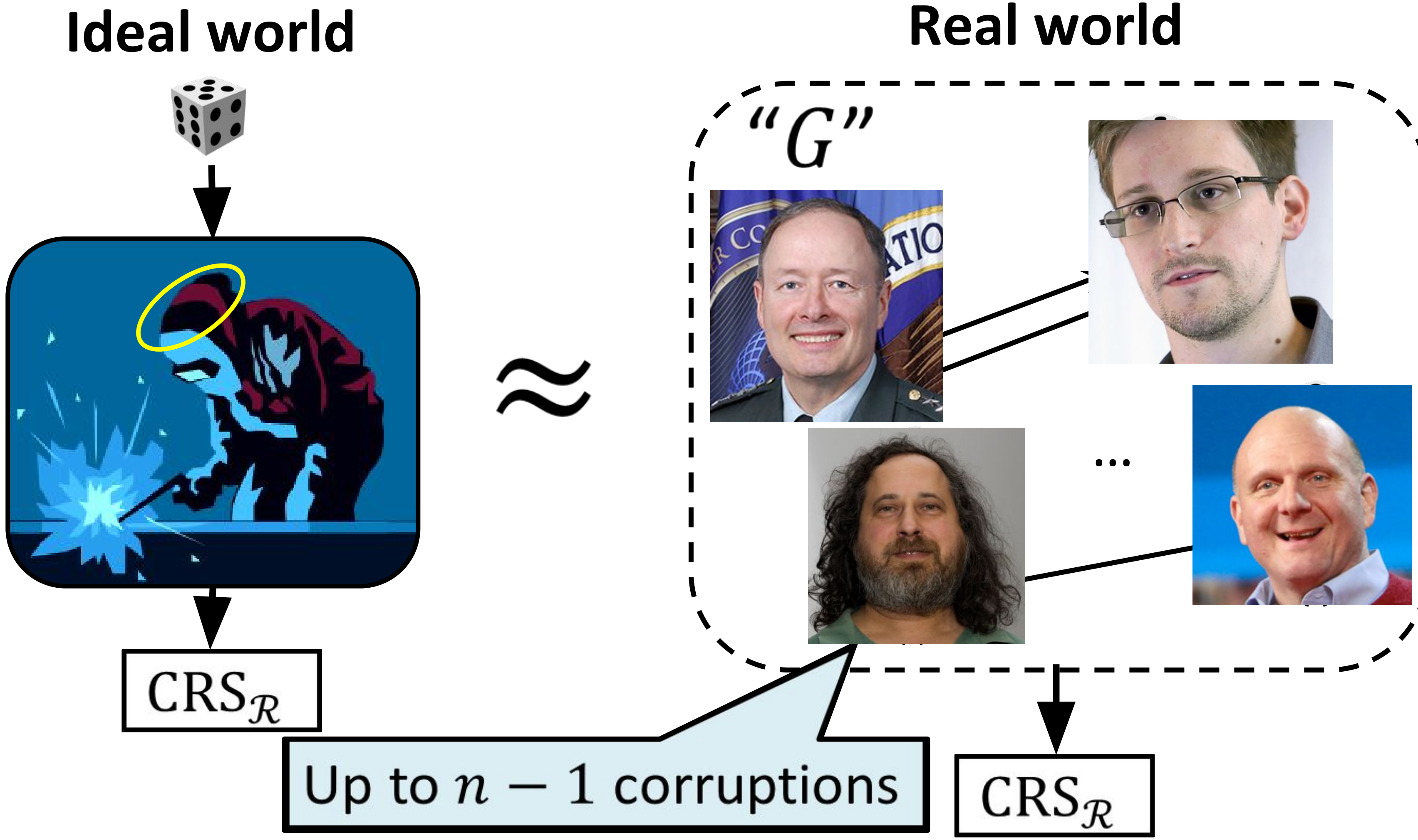
Some uses of SNARKs are not consensus-critical

E.g.: Peter Todd’s proposal for faster block propagation:
Send block header + proof of “∃ block”

bad CRS → a couple of lost blocks, but long term durability OK!

Q: Can consensus-critical SNARKs be deployed?

Goal: distributed protocol for CRS of SNARKs



Result: a protocol achieving this! [BCGTV15]

Costs for Zerocash CRS: 4h/party CPU and 13GB/party data.

Conclusion

Many scalability problems can be traced back to privacy

Zero-knowledge proofs are a very useful tool for building privacy-preserving systems.

Generic zero-knowledge is not "ten-year-away crypto"

… it was "ten-year-away crypto" *ten years ago*!

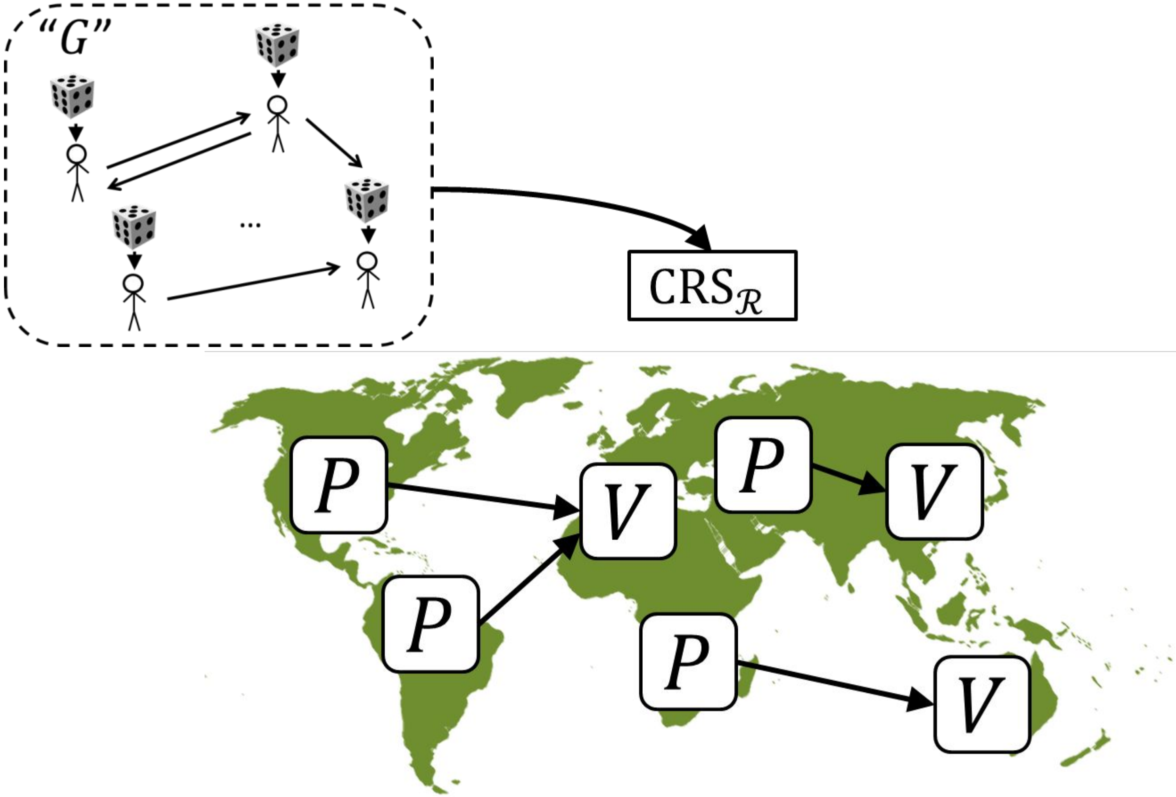
Feasible today: can be programmed & can be deployed!

Call for collaboration: **help us improve *libsnark***!

We seek all kinds of contributions: security audits, performance enhancements, new features, …

http://libsnark.org/help

Thank you!



http://libsnark.org/help