Verifying Computations (with secrets) in the Blink of an Eye

Matteo Campanelli
IMDEA Software
matteo.campanelli@imdea.org

UCM, July 1st 2019

“Just Stop Owning it” — The Rise of Cloud Computing

- Nowadays businesses usually **outsource** computation.
  - The bulk of it is rented, remote or both.
- Service Providers (SPs) can provide storage, computation, managed desktop, etc.

Advantages:

- Reduced infrastructure;
- Flexibility;
- Cost at scale.

Example:
A lone PhD student can analyze large genome data by renting 100 computers for 10 hours.
And for less then $200.

What Risks When Delegating?

- **Complexity:**
  the computing capabilities I'm renting are complex systems and their execution may not always be correct.

- **Incentives:**
  Service Providers do not necessarily have strong incentives to ensure correctness;

- **Opaqueness** (rented computers are "black boxes"):
  misconfigurations, HW problems, malicious operations, etc.,
  these are all hard to detect.

How can we ever trust results computed by a third party then?

Some Solutions

- **Replicating**
  - Caveat: HW and SW configuration might be homogeneous
- **Auditing** (checking the responses in a small sample)
  - Caveat: this doesn't help if the failures are too infrequent
- **Trusted HW**
  - Caveat: assumes a chain of trust/additional assumptions

Proof-based verification:

What if the server returned its results along with a (probabilistic) proof that the results were computed correctly?

This Talk: Proofs and Computation

Part I:
Proving Correctness of Computation

- The setting
- State of the art

Part II:
Beyond correctness — Proving Knowledge of Secrets

- zkSNARKs and their applications

Part I:
Proving Correctness of Computation

Proof-based Verification —Setting

*"Please compute function **f** on data **D**"*

*"Here is result **y=f(D)** and a proof **π** "*

Verify(f, D, y , π)
=> accept/reject

(weak) Client

(powerful) Server

Examples of functions *f:*

- *video encoding*
- *ML training*
- *image processing*
- *data analysis*

Proof-based Verification — Desiderata

(weak) Client

"Please compute function $\boldsymbol{f}$ *on data* $\boldsymbol{D}$*"*

"Here is result $\boldsymbol{y}=\boldsymbol{f(D)}$ *and a proof* $\boldsymbol{\pi}$ *"*

Desired Properties:

- *short proof (succinctness)*
- *inexpensive to verify*
- *little overhead for server*

(powerful) Server

Is this possible?

Would (e.g.) an average phone be able to verify a herd of (untrusted) supercomputers?

Inside Proof-Based Verification

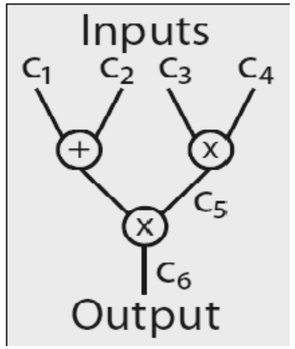
Long list of works:
[ALMS98, GKR08, GGPR13, …, Hyrax17, LegoSNARK19, …]

Basic Approach:

C Code

```
1  #include <Rcpp.h>
2
3  // [[Rcpp::export]]
4  int fibonacci(const int x) {
5    if (x < 2)
6      return x;
7    else
8      return (fibonacci(x - 1)) + fibonacci(x - 2);
9  }
10
11 /*** R
12 # Call the fibonacci function defined in C++
13 fibonacci(10)
14 */
```

Arithmetic Circuit



System of Equations

$$x - 2y - 3z = 0$$
$$3x + 2y - z = 0$$
$$3y + z = 0$$

Single randomized equation that client can verify efficiently

A “proof” is usually
a cryptographic encoding of this

What Can We Achieve Nowadays?

- **Very short proofs for <u>any</u> program (<1KB)** [Groth16]
- **How Efficiently?**
  - Verification is super fast;
    - multiplication of two 1000x1000 matrices: few milliseconds
  - Proving is meh (can give a 100-10000x overhead)
    - matrix multiplication above: few minutes

Main Challenges

- **Main challenge**: reducing resources for proving.
  - **Example:** Hashing a 13kB file (with SHA2) requires:
    >256GB RAM; few hours of work.

- **Some recent works in this directions:**
  - [WZCPS18]: Distributed generation of proofs (100x speedup)
  - [**C**FQ19]: LegoSNARK — breaking proof generation in modular parts
    - in some cases 3 orders of magnitude speedup

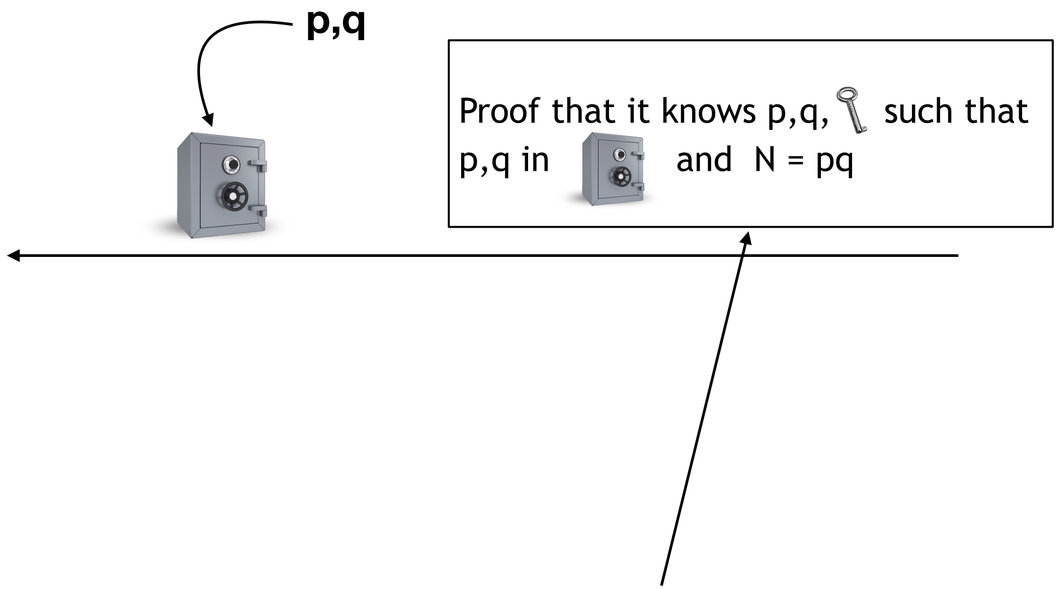
Part II:
Beyond Verifiable Computation — Proving Knowledge of Secrets

(Not So Much of)

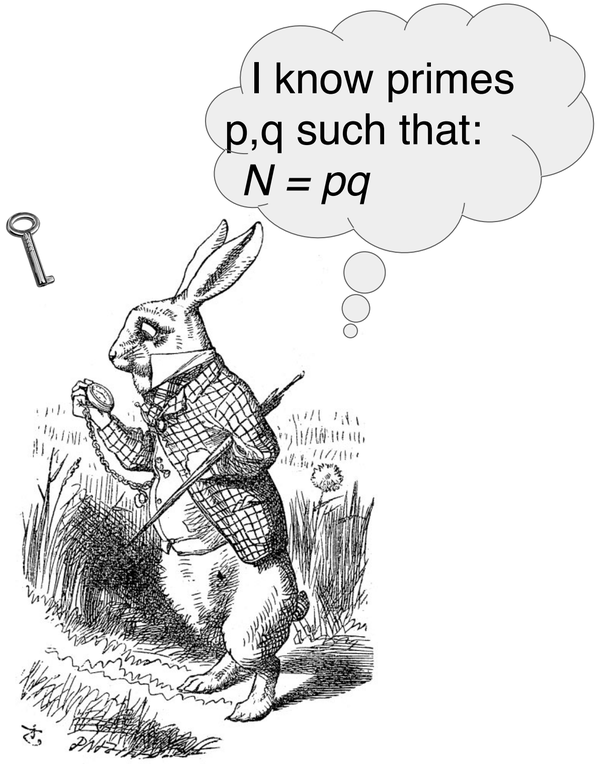
A Toy Example: Selling Secrets

The start of a sketchy transaction:

Buyer/Verifier



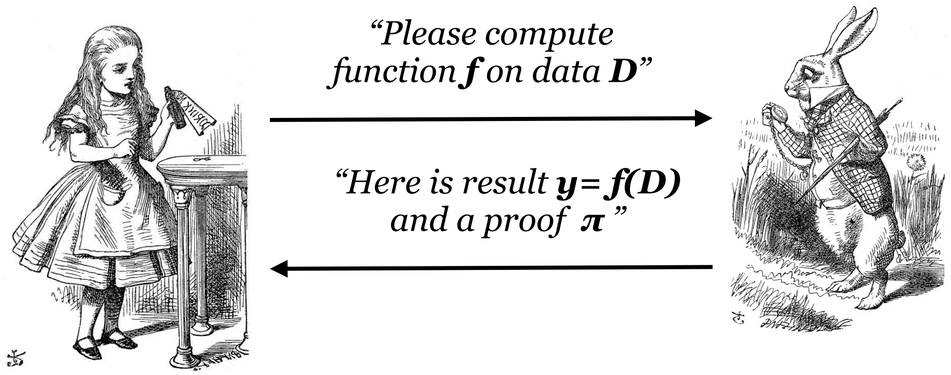
This “box” should
a) **convince** Buyer (proof)
b) not reveal anything on the secrets
(**zero-knowledge**)



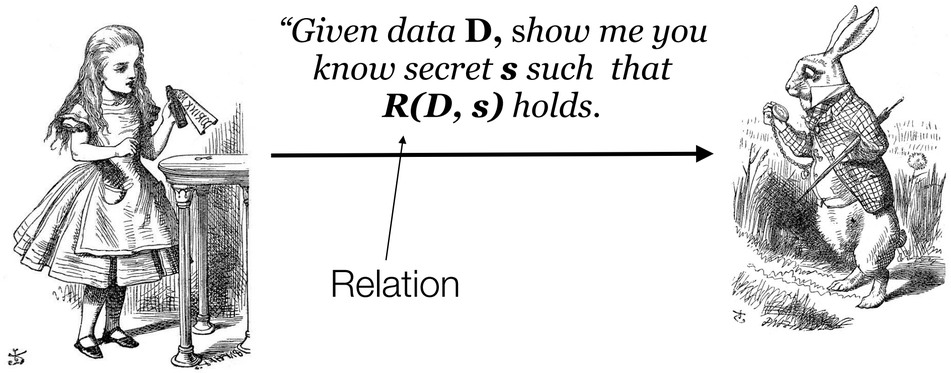
Seller/Prover

Proofs with Secrets: Zero-Knowledge Proofs

So far: Verifiable Computation



Now: Zero-Knowledge Proofs



Example from before:

D *was N and* 

s *was (p,q) and* 

R *was ~ "decryption and product"*

Focus of this talk: zkSNARKs

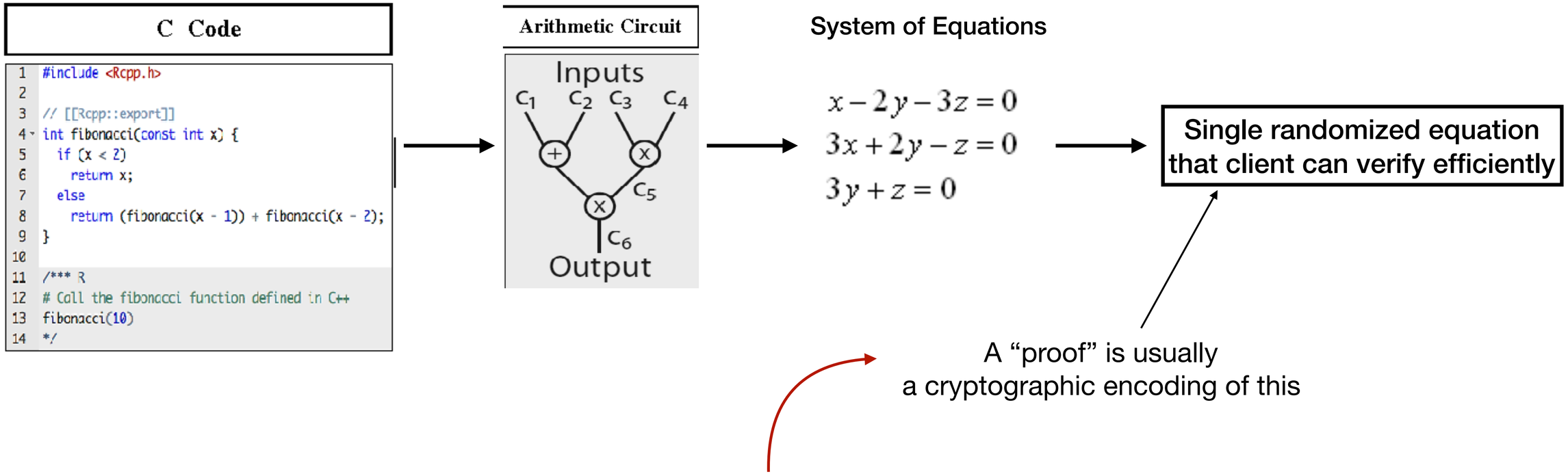
(zk **S**uccinct **N**on-Interactive **A**rguments of **K**nowledge)

Informally: leaks nothing except that the relation holds.

Informally: we *could* efficiently "extract" the secret from the prover.

zkSNARKs and Verifiable Computation

Techniques (and most of the efficiency) are the same:



NEXT: Applications.

Some General Applications of zkSNARKs

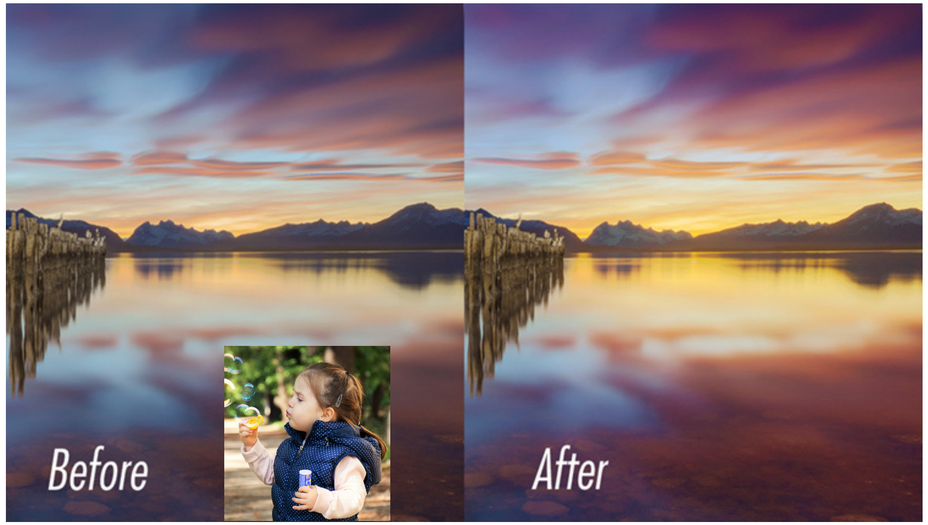
Authenticating Pictures

Takes original pic
+
Signs it

Publishing

Q: Can we still authenticate pictures if we first need to (slightly) modify them?

Authenticating *Sensitive* Pictures



Takes original pics
+
Signs it

Photoshop

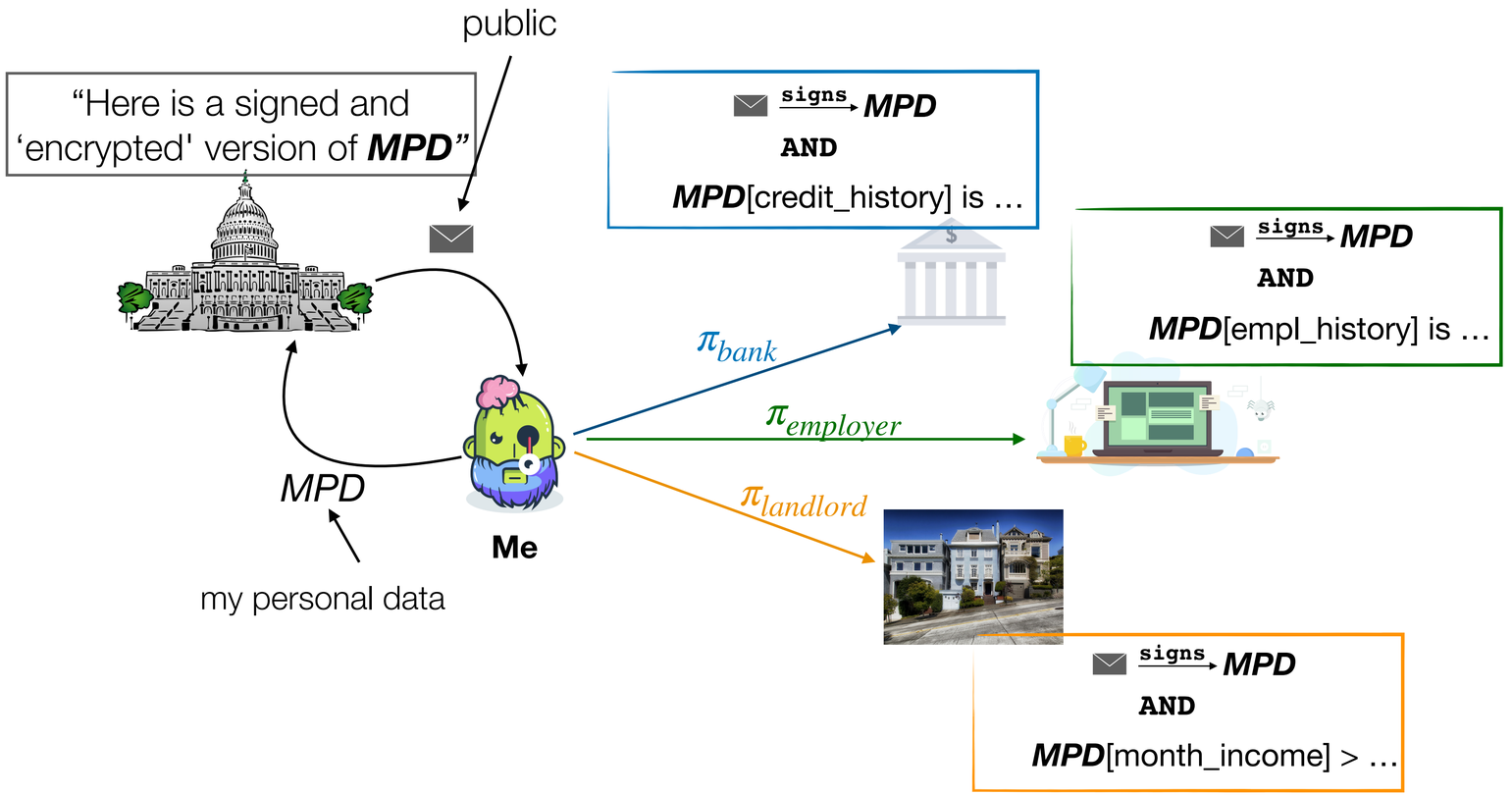
Publishing

Q: How to prove the edited pic is a transformation of an authentic pic?
(without leaking the original pic)

A: just use a zkSNARK stating knowledge of *"a signed pic to which blurring has been applied."*

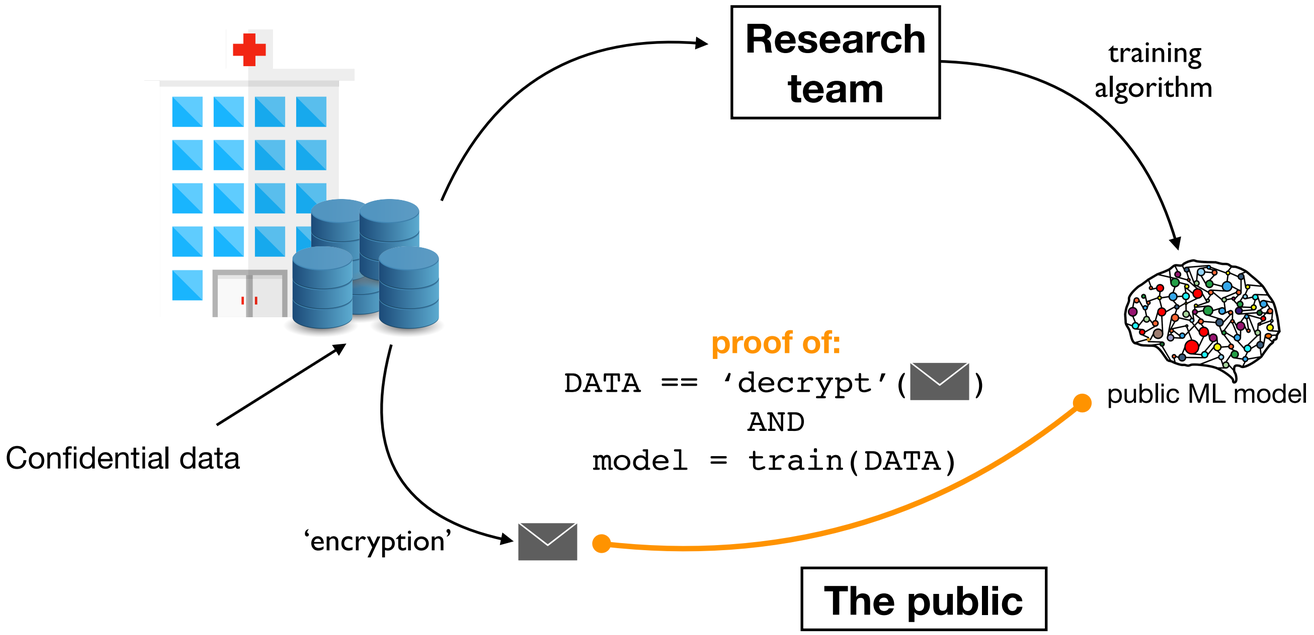
(What's public data ***D****? What's secret* ***s****?)*

Anonymous Credentials



(What’s public data ***D****? What’s secret* ***s****?)*

Verifiable ML (from [WZCPS18])



Goal: we want this ML model to be publicly auditable.

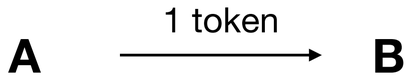
Problem:  can't be released.

Then how can the public check  ?

Some Blockchain-specific Applications of zkSNARKs

Brush-up on Bitcoin

Bitcoin: a protocol to agree on exchange of tokens
(e.g. which tokens are around? Who controls a token?)

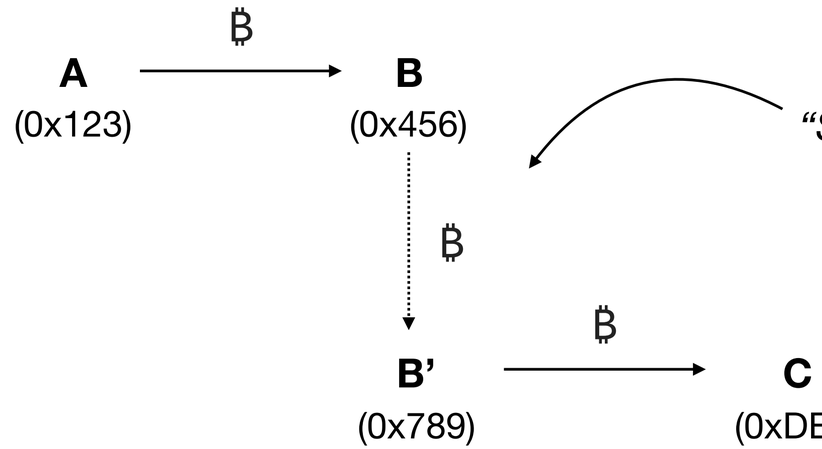


Key Property: If A gives a token to B:

1. A can’t use it again (at least consensus-wise).
2. B can use it.

Little Privacy in Bitcoin

(Attempts of) Pseudo-anonymity in Bitcoin: randomized addresses.



Randomized addresses are not enough!

We can trace back transactions to the actual senders/receivers. [GKRDN17,…]

Improving Privacy in Bitcoin

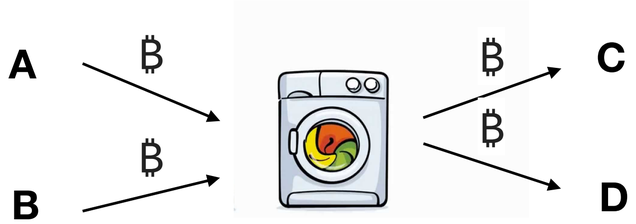
Privacy Issues of Bitcoin:

- All transactions are public (and traceable)
- Side-information (e.g. cookies…)

Possible Solutions:

- Tumblers (aka mixers or mixnets)
- Private crypto-currencies
  (e.g. ZCash)

Our focus



Tumblers

Private Crypto-Currencies

Warm up:
Transactions in Bitcoin

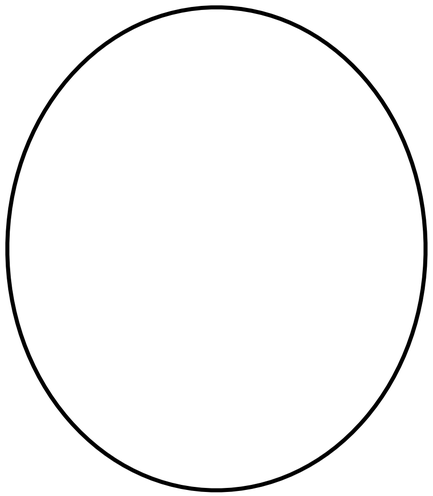
Existing tokens
(Once spent, they go to the right and "produce" a new coin)

Spent tokens
(you can't use them again)

A (oxAAA)

0xred

B (oxZZZ)

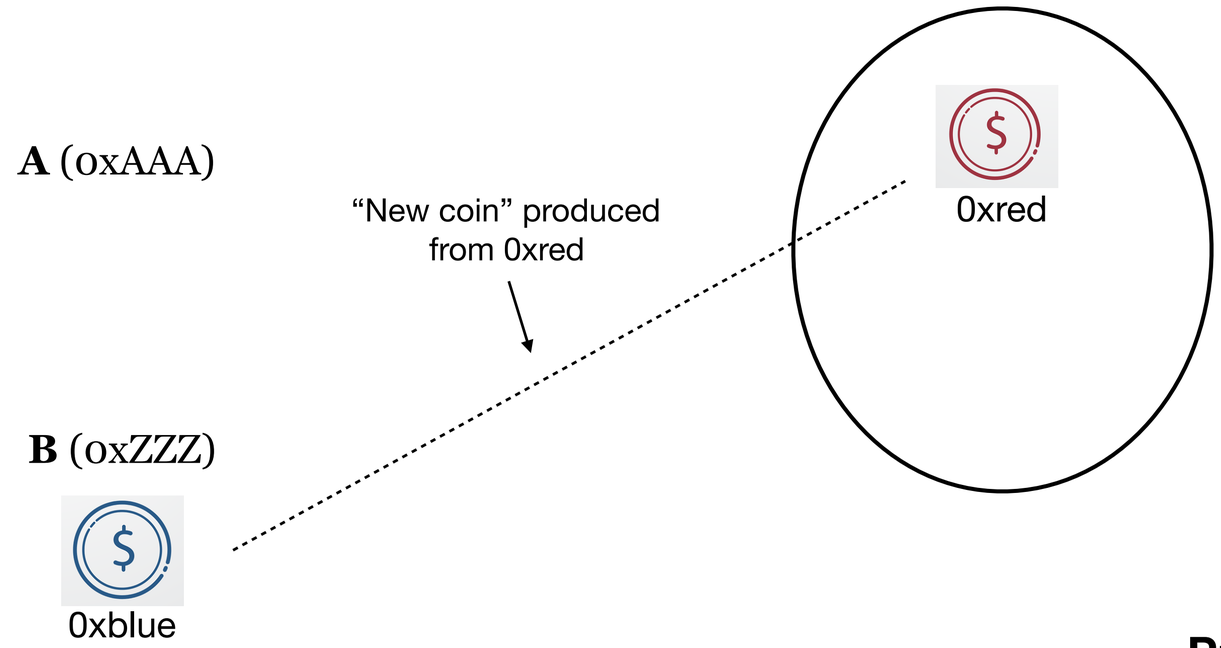


Scenario: A wants to pay **B**

Warm up:
Transactions in Bitcoin

Existing tokens
(Once spent, they go to the right and "produce" a new coin)

Spent tokens
(you can't use them again)



Scenario: A wants to pay **B**

Public transaction reads:
"**A** *makes* $ *spent;*
B *receives it as* $ "

NB: This is all public info!

ZCash: Anonymous Transactions

Traceable transaction in Bitcoin:
"**A** *makes*  *spent;*
B *receives it as*  "

Our Goal:
- **Remove** receiver/sender as public;
- **Keep** validity of transaction.

Roadmap (two steps):

1. Anonymize sets of spent/existing tokens;
2. use zkSNARKs to *prove* we are performing a valid transaction.

Anonymized Spent Tokens

Before: completely public.

Existing tokens

A (oxAAA)

0xred

Spent tokens

0xYellow

Now: somewhat private through hashing.

Existing tokens

A (oxAAA)

H_exist()

0xred

Spent tokens

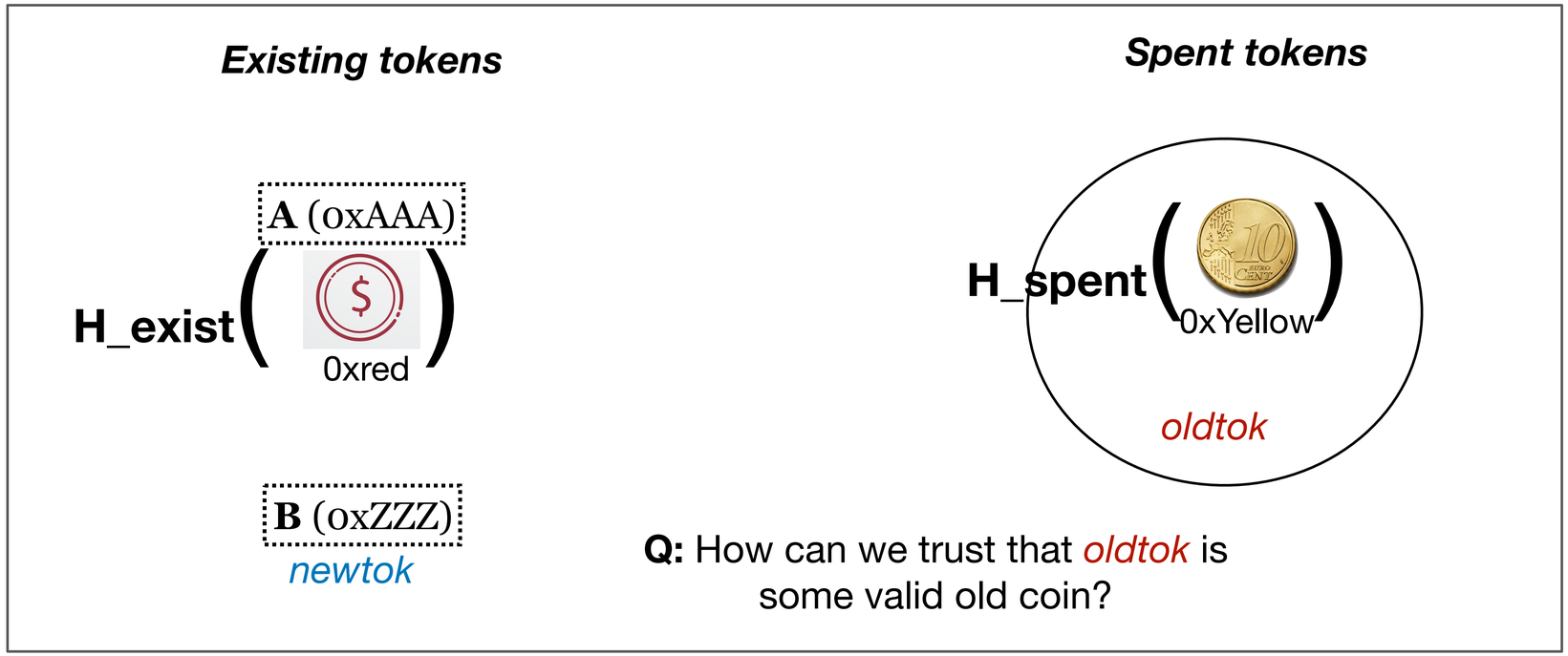
H_spent()

0xYellow

Owner of the coin
is not public any more

To anonymize:
we publish the output of two hash functions
instead of actual coins.

Transactions in ZCash



What A does to spend her coin to B:

- makes new 0xblue and send it (privately) to B
- broadcast (*newtok*, *oldtok*) where

  newtok = H_exist(0xblue)

  oldtok = H_spent(0xred)
- broadcast a zkSNARK proof **π** to validate transaction

Proof π proves that we are spending valid token:

- H_exist(0xred) is in the set of existing tokens
- A controls (0xred)
- *oldtok* == H_spent(0xred)

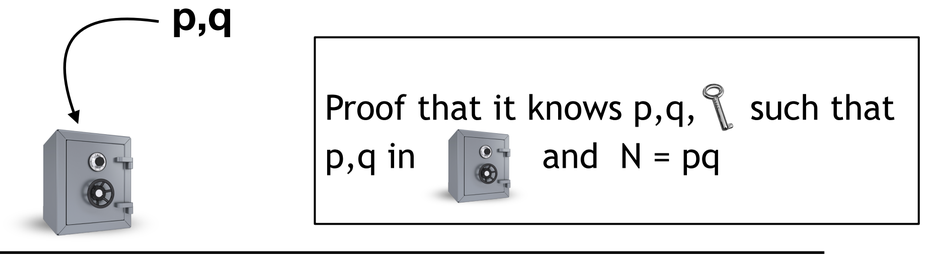
zkSNARKs in ZCash: Summary

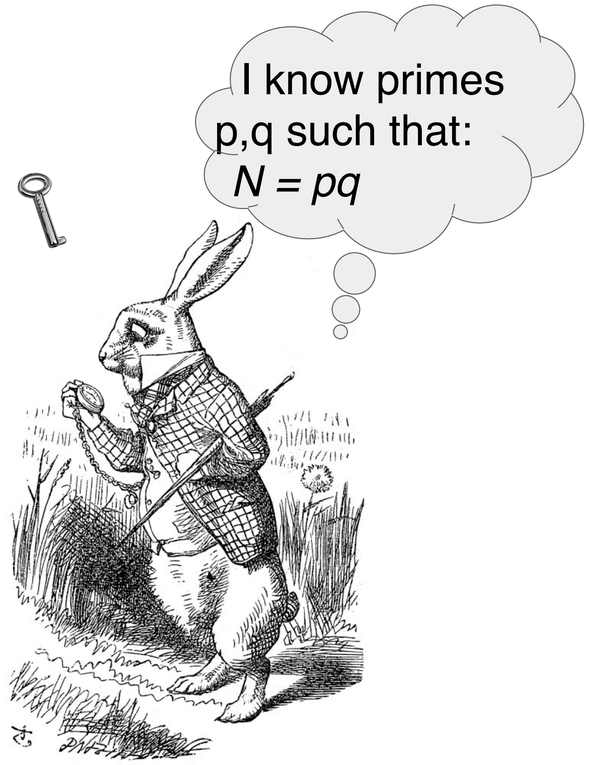
- Bitcoin is not anonymous
- We can make it anonymous by:
  - extending it with anonymized sets
  - using SNARKs to prove transactions are valid without leaking sender/receiver

Fair Exchange of Digital Goods with Bitcoin

Recall the sketchy exchange from before:

Buyer/Verifier





Seller/Prover

Q: What's going to happen next?

And how to ensure **fairness** with no trusted party?

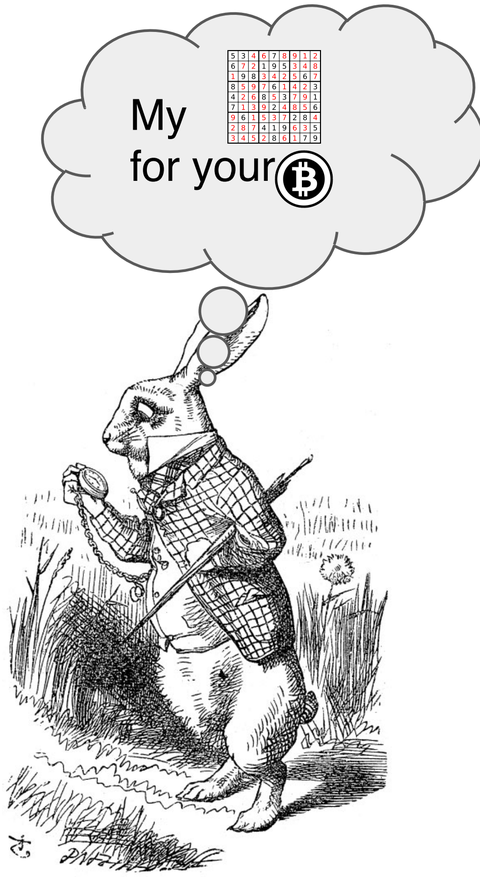
The Buyer

The Seller

How to buy a Sudoku solution [Maxwell16, cGGN17]

The Buyer

The Blockchain



The Seller

How to buy a Sudoku solution [Maxwell16, cGGN17]

My ₿ for your

The Buyer

Buyer gets **iff** **Seller** gets ₿

The Blockchain

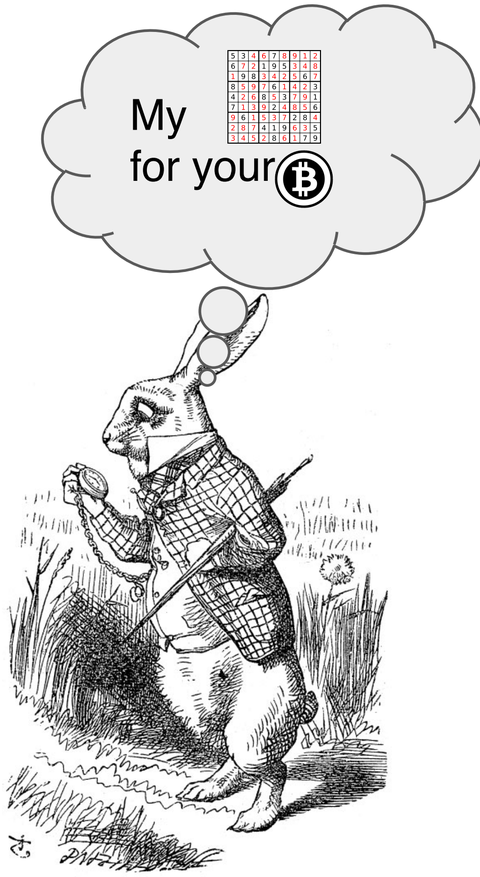
My  for your ₿

The Seller

How to buy a Sudoku solution [Maxwell16, cGGN17]

The Buyer

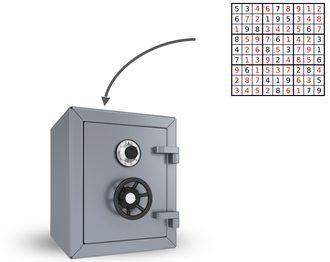
The Blockchain



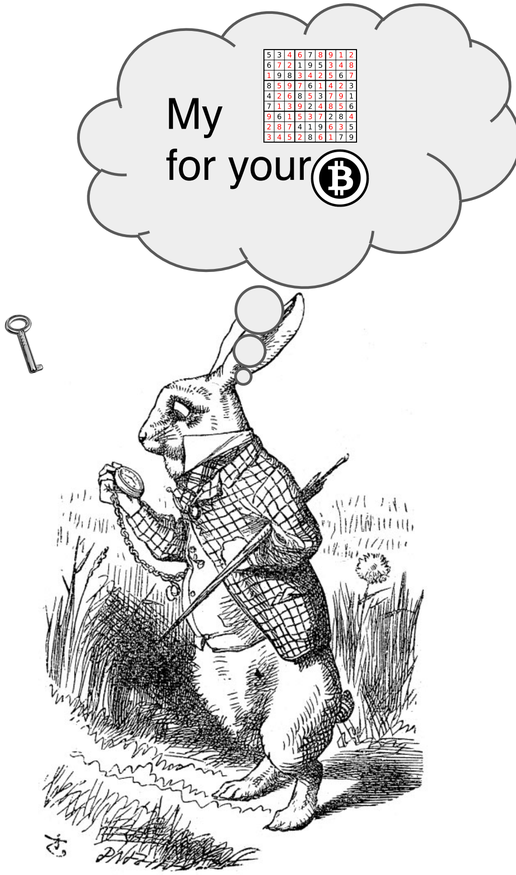
The Seller

How to buy a Sudoku solution [Maxwell16, cGGN17]

The Buyer



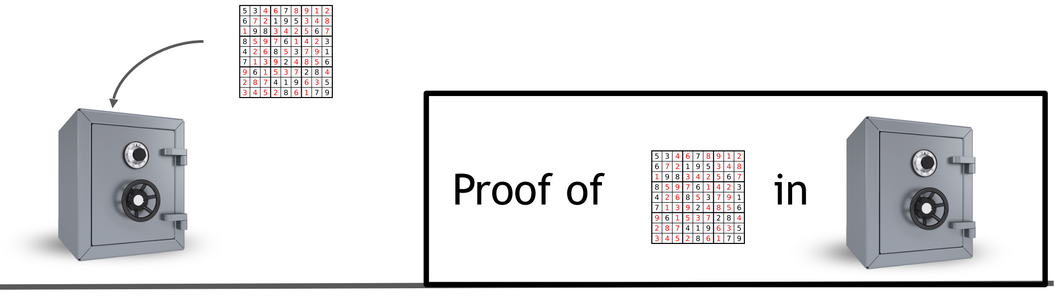
The Blockchain



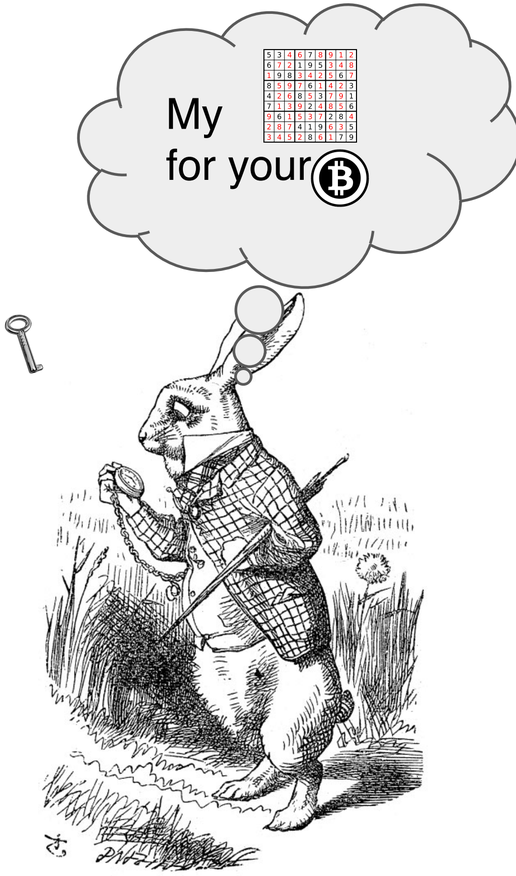
The Seller

How to buy a Sudoku solution [Maxwell16, CGGN17]

The Buyer

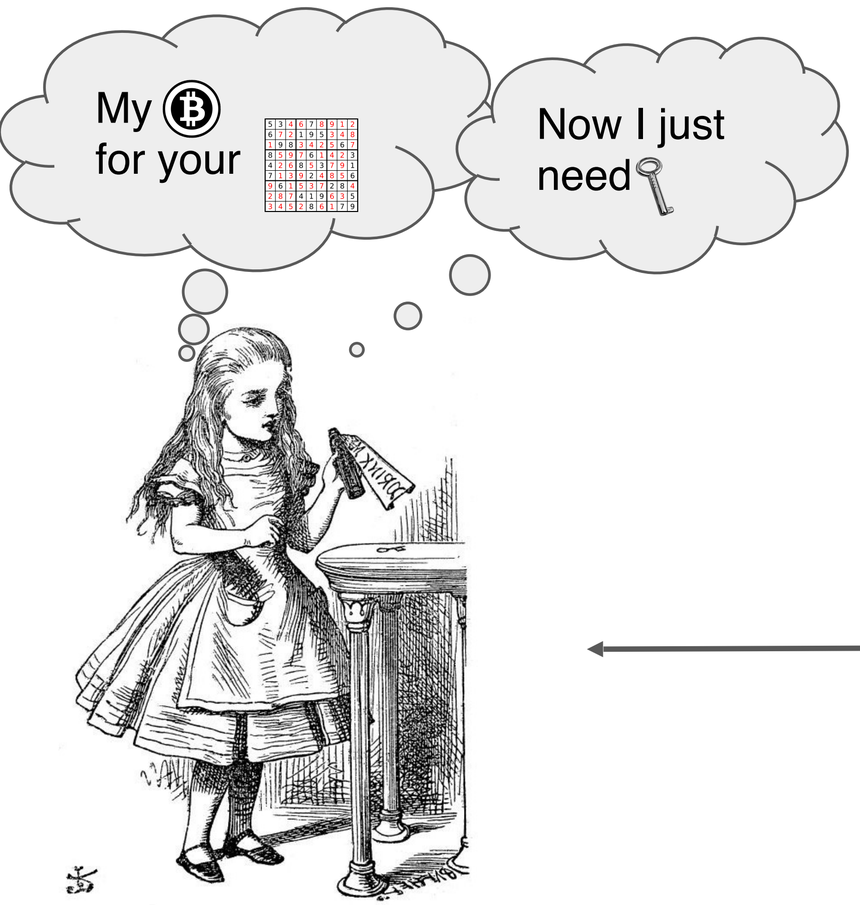


The Blockchain

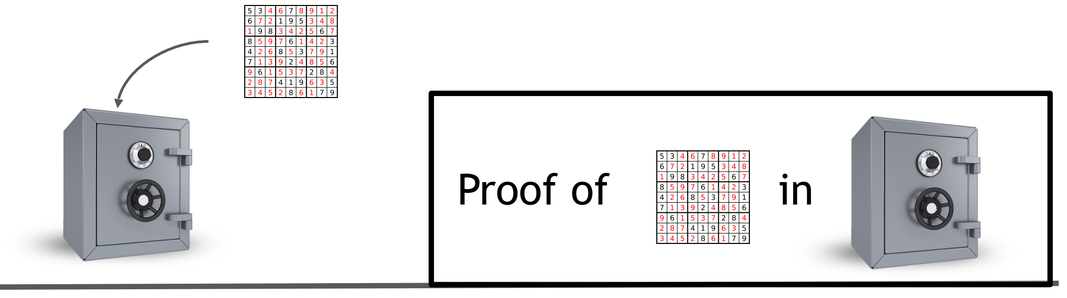


The Seller

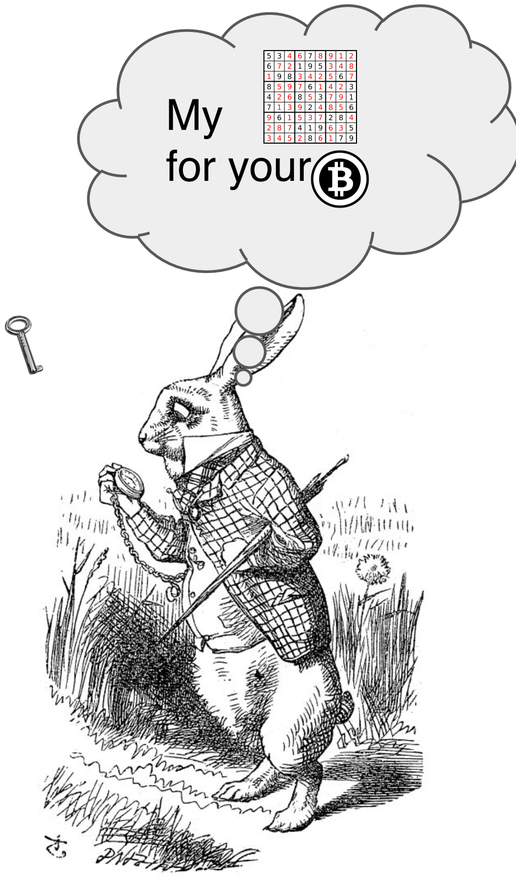
How to buy a Sudoku solution [Maxwell16, CGGN17]



The Buyer



The Blockchain



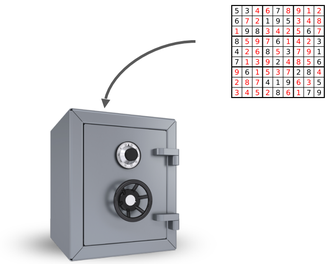
The Seller

How to buy a Sudoku solution [Maxwell16, CGGN17]

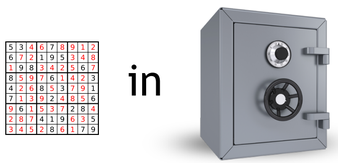
My ₿ for your

Now I just need

The Buyer



Proof of in



"Pay my ₿ to **Seller** *if it provides for "*

The Blockchain

My for your ₿

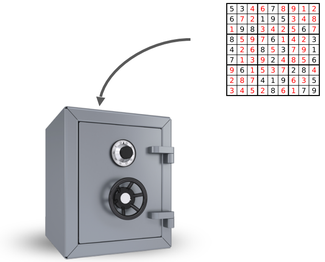
The Seller

How to buy a Sudoku solution [Maxwell16, CGGN17]

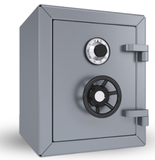
My ₿ for your

Now I just need

The Buyer



Proof of in



“Pay my ₿ Seller *if it provi* ”

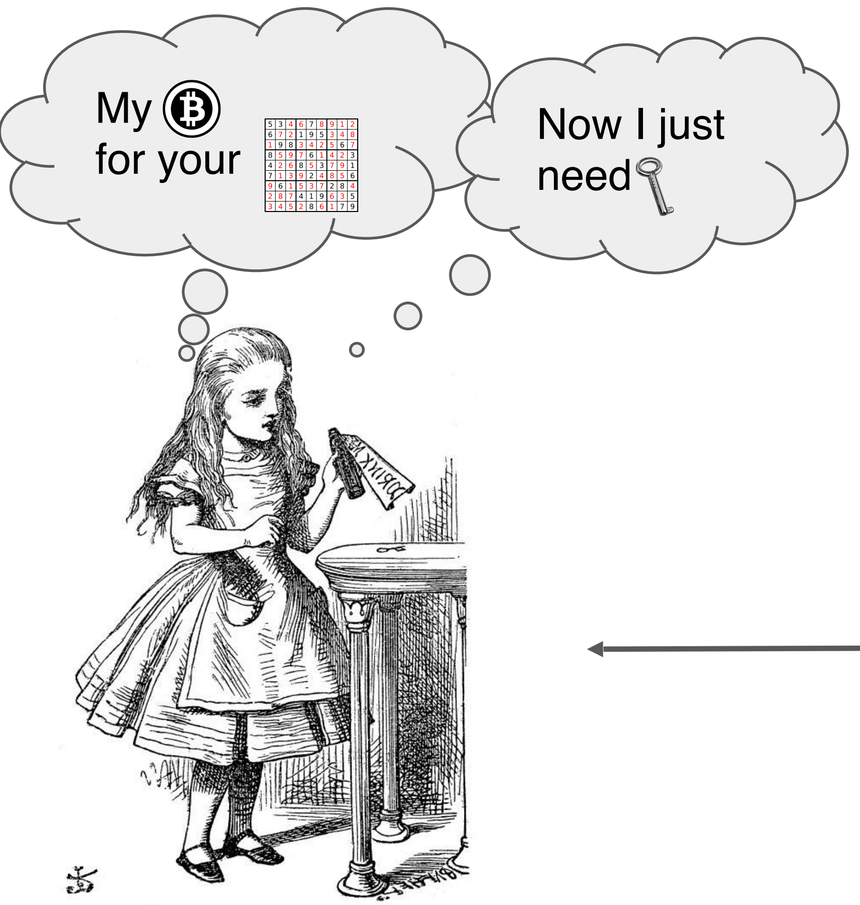
The Blockchain

Can’t do: too complex for Bitcoin transactions

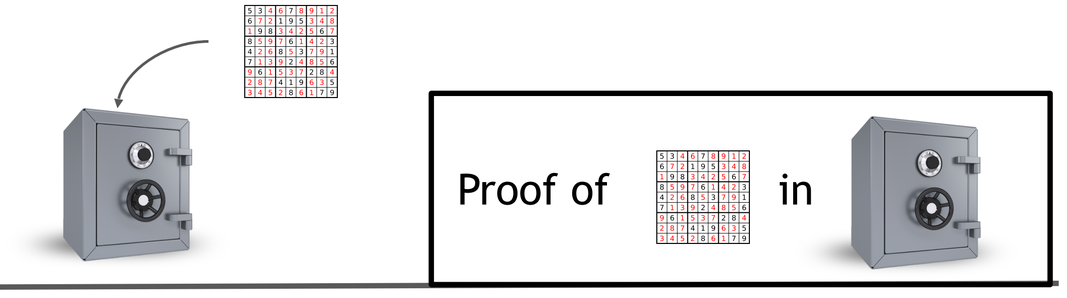
My for your ₿

The Seller

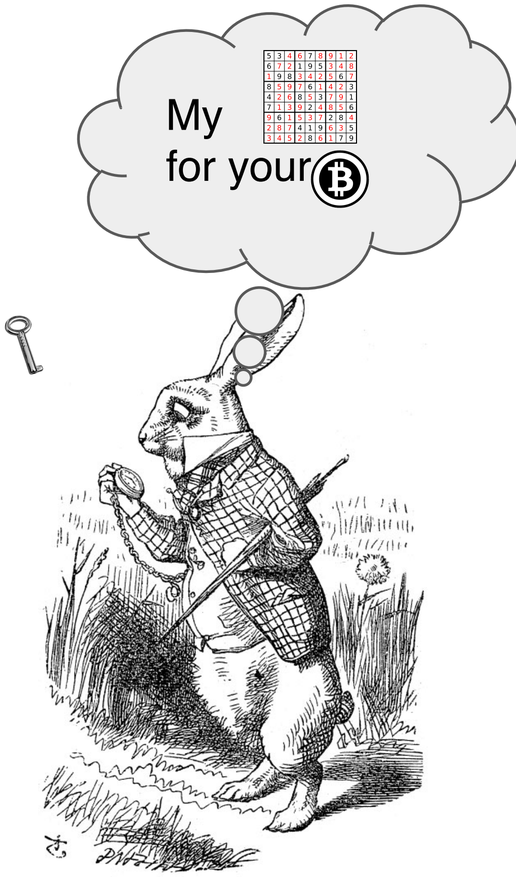
How to buy a Sudoku solution [Maxwell16, CGGN17]



The Buyer



The Blockchain



The Seller

How to buy a Sudoku solution [Maxwell16, CGGN17]

My ₿ for your

Now I just need

The Buyer

$y = SHA_{256}(\ )$

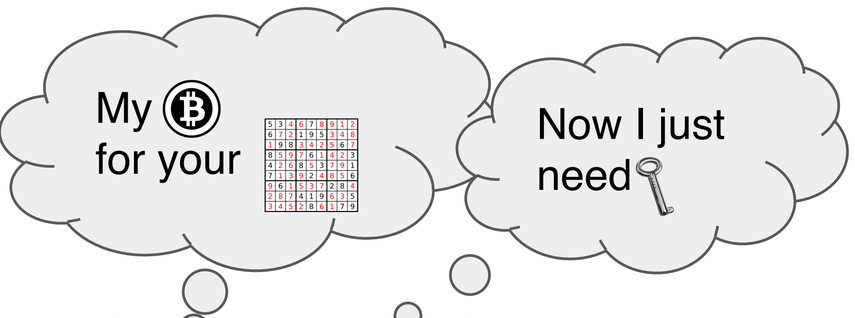
Proof of in

My for your ₿

The Seller

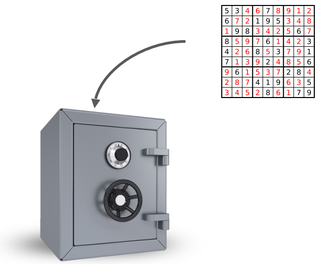
The Blockchain

How to buy a Sudoku solution [Maxwell16, cGGN17]



The Buyer

$y = SHA_{256}(\ \ )$

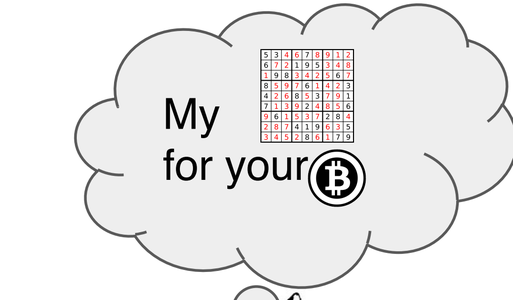


Also extend proof to account for

$SHA_{256}(\ \ )$

Proof of  in 

The Blockchain



The Seller

How to buy a Sudoku solution [Maxwell16, CGGN17]

My ₿ for your

Now I just need

The Buyer

$y = SHA_{256}(\ )$

Also extend proof to account for

$SHA_{256}(\ )$

Proof of in

The Seller

"Pay my ₿ to **Seller** *if it provides* $SHA^{-1}_{256}(y)$*"*

The Blockchain

How to buy a Sudoku solution [Maxwell16, cGGN17]

My ₿ for your

Now I just need

The Buyer

Also extend proof to account for

$SHA_{256}(\ \ )$

$y = SHA_{256}(\ \ )$

Proof of in

"Pay my ₿ to **Seller** *if it provides* $SHA^{-1}_{256}(y)$*"*

The Blockchain

My for your ₿

The Seller

Wrapping up

- With verifiable computation we **do not need trust** a remote computer.
- We can give **proofs** of correctness that are **short** and **quickly verifiable**.
  - Proving often requires a large overhead.
- Beyond Verifiable Computation: we can prove **knowledge of secrets** (zkSNARKs).
- zkSNARKs have **applications** both to society at large and to blockchains.

Thanks! And questions?

For more info:

- visit https://github.com/scipr-lab/libsnark
- or drop me a line at matteo.campanelli@imdea.org