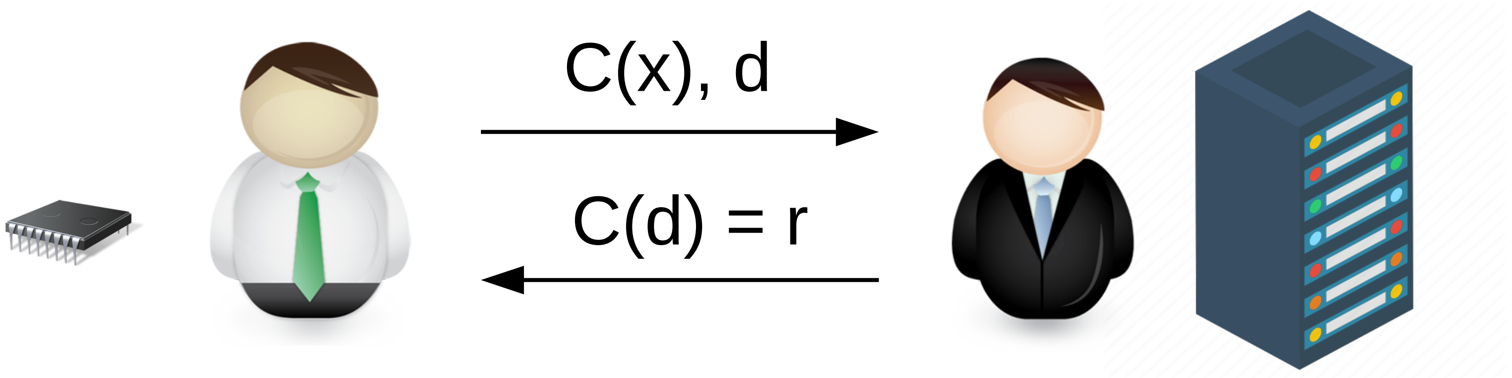
Cloud Computations Integrity Certification Protocol

Evgeny Shishkin
Evgeny Kislitsyn

infotecs®

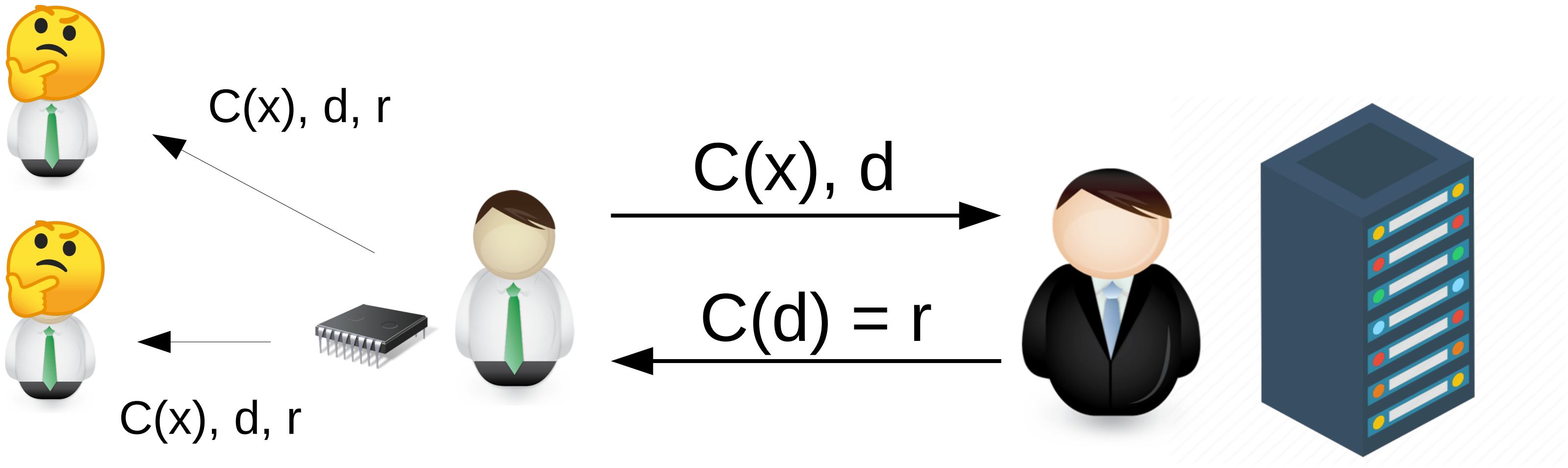
The Problem



- How can we ensure that computation C(d) was performed correctly?

Correctly = semantics of C(x) has not been distored by the computation provider neither intentionally (malicious party), nor by accident (software, hardware bugs).

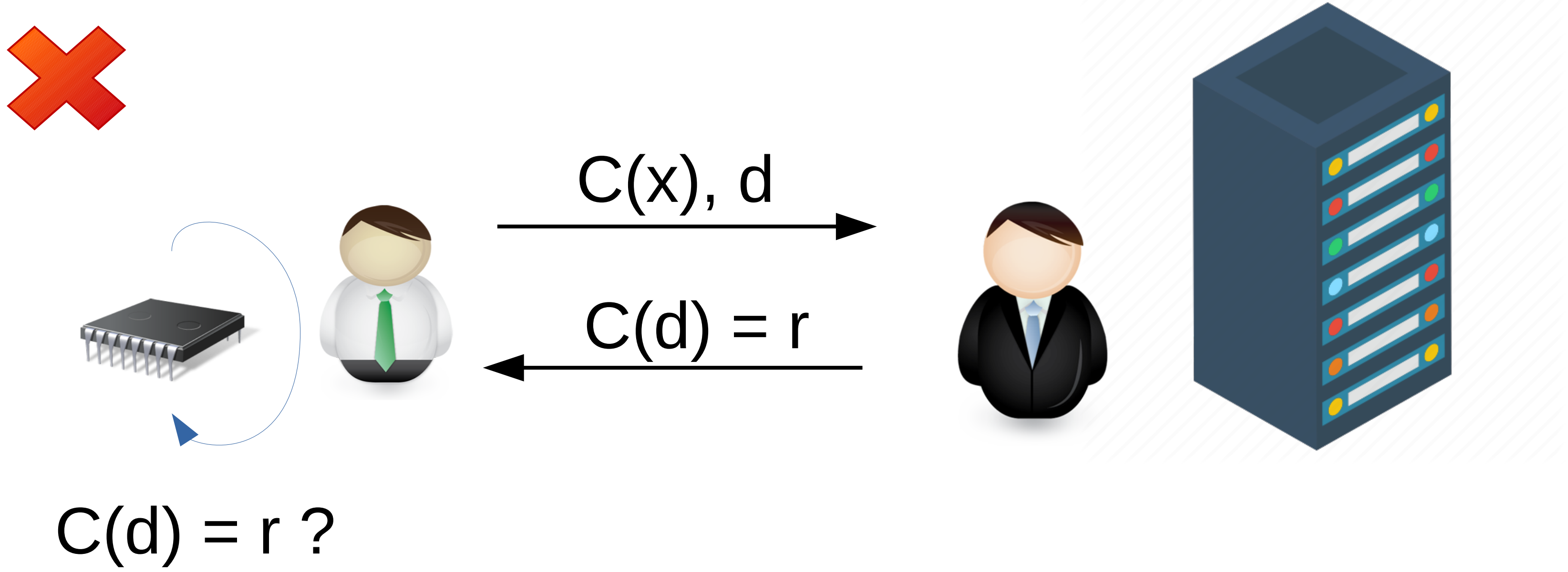
More General Problem



- How can we ensure that C(d) was computed correctly?
- How *to assure other* users that C(d)=r was computed correct and *do it fast*?

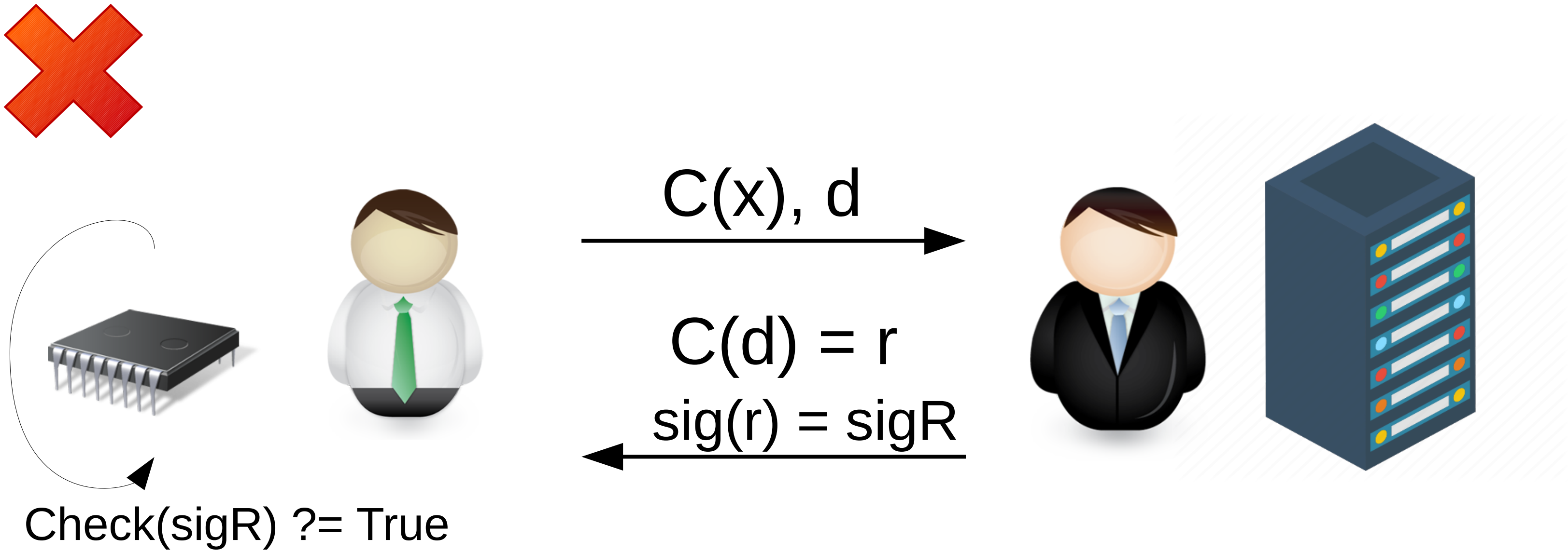
Approaches

Repeat computation



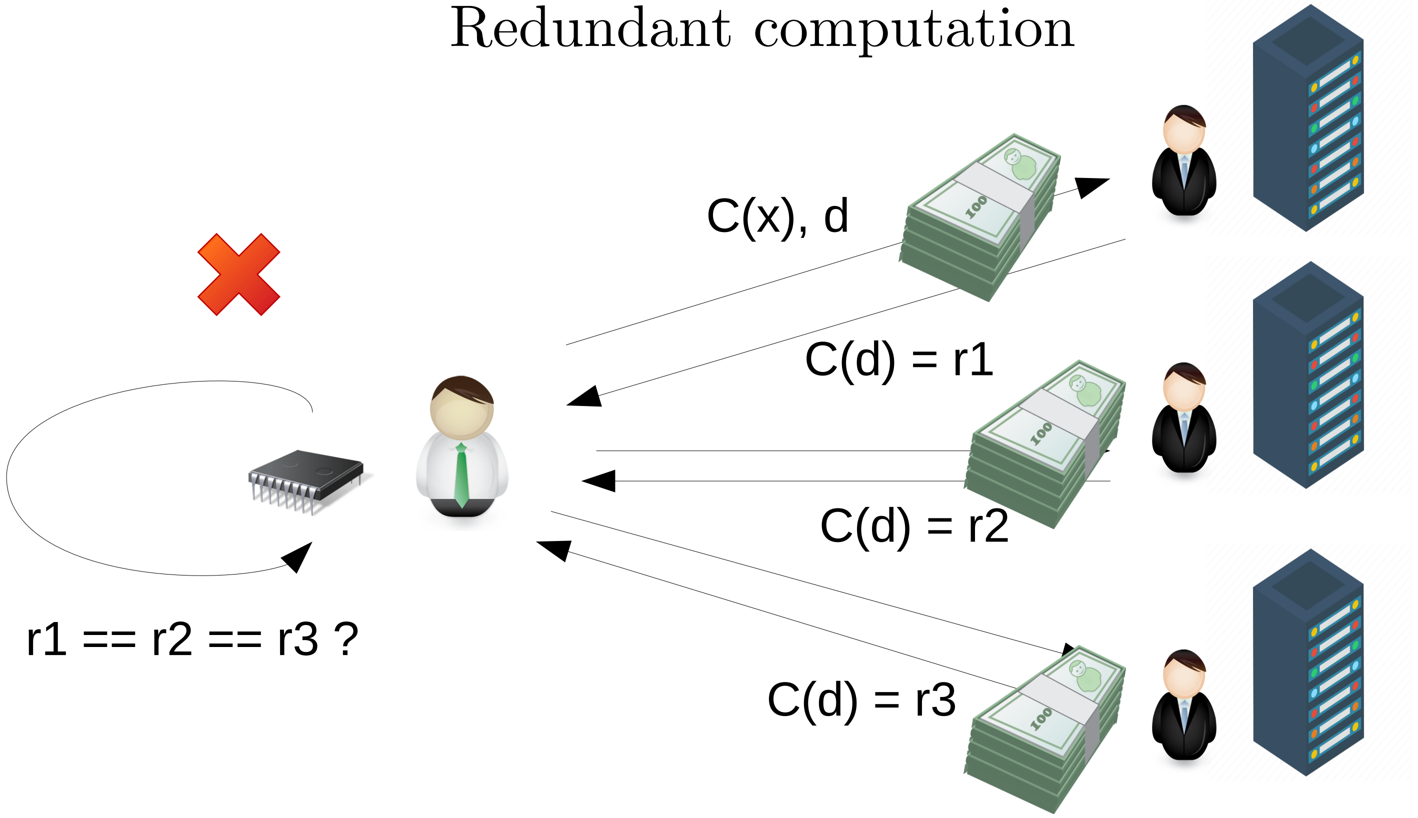
Approaches

Digital Signature =?= Trust



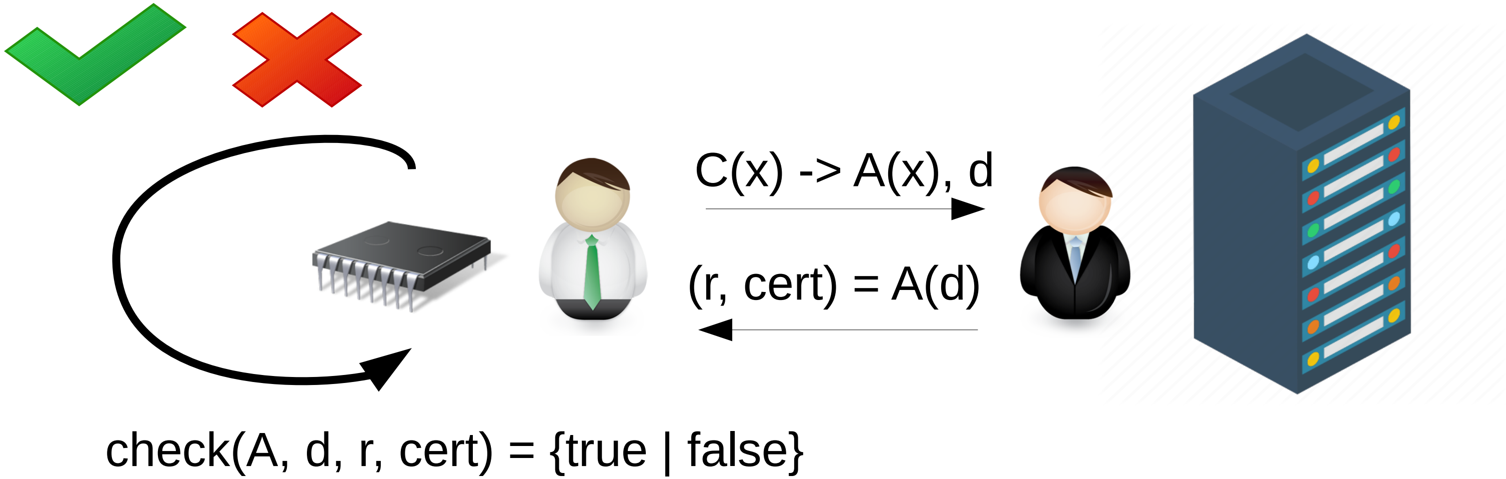
Approaches

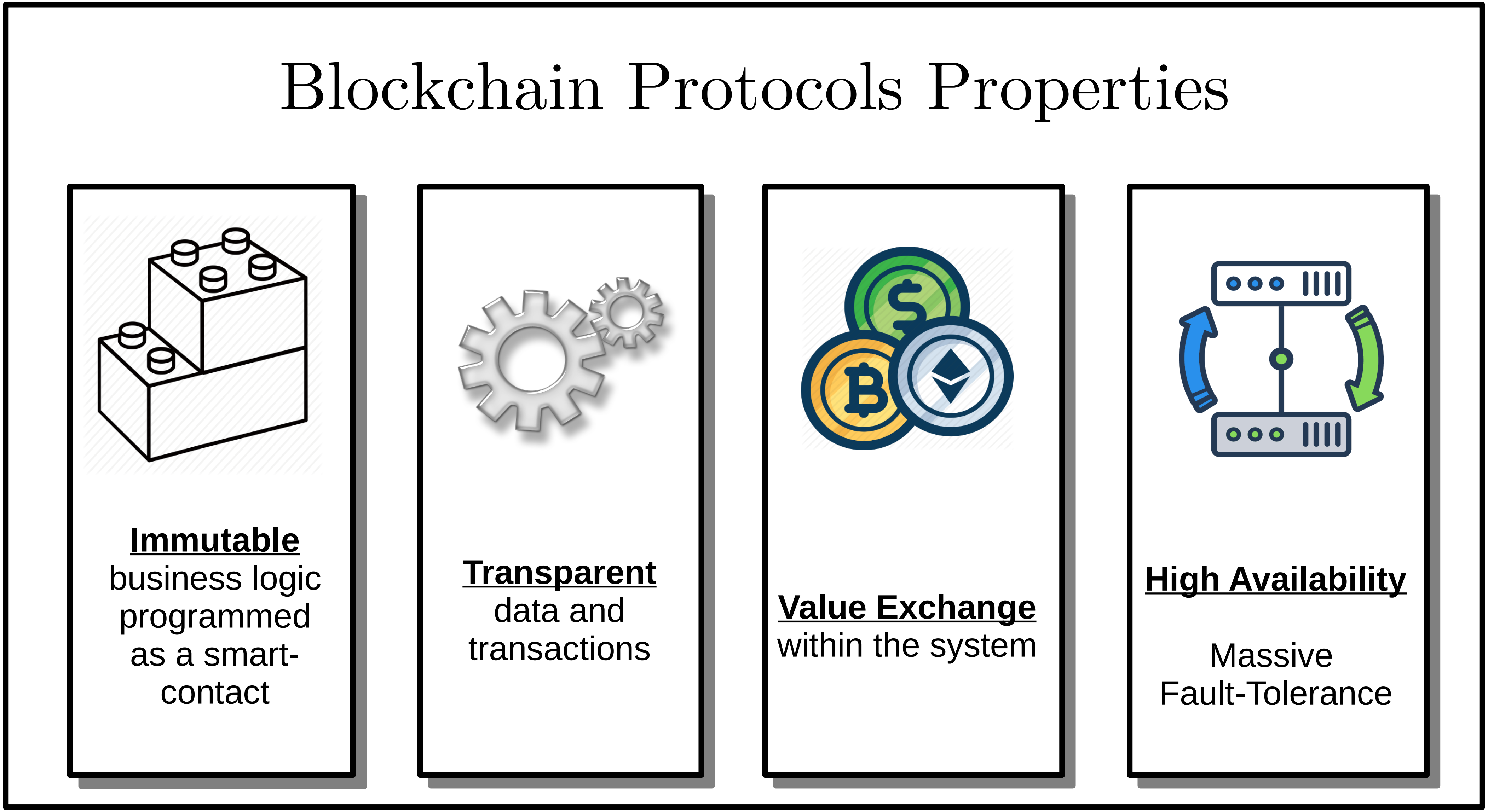
Redundant computation



Approaches

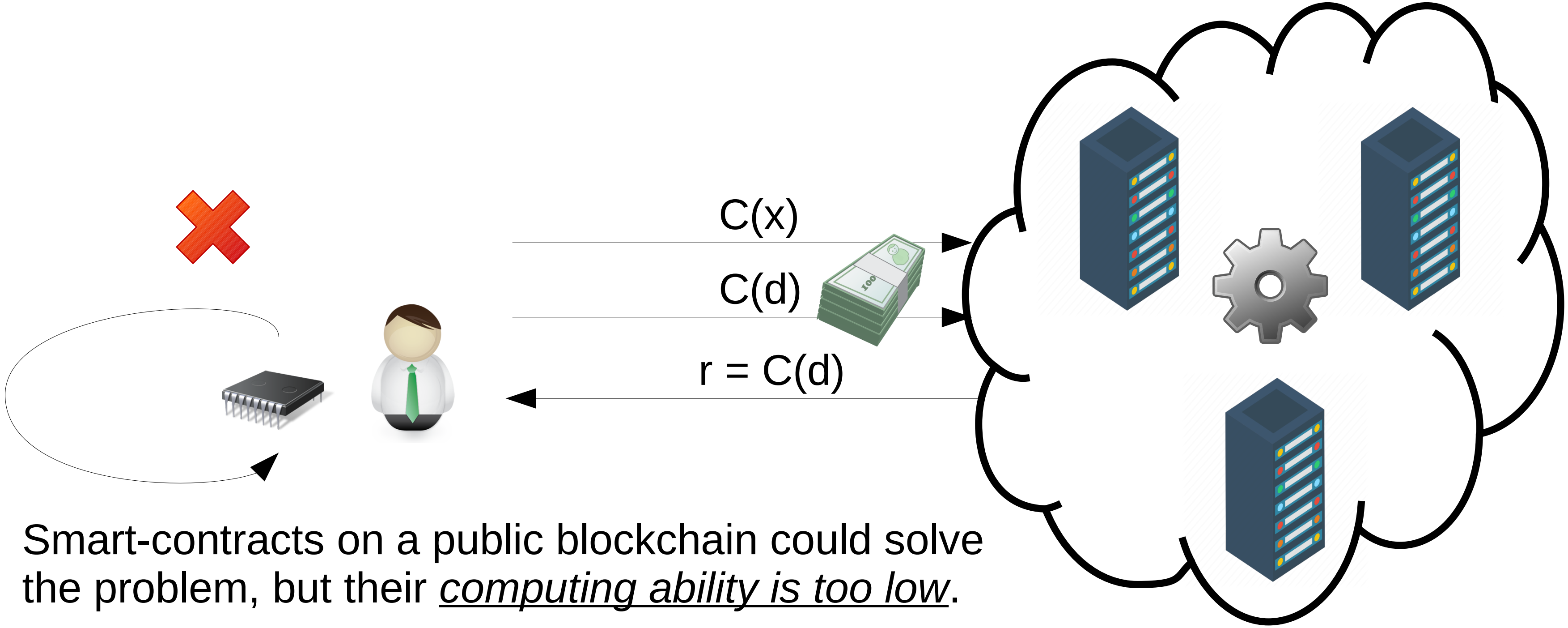
Approaches based on PCP-theorem




Blockchain Protocols Properties
Immutable
business logic programmed as a smart-contact
Transparent
data and transactions
Value Exchange
within the system
High Availability
Massive Fault-Tolerance

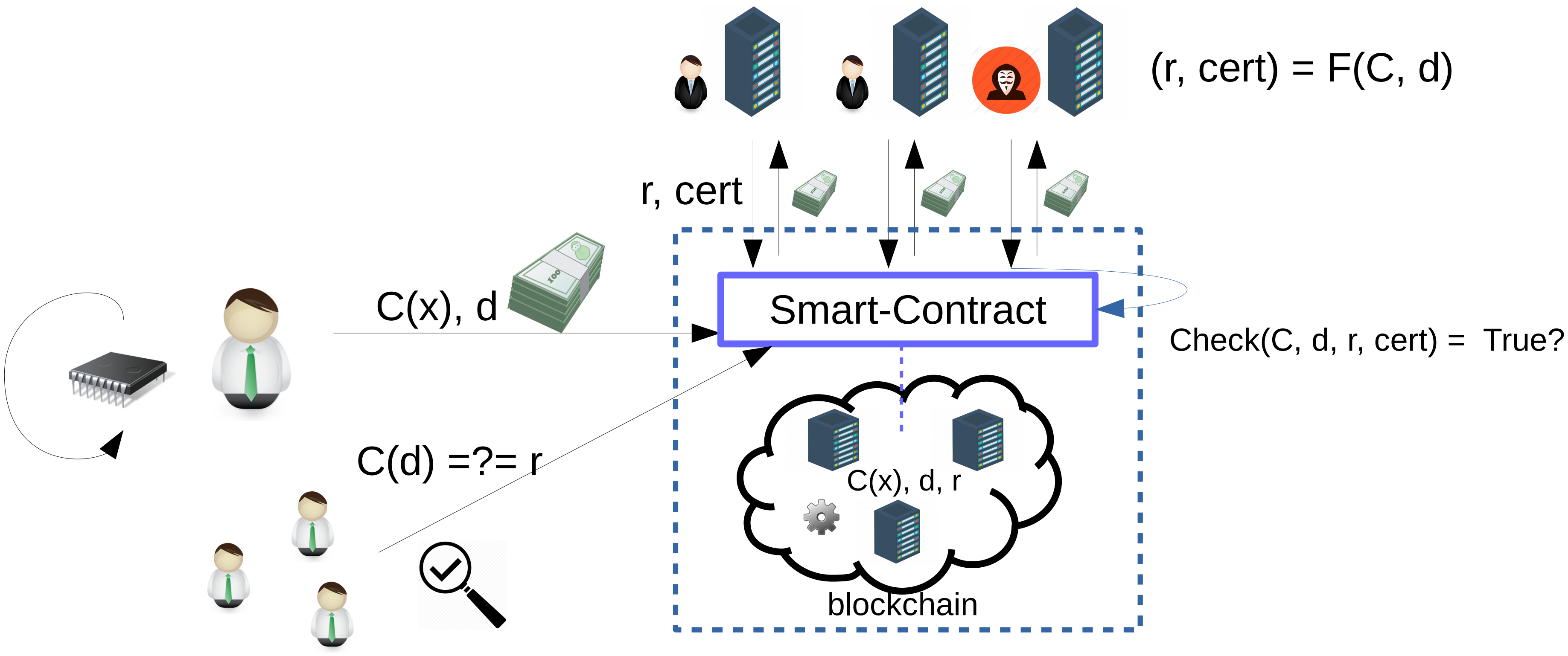
Approaches

Smart-contract on a blockchain

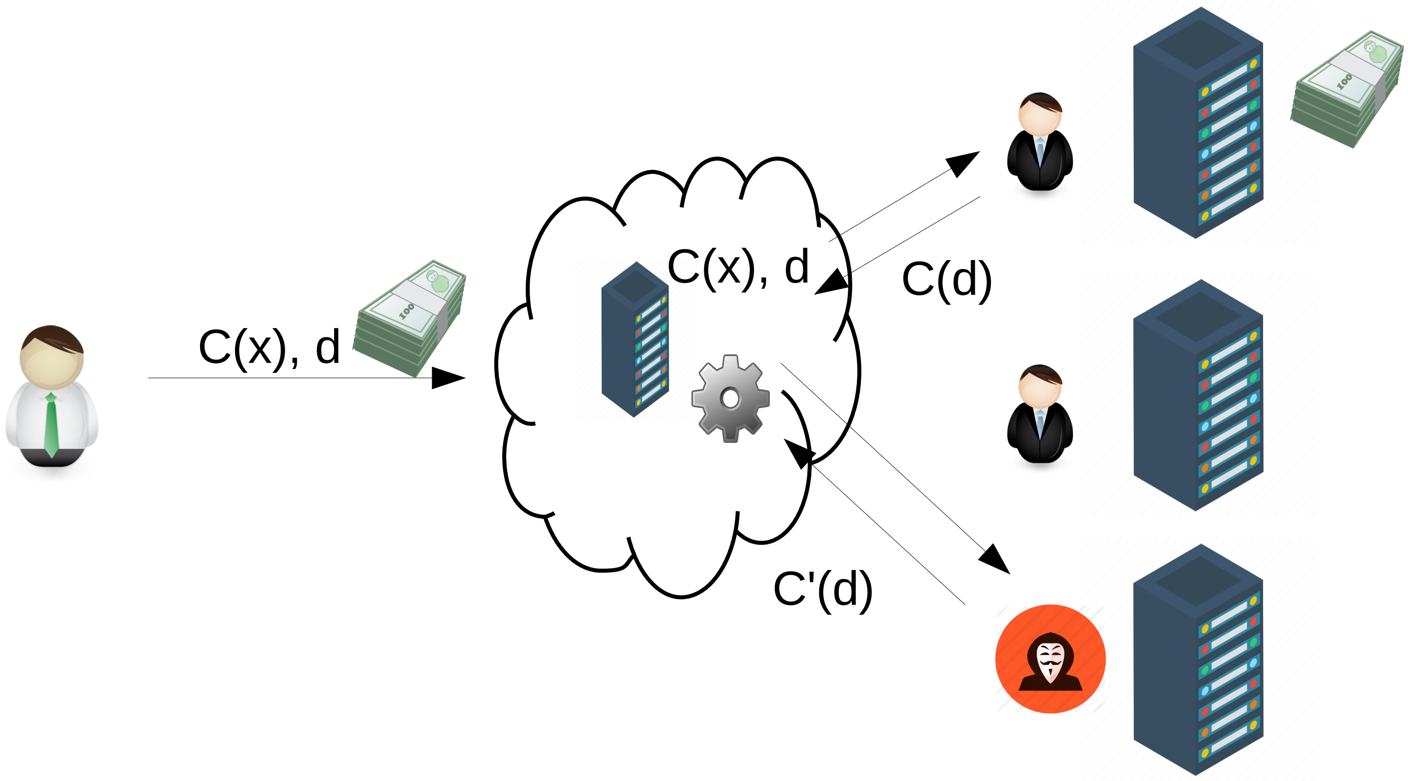


Smart-contracts on a public blockchain could solve the problem, but their *computing ability is too low*.

Blockchain Protocol Sketch



Basic Blockchain Protocol



Trust Model

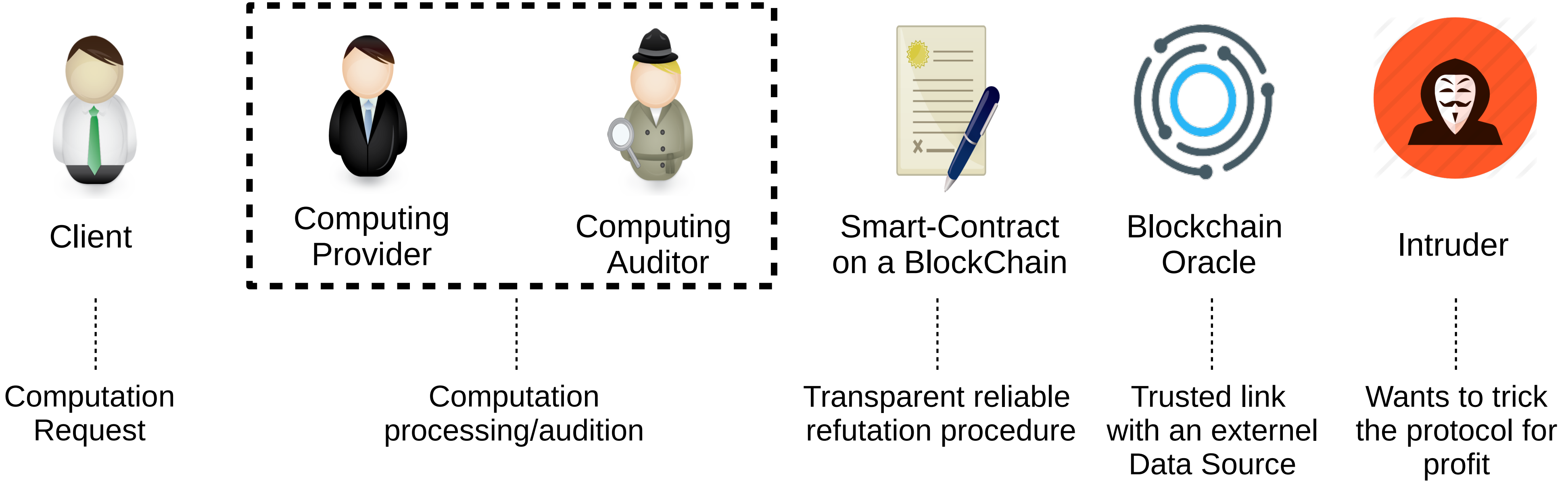
- We do not trust Computation Providers 'per se' (malicious actor, errors in computation, etc..)
- We do trust smart-contract, i.e. can inspect smart-contract logic
- All participants are rational, i.e. everything they do is motivated by an attempt to maximize their profit
- At least, 1 fair/correct motivated computation provider available in the system

Assumptions

- User program $C(x)$ and initial data d is small enough to be placed into the blockchain
- Program result $C(d)$ is small enough to be placed into the blockchain
- Program (i.e. function) $C(x)$ is terminating

SafeComp Protocol

Protocol Participants



Meet SafeComp

Main ideas of the protocol

- User program $C(x)$ is transformed into iterative function form $f(x)$, such that:

$$\mathrm{proj}\ f\ (f\ (\ f\ (\ \dots\ f\ (\ \mathrm{inj}\ (d)\ )\ ..) = C(d)$$

$$\mathrm{proj}\ ((\mathbf{fix}\ f)\ (\mathrm{inj}\ d)) = C(d)$$

- Computation provider calculates values:
  - $c_1 = H(d)$
  - $c_{i + 1} = H(c_i * f(r_i))$
  - $H(x)$ - secure hash-function
- Values $<c_1, c_2, \dots , c_k>$ forms a verifiable certificate

Meet SafeComp

Main ideas of the protocol

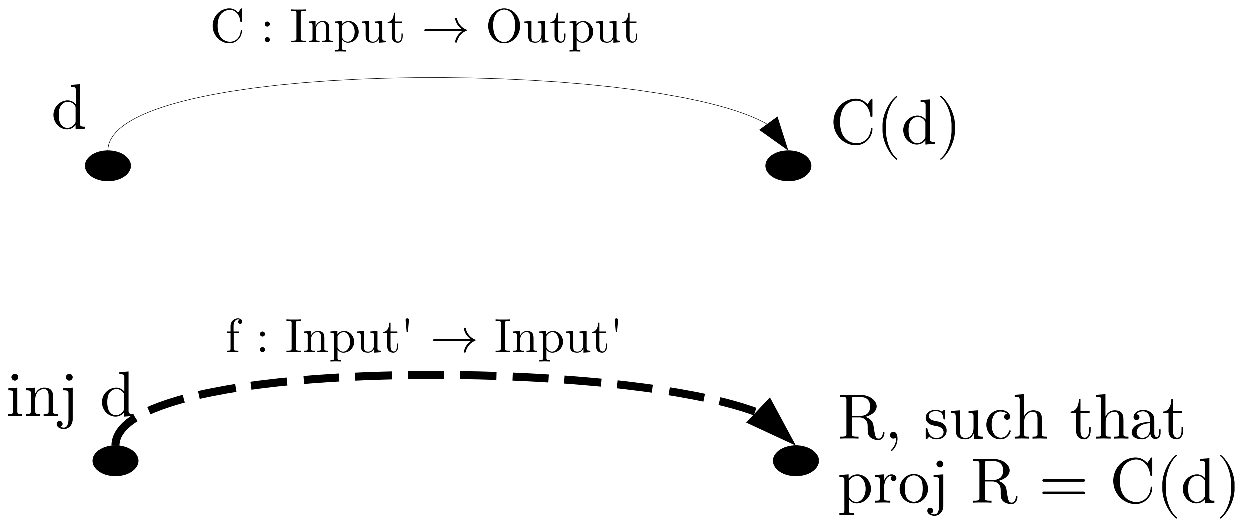
- Computation providers take a problem f(x), the point d , and compute result $r = f(..f(d))$ together with a certificate $cert$
- Provider publishes computed pair $<r, cert>$ together with a guarantee deposit. Such provider is called *the solver*.
- Other computation providers that were late on submitting the answer (called *auditors* in this case), do the check of the result and the certificate
- If error is found, the refutation is sent into the smart-contract. The refutation consist of a triple: $<c\_\{p\text{-}1\}, c\_p, r\_\{p\text{-}1\}>$

Meet SafeComp

Main ideas of the protocol

- Smart contract checks the refutation by performing only a single computation step $c\_p \stackrel{?}{=} H(c\_\{p\text{-}1\} * f(r\_\{p\text{-}1\}))$
- In case of refutation acceptance, the solver is punished by paying the guarantee deposit fee. The problem is moved back to 'published' state awaiting other solutions to be provided.
- At the end, all fair auditors and the final solver get compensated using the total deposit (initial user deposit + all guarantee deposits) of this computation task.

SafeComp Protocol



Function in Iterative Form

Non-iterative form:

```
fact [0] ->
     1;
fact [N] when N > 0 ->
     N * fact [N-1] .
```

fact : Nat → Nat

Iterative form:

```
FactFP [{0, Acc}] ->
     {0, Acc} ;
FactFP [{N, Acc}] when N > 0 ->
     {N - 1, Acc * N} .
```

factFP : Nat * Nat → Nat * Nat
inj(n) = { n, 1 };
proj({n, m}) = m

\forall n . fact (n) == proj((fix factFP) (inj n))

Function in Iterative Form

Non-iterative form:

```
C[1, _] -> 1;
C[_, 0] -> 1;
C[N, N] -> 1;
C[N, M] -> C[N-1, M] + C[N-1,M-1].
```

C : Nat * Nat → Nat

Iterative form:

```
Cfp[{[], Acc}] -> {[], Acc};
Cfp[{[{1,_} | T], Acc}] -> {T,1 + Acc};
Cfp[{[{0,_} | T], Acc}] -> {T 1 + Acc};
Cfp[{[{N,N} | T], Acc}] -> {T,1 + Acc};
Cfp[{[{N,M} | T], Acc}] ->
    {[{N-1,M} | [{N-1,M-1} | T]], Acc}.
```

Cfp : list(Nat) * Nat →
list(Nat) * Nat

inj({n, m}) = {[{n, m}], 0};
proj({_, acc}) = acc

Related Works

truebit

https://truebit.io

Evgeniy Shishkin
Senior researcher

JSC «InfoTeCS»
127287, Moscow, Stariy Petrovsko-Razumovskiy proezd, 1/23, bld. 1

infotecs®

+7 (495) 737 61 92 (ext.4726)
evgeny.shishkin@infotecs.ru

https://unboxedtype.bitbucket.io