Multiple Bugs in Multi-Party Computation: Breaking Cryptocurrency's Strongest Wallets

Omer Shlomovits

ZenGo

JP Aumasson

KUDELSKI SECURITY

TAURUS

The speakers

Designed enterprise wallets used by banks and exchanges

Audited the security of several threshold crypto and MPC products

Omer

Co-founder of ZenGo

Co-founder of MPC Alliance

Israel

@OmerShlomovits

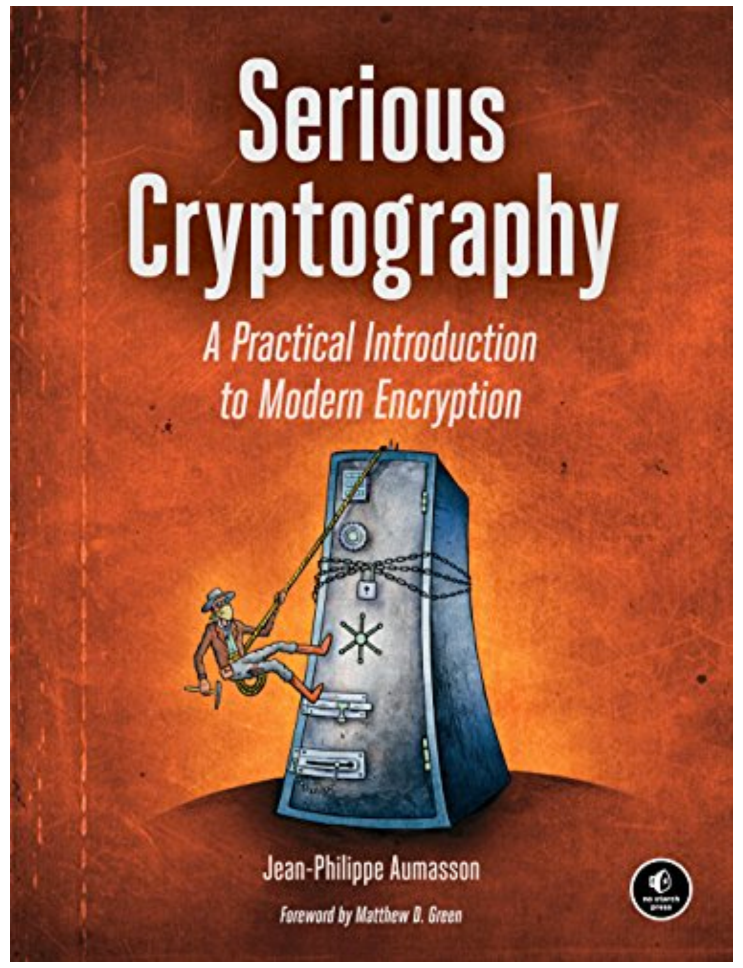
JP

VP technology at Kudelski Security

Co-founder of Taurus Group

Switzerland

@veorq



Agenda

- **Basic notions**: wallet, multi-party computation, threshold signature
- **Crypto building blocks**: secret-sharing, commitment, etc.
- **New attacks** to compromise private keys or leak key bits
- **Recommendations** to avoid such catastrophes

Basic notions

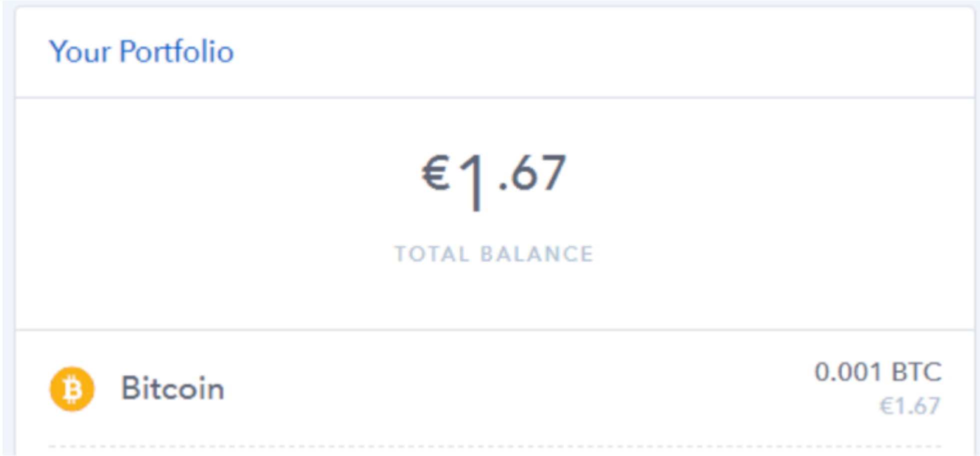
What’s a wallet?

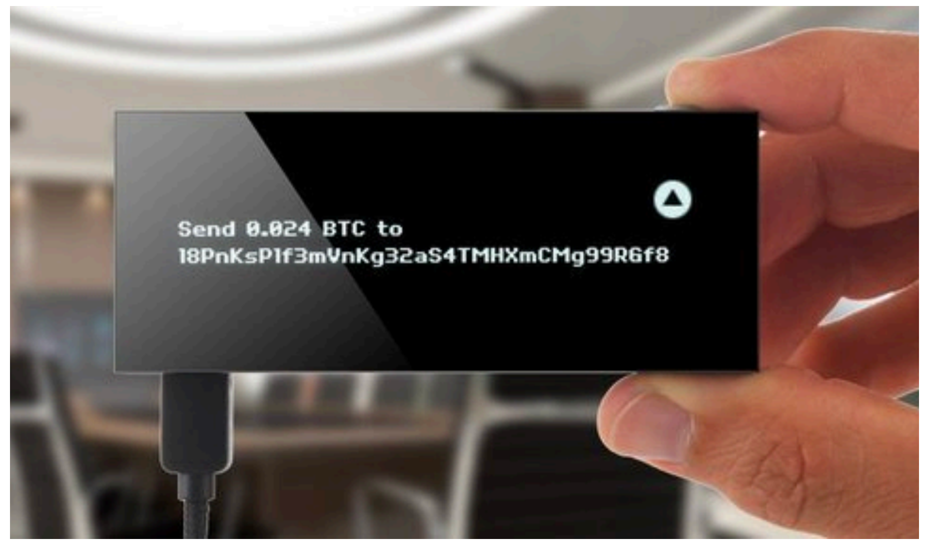
System to **store and manage keys** associated to digital asset accounts

- These keys are required to sign transactions, thus spend money
- Usually a seed is stored, from which keys are derived as per BIP32/44
- The public key or the address can be made public

Different types of wallets for individual users:

Online, mobile, desktop, paper, hardware





Enterprise wallets

Used by **institutional actors** rather than individuals: crypto exchanges, private banks, crypto banks, investment funds, etc. **Different needs** than individual wallets:

- **Higher security and privacy**: larger amounts, regulations and audits
- **Management features**: transactions settlement, proof of reserve, etc.
- **Integration** in financial IT and processes (e.g. banking network)
- **Key lifecycle assurance**, from key ceremony to BCP/DRP
- **Hot vs. cold** systems, to manage liquidity and minimize risk
- **High availability**, to work all the time

This talk is about technologies used for enterprise wallets

Distributing trust

Enterprise wallets need to distribute trust (in software, hardware, humans), to **avoid a single point of failure**, minimize the risk, and be compliant with certain regulatory frameworks:

- Dedicated hardware, strict processes, AAA layers can work, but not always suitable (environment, cost, etc.)
- Multi-signatures can help, but require multiple keys, and tend to work differently for different cryptocurrencies
- Funds can be distributed over per-account key, per-asset seed, depending on the pooling model
- Cold storage systems often work with one or few fixed keys; processes can be insufficient when $100Ms are under custody

Multi-party computation (MPC)

An approach to distribute trust, particularly interesting when only software components are available, backed by established crypto research:

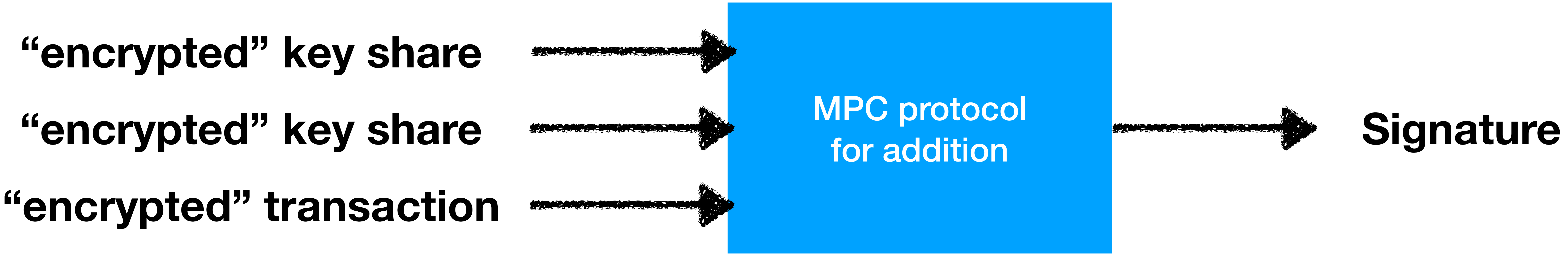
MPC components received “encrypted” inputs, and only learns the output:

For a wallet application, what are the inputs and the MPC functionality?

Multi-party computation (MPC)

An approach to distribute trust, particularly interesting when only software components are available, backed by established crypto research:

MPC components received “encrypted” inputs, and only learns the output:

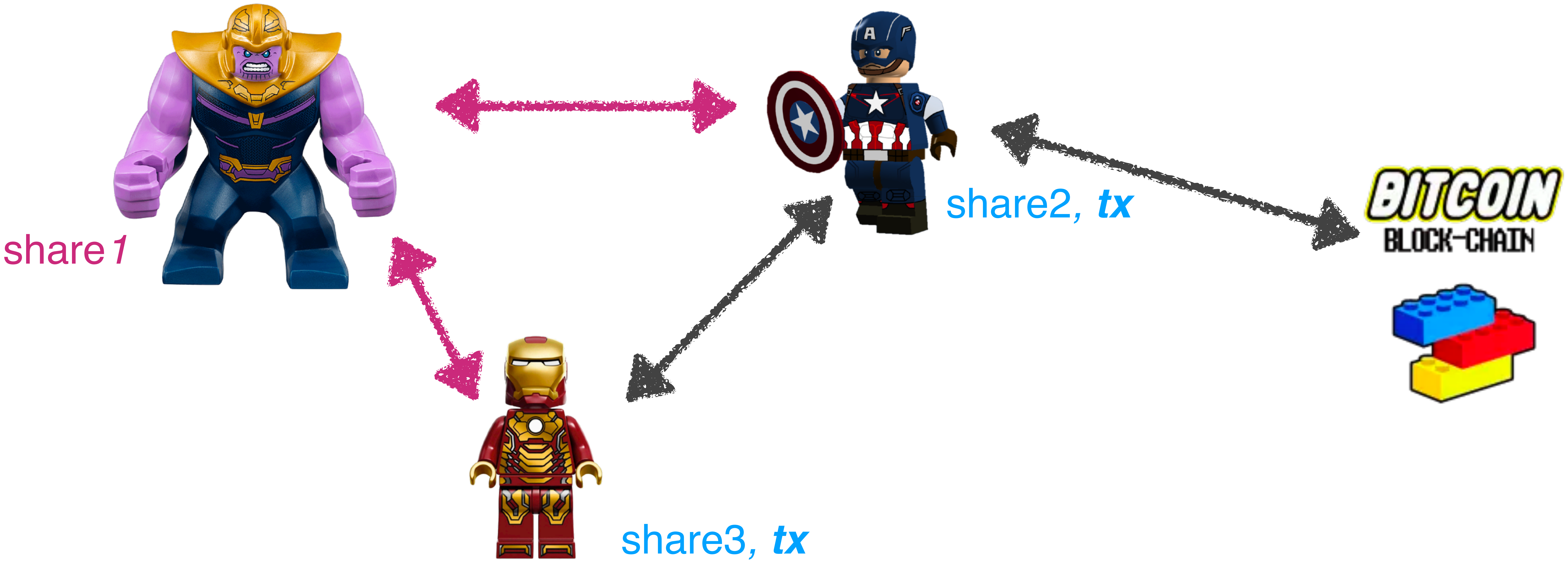


How to efficiently manage quorum-based protocols (m-of-n participants)?

Threshold signatures (TSS)

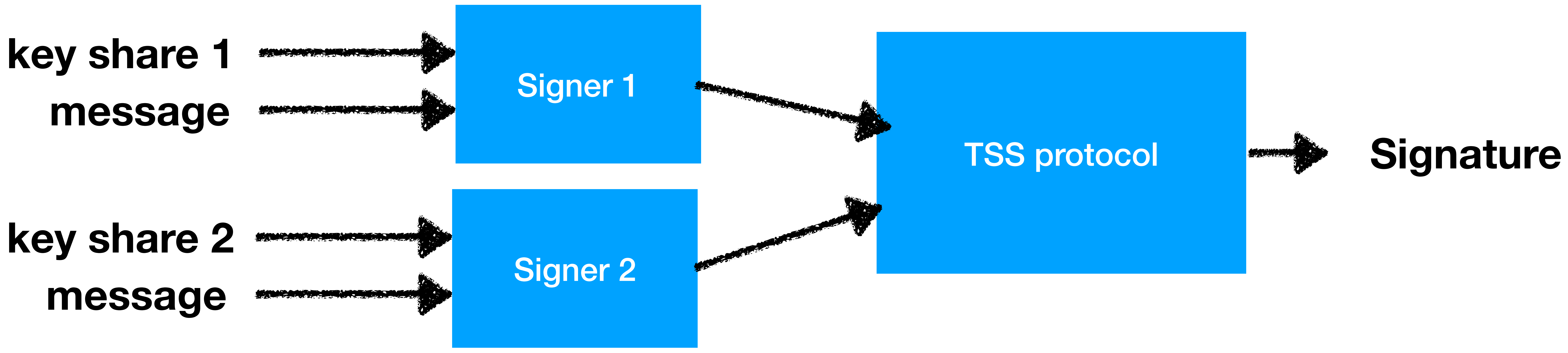
Goal: Distribute signing power among multiple parties, to prevent a single signer (or a small collusion of signers) to issue a signature

A (*t*, *n*) scheme allows of any set of *t* parties to sign, among *n*>*t* possible signers holding distinct shares and common parameters



Threshold signatures (TSS)

Special case of MPC, based on research in **threshold cryptography**

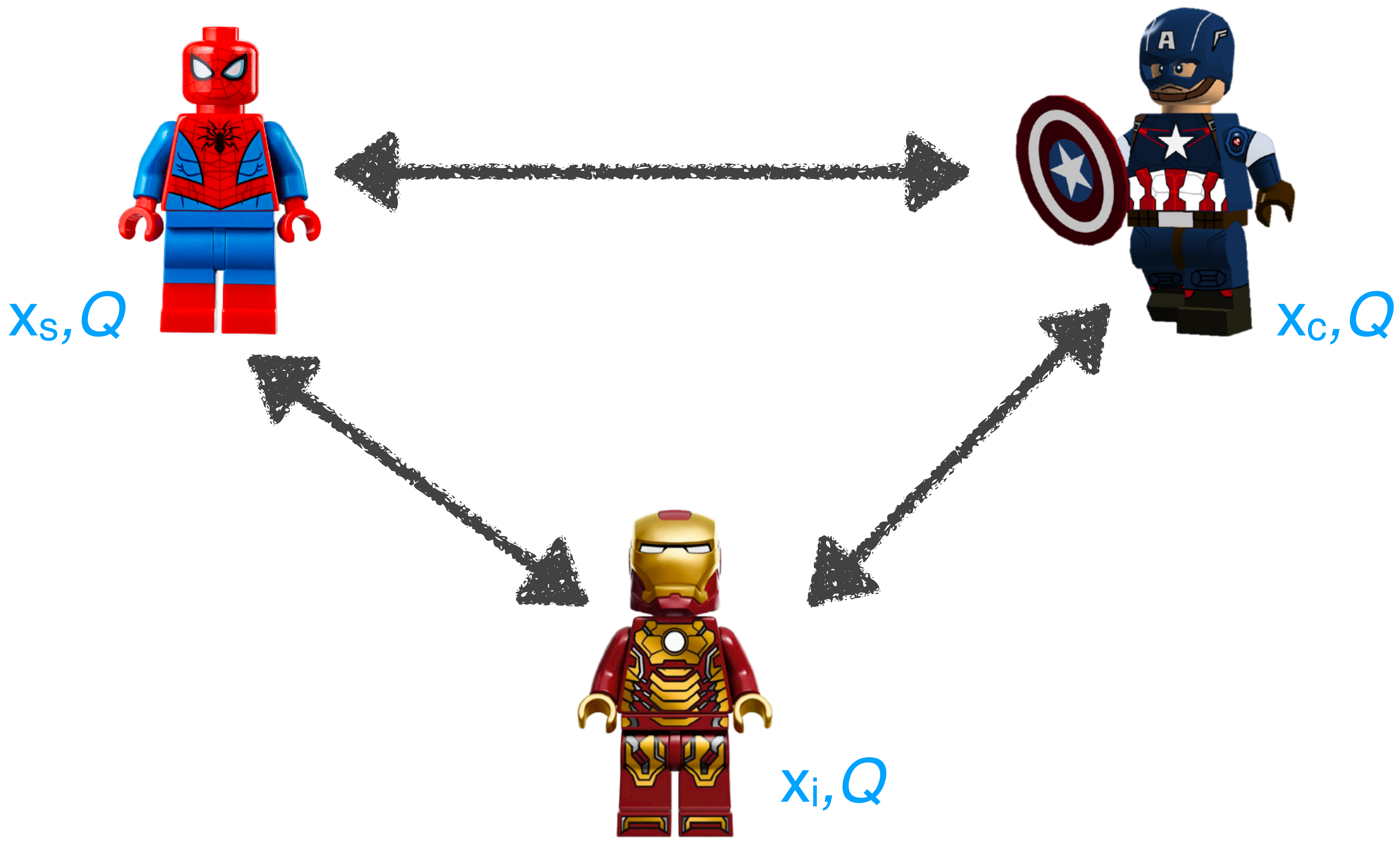


How to generate the shared key? Trust a server then split it and share it?

Distributed key generation

Or how to generate keys to TSS without a single trusted dealer, by distributing the computation in a verifiable way

A **protocol**, based on agreed-upon TSS parameters



Where are MPC and TSS used?

Common use case is **cryptocurrency exchanges**:

- Cold storage with $100Ms stored
- Used to **distribute trust** among multiple parties/locations/infrastructures

For example, TSS may include shares on smart cards, cloud HSMs, and VMs.

MPC/TSS find use cases in software-only deployments, for example to mitigate the risk of using a mobile phone to store key share

Crypto building blocks

2. A Simple (k, n) Threshold Scheme

Our scheme is based on polynomial[1] interpolation: given k points in the 2-dimensional plane $(x_1, y_1), \ldots,$ (x_k, y_k) with distinct x_i's, there is one and only one polynomial $q(x)$ of degree $k-1$ such that $q(x_i) = y_i$ for all i. Without loss of generality, we can assume that the data D is (or can be made) a number. To divide it into pieces D_i, we pick a random $k-1$ degree polynomial $q(x) = a_0 + a_1 x + \ldots a_{k-1} x^{k-1}$ in which $a_0 = D$, and evaluate:

$$D_1 = q(1), \ldots, D_i = q(i), \ldots, D_n = q(n).$$

Digital signatures

Signatures schemes to support are those used to **sign transactions**

Of 2 main types, using elliptic-curve crypto:

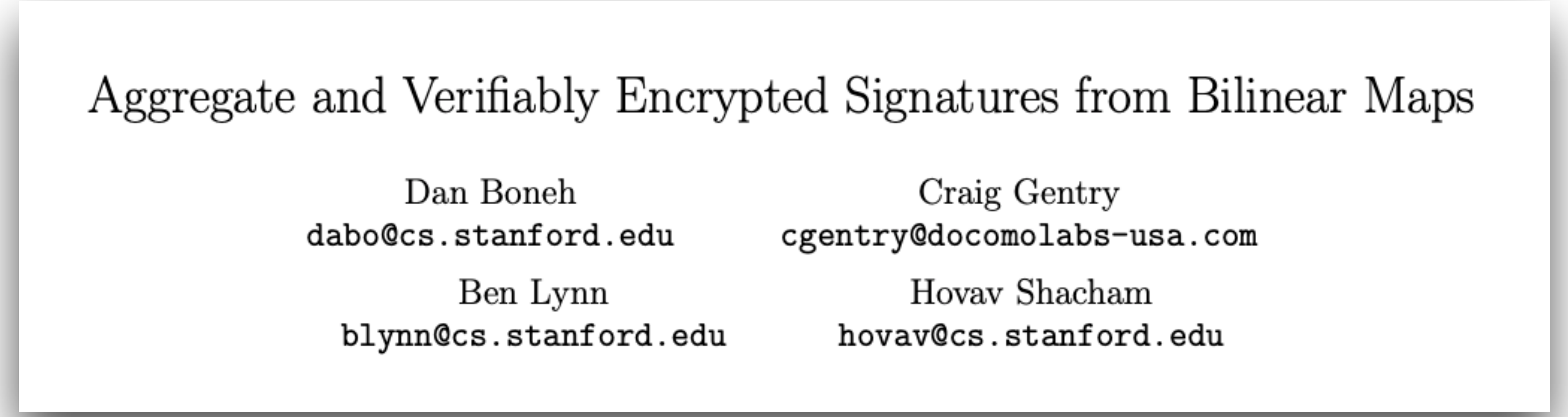
- **ECDSA**, as used in Bitcoin and Ethereum with secp256k1
- **Schnorr and EdDSA**, mainly via Ed25519 (deterministic Schnorr)
  Supports **aggregation** of keys & signatures, thanks to its linearity

Another important construction is **BLS signatures** (which use pairings)


Compact Multi-Signatures for Smaller Blockchains

Dan Boneh[1], Manu Drijvers[2,3], and Gregory Neven[2]

https://ia.cr/2018/483


Aggregate and Verifiably Encrypted Signatures from Bilinear Maps

Dan Boneh dabo@cs.stanford.edu
Craig Gentry cgentry@docomolabs-usa.com
Ben Lynn blynn@cs.stanford.edu
Hovav Shacham hovav@cs.stanford.edu

https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html

Homomorphic encryption

When Dec(Enc(**M1**) ○ Enc(**M2**)) = **M1 ❂ M2**

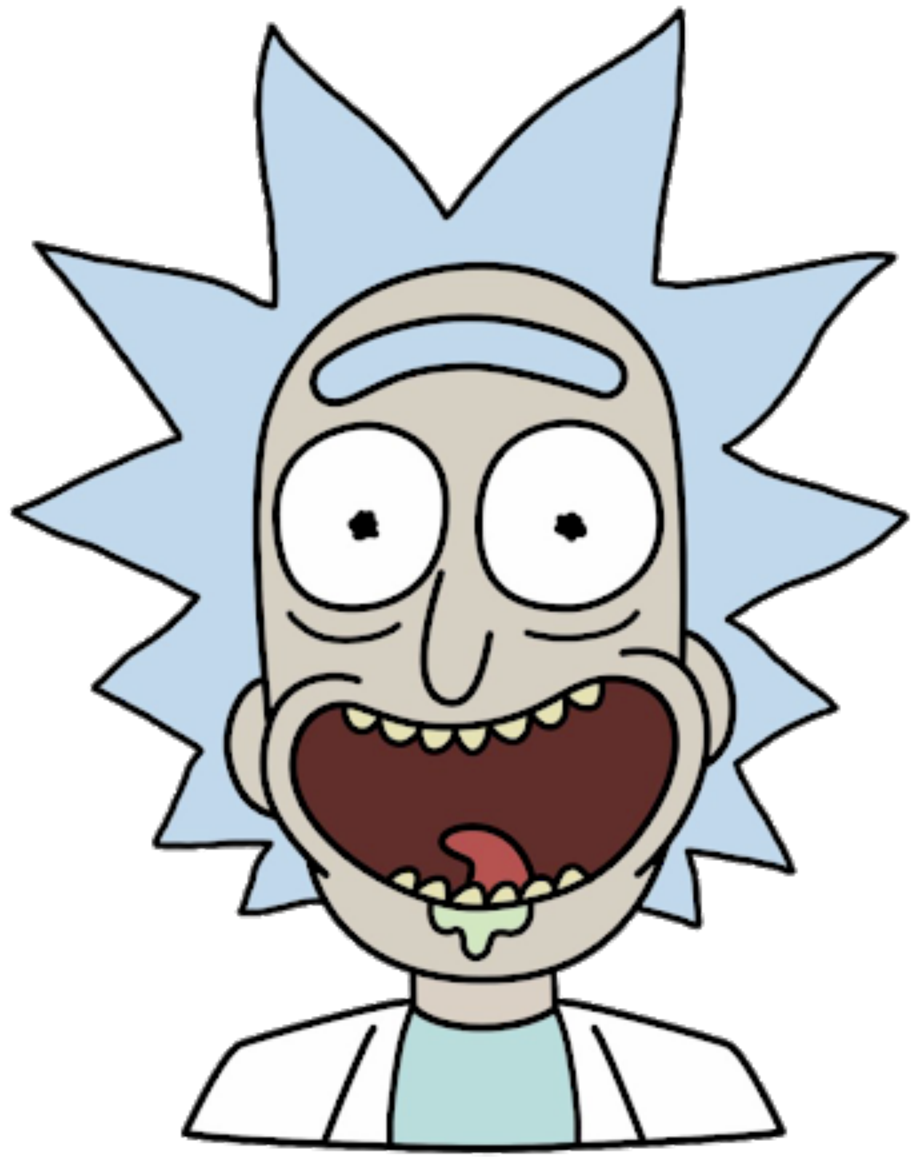
Operators ○ and ❂ can be the same operation (× with textbook RSA), or distinct ones (× and + in Paillier)

Depending on the context, a vulnerability or a feature

Leveraged in e-voting schemes, and in TSS constructions…

Commitments

Prover



Verifier

Setup()

c := **Commit**(x, r)

Hiding property: Verifier does not learn information on x

x, r

Binding property: Prover cannot reveal another value than x

Verify(x, r)

Threshold secret-sharing

Mainly based on **Shamir**'s scheme


Programming Techniques

R. Rivest Editor

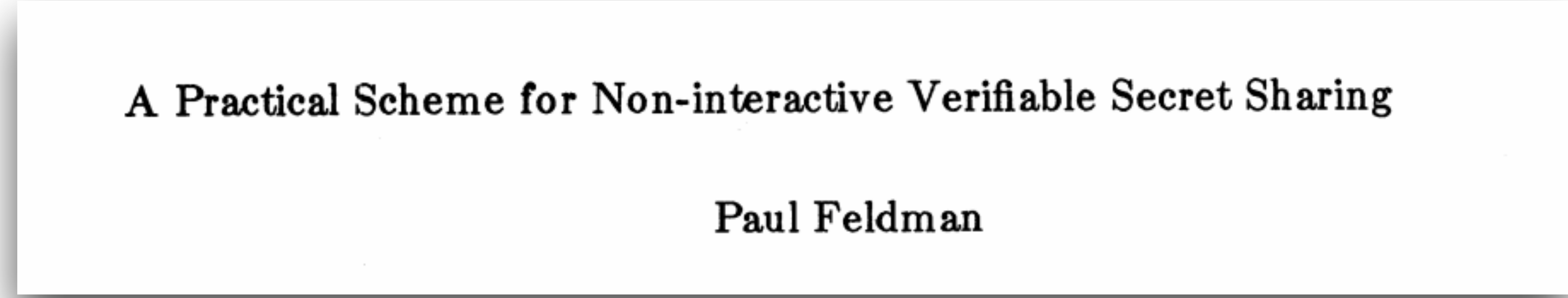
How to Share a Secret

Adi Shamir
Massachusetts Institute of Technology

Uses polynomial interpolation to reconstruct a secret from it shares, while preventing recovery with fewer shares than required

Verifiable secret sharing (**VSS**): participants get a cryptographic proof that the right secret was recovered, protecting against malicious dealers/ participants

A common VSS scheme is **Feldman's**, which uses homomorphic encryption


A Practical Scheme for Non-interactive Verifiable Secret Sharing

Paul Feldman

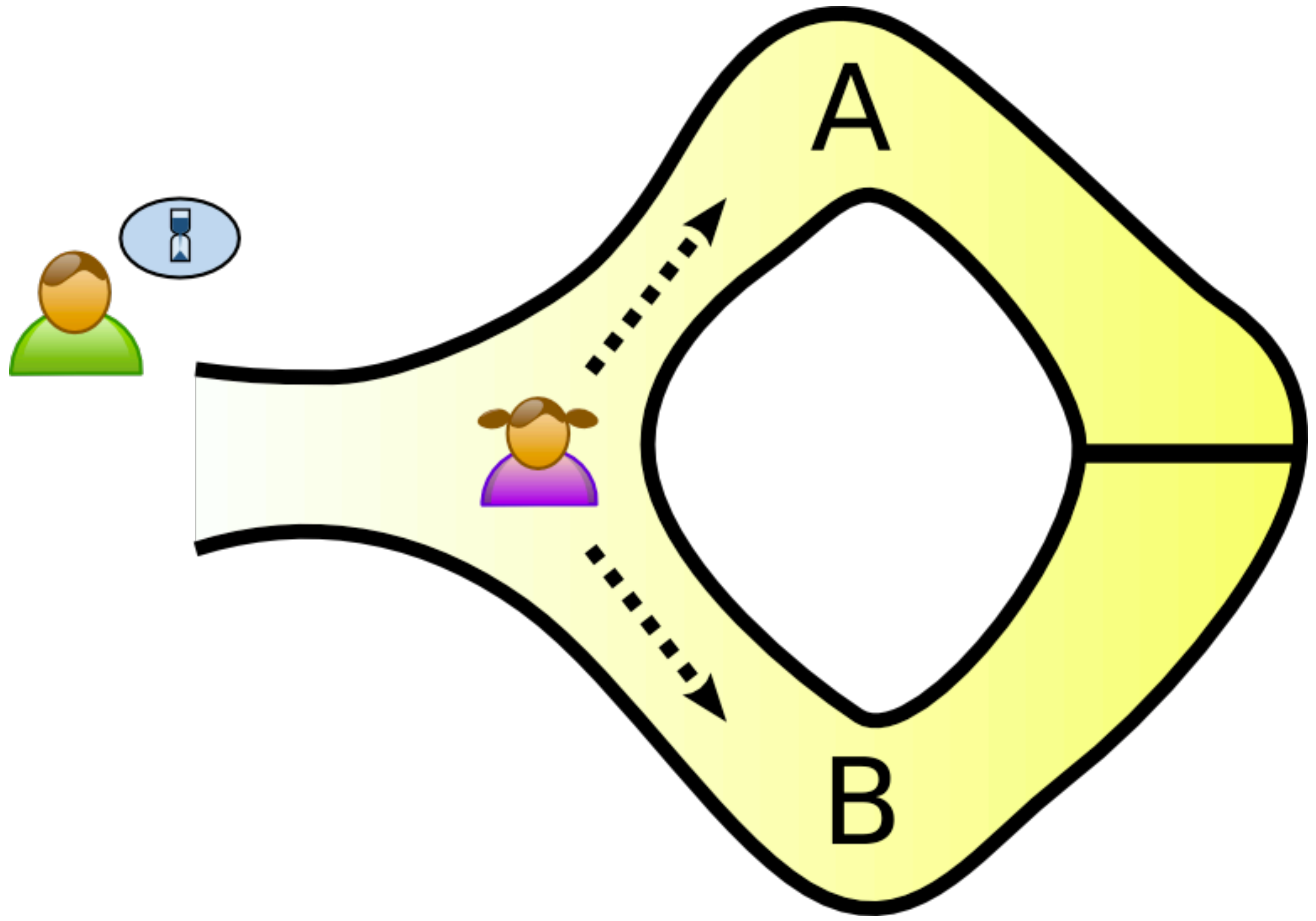
Zero-knowledge proofs

Protocol where a **prover convinces a verifier** that they know some mathematical statement (for example, a solution to the discrete log problem) without revealing any info on the statement (zero-knowledge)

Completeness: A prover should be able to convince a verifier if the statement is true

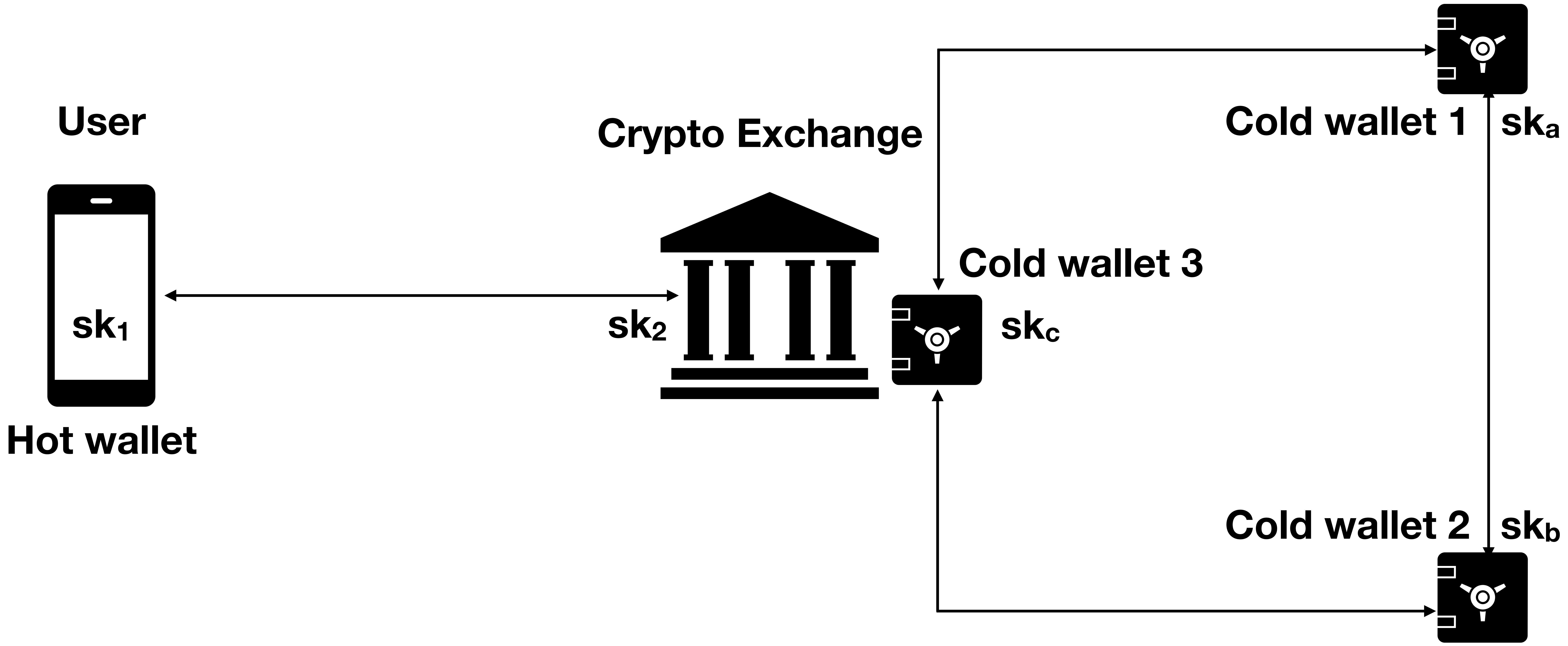
Soundness: A prover should not be able to convince a verifier if the statement is false

Non-Interactive Zero-Knowledge (NIZK): Not really a protocol, just a single data blob

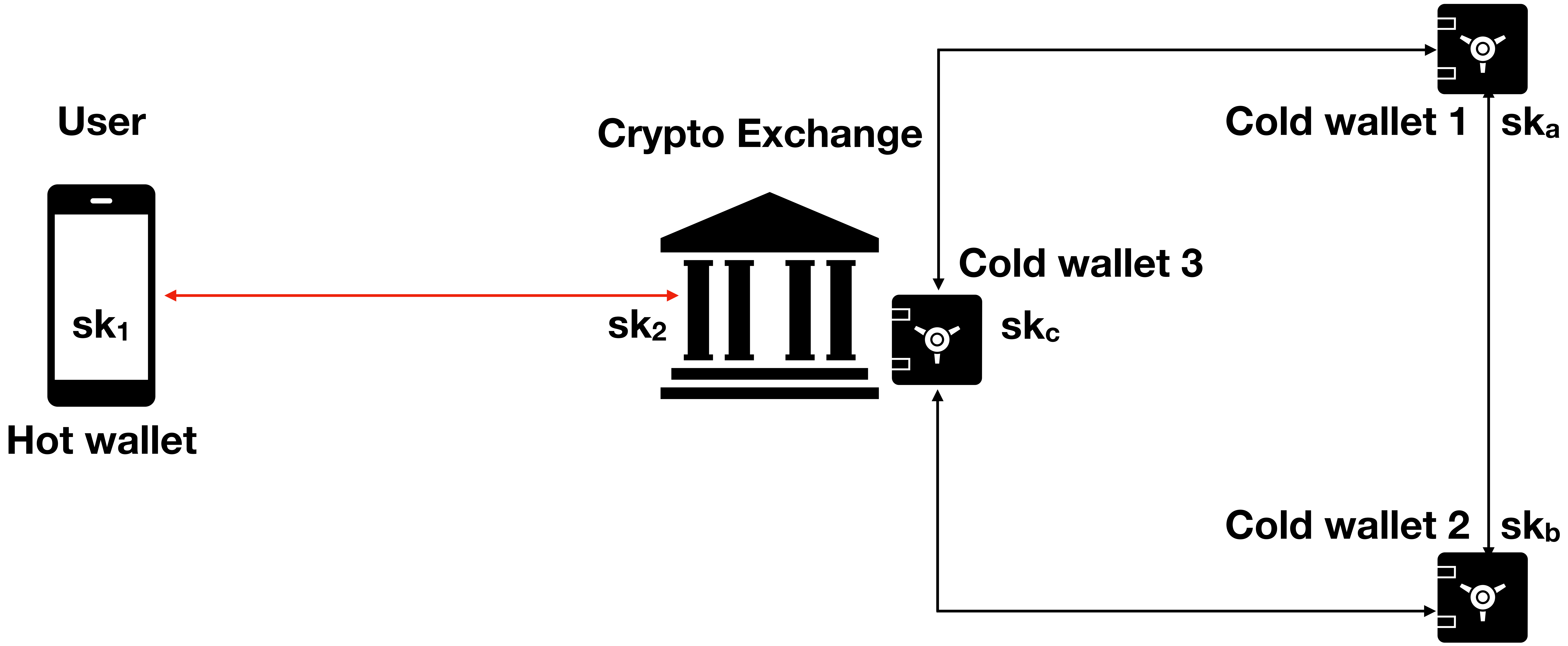


New attacks

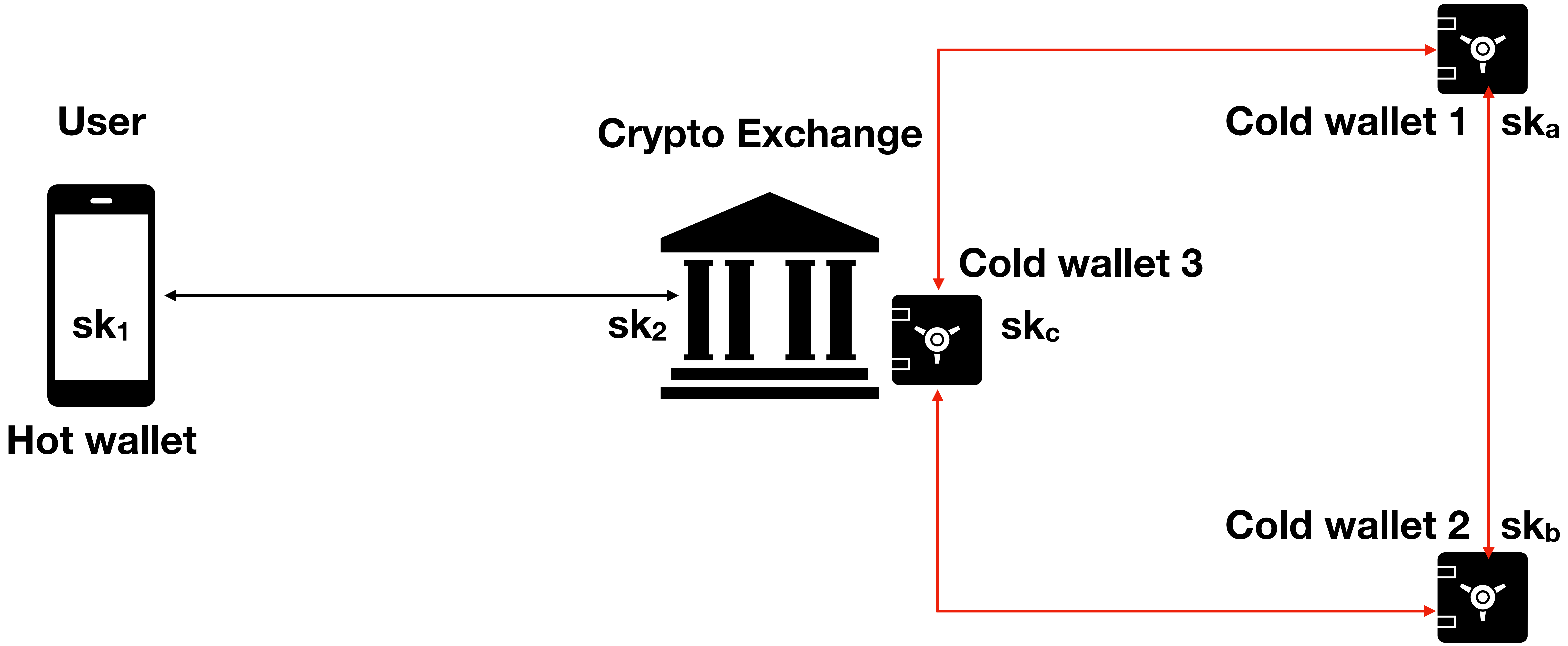
Setup


User
Crypto Exchange
Cold wallet 1
sk_a
Cold wallet 3
sk_1
sk_2
sk_c
Hot wallet
Cold wallet 2
sk_b

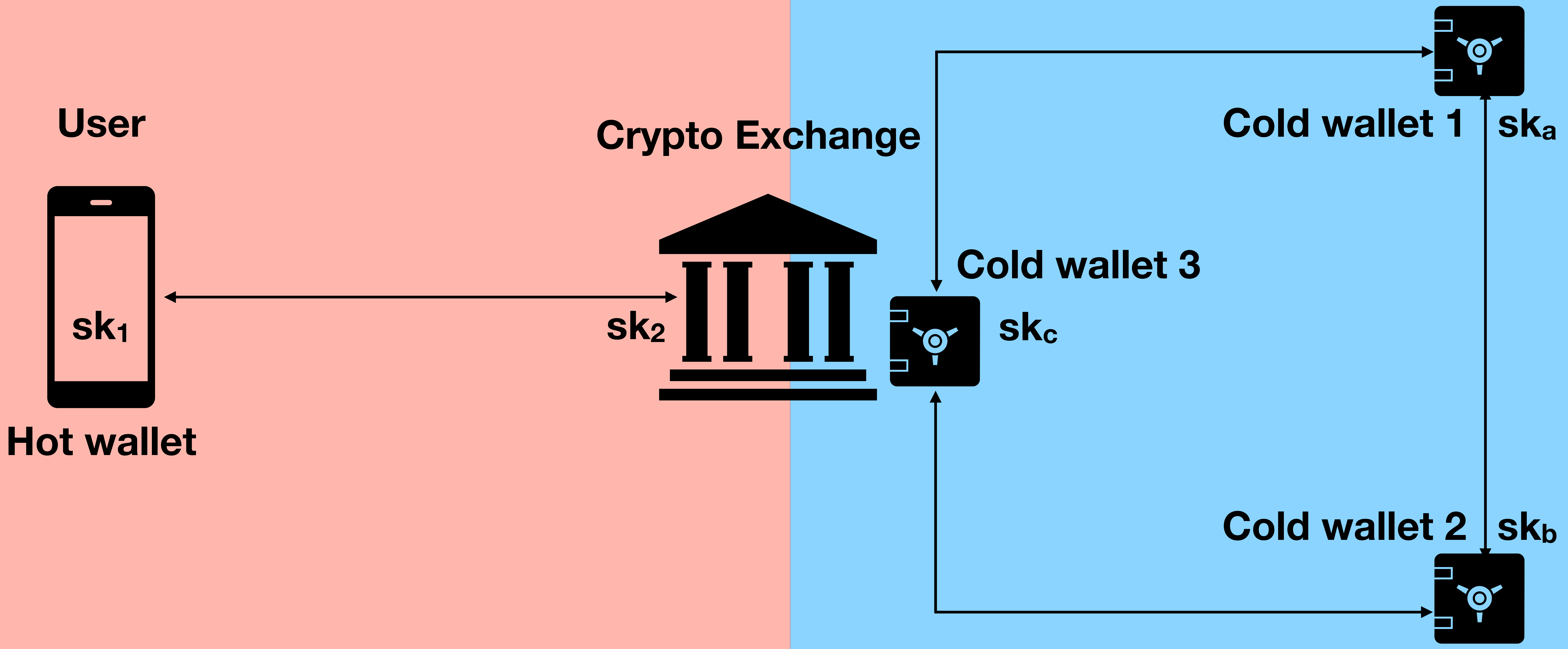
Setup


User
Hot wallet
sk1
Crypto Exchange
sk2
Cold wallet 3
skc
Cold wallet 1
ska
Cold wallet 2
skb

Setup


User
Crypto Exchange
Cold wallet 1
ska
Cold wallet 3
sk1
sk2
skc
Hot wallet
Cold wallet 2
skb

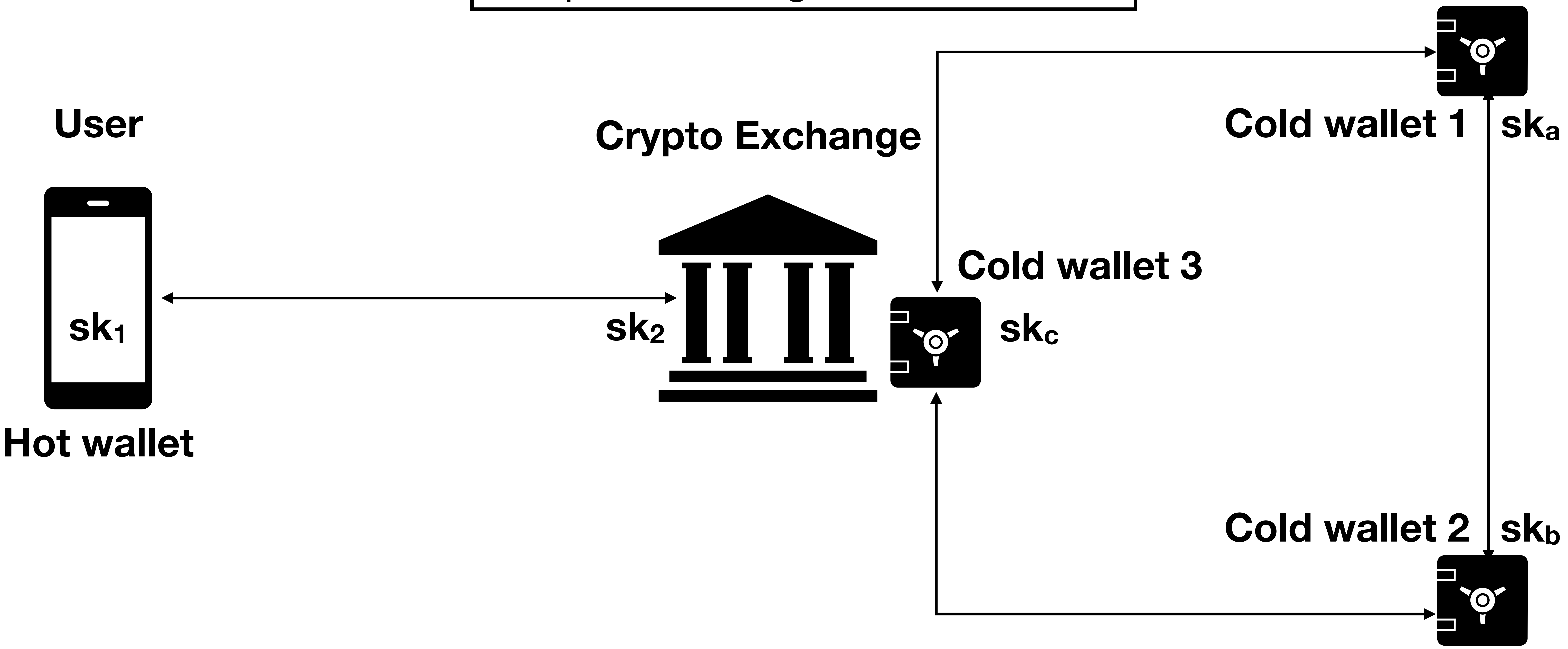
Setup



User
Crypto Exchange
sk1
sk2
Hot wallet
Cold wallet 1
ska
Cold wallet 3
skc
Cold wallet 2
skb

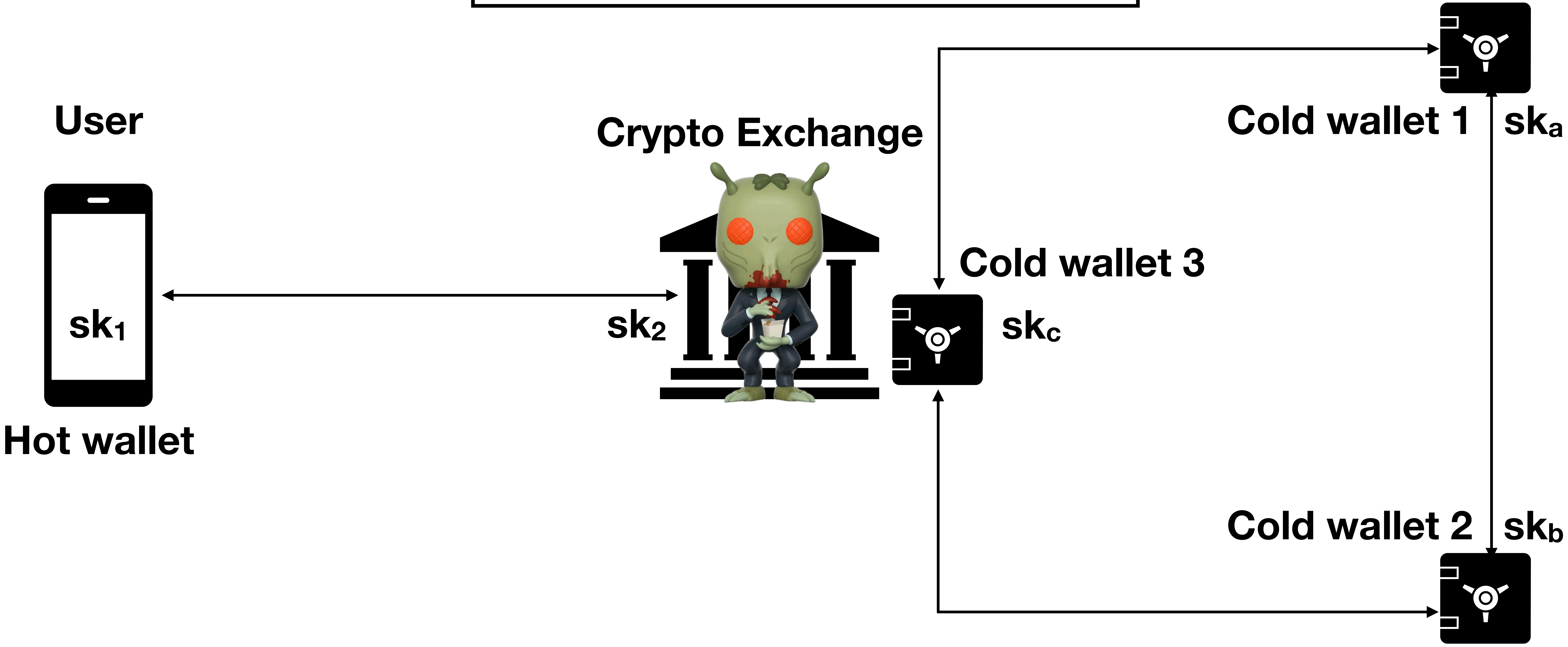
Attacker Model

TSS prevents Single Point of Failure



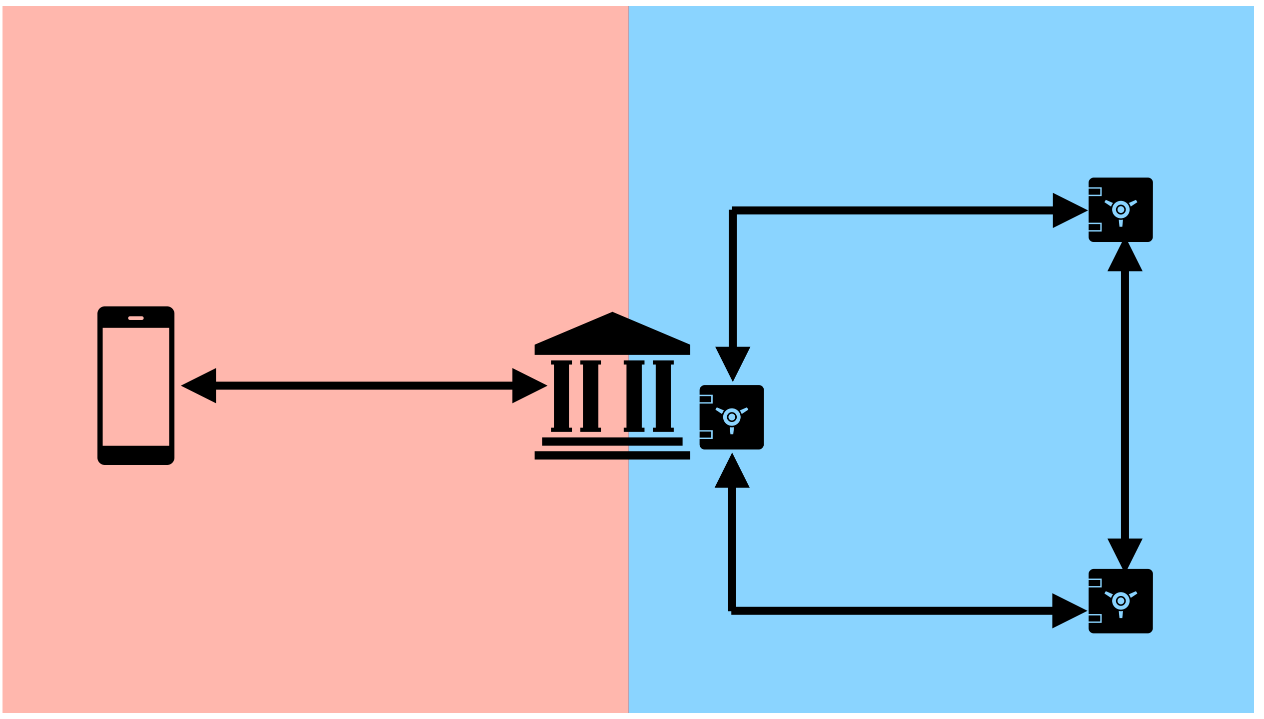
Attacker Model

TSS prevents Single Point of Failure



Our Attacks

Forget-and-Forgive		DoS (e.g. for blackmail)
Lather, Rinse, Repeat		Stolen funds
Golden Shoe		Stolen funds



Our attacks

Forget-and-Forgive		DoS (e.g. for blackmail)
Lather, Rinse, Repeat		Stolen funds
Golden Shoe CVE-2020-12118		Stolen funds

All responsibly disclosed

Our attacks

Forget-and-Forgive		DoS (e.g. for blackmail)
Lather, Rinse, Repeat		Stolen funds
Golden Shoe CVE-2020-12118		Stolen funds

All responsibly disclosed

Our attacks

Forget-and-Forgive		DoS (e.g. for blackmail)
Lather, Rinse, Repeat		Stolen funds
Golden Shoe CVE-2020-12118		Stolen funds

All responsibly disclosed

Our attacks

Forget-and-Forgive		DoS (e.g. for blackmail)
Lather, Rinse, Repeat		Stolen funds
Golden Shoe CVE-2020-12118		Stolen funds

All responsibly disclosed

Attacks taxonomy

	Forget-and-Forgive	Lather, Rinse, Repeat	Golden Shoe
Interactive — multiple rounds of communication between multiple end points	🔥		🔥
Use of non-standard/cutting edge crypto primitives		🔥	🔥
Each step must be checked for correctness by all participants		🔥	🔥

Attacks taxonomy

	Forget-and-Forgive	Lather, Rinse, Repeat	Golden Shoe
Interactive — multiple rounds of communication between multiple end points	🔥		🔥
Use of non-standard/cutting edge crypto primitives		🔥	🔥
Each step must be checked for correctness by all participants		🔥	🔥

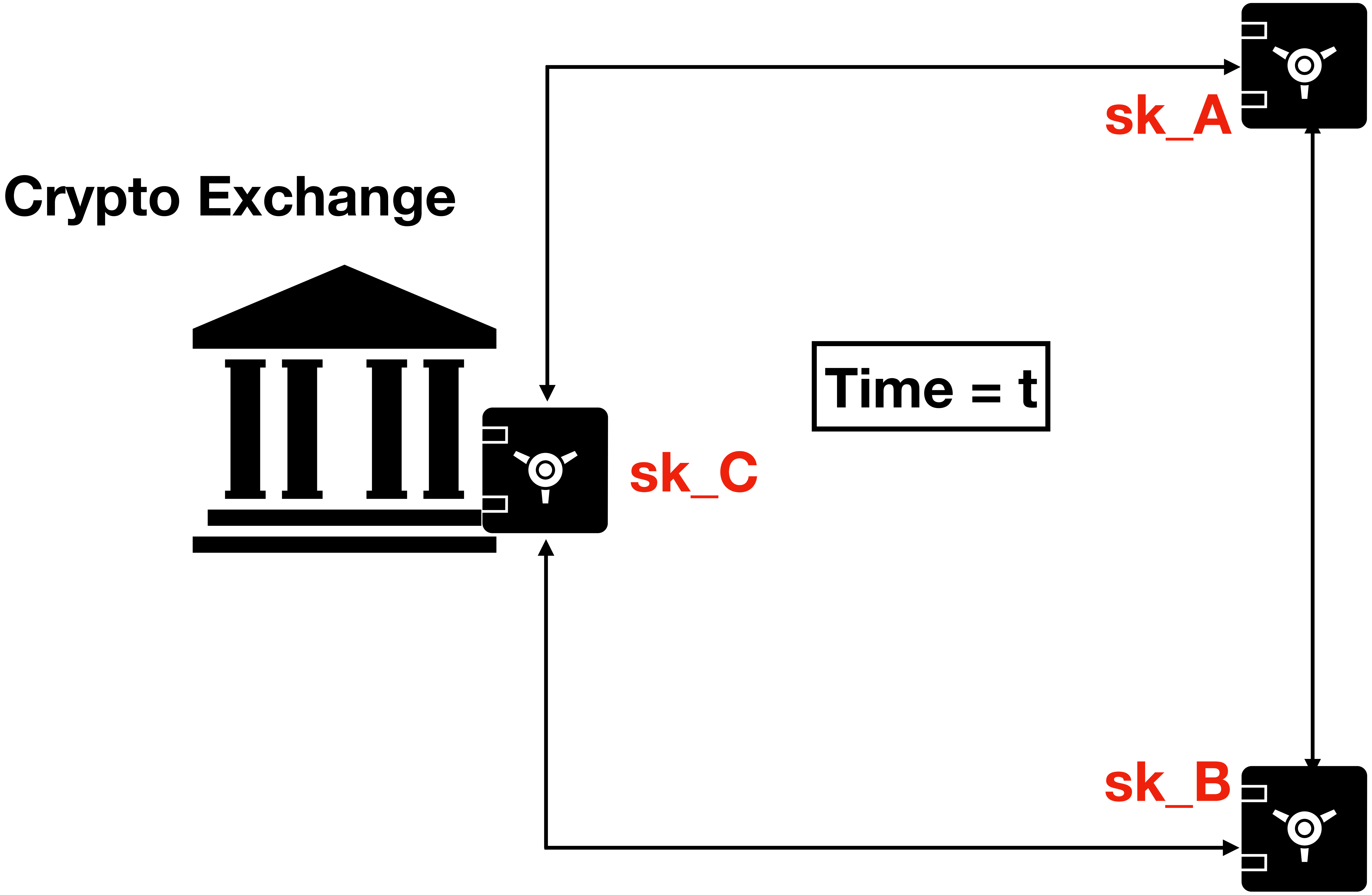
Attacks taxonomy

	Forget-and-Forgive	Lather, Rinse, Repeat	Golden Shoe
Interactive — multiple rounds of communication between multiple end points	🔥		🔥
Use of non-standard/cutting edge crypto primitives		🔥	🔥
Each step must be checked for correctness by all participants		🔥	🔥

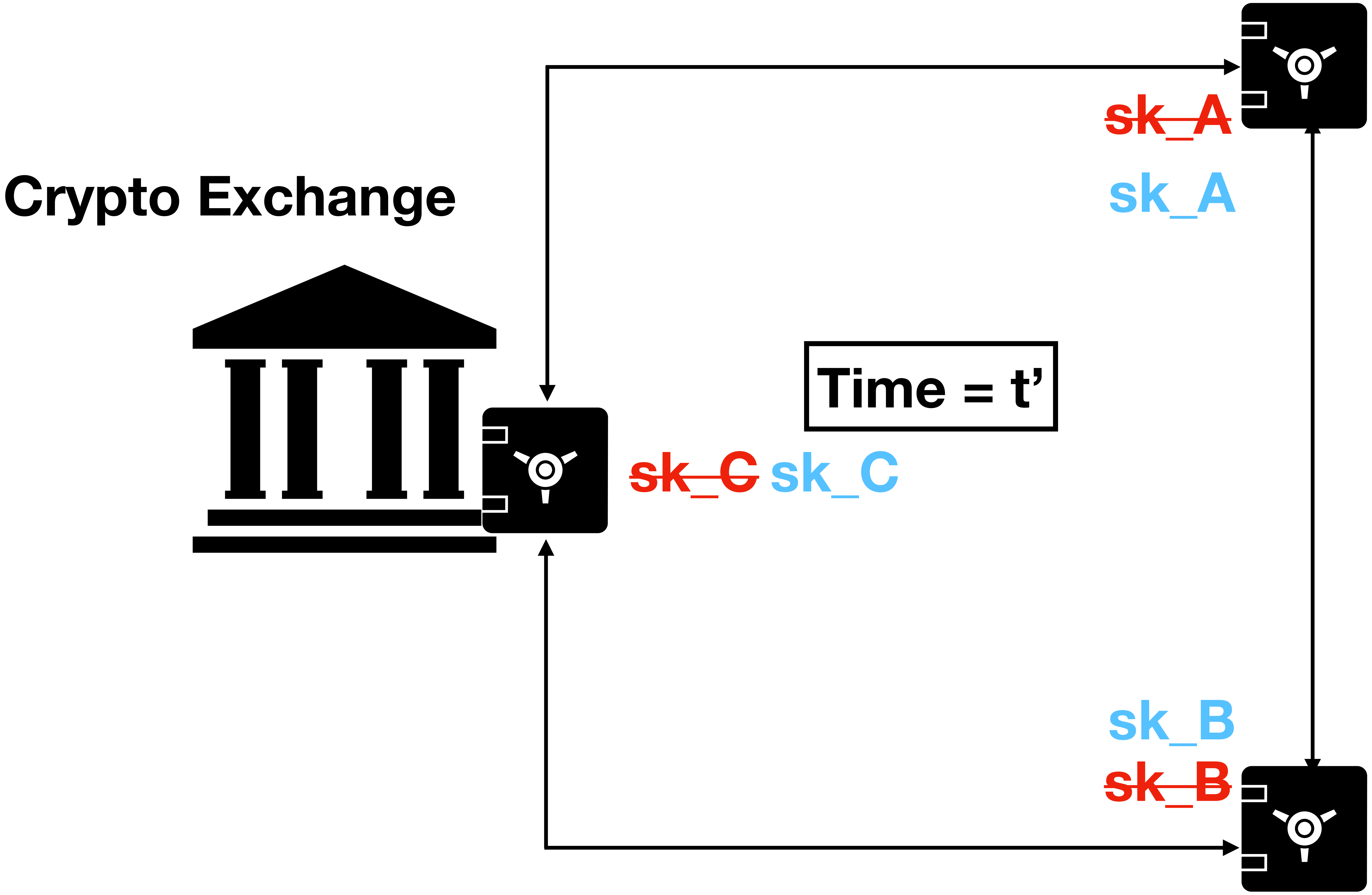
Attacks taxonomy

	Forget-and-Forgive	Lather, Rinse, Repeat	Golden Shoe
Interactive — multiple rounds of communication between multiple end points	🔥		🔥
Use of non-standard/cutting edge crypto primitives		🔥	🔥
Each step must be checked for correctness by all participants		🔥	🔥

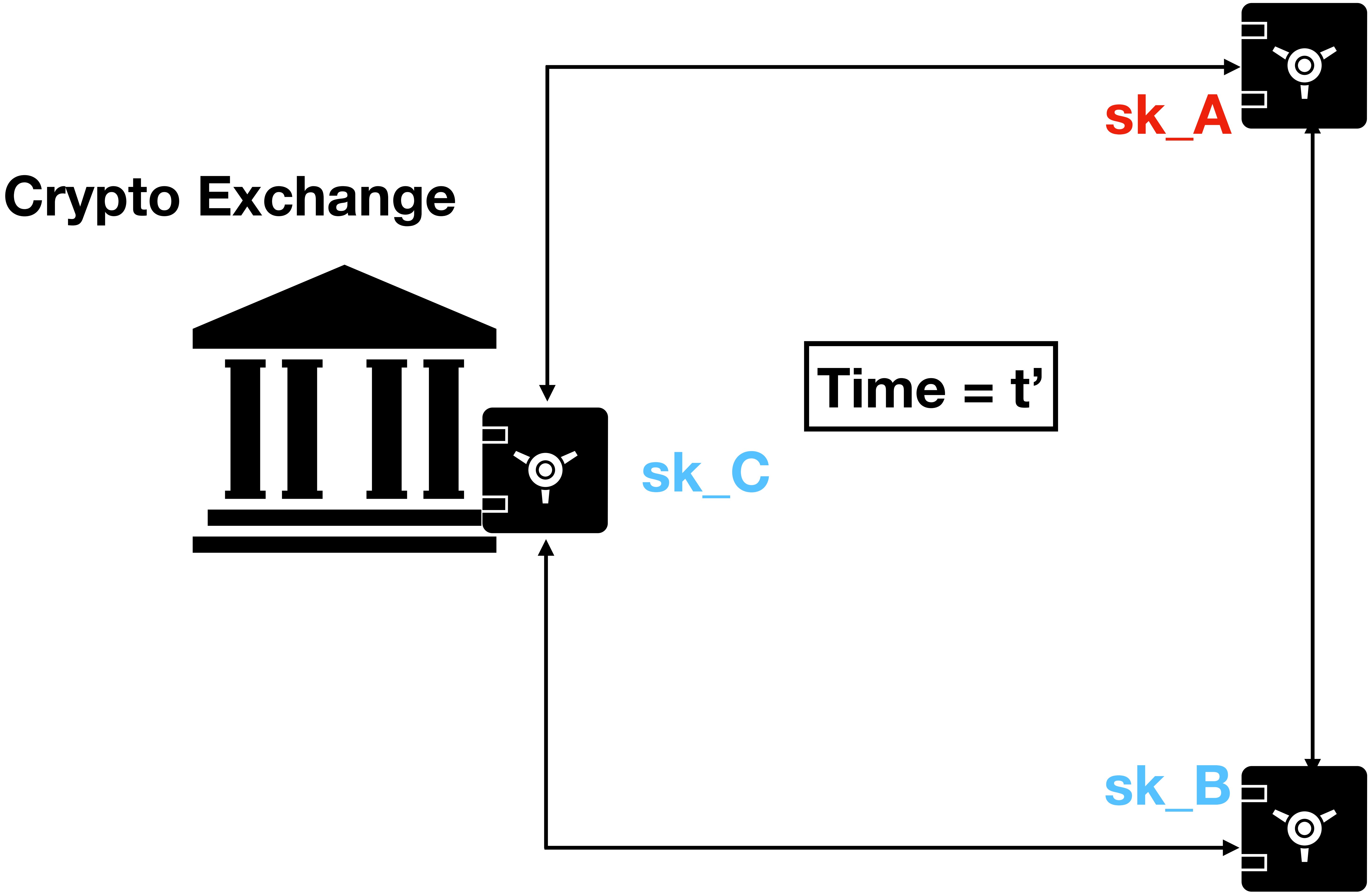
Forget-and-Forgive



Forget-and-Forgive

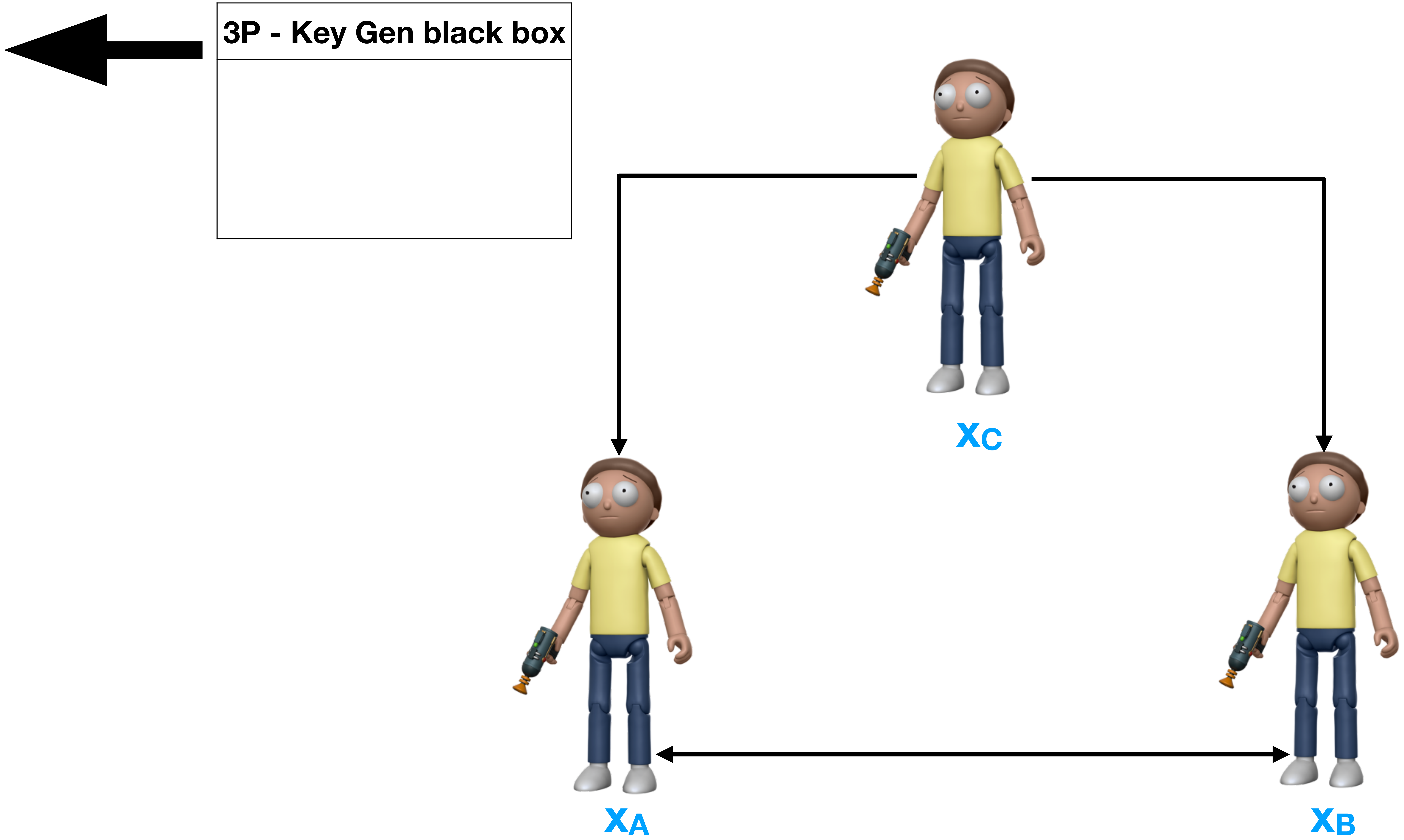


Forget-and-Forgive



Forget-and-Forgive

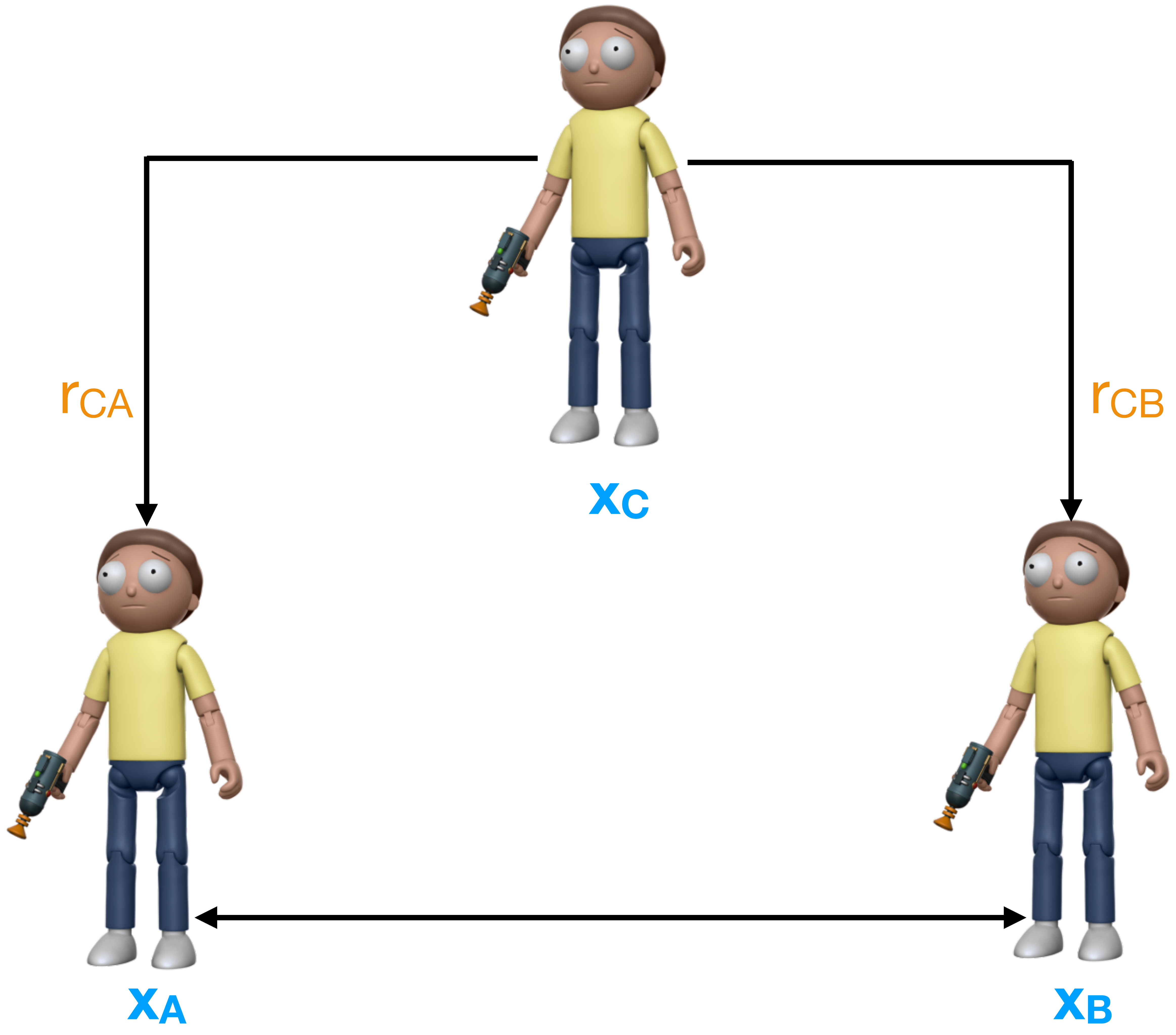
$y = x_A + x_B + x_C$



Forget-and-Forgive

$$y = x_A + x_B + x_C$$

$$r_{CA} + r_{CB} = 0$$



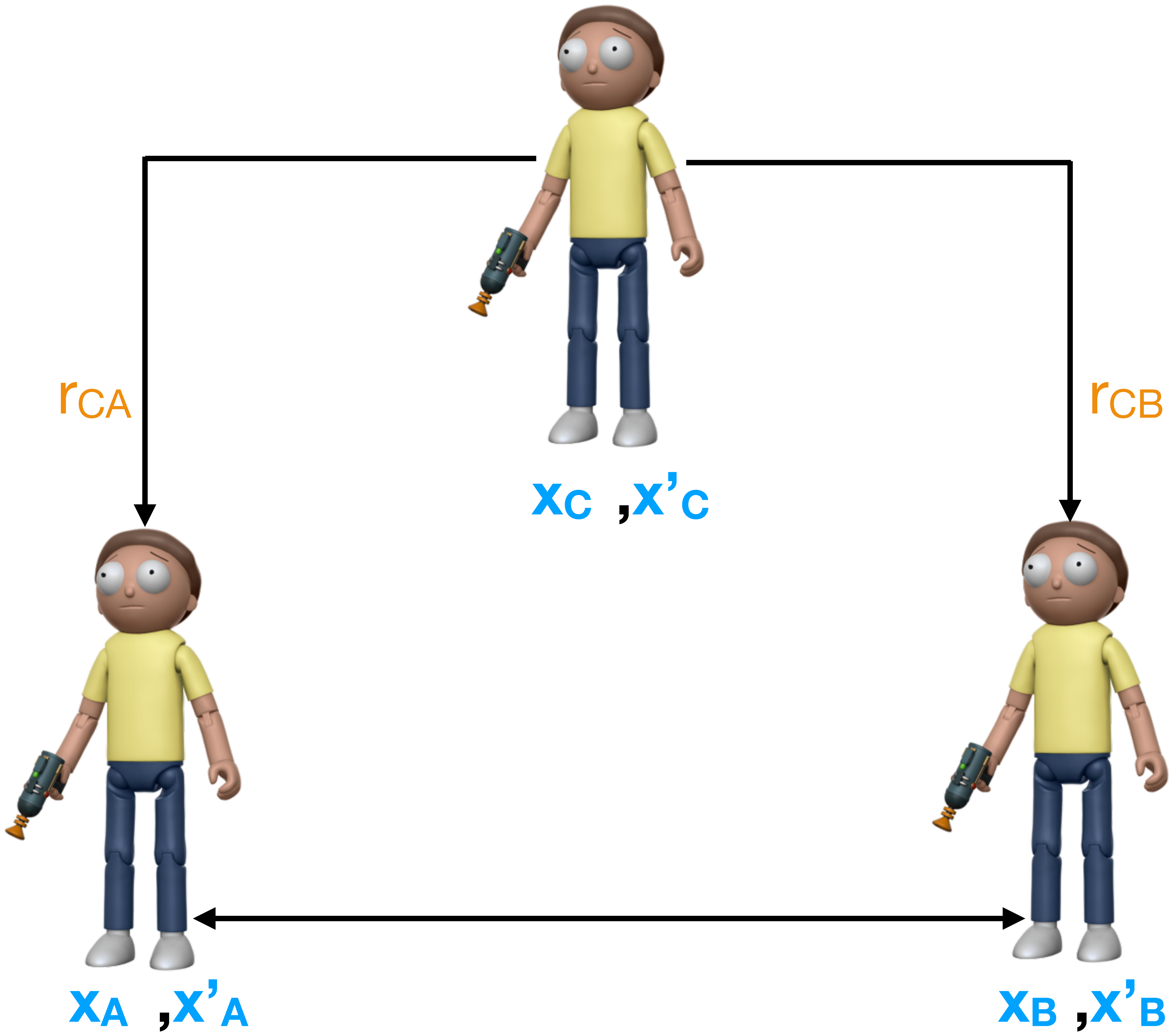
Forget-and-Forgive

$$y = x_A + x_B + x_C$$

$$r_{CA} + r_{CB} = 0$$

$$x'_A = x_A + r_{CA}, \quad x'_B = x_B + r_{CB}, \quad x'_C = x_C$$

$$y' = y$$



Forget-and-Forgive

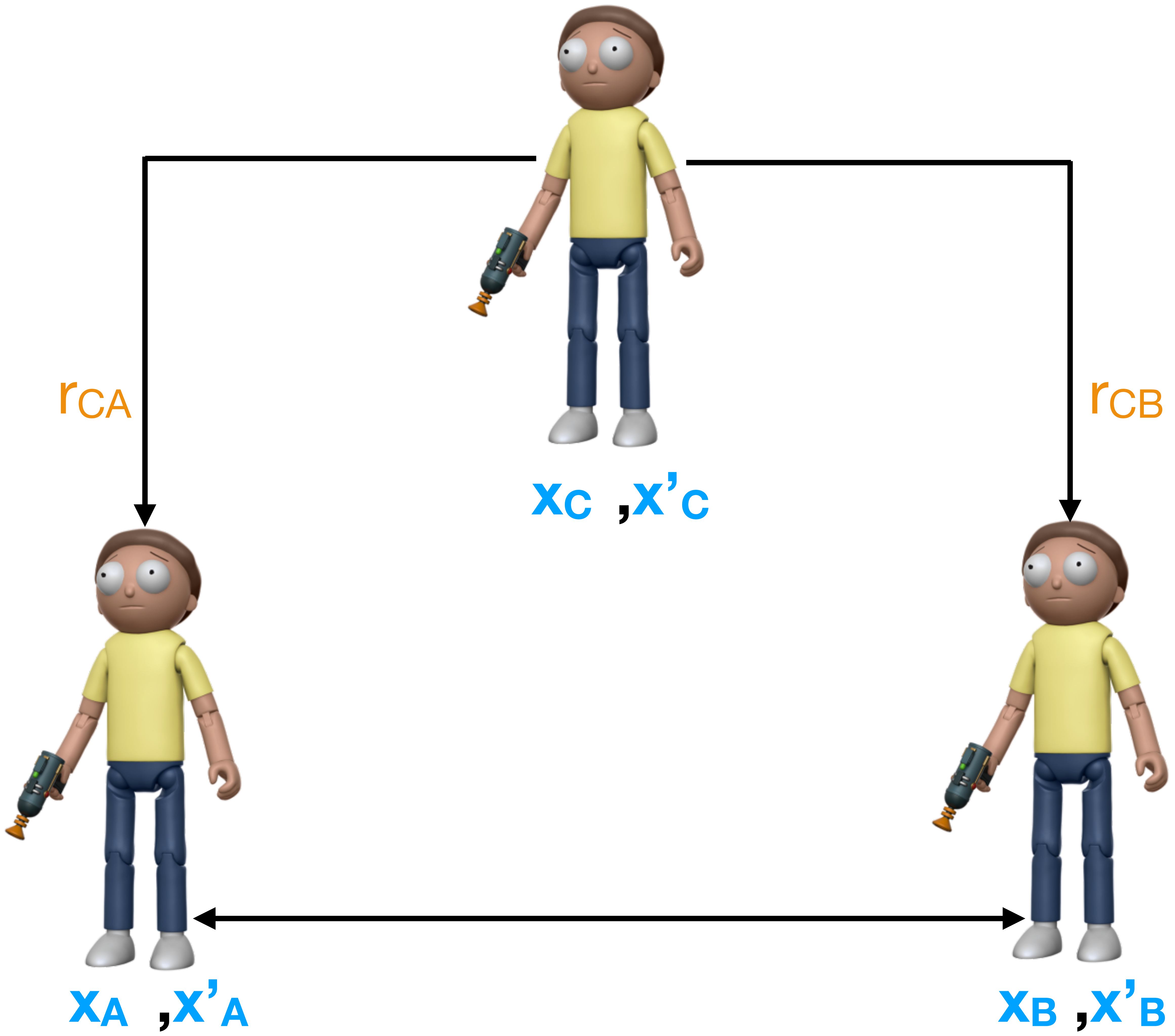
$$y = x_A + x_B + x_C$$

$$r_{CA} + r_{CB} = 0$$

$$x'_A = x_A + r_{CA} \quad , x'_B = x_B + r_{CB} \quad , x'_C = x_C$$

$$y' = y$$

Done ?



Forget-and-Forgive

$y = x_A + x_B + x_C$

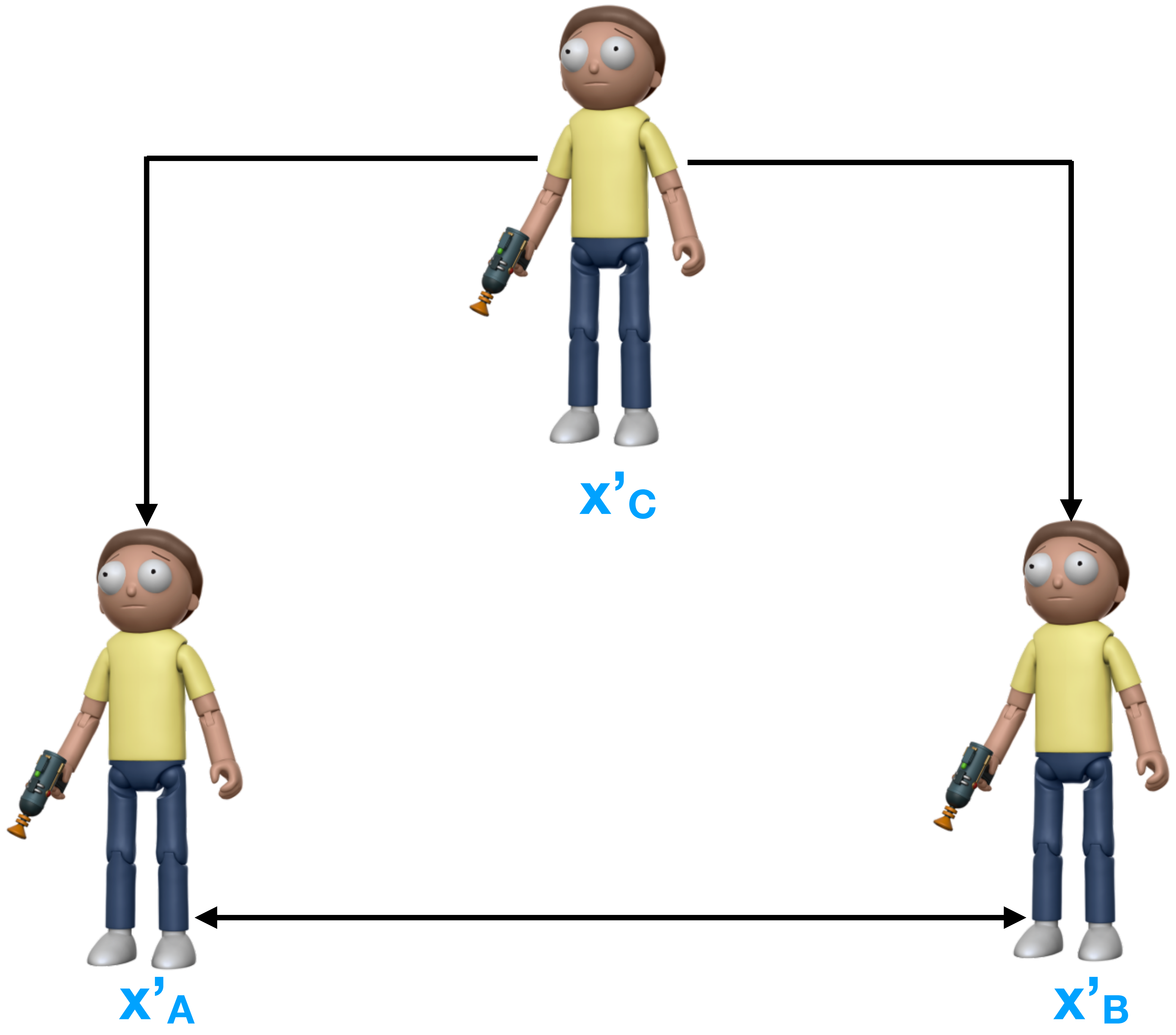
$r_{CA} + r_{CB} = 0$

$x'_A = x_A + r_{CA}$, $x'_B = x_B + r_{CB}$, $x'_C = x_C$

$y' = y$

Done ?

Delete x_A , x_B , x_C



Forget-and-Forgive

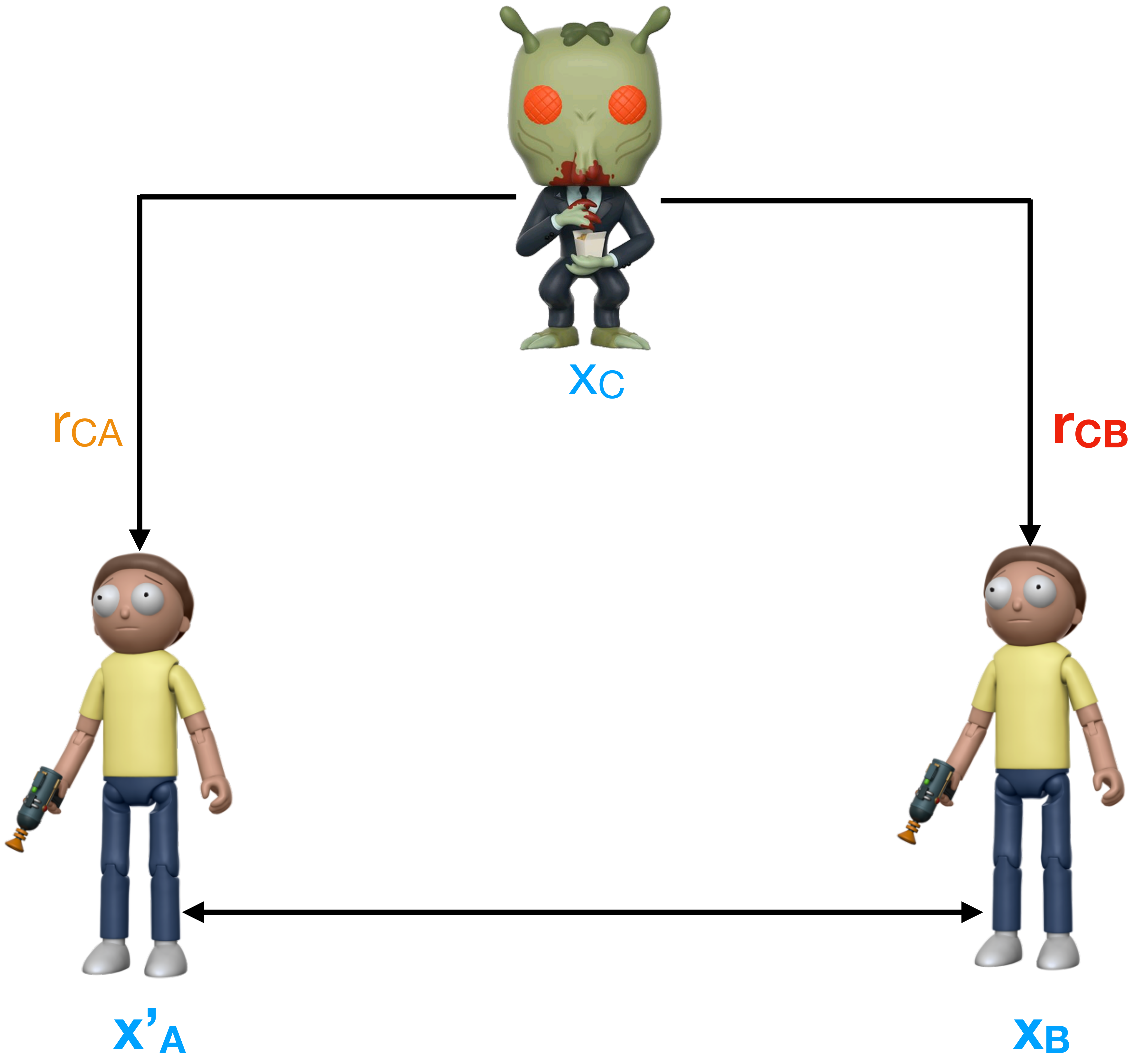
$y = x_A + x_B + x_C$

$r_{CA} + r_{CB} = 0$

$x'_A = x_A + r_{CA}$

↓

Delete x_A



$r_{CA} + r_{CB} \neq 0$

$x'_B = x_B + r_{CB}$

↓

Abort

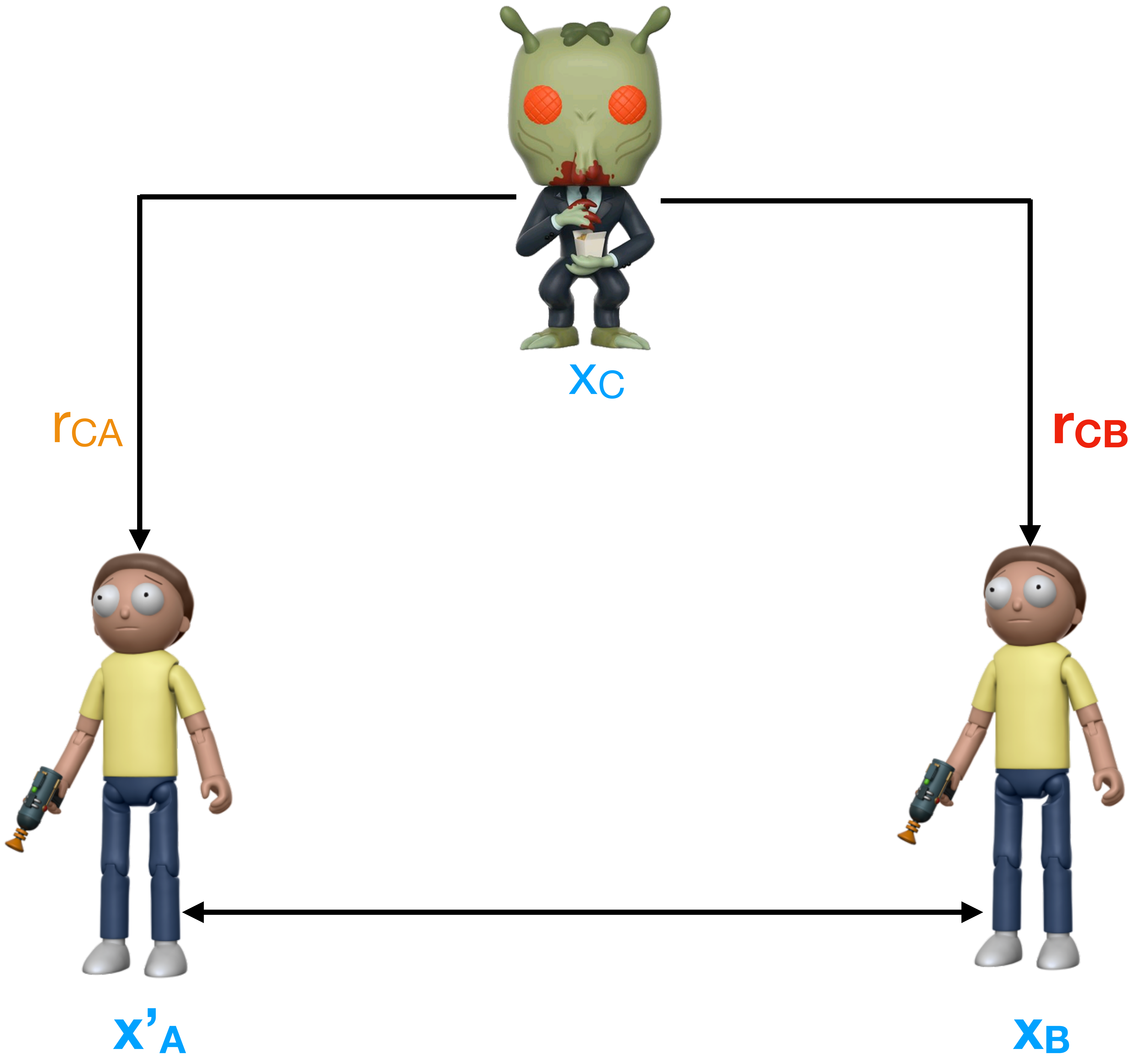
Forget-and-Forgive

$y = x_A + x_B + x_C$

$r_{CA} + r_{CB} = 0$

$x'_A = x_A + r_{CA}$

→ Delete x_A



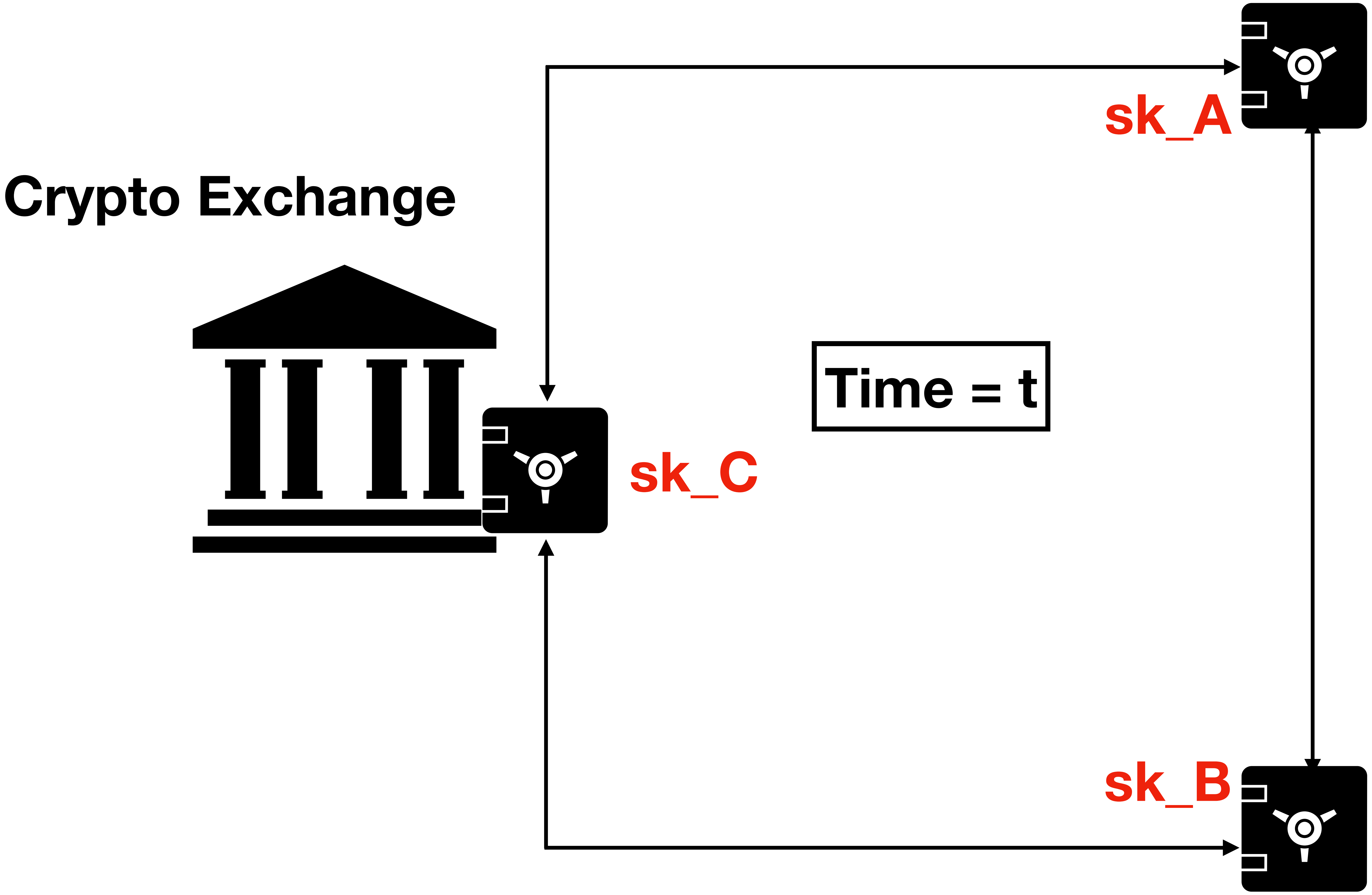
$r_{CA} + r_{CB} \,!= 0$

$x'_B = x_B + r_{CB}$

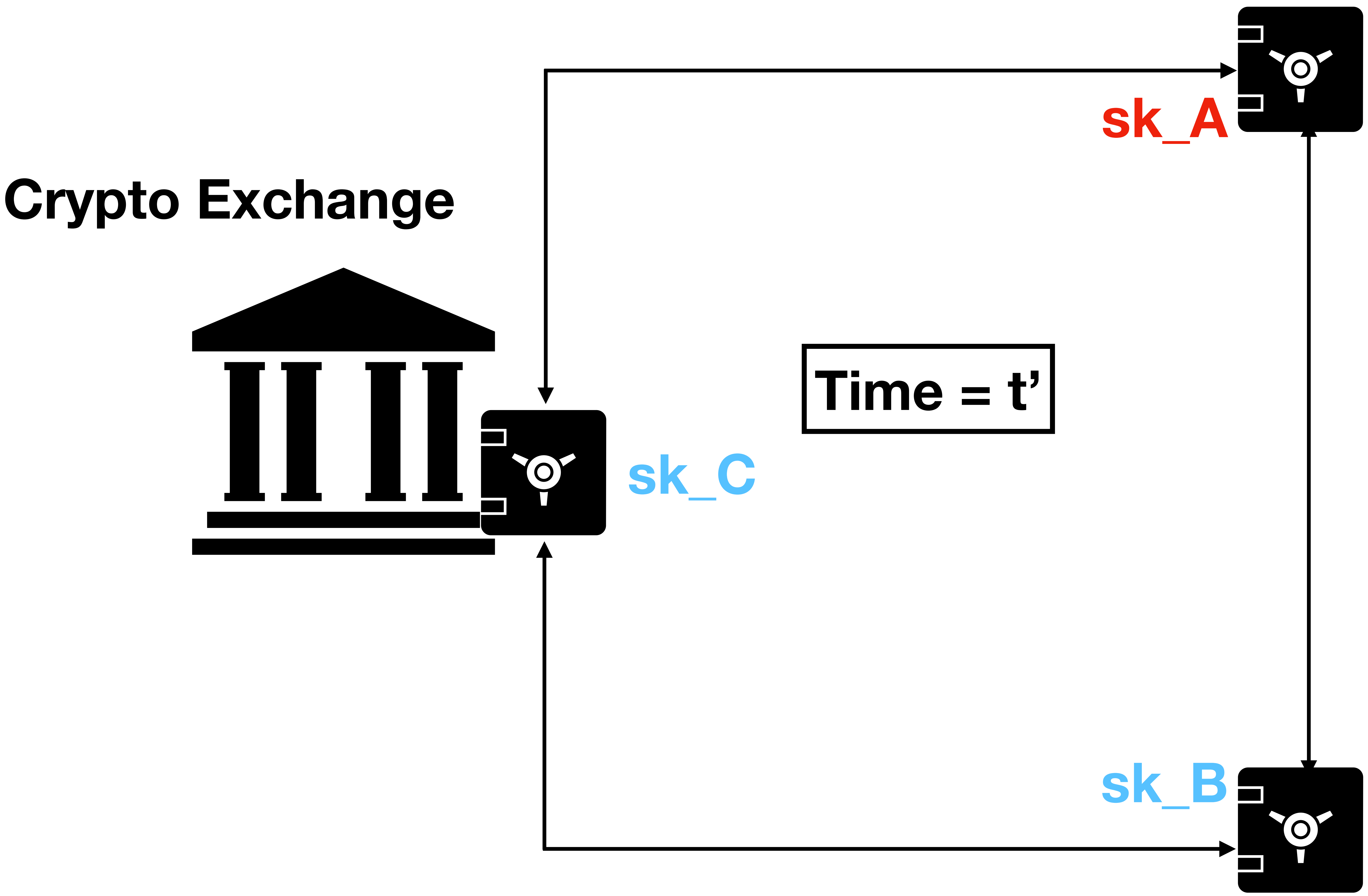
→ Abort

$y' = x'_A + x_B + x'_C \quad != y$

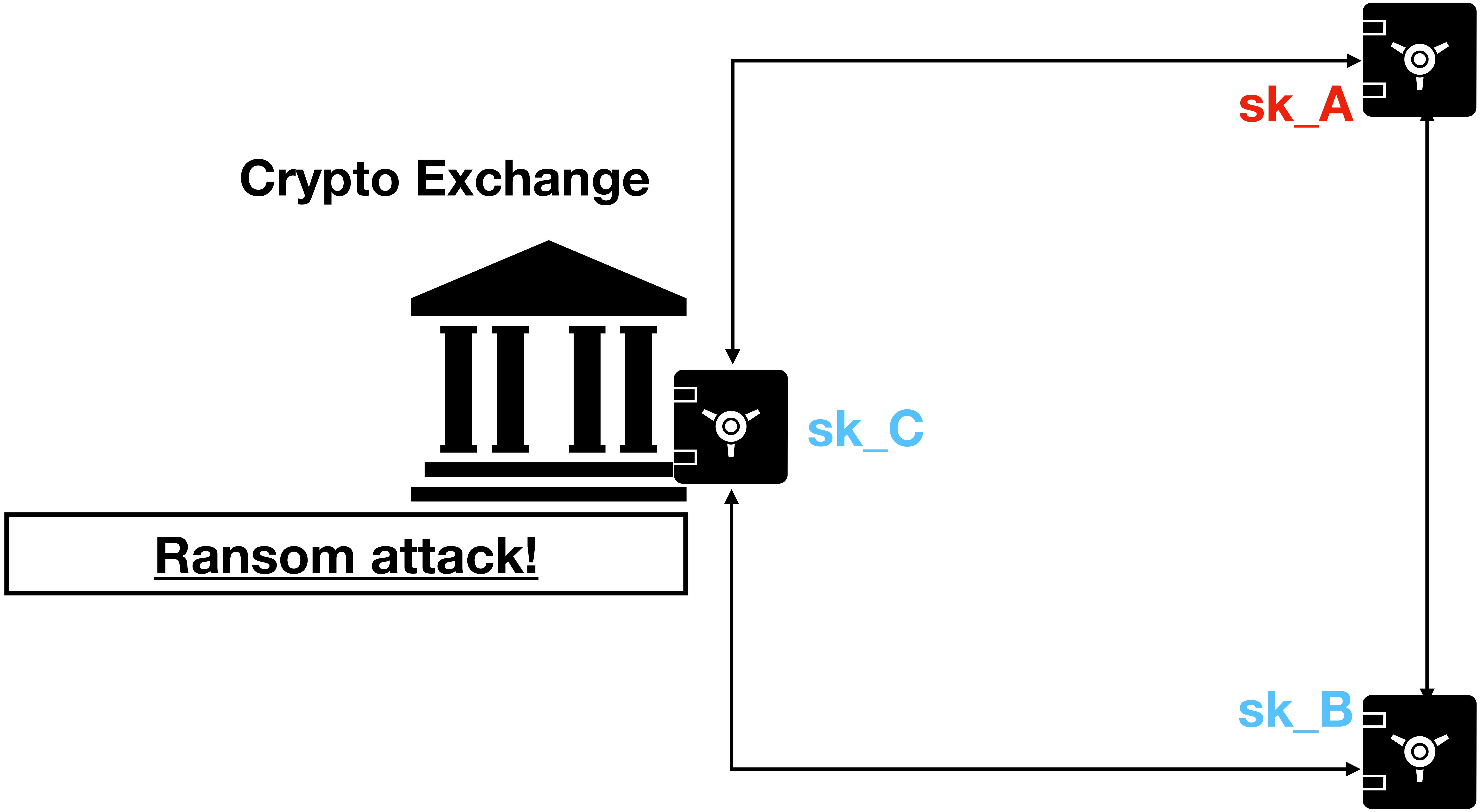
Forget-and-Forgive



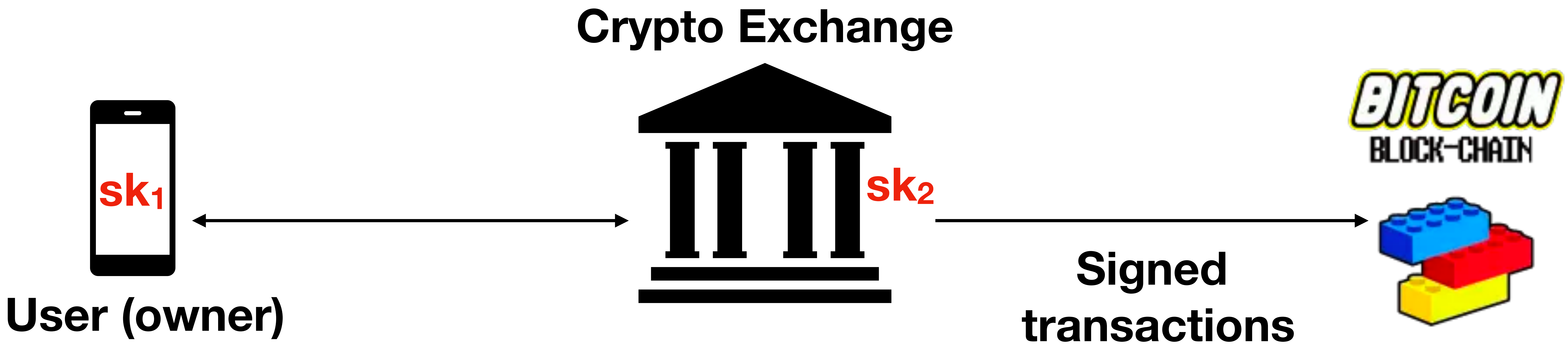
Forget-and-Forgive



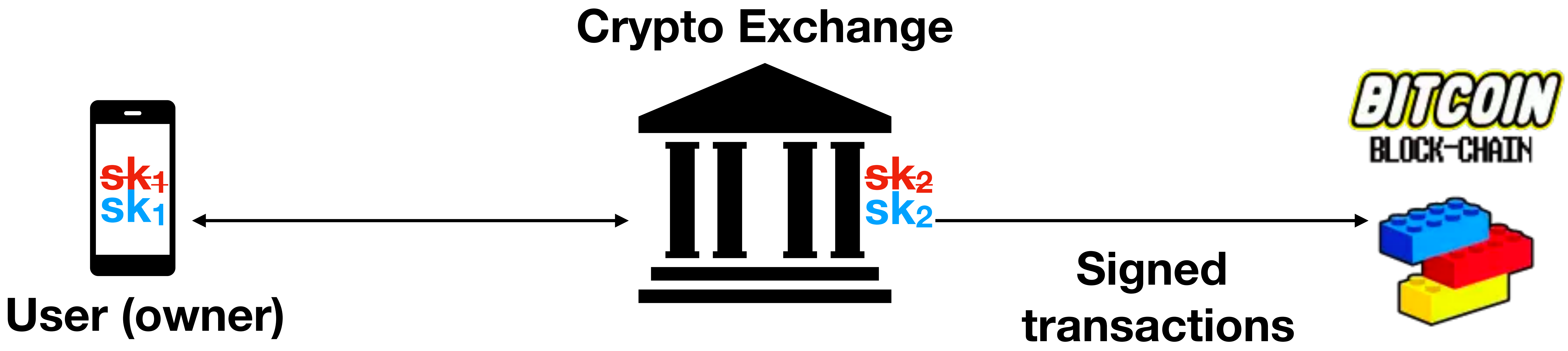
Forget-and-Forgive



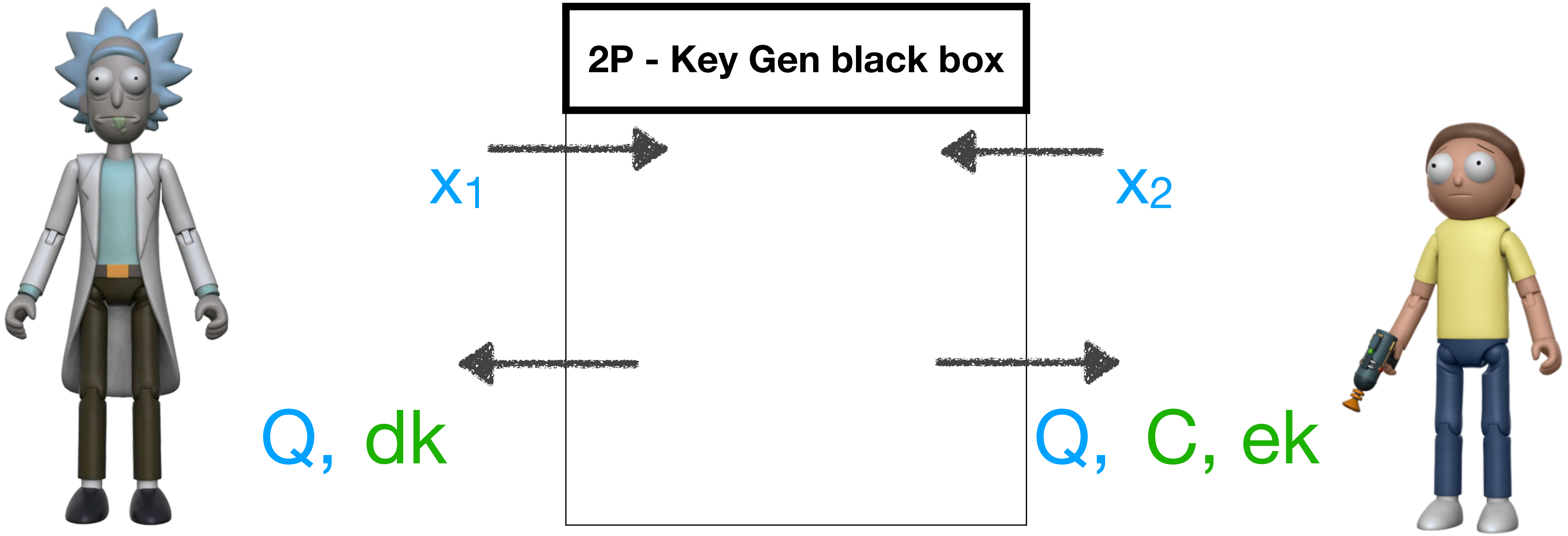
Lather, Rinse, Repeat



Lather, Rinse, Repeat



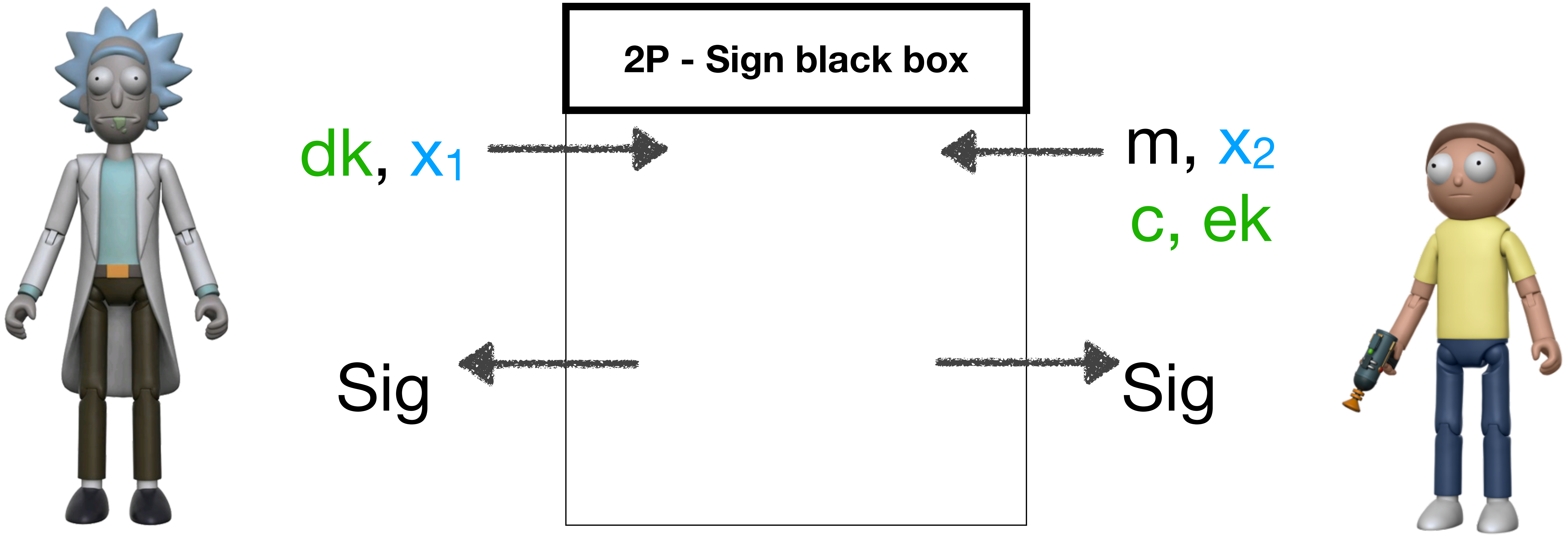
Lather, Rinse, Repeat



{dk,ek} is a key pair for homomorphic encryption scheme {Enc,Dec}

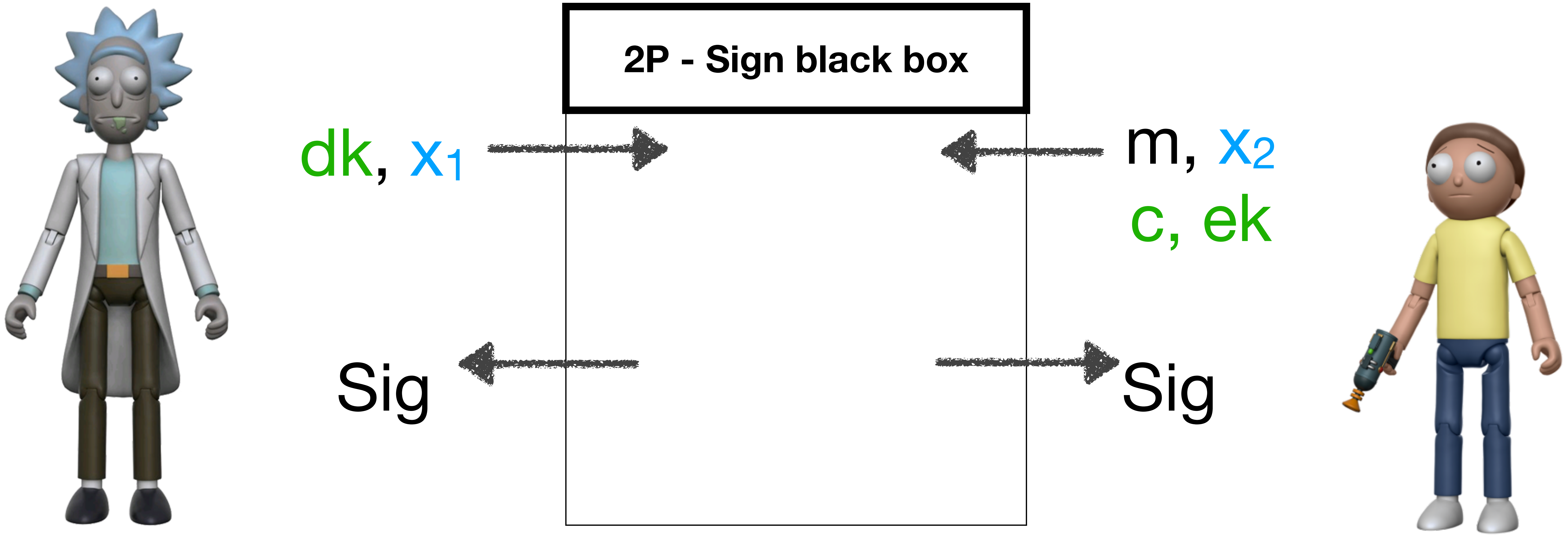
$C = Enc(x_1)$

Lather, Rinse, Repeat

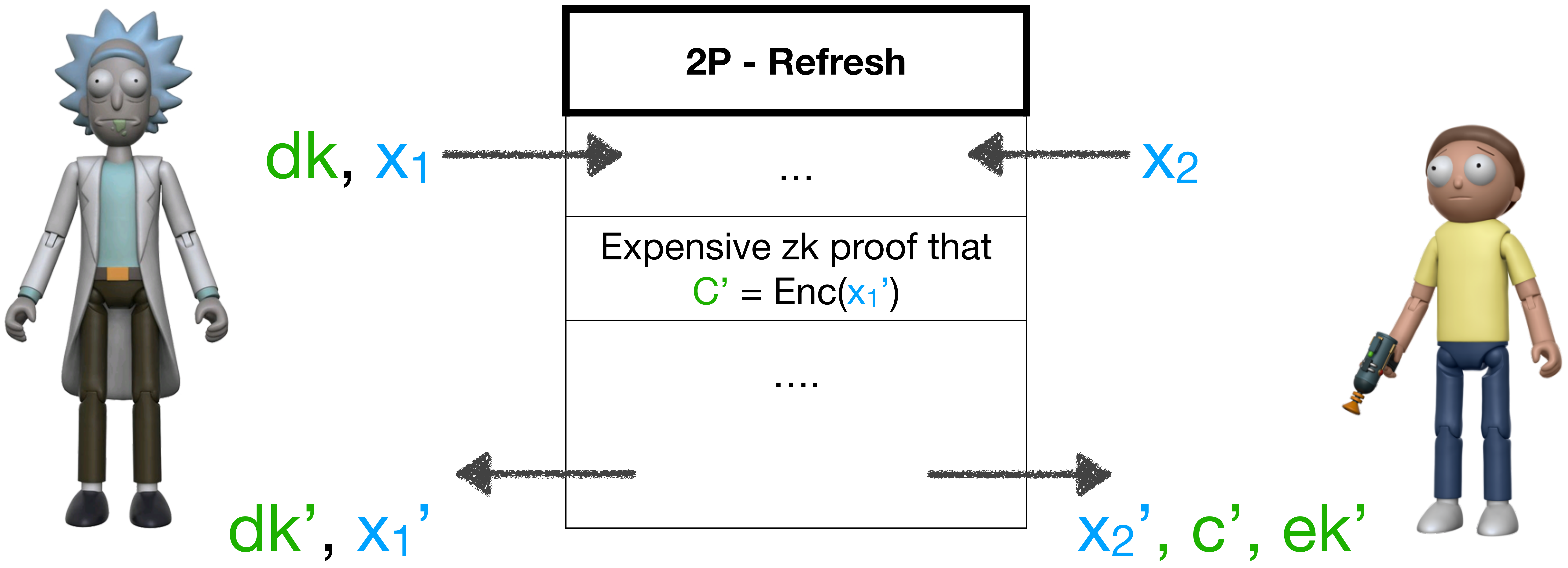


Lather, Rinse, Repeat

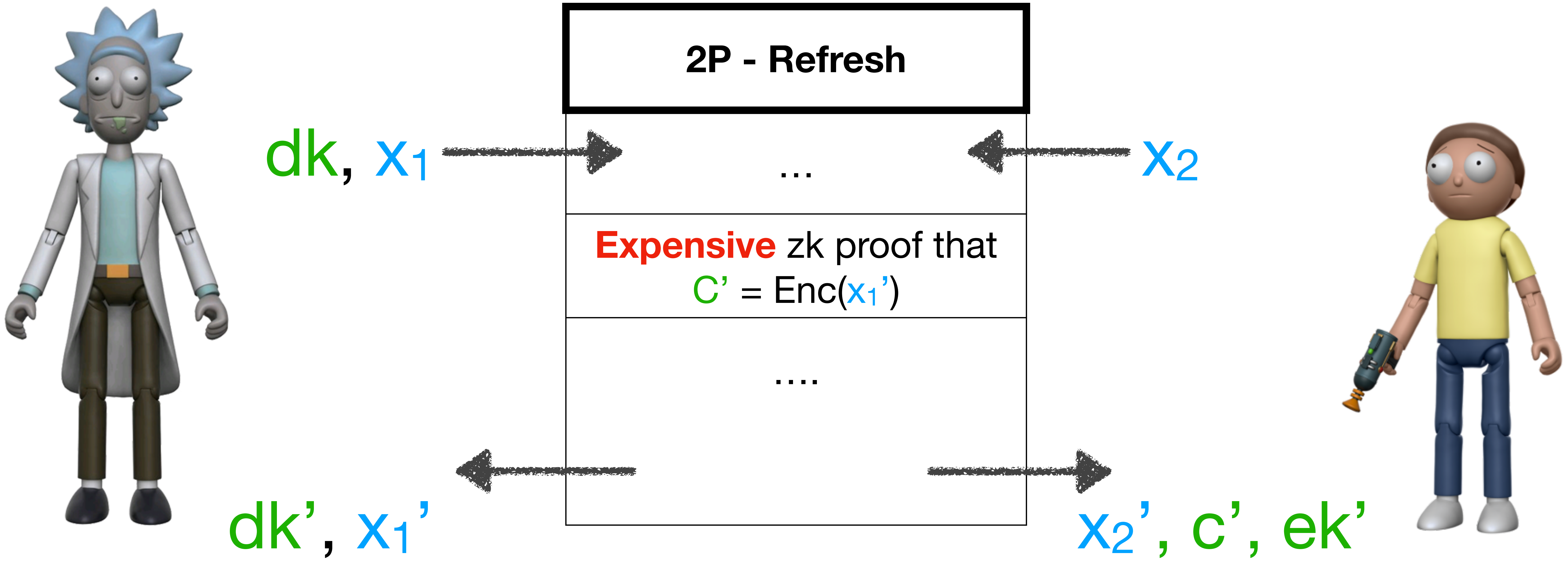
To refresh, a new dk,ek,c must be generated as well!



Lather, Rinse, Repeat



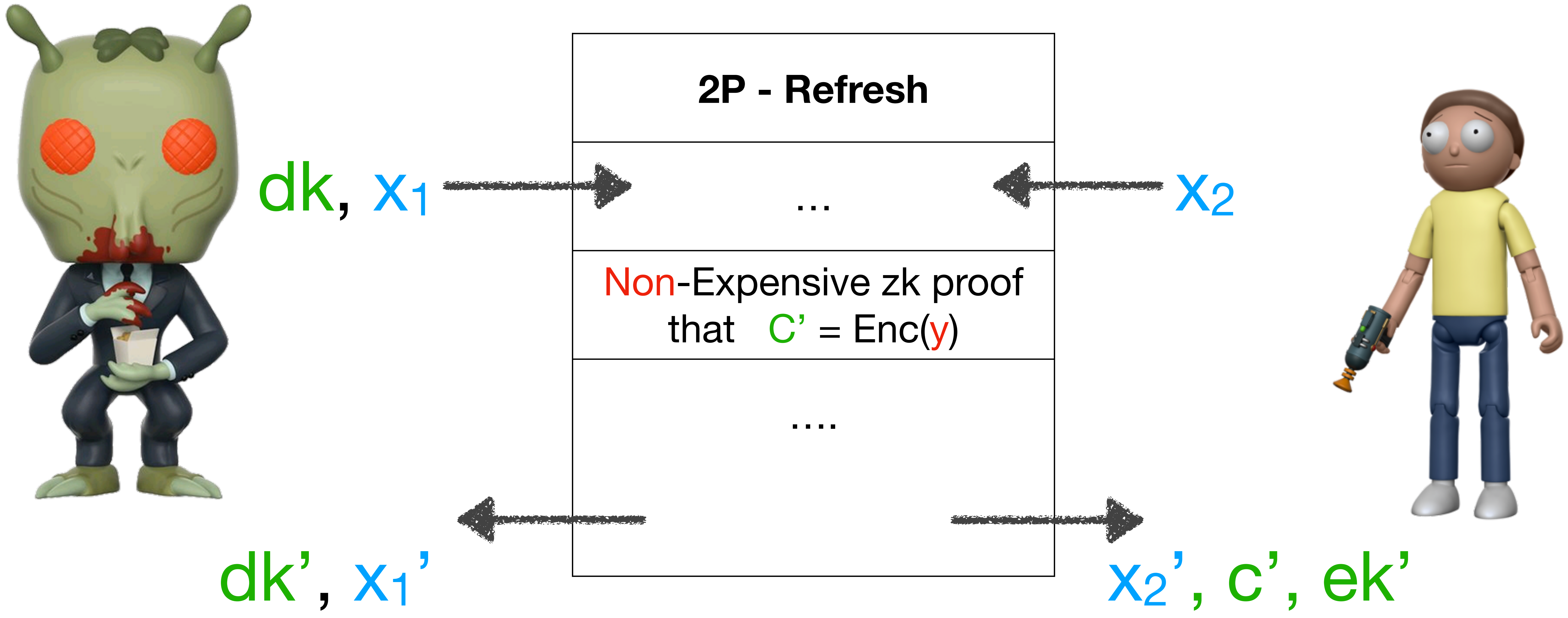
Lather, Rinse, Repeat



Lather, Rinse, Repeat

Idea: Since $x_1' = x_1 + r$, it is enough to prove: $C_{new} = C_{old} \circledast Enc(r)$

Lather, Rinse, Repeat



Lather, Rinse, Repeat

2P - Refresh

2P - Sign

fail/success

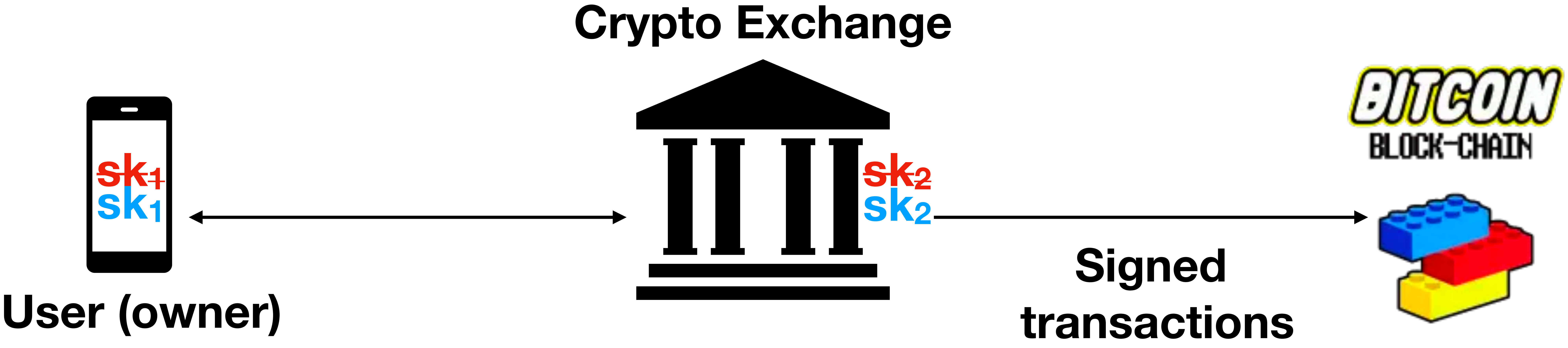
.
.
.

2P - Refresh

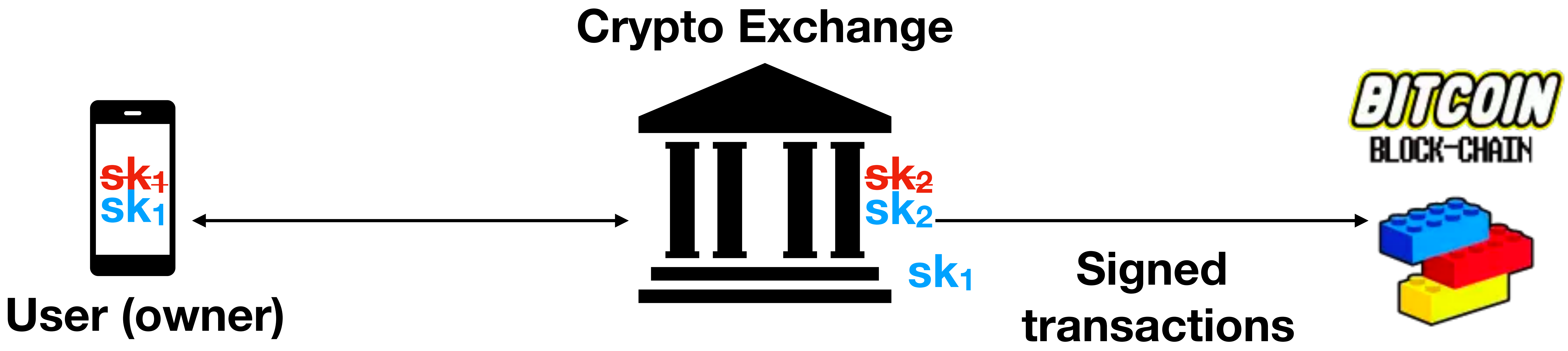
2P - Sign

fail/success

Lather, Rinse, Repeat

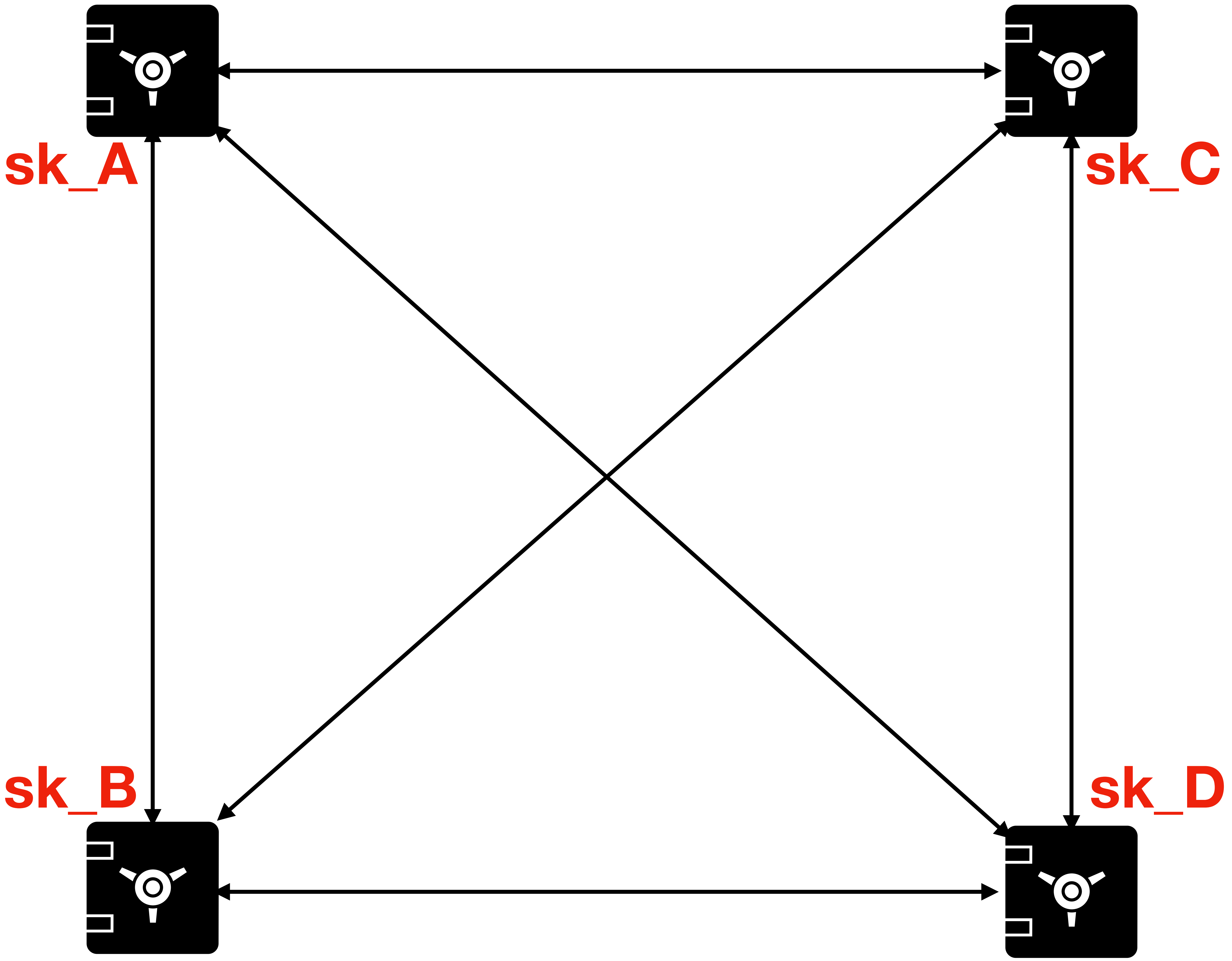


Lather, Rinse, Repeat



Lather, Rinse, Repeat

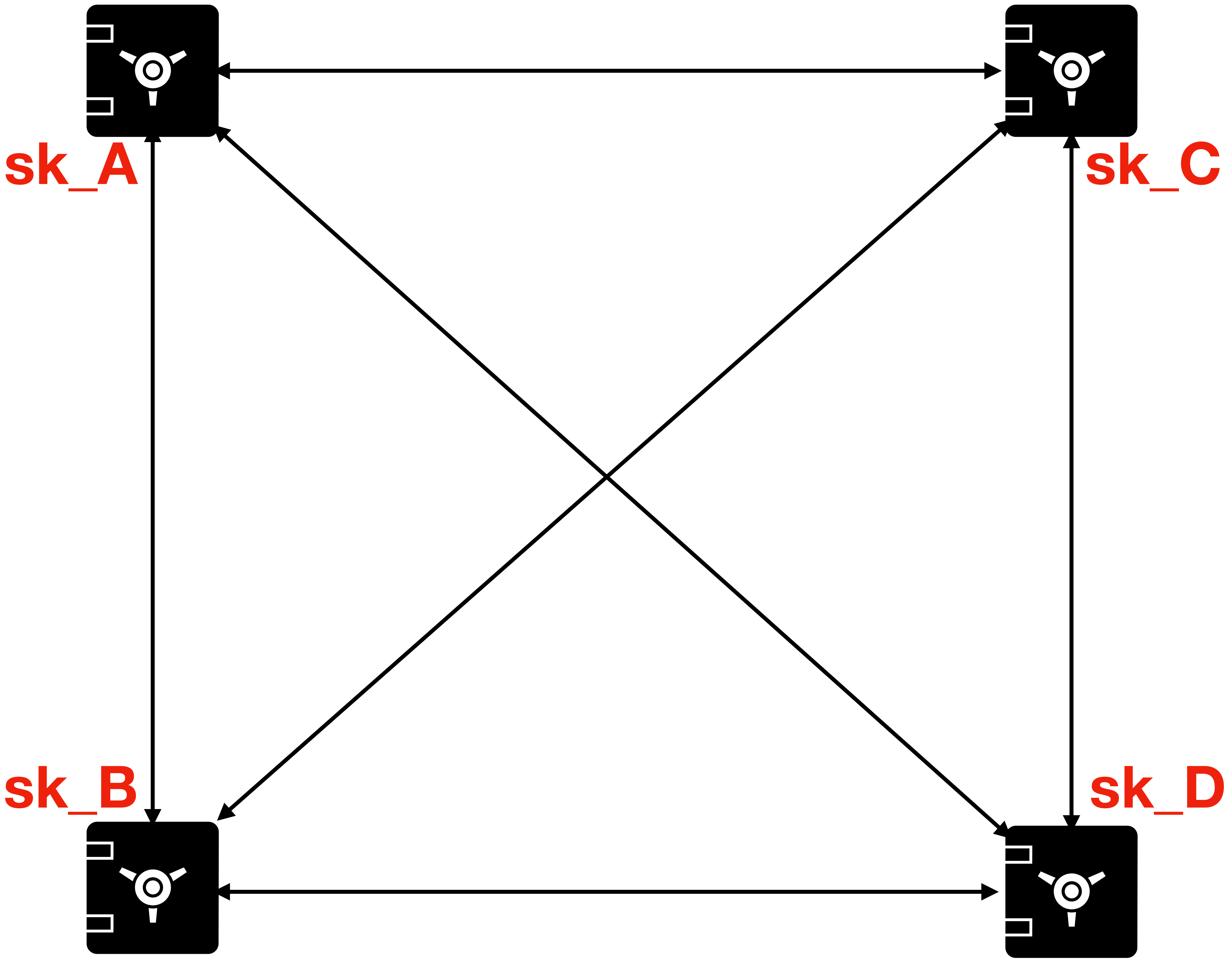
Golden Shoe



Golden Shoe

Rule in real-world interactive protocols:

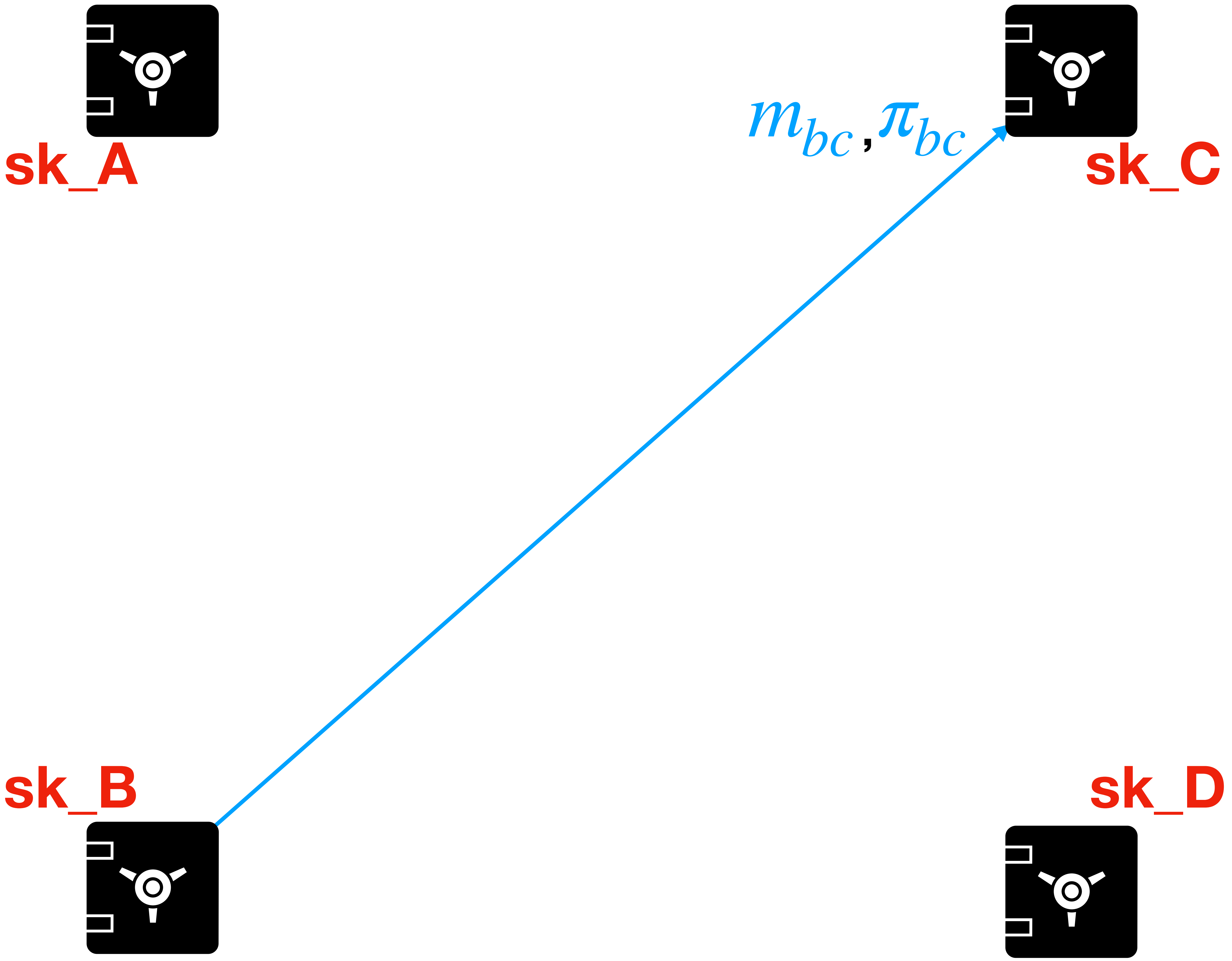
Every message received must be tested to correctly follow the protocol



Golden Shoe

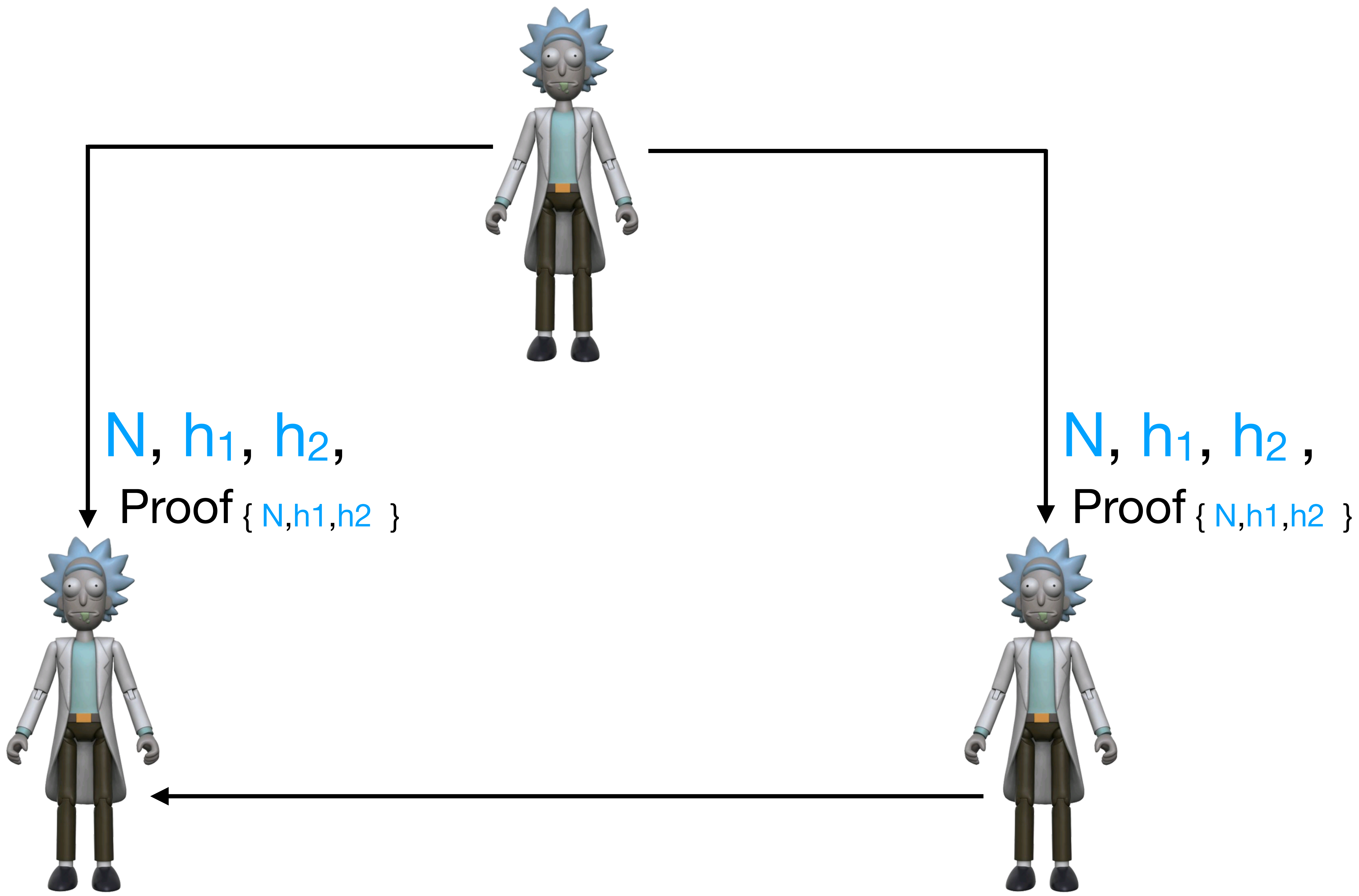
Rule in real-world interactive protocols:

Every message received must be tested to correctly follow the protocol



Golden Shoe

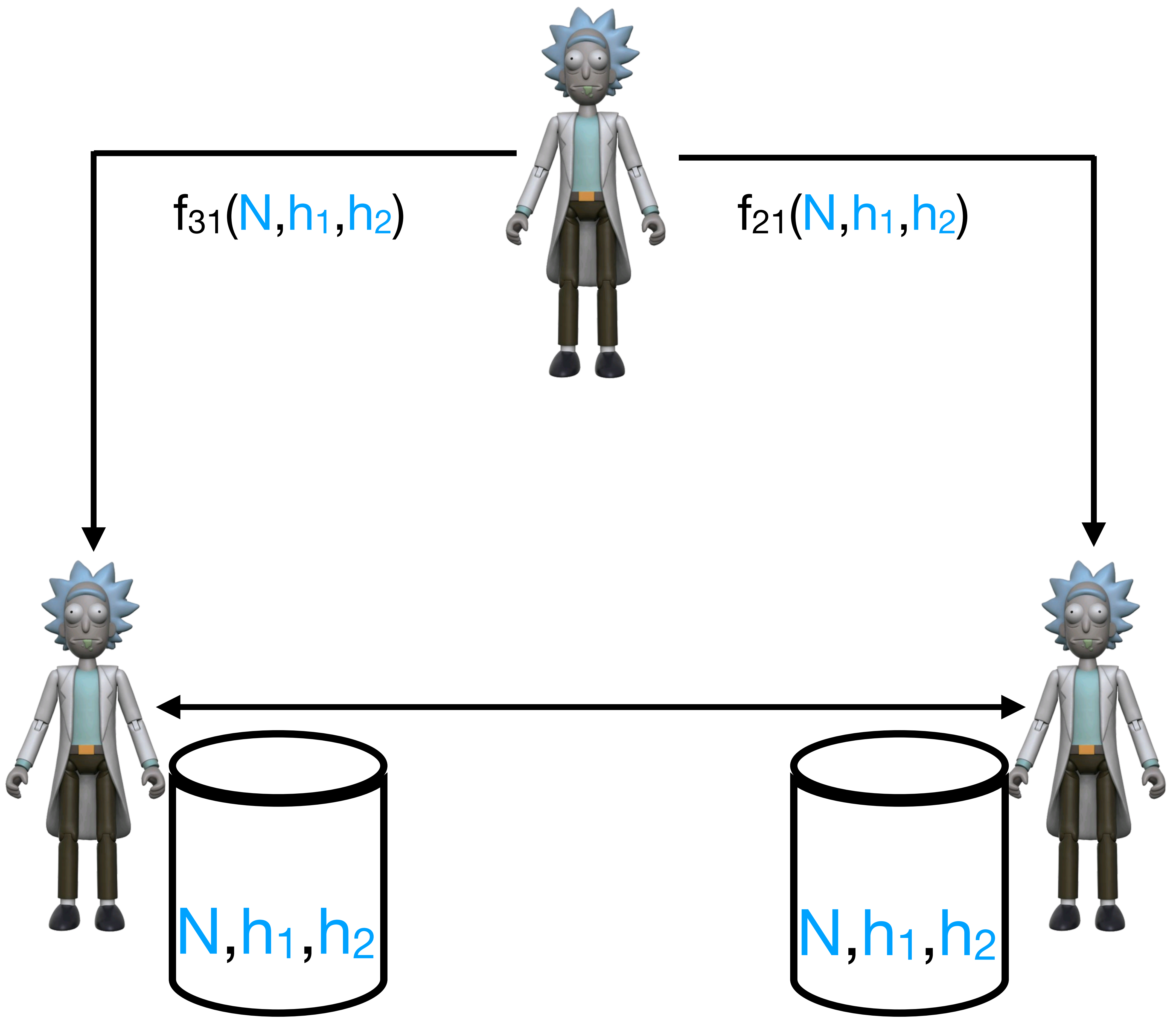
3P - Key Gen black box
…
$\{N, h_1, h_2\}$, Proof $_{\{N,h1,h2\}}$
…



Golden Shoe

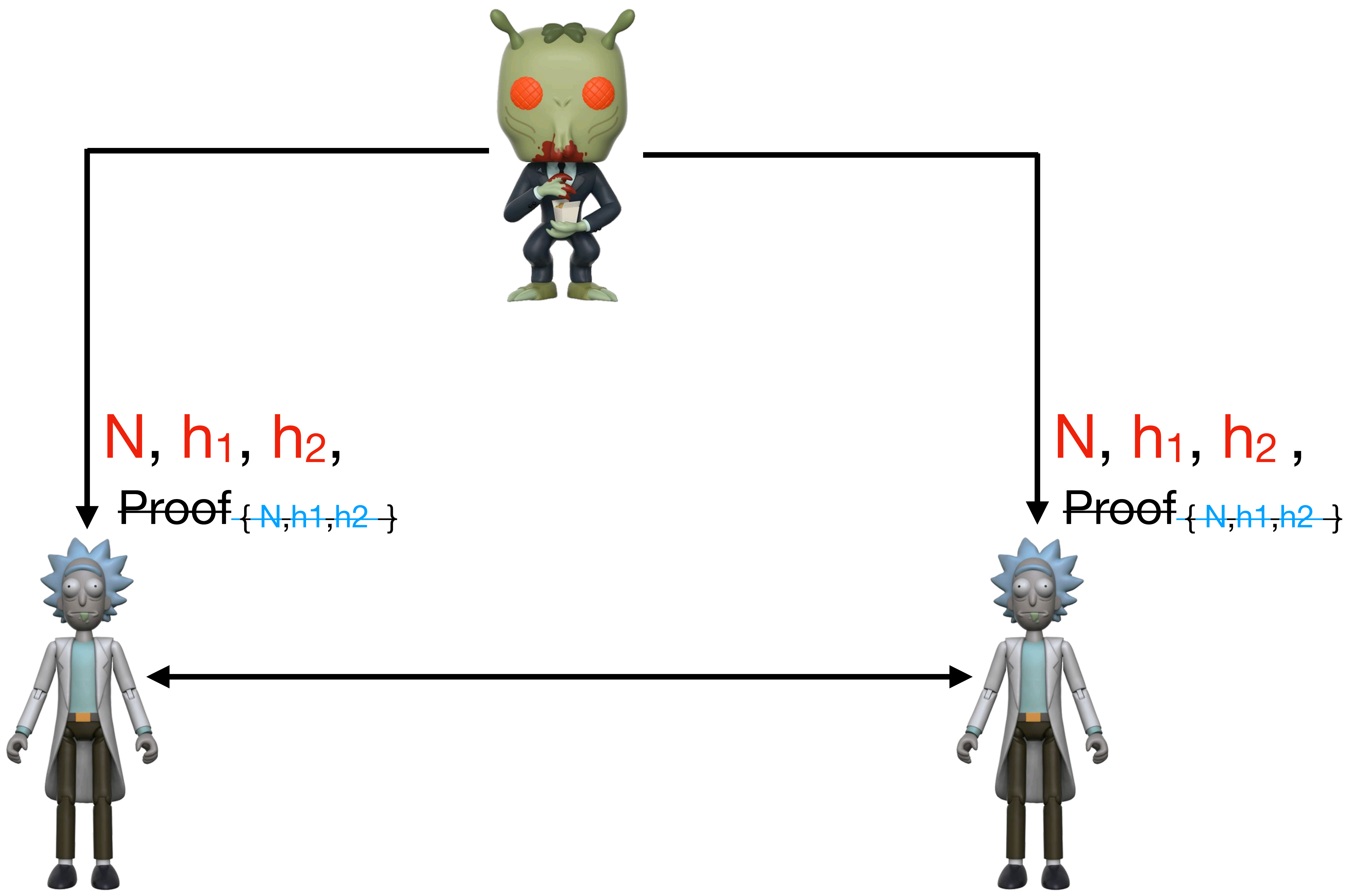
3P - Key Gen black box
…
$\{N, h_1, h_2\}$, Proof$_{\{N, h_1, h_2\}}$
…

2P - Sign
…
$f_{31}(N, h_1, h_2)$, $f_{21}(N, h_1, h_2)$
…



Golden Shoe

3P - Key Gen black box
…
$\{N, h_1, h_2\}$, ~~Proof~~ $_{\{N, h1, h2\}}$
…



Golden Shoe

3P - Key Gen black box

…

$\{N, h_1, h_2\}$, ~~Proof $_{\{N,h1,h2\}}$~~

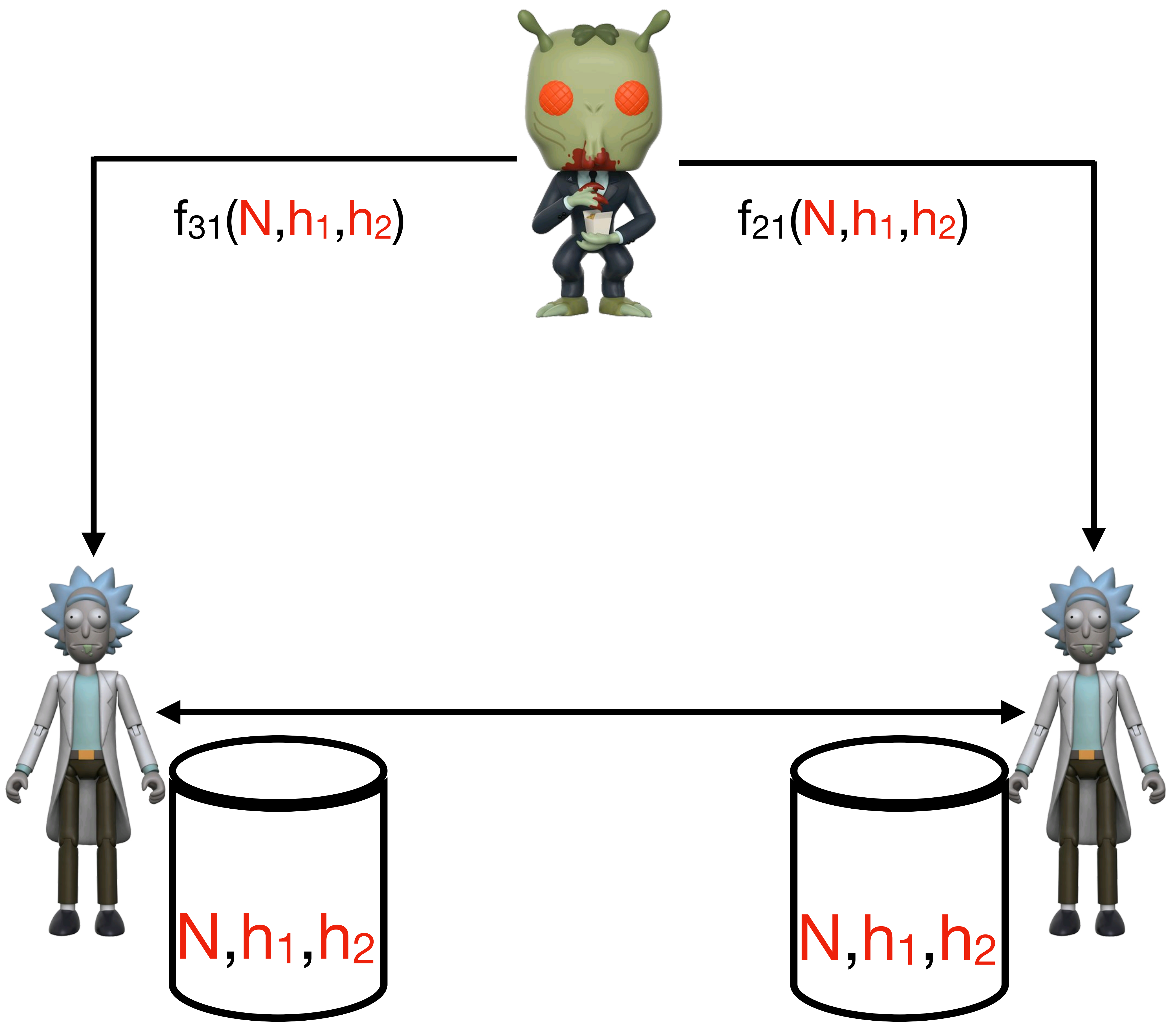
…

2P - Sign

…

$f_{31}(N, h_1, h_2)$, $f_{21}(N, h_1, h_2)$

…

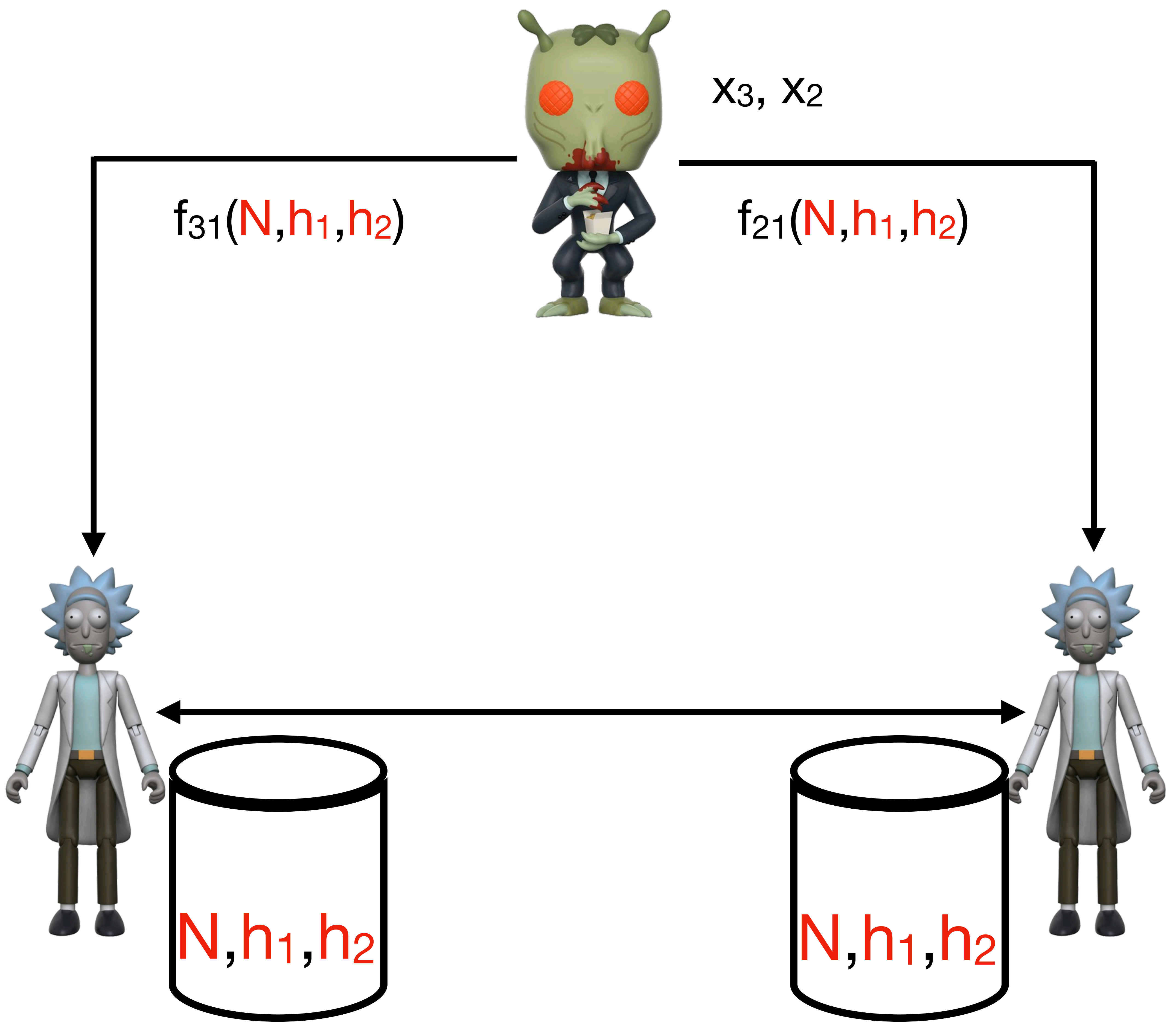


Golden Shoe

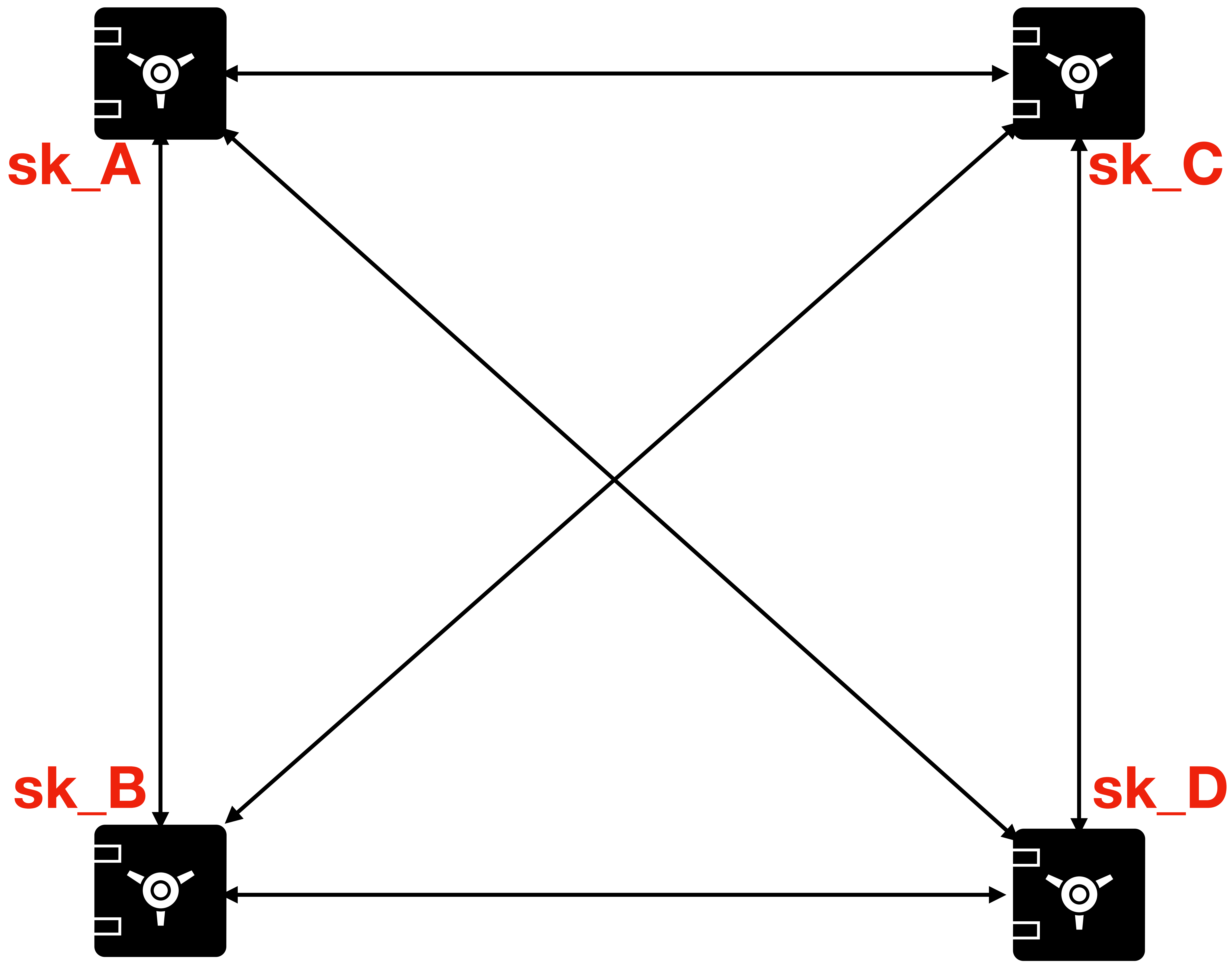
3P - Key Gen black box
…
{N,h_1,h_2} , ~~Proof$_{\{N,h1,h2\}}$~~
…

2P - Sign
…
$f_{31}(N,h_1,h_2)$, $f_{21}(N,h_1,h_2)$
…

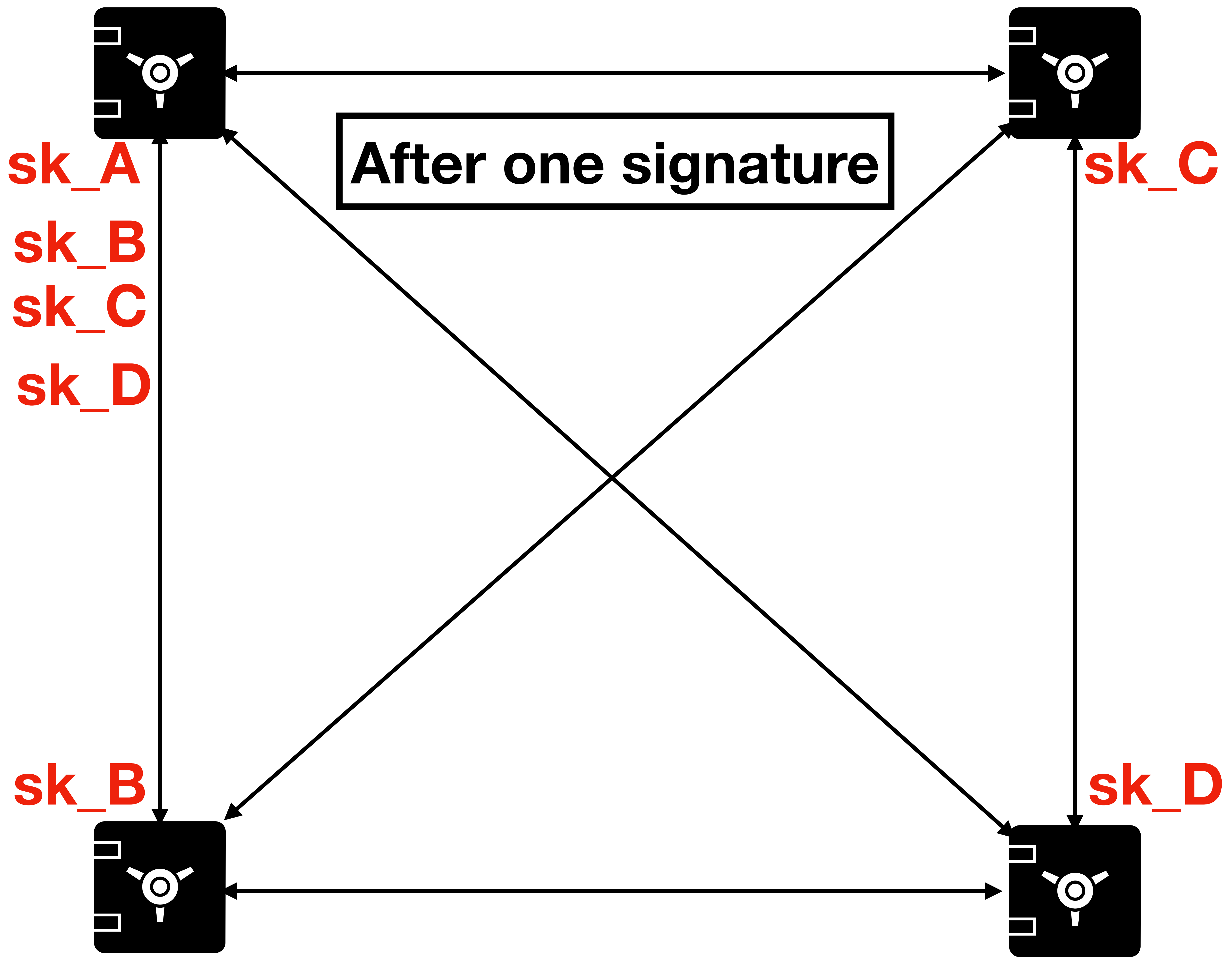
x_3, x_2 ← $f_{31}(N,h_1,h_2)$, $f_{21}(N,h_1,h_2)$



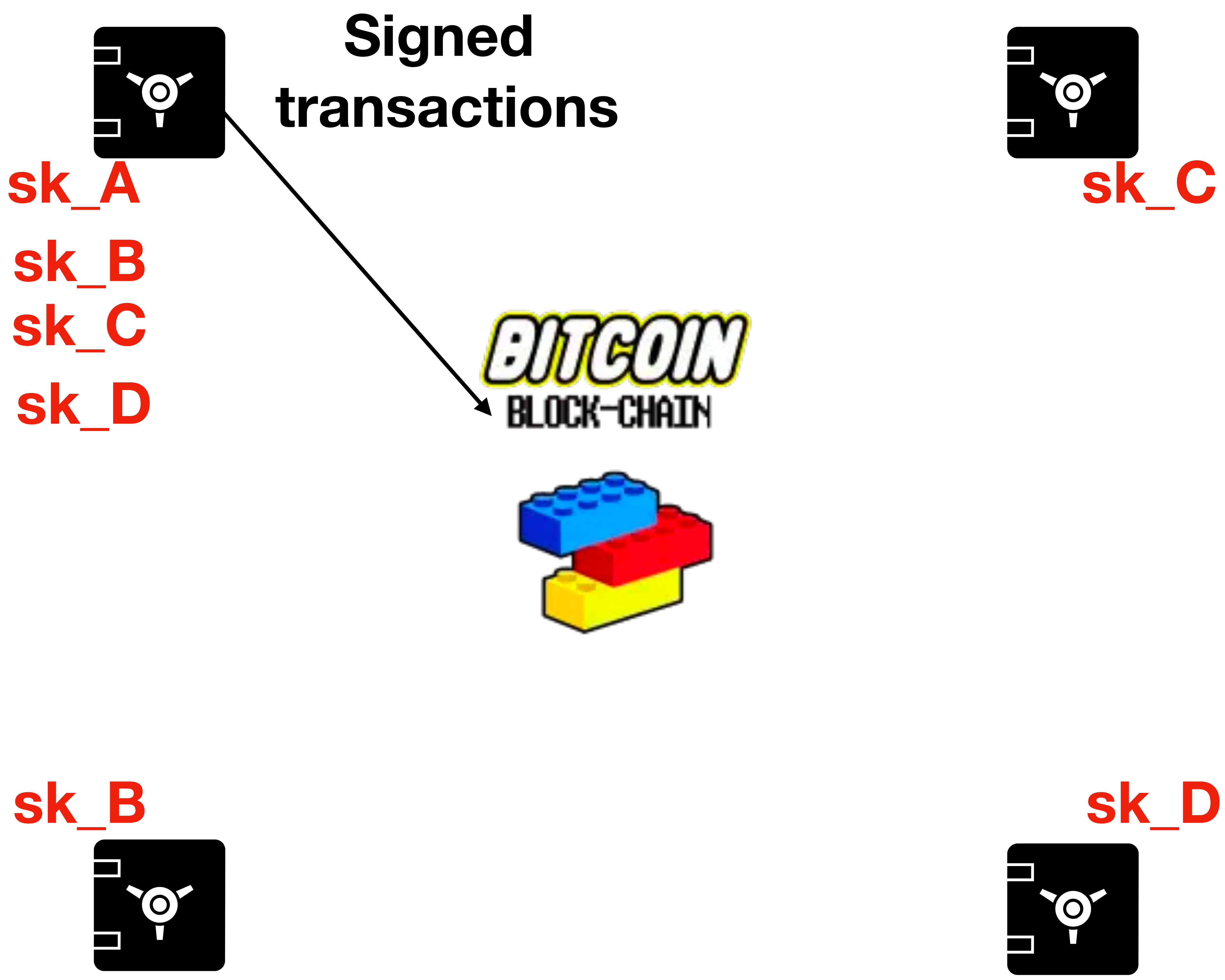
Golden Shoe



Golden Shoe



Golden Shoe


Signed
transactions
sk_A
sk_B
sk_C
sk_D
BITCOIN
BLOCK-CHAIN
sk_C
sk_B
sk_D

Recommendations

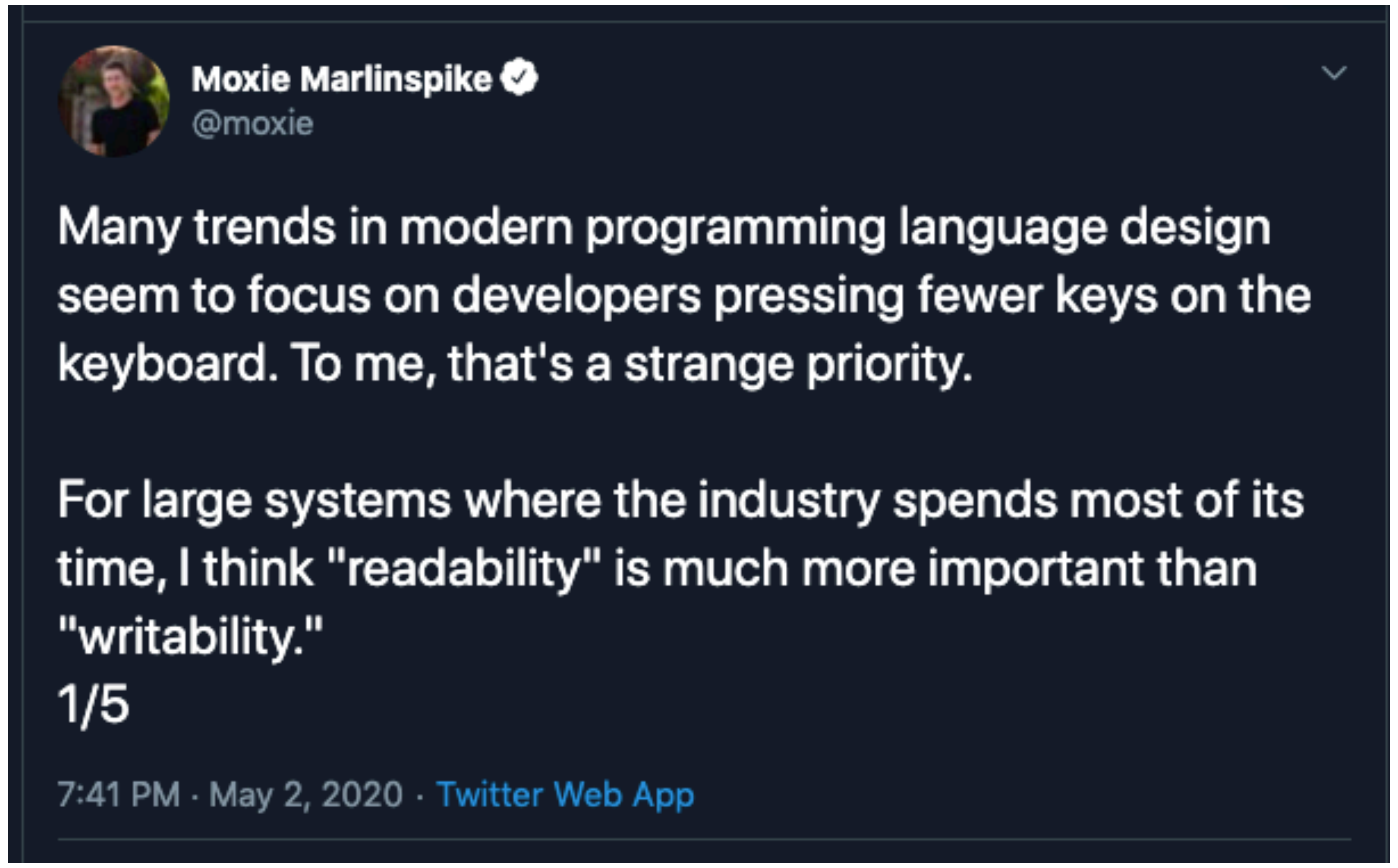
Minimize complexity

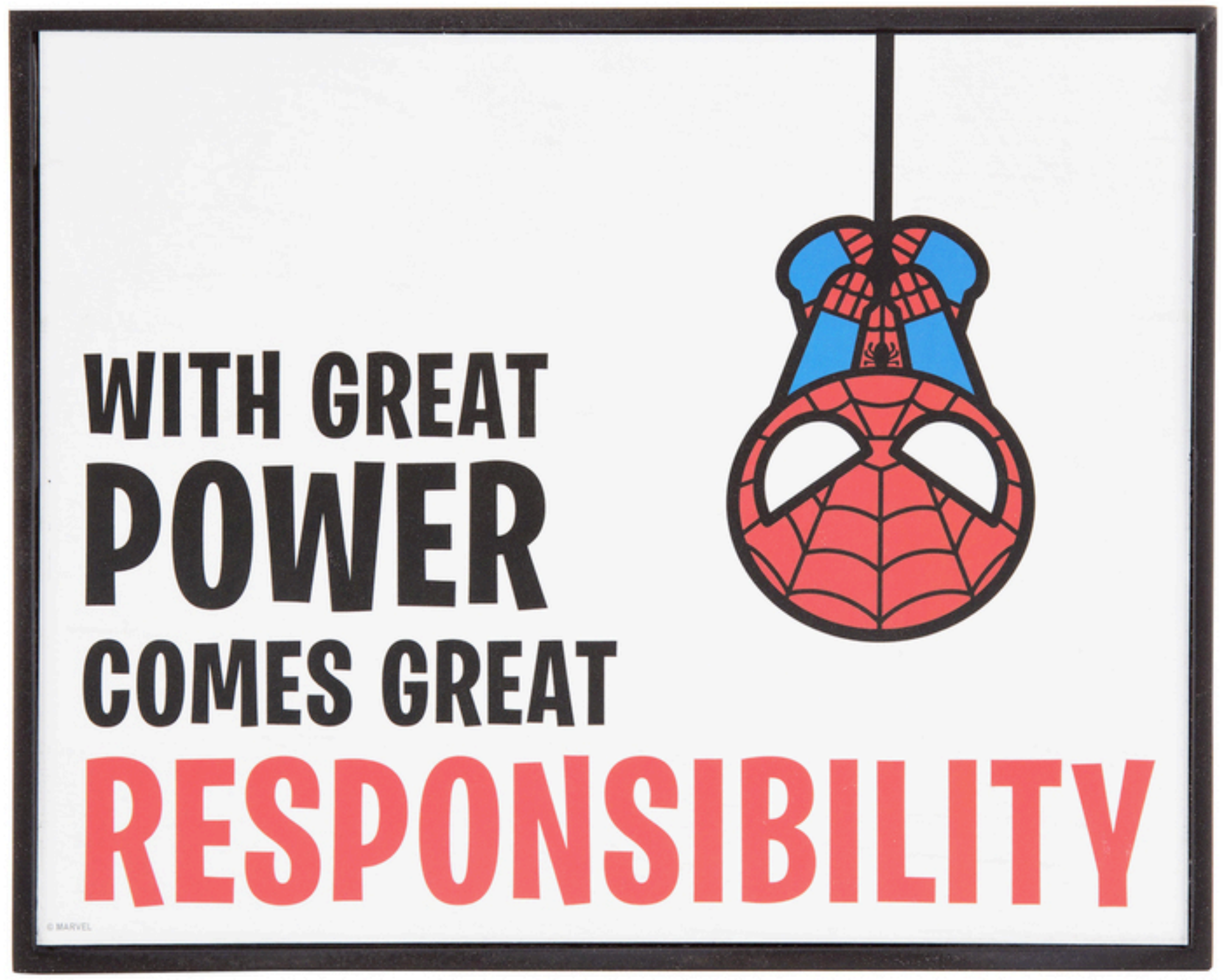
Advanced crypto is really cool and powerful (ZKP, MPC, TSS, pairings, etc.)

Modern languages like Rust, Haskell, or C++17 are cool and powerful too

But their complexity and the skills required to understand them, can also act as obfuscation layers hiding subtle bugs

Example: Zcash 2019 bug https://electriccoin.co/blog/zcash-counterfeiting-vulnerability-successfully-remediated/





Careful with academic papers

Implementing a scheme proven secure on paper might still lead to security disasters, because of:

- Flaws in components not specified in the paper (encodings, parsing, etc.)
- Checks for insecure parameters not in the paper and not implemented
- Incomplete or confusing definition

Example from a recent audit:

In a **zero-knowledge factorization proof**, misunderstanding of “Common Input” lead to trivially forgeable proofs.

Spoiler: *N* must not be selected only by the prover, otherwise it can pick *M* and determine the corresponding *N*

Theorem 1 *The non-interactive proof system defined by*

- Common Input: N
- Random Input: $x \in Z_N^*$
- Prover: *compute* $M = N^{-1} \bmod \phi(N)$ *and output* $y = x^M \bmod N$
- Verifier: *accept iff* $y^N = x \bmod N$.

is one-sided error perfect zero-knowledge with soundness error at most $1/d$ *for the language* SF', *where* d *is the smallest factor of* N.

https://dl.acm.org/doi/10.1145/288090.288108

Should I use MPC and TSS?

Depends on your requirements: wallet types, environment, etc.

MPC and TSS offer high assurance on paper thanks to math proofs, but remain susceptible to misimplementations or overlooked threat vectors

Reduce the risk by relying on trusted solutions, established protocols, audited code bases, and distribute trust across different platforms/hardware systems

Multiple Bugs in Multi-Party Computation: Breaking Cryptocurrency's Strongest Wallets

More details in the associated paper

Omer Shlomovits
omer@kzencorp.com

ZenGo

JP Aumasson
jpa@pm.me

KUDELSKI SECURITY

TAURUS