

*Peer-to-peer Affine Commitment  
using Bitcoin*

Karl Crary and **Michael J. Sullivan**

Carnegie Mellon University  
PLDI '15, Portland

June 17, 2015

*Massively Multiplayer Online  
Linear Logic*

Karl Crary and **Michael J. Sullivan**

Carnegie Mellon University  
PLDI '15, Portland

June 17, 2015

## *Typecoin*

- ▶ A general peer-to-peer commitment mechanism - using the language of linear logic
- ▶ Implemented on top of the Bitcoin network
- ▶ With applications for proof-carrying authorization

## *Proof-carrying authorization*

- ▶ Idea: represent authorization as logical propositions (Appel and Felten 1999)

## *Proof-carrying authorization*

- ▶ Idea: represent authorization as logical propositions (Appel and Felten 1999)
- ▶ ... in a logic with a notion of affirmation
- ▶  $\langle K \rangle A$  means “the principal  $K$  says  $A$ ”

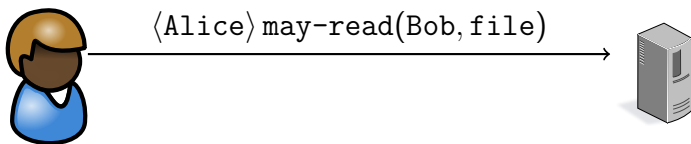
## *Proof-carrying authorization*

- ▶ Alice wants to give access to a file, so affirms:
  - ▶  $\langle \text{Alice} \rangle \text{ may-read}(\text{Bob}, \text{file})$
  - ▶  $\langle \text{Alice} \rangle \text{ may-read}(\text{Charlie}, \text{file})$



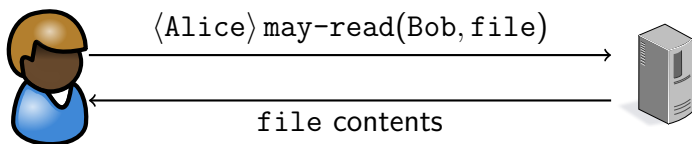
## *Proof-carrying authorization*

- ▶ Alice wants to give access to a file, so affirms:
  - ▶  $\langle \text{Alice} \rangle \text{ may-read}(\text{Bob}, \text{file})$
  - ▶  $\langle \text{Alice} \rangle \text{ may-read}(\text{Charlie}, \text{file})$



## *Proof-carrying authorization*

- ▶ Alice wants to give access to a file, so affirms:
  - ▶  $\langle \text{Alice} \rangle \text{ may-read}(\text{Bob}, \text{file})$
  - ▶  $\langle \text{Alice} \rangle \text{ may-read}(\text{Charlie}, \text{file})$





*Proof-carrying authorization - higher order use*

- ▶ Much more flexible policies are possible:

$$\langle \text{Alice} \rangle \forall K. \langle \text{Registrar} \rangle \text{in-Alice's-class}(K) \\ \supset \text{may-read}(K, \text{file})$$

*Proof-carrying authorization - higher order use*

- ▶ Much more flexible policies are possible:

$$\langle \text{Alice} \rangle \forall K. \langle \text{Registrar} \rangle \text{in-Alice's-class}(K) \\ \supset \text{may-read}(K, \text{file})$$

- ▶ Then can derive:

$$\forall K. \langle \text{Registrar} \rangle \text{in-Alice's-class}(K) \\ \supset \langle \text{Alice} \rangle \text{may-read}(K, \text{file})$$

## *Implementing proof-carrying authorization*

- ▶ Straightforward to make work even in a decentralized/peer-to-peer system
- ▶ Proofs are self-contained
- ▶ Digital signatures used for affirmation

## *Consumable credentials*

What if we want *one time use* authorization?



## *Linear logic*

- ▶ Garg et al. 2006; *linear* proof-carrying authorization
- ▶ Linear logic treats hypotheses as scarce resources that must be used once

*For logicians*

Linear logic allows *exchange*, but not *weakening* or *contraction*

## *Linear logic*

- ▶ Garg et al. 2006; *linear* proof-carrying authorization
- ▶ Linear logic treats hypotheses as scarce resources that must be used once
- ▶ Good for modeling state change:

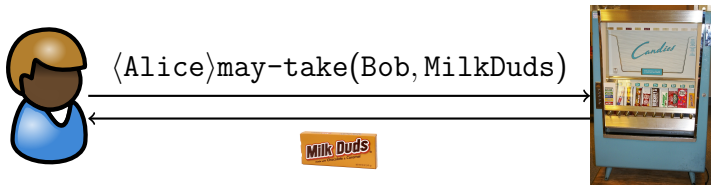
bread  $\otimes$  ham  $\multimap$  ham sandwich

$\forall i. \text{counter}(i) \multimap \text{counter}(i + 1)$

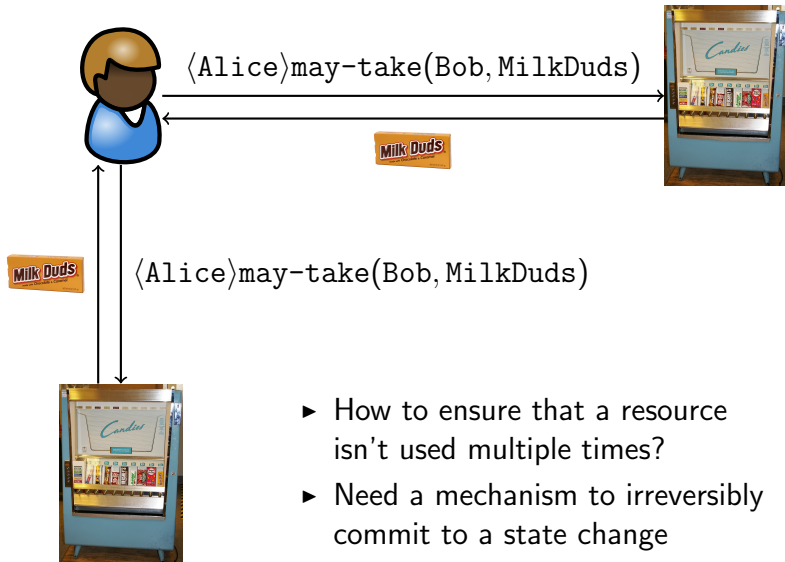
*For logicians*

Linear logic allows *exchange*, but not *weakening* or *contraction*

*Linear authorization*



## Linear authorization



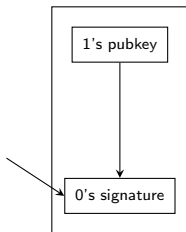


## *Bitcoin*

- ▶ On a completely different note: consider designing a decentralized digital currency

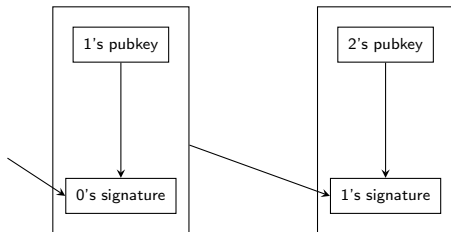
## *Bitcoin*

- ▶ On a completely different note: consider designing a decentralized digital currency
- ▶ A coin is a chain of digital certificates
- ▶ A coin is spent by signing it over to somebody else



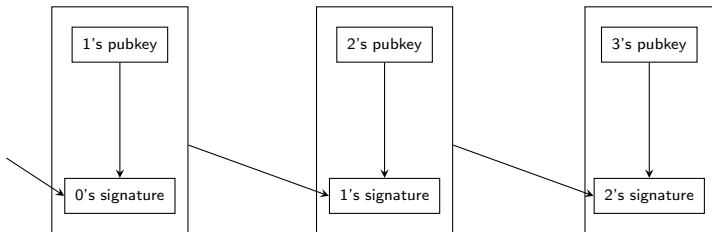
## *Bitcoin*

- ▶ On a completely different note: consider designing a decentralized digital currency
- ▶ A coin is a chain of digital certificates
- ▶ A coin is spent by signing it over to somebody else

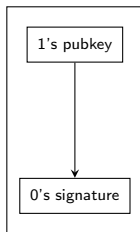


## *Bitcoin*

- ▶ On a completely different note: consider designing a decentralized digital currency
- ▶ A coin is a chain of digital certificates
- ▶ A coin is spent by signing it over to somebody else

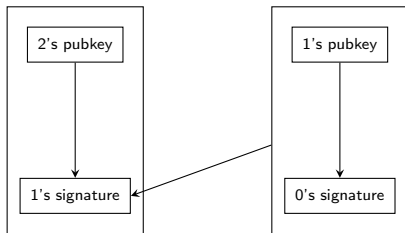


## *Bitcoin - the catch*



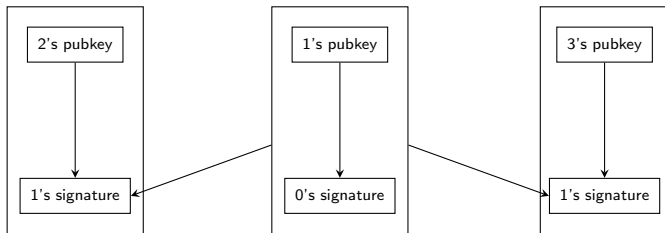
- ▶ But how do we prevent an owner from spending a coin multiple times?

## *Bitcoin - the catch*



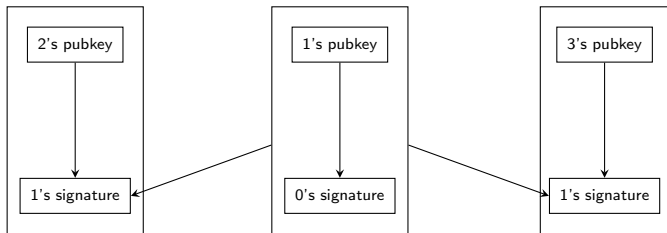
- ▶ But how do we prevent an owner from spending a coin multiple times?

## *Bitcoin - the catch*



- ▶ But how do we prevent an owner from spending a coin multiple times?

## *Bitcoin - the catch*



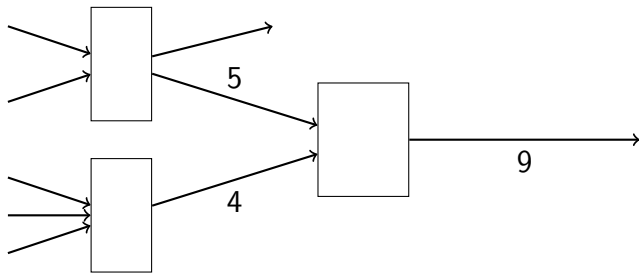
- ▶ But how do we prevent an owner from spending a coin multiple times?
- ▶ Need a mechanism to irreversibly commit to a state change



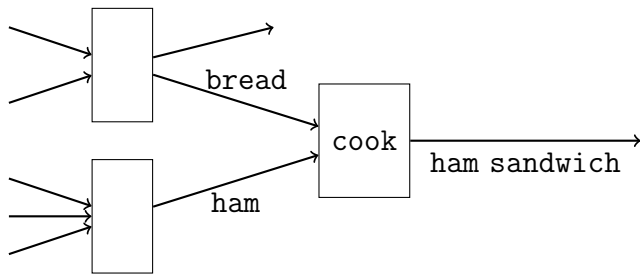
## *Bitcoin implementation*

- ▶ Bitcoin (Nakamoto 2008) does this with a global ledger of all transactions - the “blockchain”
- ▶ Ledger maintained by distributed process called “mining”

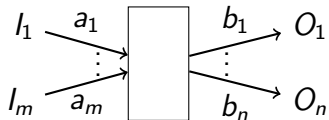
# *From Bitcoin to Typecoin*



*From Bitcoin to Typecoin*

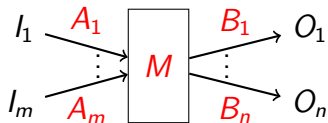


*From Bitcoin to Typecoin - transactions*



►  $a_1 + \cdots + a_m = b_1 + \cdots + b_n$

## *From Bitcoin to Typecoin - transactions*



- ▶  $\vdash M : (A_1 \otimes \cdots \otimes A_m) \multimap (B_1 \otimes \cdots \otimes B_n)$
- ▶ Carry linear logic<sup>1</sup> propositions instead of numbers

---

<sup>1</sup>actually affine logic

# *Authorization example*



## Authorization example

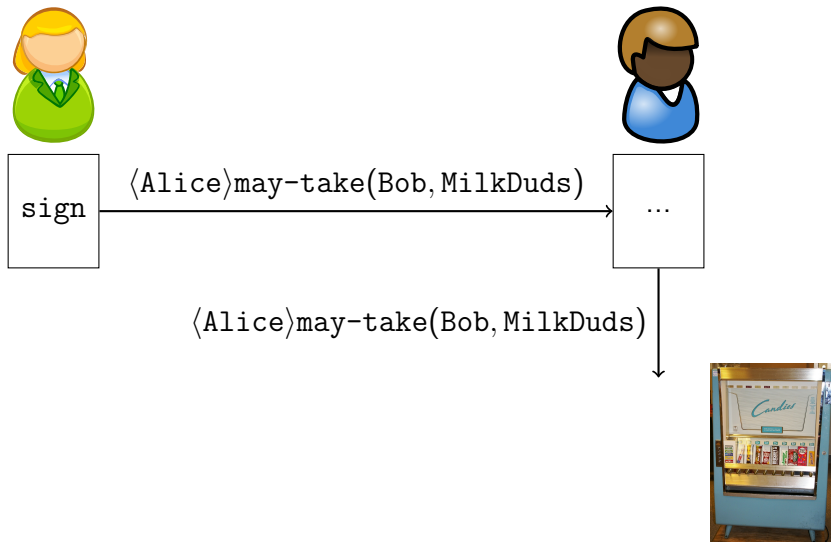


sign

$\langle \text{Alice} \rangle \text{may-take}(\text{Bob}, \text{MilkDuds})$

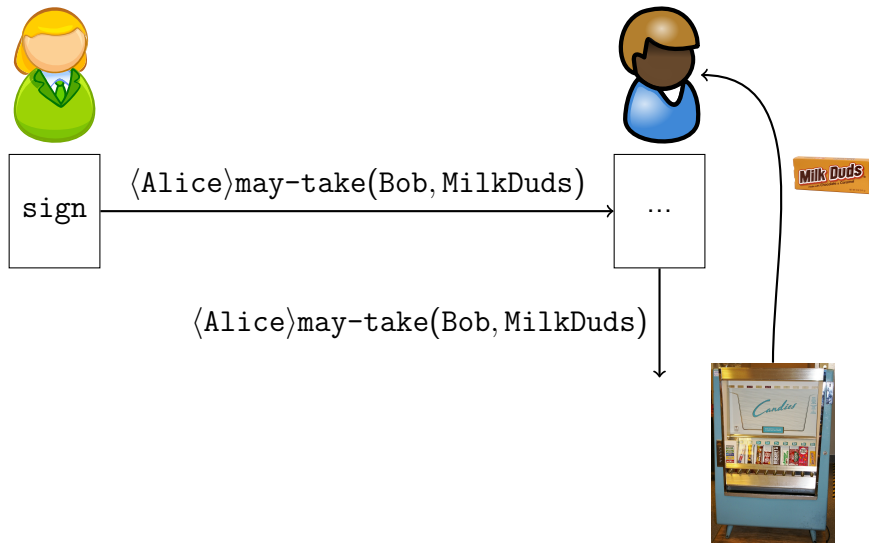


## Authorization example

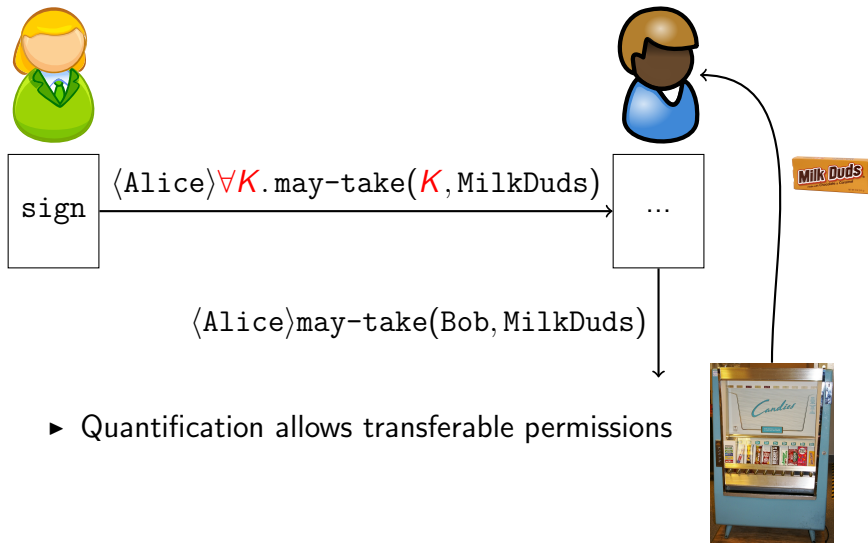




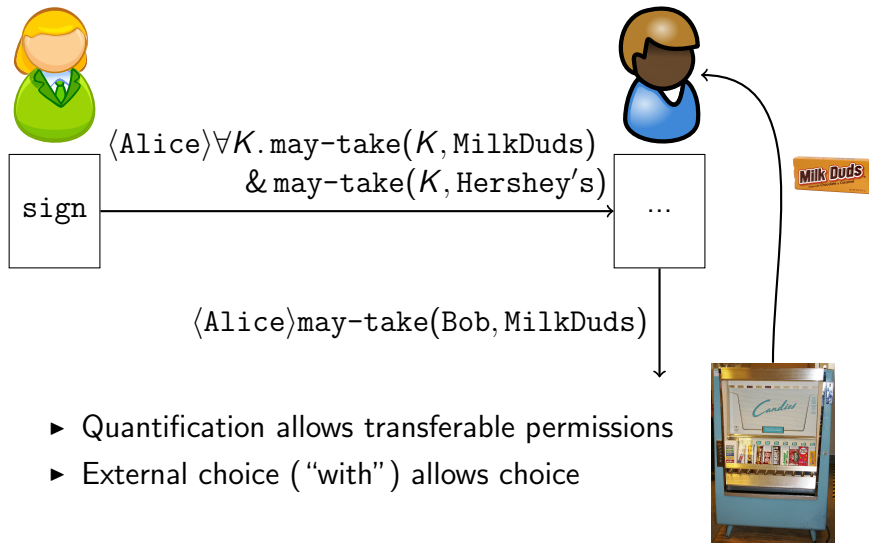
# Authorization example



## Authorization example



## Authorization example



- ▶ Quantification allows transferable permissions
- ▶ External choice ("with") allows choice

## *Declarations*

- ▶ Where do may-take, MilkDuds, etc. come from?

## *Declarations*

- ▶ Where do `may-take`, `MilkDuds`, etc. come from?
- ▶ Transactions can declare types and propositions

`may-take` : `principal`  $\rightarrow$  `candy`  $\rightarrow$  `prop`

## *Building a new currency*

- ▶ Can turn Typecoin back into a currency (S-coins)

coin : nat  $\rightarrow$  prop

merge :  $\forall N, M : \text{nat.}$

coin  $N \otimes$  coin  $M \multimap$  coin  $N + M$

split :  $\forall N, M, P : \text{nat.}$

coin  $N + M \multimap$  coin  $N \otimes$  coin  $M$

## *Central banking*

- ▶ Need some way to mint a new S-coin

`print` : `nat`  $\rightarrow$  `prop`

`issue` :  $\forall N:\text{nat. } \langle \text{Janet} \rangle (\text{print } N) \multimap \text{coin } N$

## *How to implement?*

- ▶ We *could* build Typecoin in a standalone way
- ▶ Use adapted versions of the Bitcoin mining algorithms and protocol
- ▶ Could typecheck transactions before they enter the chain

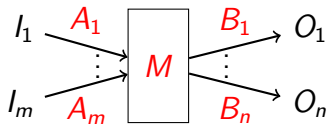


## *How to implement?*

- ▶ How to incentivize people to mine on a Typecoin chain?
- ▶ Bitcoin already has a lot of mining power
- ▶ Typechecking transactions in the chain not an obvious win: proofs might be big or not public

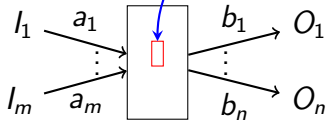
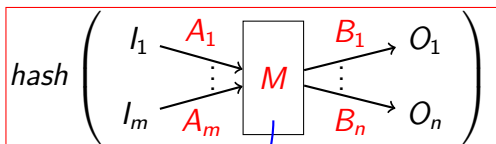
## Overlaying on Bitcoin

- ▶ New plan: actually overlay on top of Bitcoin



## Overlaying on Bitcoin

- ▶ New plan: actually overlay on top of Bitcoin



- ▶ Embed a hash in the metadata of the Bitcoin transaction
- ▶ Send the Typecoin transactions to interested parties

## *Metadata in Bitcoin*

- ▶ Bitcoin historically lacked a nice place to put metadata - on principle
- ▶ (Nodes would not forward transactions that used the straightforward methods)
- ▶ Paper describes a somewhat hacky workaround

## *Metadata in Bitcoin*

- ▶ Bitcoin historically lacked a nice place to put metadata - on principle
- ▶ (Nodes would not forward transactions that used the straightforward methods)
- ▶ Paper describes a somewhat hacky workaround
- ▶ But the Bitcoin developers have since caved

## Receipts

- ▶ Receipts that attest to outputs:  $\text{receipt}(A \rightarrow \text{addr})$

$$\langle \text{Alice} \rangle (\text{receipt}(\text{coin}(5) \rightarrow \text{Alice}) \multimap \forall K. \text{may-take}(K, \text{MilkDuds}))$$

## *Expiration/revocation*

- ▶ Conditional modality permits revocation and expiration:  
if(before(July 10)), may-write(Alice, POPL-paper))

## *Implementation*

- ▶ Implemented in Standard ML
- ▶ With a new Bitcoin client, in SML



## *Related Work*

- ▶ Bowers et al. 2007; consumable credentials
- ▶ Rosenfeld 2013; colored coins
- ▶ Wood 2014; Ethereum

## *Conclusion*

- ▶ Typecoin is a flexible peer-to-peer logical commitment mechanism
- ▶ Based on generalizing Bitcoin to carry logical propositions
- ▶ Actually implemented on top of Bitcoin
- ▶ Details on the logic are in the paper

Thank you!

## *Why not linear?*

- ▶ Typecoin sort of *fundamentally* affine - can always throw away an output
- ▶ Allowing rule declarations in signatures makes it trivial
- ▶ `trash :  $\top \multimap 1$`

## *Why not linear?*

- ▶ Typecoin sort of *fundamentally* affine - can always throw away an output
- ▶ Allowing rule declarations in signatures makes it trivial
- ▶  $\text{trash} : \top \multimap 1$
- ▶ Prohibit  $\top$ ?  $\text{trash} : A \multimap 1$
- ▶ Prohibit proving 1?  $\text{dummy} : \text{prop. trash} : A \multimap !\text{dummy}$
- ▶ Prohibit consuming A?  $\text{trash} : \langle K \rangle \text{dummy} \multimap !\text{dummy}, \text{sign} \langle K \rangle (A \multimap \text{dummy})$

## *Metadata: “m-of-n” outputs*

- ▶ An “ $m$ -of- $n$ ” output lists  $n$  public keys
- ▶ To spend it, provide signatures using  $m$
- ▶ 2-of-3 outputs useful for two-party escrow
- ▶ We use 1-of-2 outputs to embed metadata
- ▶ One public key is the real destination
- ▶ The other is actually the hash of our transaction