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:
  - $\langle$Alice$\rangle$ may-read(Bob, file)
  - $\langle$Alice$\rangle$ may-read(Charlie, file)
Proof-carrying authorization

- Alice wants to give access to a file, so affirms:
  - ⟨Alice⟩ may-read(Bob, file)
  - ⟨Alice⟩ may-read(Charlie, file)
Proof-carrying authorization

Alice wants to give access to a file, so affirms:

- ⟨Alice⟩ may-read(Bob, file)
- ⟨Alice⟩ may-read(Charlie, 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:
  \[
  \text{bread } \otimes \text{ham } \rightarrow \circ \text{ham sandwich}
  \]
  \[
  \forall i. \text{counter}(i) \rightarrow \circ \text{counter}(i + 1)
  \]

*For logicians*

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

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

▶ How to ensure that a resource isn't used multiple times?
▶ Need a mechanism to irreversibly commit to a state change
Linear authorization

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

- How to ensure that a resource isn't used multiple times?
- Need a mechanism to irreversibly commit to a state change
Bitcoin

- On a completely different note: consider designing a decentralized digital currency
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
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
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
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?
But how do we prevent an owner from spending a coin multiple times?
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

- bread
- ham

cook → ham sandwich
From Bitcoin to Typecoin - transactions

\[ a_1 + \cdots + a_m = b_1 + \cdots + b_n \]
From Bitcoin to Typecoin - transactions

\[ \vdash M : (A_1 \otimes \cdots \otimes A_m) \rightarrow (B_1 \otimes \cdots \otimes B_n) \]

- Carry linear logic\(^1\) propositions instead of numbers

\(^1\)actually affine logic
Authorization example

Alice may take (Bob, MilkDuds)

⟨Alice⟩ ∀ K. may-take (K, MilkDuds)

⟨Alice⟩ ∀ K. may-take (K, Hershey’s)

& may-take (K, Hershey’s)

⟨Alice⟩ may-take (Bob, MilkDuds)

▶ Quantification allows transferable permissions

▶ External choice ("with") allows choice
Authorization example

\[
\langle Alice \rangle \text{may-take}(Bob, \text{MilkDuds})
\]
Authorization example

\[\langle Alice \rangle \text{may-take}(Bob, \text{MilkDuds})\]
Authorization example

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

Quantification allows transferable permissions

External choice ("with") allows choice
Authorization example

$\langle Alice \rangle \forall K. \text{may-take}(K, \text{MilkDuds})$

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

- Quantification allows transferable permissions
Authorization example

\[\langle Alice \rangle \forall K. \text{may-take}(K, \text{MilkDuds}) \land \text{may-take}(K, \text{Hershey's})\]

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

- 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

\[
\text{may-take} : \text{principal} \rightarrow \text{candy} \rightarrow \text{prop}
\]
Building a new currency

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

\[
\begin{align*}
\text{coin} & : \text{nat} \to \text{prop} \\
\text{merge} & : \forall N, M : \text{nat}. \quad \text{coin} N \otimes \text{coin} M \to \text{coin} N + M \\
\text{split} & : \forall N, M, P : \text{nat}. \quad \text{coin} N + M \to \text{coin} N \otimes \text{coin} M
\end{align*}
\]
Need some way to mint a new S-coin

\[
\begin{align*}
\text{print} & : \text{nat} \rightarrow \text{prop} \\
\text{issue} & : \forall N:\text{nat}. <\text{Janet}>(\text{print } N) \rightarrow \text{coin } N
\end{align*}
\]
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

\[ I_1 \xrightarrow{A_1} M \xrightarrow{B_1} O_1 \]
\[ I_m \xrightarrow{A_m} M \xrightarrow{B_n} O_n \]
Overlaying on Bitcoin

- New plan: actually overlay on top of Bitcoin
  
  \[
  \text{hash} \begin{pmatrix}
  I_1 & \cdots & A_1 \\
  I_m & \cdots & A_m \\
  \end{pmatrix}
  \begin{array}{c}
  M \\
  B_1 \\
  B_n
  \end{array}
  \begin{pmatrix}
  O_1 \\
  \cdots \\
  O_n
  \end{pmatrix}
  \]

- 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: receipt($A \rightarrow addr$)

$\langle Alice \rangle (\text{receipt(coin(5)} \rightarrow Alice) \rightarrow \forall K. \text{may-take}(K, \text{MilkDuds}))$
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 : ⊤ → 1
Why not linear?

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