Sprites and State Channels: Payment Networks that Go Faster than Lightning

Suyeong Lee

suyeong.lee@kaist.ac.kr

2019-05-01

Overview

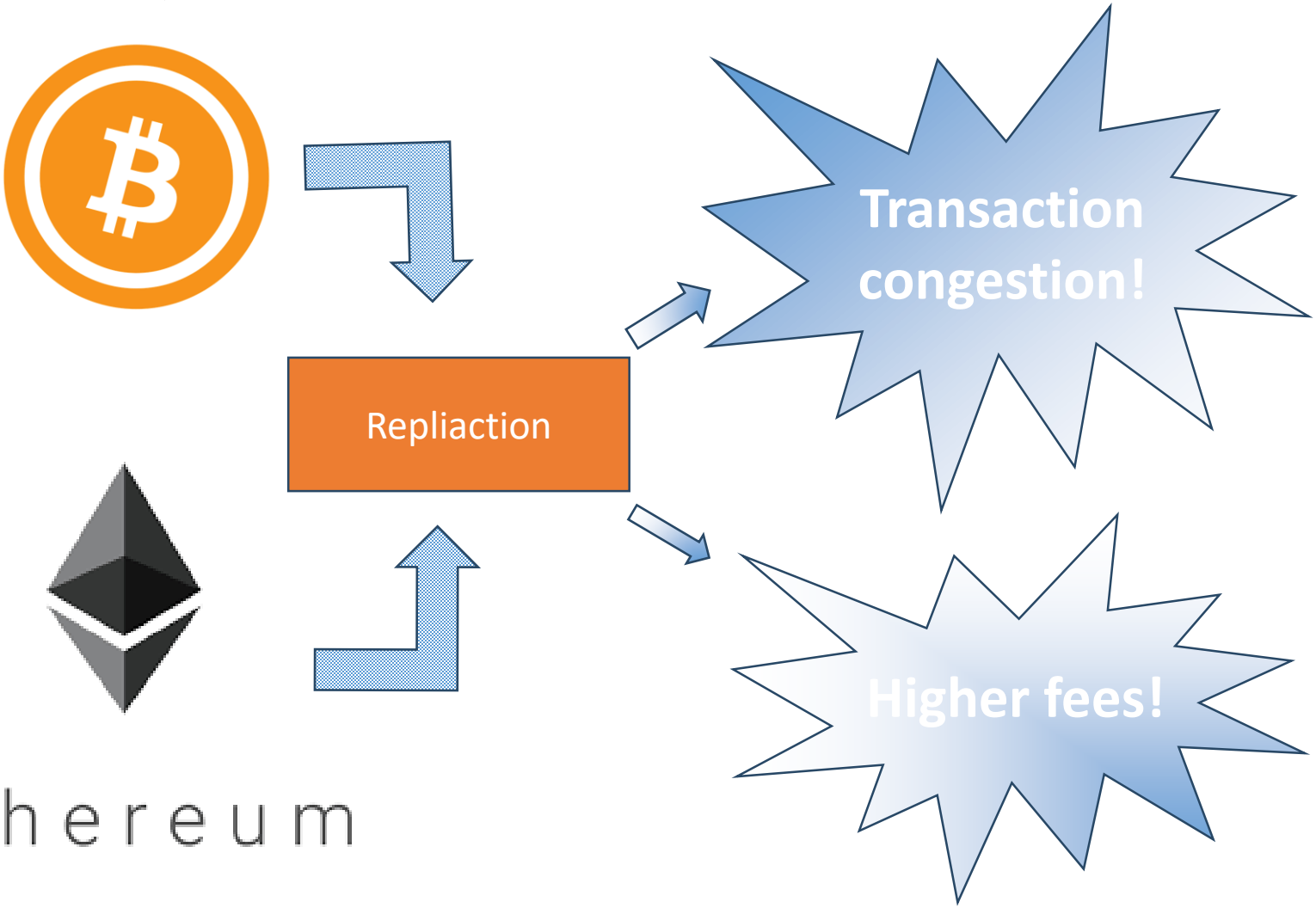
- 1. Introduction
- 2. Background and Preliminaries
- 3. Overview of the Sprites construction
- 4. The State Channel Abstraction
- 5. Linked Payments from State Channels
- 6. Related Works
- 7. Conclusion

1. Introduction

Why Sprites?

1. Introduction

- Bitcoin, Ethereum have definite limitations.



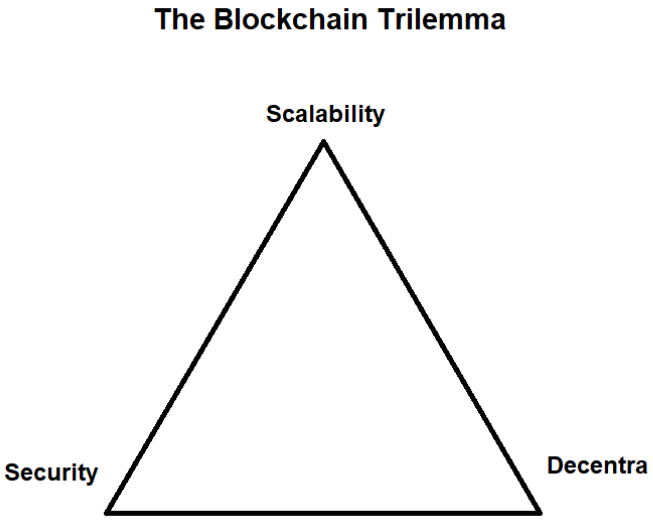
1. Introduction

- Leading proposal to improve the **scalability**.
  - Using "off-chain" rapid payment channels.
  - It require initial deposits of "on-chain" currency.

Scalability

1. Introduction

- Previous class, we learned OmniLedger.....



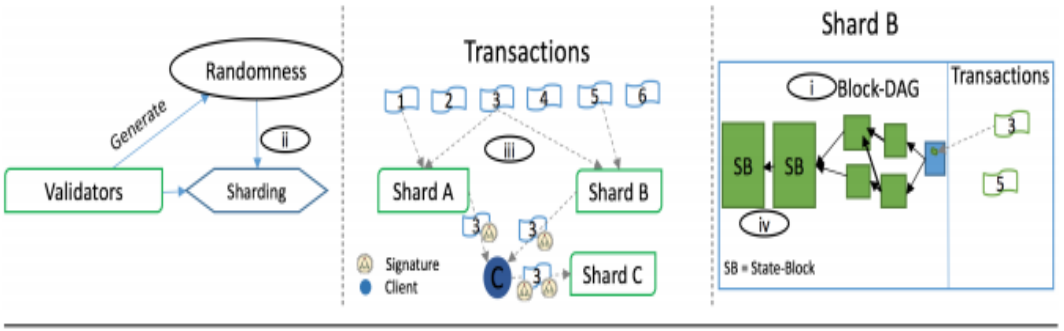


Figure 2. Architectural overview of OmniLedger

Sharding!

1. Introduction

- "**Collateral cost**" of a payment channel
  - Time x money (money is locked up in the smart contract)
  - Sprites improves the worst-case "collateral cost"

Collater = Time x Money

1. Introduction

Collateral Costs in Payment Channles

- Is payment channel networks are **feasible**?
  - ✓ Enough collateral will be available for payments to be routed at high throughput!
  - ✓ "locktime": Reserved money as collateral until the payment is completed.
  - ✓ If parties fail, the collateral can be locked up for longer, until a dispute handler can be activated on-chain.

1. Introduction

Collateral Costs in Payment Channles

- Performance of a payment channel protocol :

  "collateral cost"

  ✓ The **longer** the payment **path**, the **more total collateral** must be reserved.

  ✓ Δ is a safe bound on how long it takes to oberve a transaction committed on the blockchain and commit one new transaction in response.

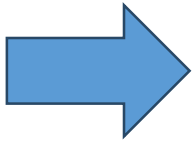
1. Introduction

Collateral Costs in Payment Channles

Lightning Network & Raiden

VS

Sprites



X 3 Improvements!

1. Introduction

Sprites: Constant-Locktime Payment Channels

- Sprites improved by avoiding the need to add an additional Δ delay for each payment on the path.
  - ✓ **Globally accessible smart contract** : provides shared state between individual payment channels.
- **State channel** serves two roles:
  - ➢ Encapsulates necessary cryptography.
  - ➢ Provides a flexible interface bridging the off-chain and on-chain worlds.

2. Background and Preliminaries

Principles & Concepts

2. Background and Preliminaries

2.1 Blockchains and Smart Contracts

- Blockchain ensures the following properties
  1. All parties can agree on a consistent log of committed transactions
  2. All parties are guaranteed to be able to commit new transactions in a predictable amount of time, Δ .
- Δ -> the worst case bound on how long it takes to learn a new transaction.  Time to learn!
- Smart contracts.
  - Autonomous machines that always execute their code correctly.
  - Used in this paper as reactive processes.

2. Background and Preliminaries

2.2 Blockchain Scaling

- On-chain scaling
  - ✓ Make the blockchain **itself run faster**

VS

- Off-chain scaling
  - ✓ **Minimize the use** of the blockchain itself.
  - ✓ Parties are exchanging off-chain messages and interact with the blockchain **only to settle disputes or withdraw funds**.

2. Background and Preliminaries

2.3 Off-chain Payment Channels

- Signatures over round numbers

  → For simplicity

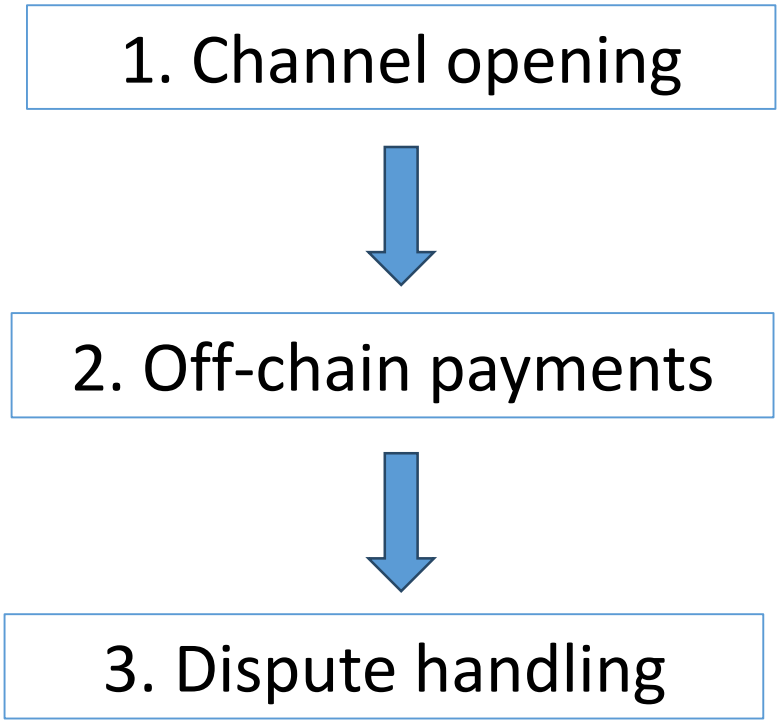
- ”global” event recorded in the blockchain can affect on transactions.

  → Sprites cannot be implemented in Bitcoin!

2. Background and Preliminaries

2.3 Off-chain Payment Channels

- Protocol comprises the following three phases



2. Background and Preliminaries

2.3 Off-chain Payment Channels

- Guaranteed securities are as follows.

Liveness

Each party can initiate a withdrawal, and the withdrawal is processed within a predictable amount of time.

No couterparty risk

The payment channel interface guarantees that local views are accurate. -> Each party can withdraw the amount they expect!

2. Background and Preliminaries

2.4 Linked payments and payment channel networks

- Connecting every pair of parties takes $O(N^2)$ transactions.

- "Hashed TimeLock Contract(HTLC)" helps conditional payments.

synchronize all channel!

$$P_1 \xrightarrow[h,T]{\$X} P_2 \qquad h = \mathcal{H}(x)$$

2. Background and Preliminaries

2.4 Linked payments and payment channel networks

$$P_1 \xrightarrow[h,T_1=T_{\ell-1}+\Theta(\ell\Delta)]{\$X} P_2 \ \ldots\ P_{\ell-1} \xrightarrow[h,T_{\ell-1}]{\$X} P_\ell$$

- The hash condition h is the same for all channels.
- The deadlines T may be different.

$$T_1 = T_\ell + \Theta(\ell\Delta)$$

For Lightning

Liveness

The entire chain of payments concludes within a bounded amount of on-chain cycles.

N. C. R.

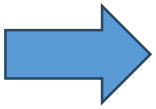
a portion of the channel balance may be “locked”, but it must returned by the conclusion of the protocol.

2. Background and Preliminaries

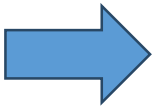
2.4 Linked payments and payment channel networks

$$\ldots\ P_{i-1} \xrightarrow[h,T_i]{\$X} P_i \xrightarrow[h,T_{i+1}]{\$X} P_{i+1}\ \ldots$$

- We need to ensure that if the outgoing conditional payment to P_{i+1} completes, then the incoming payment from P_{i-1} also completes.
- In the worst case, overall collateral cost for each party.



$$\Theta(\ell^2 \$X \Delta)$$



Sprites reduce this!

3. Overview of the Sprites construction

Main concepts of Sprites

3. Overview of the Sprites construction

3.1 Constant locktime linked payments.

- Using the variation of the standard “hashed timelock contract”
- “the preimage x of hash h = H(x) was published on the blockchain before time T_{Expiry}.” -> implemented on Ethereum smart contract.

$$P_1 \xrightarrow[\text{PM}[h, T_{\text{Expiry}}]]{\$X} P_2 \ \ldots \ P_{\ell-1} \xrightarrow[\text{PM}[h, T_{\text{Expiry}}]]{\$X} P_\ell$$

3. Overview of the Sprites construction

3.1 Constant locktime linked payments.

- The difference between Sprites and Lightning is how Sprites handling disputes.

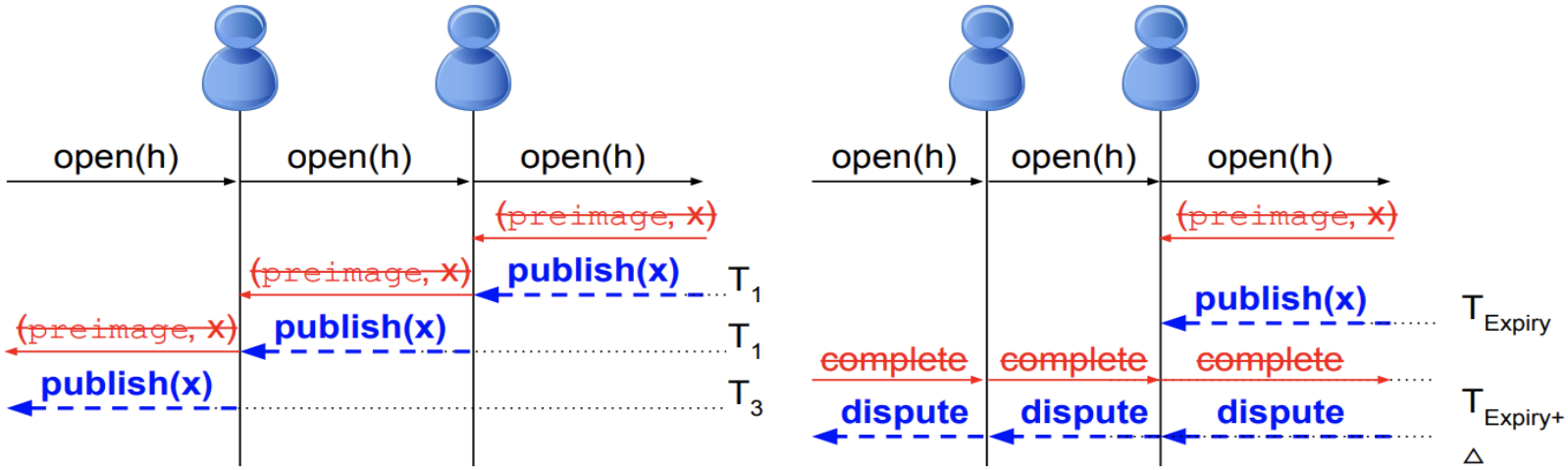
Delegate!

Locally enforce the preimage x ⟹ Global PM contract

- The preimage x is initially known to the recipient. After the final conditional payment to the recipient is opened, the recipient publishes x, and each party completes their outgoing payment.

3. Overview of the Sprites construction

3.1 Constant locktime linked payments.



- In the worst case, the attacker publishes x at the latest possible time.
- However, the use of a global synchronizing gadget, the PM contract, ensures that all payments along the path are settled consistently.
- In constrast, Lightning require the preimage to be submitted to each payment channel contract separately, leading to longer locktimes.

3. Overview of the Sprites construction

3.2 Supporting incremental deposits and withdrawals.

- A Lightning channel must be closed and re-opened in order for either party to withdraw or deposit currency.

- On the other hand, Sprites permits either party to deposit/withdraw a portion of currency without interrupting the channel.

3. Overview of the Sprites construction

3.2 Supporting incremental deposits and withdrawals.

- Incremental deposits: off-chain includes local view!
  - $\text{deposits}_{\{L,R\}}$ reflects the total amount of deposits from each party.
- Incremental withdrawals: off-chain with an optional withdrawal value wd_i
  - Smart contract verifies the signatures that signed by the party who want to withdraw.
  - Disburses the withdrawal and advances the round number to prevent the replay attacks.

4. The State Channel Abstraction

Core of Sprites

4. The State Channel Abstraction

- State Channel generalizes off-chain payment channels
- Each time the parties provide input to the state channel, they exchange signed messages on the newly updated state, along with an increasing round number.
- Once activated, the dispute handler proceeds in two phases.
  - The dispute handler waits for one on-chain round, during which any party can submit their evidence.
  - Then, the dispute handler checks the signatures on the submitted evidence, and ultimately commits the state with the highest round number. After committing the previous state, the dispute handler then allow parties to submit new inputs for the next round.

Liveness	Each party is able to provide input to each iteration of the state machine, and a corrupt party cannot stall.
Safety	Each party's local view of the most recent state is finalized and consistent with every other party's view

4. The State Channel Abstraction

4.1 Instantiating state channels

- off-chain state can be advanced by having parties exchange a signed message of the following form. P_i : party i.

$$\sigma_{r,i} := \mathsf{Sign}_{P_i}(r \| \mathsf{state}_r \| \mathsf{out}_r).$$

- r: the number of the current round
- state_r : result after applying the state transition function to every party's inputs
- out_r : resulting blockchain output

4. The State Channel Abstraction

4.1 Instantiating state channels

Protocol $\Pi_{\mathsf{State}}(U, P_1, ...P_N)$

Contract $\mathsf{Contract}_{\mathsf{State}}$

Initialize bestRound $:= -1$
Initialize state $:= \emptyset$
Initialize flag $:=$ `OK`
Initialize deadline $:= \perp$
Initialize applied $:= \emptyset$

on **contract input** `evidence`$(r, \mathsf{state}', \mathsf{out}, \{\sigma_{r,j}\})$:
  discard if $r \leq$ bestRound
  verify all signatures on the message $(r \| \mathsf{state}' \| \mathsf{out})$
  if flag $==$ `DISPUTE` then
    flag $:=$ `OK`
    emit `EventOffchain`(bestRound $+ 1$)
  bestRound $:= r$
  state $:= \mathsf{state}'$
  invoke C.`aux_output`(out)
  applied $:=$ applied $\cup \{r\}$

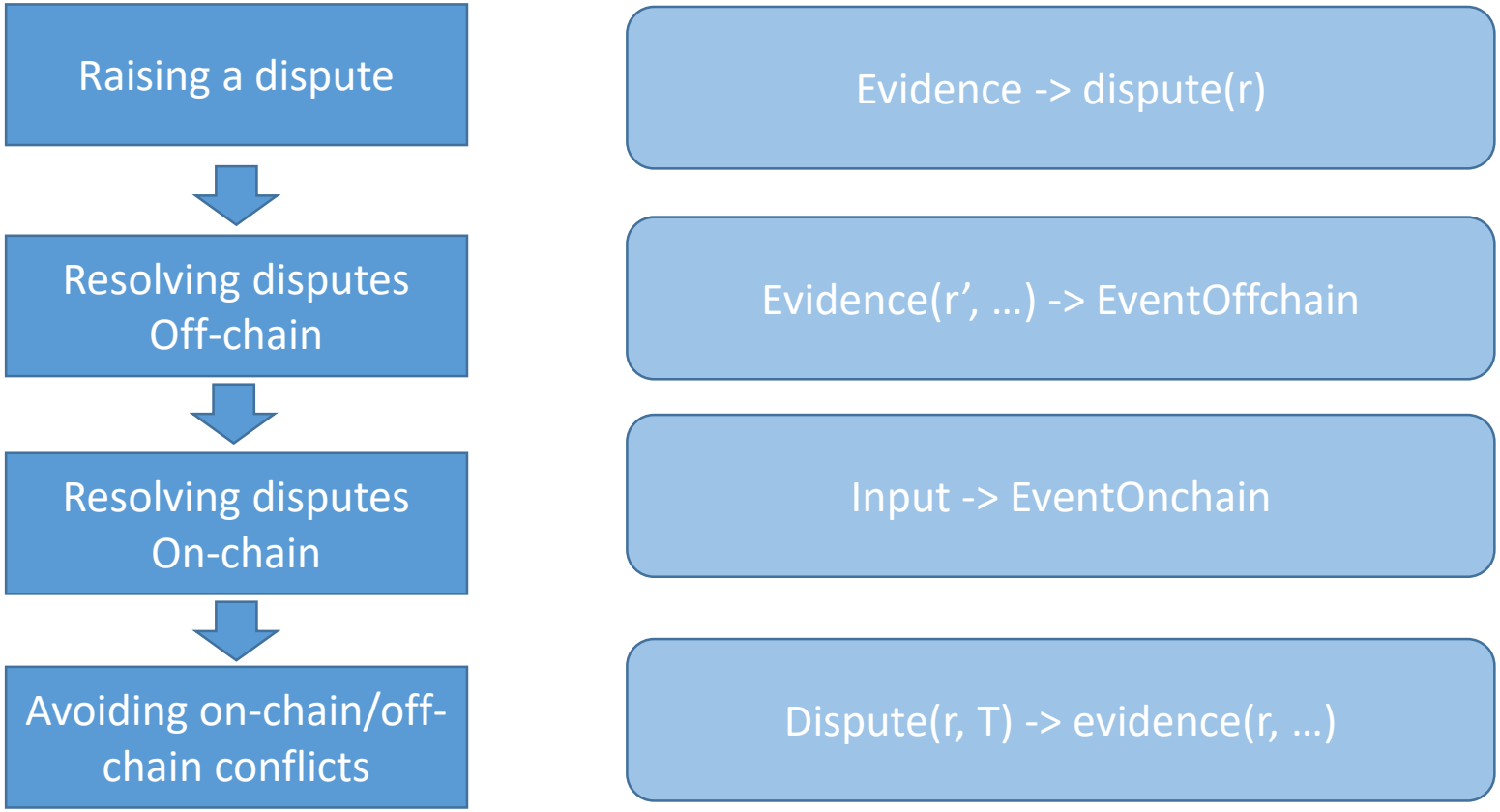
on **contract input** `dispute`(r) at time T:
  discard if $r \neq$ bestRound $+ 1$
  discard if flag $\neq$ `OK`
  set flag $:=$ `DISPUTE`
  set deadline $:= T + \Delta$
  emit `EventDispute`$(r,$ deadline$)$
on **contract input** `input`$(r, v_{r,j})$ from party P_j:
  if this is the first such activation, store $v_{r,j}$
on **contract input** `resolve`(r) at time T:
  discard if $r \neq$ bestRound $+ 1$
  discard if flag $\neq$ `PENDING`
  discard if $T <$ deadline
  apply the update function state $:= U(\mathsf{state}, \{v_{r,j}\}, \mathsf{aux}_{in})$, where the default value is used for any $v_{r,j}$ such that party P_j has not provided input
  set flag $:=$ `OK`
  emit `EventOnchain`$(r,$ state$)$
  bestRound $:=$ bestRound $+ 1$

Fig. 3: Contract portion of the protocol Π_{State} for implementing a general purpose state channel.

4. The State Channel Abstraction

4.1 Instantiating state channels

- How ContractState handles disputes are as follows.



4. The State Channel Abstraction

4.2 Modeling payments channels with state channels

Update function U_{Pay}

$U_{\mathsf{Pay}}(\mathsf{state}, (\mathsf{input}_{\mathtt{L}}, \mathsf{input}_{\mathtt{R}}), \mathsf{aux}_{in})$:
- if $\mathsf{state} = \bot$, set $\mathsf{state} := (0, \emptyset, 0, \emptyset)$
- parse state as $(\mathsf{cred}_{\mathtt{L}}, \mathsf{oldarr}_{\mathtt{L}}, \mathsf{cred}_{\mathtt{R}}, \mathsf{oldarr}_{\mathtt{R}})$
- parse $\mathsf{aux}_{\mathsf{in}}$ as $\{\mathsf{deposits}_i\}_{i\in\{\mathtt{L},\mathtt{R}\}}$
- for $i \in \{\mathtt{L}, \mathtt{R}\}$:
  - if $\mathsf{input}_i = \bot$ then $\mathsf{input}_i := (\emptyset, 0)$
  - parse each input_i as $(\mathsf{arr}_i, \mathsf{wd}_i)$
  - $\mathsf{pay}_i := 0, \mathsf{newarr}_i := \emptyset$
  - while $\mathsf{arr}_i \neq \emptyset$
    - pop first element of arr_i into e
    - if $e + \mathsf{pay}_i \leq \mathsf{deposits}_i + \mathsf{cred}_i$:
      - append e to newarr_i
      - $\mathsf{pay}_i \mathrel{+}= e$
  - if $\mathsf{wd}_i > \mathsf{deposits}_i + \mathsf{cred}_i - \mathsf{pay}_i$: $\mathsf{wd}_i := 0$
- $\mathsf{cred}_{\mathtt{L}} \mathrel{+}= \mathsf{pay}_{\mathtt{R}} - \mathsf{pay}_{\mathtt{L}} - \mathsf{wd}_{\mathtt{L}}$
- $\mathsf{cred}_{\mathtt{R}} \mathrel{+}= \mathsf{pay}_{\mathtt{L}} - \mathsf{pay}_{\mathtt{R}} - \mathsf{wd}_{\mathtt{R}}$
- if $\mathsf{wd}_{\mathtt{L}} \neq 0$ or $\mathsf{wd}_{\mathtt{R}} \neq 0$:
  - $\mathsf{aux}_{out} := (\mathsf{wd}_{\mathtt{L}}, \mathsf{wd}_{\mathtt{R}})$
- otherwise $\mathsf{aux}_{out} := \bot$
- $\mathsf{state} := (\mathsf{cred}_{\mathtt{L}}, \mathsf{newarr}_{\mathtt{L}}, \mathsf{cred}_{\mathtt{R}}, \mathsf{newarr}_{\mathtt{R}})$
- return $(\mathsf{aux}_{out}, \mathsf{state})$

Auxiliary smart contract $\mathsf{Contract}_{\mathsf{Pay}}(P_{\mathtt{L}}, P_{\mathtt{R}})$

Initially, $\mathsf{deposits}_{\mathtt{L}} := 0, \mathsf{deposits}_{\mathtt{R}} := 0$

on **contract input** $\mathtt{deposit}(\mathbf{coins}(\$X))$ from P_i :
- $\mathsf{deposits}_i \mathrel{+}= \$X$
- $\mathsf{aux}_{in}.\mathsf{send}(\mathsf{deposits}_{\mathtt{L}}, \mathsf{deposits}_{\mathtt{R}})$

on **contract input** $\mathtt{output}(\mathsf{aux}_{out})$:
- parse aux_{out} as $(\mathsf{wd}_{\mathtt{L}}, \mathsf{wd}_{\mathtt{R}})$
- for $i \in \{\mathtt{L}, \mathtt{R}\}$ send $\mathbf{coins}(\mathsf{wd}_i)$ to P_i

Local protocol Π_{Pay} for party P_i

initialize $\mathsf{pay}_i := 0$, $\mathsf{wd}_i := 0$, $\mathsf{paid}_i = 0$

on **receiving state** $(\mathsf{cred}_{\mathtt{L}}, \mathsf{new}_{\mathtt{L}}, \mathsf{cred}_{\mathtt{R}}, \mathsf{new}_{\mathtt{R}})$ from Π_{State},
- foreach e in new_i: set $\mathsf{paid}_i \mathrel{+}= e$
- provide $(\mathsf{arr}_i, \mathsf{wd}_i)$ as input to Π_{State}
- $\mathsf{arr}_i := \emptyset$

on **input** $\mathsf{pay}(\$X)$ from $\mathsf{Contract}_{\mathsf{Pay}}$,
- if $\$X \leq \mathsf{Contract}_{\mathsf{Pay}}.\mathsf{deposits}_i + \mathsf{paid}_i - \mathsf{pay}_i - \mathsf{wd}_i$:
  - append $\$X$ to arr_i
  - $\mathsf{pay}_i \mathrel{+}= \$X$

on **input** $\mathtt{withdraw}(\$X)$ from $\mathsf{Contract}_{\mathsf{Pay}}$,
- if $\$X \leq \mathsf{Contract}_{\mathsf{Pay}}.\mathsf{deposits}_i + \mathsf{paid}_i - \mathsf{pay}_i - \mathsf{wd}_i$ then $\mathsf{wd}_i \mathrel{+}= \$X$

Fig. 4: Implementation of a duplex payment channel with the Π_{State} primitive.

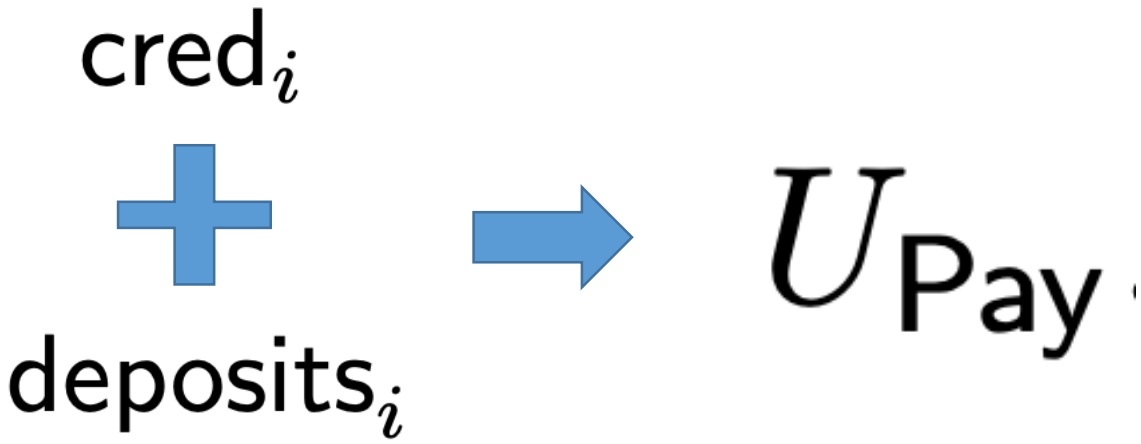
4. The State Channel Abstraction

4.2 Modeling payment channels with state channels

- Implementation of a duplex payment channel consists of as follows.
  - U_{Pay} : defines the structure of the state and the inputs provided by the parties.
  - $\text{Contract}_{\text{Pay}}$: handles deposits and withdrawals.
  - Local behavior for each party.

4. The State Channel Abstraction

4.2 Modeling payment channels with state channels



Balance Available

$$\text{deposits}_i + \text{cred}_i$$

5. Linked Payments from State Channels

How we link payments together along a path of payment channels?

5. Linked Payments from State Channels

How can we ensure the collateral is returned within a bounded time?

- Duplex channels Π_{Pay}
- Linked payments consists as follows.
  - U_{Linked} : Update function -> outer layer around the U_{Pay}
  - Auxiliary contract
  - Local protocol for party

5. Linked Payments from State Channels

How can we ensure the collateral is returned within a bounded time?

- Establishing a path of linked payments off-chain:
  1. Sender P_1 creates a secret xm shares it with the recipient P_1
  2. P_1 creates an outgoin conditional payment to P_2 using h = H(x).
  3. Each subsequent party P_i in turn, upon receiving the incoming conditional payment, establishes an outgoing conditional payment to P_{i+1} .
  4. Once the recipient P_ℓ receives the final conditional payment, it multicasts x to every other party.

5. Linked Payments from State Channels

Protocol $\Pi_{\mathsf{Linked}}(\$X, T, P_1, ...P_\ell)$

Let $T_{\mathsf{Expiry}} := T + 6\ell + \Delta$.
Let $T_{\mathsf{Crit}} := T_{\mathsf{Expiry}} - \Delta$
Let $T_{\mathsf{Dispute}} := T_{\mathsf{Expiry}} + \Delta + 3$.

Update Function
$U_{\mathsf{Linked},\$X}(\mathsf{state}, \mathsf{in}_{\mathtt{L}}, \mathsf{in}_{\mathtt{R}}, \mathsf{aux}_{in})$

if $\mathsf{state} = \bot$, set $\mathsf{state} := (\mathtt{init}, \bot, (0,0))$
parse state as $(\mathsf{flag}, h, (\mathsf{cred}_{\mathtt{L}}, \mathsf{cred}_{\mathtt{R}}))$
parse $\mathsf{in}_{\mathtt{i}}$ as $(\mathsf{cmd}_i, \mathsf{in}_{\mathtt{i}}^{\mathsf{Pay}})$, for $i \in \{\mathtt{L}, \mathtt{R}\}$
if $\mathsf{cmd}_{\mathtt{L}} = \mathtt{open}(h')$ and $\mathsf{flag} = \mathtt{init}$, then
  set $\mathsf{cred}_{\mathtt{L}}$ $-= \$X$, $\mathsf{flag} := \mathtt{inflight}$, and $h := h'$
else if $\mathsf{cmd}_{\mathtt{L}} = \mathsf{complete}$ and $\mathsf{flag} = \mathtt{inflight}$,
  set $\mathsf{cred}_{\mathtt{R}}$ $+= \$X$, and $\mathsf{flag} := \mathtt{complete}$
else if $\mathsf{cmd}_{\mathtt{R}} = \mathsf{cancel}$ and $\mathsf{flag} = \mathtt{inflight}$,
  set $\mathsf{cred}_{\mathtt{L}}$ $+= \$X$ and $\mathsf{flag} := \mathtt{cancel}$
else if $\mathsf{cmd}_{\mathtt{R}} = \mathtt{dispute}$ or $\mathsf{cmd}_{\mathtt{L}} = \mathtt{dispute}$, and $\mathsf{flag} = \mathtt{inflight}$, and current time $> T_{\mathsf{Expiry}}$, then
  $\mathsf{aux}_{\mathsf{out}} := (\mathtt{dispute}, h, \$X)$ and $\mathsf{flag} = \mathtt{dispute}$
let $\mathsf{state}^{\mathsf{Pay}} := (\mathsf{cred}_{\mathtt{L}}, \mathsf{cred}_{\mathtt{R}})$
$(\mathsf{aux}_{\mathsf{out}}^{\mathsf{Pay}}, \mathsf{state}^{\mathsf{Pay}}) := U_{\mathsf{Pay}}(\mathsf{state}^{\mathsf{Pay}}, \mathsf{in}_{\mathtt{L}}^{\mathsf{Pay}}, \mathsf{in}_{\mathtt{R}}^{\mathsf{Pay}}, \mathsf{aux}_{\mathsf{in}})$
set $\mathsf{state} := (\mathsf{flag}, h, \mathsf{state}^{\mathsf{Pay}})$
return $(\mathsf{state}, (\mathsf{aux}_{\mathsf{out}}, \mathsf{aux}_{\mathsf{out}}^{\mathsf{Pay}}))$

Auxiliary contract $\mathsf{Contract}_{\mathsf{Linked}}$

Copy the auxiliary contract from Figure 5, renaming the output handler to $\mathtt{output}^{\mathsf{Pay}}$
on **contract input** $\mathtt{output}(\mathsf{aux}_{\mathsf{out}}^*)$:
  parse $\mathsf{aux}_{\mathsf{out}}$ as $(\mathsf{aux}_{\mathsf{out}}, \mathsf{aux}_{\mathsf{out}}^{\mathsf{Pay}})$
  if $\mathsf{aux}_{\mathsf{out}}^*$ parses as $(\mathtt{dispute}, h, \$X)$ then
    if $\mathsf{PM}.\mathtt{published}(T_{\mathsf{Expiry}}, h)$, then
      $\mathsf{deposits}_{\mathtt{R}}$ $+= \$X$
    else
      $\mathsf{deposits}_{\mathtt{L}}$ $+= \$X$
    $\mathsf{aux}_{\mathsf{in}} := (\mathsf{deposits}_{\mathtt{L}}, \mathsf{deposits}_{\mathtt{R}})$
  invoke $\mathtt{output}^{\mathsf{Pay}}(\mathsf{aux}_{\mathsf{out}}^{\mathsf{Pay}})$

Global Contract $\mathsf{Contract}_{\mathsf{PM}}$

initially timestamp[] is an empty mapping
on contract input $\mathtt{publish}(x)$ at time T:
  if $\mathcal{H}(x) \notin$ timestamp: then set $\mathsf{timestamp}[\mathcal{H}(x)] := T$
constant function $\mathtt{published}(h, T')$:
  return True if $h \in$ timestamp and $\mathsf{timestamp}[h] \leq T'$
  return False otherwise

Fig. 5: Smart contract for protocol Π_{Linked} that implements linked payments with the Π_{State} primitive. Parts of $U_{\mathsf{Linked},\$X}$ that are delegated to the underlying U_{Pay} are colored blue to help readability. See Appendix (Figure 6) for local behavior.

5. Linked Payments from State Channels

Security Analysis of Linked Payments

➢ Liveness

- ✓ $O(\ell)$ rounds are enough to complete chained payments. With two assumptions.
- ✓ $O(\ell + \Delta)$ rounds are enough to complete or cancel. If the sender and receiver are honest.

➢ No counterparty risk

- ✓ Even if some parties are corrupt, no honest party on the path should lose any money.
- ✓ In the dispute case, the preimage manager, ContractPM acts like a global condition.
- ✓ If the preimage manager receives x before time T_{Expiry}, then every conditional payment that is disputed will complete. Otherwise they are canceled.
- ✓ An honest party that receives x before $T_{\text{Expiry}} - \Delta$, it is safe to complete their outgoing payment.
- ✓ In the worst case then can use the preimage manager and claim their incoming payment.

5. Linked Payments from State Channels

Implementation and performance analysis

- Using Solidity and pyethereum available online.
- In the typical case:
  - Off-chain communication pattern of Sprites is similar to that of Lightning.
  - One round of communication between each adjacent pair of parties to open each conditional payment.
  - One round to complete all the payments.
- In the worst-case:
  - Each channel that must be resolved via the dispute handler requires one on-chain transaction to initiate the dispute and send the preimage to ContractPM
  - Later, send a transaction to complete the dispute and withdraw the balance.
- On November 2018, 137294 gas per disputed channel ~ $0.20
  - Lightnin Network the typical cost of closing a channel ~ ($0.072)^9.

5. Linked Payments from State Channels

Protocol $\Pi_{\mathsf{Linked}}(\$X, T, P_1, \ldots P_\ell)$

Local protocol for sender, P_1

on **input pay** from the environment:
  $x \xleftarrow{\$} \{0,1\}^\lambda$, and $h \leftarrow \mathcal{H}(x)$
  pass $(\texttt{open}, h, \$X, T_{\mathsf{Expiry}})$ as input to Π^1_{State}
  send $(\texttt{preimage}, x)$ to P_ℓ
  if $(\texttt{preimage}, x)$ is received from P_2 before T_{Expiry}, then pass `complete` to Π^1_{State}
at time $T_{\mathsf{Expiry}} + \Delta$, if $\mathsf{PM.published}(T_{\mathsf{Expiry}}, h)$, then
  pass input `complete` to Π^1_{State}
at time T_{Dispute}, then pass input `dispute` to Π^1_{State}

Local protocol for party P_i**, where** $2 \leq i \leq \ell - 1$

on receiving state $(\texttt{inflight}, h, \_)$ from $\Pi^{i-1}_{\mathsf{State}}$
  store h
  provide input $(\texttt{open}, h, \$X, T_{\mathsf{Expiry}})$ to Π^i_{State}
on receiving state $(\texttt{cancel}, \_, \_)$ from Π^i_{State},
  provide input $(\texttt{cancel})$ to $\Pi^{i-1}_{\mathsf{State}}$
on receiving $(\texttt{preimage}, x)$ from P_ℓ before time T_{Crit}, where $\mathcal{H}(x) = h$,
  pass `complete` to Π^i_{State}
  at time T_{Crit}, if state $(\texttt{complete}, \_, \_)$ has not been received from Π^i_{State}, then
    pass contract input $\mathsf{PM.publish}(x)$
at time $T_{\mathsf{Expiry}} + \Delta$,
  if $\mathsf{PM.published}(T_{\mathsf{Expiry}}, h)$, pass `complete` to Π^i_{State}
  otherwise, pass `cancel` to $\Pi^{i-1}_{\mathsf{State}}$
at time T_{Dispute}, pass input `dispute` to $\Pi^{i-1}_{\mathsf{State}}$ and Π^i_{State}

Local protocol for recipient, P_ℓ

on receiving $(\texttt{preimage}, x)$ from P_1, store x and $h := \mathcal{H}(x)$
on receiving state $(\texttt{inflight}, h, \_)$ from $\Pi^{\ell-1}_{\mathsf{State}}$,
  multicast $(\texttt{preimage}, x)$ to each party
  at time T_{Crit}, if state $(\texttt{complete}, \_, \_)$ has not been received from $\Pi^\ell_{\mathsf{State}}$, then
    pass contract input $\mathsf{PM.publish}(x)$
at time T_{Dispute}, pass input `dispute` to $\Pi^{\ell-1}_{\mathsf{State}}$

Fig. 6: Construction for Π_{Linked} with the Π_{State} primitive. (Local portion only. See Figure 5 for the smart contract portion.) Portions of the update function $U_{\mathsf{Linked},\$X}$ that are delegated to the underlying U_{Pay} update function (Figure 5) are colored blue to help readability.

6. Related Works

6. Related Works

- 1st off-chain protocols : Bitcoin payment channels by Spilman.
  - Only allow for ”simplex” payments.
- Decker & Wattenhofer <-> Poon and Dryja
  - “duplex” payments but require every growing list of keys to defend against malicious behavior.

Improvements to Payment Channels

- Rebalancing payment channels entirely off-chain.
- Virtual payment channel overlays. -> rapid payment channels.
  - However, honest parties must be online at all times for security!

6. Related Works

Routing in payment channels

- Payment path is not given in reality, so the route finding process is complementary.
- T deadline is defined in terms of the path length l
- Path length must not be revealed
  - Pad the deadline to a conservative upper bound.
- Expiration time is dominated by block time delta.
  - 1 day : Lightning and Raiden.

6. Related Works

Deadlock when multiple concurrent payments

- Global identifiers for payments and a global payment ordering
- **Sprites** also conjecture such **a global identifier** can be implemented on top.

Credit networks

- Privacy-preserving credit networks
  - Payment channel balances are fully backed by on-chain deposits.
  - Can be settled without any counterparty risk.

7. Conclusion

7. Conclusion

- Cryptocurrencies must be scaled up .
  - several transactions per second is not enough!
- Off-chain payment channel networks are currently a leading proposal to scale blockchain-based cryptocurrencies.
  - Lightning require collateral to be locked up for a maximum period that scales linearly with the number of hops, $O(\ell\Delta)$.
  - Sprites reduced it to a constant, $O(\ell + \Delta)$

7. Conclusion

- Current constant locktime construction relies on Ethereum, but what about Bitcoin?
  - Future work: Modify the Bitcoin script so as to enable constant locktimes.

Thank you!