Lesson 13

CRYPTOCURRENCIES: AN INTRODUCTION

Laura Ricci

5/04/2017
for the moment we define a set of simple cryptocurrencies:

- simplified with respect to Bitcoin, even if they share a set of common characteristics
  - exploit decentralized identities
  - the first one presents the problem of double spending
  - the second one introduces a centralized ledger to solve this problem
DECENTRALIZED IDENTITY MANAGEMENT

- how to make a new identity in a cryptographic system:
  - useful trick $\text{Public Keys} == \text{Identity}$
  - create a new, random key-pair (sk secret key, pk public key)
  - pk is the public “name” a user can use (usually better to use $\text{Hash}(pk)$)
  - sk lets you “speak for” the identity

- you control the identity, because only you know sk

- if you see $\text{sig}$, such that $\text{verify}(pk, msg, sig) == \text{true}$, think of it as $pk$ says “[msg]”

- if pk “looks random”, nobody needs to know who you are
DECENTRALIZED IDENTITY MANAGEMENT

• anybody can make a new identity at any time
  make as many identities as you want!

• no central point of coordination

• these identities are called addresses in Bitcoin

• as far as concerns privacy...
  • addresses (public keys) not directly connected to real world identities
  • but an observer can link together an address's activity over time, and make inferences...
A SIMPLE CRYPTOCURRENCY

• a simple cryptocurrency, governed by two simple rules:
  • there is only a designated entity that can create new coins whenever he wants and these newly created coins belong to this entity
  • whoever owns the coin can transfer it to someone else
  • how are the transaction represented in our system?
## A SIMPLE CRYPTOCURRENCY

Goofy can create new coins

<table>
<thead>
<tr>
<th>signed by $sk_{Goofy}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCoin [uniqueCoinID]</td>
</tr>
</tbody>
</table>

New coins belong to me.
A coin’s owner can spend it.

signed by $sk_{Goofy}$

Pay to $pk_{Alice} : H(\cdot)$

signed by $sk_{Goofy}$

CreateCoin [uniqueCoinID]

Alice owns it now.
A SIMPLE CRYPTOCURRENCY

The recipient can pass on the coin again.

signed by $sk_{\text{Alice}}$

Pay to $pk_{\text{Bob}} : H(\cdot)$

signed by $sk_{\text{Goofy}}$

Pay to $pk_{\text{Alice}} : H(\cdot)$

signed by $sk_{\text{Goofy}}$

CreateCoin [uniqueCoinID]

Bob owns it now.
DIGITALLY SIGNED TRANSACTIONS

- When Alice wants to pay another person a coin, she:
  - digitally signs every input, and must also attach the corresponding public key to allow signature verifications.
  - this is an ownership proof

- the digital signature guarantees that only the rightful owner can spend its funds
  - establishment of intent on Alice’s part. Alice truly intends to give Bob one coin and everyone is able to verify this
  - anyone in the world (including Bob) can use Alice’s public key to verify that Alice really was the person who signed the message
  - limited protection from forgery
    - no-one else could have composed such a signed message from scratch
    - however, someone could duplicate the message...
    - it doesn't prevent him/her from spending them more than once in different transactions.
A DOUBLE SPENDING ATTACK

Double spending attack: one of the main challenges in digital currencies

Signed by sk_{Alice}
Pay to pk_{Bob}: H(.)
Signed by sk_{Goofy}
Pay to pk_{Alice}: H(.)
Signed by sk_{Goofy}
CreateCoin [uniqueCoinID]
THE DOUBLE SPENDING PROBLEM

A transaction is valid if:

• the transaction is structurally correct (output funds do not exceed input funds, …)

• input funds are used by its rightful owner

• the input funds do exist and were not already spent in a previous transaction
  • double spending problem!

• the digital signature guarantees that only the rightful owner can spend the funds, but it does not prevent it from spending them more than once in different transactions
  • a different mechanism is required!
THE DOUBLE SPENDING PROBLEM

- to avoid the problem of double spending, we can exploit an append-only ledger
- the ledger is managed by a central, entity,

Scrooge publishes a history of all transactions (a block chain, signed by Scrooge)

prev: $H(\cdot)$
transID: 71
trans

prev: $H(\cdot)$
transID: 72
trans

prev: $H(\cdot)$
transID: 73
trans
THE DOUBLE SPENDING PROBLEM

• append-only property: the data appended to the ledger remains there forever
  • the history of all transactions is permanently stored
  • all the transactions must be written to the ledger before they are accepted
  • a transaction is valid only if it is written in the ledger

• the central entity signs the final hash pointer

• if some modification/insertion is made on the blockchain it will affect following blocks because of the hash pointers
  • even if the central authority modifies the blockchain, it will result in a change of the first hash pointer....

• possible optimization (implemented by Bitcoin): put more transactions in a single block
• The central entity can create new coins

CreateCoins transaction creates new coins

<table>
<thead>
<tr>
<th>transID: 73</th>
<th>type: CreateCoins</th>
</tr>
</thead>
<tbody>
<tr>
<td>coins created</td>
<td></td>
</tr>
<tr>
<td>num</td>
<td>value</td>
</tr>
<tr>
<td>0</td>
<td>3.2</td>
</tr>
<tr>
<td>1</td>
<td>1.4</td>
</tr>
<tr>
<td>2</td>
<td>7.1</td>
</tr>
</tbody>
</table>
THE DOUBLE SPENDING PROBLEM

- The structure of the transaction is the following one:

PayCoins transaction consumes (and destroys) some coins, and creates new coins of the same total value

<table>
<thead>
<tr>
<th>transID: 73</th>
<th>type: PayCoins</th>
</tr>
</thead>
<tbody>
<tr>
<td>consumed coinIDs:</td>
<td>68(1), 42(0), 72(3)</td>
</tr>
<tr>
<td>coins created</td>
<td></td>
</tr>
<tr>
<td>num</td>
<td>value</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>3.2</td>
</tr>
<tr>
<td>1</td>
<td>1.4</td>
</tr>
<tr>
<td>2</td>
<td>7.1</td>
</tr>
</tbody>
</table>

Valid if:
- consumed coins valid,
- not already consumed,
- total value out = total value in, and
- signed by owners of all consumed coins