Cryptographic Foundations of Bitcoin

Seminar on Blockchain Technology

Michał Zając
University of Tartu
Cryptography Research Group
What is Bitcoin

Economics

Cryptography

Distributed systems
Short introduction to Bitcoin

• History
  • proposed in 2009 by Satoshi Nakamoto (alias)
  • no true identity of the creator is known

• Decentralized cryptocurrency
  • Decentralized
    • no central server that keeps transaction logs, sets value of the currency
    • all logs are kept by society in peer-to-peer network
  • Crypto
    • security and functionality based on cryptographic primitives
  • Currency
    • value of the currency set by a market
    • programmed to deflate
Short introduction to Bitcoin

- Transaction logs distributed in peer-to-peer network
  - *Chain of transactions*
  - Length of the chain: over 90GB
- No central bank authority

![Graph showing Bitcoin price over time](https://www.coindesk.com)
1. Separate transactions are **gathered** into a list
2. List of transaction is **confirmed**
3. New list of transactions is **created**
Mr. Burns wants to buy some donuts from Homer for a coin
**Bitcoin workflow - overview**

1. Homer and Mr. Burns have wallets on their computers
2. Wallets consist of bitcoin addresses, e.g. hex numbers: 31c25a87bc 4c4b9ac671d0df39df221
3. Say that Mr. Burns wants to buy a donut from Homer. He creates a new address and tells his bitcoin client to transfer cash to Homer
4. Everyone can verify that the transaction was submitted by Mr. Burns, not by some impostor
5. Bitcoin client propagate information on the transaction to all peer-to-peer nodes
6. Transaction (Mr. Burns -> Homer, bitcoin address) is digitally signed by Mr. Burns
7. Transaction is enlisted to a current Transaction list
8. Transaction list is verified and confirmed. It is added to transaction list chain
9. Homer sends donut
Homer and Mr. Burns have wallets on their computers

Wallets consist of bitcoin addresses, e.g. hex numbers: 31c25a87bc4c4b9ac671d0df39df221

Say that Mr. Burns wants to buy a donut from Homer. He creates a new address and tells his bitcoin client to transfer cash to Homer

Everyone can verify that the transaction was submitted by Mr. Burns, not by some impostor

Bitcoin client propagate information on the transaction to all peer-to-peer nodes

Transaction is enlisted to a current Transaction list

Transaction list is verified and confirmed. It is added to transaction list chain

Transaction is digitally signed by Mr. Burns

Homer sends donut
How Mr. Burns knows he owns a coin?

Mr. Burns owns a bitcoin if in the history of transactions there is a transaction list with entry like:

I send a coin to Mr. Burns

Trans. list 1

| Trans. (1, 1) |
| Trans. (1, 2) |
| ... |
| Trans. (1, n) |

Lisa transfers a coin to Mr. Burns

Mr. Burns can keep this information e.g. as a string of hex-values that identifies the coin:

31c25a87bc4b9ac671d0df39df221ef157940806cdd2cc0607edac2768c6d179
How a transaction looks like?

I, Mr. Burns, want to buy Homer’s donuts and pay him a coin

a coin is a bitstring like:
31c25a87bc4b9ac671d0df39df
221ef157940806cdd2cc0607edac2768c6d179
How a transaction looks like?

I, Mr. Burns, want to buy Homer’s donuts and pay him a coin

Other transactions

Trans. list 1
- Trans. (1, 1)
- Trans. (1, 2)
- ...
- Trans. (1, n)
How a transaction looks like?

I, Mr. Burns, want to buy Homer’s donuts and pay him a coin

Trans. list 1

Trans. (1, 1)
Trans. (1, 2)
...
Trans. (1, n)

CONFIRMED

a coin:
31c25a87bc4b9ac671d0df39df221ef157940806cdd2cc0607edac2768c6d179
1. Separate transactions are gathered into a list
   • propagate transaction through peer-to-peer network
2. List of transaction is confirmed
   • solve a puzzle to confirm list
3. Create new list of transactions
Confirm list of transactions

**Transaction list puzzle:**
- get a *puzzle-friendly* function \( H \)
- find a nonce \( r \) such that \( H(r \ || \ \text{Trans. list 1}) < \text{TARGET} \)
- When such a nonce if found it is propagated through the network and transaction list is considered confirmed

<table>
<thead>
<tr>
<th>Trans. list 1</th>
<th>Trans. list 2</th>
<th>Trans. list 3</th>
<th>Trans. list 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (1, 1)</td>
<td>Trans. (2, 1)</td>
<td>Trans. (3, 1)</td>
<td>Trans. (4, 1)</td>
</tr>
<tr>
<td>Trans. (1, 2)</td>
<td>Trans. (2, 2)</td>
<td>Trans. (3, 2)</td>
<td>Trans. (4, 2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Trans. (1, n)</td>
<td>Trans. (2, n)</td>
<td>Trans. (3, n)</td>
<td>Trans. (4, n)</td>
</tr>
<tr>
<td>( H(r_0 \</td>
<td></td>
<td>\ \text{Tr.0}) )</td>
<td>( H(r_1 \</td>
</tr>
<tr>
<td>( r_1 )</td>
<td>( r_2 )</td>
<td>( r_3 )</td>
<td>??????????????</td>
</tr>
</tbody>
</table>

Function \( H \) and \( \text{TARGET} \) are set in such a way that expected time needed to confirm transaction list is 10 minutes.
Confirm list of transactions

Transaction list puzzle:
- get a *puzzle-friendly* function $H$
- find a nonce $r$ such that $H(r || \text{Trans. list 1}) < \text{TARGERT}$
- When such a nonce if found it is propagated through the network and transaction list is considered confirmed

Function $H$ and $\text{TARGET}$ are set in such a way that expected time needed to confirm transaction list is 10 minutes

Who looks for nonce $r$?
- Basically, everyone can find a proper $r$.
- But this takes time and energy, thus it generates costs
- Thus, people who find proper $r$ are awarded
  - (for now 25BTC = 16000EUR)
Confirm list of transactions

**Transaction list puzzle:**
- get a *puzzle-friendly* function $H$
- find a nonce $r$ such that $H(r \ || \ \text{Trans. list 1}) < \text{TARGET}$
- When such a nonce if found it is propagated through the network and transaction list is considered confirmed

**Who looks for nonce $r$?**
- Basically, everyone can find a proper $r$.
- But this takes time and energy, thus it generates costs
- Thus, people who find proper $r$ are awarded
  - (for now 25BTC = 16000EUR)

Function $H$ and $\text{TARGET}$ are set in such a way that expected time needed to confirm transaction list is 10 minutes

This is called **MINING**
This is how bitcoins are **CREATED**
Cryptographic Foundations of Bitcoin

UNFORGEABILITY OF HISTORY
### Unforgeability of history

<table>
<thead>
<tr>
<th>Trans. list 1</th>
<th>Trans. list 2</th>
<th>Trans. list 3</th>
<th>Trans. list 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (1, 1)</td>
<td>Trans. (2, 1)</td>
<td>Trans. (3, 1)</td>
<td>Trans. (4, 1)</td>
</tr>
<tr>
<td>Trans. (1, 2)</td>
<td>Trans. (2, 2)</td>
<td>Trans. (3, 2)</td>
<td>Trans. (4, 2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Trans. (1, n)</td>
<td>Trans. (2, n)</td>
<td>Trans. (3, n)</td>
<td>Trans. (4, n)</td>
</tr>
<tr>
<td>H(r0</td>
<td></td>
<td>Tr.0)</td>
<td>H(r1</td>
</tr>
<tr>
<td>r1</td>
<td>r2</td>
<td>r3</td>
<td>r4</td>
</tr>
</tbody>
</table>

Mr. Burns transaction
Unforgeability of history

- Denote by Trans. list 3’ transaction list with Mr. Burns purchase removed.
- Before Trans. list 4 can be confirmed it is checked whether $H(r3 || Tr.3)$ is correct (by looking e.g. at Trans. list 4)
- $H$ chosen in such a way that even a small change in input changes output.
- Thus, condition $H(r3 || Tr.3') < \text{TARGET}$ is not fulfilled anymore (with a probability close to 1)
- Mr. Burns needs to find a new nonce for Trans. list 3
### Unforgeability of history

<table>
<thead>
<tr>
<th>Trans. list 1</th>
<th>Trans. list 2</th>
<th>Trans. list 3</th>
<th>Trans. list 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (1, 1)</td>
<td>Trans. (2, 1)</td>
<td>Trans. (3, 1)</td>
<td>Trans. (4, 1)</td>
</tr>
<tr>
<td>Trans. (1, 2)</td>
<td>Trans. (2, 2)</td>
<td>Trans. (3, 2)</td>
<td>Trans. (4, 2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>XXXXXXXXX</td>
<td>...</td>
</tr>
<tr>
<td>Trans. (1, n)</td>
<td>Trans. (2, n)</td>
<td>Trans. (3, n)</td>
<td>Trans. (4, n)</td>
</tr>
<tr>
<td>H(r0</td>
<td></td>
<td>Tr.0)</td>
<td>H(r1</td>
</tr>
<tr>
<td>r1</td>
<td>r2</td>
<td>r3'</td>
<td>r4</td>
</tr>
</tbody>
</table>

Can Mr. Burns come with such r3’ that
\[ H(r3' || Tr.3') = H(r3 || Tr.3') \]

i.e. such a r3’ that change in Trans. list 3 does not propagate to the further blocks?
Can Mr. Burns come with such $r_3'$ that $H(r_3' || \text{Tr.3'}) = H(r_3 || \text{Tr.3'})$? 
i.e. such a $r_3'$ that change in Trans. list 3 does not propagate to the further blocks?

Fortunately, if $H$ is a **collision resilient** hash function Mr. Burns won’t find such a $r_3'$

Every change made in the middle of a chain propagates till its end.
Cryptographic hash function

\[ H: \{0,1\}^{kn} \rightarrow \{0,1\}^n \]

Collision resiliency

For any PPT algorithm \( A \)
- \( A \) gets a hash function \( H \)
- It is **hard** for \( A \) to come up with (different) \( X, Y \) such that \( H(Y) = H(X) \)
Cryptographic hash function

\[ H : \{0,1\}^{kn} \rightarrow \{0,1\}^n \]

Collision resiliency

For any PPT algorithm \( A \)
- \( A \) gets a hash function \( H \)
- It is \textbf{hard} for \( A \) to come up with (different) \( X, Y \) such that \( H(Y) = H(X) \)

\textit{Actually, we need a little less: in the proposed scenario it is enough if \( A \) gets \( H \) with a value \( X \) and for such a particular value she has to come up with proper \( Y \).}
**Cryptographic hash function**

\[ H: \{0,1\}^{kn} \rightarrow \{0,1\}^n \]

**Collision resiliency**

For any PPT algorithm \( A \)
- \( A \) gets a hash function \( H \)
- It is **hard** for \( A \) to come up with (different) \( X, Y \) such that \( H(Y) = H(X) \)

**But** if we consider *any* \( A \), can come up with some dummy \( A \) that has sewed in values \( X, Y \) such that \( H(X) = H(Y) \)

**We need to tune the definition:**
we introduce *keyed* hash *family* \( \{H_k\} \) for \( k \) in key space \( K \)

**Now:**
For any PPT algorithm \( A \)
- \( A \) gets a key \( k \) and hash function \( H \) from family \( \{H_k\} \)
- It is hard for \( A \) to come up with (different) \( X, Y \) such that \( H(k, Y) = H(k, X) \) for \( H \) randomly picked from \( \{H_k\} \)
Cryptographic hash function

\[ H: \{0,1\}^k \rightarrow \{0,1\}^n \]

**Collision resiliency**

For any PPT algorithm \( A \)
- \( A \) gets a hash function \( H \)
- It is **hard** for \( A \) to come up with (different) \( X, Y \) such that \( H(Y) = H(X) \)

**But** if we consider *any* \( A \), can come up with some dummy \( A \) that has sewed in values \( X, Y \) such that \( H(X) = H(Y) \)

**We need to tune the definition:**

In real world we use concrete hash functions like SHA256, SHA3, etc. **not** hash function families

\[ H(Y) = H(X) \text{ for } H \text{ randomly picked from } \{H_k\} \]
Collisions always occur

$2^{nk}$ elements

$2^n$ elements

Input can have arbitrary length but the length of output is fixed.

In Bitcoin we use SHA256 hash function that on produces 256 bit long output
Collisions always occur

For any $X$:
- Say we know $X$ and $H(X)$
- It is **hard** to come up with $Y$ such that $H(Y) = H(X)$

What does **hard** mean?
- Output of length 256.
- If we check $2^{130}$ random inputs we will find $X$ and $Y$ such that $H(X) = H(Y)$ with probability at least 99.8%

Can we do better than check random inputs? Find a clever way to come up with $X, Y$ such that $H(X) = H(Y)$?

**Q:** How long does it take to check $2^{130}$ inputs?

**A:** More than the universe exists

**We don’t know 😞**

**No** proof available...

That’s why we have hash function *candidates*
How to hash long messages?

To construct function that takes arbitrary long inputs and produce output from \( \{0, 1\}^n \) it is enough to get a function that maps \( \{0, 1\}^{2n} \) to \( \{0, 1\}^n \)

**Merkle – Damgaard construction**

Actually for Bitcoin we use SHA256 that maps \( \{0, 1\}^{3n} \) to \( \{0, 1\}^n \)
How to hash long messages?

To construct function that takes arbitrary long inputs and produce output from $\{0, 1\}^n$ it is enough to get a function that maps $\{0, 1\}^{2n}$ to $\{0, 1\}^n$.

Merkle – Damgaard construction

Actually for Bitcoin we use SHA256 that maps $\{0, 1\}^{3n}$ to $\{0, 1\}^n$.

Problem: usually the message length is not a multiple of 256, 512 etc.
How to hash long messages?

To construct a function that takes arbitrary long inputs and produce output from \(\{0, 1\}^n\) it is enough to get a function that maps \(\{0, 1\}^{2n}\) to \(\{0, 1\}^n\).

**Problem:** usually the message length is not a multiple of 256, 512 etc.

**Idea 1:** add ‘0000...0’ to have proper length
How to hash long messages?

To construct function that takes arbitrary long inputs and produce output from \( \{0, 1\}^n \) it is enough to get a function \( f \) that maps \( \{0, 1\}^{2n} \) to \( \{0, 1\}^n \)

**Problem:** usually the message length is not a multiple of 256, 512 etc.

**Idea 1:** add ‘0000...0’ to have proper length

**Doesn’t work:** \( H(t) = H(t | 00...00) \)
How to hash long messages?

To construct function that takes arbitrary long inputs and produce output from \( \{0, 1\}^n \) it is enough to get a function that maps \( \{0, 1\}^{2n} \) to \( \{0, 1\}^n \)

**Problem:** usually the message length is not a multiple of 256, 512 etc.

**Idea 2:** add ‘0…\(|t|\)’ to have proper length

**This actually works** (proof omitted)
Q: How to quickly check collection of 142 music CDs for errors?

This won’t tell you where the error is!

A: Compute SHA sum and compare it with SHA sum available online
Why it is great?

Even small change in an input, changes output a lot

Observation (informal): outputs are unrelated and random

Cryptographic hash function is a puzzle-friendly function
• you cannot predict the output before you compute it
• even a slightly modification in the input, changes output completely
• To find for $Y$ s.t $H(Y) < \text{TARGET}$ you cannot do better then try random $Y$.
• Even if you tried a lot $Y$s, it does not get you any closer to the solution
• It is easy to adjust difficulty (by specifying $\text{TARGET}$ value)
  • Important if we want to keep expected time need to confirm a transaction around 10 minutes – otherwise it will be highly dependent on the power of nodes in the network
How to determine which transaction list is next?
How to determine which Transaction list is next?

Decentralization of Bitcoin
- No central server than can provide timestamp

A number of next Transaction list candidates. Which one should we choose?
Everyone is fine.
How to determine which Transaction list is next?

Decentralization of Bitcoin
- No central server than can provide timestamp

A number of next Transaction list candidates. Which one should we choose? Everyone is fine.

Maybe we can run Byzantine agreement protocol and allow servers to agree on the timestamp?

Unfortunately, that won’t work: dishonest parties can easily set up new entities and to set Byzantine agreement we need more than 2/3 parties to be honest.
How to determine which Transaction list is next?

**Decentralization of Bitcoin**
- No central server than can provide timestamp

A **number** of next Transaction list candidates. Which one should we choose? **Everyone is fine.**
How to determine which Transaction list is next?

**Decentralization of Bitcoin**
- No central server than can provide timestamp

<table>
<thead>
<tr>
<th>Trans. list 1</th>
<th>Trans. list 2</th>
<th>Trans. list 3a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (1, 1)</td>
<td>Trans. (2, 1)</td>
<td>Trans. (3a, 1)</td>
</tr>
<tr>
<td>Trans. (1, 2)</td>
<td>Trans. (2, 2)</td>
<td>Trans. (3a, 2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Trans. (1, n)</td>
<td>Trans. (2, n)</td>
<td>Trans. (3a, n)</td>
</tr>
<tr>
<td>H(r0</td>
<td></td>
<td>Tr.0)</td>
</tr>
<tr>
<td>r1</td>
<td>r2</td>
<td>r3</td>
</tr>
</tbody>
</table>

A **number** of next Transaction list candidates. Which one should we choose? **Everyone is fine.**

**Again:** First come, first served. We add the list that a proper nonce \( r3 \) was found first.
How to determine which Transaction list is next?

Decentralization of Bitcoin
- No central server than can provide timestamp

What if we find a proper $r$ for two lists? Then we have a fork.
How to determine which Transaction list is next?

The race begins

What if we find a proper rs for two lists? Then we have a *fork*.

Rule of thumb: add to the longer chain, forget short forks, or in biblical words: *Whoever has will be given more; whoever does not have, even what they think they have will be taken from them*
How to determine which Transaction list is next?

The race begins

What if we find a proper rs for two lists? Then we have a fork.

Rule of thumb: add to the longer chain, forget short forks, or in biblical words:
*Whoever has will be given more; whoever does not have, even what they think they have will be taken from them*
CAN MR. BURNS SPEND THE SAME COIN TWICE?
Can Mr. Burns spend the same coin twice?

I, Mr. Burns, want to buy Homer’s donuts and pay him a coin

I, Mr. Burns, want to buy Lisa’s donuts and pay her a coin

a coin:
31c25a87bc4b9ac671d0df3
9df221ef157940806cdd2cc0
607edac2768c6d179
Can Mr. Burns spend the same coin twice?

- Burns buys from Homer
  - Trans. list 1
    - Trans. (1, 1)
    - ...
    - Trans. (1, n)
  - Trans. list 2
    - Trans. (2, 1)
    - ...
    - Trans. (2, n)
  - Trans. list 3
    - Trans. (3, 1)
    - ...
    - Trans. (3, n)
  - Trans. list 4
    - Trans. (4, 1)
    - ...
    - Trans. (4, n)

- Burns buys from Lisa
  - Trans. list 2’
    - Trans. (2’, 1)
    - ...
    - Trans. (2’, n)
  - Trans. list 3’
    - Trans. (3’, 1)
    - ...
    - Trans. (3’, n)
Can Mr. Burns spend the same coin twice?

We don’t know which transaction occurred first. We consider this transaction valid that was followed by the longer chain.
Can Mr. Burns spend the same coin twice?

We don’t know which transaction occurred first. We consider this transaction valid that was followed by the longer chain.
Can Mr. Burns spend the same coin twice?

We don’t know which transaction occurred first. We consider this transaction valid that was followed by the longer chain.

In general we **cannot** prevent such situations.

We assume that seller sends goods after there is a long chain after Transaction list.

If a transaction is done in `Transaction_list_j`, we assume that this Transaction won’t be rejected if `Transaction_list_(j + 6)` is confirmed.
Can Mr. Burns spend the same coin twice?

Crucial assumption for Bitcoin security

*Majority is honest*
Can Mr. Burns spend the same coin twice?

Crucial assumption for Bitcoin security

*Majority is honest*

What is *majority*?
- number of nodes in Bitcoin network – bad idea dishonest participants can easily create mockup identities
- *Majority of computational power* is honest
What if majority is not honest?

Burns buys from Homer

Trans. list 1
Trans. (1, 1)
...
Trans. (1, n)

Trans. list 2
Trans. (2, 1)
...
Trans. (2, n)

Trans. list 3
Trans. (3, 1)
...
Trans. (3, n)

Burns buys from Lisa

Trans. list 2'
Trans. (2', 1)
...
Trans. (2', n)

Trans. list 3'
Trans. (3', 1)
...
Trans. (3', n)

Trans. list 4'
Trans. (4', 1)
...
Trans. (4', n)

Lisa cooperates with Burns

Majority computational power makes this chain longer.
Cryptographic Foundations of Bitcoin

CAN SELLER FAKE THE TRANSACTION?
Can seller fake the transaction?

I, Mr. Burns, want to buy Homer’s donuts and pay him a coin.
Can seller fake the transaction?

I, Mr. Burns, want to buy Homer’s donuts and pay him a coin.
Can seller fake the transaction?

<table>
<thead>
<tr>
<th>Trans. list k</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (k, 1)</td>
<td></td>
</tr>
<tr>
<td>Trans. (k, 2)</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Trans. (k, n)</td>
<td></td>
</tr>
<tr>
<td>(H(r(k-1) \</td>
<td></td>
</tr>
<tr>
<td>(r_k)</td>
<td></td>
</tr>
</tbody>
</table>

We need to check whether every transaction was submitted by an authorized user.

Need a digital world equivalent of a signature.
Can seller fake the transaction?

<table>
<thead>
<tr>
<th>Trans. list k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (k, 1)</td>
</tr>
<tr>
<td>Trans. (k, 2)</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Trans. (k, n)</td>
</tr>
<tr>
<td>H(r(k-1)</td>
</tr>
<tr>
<td>rk</td>
</tr>
</tbody>
</table>

We need to check whether every transaction was submitted by an authorized user.

Digital world equivalent of a signature

0x7fce0b83e400...
Can seller fake the transaction?

<table>
<thead>
<tr>
<th>Trans. list k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (k, 1)</td>
</tr>
<tr>
<td>Trans. (k, 2)</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Trans. (k, n)</td>
</tr>
<tr>
<td>H(r(k-1)</td>
</tr>
<tr>
<td>rk</td>
</tr>
</tbody>
</table>

We need to check whether every transaction was submitted by an authorized user.

Digital world equivalent of a signature

0x7fce0b83e400...
Signatures

- **Undeniability**
  - If a document is signed by Person A, she cannot deny being responsible for a document

- **Unforgeability**
  - None can sign a document on behalf of Person A, (without permission of Person A)
  - Signature put on Document A cannot be transferred to Document B.

- **Public verifiability**
  - Everyone can verify signature on a document
Can seller fake the transaction?

<table>
<thead>
<tr>
<th>Trans. list k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trans. (k, 1)</td>
</tr>
<tr>
<td>Trans. (k, 2)</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Trans. (k, n)</td>
</tr>
<tr>
<td>H(r(k-1)</td>
</tr>
<tr>
<td>rk</td>
</tr>
</tbody>
</table>

We need to check whether every transaction was submitted by an authorized user.

Need a digital world equivalent of a signature.

- From **undeniability** we know that signed transaction occurred.
- **Unforgeability** assures us that if transaction was submitted by an authorized user.
- **Public verifiability** gives every node in a bitcoin network possibility to verify that the signature is correct.
Cryptographic signature schemes

Cryptographic scheme: GenKey, Sign, Verf

Generate public key, secret key pair
(pk, sk) <- GenKey

pk

m, s

sk

Accept if Verf(pk, s, m) accepts

s = Sign(sk, m; r)
Cryptographic signature schemes

Cryptographic scheme: algorithms: **GenKey, Sign, Verf**

Generate public key, secret key pair

\[(pk, sk) \leftarrow \text{GenKey} \]

**(pseudo-) anonymity in Bitcoin:**
- none needs to reveal his/her true identity to get \((pk, sk)\) pair – everyone can generate it himself
- use \(pk\)-s instead of names
- every entity can create as many \((pk, sk)\) pairs as it wants
- if signature key \(sk\) is lost, bitcoins connected to this entity are lost too
Cryptographic signature schemes

Cryptographic scheme:
algorithms: GenKey, Sign, Verf

Generate public key, secret key pair
\((pk, sk) \leftarrow \text{GenKey}\)

(pseudo-) anonymity in Bitcoin:
• none needs to reveal his/her true identity to get \((pk, sk)\) pair – everyone can generate it himself
• use \(pk\)-s instead of names
• every entity can create as many \((pk, sk)\) pairs as it wants
• if signature key \(sk\) is lost, bitcoins connected to this entity are lost too

Problem: How to make sure that everyone in the network has the same set of \(pks\)?

Check Public Key Infrastructure, we won’t cover this topic here 😞
Security of signature

Security game

We say that signature scheme is secure if for every PPT Homer, probability that Homer produces acceptable signature is negligible.

\[
\begin{align*}
&\text{Learning phase} \\
&m_0 \\
&\text{Sign}(sk, m_0; r_0) \\
&\text{...} \\
&m_k \\
&\text{Sign}(sk, m_k; r_k) \\

&\text{Challenge phase} \\
&m, s \\
&\text{such that} \\
&\text{m not in } m_0, ..., m_k \\
&\text{Verf}(pk, m, s) \text{ accepts}
\end{align*}
\]
Cryptography in Bitcoin

- Collision resilient hash functions
  - Provides discrete timing in a chain
  - Prevents from double spending
  - Allows fast verification of transaction lists chain

- Signature schemes
  - Protects users from fake transactions
Thank you*!

* and Matt Groening for creating The Simpsons