Transactions and Mining attacks

DAT655 Blockchain tech & app

Leander Jehl

Signatures and Transactions

Transactions

How can we create an application/cryptocurrency on a blockchain?

- What is in the blocks?
- How to build a meaningful application from it?
- Assume anyone can submit data to the blockchain.

Transactions

Digital Signatures

$$pk, sk \leftarrow setup(\kappa)$$

$$\sigma \leftarrow sign(sk, msg)$$

$$bool \leftarrow verify(\sigma, msg, pk)$$

Ideas:

- Use public key as identity.
- Put signed messages on the blockchain. $\langle msg \rangle_\sigma$
- Signed messages are called *transactions.*

Transactions

Accounts

Transactions are: $\langle pk_{from}, pk_{to}, value \rangle_\sigma$

State is: balance for each public-key

Checks:

- Is signature correct?
- Does pk_{from} have enough money?

Transactions

Accounts

Transactions are: $\langle pk_{from}, pk_{to}, value\rangle_\sigma$

Algorithm 1 Account transactions

```
balances := [pk]uint
for block in chain do
    for ⟨pk_from, pk_to, value⟩_σ in block.data do
        if !verify(pk_to||value, pk_from, σ) then
            continue
        if balances[pk_from] < value then
            continue
        balances[pk_from] −= value
        balances[pk_to] += value
```

Transactions

Accounts

Transactions are: $\langle pk_{from}, pk_{to}, value \rangle_\sigma$

State is: balance for each public-key

Checks:

- Is signature correct?
- Does pk_{from} have enough money?

Problems:

1. **How to deposit money?**
2. **Replay attack!**

Transactions

Accounts

Transactions are: $\langle pk_{from}, pk_{to}, value \rangle_\sigma$

Deposit:

- Give out some money
- Deposit with someone who has money

Replay attack:

- A signed transaction can be submitted multiple times.
- Sequence numbers!

Transactions

Accounts

Algorithm 2 Account transactions

$balances := [pk]\text{uint}$
$sqNrs := [pk]\text{uint}$
for $block$ **in** chain **do**
    for $\langle pk_{from}, pk_{to}, value, sqNr\rangle_\sigma$ **in** $block$.data **do**
        if $!\text{verify}(pk_{to}||value||sqNr, pk_{from}, \sigma)$ **then**
            continue
        if $balances[pk_{from}] < value$ **then**
            continue
        if $sqNrs[pk_{from}] = sqNr$ **then**
            $balances[pk_{from}] -= value$
            $balances[pk_{to}] += value$
            $sqNrs[pk_{from}]++$

Idea: do checks when adding transaction to chain.

Transactions
UTXO

UTXO: Unspent transaction output

Idea: *No balances but coins*

- For each coin store pk of owner and unique id
- Transaction spends some coints and creates new ones.

Transactions
UTXO

Transactions:

$$tx = \langle \underbrace{[(id_1, \sigma_1), (id_2, \sigma_2)]}_{\text{Inputs}}, \underbrace{[(pk_a, value_a), (pk_b, value_b)]}_{\text{Outputs}} \rangle$$

State is unspent outpus $map[id](pk, value)$

Transactions

UTXO

Transactions:

$$tx = \langle \underbrace{[(id_1, \sigma_1), (id_2, \sigma_2)]}_{\text{Inputs}}, \underbrace{[(pk_a, value_a), (pk_b, value_b)]}_{\text{Outputs}} \rangle$$

Valid if:

- Inputs refer to unspent outputs.
- Signatures are correct (with outputs public key)
- Value of all inputs larger or equal than all output values.

Transactions
UTXO

Algorithm 3 Transaction validation and maintenance of UTXO

$UTXO := map[id](value, pk)$
for $tx = \langle inputs, outputs \rangle$ **do**
    for $(id, \sigma) \in inputs$ **do**
        if $UTXO[id]$ does not exist **then**
            abort ▷ invalid transaction
        if verify$(tx, \sigma, UTXO[id].pk) ==$ **false then**
            abort ▷ invalid transaction
    if sum of values of inputs $<$ sum of values of new outputs **then**
        abort ▷ invalid transaction
    for $((id, \sigma) \in inputs$ **do**
        remove$(UTXO[id])$ ▷ output spent
    for $((pk, value) \in inputs$ **do**
        $UTXO[newid] = (pk, value)$ ▷ add new outputs

Transactions

UTXO

Transactions:

$$tx = \langle \underbrace{[(id_1, \sigma_1), (id_2, \sigma_2)]}_{\text{Inputs}}, \underbrace{[(pk_a, value_a), (pk_b, value_b)]}_{\text{Outputs}} \rangle$$

- No replay attack
- What to sign: $\langle [id_1, id_2], [(pk_a, value_a), (pk_b, value_b)] \rangle$

Transactions

Accounts vs. UTXO

Assuming only valid transactions on chain,
how to verify that a pk has money.

Accounts: Check all received and sent transactions.

UTXO: Check received output and that it is unspent.

Transactions
Accounts vs. UTXO

Assuming only valid transactions on chain,
how to verify that a pk has money.

Accounts: Check all received and sent transactions.

UTXO: Check received output and that it is unspent.

Does UTXO provide anonymity/prevent tracing?

Transactions

Accounts vs. UTXO

Assuming only valid transactions on chain,
how to verify that a pk has money.

Accounts: Check all received and sent transactions.

UTXO: Check received output and that it is unspent.

Does UTXO provide anonymity/prevent tracing?

- Also in UTXO transactions from one sender can be traced.
- But most untracable solutions build on UTXO

Take away

Transactions/state changes are recorded in the blockchain.

Application state can be recreated by applying all transactions.

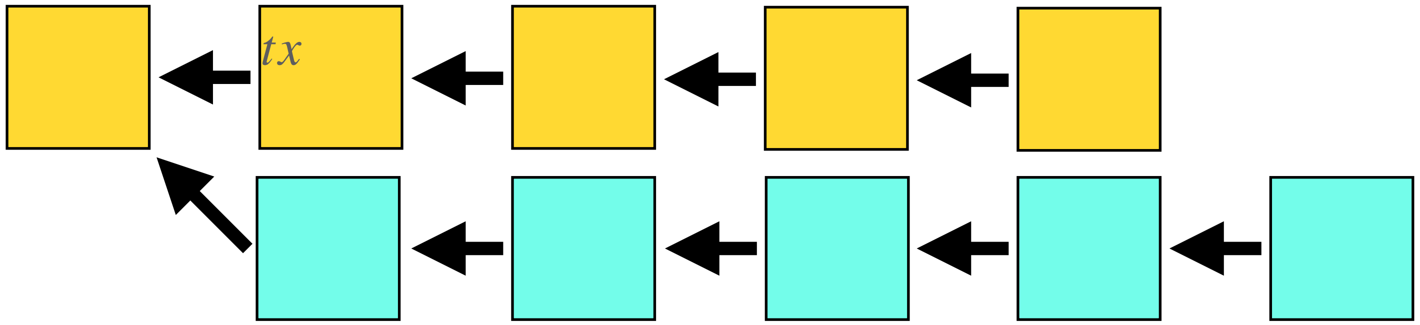
Attacks

Attacks

51% attack

- Assume the attacker has $\alpha > 50\,\%$ of the hashing power.
    - Attacker can grow a private chain faster than the public chain.

A private chain is a fork with blocks not propagated through the network.



Attacker can:
- Double spend
- Get all the reward

Attacks

Stubborn mining:

- Attacker does not follow longest chain rule.

Selfish mining:

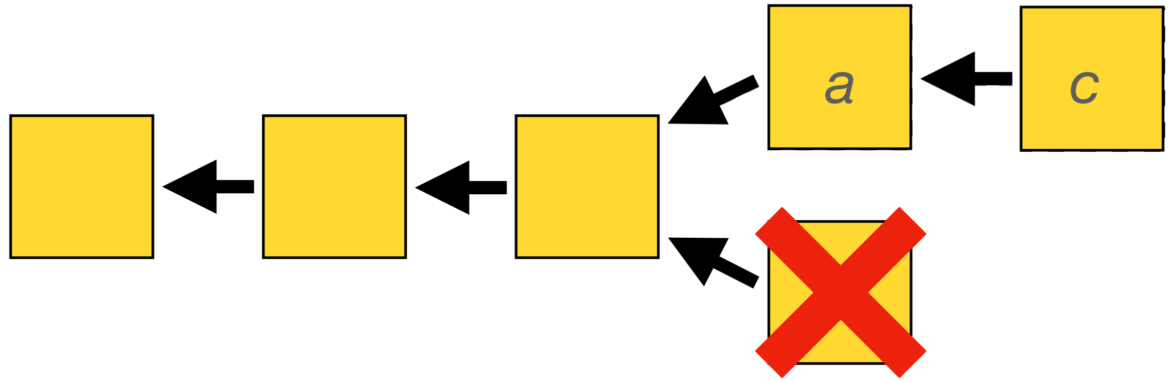
- Attacker keeps blocks secret.

Attacks

Selfish mining

Case 1, successfull attack:

1. attacker finds block *a,* keeps it secret
2. attacker finds block *c,* keeps it secret
3. other nodes find block *b* and propagate it
4. attacker propagates blocks *a* and *c*

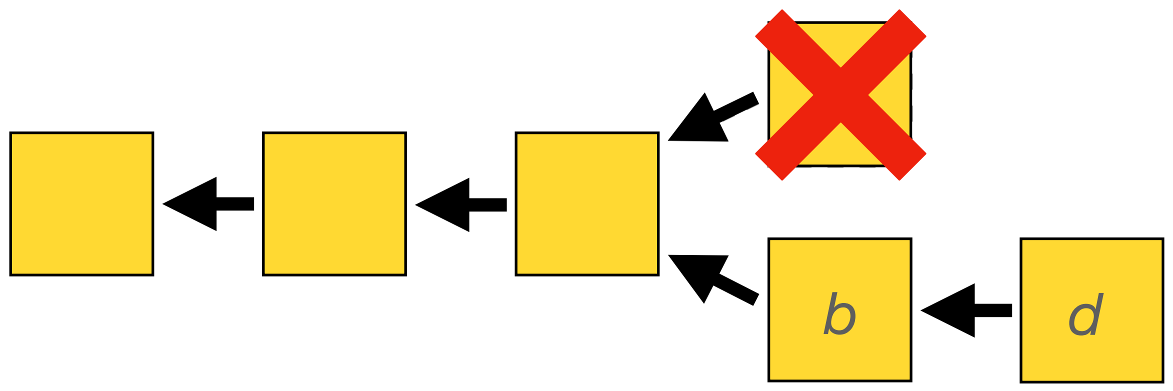


Attacks

Selfish mining

Case 2, unsuccessfull attack:

1. attacker finds block *a,* keeps it secret
2. other nodes find block *b* and propagate it
3. attacker propagates block *a*
4. other nodes find block *d* extending *b*

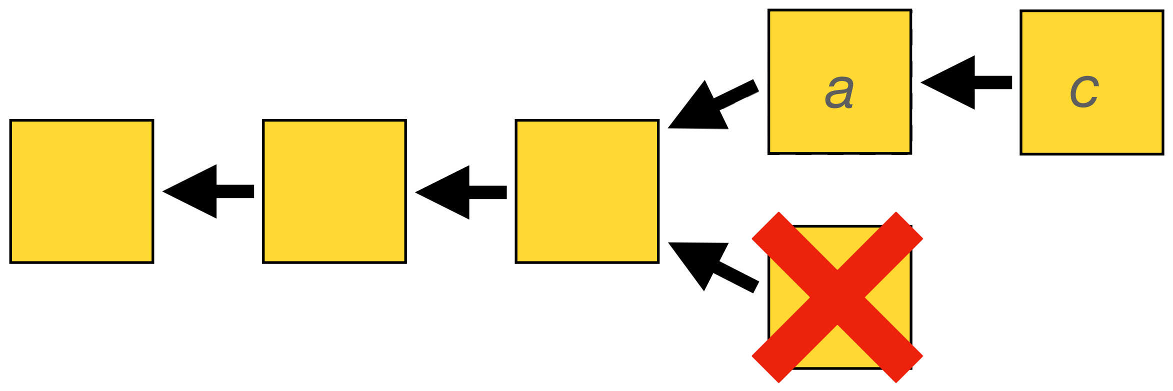


Attacks

Selfish mining

Case 3, kind of successfull attack:

1. attacker finds block *a,* keeps it secret
2. other nodes find block *b* and propagate it
3. attacker propagates block *a*
4. some node finds block *c* extending *a*

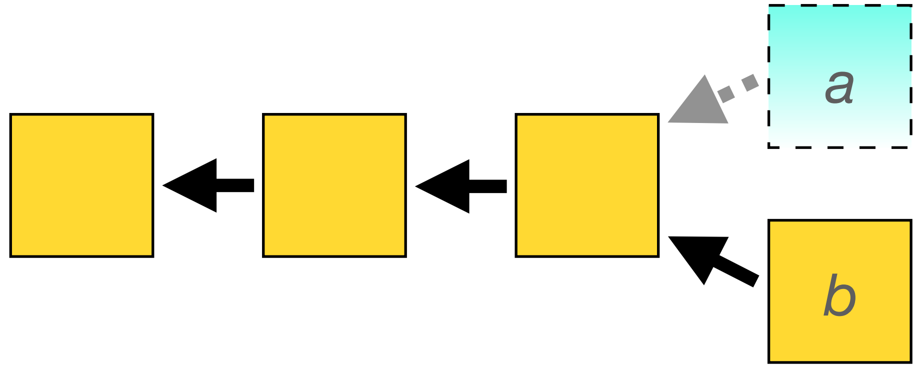


Attacks

Selfish mining

To get Case 3 instead of Case 2 attacker needs to

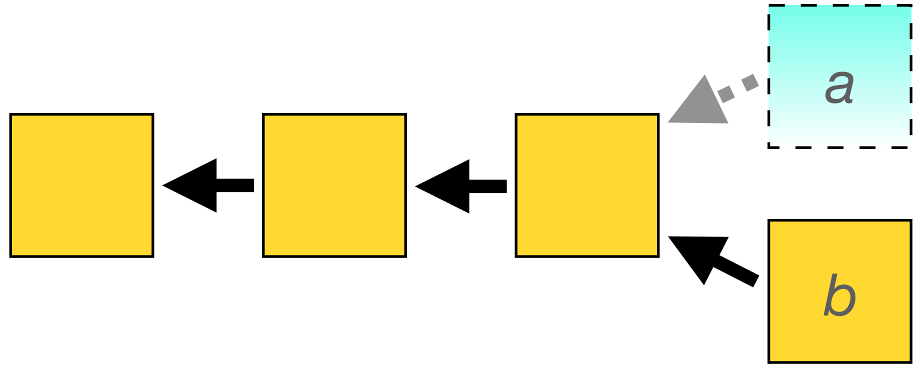
- detect new blocks fast
- propagate its block faster



Attacks

Selfish mining - take away

- Attacker does not get more blocks, but others get less.
- Good control of network makes attack work better.



Attacks

Selfish mining

Algorithm 6 Selfish mining

Idea: Mine secretly, without immediately publishing newly found blocks
Let l_p be length of the public chain
Let l_s be length of the secret chain
if a new block b_p is published, i.e. l_p has increased by 1 **then**
  if $l_p > l_s$ **then**
    Start mining on b_p
  else if $l_p = l_s$ **then**
    Publish secretly mined block b_s
    Mine on b_s and immediately publish new block
  else if $l_p = l_s - 1$ **then**
    Push all secretly mined blocks

Attacks

Selfish mining

α the attackers hashing power, and
γ be the attackers network power.

Selfish mining is profitable, if

$$\alpha > 0.33$$

$\alpha > 0.25$ and $\gamma > 0.5$

$\alpha > 0$ and $\gamma = 1$

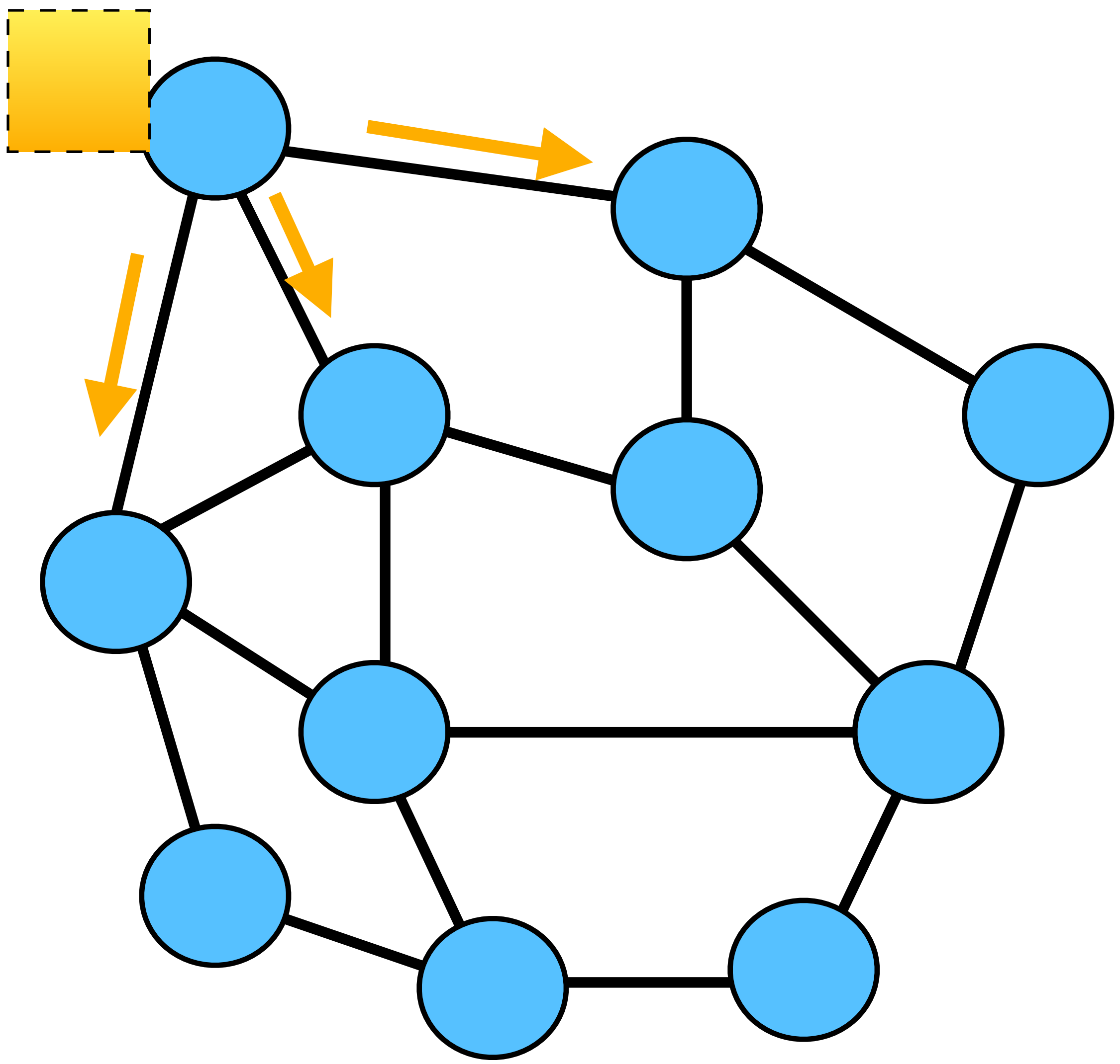
Attacks

Delivery denial

Broadcast block:

- Broadcast inventory message including block hash
- Receiving new inventory, request block
- Send block

Block is only send from one neighbor



Attacks

Delivery denial

Broadcast block:

- Broadcast inventory
- Request block
- Send block

Attack

- Broadcast inventory
- Do not send out blocks

  Victims wait for timeout.

Bitcoin

Downsides

Throughput at most 7tx per second

Confirmation latency approx 1h

Enormous energy consumption