Algorand transactions and smart signatures

Massimo Bartoletti
University of Cagliari

Roberto Zunino
University of Trento

Overview

Algorand is a new-gen permissionless blockchain:

- New consensus protocol:
  - Proof-of-Stake, with precise security proofs
  - Deterministic finality (no forks)
- **Transactions & smart contracts:**
  - complex transaction mechanisms
  - **smart signatures** (= stateless smart contracts)
  - smart contracts (stateful)

Many subtle features: improper use may affect the security of contracts!

Lecture summary

1. Simple transfers
2. Multisig accounts
3. Smart signatures
    - Intuition
    - Birthday present
    - Oracle
4. Custom assets
5. Atomic groups of transactions
6. Rekeying

Not enough time to give full details
⇒ Algorand Developer Portal

Do your experiments (using the sandbox)

Install docker and the Algorand sandbox:

https://github.com/algorand/sandbox

For the examples we use the CLI tools:

```
# sudo ./sandbox up testnet
# sudo ./sandbox enter algod
```

(for smart contract development, better using SDKs)

Simple transfers

Accounts & transactions

Algorand can be seen as a state machine, where:

- states are sets of **accounts**;
- state transitions are triggered by **transactions**.

Example:

$$\mathbf{A}[5{:}\mathrm{Algo}] \mid \mathbf{B}[10{:}\mathrm{Algo}] \mid \ldots$$

$$\xrightarrow{T} \mathbf{A}[9{:}\mathrm{Algo}] \mid \mathbf{B}[6{:}\mathrm{Algo}] \mid \ldots$$

A,**B** = **users** (identified by addresses)

A[...] = user account

Using the **goal** tool

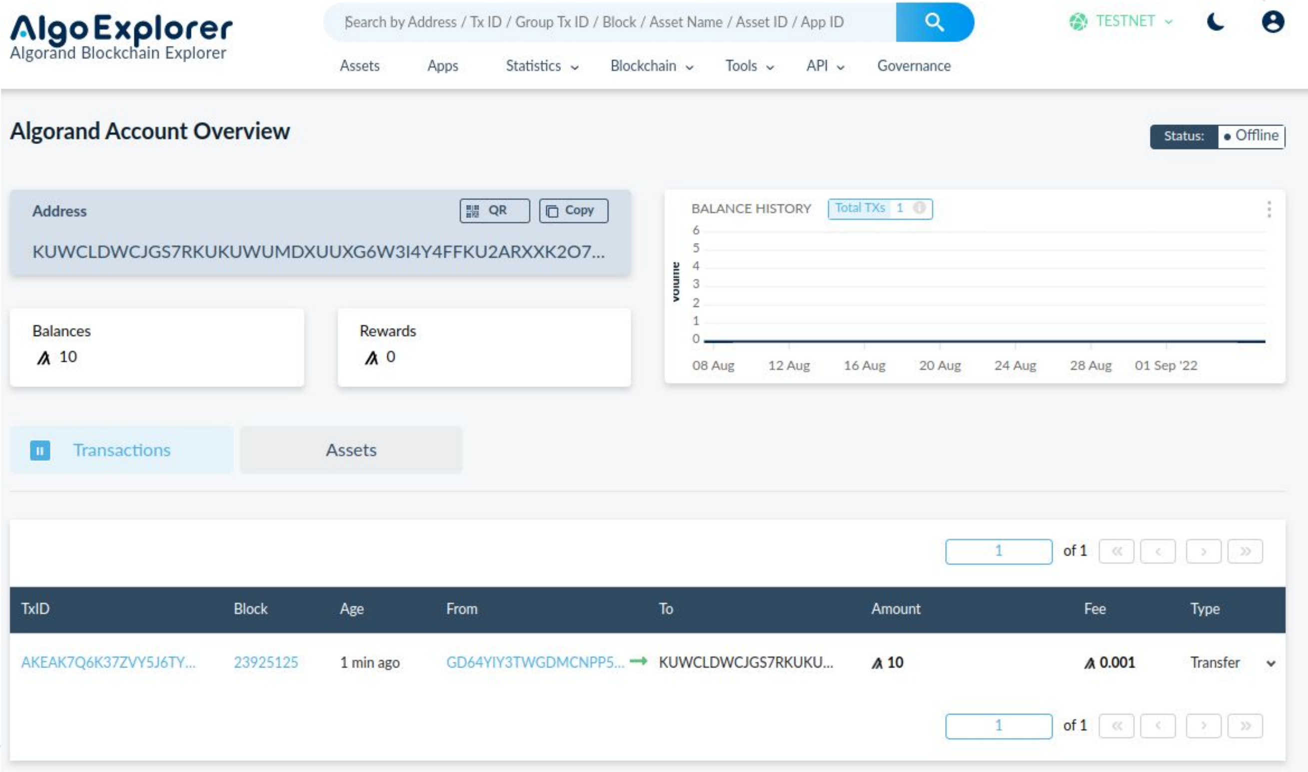
```
# goal account new

# goal account rename Unnamed-0 A

# goal account list

[offline]   O     IPX7RJQPIHEEESTRRKF4QGNERGZE325NNFSYA5IX76VZRUTPQXZWNEMS7Q  0 microAlgos

[offline]   A     2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU  5000000 microAlgos

[offline]   B     3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY  10000000 microAlgos
```

Algo faucet

Use a faucet to get free testnet Algos for experiments

https://bank.testnet.algorand.network/

Algorand testnet explorer



https://testnet.algoexplorer.io/

Pay transactions

A[5:Algo] | B[10:Algo] | …

"B pays 4 Algo to A"

$\longrightarrow$

A[9:Algo] | B[6:Algo] | …

Pay transactions

```
# goal clerk send -f B -t A -a 4000000 -o T

# goal clerk inspect T

T[0]
{
  "txn": {
    "amt": 4000000,
    "fee": 1000,
    "fv": 24228391,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 24229391,
    "note": "Xtjt60VLkgA=",
    "rcv": "2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU",
    "snd": "3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY",
    "type": "pay"
  }
}
```

tx type

Pay transactions

```
# goal clerk send -f B -t A -a 4000000 -o T

# goal clerk inspect T
```


```
T[0]
{
  "txn": {
    "amt": 4000000,
    "fee": 1000,
    "fv": 24228391,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 24229391,
    "note": "Xtjt60VLkgA=",
    "rcv": "2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU",
    "snd": "3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY",
    "type": "pay"
  }
}
```


sender

Pay transactions

```
# goal clerk send -f B -t A -a 4000000 -o T

# goal clerk inspect T

T[0]
{
  "txn": {
    "amt": 4000000,
    "fee": 1000,
    "fv": 24228391,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 24229391,
    "note": "Xtjt60VLkgA=",
    "rcv": "2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU",
    "snd": "3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY",
    "type": "pay"
  }
}
```

receiver

Pay transactions

```
# goal clerk send -f B -t A -a 4000000 -o T

# goal clerk inspect T

T[0]
{
  "txn": {
    "amt": 4000000,
    "fee": 1000,
    "fv": 24228391,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 24229391,
    "note": "Xtjt60VLkgA=",
    "rcv": "2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU",
    "snd": "3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY",
    "type": "pay"
  }
}
```

= 4 Algos

The amount is in µAlgos = 10^{-6} Algos

(atomic amount of currency)

Pay transactions

```
# goal clerk send -f B -t A -a 4000000 -o T

# goal clerk inspect T

T[0]
{
  "txn": {
    "amt": 4000000,
    "fee": 1000,
    "fv": 24228391,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 24229391,
    "note": "Xtjt6OVLkgA=",
    "rcv": "2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU",
    "snd": "3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY",
    "type": "pay"
  }
}
```

tx fee (min = 0.001 Algo)

Pay transactions

```
# goal clerk send -f B -t A -a 4000000 -o T

# goal clerk inspect T

T[0]
{
  "txn": {
    "amt": 4000000,
    "fee": 1000,
    "fv": 24228391,
    "gen": "testnet-v1.0",
    "gh": "SGO1GKSzyE7IEPItTxCByw9x8FmnrCDexi9/cOUJOiI=",
    "lv": 24229391,
    "note": "Xtjt60VLkgA=",
    "rcv": "2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU",
    "snd": "3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY",
    "type": "pay"
  }
}
```

first valid

last valid (≤ fv+1000)

Validity interval in rounds
(one round ~ 4 seconds)

No double spending:
we can not send the same tx twice

To make two equal payments,
use distinct fv/lv fields

Signing a transaction

```
# goal clerk sign -i T -o TB

# goal clerk inspect TB

TB[0]
{
  "sig": "Q+vW9FFI8cqTj7iJ4dM92s5LvxT4c/qlaauH0qhsiM6Zyey8TBZT2obBpYz958fKc/PDo7h1uPZnDsDjQVQFAw==",
  "txn": {
    "amt": 4000000,
    "fee": 1000,
    ...
    ...
    ...
  }
}
```

Sending the transaction

```
# goal clerk rawsend -f TB

# goal account list

[offline]   O     IPX7RJQPIHEEESTRRKF4QGNERGZE325NNFSYA5IX76VZRUTPQXZWNEMS7Q  0 microAlgos

[offline]   A     2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU  9000000 microAlgos

[offline]   B     3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY  5999000 microAlgos
```

A shortcut

To generate a transaction, sign it, and send it immediately

```
# goal clerk send -f A -t B -a 4000000
```

(note the omission of the **-o** flag to write the transaction in a file)

Back to the abstract notation…

A[5:Algo] | B[10:Algo] | …

$$\xrightarrow{T = \text{pay}(\text{snd}=B,\ \text{rcv}=A,\ \text{amt}=4\ \text{Algo},\ \text{fee}=0.001\ \text{Algo}),\ \text{sig}_B(T)}$$

A[**9**:Algo] | B[**5.999**:Algo] | …

For simplicity, we will often omit fees & signatures

Closing an account

A[5:Algo] | B[5:Algo] | C[2:Algo]

pay(snd=**B**,rcv=**A**,amt=4 Algo,close=**C**)

A[**9**:Algo] | C[**3**:Algo]

Tx type is still **pay**, but the actual behaviour changes if the field **close** is set

B sends
4 Algos to A,
and <u>all the rest</u> to C

After that, B is closed
(to reopen it, send new Algos to it)

Closing an account

```
# goal clerk send -f B -t A -a 4000000 -c C -o T2
# goal clerk inspect T2
T2[0]
{
  "txn": {
    "amt": 4000000,
    "close": "IPX7RJQPIHEEESTRRKF4QGNERGZE325NNFSYA5IX76VZRUTPQXZWNEMS7Q",
    "rcv": "2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU",
    "snd": "3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY",
    "type": "pay"
    ...
  }
}
```

C

A

B

Balance constraints

Can we litter the blockchain with crumb-valued accounts?

A[1:Algo] $\xrightarrow{\text{pay(snd=A,rcv=B1,amt=0.0001:Algo)}}$

A[...] | **B1**[0.0001:Algo] $\xrightarrow{\text{pay(snd=A,rcv=B2,amt=0.0001:Algo)}}$

A[...] | **B1**[0.0001:Algo] | **B2**[0.0001:Algo]

?!?

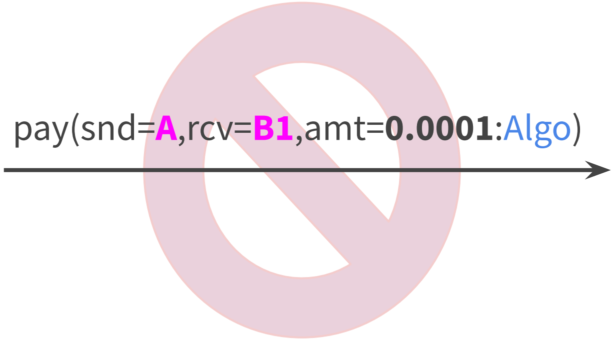
...

Balance constraints

No, we can not!

A[1:Algo]

A[...] | B1[0.0001:Algo]



The Algos in any account must be **at least** 0.1 Algo (or zero, closing it)

A small puzzle

A[5:Algo] | **B**[5:Algo]

pay(snd=**A**,rcv=**B**,amt=5 Algo)

B[10:Algo]

?!?

What would happen if an adversary sent 1 μAlgo to A first?

A small puzzle

A[5:Algo] | B[5:Algo] $\xrightarrow{\text{pay(snd=A,rcv=B,amt=5 Algo)}}$ B[10:Algo]

If an adversary sends 1 μAlgo to A first ...

... the pay transaction is no longer valid since the balance of A would be too low!

No way to prevent an account from receiving Algos

To prevent the issue, Algorand

- considers the above pay transaction as **invalid**
- requires to use **close** to make an account empty

Multisig

Multisig

({A,B,C},2)[5:Algo]

Transactions from this account can be authorized by any 2 users out of **A**, **B**, **C**

Multisig

({A,B,C},2)[5:Algo] | D[1:Algo] | ...

$\xrightarrow{\text{T= pay(snd=(\{A,B,C\},2), rcv=D, amt=1 Algo), } sig_A\text{(T), } sig_B\text{(T)}}$

({A,B,C},2)[4:Algo] | D[2:Algo] | ...

Using goal: multisig example

```
# goal account multisig new AA... BB... CC... -T 2      2 out of 3 multisig

# goal account rename Unnamed-0 ABC

// here: add 0.2 Algos to multi-1 as usual

# goal clerk send -a 1000 -f ABC -t D -o Tx        create tx to pay 1000 µAlgos to D

# goal clerk multisig sign -t Tx -a AA...          add first sig (overwrites file Tx)

# goal clerk multisig sign -t Tx -a BB...          add second sig

# goal clerk rawsend -f Tx
```

Smart signatures

Smart Signatures: basic idea

A[5:Algo] $\xrightarrow{\text{T=pay(snd=A, ...), } sig_A\text{(T)}}$

({A,B,C},2)[5:Algo] $\xrightarrow{\text{T=pay(snd=(\{A,B,C\},2), ...), } sig_A\text{(T), } sig_B\text{(T)}}$

To authorize the transactions, one or more cryptographic signatures are needed

ssig[5:Algo] $\xrightarrow{\text{T=pay(snd=ssig, ...), } arg_0\text{, ...}}$

Generalize:

a program **ssig** decides whether to accept T

any user can publish their programs

Smart Signatures: basic idea

ssig[5:Algo] $\xrightarrow{T=pay(snd=\textbf{ssig}, \ldots), arg_0, \ldots}$

- ssig is an account
  - can receive Algo from anyone (no way to constrain)
  - can send Algo in a program-controlled way

- The actual code of ssig is written in the **TEAL** language
  - stack-based assembly-like language
  - many opcodes to inspect T and the arg_i arguments
  - general: we can put (almost) arbitrary constraints on T, arg_i, ...
  - compiles to AVM bytecode

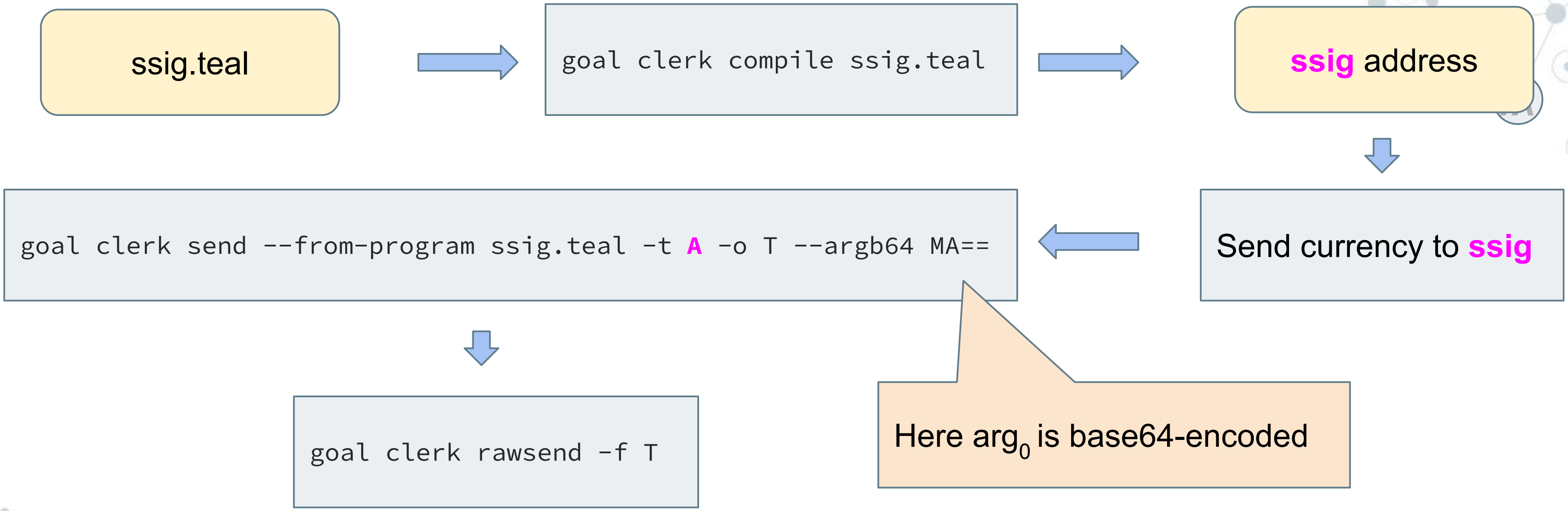
- The address of ssig is the hash of the bytecode

TEAL-enforceable requirements (examples)

ssig[5:Algo] $\xrightarrow{T=pay(snd=\mathbf{ssig}, \ldots), arg_0, \ldots}$

- the amount must be less than 5 Algo
- the receiver must be one among a few hard-coded addresses
  - including addresses of other smart signatures !
- arg_1 must be a signature by **A** of message "hello!"
- arg_2 is a hash preimage of some known constant
- temporal constraints (fv / lv)
- ...

Possible TEAL workflow



TEAL workflow: adding signatures as arguments

```
goal clerk send --from-program ssig.teal -t A -o T --argb64 MA==
```

• • •

```
goal clerk tealsign --sign-txid --keyfile keyfile.sk --lsig-txn T --set-lsig-arg-idx 1
```

```
goal clerk rawsend -f T
```

Add arg_1= signature of T to T

(see the docs for more details)

Generating TEAL through PyTeal

Python PyTeal code
ssig.py

```
python ssig.py > ssig.teal
```

ssig.teal

```
goal clerk compile ssig.teal
```

• • •

Writing PyTeal code is more programmer-friendly

Example: birthday present

A sends a 1000 Algo present to a smart signature **bday**, which allows **B** to redeem the present after his birthday (2023-01-01)

bday checks the following conditions:

- the tx has type "pay"
- the receiver is **B**
- the first round where the tx is valid (fv) is >= round(2023-01-01)

Birthday's present in PyTeal

```
from pyteal import *

B = Addr("3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY")
birthday_round = Int(4444444)   # round expected on 2023-01-01

def bday():
    return And(Txn.type_enum() == TxnType.Payment,
               Txn.receiver() == B,
               Txn.first_valid() >= birthday_round)

if __name__ == "__main__":
      print(compileTeal(bday(), Mode.Signature))
```

Birthday's present in TEAL

```
#pragma version 2
txn TypeEnum
int pay
==
txn Receiver
addr 3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY
==
&&
txn FirstValid
int 44444444
>=
&&
return
```

Attack

The previous smart signature is **vulnerable**!

On the birthday date, **M** steals the present:

bday[1000:Algo] $\xrightarrow{\text{T=pay(snd=bday, rcv=B, amt=0 Algo, fv=44444444, close=M)}}$

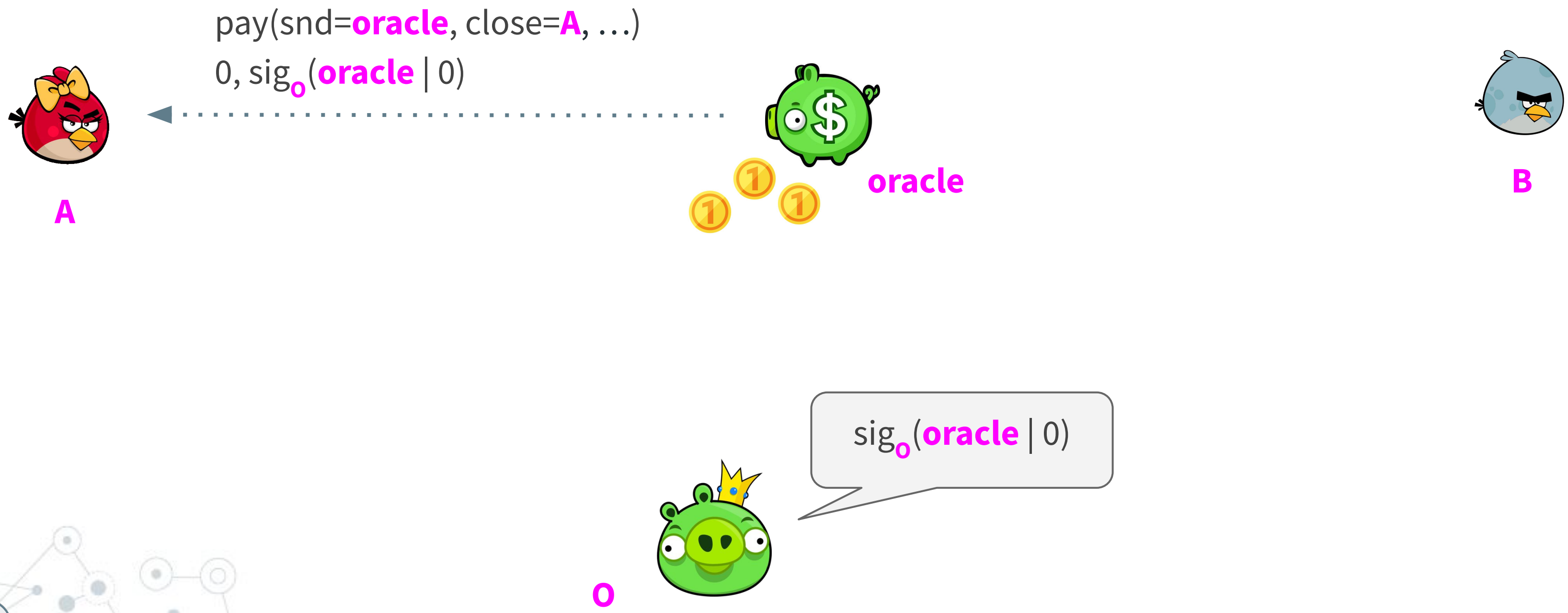
T=pay(snd=**bday**, rcv=**B**, amt=0 Algo, fv=44444444, close=**M**)

The TEAL script must be more careful and check the other transaction fields!

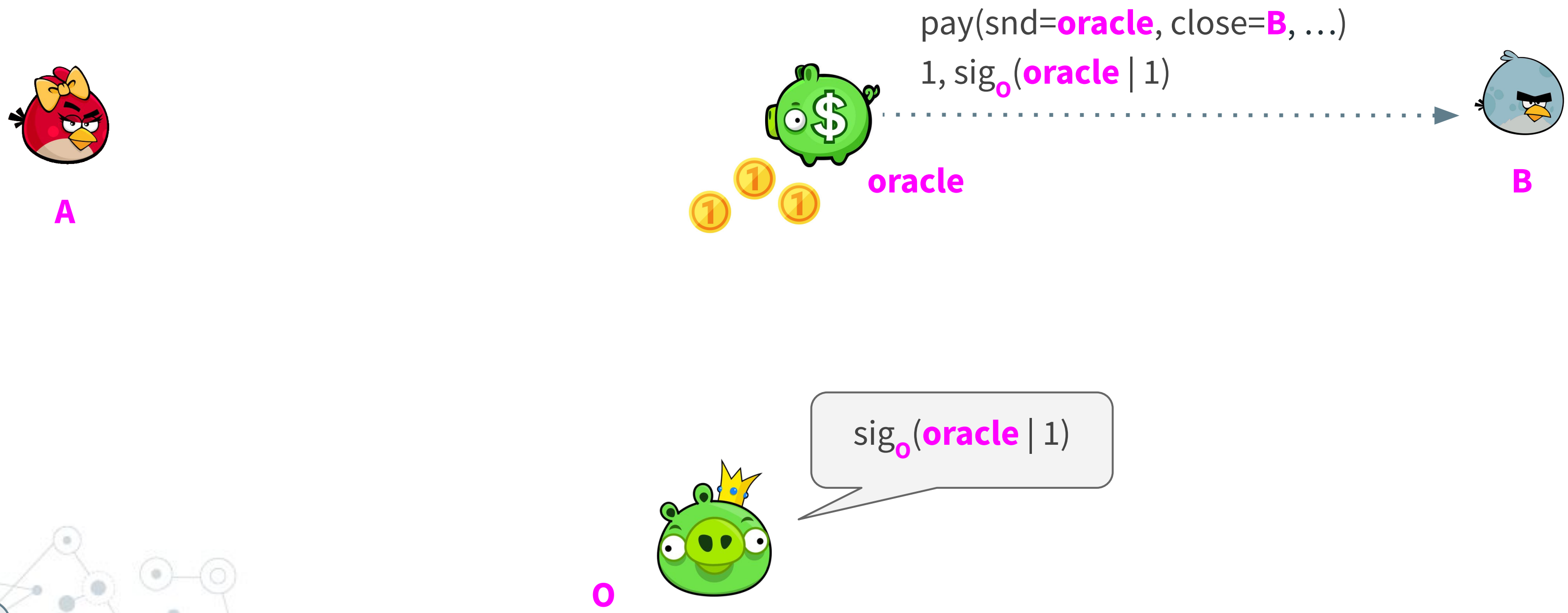
(Exercise: other attacks are possible beyond the above one…)

Oracle: intuition

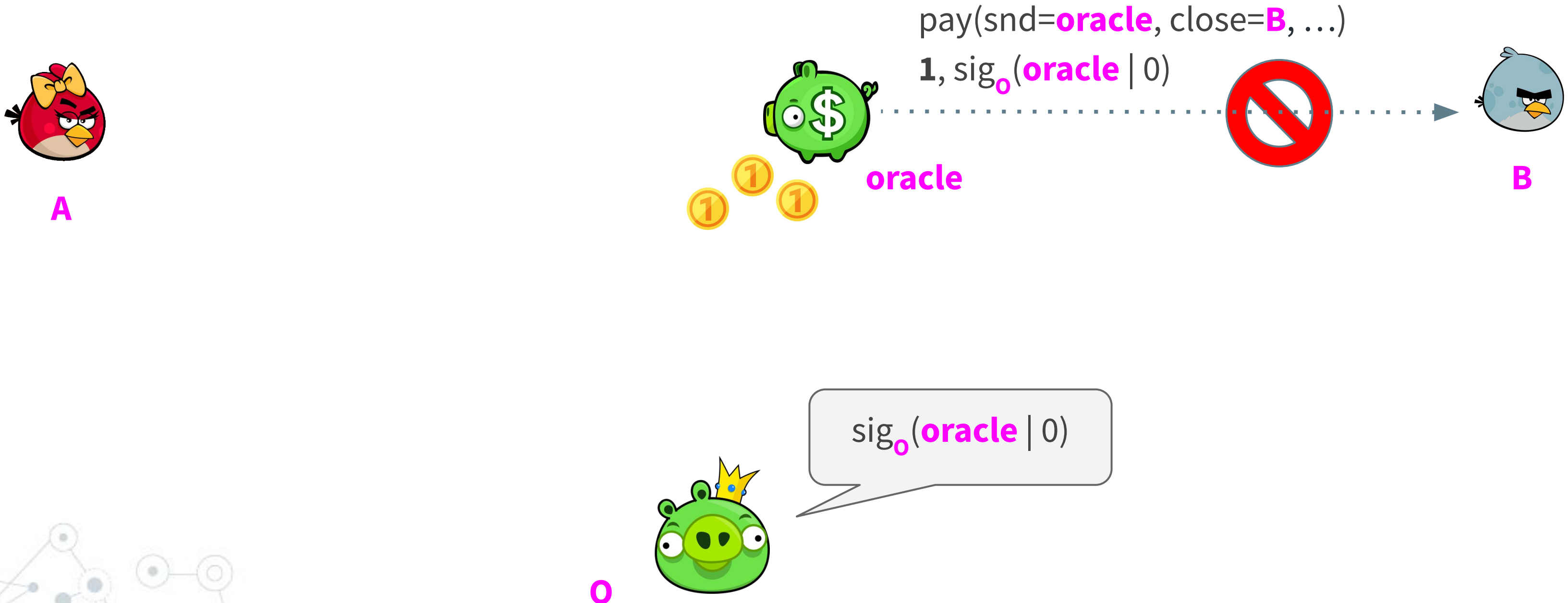
Oracle: intuition



Oracle: intuition



Oracle: intuition

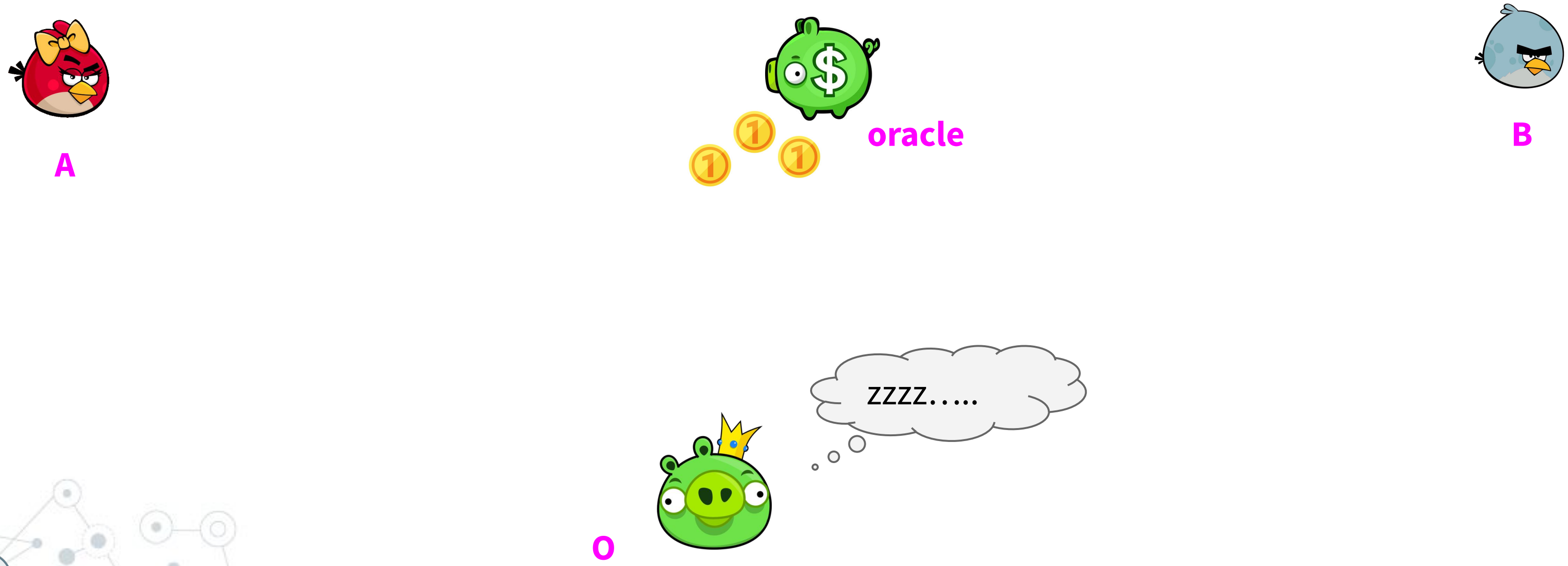


Oracle

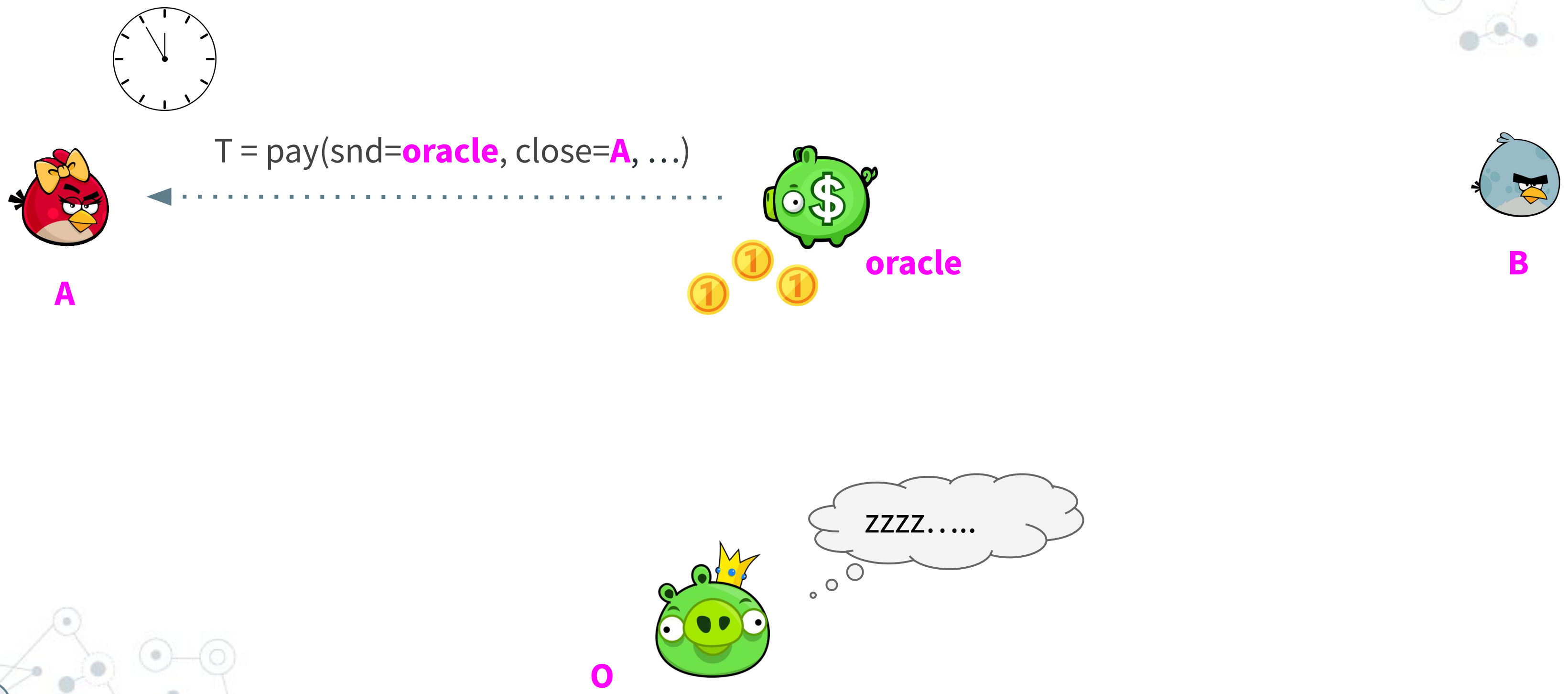
github.com/algorand-school/algorand-school/tree/main/contracts/oracle

```
from pyteal import *

A = Addr("2GYIH5HXKDNXA3F7BBIAT5IX744E2WY75GIQRLEWURVRK3XXDQ6LMRAHXU")
B = Addr("3MTDHUNSO4RXC3ZPJ67C7TLEOFHFO2UNXHE34PN52VN2CSNYSEOXXHPFNY")
O = Addr("IPX7RJQPIHEEESTRRKF4QGNERGZE325NNFSYA5IX76VZRUTPQXZWNEMS7Q")

arg0 = Bytes("0")
arg1 = Bytes("1")

def oracle():

    txPay    = And(Txn.type_enum() == TxnType.Payment, Txn.amount() == Int(0))
    versigO  = Ed25519Verify(Arg(0), Arg(1), O)
    closeToA = And(Arg(0) == arg0, Txn.close_remainder_to() == A)
    closeToB = And(Arg(0) == arg1, Txn.close_remainder_to() == B)

    return And(txPay, versigO, Or(closeToA, closeToB))

if __name__ == "__main__":
      print(compileTeal(oracle(), Mode.Signature))
```

Exercise: dealing with a lazy oracle



Exercise: dealing with a lazy oracle



Custom assets

Custom assets: generation

B[5:Algo] $\xrightarrow{\text{acfg(snd=}\mathbf{B}\text{, t=1000, ...)}}$

B[5:Algo, 1000:FooCoin]

Anti-spam constraint:

An account must contain at least 0.1 Algo for each currency type

Custom assets: transfer

A[5:Algo] | B[5:Algo, 1000:FooCoin]

$\xrightarrow{\text{xfer(asnd=B, arcv=A, aamt=1, xaid=FooCoin, ...)}}$?

A[5:Algo, 1:FooCoin] | B[5:Algo, 999:FooCoin]

Custom assets: transfer

A[5:Algo] | B[5:Algo, 1000:FooCoin]

xfer(asnd=B, arcv=A, aamt=1, xaid=FooCoin, ...)

A must **opt-in** to accept user-defined assets

A[5:Algo, 1:FooCoin] | B[5:Algo, 999:FooCoin]

Custom asset: opt-in

A[5:Algo] | **B**[5:Algo, 1000:FooCoin] —"**A** opts-in FooCoin"→

A[5:Algo, **0:FooCoin**] | **B**[5:Algo, 1000:FooCoin] —"**B** transfers"→

A[5:Algo, **1**:FooCoin] | **B**[5:Algo, **999**:FooCoin]

Custom assets: clawback

A[5:Algo, 1:FooCoin] | B[1:Algo, 999:FooCoin] $\xrightarrow{\text{"B performs a clawback"}}$

A[5:Algo, **0**:FooCoin] | B[1:Algo, **1000**:FooCoin]

If B is the clawback address of FooCoins, he can “steal” FooCoins from other accounts!

To be sure you really own a token, check that there is no clawback address set.

Atomic groups

Atomic groups of transactions

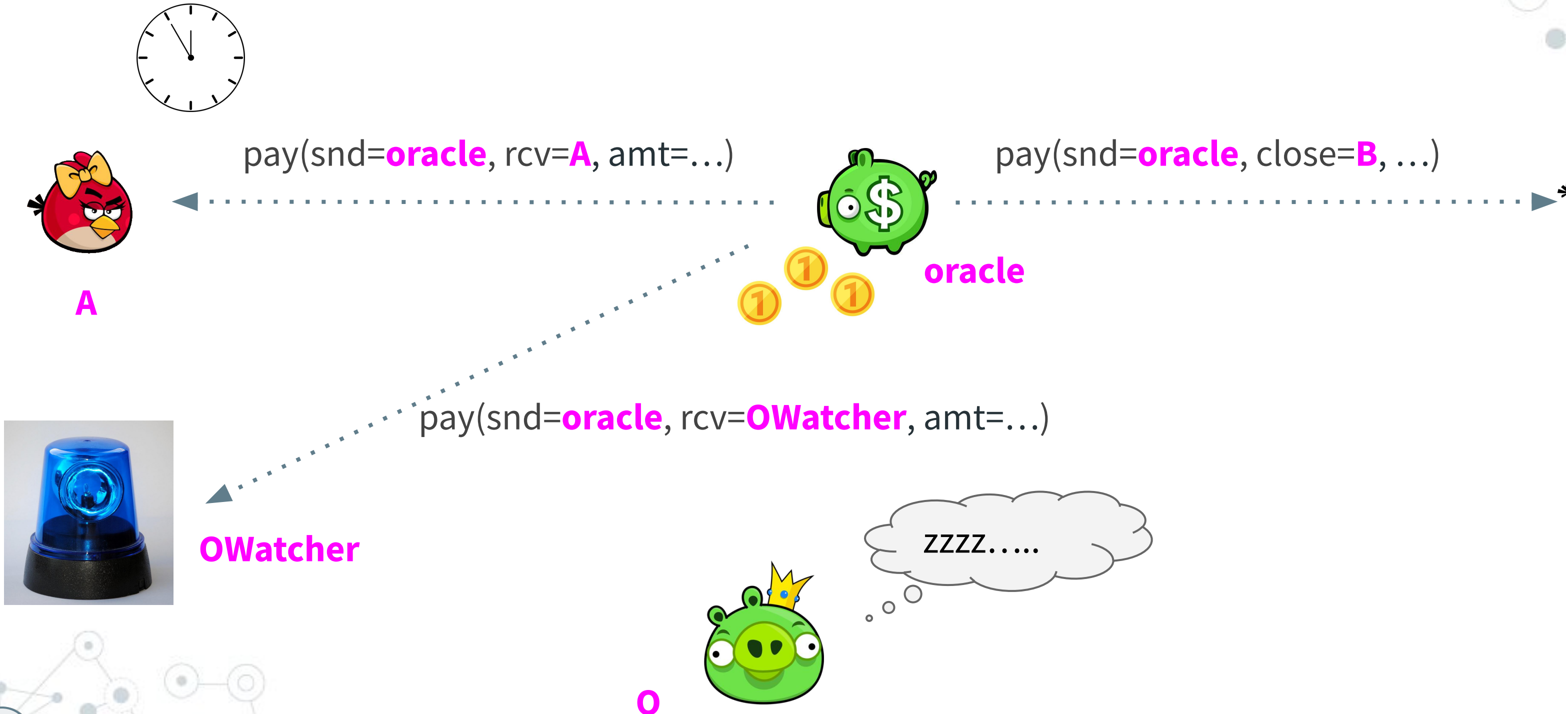
A[5:Algo, 0:FooCoin] | B[1:Algo, 2:FooCoin]

{ pay(snd=A, rcv=B,amt=1),
xfer(asnd=B, arcv=A,aamt=1, xaid=FooCoin) }

The two transactions are performed atomically

A[4:Algo, 1:FooCoin] | B[2:Algo, 1:FooCoin]

Exercise: dealing with a lazy oracle splitting the amount



Rekeying

Rekeying

A[5:Algo] | B[1:Algo] | C[1:Algo] $\xrightarrow{T=pay(snd=A,\ rcv=B,\ amt=1,\ rekey=C),\ sig_A(T)}$

A^C[4:Algo] | B[2:Algo] | C[1:Algo] $\xrightarrow{T=pay(snd=A,\ rcv=B,\ amt=1),\ sig_C(T)}$

A^C[3:Algo] | B[3:Algo] | C[1:Algo]

After the rekeying, the signature by C is needed!

Beware of rekeying in smart signatures!

Conclusions

More transactions features…

- constraints on accounts
- generate assets, freeze & unfreeze assets
- transaction lease (for mutual exclusion)
- asset managers, burn & delegate
- …

Contracts authorize transactions ⇒
they must handle these features properly!

Expressivity of smart signatures properties

- Oracle bet ✅
- Hash time-locked contract (HTLC) ✅
- 2-players lottery ✅
- DeFi: escrow, periodic payment, limit order, split
- Finite-state machines
- …

Proving the security of a contract

- consider different attack scenarios depending on who is assumed to be honest
- formalise the strategies of “honest” participants
- establish which transactions can be fired in each attack scenario
- check that in the reached state the honest participants have fulfilled their goals

Thank you