The Keys

3 uses of cryptographic keys

(1) Authentication of payments

Sign that you made that payment

- Static – long-term key

(2) Role credentials – block signing

Get your lottery ticket, prove that you won

- Static – long-term key
- Unique-signature key
- Can conveniently be the the same key as (1)

.. although (1) doesn't require uniqueness property

(3) Role duties – block signing

Sign your work

- Single use (ephemeral) key

Unique Keys

Keys that generate exactly one verifiable signature for a given message

- Static user key (PK, SK)
  PK: public-key component, SK: secret-key component
  - Randomizes for cryptographic sortition – along with hash function
    - the "ticket#" – also hides
    - Q_r – the seed to go into sortition of next round
  - Also serves as user ID
- Verifier $V(s, m, PK)$ knows that "that" signature $s=sign_{PK}(m)$ of "this" message m signed by "this" user PK is unique
  - No other $s'=sign_{PK}(m)$ that can pass $V(s', m, PK)$
- Unique signature
  - Every time $sign_{PK}(m)$ is invoked
  - Every time $sign_{PK\text{-}sub}(m)$ is invoked on a subkey of PK
    - PK can verify $sign_{PK\text{-}sub}(m)$ without having to know *PK-sub*
- A feasible implementation of *Verifiable Random Functions*[*]

[*]Micali, S., Rabin M. and Vadhan, S., (1999) Verifiable Random Functions, FOCS, New York

Unique Keys – Otherwise ..

Keep generating "ticket#"s till you obtain a winning one

- Keep signing till you get the signature to ..
  - .. get you a role in the round
    cryptographic sortition
  - .. get you/your bots roles the next round
    round leader generating the seed Q_r
- Lets computing power into Algorand
- Gives options to adversary
- Another scenario: inject in new users with the "right" PKs, unique keys still, for the next round
  - PoS – chances of winning are proportional to stakes
  - Look-back parameter for honesty

Look-back Parameter

*Users that are **k**+ rounds old in the system*

- *k*: Look-back parameter
- Role players of current round
  - Restricted to users that existed in the system past *k* rounds
  - Adversary/bots newly injected into system
  - Set aside possibility of adversary manipulating "who"
    - Can influence only if closer to round
- Suggested value minimum *k*=40 [CM17]

Ephemeral Keys

Single-use keys – Destroy right after ***sign()***

- Role players sending out messages during round
  - Have to authenticate their messages – *sign*()
  - Revealing themselves

    Messages tell who the signing role players are
- Destroy key right after *sign*() – can't *sign*() again
  - Even if corrupted by the adversary by then
  - Unique for each round/step – no other key can be issued

Otherwise

- Corrupt verifiers to produce a second, fake block
  - Know all verifiers eventually
  - Have all verifiers sign a fake block
  - Fork – circulate the fake block right off

    After the legitimate block – less chance

Forks

- Can not fork
  - Not while there is 2/3+ honesty
    Legitimate verifiers with valid credentials out of cryptographic sortition process
  - Well, can – with probability 10^{-18}
    Once every few million years!
  - Each block is safely final as soon as it's certified and circulated in the network
- Tie-breakers, just in case
  i) Longest chain
  ii) Non-empty block at most recent round
  iii) Round leader with smaller credential
    The lottery ticket# in cryptographic sortition
  iv) Block with smallest hash value
    Collision-resilient hash – tie should be broken by now !!!

A Fork Scenario – The Setting

No sufficient majority, adversary can swing the votes

The network got partitioned, the adversary got lucky

- Honest votes do not constitute sufficient majority *against Algorand assumptions*!
- The adversary controls a certain amount of votes, say V_{adv}
- Honest votes + V_{adv} constitute sufficient majority

Relevant Background:

- Algorand first looks to see sufficient majority (2/3+ of votes) on a valid block* and acts on it. ***Coin-Fixed-To-0 step***
- Otherwise – if no such block, acts to vote for empty-block in the step after. ***Coin-Fixed-To-1 step***

*Valid block: block proposed by the legitimate leader of that round and contains legitimate payments

A Fork Scenario – The Act

- Make V_{adv} hold back votes in *Coin-Fixed-To-0* step
- Let Algorand proceed to *Coin-Fixed-To-1* step
- Make V_{adv} vote for empty-block
- Make V_{adv} release votes on valid block

Result: a temporary fork

- Both empty-block and the valid block are certified
- Temporary

  Branch with empty-block discarded the round after
  - Legitimate verifiers on both branches
  - The tie-breaker

Details: Section 10.1 of [CM17]

References

- [CM17] Chen, J., Micali, S., (2017), *ALGORAND*, https://arxiv.org/pdf/1607.01341.pdf
- [G+17] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N., (2017), *Algorand: Scaling Byzantine Agreements for Cryptocurrencies*, https://people.csail.mit.edu/nickolai/papers/gilad-algorand-eprint.pdf
- [M17] Micali, S., (2017), "*Algorand: A Better Distributed Ledger," with Silvio Micali*, ACM Channel on Youtube, https://www.youtube.com/watch?v=_nQE_HAGlmM&t=213s