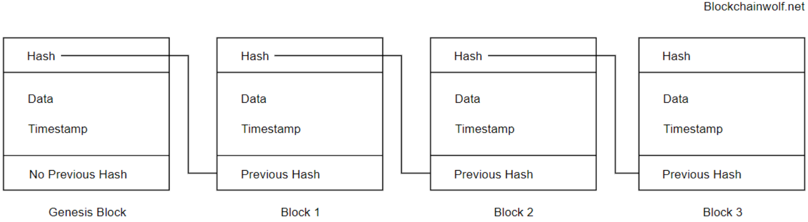
Cryptography in Algorand

Zhenfei Zhang

Algorand

June 26, 2020

Introduction



Consensus on next blocks

- Proof of Work: bitcoin, ethereum, . . .
- Proof of Stake: Algorand

Introduction

ALGORAND AGREEMENT
Super Fast and Partition Resilient Byzantine Agreement

Jing Chen Sergey Gorbunov Silvio Micali Georgios Vlachos
{jing, sergey, silvio, georgios@algorand.com}

April 25, 2018

Abstract

We present a simple Byzantine agreement protocol with leader election, that works under $> 2/3$ honest majority and does not rely on the participants having synchronized clocks. When honest messages are delivered within a bounded worst-case delay, agreement is reached in expected constant number of steps when the elected leader is malicious, and is reached after two steps when the elected leader is honest. Our protocol is resilient to arbitrary network partitions with unknown length, and recovers fast after the partition is resolved and bounded message delay is restored.

We will briefly discuss how the protocol applies to blockchains in a permissionless system. In particular, when an honest leader proposes a block of transactions, the first voting step happens in parallel with the block propagation. Effectively, after the block propagates, a certificate is generated in just one step of voting.

Scope

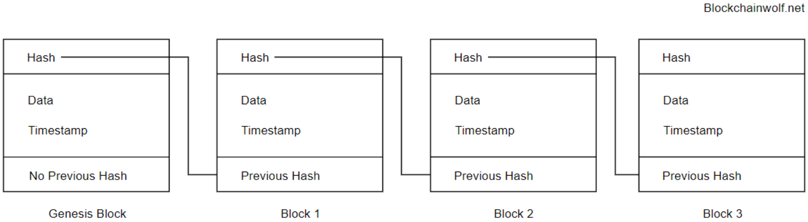
This talk

- Cryptography that makes Algorand possible:
  - ed25519 signature
  - Verifiable random function (VRF)[MRV99, GNP+15]
- Cryptography that improves efficiency/security
  - Boneh–Lynn–Shacham (BLS) signature [BLS01, BGLS03, BDN18]
  - Pixel signature [DGNW19, GW19]
- Focus on use cases, constructions
- Not on security proofs, implementations

Will not cover

- Algorand's consensus protocol
- Algorand's smart contract
- Tokenomics
- zk-proofs

What is the issue?



Consensus on next blocks

- Blockchain has > 1 million users
- Consensus is practical iff #users < 3k, via Byzantine agreement

Feasibility

Problem Statement

1. Shrink user size: 1 million $\rightarrow$ 3 thousand
2. Authentication

Feasibility

Problem Statement

2. Authentication

```
int
crypto_sign_keypair(unsigned char *pk, unsigned char *sk)
{
    return crypto_sign_ed25519_keypair(pk, sk);
}

int
crypto_sign(unsigned char *sm, unsigned long long *smlen_p,
            const unsigned char *m, unsigned long long mlen,
            const unsigned char *sk)
{
    return crypto_sign_ed25519(sm, smlen_p, m, mlen, sk);
}

int
crypto_sign_open(unsigned char *m, unsigned long long *mlen_p,
                 const unsigned char *sm, unsigned long long smlen,
                 const unsigned char *pk)
{
    return crypto_sign_ed25519_open(m, mlen_p, sm, smlen, pk);
}
```

Feasibility

Problem Statement

2. Authentication

Schnorr identification

Sender $(x, X := xG)$		Receiver (X)
$r \leftarrow_\$ \mathbb{Z}_{\lvert\mathbb{G}\rvert}$, $R = rG$		
	$\xrightarrow{R}$	
		$c \leftarrow_\$ \mathbb{Z}_{\lvert\mathbb{G}\rvert}$
	$\xleftarrow{c}$	
$z = r - xc$		
	$\xrightarrow{z}$	
		$zG \stackrel{?}{=} R - cX$

Feasibility

Problem Statement

2. Authentication

Schnorr signature (with Fiat-Shamir transformation)

- Sign(x, X, msg):
  - $r \leftarrow_{\$} \mathbb{Z}_{|\mathbb{G}|}$, $R = rG$
  - $c = hash(msg|pk|R)$
  - $z = r - xc$
  - $\sigma = \{z, c\}$
- Verify(X, msg, σ):
  - $R' = zG + cX$
  - $c \stackrel{?}{=} hash(msg|pk|R')$

Sketch security proof

- Rewind and extract x

Feasibility

Problem Statement

1. Shrink user size: 1 million $\rightarrow$ 3 thousand

中国福利彩票

Feasibility

Problem Statement

1. Shrink user size: 1 million $\rightarrow$ 3 thousand

Algorand's self-lottery

- Each user draws a random number r_i
- User is in voting committee if it wins lottery: $r_i < b$
- Use **VRF** to build lottery
- Set b so that 3k users are expected to win for each round
- Invoke BA to achieve consensus among committee members

Feasibility

Problem Statement

1. Shrink user size: 1 million $\rightarrow$ 3 thousand

```
int
crypto_vrf_prove(unsigned char *proof, const unsigned char *skpk,
                 const unsigned char *m, const unsigned long long mlen)
{
    return crypto_vrf_ietfdraft03_prove(proof, skpk, m, mlen);
}

int
crypto_vrf_verify(unsigned char *output, const unsigned char *pk,
                  const unsigned char *proof, const unsigned char *m,
                  const unsigned long long mlen)
{
    return crypto_vrf_ietfdraft03_verify(output, pk, proof, m, mlen);
}
```

Feasibility

Problem Statement

1. Shrink user size: 1 million → 3 thousand

Schnorr identification

Sender $(x, X := xG)$		Receiver (X)
$r \leftarrow_\$ \mathbb{Z}_{\lvert\mathbb{G}\rvert},\ R = rG$		
	$\xrightarrow{R}$	
		$c \leftarrow_\$ \mathbb{Z}_{\lvert\mathbb{G}\rvert}$
	$\xleftarrow{c}$	
$z = r - xc$		
	$\xrightarrow{z}$	
		$zG \stackrel{?}{=} R - cX$

Feasibility

Problem Statement

1. Shrink user size: 1 million $\rightarrow$ 3 thousand

"Double" Schnorr identification

Sender $(x, X := xG)$		Receiver (X)
$r \leftarrow_{\$} \mathbb{Z}_{\lvert\mathbb{G}\rvert},\ R_1 = rG$		
$H = hash(X),\ R_2 = rH,\ Y = xH$		
	$\xrightarrow{R_1, R_2, Y}$	
		$c \leftarrow_{\$} \mathbb{Z}_{\lvert\mathbb{G}\rvert}$
	$\xleftarrow{c}$	
$z = r - xc$		
	$\xrightarrow{z}$	
		$H = hash(X)$
		$zG \stackrel{?}{=} R_1 - cX$
		$zH \stackrel{?}{=} R_2 - cY$

Feasibility

Problem Statement

1. Shrink user size: 1 million $\rightarrow$ 3 thousand

ECVRF[GNP+15]

- Prove(x, X, msg):
  - $r \leftarrow_{\$} \mathbb{Z}_{|\mathbb{G}|}$, $R_1 = rG$
  - $H = hash_1(msg|pk)$, $R_2 = rH$, $Y = xH$
  - $c = hash_2(msg|pk|R_1|R_2)$
  - $z = r - xc$
  - $\sigma = \{z, c\}$, $\pi = Y$
- Verify(X, msg, σ, π):
  - $R'_1 = zG + cX$
  - $H = hash_1(msg|pk)$
  - $R'_2 = zH + cY$
  - $c \stackrel{?}{=} hash_2(msg|pk|R'_1|R'_2)$

Feasibility

ECVRF[GNP+15]

- Prove(x, X, msg):
  - $r \leftarrow_\$ \mathbb{Z}_{|\mathbb{G}|}$, $R_1 = rG$
  - $H = hash_1(msg|pk)$, $R_2 = rH$, $Y = xH$
  - $c = hash_2(msg|pk|R_1|R_2)$
  - $z = r - xc$
  - $\sigma = \{z, c\}$, $\pi = Y$
- Verify(X, msg, σ, π):
  - $R_1' = zG + cX$
  - $H = hash_1(msg|pk)$
  - $R_2' = zH + cY$
  - $c \stackrel{?}{=} hash_2(msg|pk|R_1'|R_2')$

Security requirements

- Correctness
- Unforgability follows Schnorr signature
- Uniqueness: fix X, msg, there is only one Y
- Pseudorandomness: Y is IND from random

Scalability

Problem Statement

- New block generated every 4.5 sec $\rightarrow$ 250 ms budget for cryptography
- Verifies **5k** transactions per block
- Verifies **3k** votes per block

Scalability

Problem Statement

- New block generated every 4.5 sec $\rightarrow$ 250 *ms* budget for cryptography
- Verifies 5k transactions per block

$$\text{Bilinear pairing}: \quad \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_t, \quad e(aG_1, bG_2) = e(G_1, G_2)^{ab}$$

BLS signature[BLS01, BGLS03, BDN18]

- Sign(x, msg):
  - $H = hash(msg)$
  - $S = xH$
- Verify$(X := xG_1, msg, S)$:
  - $H = hash(msg)$
  - $e(G_1, S) \stackrel{?}{=} e(X, H)$

Scalability

Problem Statement

- New block generated every 4.5 sec $\rightarrow$ 250 *ms* budget for cryptography
- Verifies 5k transactions per block

Bilinear pairing : $\mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_t, \quad e(aG_1, bG_2) = e(G_1, G_2)^{ab}$

BLS signature[BLS01, BGLS03, BDN18]

- Sign(x, msg):
  - $H = hash(msg)$
  - $S = xH$
- Aggregate$(S_1, \ldots, S_k)$:
  - $S = \sum_{i=1}^{k} S_i$
- Verify$(X := xG_1, msg, S)$:
  - $H = hash(msg)$
  - $e(G_1, S) \stackrel{?}{=} e(X, H)$
- AggreVerify$(\{X_i, msg_i\}_{i=1}^{k}, S)$:
  - $H_i = hash(msg_i)$
  - $e(G_1, S) \stackrel{?}{=} \prod_{i=1}^{k} e(X_i, H_i)$

Scalability

Problem Statement

- New block generated every 4.5 sec $\rightarrow$ 250 *ms* budget for cryptography
- Verifies 5k transactions per block

Same message rogue key attack[BDN18]

- Attacker sees $\{msg, X\}$ for a user who knows x
- Attacker claims his public key is $Y - X$ (he does not know y/x)
- Attacker forges an agg. signature $S_y := yH$ for user and attacker
- Can be verified by AggreVerify$(X, Y - X, msg, S_y)$

Scalability

Problem Statement

- New block generated every 4.5 sec $\rightarrow$ 250 *ms* budget for cryptography
- Verifies 5k transactions per block

Same message rogue key attack[BDN18]

- Attacker sees $\{msg, X\}$ for a user who knows x
- Attacker claims his public key is $Y - X$ (he does not know y/x)
- Attacker forges an agg. signature $S_y := yH$ for user and attacker
- Can be verified by $\text{AggreVerify}(X, Y - X, msg, S_y)$

Solutions

- Proof of possession
- $H_i = hash(msg|PK_i)$
- BLS+

Scalability

Problem Statement

- New block generated every 4.5 sec $\rightarrow$ 250 *ms* budget for cryptography
- Verifies 5k transactions per block

BLS+ signature[BDN18]

- Sign(x, msg):
  - $H = hash(msg)$
  - $S = xH$
- Aggregate$(\{X_i, S_i\}_{i=1}^k)$:
  - $r_1 \ldots, r_k = hash_1(\{X_i\}_{i=1}^k)$
  - $S = \sum_{i=1}^k r_i S_i$
- Verify$(X := xG_1, msg, S)$:
  - $H = hash(msg)$
  - $e(G_1, S) \stackrel{?}{=} e(X, H)$
- AggreVerify$(\{X_i\}_{i=1}^k, msg, S)$:
  - $r_1 \ldots, r_k = hash_1(\{X_i\}_{i=1}^k)$
  - $H = hash_2(msg)$
  - $e(G_1, S) \stackrel{?}{=} \prod_{i=1}^k e(r_i X_i, H)$

Scalability

Problem Statement

- New block generated every 4.5 sec
- Verifies 5k transactions per block
- Verifies 1k votes per block

Forward security

- Attacker corrupts voters **after** they have voted
- Ask them to vote for another block – creates a fork

Solution: Pixel signature [DGNW19, GW19]

- Same message aggregatable
- Forward secure

Pixel Signature

High level description (HIBE)

- A master public key pk; t secret keys for t levels
- for $i = 1, \ldots, t$
  - use sk_i for level i to sign; (HIBE encryption)
  - use sk_i to generate sk_{i+1}; (HIBE delegation)
  - throw away sk_i
- support t time slots naively
- support 2^t time slots using a tree structure

Pixel Signature

Toy example with t slots

- PP: $G \in \mathbb{G}_1; H, H_1, \dots, H_t \in \mathbb{G}_2$
- pk: xG, msk: xH (Note $e(pk, H) = e(G, msk)$)
- SKUpdate(sk_i)$\mapsto$ sk_{i+1}:
  - $sk_{i+1}[0] = sk_i[0] + r_{i+1}G$
  - $sk_{i+1}[1] = sk_i[1] + r_{i+1}\sum_{j=1}^{i+1} H_j$
  - $sk_{i+1}[2][j] = sk_{i+1}[2][j+1] + r_{i+1}H_{j+i+2}$

	randomize $\mathbb{G}_1$ gen.	randomize x w. new gen.	randomize $\mathbb{G}_2$ gen.
sk_1	r_1G	$xH + r_1H_1$	$r_1\langle H_2, \dots, H_t\rangle$
sk_2	$(r_1+r_2)G$	$xH + r_1H_1 + r_2(H_1+H_2)$	$(r_1+r_2)\langle H_3, \dots, H_t\rangle$
sk_3	$(r_1+r_2+r_3)G$	$xH + r_1H_1 + r_2(H_1+H_2)$ $+r_3(H_1+H_2+H_3)$	$(r_1+r_2+r_3)\langle H_4, \dots, H_t\rangle$

Pixel Signature

Toy example with t slots, continued

- PP: $G \in \mathbb{G}_1; H, H_1, \ldots, H_t \in \mathbb{G}_2$
- pk: xG, msk: xH
- $sk_1 = \{r_1 G, xH + r_1 H_1, r_1 \langle H_2, \ldots, H_{t-1}, H_t \rangle\}$
- Sign(msg, sk_1)$\mapsto \sigma$:
  - $a \leftarrow_\$ \mathbb{Z}_{|\mathbb{G}_1|}$
  - $h = hash(msg)$
  - $X = aG + r_1 G$
  - $Y = (xH + r_1 H_1) + h(r_1 H_t) + a(H_1 + hH_t)$
  - $\sigma = \{X, Y\}$
- Verify(msg, pk, $\sigma := \{X, Y\}$):

$$e(xG, H) \cdot e(X, H_1 + hH_t) \stackrel{?}{=} e(G, Y)$$

Pixel Signature

Correctness:

- $e(xG, H) \cdot e(X, H_1 + hH_t) \stackrel{?}{=} e(G, Y)$
- $X = aG + r_1 G$
- $Y = (xH + r_1 H_1) + h(r_1 H_t) + a(H_1 + hH_t)$

$$\begin{aligned} & e(xG, H) \cdot e(X, H_1 + hH_t) \\ & \qquad = e(G, H)^x \cdot e(X, H_1) \cdot e(X, H_t)^h \\ & \qquad = e(G, H)^x \cdot e(G, H_1)^{a+r_1} \cdot e(G, H_t)^{h(a+r_1)} \end{aligned}$$

$$\begin{aligned} e(G, Y) &= e(G, xH + r_1 H_1) \cdot e(G, hr_1 H_t) \cdot e(G, a(H_1 + hH_t)) \\ &= e(G, xH) \cdot e(G, (r_1 + a)H_1) \cdot e(G, (hr_1 + ah)H_t) \\ &= e(G, H)^x \cdot e(G, H_1)^{a+r_1} \cdot e(G, H_t)^{h(a+r_1)} \end{aligned}$$

Pixel Signature

Forward security

- pk: xG, msk: xH
- $sk_1 = \{r_1 G, xH + r_1 H_1, r_1 \langle H_2, \ldots, H_t \rangle\}$
- $sk_2 = \{(r_1 + r_2)G, xH + r_1 H_1 + r_2(H_1 + H_2), (r_1 + r_2) \langle H_3, \ldots, H_t \rangle\}$
- Cannot find *msk* from sk_1
- Cannot find *msk* or sk_1 from sk_2

Pixel Signature

Same message aggregation

- User 1
  - $X_1 = a_1 G + r_{1,1} G$
  - $Y_1 = (x_1 H + r_{1,1} H_1) + h(r_{1,1} H_t) + a_1(H_1 + hH_t)$
- User 2
  - $X_2 = a_2 G + r_{2,1} G$
  - $Y_2 = (x_2 H + r_{2,1} H_1) + h(r_{2,1} H_t) + a_2(H_1 + hH_t)$
- Aggregated sig: $X_1 + X_2, Y_1 + Y_2$
- Correctness:

$$\begin{aligned}
&e(x_1 G + x_2 G, H) \cdot e(X_1 + X_2, H_1 + hH_t) \\
&\quad = e(x_1 G, H) \cdot e(x_2 G, H) \cdot e(X_1, H_1 + hH_t) \cdot e(X_2, H_1 + hH_t) \\
&\quad = (e(x_1 G, H) \cdot e(X_1, H_1 + hH_t)) \cdot (e(x_2 G, H) \cdot e(X_2, H_1 + hH_t)) \\
&\quad = e(G, Y_1) \cdot e(G, Y_2) \\
&\quad = e(G, Y_1 + Y_2)
\end{aligned}$$

Challenges in a post quantum world

- Signatures – reduce signature size (Falcon $\approx$ 1kB)
- VRF – WIP
- (None-interactive) aggregatable signature – no known solution
- Forward secure signatures – lattice based HIBE does not scale well

Thanks!

- This talk: https://zhenfeizhang.github.io/material/talks/
- ECVRF reference implementation: https://github.com/algorand/libsodium/tree/draft-irtf-cfrg-vrf-03
- BLS reference implementation: https://github.com/algorand/bls_sigs_ref
- Pixel reference implementation: https://github.com/algorand/Pixel

Dan Boneh, Manu Drijvers, and Gregory Neven.
Compact multi-signatures for smaller blockchains.
In *ASIACRYPT 2018*, 2018.

Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham.
Aggregate and verifiably encrypted signatures from bilinear maps.
In *EUROCRYPT 2003*, 2003.

Dan Boneh, Ben Lynn, and Hovav Shacham.
Short signatures from the weil pairing.
In *ASIACRYPT 2001*, 2001.

Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee.
Pixel: Multi-signatures for consensus.
IACR Cryptol. ePrint Arch., 2019:514, 2019.

Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv.
NSEC5: provably preventing DNSSEC zone enumeration.
In *NDSS 2015*, 2015.

Sergey Gorbunov and Hoeteck Wee.
Digital signatures for consensus.
IACR Cryptol. ePrint Arch., 2019:269, 2019.

Silvio Micali, Michael O. Rabin, and Salil P. Vadhan.
Verifiable random functions.
In *FOCS '99*, 1999.