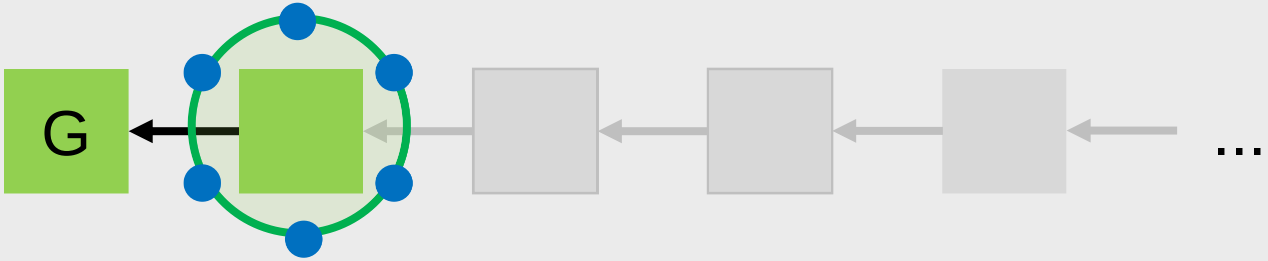
Hard Problems in Blockchains

Valeria (Lera) Nikolaenko
for the 13th BIU Winter School on cryptography

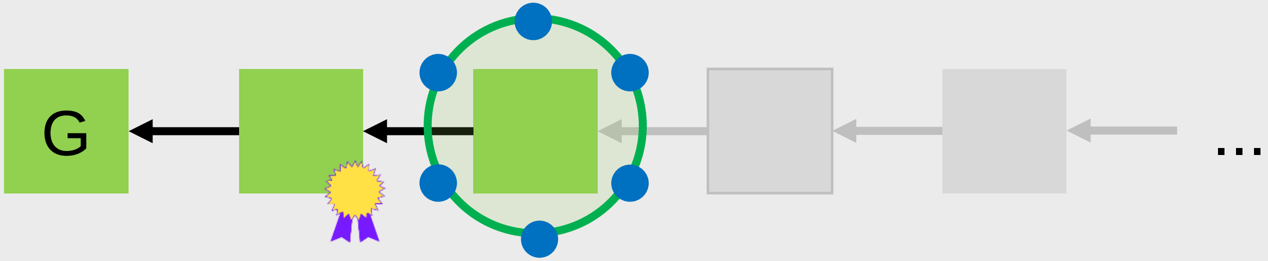
1: Long-Range Attacks on PoS
2: Proposer Election in PoS
3: Post-quantum blockchains

1: Long-Range Attacks on PoS
2: Proposer Election in PoS
3: Post-quantum blockchains

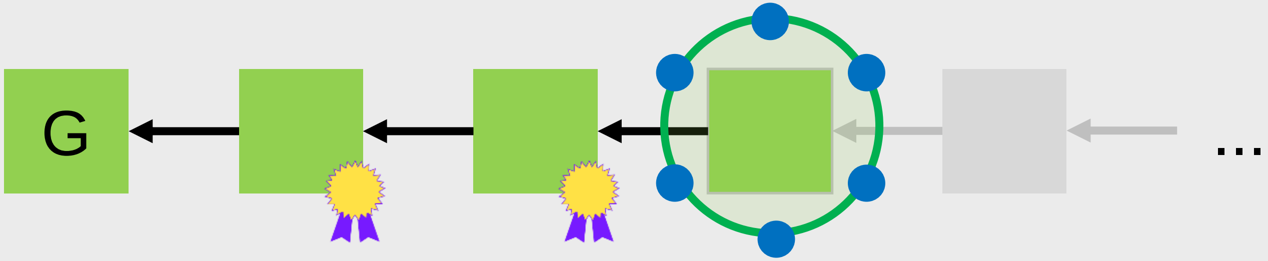
Proof-of-Stake blockchains



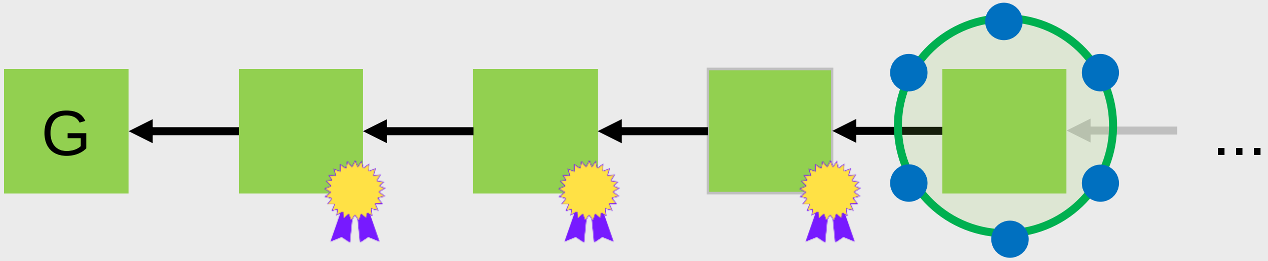
Proof-of-Stake blockchains



Proof-of-Stake blockchains



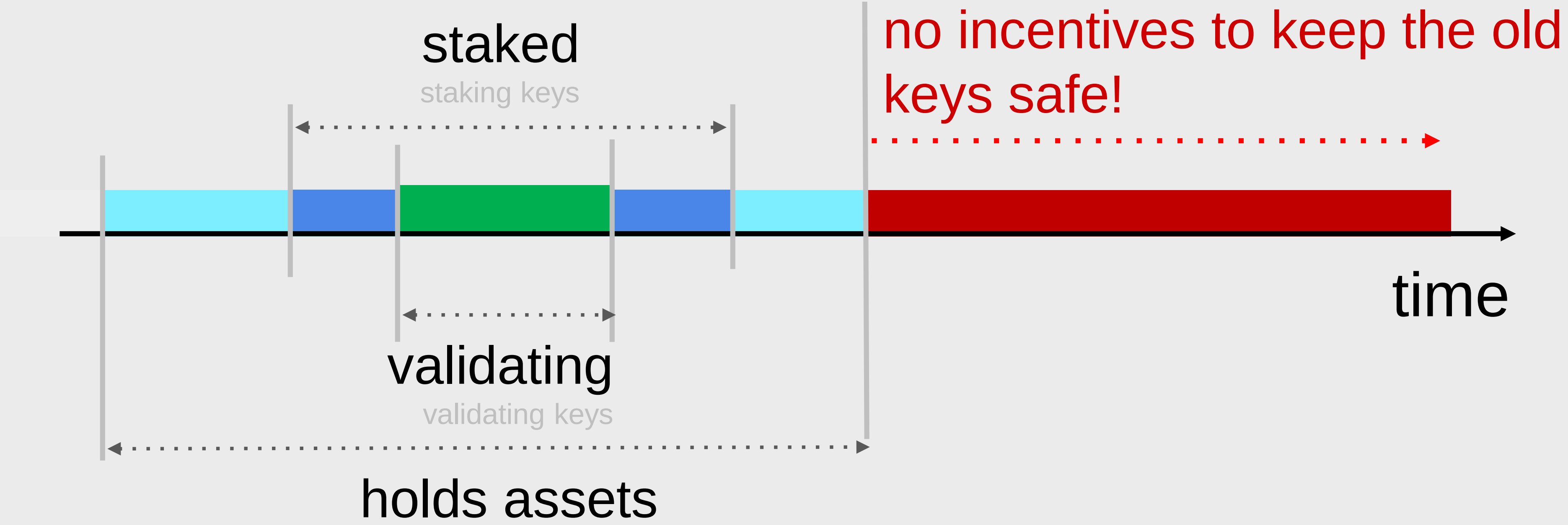
Proof-of-Stake blockchains



Validators "notarize" blocks

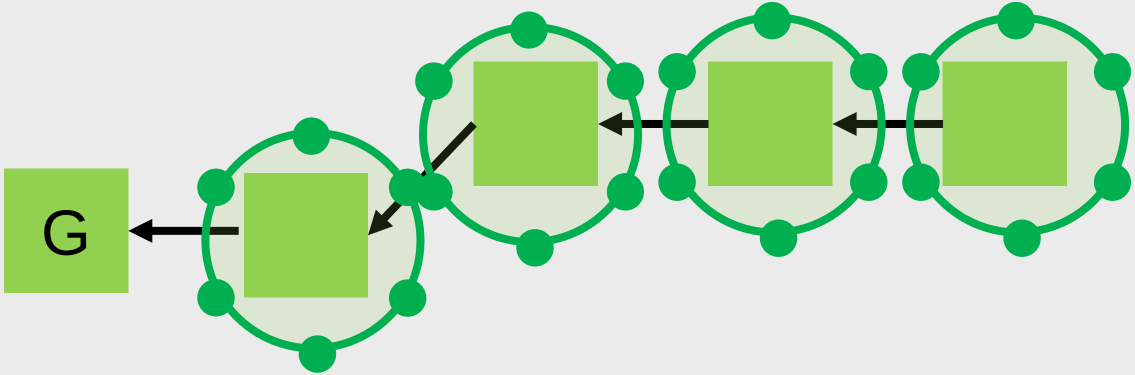
Active validators stay honest due to incentives

Validator with no incentives can leak old keys

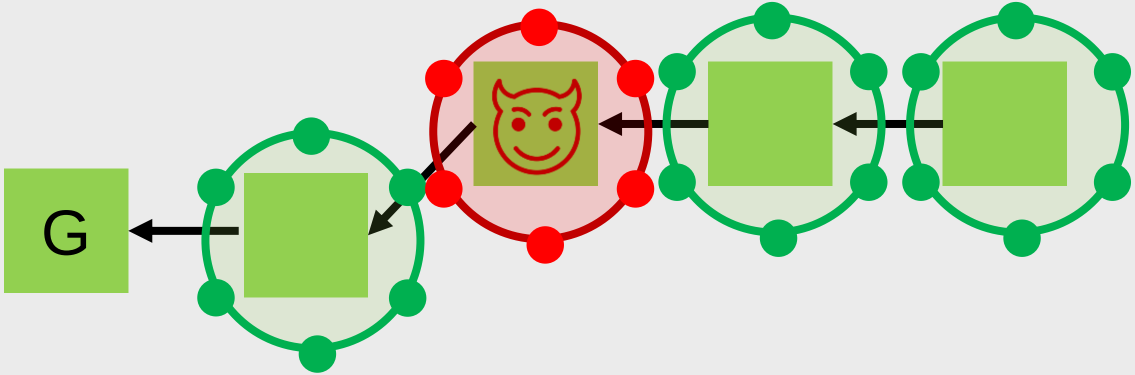


If old validators become corrupt
safety is broken

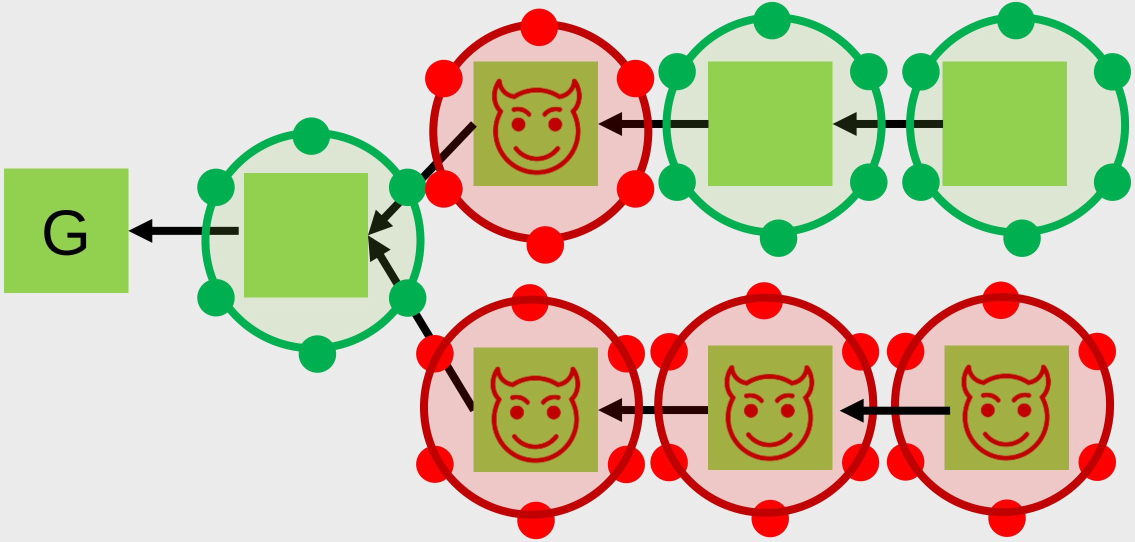
Corrupt validators may fork history



Corrupt validators may fork history



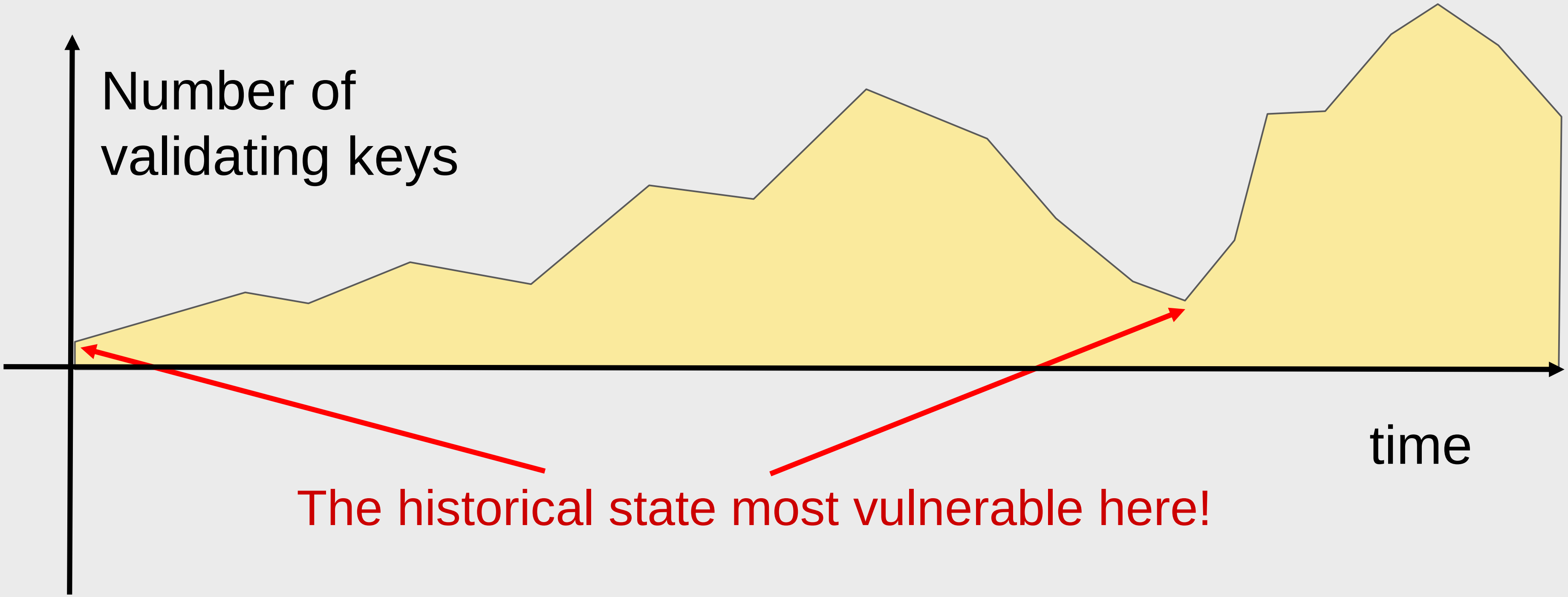
Corrupt validators may fork history



Users can not differentiate!

unless they've been following all consensus rounds actively

Low number of keys don’t protect history well



Mitigations for long-range attacks

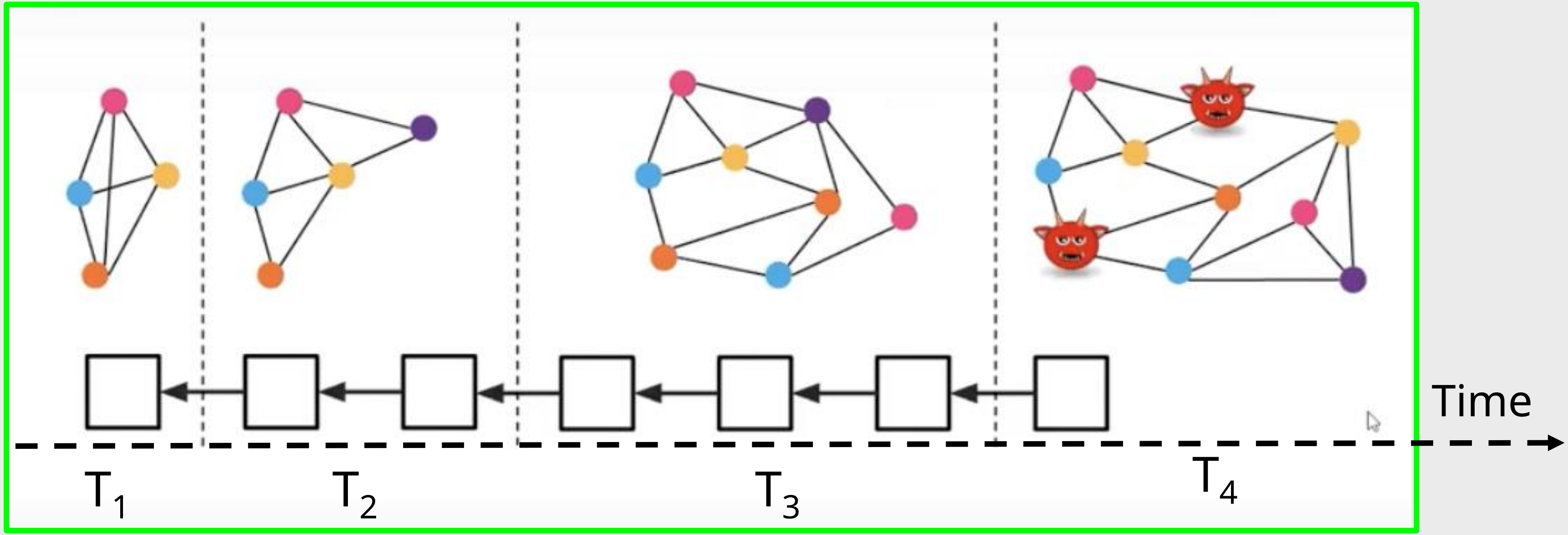
1. Checkpointing
2. Key-evolving cryptography
3. Keep everybody online
4. Winkle (user-based consensus)

#1 Checkpointing

- Centralized checkpointing (i.e. hardcode the checkpoints into the github codebase)
    - Checkpoint = hash of a block (very small)
    - When synching check that the checkpoint matches the hardcoded one
- Checkpoint to a PoW chain

Problem: centralized!

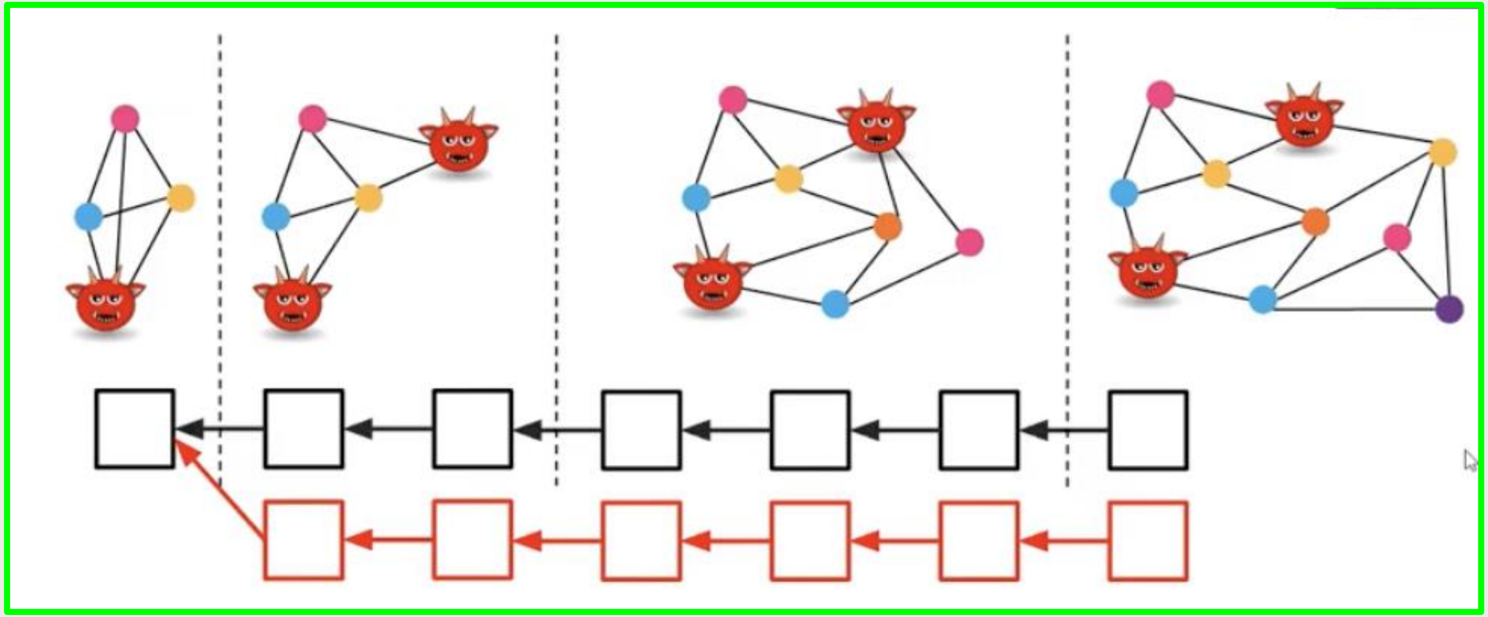
Easier to attack if validators are not rotating their keys



Attacking 2 at time T_4 equivalent to attacking 2 at time T_2!

Gregory Neven's presentation at USENIX'20
www.usenix.org/conference/usenixsecurity20/presentation/drijvers

Easier to attack if validators are not rotating their keys



2-out-of-5 is enough to attack!

2-out-of-10 is not enough to attack

Gregory Neven's presentation at USENIX'20
www.usenix.org/conference/usenixsecurity20/presentation/drijvers

#2 Key-Evolving cryptography

- Rotate validator keys frequently: **(pk, sk) → (pk', sk')**
- Time-evolving secret-key (public key stays the same!) [DGNW20]:
  $(pk, sk_1) \rightarrow (pk, sk_2) \rightarrow (pk, sk_3) \rightarrow (pk, sk_4) \rightarrow ...$
- Assume honest validators forget old keys.

Does not solve our problem, but good practice anyway!

Problem: erasing the old secret keys is incentive incompatible!

#3: Keep everybody online

When all the nodes are online and monitoring the blockchain closely, it is very hard to make them believe a deep fork.

Casper the Friendly Finality Gadget

Vitalik Buterin and **Virgil Griffith**
Ethereum Foundation

. . .

In simple terms, long-range attacks are prevented by a fork choice rule to never revert a finalized block, as well as an expectation that each client will "log on" and gain a complete up-to-date view of the chain at some regular frequency (e.g., once per 1–2 months). A "long range revision" fork that finalizes blocks older than that will

Problem is: clients/validators can be sleepy.

#4 Winkle :

make users “vote” inside their transactions on the current state of the blockchain

Winkle: Foiling Long-Range Attacks in Proof-of-Stake Systems

Sarah Azouvi
University College London,
Protocol Labs

George Danezis
University College London,
Facebook Novi

Valeria Nikolaenko
Facebook Novi

ABSTRACT

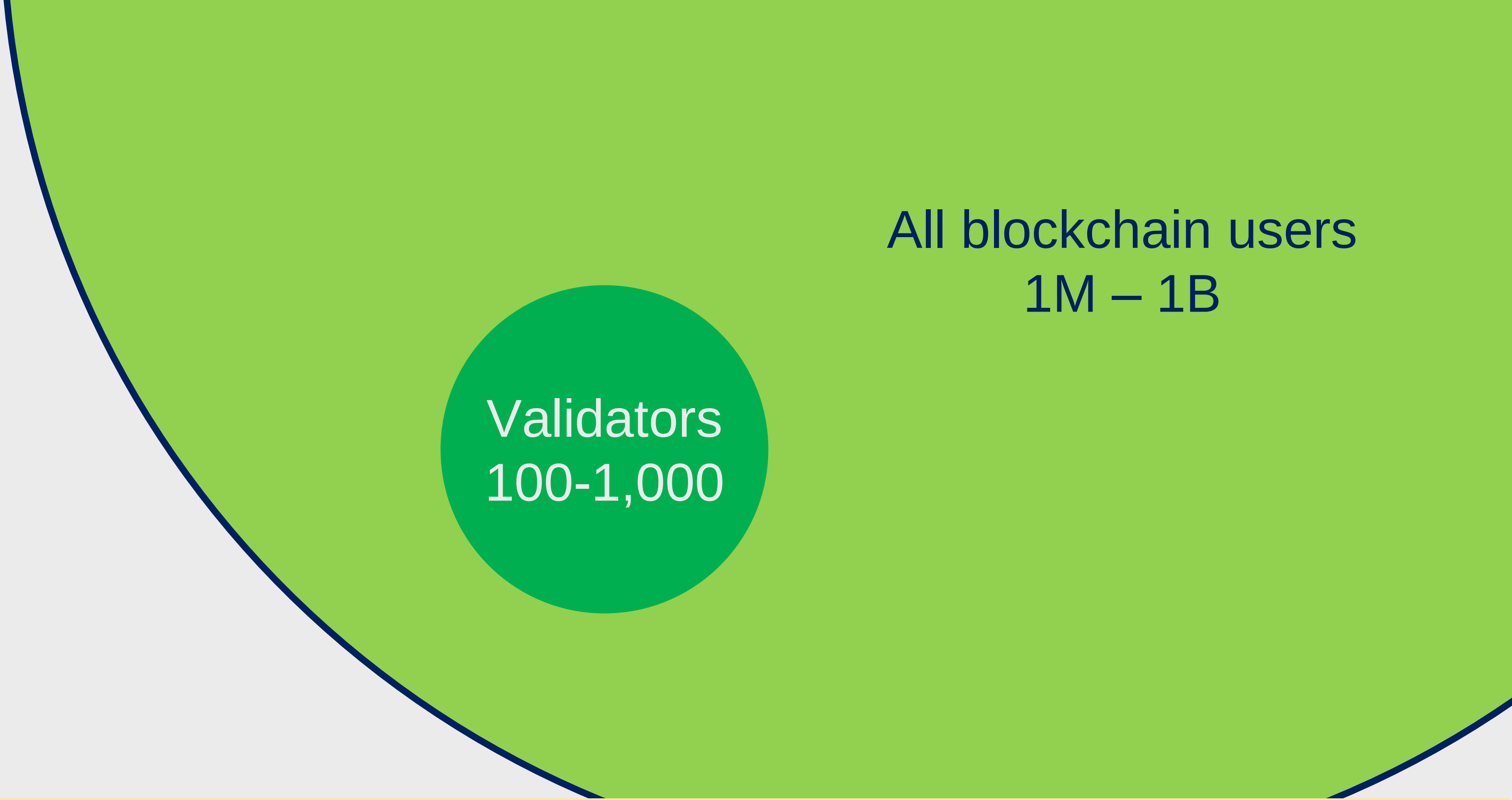
Winkle protects any validator-based byzantine fault tolerant consensus mechanisms, such as those used in modern Proof-of-Stake blockchains, against long-range attacks where old validators' signature keys get compromised. Winkle is a decentralized secondary layer of client-based validation, where a client includes a single additional field into a transaction that they sign: a hash of the previously sequenced block. The block that gets a threshold of signatures (confirmations) weighted by clients' coins is called a "confirmed" checkpoint. We show that under plausible and flexible security assumptions about clients the confirmed checkpoints can not be equivocated. We discuss how client key rotation increases security, how to accommodate for coins' minting and how delegation allows for faster checkpoints. We evaluate checkpoint latency experimentally using Bitcoin and Ethereum transaction graphs, with and without delegation of stake.

Validator key rotations help alleviate the problem, assuming secure destruction of older keys. However, validators might have auxiliary incentives to sell their old keys to an adversary, especially when real-world identities of validators are unknown in a permissionless system and reputation is not at risk. When dishonest behaviour of a validator becomes rational, real-world security of the whole system is at great risk. We notice that corrupting a significant number of coin holders, even after they have no more stake in the system, is far more challenging as they are much more numerous than validators (we justify this assumption in Section 4). This observation brings us to introducing Winkle — a novel mechanism that leverages votes from clients creating a decentralized secondary layer of client-based validation to confirm checkpoints (snapshots of the blockchain) and to prevent long-range attacks on proof-of-stake protocols. The voting mechanism is very simple: each client augments their transaction with a single additional field — a hash of a previously sequenced block. Once this transaction gets signed

Winkle: users vote on blocks when transacting

- New transaction format:
  Tx = [sender, receiver, amount , **LAST_BLOCK**]$_\sigma$
- The block is checkpointed when 50% of all coins vote on it.
- Checkpoint can’t be reverted even under Long-Range-Attack.

- To attack the adversary needs to obtain:
  validators’ keys AND users’ keys



To attack the adversary needs to obtain: validators' keys AND users' keys

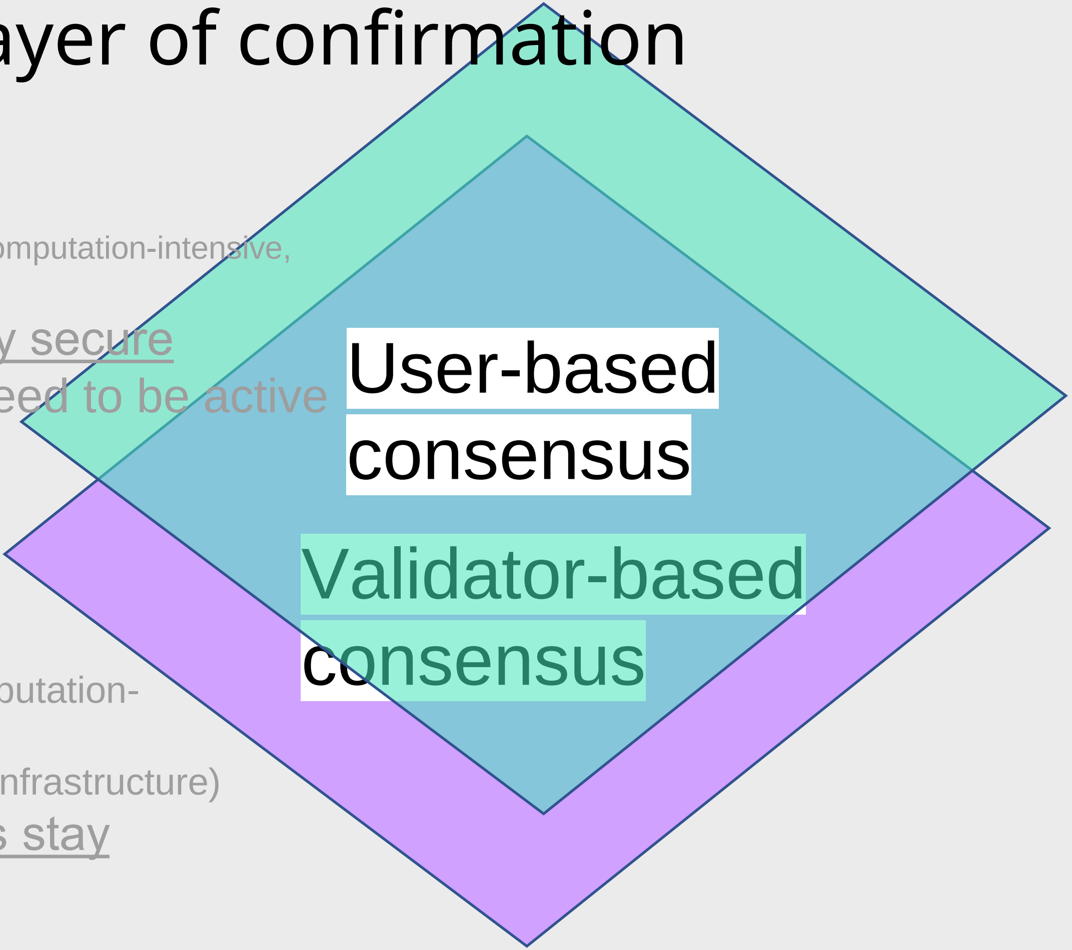
Winkle: second layer of confirmation

Consensus by users:

- simple (NON-INTERACTIVE, not computation-intensive, no additional infrastructure)
- safe while users' keys stay secure
- large threshold of users need to be active

Consensus by validators:

- < 1/3 byzantine
- complicated (interactive, computation-intensive, requires dedicated infrastructure)
- safe while validators' keys stay secure



Mitigations for long-range attacks

1. Checkpointing
2. Key-evolving cryptography
3. Keep everybody online
4. Winkle (user-based consensus)

    philosophically, a good idea to make all users (not just the validators) work on maintaining the security of blockchain

1: Long-Range Attacks on PoS
2: Proposer Election in PoS
3: Post-quantum blockchains

Proposer election goals

- A proposer is elected per time-slot to propose a block.
- A proposer gets rewarded for proposing a good block.
- Election properties
    - **Uniqueness** of elected proposer
    - **Unpredictability$_T$** : T is the time between proposer getting publicly known and proposer announcing a block
        - $T > 0$: “public election”
        - $T = 0$: “secret election” - the proposer announces itself when published the block
        - => fair (each leader is elected with equal probability)
    - **Unbiasability** (nobody should be able to influence the proposer election in its favor)
      = Unpredictability under active attacks

#1 Proposer election : Round-Robin

Round-robin proposer election: proposers are chosen one after the other in a lexicographical order.

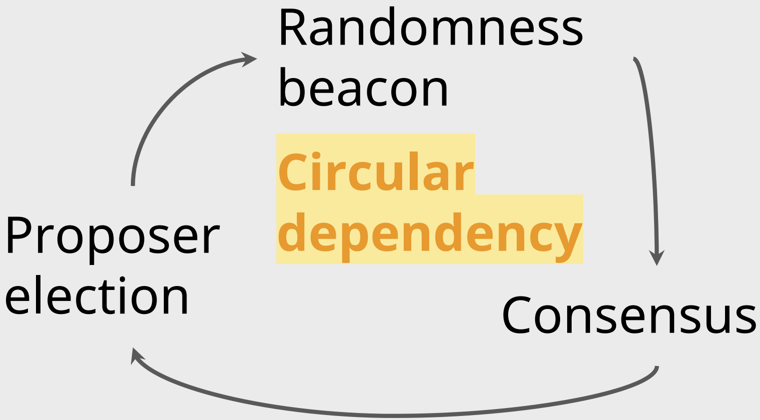
Unique

Predictable = public: the proposer for the slot is known well in advance: proposers can be DDoSed

Biasable

#2 Proposer election : Randomness Beacon

Randomness beacon: a distributed protocol that outputs (pseudo)-random values at regular time intervals



Circular dependency is broken by induction:

- fix proposer schedule of the current epoch,
- build randomness beacon to randomize proposer schedule of the next epoch.

Unique; Unbiasable;

Predictable: schedule known 1 epoch in advance

#2 Proposer election : Randomness Beacon

Protocol	Network Model	Adaptive Adversary	Liveness	Unpredictability	Bias-Resistance	Fault-tolerance	Communication Complexity	Computation Complexity	Verification Complexity	Cryptographic Primitive	No Trusted Dealer or DKG required
ALBATROSS [18]	syn.	✗	✓	✓	✓	1/2	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	PVSS	✓
Algorand [39]	semi-syn.	✗	✓	✓‡	✗	1/3°	$\mathcal{O}(cn)$	$\mathcal{O}(c)$	$\mathcal{O}(1)$	VRF	✓
BRandPiper [8]	syn.	✓	✓	✓	✓	1/2	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS	✓
Cachin et. al [16]	asyn.	✗	✓	✓	✓	1/3	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Uniq. thr-sig.	✗
Caucus [2]	syn.	✗	✓	✓‡	✗	1/2	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Hash func.	✓
Continuous VDF [34]	asyn.	✗	✗†	✓	✓	1/2	$\mathcal{O}(1)$	VDF	$\mathcal{O}(1)$	VDF	✓
DFINITY [45]	semi-syn.	✗	✓	✓	✓	1/3	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	BLS thr-sig.	✗
Drand [31]	syn.	✗	✓	✓	✓	1/2	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Uniq. thr-sig.	✗
GLOW [36]	syn.	✗	✓	✓	✓	1/3	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	DVRF	✗
GRandPiper [8]	syn.	✗	✓	✓‡	✓	1/2	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS	✓
HERB [20]	syn.	✗	✓	✓	✓	1/3	$\mathcal{O}(n^2)^*$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PHE	✗
HydRand [66]	syn.	✗	✓	✓‡	✓	1/3	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PVSS	✓
Nguyen-Van et. al [55]	syn	✗	✗	✓	✗	1/2	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	PHE, VRF	✓
Ouroboros [47]	syn.	✗	✓	✓	✓	1/2	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	✓
Ouroboros Praos [27]	semi-syn.	✓	✓	✓‡	✗	1/2	$\mathcal{O}(n)^*$	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	VRF	✓
Proof-of-Delay [14]	syn.	✗	✓	✓	✓	1/2	$\mathcal{O}(n)$	very high	$\mathcal{O}(\log \Delta)^\circ$	Hash func.	✓
Proof-of-Work [53]	syn.	✗	✓	✓‡	✗	1/2	$\mathcal{O}(n)$	very high	$\mathcal{O}(1)$	Hash func.	✓
RandChain [71]	syn.	✗	✓	✓	✓	1/3	$\mathcal{O}(cn)$	$\mathcal{O}(cn)$	$\mathcal{O}(n)$	VRF	✓
RANDCHAIN [44]	syn.	✓	✓	✓	✓	1/3	$\mathcal{O}(n)$	VDF	$\mathcal{O}(1)$	VDF	✓
RANDAO [59]	asyn.	✗	✓	✗	✗	1/2	$\mathcal{O}(n)$	VDF	$\mathcal{O}(1)$	VDF	✓
RandHerd [69]	syn.	✗	✓	✓	✓	1/3	$\mathcal{O}(c^2 \log n)$	$\mathcal{O}(c^2 \log n)$	$\mathcal{O}(1)$	PVSS, CoSi	✗
RandHound [69]	syn.	✗	✓	✓	✓	1/3	$\mathcal{O}(c^2 n)$	$\mathcal{O}(c^2 n)$	$\mathcal{O}(c^2 n)$	PVSS	✓
RandRunner [65]	syn.	✓	✓	✓‡	✓	1/2	$\mathcal{O}(n^2)$	VDF	$\mathcal{O}(1)$	VDF	✓
RandShare [69]	asyn.	✓	✗⊙	✓	✓	1/3	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	VSS	✓
Rand Extractor [21,13]	asyn.±	✓	✓‖	✓	✓	1/2	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Hash func.	✓
SCRAPE [17]	syn.	✗	✓	✓	✓	1/2	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS	✓
SPURT [25]	semi-syn	✗	✓	✓	✓	1/3	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PVSS, Pairing	✓
Unicorn [49]	asyn.	✗	✓†	✓	✓	1/2	$\mathcal{O}(1)$	high	$\mathcal{O}(1)$	Sloth	✓

SoK: Decentralized Randomness Beacon Protocols (2022) by M. Raikwar and D. Gligoroski

Practical projects:

drand.love
github.com/drand/drand

blog.chain.link/vrf-v2-mainnet-launch

#2 Simplest Randomness Beacon using VRFs and VDFs

Each node i pre-registers VRF public key: pk_i

During an epoch:

- node i submits v_i = VRF_Eval(sk_i, epoch_number)

At the end of the epoch:

- beacon = VDF_Eval($v_1 \oplus v_2 \oplus \dots \oplus v_n$)

Contributes deterministic verifiable randomness:

- **can't compute VRF in two possible ways**
- **output is pseudorandom**

Long computation (longer than one epoch), fast to verify.

#2 Simplest Randomness Beacon using VRFs and VDFs

Each node i pre-registers VRF public key: pk_i

During an epoch:

- node i submits v_i = VRF_Eval(sk_i, epoch_number)

At the end of the epoch:

- beacon = VDF_Eval($v_1 \oplus v_2 \oplus \dots \oplus v_n$)

Unique; Unbiasable;

Predictable: schedule of leader is known 1 epoch in advance

#3 Proposer election : SSLE

SSLE: Single Secret Leader Election

1. Each validator publishes a commitment to a secret value.
2. Next, validators take turn shuffling and rerandomizing the list of commitments.
3. The random beacon is used to do the final open shuffle, and the final list determines the sequence of proposers for the next epoch.
4. Only the proposer knows its position in the list.

Unique
Unbiasable
Unpredictable

Expensive!

- "Single Secret Leader Election" (2020) by D.Boneh,S.Eskandarian,L.Hanzlik,N.Greco
- Ethereum's SSLE: Whisk

Proposer election approaches

		Unique	Unbiasable	Unpredictable
Proof-of-Work		NO	NO	NO
Proof-of-Stake	Round-robin	YES	NO	NO
	Randomness-beacon	YES	YES	NO
	SSLE	YES	YES	YES

1: Long-Range Attacks on PoS

2: Proposer Election in PoS

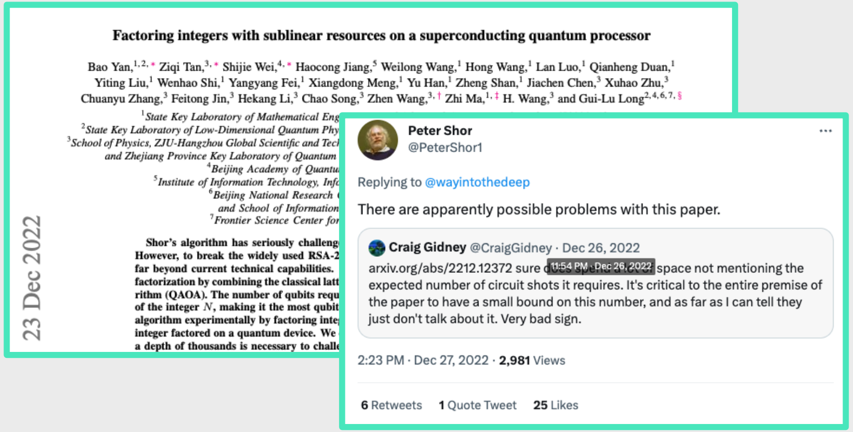
3: Post-quantum blockchains

Progress in quantum computing

* 1998 - 3 qubits
* 2000 - 7 qubits
* 2005 - 8 qubits
* 2006 - 12 qubits
* 2011 - 14 qubits
* 2017 - 50 qubits (IBM)
* 2018 - 72 qubits (Google)
* 2019 - 27 qbits IBM Falcon
* 2020 - 65 qbits IBM Hummingbird
* 2021 - 127 qbits IBM Eagle
* 2022 - 433 qbits IBM Osprey

...

- 6,146 logical qubits to break RSA-3072 (Häner-Roetteler-Svore 2017)
- 2,330 logical qubits to break discrete log over NIST P-256 curve (Roetteler-Naehrig-Svore-Lauter 2017)


Factoring integers with sublinear resources on a superconducting quantum processor

Bao Yan,[1,2,*] Ziqi Tan,[3,*] Shijie Wei,[4,*] Haocong Jiang,[5] Weilong Wang,[1] Hong Wang,[1] Lan Luo,[1] Qianheng Duan,[1] Yiting Liu,[1] Wenhao Shi,[1] Yangyang Fei,[1] Xiangdong Meng,[1] Yu Han,[1] Zheng Shan,[1] Jiachen Chen,[3] Xuhao Zhu,[3] Chuanyu Zhang,[3] Feitong Jin,[3] Hekang Li,[3] Chao Song,[3] Zhen Wang,[3,†] Zhi Ma,[1,‡] H. Wang,[3] and Gui-Lu Long[2,4,6,7,§]

[1]State Key Laboratory of Mathematical Eng
[2]State Key Laboratory of Low-Dimensional Quantum Phy
[3]School of Physics, ZJU-Hangzhou Global Scientific and Tech
and Zhejiang Province Key Laboratory of Quantum
[4]Beijing Academy of Quantu
[5]Institute of Information Technology, Info
[6]Beijing National Research
and School of Information
[7]Frontier Science Center for

Shor's algorithm has seriously challeng
However, to break the widely used RSA-2
far beyond current technical capabilities.
factorization by combining the classical latt
rithm (QAOA). The number of qubits requ
of the integer N, making it the most qubit
algorithm experimentally by factoring inte
integer factored on a quantum device. We
a depth of thousands is necessary to challe

23 Dec 2022

Peter Shor
@PeterShor1

Replying to @wayintothedeep

There are apparently possible problems with this paper.

Craig Gidney @CraigGidney · Dec 26, 2022
arxiv.org/abs/2212.12372 sure d[11:54 PM · Dec 26, 2022] space not mentioning the expected number of circuit shots it requires. It's critical to the entire premise of the paper to have a small bound on this number, and as far as I can tell they just don't talk about it. Very bad sign.

2:23 PM · Dec 27, 2022 · 2,981 Views

6 Retweets 1 Quote Tweet 25 Likes

The power of a quantum adversary on a blockchain

- Quantum adversary can forge currently used digital signatures = steal funds or fork consensus
  - ECDSA, Schnorr/EdDSA, RSA - breakable by a quantum computer
  - There are secure alternatives
- Solve PoW-puzzles faster: $D^{1/2}$ instead of D to search D-size space
  - Classical miner one thread: T time to search T space
  - Quantum miner one thread: T time to search T^2 space (Grover's search–1996)
  - Superlinearity problem: quantum miners have more advantage [Park-Spooner-2022]
- Hash functions stay secure
  - Collision: classical algorithm $O(2^{n/2})$ quantum algorithms $O(2^{n/3})$ [Brassard-Hoyer-Tapp-1997]
  - Quantum speed-up is not practical [Bernstein-2009]
  - Wide-believe is that SHA-256 still provides 128-bits collision resistance even post-quantum

NIST post-quantum standardization

- 2016 - NIST announced a competition
- 2022 (summer) - NIST announced finalists
- 2023 - to open drafts for public comments
- 2024 - to have the first PQC standards

Digital Signatures to be standardized: Crystals-Dilithium, Falcon, Sphincs+

Today

- 3 standard signature schemes: ECDSA, RSA and EdDSA
- BLS is widely used but is not standardized

Post-quantum signature finalists

Compared at level 3 (AES 192)

	pk	sig	sigs (ms)	Verifies (ms)	Assumption
Dilithium 3	**1.9 KB**	**3.3 KB**	AVX: 0.06 0.2	AVX: 0.06 0.2	Lattices
Falcon 1024	**1.8 KB**	**1.3 KB**	AVX: 0.7 **10**	AVX: 0.1 0.1	Lattices
Sphincs+ 192s,f+ SHAKE	48 B	**16 KB** **35 KB**	**5,000** **250**	**4** **10**	Hashes

BLS	**96 B**	**48 B**	**0.2**	**0.9**	**BDH**
EdDSA	**32 B**	**64 B**	**0.02**	**0.07**	**DH**
ECDSA	**32 B**	**72 B**	**0.04**	**0.07**	**Ideal model**
RSA	**348 B**	**348 B**	**2.5**	**0.05**	**RSA**

Post-quantum signature finalists

Compared at level 3 (AES 192)

	pk	sig
Dilithium 3	**1.9 KB**	**3.3 KB**
Falcon 1024	**1.8 KB**	**1.3 KB**
Sphincs+ 192s,f+ SHAKE	48 B	**16 KB** **35 KB**

Post-quantum signatures to be standardized by 2024

LMS/XMSS	48 B	**1-5 KB**

Stateful (few-times) post-quantum signatures that are standardized (2020)

BLS	**96 B**	**48 B**
EdDSA	**32 B**	**64 B**
ECDSA	**32 B**	**72 B**

Our current signatures, quantum-breakable

NIST: new call for digital signatures


COMPUTER SECURITY
RESOURCE CENTER
CSRC

UPDATES

2022

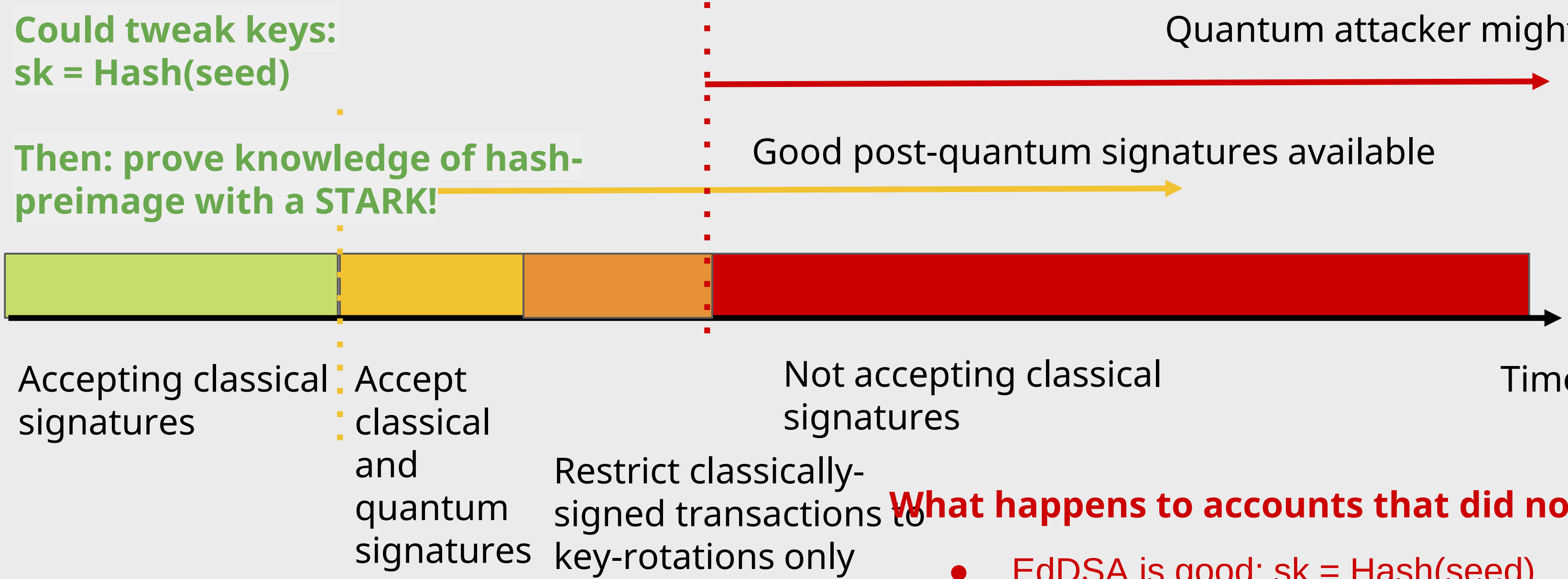
Request for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process

September 06, 2022

Ask: non-lattice-based signatures and/or short signatures

Deadline for submission: June 1, 2023

A happy transition of a blockchain to post-quantum



What happens to accounts that did not rotate?

- EdDSA is good: sk = Hash(seed)
- Bitcoin is good: Address = Hash(pk)

Schnorr/EdDSA vs. Dilithium

Schnorr/EdDSA

KeyGen():

sk ← random integer mod p
pk = G * sk,
// G generates a p-order group

Sign():

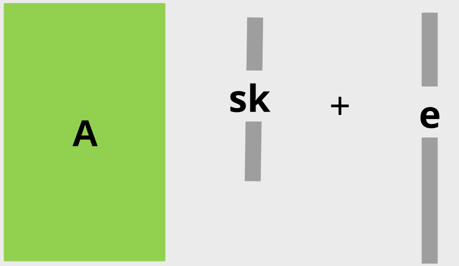
r ← random mod p
R = G * r
h = Hash(R, tx)
σ = (h, r + h*sk)

Dilithium

KeyGen():

sk ← random vector
e ← random small vector

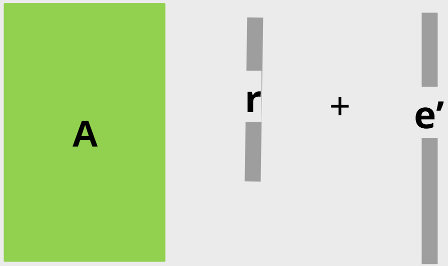
pk =



Sign():

r ← random vector
e ← random small vector

R =



h = Hash(R, tx)
σ = (h, r + h*sk)

Dilithium is essentially Schnorr but with matrix A instead of generator G

1: Long-Range Attacks on PoS
2: Proposer Election in PoS
3: Post-quantum blockchains
4: More:
Threshold Signatures

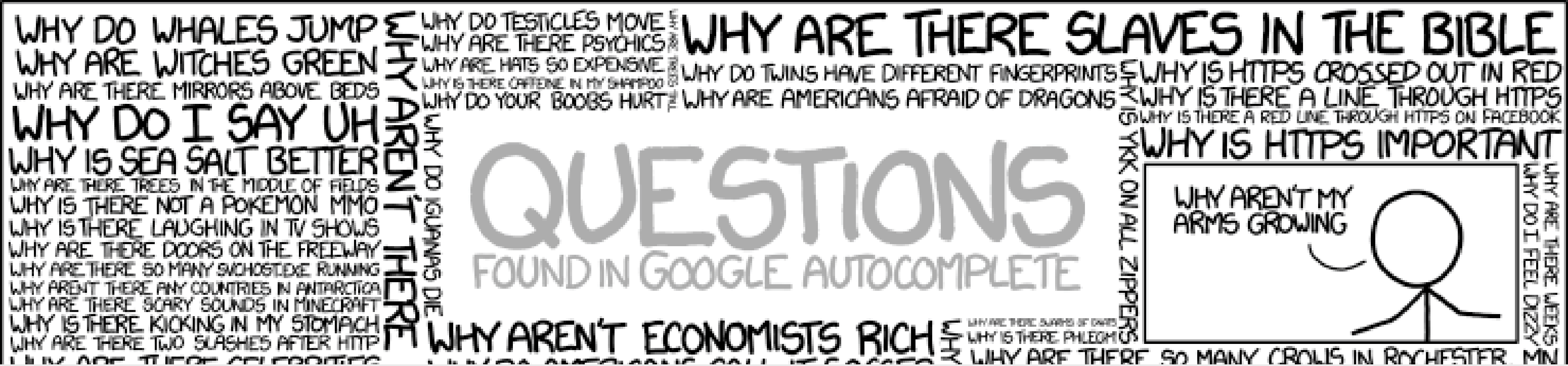
Threshold signatures

- In t-out-of-n threshold signature
  - A single public key, the secret key is split between n nodes.
  - Any t nodes can reconstruct the secret key, or sign through a multi-party protocol.
- Important for wallets: split the key between servers
- Blockchains signatures: ECDSA, Schnorr/EdDSA, BLS
- ECDSA is most widely used on blockchains (NIST standard 1994)
  - Hard to thresholdize
  - [CGGMP-2021]: 4 rounds (= 3 offline + 1 online)

Need good implementations!

- Schnorr/EdDSA (patent expired 2008)
  - Easy to thresholdize
  - FROST [KG-2020]: 2 rounds (= 1 offline + 1 online)
  - Being standardized by IETF
- BLS
  - Trivial to thresholdize: 1 round
  - Being standardized by IETF (expired)

Disclosures


QUESTIONS
FOUND IN GOOGLE AUTOCOMPLETE
WHY DO WHALES JUMP
WHY ARE WITCHES GREEN
WHY ARE THERE MIRRORS ABOVE BEDS
WHY DO I SAY UH
WHY IS SEA SALT BETTER
WHY ARE THERE TREES IN THE MIDDLE OF FIELDS
WHY IS THERE NOT A POKEMON MMO
WHY IS THERE LAUGHING IN TV SHOWS
WHY ARE THERE DOORS ON THE FREEWAY
WHY ARE THERE SO MANY SVCHOST.EXE RUNNING
WHY AREN'T THERE ANY COUNTRIES IN ANTARCTICA
WHY ARE THERE SCARY SOUNDS IN MINECRAFT
WHY IS THERE KICKING IN MY STOMACH
WHY ARE THERE TWO SLASHES AFTER HTTP
WHY AREN'T THERE
WHY DO TESTICLES MOVE
WHY ARE THERE PSYCHICS
WHY ARE HATS SO EXPENSIVE
WHY IS THERE CAFFEINE IN MY SHAMPOO
WHY DO YOUR BOOBS HURT
WHY DO IGUANAS DIE
WHY ARE THERE SLAVES IN THE BIBLE
WHY DO TWINS HAVE DIFFERENT FINGERPRINTS
WHY ARE AMERICANS AFRAID OF DRAGONS
WHY IS HTTPS CROSSED OUT IN RED
WHY IS THERE A LINE THROUGH HTTPS
WHY IS THERE A RED LINE THROUGH HTTPS ON FACEBOOK
WHY IS HTTPS IMPORTANT
WHY IS YKK ON ALL ZIPPERS
WHY AREN'T MY ARMS GROWING
WHY ARE THERE WEEKS
WHY DO I FEEL DIZZY
WHY AREN'T ECONOMISTS RICH
WHY IS THERE PHLEGM
WHY ARE THERE SO MANY CROWS IN ROCHESTER, MN

Thank you!


WHY ARE KYLE AND CARTMAN FRIENDS
WHY IS THERE AN ARROW ON AANG'S HEAD
WHY ARE TEXT MESSAGES BLUE
WHY ARE THERE MUSTACHES ON CLOTHES
WHY ARE THERE MUSTACHES ON CARS
WHY ARE THERE MUSTACHES EVERYWHERE
WHY ARE THERE SO MANY BIRDS IN OHIO
WHY IS THERE SO MUCH RAIN IN OHIO
WHY IS OHIO WEATHER SO WEIRD
WHY ARE THERE MALE AND FEMALE BIKES
WHY ARE THERE BRIDESMAIDS
WHY DO DYING PEOPLE REACH UP
WHY AREN'T THERE VARICOSE ARTERIES
WHY ARE OLD KLINGONS DIFFERENT
WHY ARE THERE SQUIRRELS
WHY IS WOLVERINE NOT IN THE AVENGERS
WHY ARE THERE ANTS IN MY LAPTOP
WHY IS EARTH TILTED
WHY IS SPACE BLACK
WHY IS OUTER SPACE SO COLD
WHY ARE THERE PYRAMIDS ON THE MOON
WHY IS NASA SHUTTING DOWN
WHY ARE THERE TINY SPIDERS IN MY HOUSE
WHY DO SPIDERS COME INSIDE
WHY ARE THERE HUGE SPIDERS IN MY HOUSE
WHY ARE THERE LOTS OF SPIDERS IN MY HOUSE
WHY ARE THERE SPIDERS IN MY ROOM
WHY ARE THERE GHOSTS
WHY IS THERE ICE IN SPACE
WHY IS THERE AN OWL IN MY BACKYARD
WHY IS THERE AN OWL OUTSIDE MY WINDOW
WHY IS THERE AN OWL ON THE DOLLAR BILL
WHY DO OWLS ATTACK PEOPLE
WHY ARE AK 47s SO EXPENSIVE
WHY ARE THERE HELICOPTERS CIRCLING MY HOUSE
WHY ARE THERE GODS
WHY ARE THERE TWO SPOCKS
WHY ARE MY BOOBS ITCHY
WHY ARE CIGARETTES LEGAL
WHY ARE THERE DUCKS IN MY POOL
WHY IS JESUS WHITE
WHY IS MT VESUVIUS THERE

#2 Simplest Randomness Beacon using VRFs and VDFs

Each node i pre-registers VRF public key: pk_i

In each epoch:

Contributes deterministic verifiable randomness

- node i submits v_i = VRF_Eval(sk_i, epoch_number)
- beacon = VDF_Eval($v_1 \oplus v_2 \oplus \ldots \oplus v_n$)

VRF = Verifiable Random Function

- Setup → (sk, pk)
- Eval(sk, x) → (y, [π])
- Verify(pk, x, y, [π]) → {yes/no}

- can't prove different $y_1 \neq y_2$ same x
- y is indistinguishable from random

Instantiated from unique signatures (BLS or RSA).

VDF = Verifiable Delay Function

- Setup → (pk)
- Eval(pk, x) → (y, [π])
- Verify(pk, x, y, [π]) → {yes/no}

- long to Eval, fast to Verify

Some VDFs require a trusted setup

VDFs = Verifiable Delay Functions

VDF = Verifiable Delay Function

- Setup $\rightarrow$ (pk)
- Eval(pk, x) $\rightarrow$ (y, π)
- Verify(pk, x, y, π) $\rightarrow$ {yes/no}

Long to Eval, fast to Verify.

Some VDFs require a trusted setup

- Unbiasable randomness beacons
- Time-release encryption or time-release commitments
- Proof-of-storage

Dilithium

$\underline{\mathsf{Gen}}$

$\mathbf{A} \leftarrow R_q^{k \times \ell}$

$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow S_\eta^\ell \times S_\eta^k$

$\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$

return $(pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2))$

$\underline{\mathsf{Sign}(sk, M)}$

$\mathbf{z} := \perp$

while $\mathbf{z} = \perp$ do

  $\mathbf{y} \leftarrow S_{\gamma_1 - 1}^\ell$

  $\mathbf{w}_1 := \mathsf{HighBits}(\mathbf{A}\mathbf{y}, 2\gamma_2)$

  $c \in B_\tau := \mathrm{H}(M \parallel \mathbf{w}_1)$

  $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$

  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathsf{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2, 2\gamma_2)\|_\infty \geq \gamma_2 - \beta$, then $\mathbf{z} := \perp$

return $\sigma = (\mathbf{z}, c)$

$\underline{\mathsf{Verify}(pk, M, \sigma = (\mathbf{z}, c))}$

$\mathbf{w}_1' := \mathsf{HighBits}(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma_2)$

if return $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![c = \mathrm{H}(M \parallel \mathbf{w}_1')]\!]$

Figure 1: Template for our signature scheme without public key compression.

Key Generation. The key generation algorithm generates a $k \times \ell$ matrix $\mathbf{A}$ each of whose entries is a polynomial in the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. As previously mentioned, we will always have $q = 2^{23} - 2^{13} + 1$ and $n = 256$. Afterwards, the algorithm samples random secret key vectors $\mathbf{s}_1$ and $\mathbf{s}_2$. Each coefficient of these vectors is an element of R_q with small coefficients of size at most η. Finally, the second part of the public key is computed as $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$. All algebraic operations in this scheme are assumed to be over the polynomial ring R_q.

1: Long-Range Attacks on PoS
2: Proposer Election in PoS
3: Post-quantum blockchains
4: More:

Threshold signatures

VDFs

Merkle/Verkle trees

Threshold Signatures

Other options

- **Double-sign**: generate a post-quantum signature alongside the classical signature
- **Onion-key-generation**: generate keys (qpk, qsk) for a stateful hash-based signature (LMS/XMSS), generate classical key: sk = $Hash^*$(qpk).

Categories of quantum breakable signatures

1. The quantum-adversary can fully recover the secret key from on-chain information
    a. ECDSA, BLS, Schnorr
2. The quantum-adversary can forge signatures but can not recover the secret key from on-chain information
    a. ECDSA, BLS, Schnorr with tweaks or EdDSA
3. The quantum adversary can't forge signatures from on-chain information
    a. Only new post-quantum secure signatures

Post-quantum signature finalists

Compared at level 3 (AES 192)

	pk	sig	keygen (ms)	sigs (ms)	Verifies (ms)	assumption
Dilithium 3	**1.9 KB**	**3.3 KB**	AVX: 0.07 0.2	AVX: 0.06 0.2	AVX: 0.06 0.2	Lattices
Falcon 1024	**1.8 KB**	**1.3 KB**	AVX: **25** **50**	AVX: 0.7 **10**	AVX: 0.1 0.1	Lattices
Sphincs+ 192s,f+ SHAKE	48 B	**16 KB** **35 KB**	**500** **8.3**	**5,000** **250**	**4** **10**	Hashes

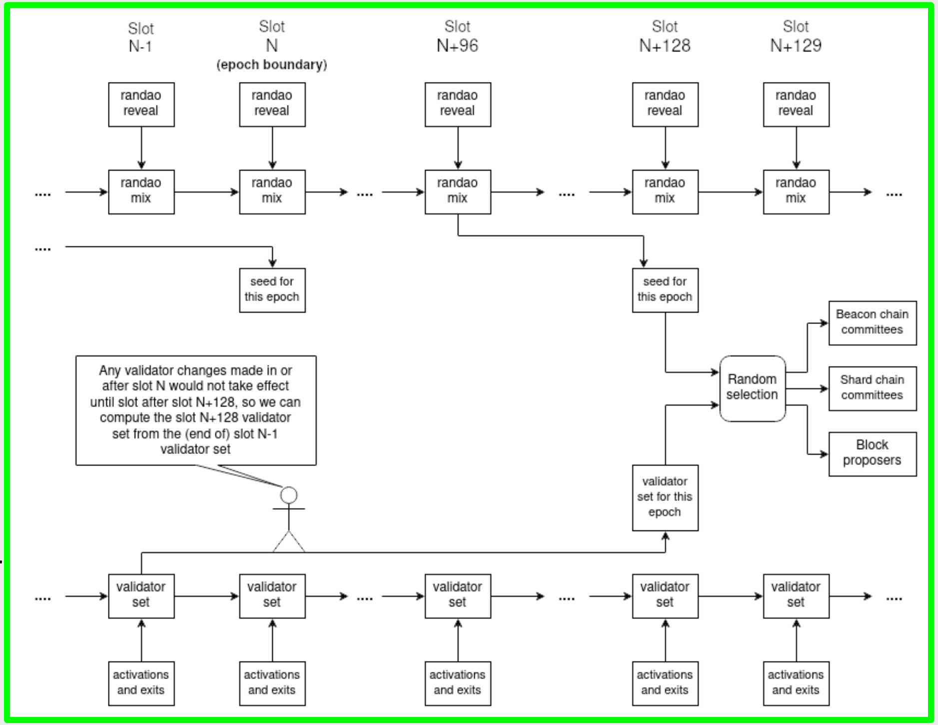
BLS	**96 B**	**48 B**		**0.2**	**0.9**
EdDSA	**32 B**	**64 B**	**0.003**	**0.02**	**0.07**
ECDSA	**32 B**	**72 B**	**-**	**0.04**	**0.07**
RSA	**348 B**	**348 B**	**-**	**2.5**	**0.05**

More post-quantum crypto-primitives

- Threshold signatures
- Zero-knowledge proofs
  - STARKs

Leader election approaches: RANDAO

RANDAO (Ethereum 2.0 approach):
Leaders of the previous epoch contribute verifiable deterministic randomness (BLS signing epoch number) to the next epoch.
Leader schedule gets known 1 epoch in advance.



Public: a leader learns they are a leader at the same time
the public learns.
Fair: all nodes are given equal chances to be leaders
(random sampling).
Unbiasable: last leader has only 1 bit of bias by either
revealing/withholding block
Unpredictable: the leader for the slot is known only 1
epoch in advance.

Good for leaders that take longer than an epoch (6.4 min) to DDoS

https://github.com/ethereum/annotated-spec/blob/master/phase0/beacon-chain.md#aside-randao-seeds-and-committee-generation

Leader election approaches: unbiasable and unpredictable

	Assumption	Setup	Single round	Self-cerifying
Albatross, HydRand, RandHerd	PVSS	**Free**	**No**	**No**
RandRunner, RANDAO++, cVDF, Veedo	VDF	**Yes/No**	**Yes**	**Yes**
Dfinity, drand	Threshold-signatures	**DKG**	**Yes**	**Yes**

- **Public**: a leader learns they are a leader at the same time the public learns.
- **Fair**: all nodes are given equal chances to be leaders.
- **Unbiasable:** no node can increase its probability of being selected.
- **Unpredictable**

⇓

- Random lotteries
- Randomness API for smart contracts
- Bootstrapping asynchronous consensus*

Good for leaders that are hard to DDoS within the time it takes them to create and broadcast a proposal (< 10 sec))!

VDF-based leader election: biasable → unbiasable

VDF = Verifiable Delay Function

- Setup → (pk)
- Eval(pk, x) → (y, π)
- Verify(pk, x, y, π) → {yes/no}

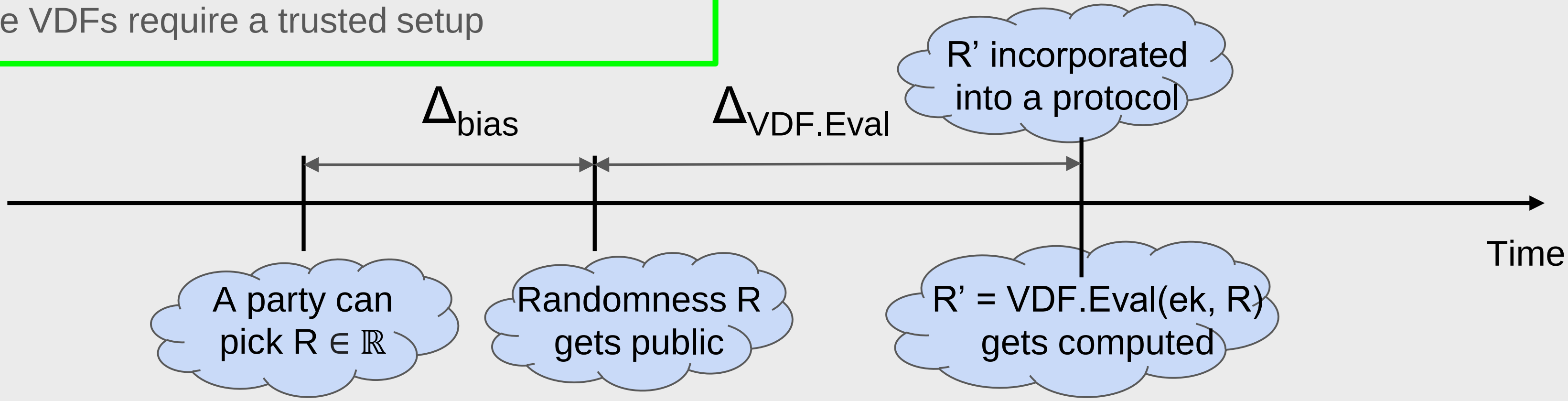
Long to Eval, fast to Verify.

Some VDFs require a trusted setup

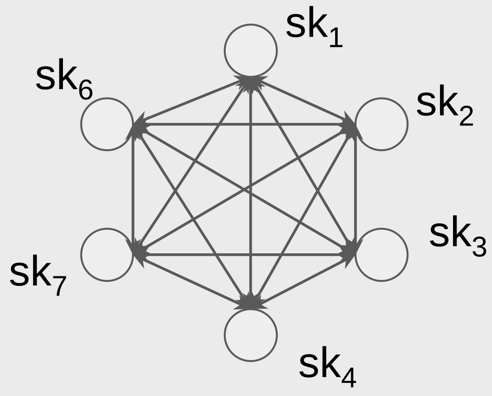
Passing randomness through a VDF makes it unbiasable!

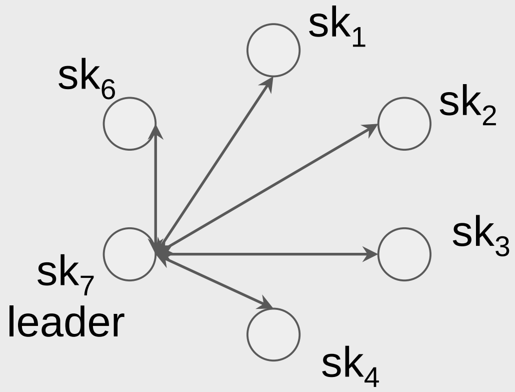
R' is unbiasable as long as

$$\Delta_{bias} < \Delta_{VDF.Eval}$$



Threshold-Signature-based leader election





- Setup: collectively generate sk (DKG)
  - sk can be reconstructed from any ⅔n subset of $\{sk_1, sk_2, \ldots, sk_n\}$
- Parties collectively-sign a slot number by submitting signature shares
  $\sigma_i = \mathrm{Sign}(sk_i, msg = \text{slot-number})$
- Full signature σ is reconstructed: from ⅔ n subset of $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$
- **Randomness is generated as Hash(σ)**
- Communication can be pipelined through a leader

Only works with unique signatures e.g. BLS

Private leader election : VRF-based

VRF = Verifiable Random Function

- Setup → (sk, pk)
- Eval(sk, x) → (y, π)
- Verify(pk, x, y, π) → {yes/no}

y is indistinguishable from random

Instantiated from unique signatures (BLS or RSA).

- Each leader computes VRF.Eval(sk, #slot) → (y, π), if y < threshold, the leader is elected
- The leader creates a block and broadcasts it together with π
  - No DDoS window!
- **Problem**: multiple leaders can get elected or none!
  - Consensus protocol needs to handle that (e.g. Algorand)