DEFCON BlockVillage

VDF for Preventing DoS / DDoS Attacks on Ethereum 2.0

Agenda

- Who are We!
- Attack Vectors on Proof of Work
- DDoS on Ethereum
- Attack Vectors on Proof of Stake
- Ethereum 2.0 Overview
- Ethereum 2.0 Audit Findings
- DDoS on Ethereum 2.0
- Randomness on Ethereum 2.0
- Verifiable Delay Functions
- VDF and DDoS Prevention
- Our Approach
- Questions
- Appendix

Who are we!

- Penetration Tester
- Blockchain Security Researcher
- Founder @ RazzorSec
- YouTuber @ Razzor Sharp
- Cybersec Enthusiast

Tejaswa Rastogi
Twitter @razzor_tweet

Who are we!

- Founder of EPIC Knowledge Society
- Chief Scientist, Fusion Ledger
- Ambassador - Algorand, Aeternity
- Director - Tezos India Foundation
- MVP - Hedera Hashgraph
- Global Leader - XX Collective
- Programmer and Poet
- Blockchain Security Auditor
- Quantum Algorithm Architect
- Post Quantum Cryptography Researcher

Gokul Alex
GitHub - @gokulsan
Twitter - @gokulgaze
Reddit - @gokulbalex

Attack Vectors on Proof of Work

Attack Vectors on Proof of Work

- Distributed denial of service (DDoS)
- Transaction malleability attack
- Timejacking
- Routing attack
- Sybil attack
- Eclipse attack

https://www.apriorit.com/dev-blog/578-blockchain-attack-vectors

Some Other PoW Attacks

User wallet attacks:

- Phishing
- Dictionary attacks
- Vulnerable signatures
- Flawed key generation
- Attacks on cold wallets
- Attacks on hot wallets

https://www.apriorit.com/dev-blog/578-blockchain-attack-vectors

Some Other PoW Attacks

Smart contract attacks:

- Vulnerabilities in contract source code
- Vulnerabilities in virtual machines

Transaction verification mechanism attacks:

- Finney attack
- Race attack
- Vector76
- Alternative history attack
- 51% or majority attack

https://www.apriorit.com/dev-blog/578-blockchain-attack-vectors

Some Other PoW Attacks

Mining pool attacks:

- Selfish mining
- Fork-after-withhold

https://www.apriorit.com/dev-blog/578-blockchain-attack-vectors

DDoS on Ethereum

Invalid blocks on Parity Client

Some Parity Ethereum nodes lost sync with the network because of the following incident - you send to a Parity node a block with invalid transactions, but valid header (borrowed from another block). The node will mark the block header as invalid and ban this block header forever but the header is still valid.

Vulnerability in Parity Nodes

In May, global hacking research collective SRLabs claimed that only two-thirds of the Ethereum client software that ran on Ethereum nodes had been patched against a critical security flaw discovered earlier this year. The data reportedly indicated that unpatched Parity nodes comprised 15% of all scanned nodes — implying that 15% of all Ethereum nodes were vulnerable to a potential 51% attack.

Underpriced DDoS attacks - EXTCODESIZE

Underpriced opcode vulnerability is due to the improper gas cost of EVM's EXTCODESIZE opcode.

Prior to the EIP 150 hard fork, the EXTCODESIZE opcode only charged 20 gas for reading a contract bytecode from disk and then deriving its length.

As a consequence, the attacker can repeatedly send transactions to invoke a deployed smart contract with many EXTCODESIZE opcodes to cause a 2 - 3x slower block creation rate

Paper - A Survey on Ethereum Systems Security: Vulnerabilities, Attacks and Defenses by Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu

Underpriced DDoS attacks - SUICIDE

This attack exploited the improper gas cost of EVM’s SUICIDE opcode(renamed to SELFDESTRUCT after EIP6) and the empty account in the State trie Vulnerability.

Th opcode is meant to remove a deployed smart contract and send the remaining ether to the account designated by the caller. When the target doesn't exist a new account is created even though no Ether may be transferred.

The attacker created 19 million new empty accounts via this opcode at low gas consumption, which wasted disk space and increased synchronisation and processing time

Paper - A Survey on Ethereum Systems Security: Vulnerabilities, Attacks and Defenses by Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu

Dead Accounts and Bloating State Database

Certain transactions taking long time to process

These transactions happened between blocks 2,286,910 and 2,717,576.

Somebody took advantage of an underpriced opcode

Creating millions of dead Ethereum accounts

This had the effect of bloating the state database.

It created tons of transaction traces.

Impact of DDoS attacks

Hacks against Geth client

Hacks create huge amount of clean up transactions

Each clean up transaction creates large set of traces

Hard forks

Hard forks do not remove the dead accounts

Quick fixes to the DDoS

Scanning an initial set of transactions

Measuring the traces from each transaction

Checking the frequency of traces in the transactions

Attack Vectors on Proof of Stake

Proof of Stake vs Proof of Work

incentive to DDoS others is much higher in POW compared to POS (or specifically Casper). The reasons is that POW is a zero sum game where other miners gain the reward if some miners can not submit a block.

Casper in contrast is designed as a coordination game where every participant gets most if all others can include their blocks.

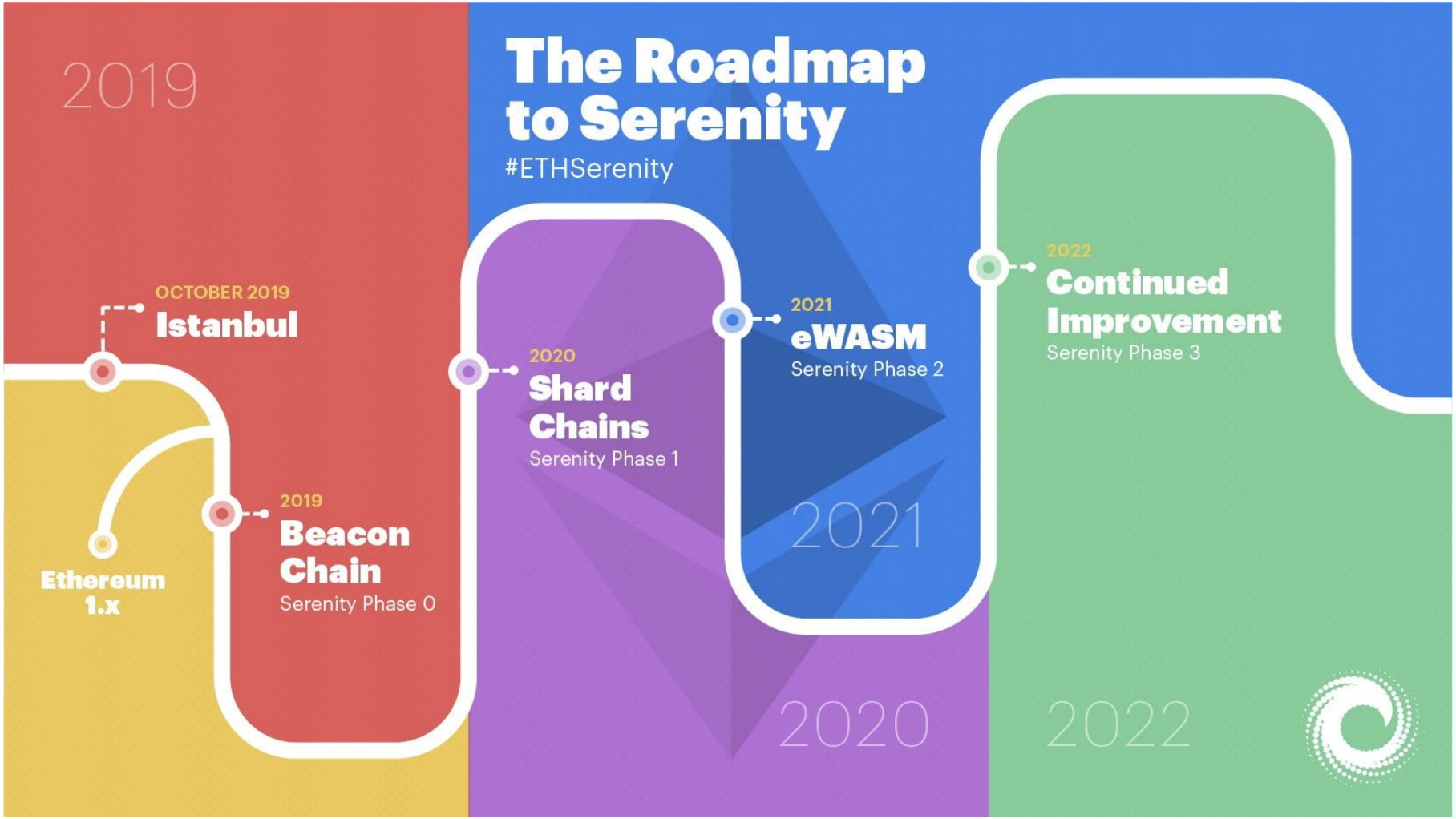
Better security against 51% attacks -High level goal of PoS is to ensure that each stakeholder chance of mining the next block is proportional to the number of coins they control.

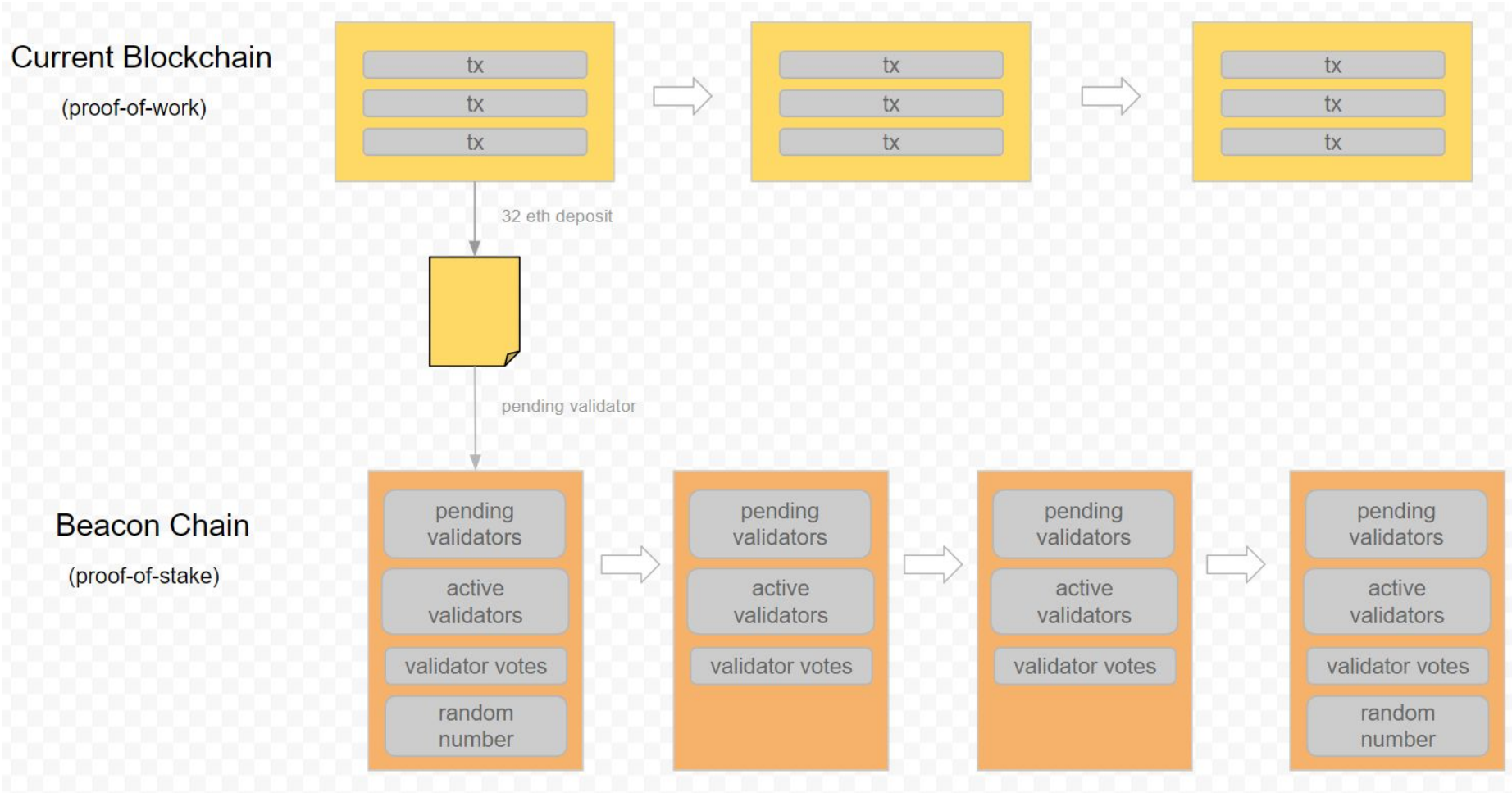
Attack against Proof of Stake

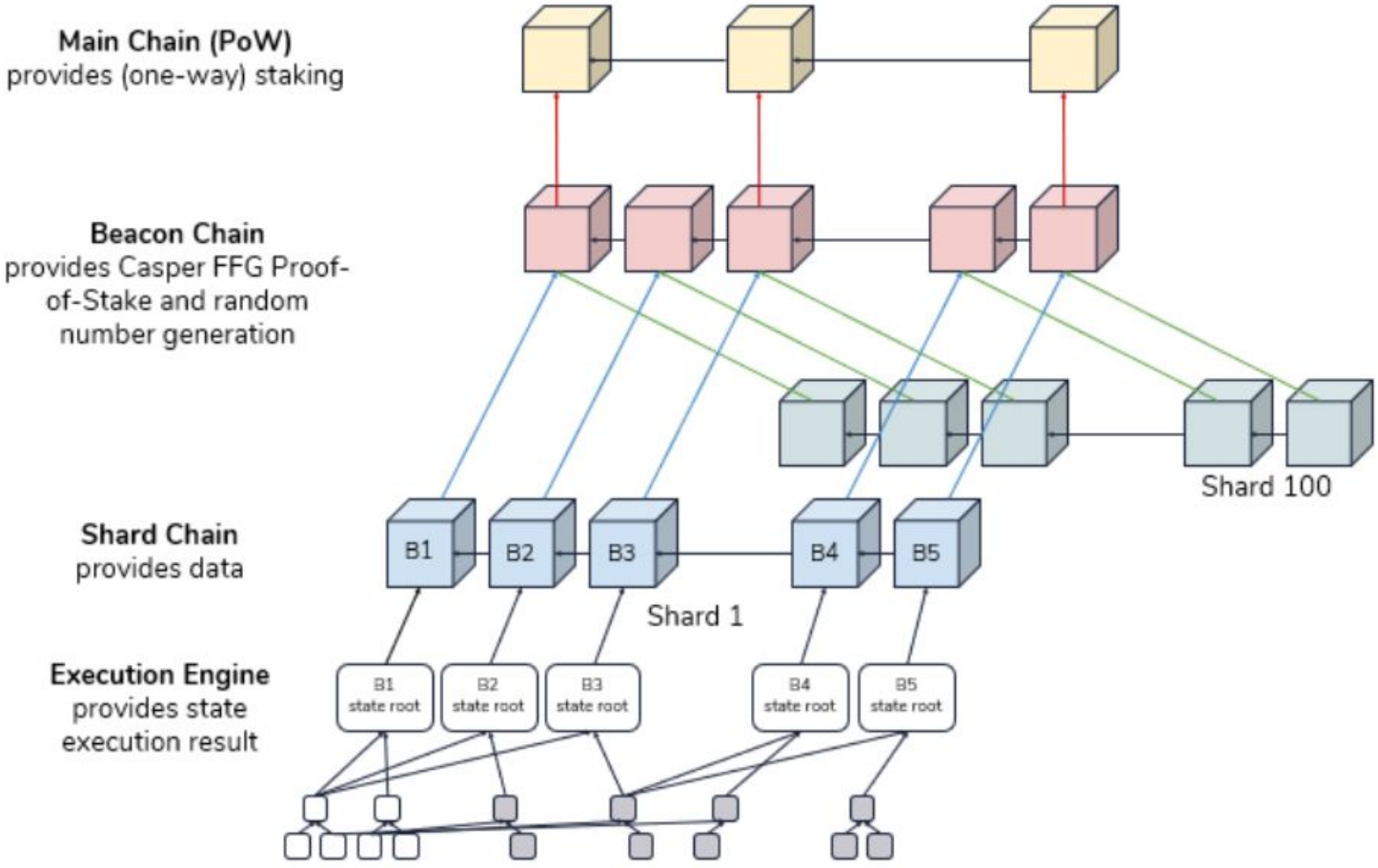
- xx% attack
- Fake Stake Attack
- Long Range Attack
- Nothing at Stake
- Sour Milk Attack

Credit - Infosecinstitute.com

Ethereum 2.0


2019
The Roadmap to Serenity
#ETHSerenity
OCTOBER 2019
Istanbul
2019
Beacon Chain
Serenity Phase 0
Ethereum 1.x
2020
Shard Chains
Serenity Phase 1
2021
eWASM
Serenity Phase 2
2021
2022
Continued Improvement
Serenity Phase 3
2020
2022


Current Blockchain
(proof-of-work)
tx
tx
tx
tx
tx
tx
tx
tx
tx
32 eth deposit
pending validator
Beacon Chain
(proof-of-stake)
pending validators
active validators
validator votes
random number
pending validators
active validators
validator votes
pending validators
active validators
validator votes
pending validators
active validators
validator votes
random number


Main Chain (PoW)
provides (one-way) staking
Beacon Chain
provides Casper FFG Proof-of-Stake and random number generation
Shard Chain
provides data
Execution Engine
provides state execution result
Shard 100
B1
B2
B3
B4
B5
Shard 1
B1 state root
B2 state root
B3 state root
B4 state root
B5 state root

Ethereum 2.0 Audit Findings

QuantStamp Audit

Possible DDoS Attack Vectors

DDoS attack through creating a mapping between public keys and validators' IPs

DDoS attack on Insecure gRPC connection

(Reported by Quantum Stamp Audit on Prysm)

https://github.com/prysmaticlabs/prysm/issues/6327

QuantStamp Audit

Quantstamp’s audit of Prysmatic Labs ETH 2.0 client involved ten engineers who examined the entire codebase over the course of two months.

They examined the beacon node logic, validator client, slasher logic, libp2p networking layer, gRPC API, client database, account management and key storage, client synchronization, and more.

QuantStamp Audit Findings

Vulnerabilities found by Quantstamp engineers and addressed by Prysmatic Labs included:

- Granularity of timestamps
- Pseudo-random number generation
- Second preimage attacks on Merkle trees

Least Authority Audit

Audit findings on the Block Proposer System

Single Secret Leader Election (SSLE) keeps the selection secret and stops the leak of information to an observer, while still allowing the chosen block proposer a fast way to verify to others that it is, in fact, the proposer.

With the information leak patched, the block proposer remains as protected as it would be in PoW chains, but without the computational overhead.

https://leastauthority.wpengine.com/static/publications/LeastAuthority-Ethereum-2.0-Specifications-Audit-Report.pdf

Audit Findings on the Gossip Protocol

Gossip protocols generally suffer from the spam problem; without a centralized judge, it can be difficult to understand whether a message is legitimate or is spam that is meant to clog the network.

This was one of the primary concerns in examining the networking layer. In Ethereum 2.0, when a node proposes a new finalized block, the block must be sent to the rest of the network.

There is an issue when a dishonest node is capable of sending an unlimited amount of older block messages to the rest of the network with minimal penalty, allowing them to overwhelm the network and block legitimate messages.

Audit Findings on the Slashing Messages

Similarly, when a node violates the rules, other nodes send out slashing messages to notify the rest of the network about who to penalize.

There is a a small loophole that allowed a node to send an unlimited amount of these types of messages with minimal penalty, causing the same message blocking if they sent enough of them.

https://leastauthority.com/blog/ethereum-2-0-specifications/

DDoS on Ethereum 2.0

Teku Attack Scenario

Two of four Teku nodes were targeted by five ordinary machines with a sustained DoS attack.

Initial loss of finality was achieved with two or three machines, but the others joined within a few epochs to ensure that the network could not recover.

https://github.com/ethereum/public-attacknets/issues/7#issuecomment-665966067

Teku Attack Implementation

> while true; do cat /dev/zero | pv -L 1M |nc XXX.XXX.XXX.XXX 9000 >/dev/null; done

https://github.com/ethereum/public-attacknets/issues/7#issuecomment-665966067

Impact of Teku Attack

The effect that the DoS attack had on the attacknet was a prolonged loss finality and required manual intervention to restore the network to a healthy state once the attack stopped.

The nodes under attack used large amounts of memory, were subject to multiple container restarts, had trouble staying connected to peers and one node's local clock was 20 mins slow.

https://github.com/ethereum/public-attacknets/issues/7#issuecomment-665966067

Teku Attack - Root Cause

Firstly responding to one byte with multiple bytes is a vector for various amplification attacks but that wasn't the issue that caused the loss of finality.

The second issue is that the responses were being written faster than they were read by the attacking peer and jvm-libp2p was not applying any throttling.

Eventually TCP back pressure kicked in, filling up the OS write buffers and the responses wound up being queued in user space memory. This pushed up both the on and off heap memory usage very substantially.

CPU also spiked significantly partly due to processing all those multi stream "messages" but mostly because of the resultant memory pressure and GC activity.

Teku Attack - Solution

Preventing this specific attack is straight forward, disconnect the peer immediately when an invalid (e.g. 0 length) multi stream message is received.

Resilience to DOS attacks increases significantly as you increase the number of nodes and diversity of clients, locations and network connections

https://github.com/ethereum/public-attacknets/issues/7#issuecomment-665966067

Randomness on Ethereum 2.0

Sharding, Staking and Randomness

Almost all sharding designs today rely on some source of randomness to assign validators to shards. Both the randomness and the validators assignment require computation that is not specific to any particular shard.

For that computation, existing sharding designs have a separate blockchain that is tasked with performing operations necessary for the maintenance of the entire network.

Sharding, Staking and Randomness

Besides generating random numbers and assigning validators to the shards, these operations often also include receiving updates from shards and taking snapshots of them, processing stakes and slashing in Proof-of-Stake systems, and rebalancing shards when that feature is supported.

Such chain is called a Beacon chain in Ethereum and Near, a Relay chain in PolkaDot, and the Cosmos Hub in Cosmos.

Distributed Randomness on Blockchain

Many modern blockchain protocols rely on a source of randomness for selecting participants that carry out certain actions in the protocol.

If a malicious actor can influence such source of randomness, they can increase their chances of being selected, and possibly compromise the security of the protocol.

Distributed randomness is also a crucial building block for many distributed applications built on the blockchain.

Essential Properties of Randomness

Three essential properties for distributed randomness on-chain :

1. It needs to be unbiased. In other words, no participant shall be able to influence in any way the outcome of the random generator.
2. It needs to be unpredictable. In other words, no participant shall be able to predict what number will be generated (or reason about any properties of it) before it is generated.
3. The protocol needs to tolerate some percentage of actors that go offline or try to intentionally stall the protocol.

RANDAO approach

The general idea is that the participants of the network first all privately choose a pseudo-random number, submit a commitment to such privately chosen number.

All agree on some set of commitments using some consensus algorithm, then all reveal their chosen numbers, reach a consensus on the revealed numbers, and have the XOR of the revealed numbers to be the output of the protocol.

Limitations of the RANDAO approach

It is unpredictable, and has the same liveness as the underlying consensus protocol, but is biasable.

Specifically, a malicious actor can observe the network once others start to reveal their numbers, and choose to reveal or not to reveal their number based on XOR of the numbers observed so far.

This allows a single malicious actor to have one bit of influence on the output, and a malicious actor controlling multiple participants have as many bits of influence as the number of participants they are controlling.

RANDAO + VDF Approach

To make RANDAO, un-biasable, one approach would be to make the output not just XOR, but something that takes more time to execute than the allocated time to reveal the numbers.

If the computation of the final output takes longer than the reveal phase, the malicious actor cannot predict the effect of them revealing or not revealing their number, and thus cannot influence the result.

Such a function that takes a long time to compute, is fast to verify the computation, and has a unique output for each input is called a verifiable delay function (VDF for short) and it turns out that designing one is extremely complex.

Ethereum Perspective on RANDAO + VDF

Ethereum presently plans to use RANDAO with VDF as their randomness beacon. Besides the fact that this approach is unpredictable and unbiasable, it has an extra advantage that is has liveness even if only two participants are online (assuming the underlying consensus protocol has liveness with so few participants).

For the family of VDFs linked above a specialized ASIC can be 100+ times faster than conventional hardware. Thus, if the reveal phase lasts 10 seconds, the VDF computed on such an ASIC must take longer than 100 seconds to have 10x safety margin, and thus the same VDF computed on the conventional hardware needs to take 100 x 100 seconds = ~3 hours.

Threshold Signatures

This approach is pioneered by DFNITY, is to use threshold BLS signatures. Boneh–Lynn–Shacham (BLS) signature scheme allows a user to verify that a signer is authentic. The scheme uses a bilinear pairing for verification, and signatures are elements of an elliptic curve group.

BLS signatures is a construction that allows multiple parties to create a single signature on a message, which is often used to save space and bandwidth by not requiring sending around multiple signatures. A common usage for BLS signatures in blockchains is signing blocks in BFT protocols.

Threshold Signatures in Practice

Say 100 participants create blocks, and a block is considered final if 67 of them sign on it. They can all submit their parts of the BLS signatures, and then use some consensus algorithm to agree on 67 of them and then aggregate them into a single BLS signature.

Any 67 parts can be used to create an accumulated signature, however the resulting signature will not be the same depending on what 67 signatures were aggregated.

Threshold Signatures in Practice

Turns out that if the private keys that the participants use are generated in a particular fashion, then no matter what 67 (or more, but not less) signatures are aggregated, the resulting multisignature will be the same. This can be used as a source of randomness:

The participants first agree on some message that they will sign (it could be an output of RANDAO, or just the hash of the last block, doesn't really matter for as long as it is different every time and is agreed upon), and create a multisignature on it. Until

Limitations of Threshold Signatures

This approach to randomness is completely unbiased and unpredictable, and is live for as long as 2/3 of participants are online (though can be configured for any threshold). While ⅓ offline or misbehaving participants can stall the algorithm, it takes at least ⅔ participants to cooperate to influence the output.

The private keys for this scheme need to be generated in a particular fashion. The procedure for the key generation, called Distributed Key Generation, or DKG for short, is extremely complex and is an area of active research.

RandShare approach

RandShare is an unbiased and unpredictable protocol that can tolerate up to ⅓ of the actors being malicious.

It is relatively slow, and the paper linked also describes two ways to speed it up, called RandHound and RandHerd, but unlike RandShare itself, RandHound and RandHerd are relatively complex.

The general problems with RandShare besides the large number of messages exchanged (the participants together will exchange O(n^3) messages), is the fact that while ⅓ is a meaningful threshold for liveness in practice, it is low for the ability to influence the output. There are several reasons for it:

RandShare approach

1. The benefit from influencing the output can significantly outweigh the benefit of stalling randomness.

2. If a participant controls more than ⅓ of participants in RandShare and uses this to influence the output, it leaves no trace. Thus a malicious actor can do it without ever being revealed. Stalling a consensus is naturally visible.

3. The situations in which someone controls ⅓ of hashpower / stake are not impossible, and given (1) and (2) above someone having such control is unlikely to attempt to stall the randomness, but can and likely will attempt to influence it.

NEAR protocol approach

1. Each participant comes up with their part of the output, splits it into 67 parts, erasure codes it to obtain 100 shares such that any 67 are enough to reconstruct the output,
2. Assigns each of the 100 shares to one of the participants and encodes it with the public key of such participant. They then share all the encoded shares.
3. The participants use some consensus (e.g. Tendermint) to agree on such encoded sets from exactly 67 participants.

NEAR protocol approach

1. Once the consensus is reached, each participant takes the encoded shares in each of the 67 sets published this way that is encoded with their public key, then decodes all such shares and publishes all such decoded shares at once.

2. Once at least 67 participants did the previous step, all the agreed upon sets can be fully decoded and reconstructed, and the final number can be obtained as an XOR of the initial parts the participants came up with in (1).

3. The idea why this protocol is unbiasable and unpredictable is similar to that of RandShare and threshold signatures: the final output is decided once the consensus is reached, but is not known to anyone until ⅔ of the participants decrypt shares encrypted with their public key.

Verifiable Delay Functions

Anatomy of a VDF

A VDF consists of a triple of algorithms: Setup, Eval, and Verify.

Setup (λ, t) takes a security parameter λ and delay parameter t and outputs public parameters pp (which fix the domain and range of the VDF and may include other information necessary to compute or verify it).

Eval (pp, x) takes an input (x) from the domain and outputs a value (y) in the range and (optionally) a short proof π.

Finally, Verify (pp, x, y, π) efficiently verifies that y is the correct output on x.

Crucially, for every input x there should be a unique output y that will verify.

Historic references to VDF for DDoS prevention

To prevent email spamming, in the early 1990s, Cynthia Dwork and Moni Naor suggested using squaring roots over finite fields puzzles as functions which take a predetermined time to compute, and are straightforward to verify.

However, their work was considered impractical because one has to use rather large finite fields to make the algorithm useful, and the libraries for handling multiple precision arithmetic at the time of the suggestion of the algorithm were orders of magnitude slower than current ones.

Essential Properties of VDF

Sequential: honest parties can compute $(y,\pi) \leftarrow \text{Eval}(pp,x)$ in (t) sequential steps,

while no parallel-machine adversary with a polynomial number of processors can distinguish the output y from random in significantly fewer steps.

Efficiently verifiable: We prefer Verify to be as fast as possible for honest parties to compute; we require it to take total time - O(polylog(t)).

Unique: for all inputs x, it is difficulty to find a y for which

Verify (pp,x,y,π) = Yes, but $y \neq \text{Eval}(pp, x)$.

Additional Properties of VDF

Decodable - A VDF is decodable if there exists a decoding algorithm Dec such that (Eval,Dec) form a lossless encoding scheme.

Incremental - A single set of public parameters (pp) supports multiple hardness parameters (t). The number of steps used to compute (y) is specified in the proof, instead of being fixed during Setup.

The main benefit of incremental VDFs over a simple chaining of VDFs to increase the delay is a reduced aggregate proof size.

This is particularly useful for applications of VDFs to computational time-stamping or blockchain consensus.

Main VDF Constructions

There are currently three candidate constructions that satisfy the VDF requirements. Each one has its own potential downsides.

The first was outlined in the original VDF paper by Boneh, et al. and uses injective rational maps. However, evaluating this VDF requires a somewhat large amount of parallel processing, leading the authors to refer to it as a “weak VDF.”

Later, Pietrzak and Wesolowski independently arrived at extremely similar constructions based on repeated squaring in groups of unknown order.

These constructions are based on modular exponentiations, where Pietrzak and Wesolowski suggest to iteratively compute τ squarings in an RSA group, with τ large.

Innovations in VDF

The main innovation in VDF is to propose a setup phase to set a trusted environment allowing the functions to be universally verifiable.

This trusted environment sets the public parameters of the VDF, including its difficulty which determines the amount of time spent on computing.

Any node who needs to solve the VDF will use the public parameters to perform certain sequential computations.

Some VDFs also allow generating a proof to facilitate the verification from the other participants.

VDF and DDoS Prevention

IOTA Approach

A DoS prevention mechanism where nodes are required to compute exactly τ modular squarings of a given input message

Calibrating the VDF evaluation time on different hardware and optimising the time needed to verify the correctness of the puzzle through multi-exponentiation techniques

Extensive experimental evaluations to compare VDFs and PoW on different hardware. In particular show that larger monetary resources cannot help to speed up the VDF evaluation

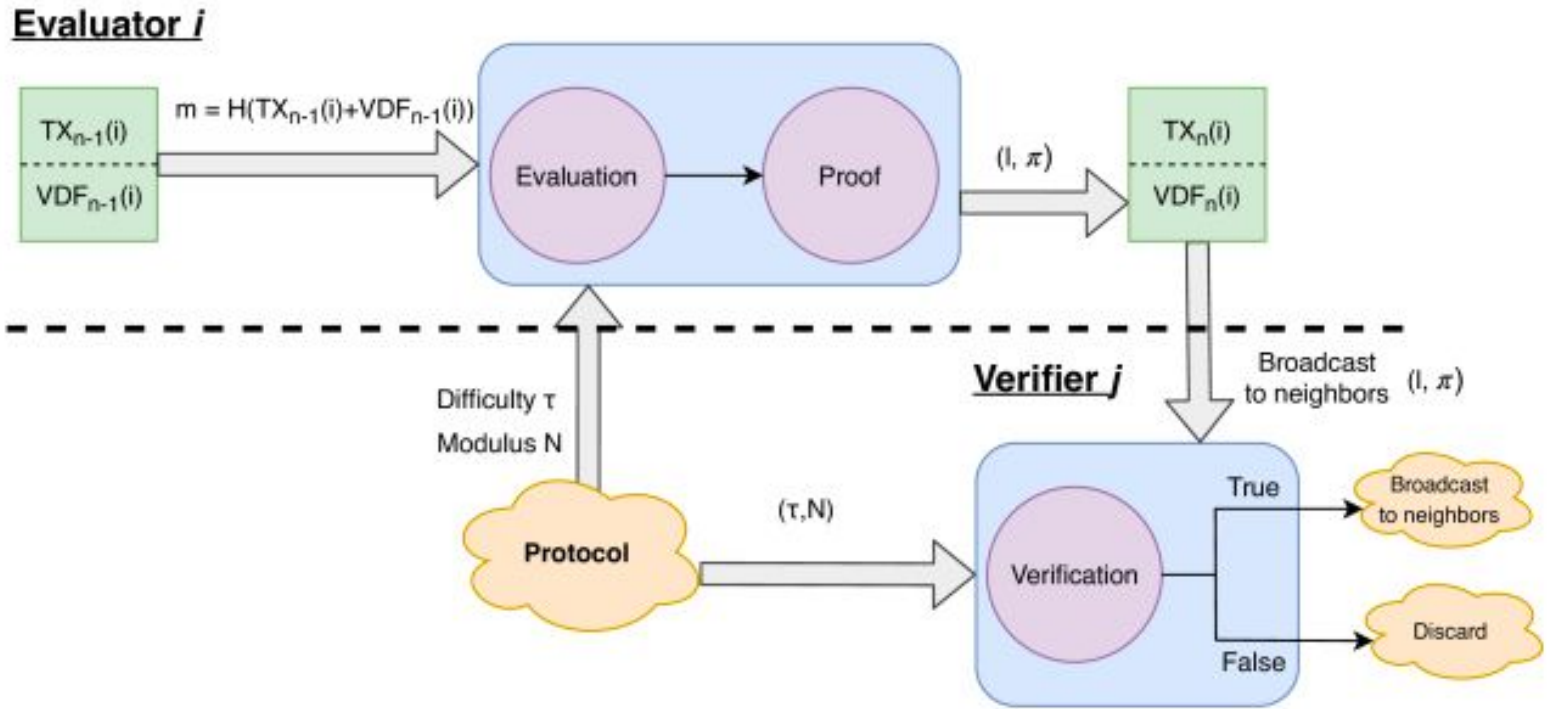
https://arxiv.org/pdf/2006.01977.pdf

IOTA DoS Prevention Mechanism

- Evaluation - When node i decides to generate transaction n, it is required to solve a VDF such that its input is the hash of transaction n − 1 issued by the same node.
- Proof - Node i also generates a proof to facilitate the verification task, which gossips along with the transaction.
- Verification - When a new transaction is received, node j verifies whether the VDF has been solved correctly. If yes, it forwards the transaction (and the proof) to its neighbors, or discards it otherwise.

https://arxiv.org/pdf/2006.01977.pdf

VDF Integration in the IOTA protocol



Public Inputs

- VDF Difficulty - A difficulty which indicates the number of sequential operations to solve and can be updated accordingly to mitigate attacks
- RSA Module - A modulus $N = p \cdot q$ which has bit-length λ (typically 1024, 2048, or 3072), and is the product of two prime numbers of the same order. The security of the entire mechanism relies on the length of N: the longer is the modulus, the more difficult is to find its factorization
- Cryptographic hash function - A hash function that will be used in the evaluation and proof of VDF

https://arxiv.org/pdf/2006.01977.pdf

Our Approach

Our Architecture Overview

Supersingular Isogeny based VDFs

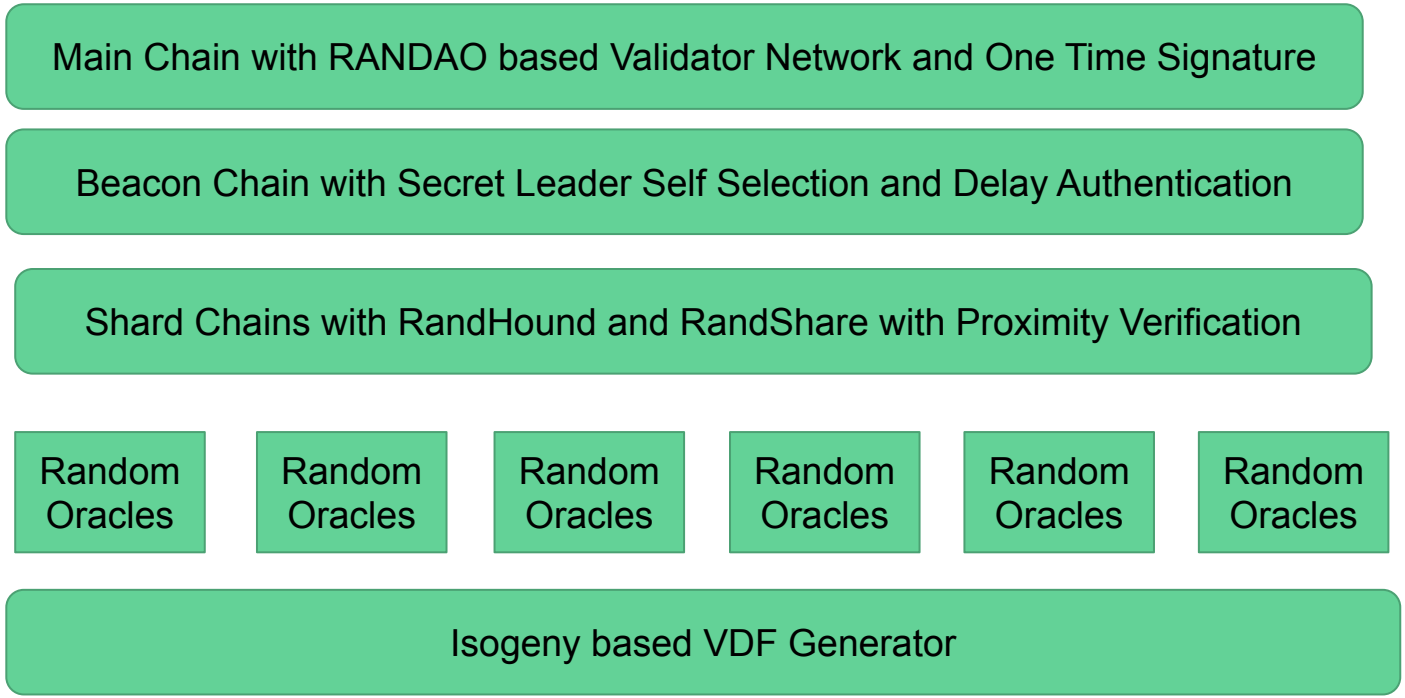
VDF based single secret leader selection

VDF based Random Oracles

VDF based delay authentication to the RANDAO

Randomness (Randshare and Randhound) powered beacon chain

Solution Architecture Framework for our approach



TemerNet - Our Proof of Concept for Random Oracle

https://github.com/EPICKnowledgeSociety/TemereNet

```
pragma solidity ^0.5.2;
import "github.com/starkware-libs/veedo/blob/master/contracts/BeaconContract.sol";
contract Beacon{
   function getLatestRandomness()external view returns(uint256,bytes32){}
}

contract WeatherOracle {
 address public oracleAddress;
 address public BeaconContractAddress=0x79474439753C7c70011C3b00e06e559378bAD040;
 constructor (address _oracleAddress) public {
   oracleAddress = _oracleAddress;
 }
```

TemerNet - Our Proof of Concept for Random Oracle

https://github.com/EPICKnowledgeSociety/TemereNet

```
function setBeaconContractAddress(address _address) public  {
       BeaconContractAddress=_address;
   }

   function generateRandomNumber() public view returns(bytes32){
       uint blockNumber;
       bytes32 randomNumber;
       Beacon beacon=Beacon(BeaconContractAddress);
       (blockNumber,randomNumber)=beacon.getLatestRandomness();
       return randomNumber;

   }
```

Single Secret Leader Election (SSLE)

In a Single Secret Leader Election (SSLE), a group of participants aim to randomly choose exactly one leader from the group with the restriction that the identity of the leader will be known to the chosen leader and nobody else.

At a later time, the elected leader should be able to publicly reveal her identity and prove that she has won the election. The election process itself should work properly even if many registered users are passive and do not send any messages.

Among the many applications of SSLEs, their potential for enabling more efficient proof-of-stake based cryptocurrencies have recently received increased attention.

https://eprint.iacr.org/2020/025.pdf

Node Topology in Shard Chain using VDF

Every node is connected to m << n neighbours and is involved in the generation and the verification of transactions, which simply carry proof carrying data (PCD).

It is important to limit the shard creation as a node could theoretically generate an infinite number of transactions per second, without an appropriate access control mechanism.

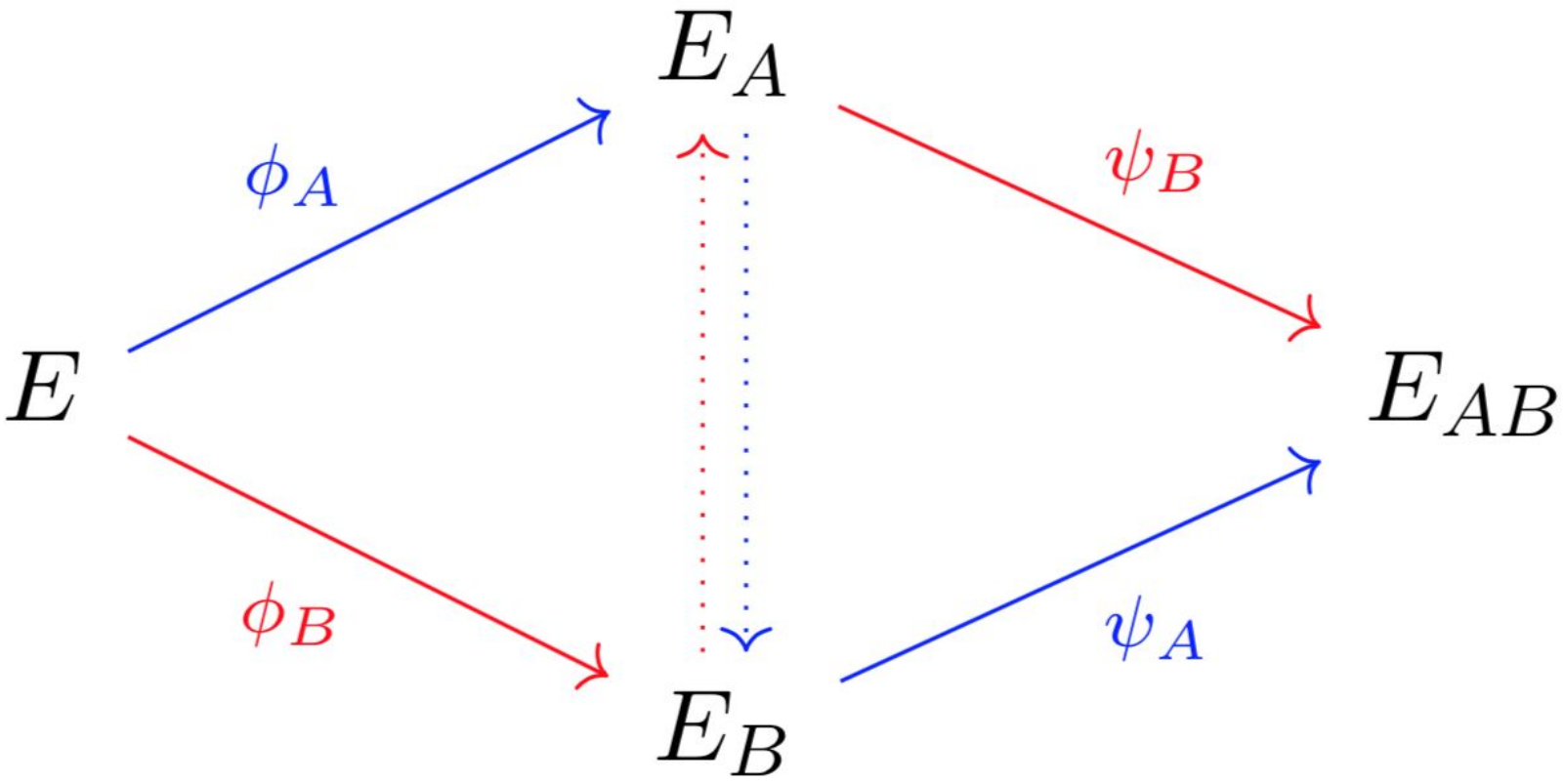
Every node has to evaluate a function (f) before issuing a transaction.

Isogeny based VDF

If we have two elliptic curves (E1 and E2), we can create a function that maps point (P) on E1 to a point Q on E2. This function is known as an isogeny. If we can map this function, every point on E1 can then be mapped to E2.

Our secret key is then the isogeny, and the public key is the elliptic curve that we are using. With isogenous elliptic curves we thus do not deal with a single elliptic curve but a family of them.

http://blog.intothesymmetry.com/2019/07/on-isogenies-verifiable-delay-functions.html


E_A
ϕ_A
ψ_B
E
E_{AB}
ϕ_B
ψ_A
E_B

History of isogeny-based cryptography

- 1996: Couveignes introduces isogeny in cryptography (**paper rejected Eurocrypt published 10 years later**)
- 2005: Rostovtsev & Stolbunov independently rediscover Couveignes ideas
- 2006: Charles, Goren & Lauter propose supersingular for a “provably secure” hash function
- 2011: Jao, De Feo introduce SIDH, an efficient post-quantum key exchange (**SIDH**)
- **2012: …**

Construction of an Isogeny based VDF

Setup: the setup phase of this VDF consists of

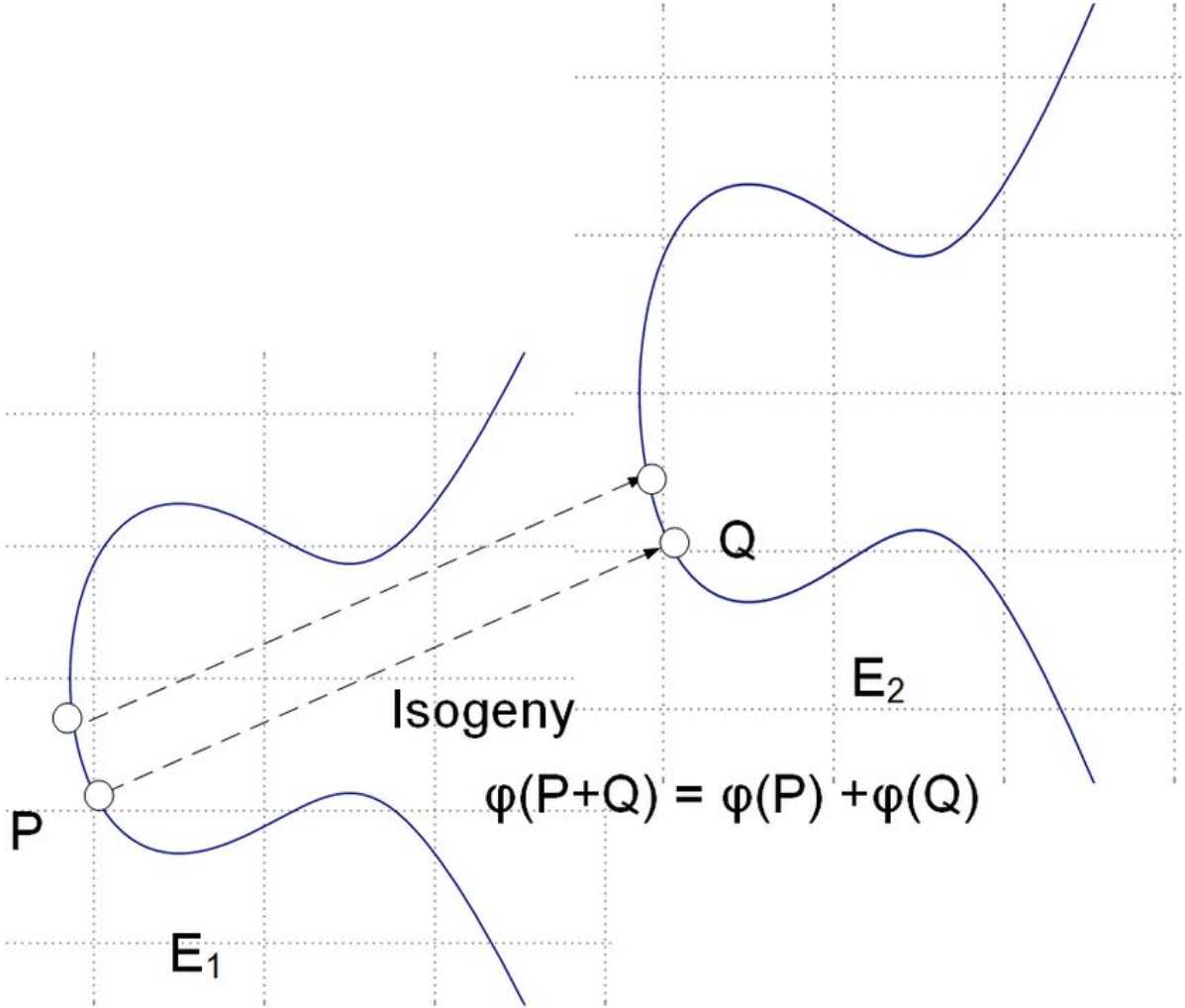
Choosing a prime number N

Selecting a supersingular curve E

Performing a random non-backtracking walk of length T having as outcome the isogeny ϕ and it's dual $\phi^\wedge$ (every isogeny has a dual isogeny).

Choosing a point P and compute $\phi(P)$

Output $\phi^\wedge, E, E', P, \phi(P)$


Q
E_2
Isogeny
$\varphi(P+Q) = \varphi(P) + \varphi(Q)$
P
E_1

Evaluation and Verification of an Isogeny based VDF

Evaluation :

Receiving a random point Q > Compute $\phi^{\wedge}(Q)$

Verification :

Use the Weil pairing and isogenies to verify $eN(P,\phi^{\wedge}(Q)) = eN(\phi(P),Q)$

http://blog.intothesymmetry.com/2019/07/on-isogenies-verifiable-delay-functions.html

Our Roadmap

- Implementation of Isogeny VDF Key Exchange using PQCrypto-SIDH
  - https://github.com/microsoft/PQCrypto-SIDH
- Implementation of Isogeny VDF Authentication using SS Isogeny
  - https://github.com/defeo/ss-isogeny-software
- Implementation of Single Secret Leader Selection using Mathcrypto
  - https://github.com/mathcrypto/SSLES
- Deployment of our Isogeny VDF with RANDAO
  - https://github.com/randao/randao