Oasis Whitepaper Overview

*Note: All words marked with the * symbol are featured in the Terminology section at the end.*

1 Introduction

- The 1st generation of blockchain (Bitcoin) allowed token transfers, then 2nd generation (Ethereum) allowed generic computation through smart contracts. The next generation will allow private computation.
- The Oasis Blockchain Platform is a PoS* layer 1 smart contract platform that provides scalability, extensibility, and privacy. It allows the consensus layer to be modified, and uses a completely different mechanism for achieving consensus, and follows any progress in the blockchain space. It describes how the consensus layer can run parallel runtimes or ParaTimes.
- Each ParaTime is a separate computing environment that offers both verifiable and confidential techniques in their environment. It verifies updated values (if correct) in the ParaTime before sending them over to the consensus layer.
- This is currently done through replicated computation. Oasis calls this discrepancy detection. It utilizes a smaller and fast path BFT* method for its execution. Due to its smaller and optimized size, the scalability for smart contracts improves, and thus the computing cost is lowered dramatically. Since smart contract executions are separated from the consensus layer in their environment, discrepancies are easier to detect because of the replication* that is required is reduced. Oasis specifies that their discrepancy technique is not rooted in their architecture, and those who wish to implement a ParaTime can use other forms of verifiable computation, such as Multiparty Computation*, Fully Homomorphic Encryption*, Zero-knowledge Proof*, and so on. The network was designed, however, with a working ParaTime that uses TEE* (Keystone*) for efficient, cost-effective, confidential smart contract execution. It does not require the ParaTimes to provide confidentiality or any particular confidential computing technique. Any ParaTime that doesn't use confidential computing can co-exist with a TEE-based ParaTime. Implementations on the consensus layer are not dependent on the ParaTimes; the same goes for the ParaTime implementations in that it's independent of the consensus layer. New ParaTimes do not affect the existing Paratimes.

2 Design goals and principles

- 2.1 Goals
  - The whitepaper states that the goal is to make the platform flexible, extensible, scalable, secure, and fault-isolated.
  - **Flexible:** Easy to modify the system parameters to accommodate deployment for the ParaTime
  - **Extensible:** It should be easy to extend by allowing/adding new components and implementations for the ParaTimes
  - **Scalable:** The transaction system needs to scale with the amount of work it does in the period through the total number of nodes. It allows the system to grow without huge costs.
  - **Secure:** The system enforces security policies that allow for a broader application scope. The focus is here on verifiable computations with the ability to perform confidential computations. The architecture favors simplicity when feasible, allowing implementations and audits to be more practical and effective.
  - **Fault-isolated:** Oasis is focused on making faults happen in isolation inside each ParaTime so that one fault doesn't affect the consensus network itself.
  - The combination of these goals gives developers and users different choices they can build and implement on the Oasis Blockchain.
- 2.2 Principles
  - The system has been designed to keep a clear separation between the bare bones consensus layer and the different independent ParaTime layers.
  - **Consensus layer:** This layer handles transactions, staking, token balance, delegation, and debonding. ParaTime committee scheduling and seeing whether the values of the ParaTime correspond with the parameters set by the deployer.

- **Independent ParaTimes:** ParaTimes can be optimized, be they confidential smart contracts or integrity-based smart contracts, where users can be confident that its content is wholly secured and less importance is based on visibility.
- There's no exact concurrency* for smart contract executions. However, multiple instances can run at the same time through separate ParaTimes. Later ParaTimes can be developed, for usage of more advanced techniques. ParaTimes can also be set to only run on a particular set of nodes to fulfill legal obligations.
- If one ParaTime fails, it won't get sent to the consensus layer, and it will not impact the rest of the operations. The scheduling of the ParaTimes prevents it from spamming the consensus layer with too many updates/values. So if one constantly spams the consensus layer, they'll have to pay for the transaction fees each time it submits a value. Other ParaTimes are not affected by it as they are considered separate entities.
- Because of this division, it does mean that the complexity increases and auditing become more difficult. If any discrepancy is detected, meaning if a value doesn't add up with the different ParaTimes, the nodes (randomly chosen) on the consensus layer try to replicate it. If even one node fails to produce the exact result as the other nodes, it is treated as a discrepancy. Then a large committee has to commit what is the correct answer. Because it gets verified on the random nodes on the consensus layer, it becomes more cost-effective. Other verifiable methods, such as the popular Arbitrum optimistic roll-up solution or a ZKP mechanism can be added as well.

3 Architecture

- 3.1 Architecture Overview
- The overall architecture consists of two main components, 1) The consensus layer and 2) ParaTimes.
    - 1) Consensus layer:
        - Deals with the creation of blocks filled with transactions to form the underlying blockchain. Includes committee selection for both validators and ParaTimes.
    - 2) ParaTimes:
        - Where the actual execution of smart contracts occurs. The exact nature of the execution environment (for example, the Ethereum Virtual Machine*) is customizable to a given ParaTime, but overall they share the ability to have sequenced events, normally in the form of transactions. ParaTimes commit their transactions to the consensus layer, and may include the following features:
        - i) Key managers*:
            - Only needed when using a TEE-based confidential ParaTime, which store the encrypted contract state. Sapphire, which has features such as obscuring token transfer amounts, is an example of such a ParaTime, while Emerald, which is a transparent EVM ledger, does not require key management.
        - ii) Committee selection pool criteria:
            - This includes any requirements for nodes to qualify for selection beyond a simple ROSE stake threshold amount.
- The consensus layer itself does not support confidential computation, but does enable the following 5 services:
    - 1) Native token operations (e.g. transfers, staking, delegation, etc.) This means wallet movements of ROSE on the consensus layer are completely public.
    - 2) Validator committee selection
    - 3) ParaTime committee selection
    - 4) ParaTime results verification
    - 5) Comittment of verified ParaTime results to a blockchain (the Oasis Network itself).
- The consensus layer allows for verifiable computing*, currently in the form of discrepancy detection* (explained more in section 4). With discrepancy detection, a result from a ParaTime can either be accepted and passed to the consensus layer for processing as part of the blockchain, or rejected, in which case the

ParaTime must perform "slow-path" processing. Confidentiality for smart contract execution is implemented solely within ParaTimes. Regardless of the confidentiality technique used, no information leaks occur when passing values to the consensus layer as long as there are no leaks on the execution layer's state. Smart contract developers can choose among the available ParaTimes based on which of them satisfies their application's security needs. They can also propose their own ParaTime as a ParaTime developer.

- 3.2 Conensus Layer
- The consensus layer is a simple BFT / Proof of Stake blockchain responsible for consensus. It is based on the Tendermint BFT consensus protocol*. Each consensus layer block contains a limited number of values from ParaTimes, and global execution order is determined by consensus. Due to the modular architecture, the consensus layer can be replaced with very little impact on the rest of the system.
- 3.3 Consensus–ParaTime Interface
- Consensus nodes each receive a submitted value from a given ParaTime layer, and then the consensus algorithm determines which values should be committed to the blockchain. There are many ways to pass messages from ParaTime nodes to consensus nodes, and ideally this process should be as efficient as possible. A streamlined process is used in which a verification mechanism on the ParaTime layer sends a single message to the validator process. This allows for better configuration and scheduling. When new ParaTimes are to be added, minor changes or library additions may be needed to the consensus layer.
- 3.4 ParaTime Layer
- The ParaTimes are where smart contract execution occurs. Separating consensus and execution allows each process to be optimized for their specific task. It is of note that ParaTimes which utilize TEEs for confidentiality can also provide protection for the computation's integrity, and as a result may require a lower replication factor than non-TEE ParaTimes. TEE-based ParaTimes require a key manager, while homomorphic encryption (FHE) does not require keys.
- 3.5 Key Manager
- The key manager is responsible for maintaining control over the cryptographic keys used to protect confidential contract state. The key properties they should provide are:
    - 1) Confidentiality: Only authorized, attested ParaTime compute nodes can access the keys for confidential smart contract execution. Proper cryptographic protection must also be used for communications between the key manager and compute nodes.
    - 2) Availability: The key manager must have replication across geographic locations for the ParaTime to execute smart contracts. This implies integrity of the key store as well as communcation security.

4 Discrepancy Detection

- Oasis dicrepancy detection is the verifiable computing technique used to verify its ParaTime executions as of this moment. It allows the use of smaller ParaTime committee to increase effiency. It does this via a random selection of compute nodes, and only requires that all randomly selected nodes agree on the result. If a discrepancy is detected a seperate protocol is used; discrepancy resolution. Detection as the name implies, finds it (detect) and the resolution is the slower, more expensive method that corrects the faults.
- How it functions in brief:
    - Results of whatever is being computed is signed by a node, then sent to the discrepancy detectors that uses a gossip network*. The process of detection starts with in the verification code that's located within the validator node.
    - The code checks the result, each computing node should only sign one result, anything else is double signing and will result in slashing.
        - Should all the results be free of discrepancy, they're submitted to the validators for consensus to process and create a new block in the chain.
        - If a discrepancy is detected however, the resolution phase begins to determine the correct result. The nodes that have discrepancies are slashed, for whatever amount pays for the re-execution cost.
- Unless there's a hardware failure, or errors in the ParaTime (a non-deterministic execution, or a non-adaptive adversary*), the resolution mechanism should never be triggered.

- By calculating the probability of selecting a committee that are all bad actors, Oasis have set their parameters to drive this probability as low as possible, so that the cost to attack would be unacceptable. The committee needed for this is, on Oasis, much smaller than other conventional systems. And thus more cost effective and easier to scale.

5 Parallell Runtimes (Refer to ParaTime definition)

- 5.1 Private Computation Runtime
    - 5.1.1 TEE Security Model
        - - TEE stands for Trusted Execution Environment
            - - "The TEE typically consists of a hardware isolation mechanism, plus a secure operating system running on top of that isolation mechanism"[1]
            - - Guarantees confidentiality, security, and integrity of your application state , even when the host is malicious[2]
        - - Code is placed in a TEE-provided secure execution environment, called **enclave.**
            - - In an enclave, the execution cannot be tampered with after initalization
            - - TEE-enabled hosts can prove their idenities to external observers through remote attestation.
                - - Remote attestation allows changes to the user's computer to be detected by authorized parties.[3]
                - - Similar to how certification authorities are used in public-key cryptographic systems.
        - - TEE is robust against threats almost everything outside of the CPU chip packaging
            - - Attackers are not allowed access to the CPU's internal state
    - 5.1.2 TEE Vulnerabilities
        - - The enclave has no persistent storage
            - - Input/Output is done via non-enclave, external component
        - - TEEs allow for the enclave components to be restarted and continue from an earlier checkpoint
    - 5.1.3 State Rollback
        - - State rollback attack is when the external environment can replay old checkpoints over and over
            - - The enclave component would not be able to detect a state rollback attack
            - - See figure *Rollback Attack on the Enclave*[4]
        - - Simplest way to avoid state rollback attacks is to be stateless
    - 5.1.4 Side-Channel Attack Mitigation
        - - Two main kinds of side channel that can cause information leakage
            - - Persistent smart contract storage access patterns and execution timing
            - - Microarchitectural side chanels in TEE realizations
- 5.2 Non-confidential Computation Runtime
    - - Similar to Ethereum
        - - No encryption of smart contract state
    - - Can use the same consensus APIs as confidential Para-Times
- 5.3 Smart Contract APIs
    - - ParaTimes provide their own interfaces to the smart contracts they host
    - - ParaTimes are free to provide bindings to smart contract APIs for arbitrary languages
- [1] https://wikiless.org/wiki/Trusted_execution_environment
- [2] https://developer.r3.com/blog/preventing-rollback-attacks-on-intel-sgx-using-conclave-sdk/
- [3] https://wikiless.org/wiki/Trusted_Computing?lang=en#Remote_attestation
- [4] https://developer.r3.com/blog/preventing-rollback-attacks-on-intel-sgx-using-conclave-sdk/

6 Key Managers

- Each confidential ParaTime has its own key manager, that is available to the TEE that's responsible for the execution of the smart contract.
- 6.1 ParaTime Key Manager
    - The design of the key managers is focused on simplicity, reliability, auditability, and easy implementation. However due to the design of the ParaTime, developers can either use the key management system of Oasis, or design their own key management system. The idea behind key management system is to have a master key, where additional keys that are bound to the different contracts can be derived.
    - 6.1.1 Application Splitting
        - The key management is split between the enclave and the enclave-external components (smart contracts / blockchain). The enclave can not handle input/output of the information processed, that is left to the external component side. The external component is trusted to not attack any side channels on the TEE. It is not trusted to manage the keys. The keys persists in operation through the enclave encryption key (the master key), but the encryption key itself is only accessible inside the TEE which, the external component can not directly access.
            - Reasons for this:
            - The external components, while trusted can not have full operational access in case of errors that may compromise the key encryption data.
            - The need to update the key material in the key manager is to ensure that contract state gets re-encrypted and remains secure. Hence why the keys are split, since one fetches the information, and the other secures it.
    - 6.1.2 Replication
        - To ensure fault tolerance all the key managers can be replicated and all the critical part it is run inside an attested enclave (SGX*) The enclave relies on the external components for all communication, but by using an SGX remote attestation, the identity of the replicated key, is verified and authenticated, which allows selective sharing of the master key inside the enclave with other keys. The key manager also decides which replica are authorised. This is done to prevent attacks on sidechannels of the enclave on ones own machine.
- 6.2 Potential Future Key Manager Features
    - What follows in 6.2 are as of the white paper's publishing, simply future proposals for additional features regarding key management.
    - 6.2.1 Access Controls and Rate Limiting
        - Refer to the formal Whitepaper both at 6.2.1 and at section 5.1.2 for more details. There is a concern of information leaks on nodes executing confidential ParaTimes. State encryption keys could only be visible to the compute node running the confidential smart contract and rate limited* so restarting a node would not give as much visibility to circumvent controls.
    - 6.2.2 Proactive Secret Sharing
        - Secret keys could be split using secret sharing techniques so that a compromise of a few key managers will not result in the compromise of the secret key. The logic for this remaining tentative here is because there will be more Paratime Confidential Compute nodes by the nature of how they work as opposed to key managers. Due to how these nodes and key managers pull data, it will be much easier to compromise via a side chain attack or other means a Paratime Node than a key manager.
    - 6.2.3 Smart Contract State Re-encryption
        - Confidential contracts could require re-keying to avoid adversaries with an existing key from observing traffic and understanding the result of new contract executions. This might decrease throughput of the contract though, and so requiring smart contracts to use ORAM* techniques may be preferable.

Terminology

Byzantine Fault Tolerance:
The Byzantine Generals' Problem is a logical dilemma of how a group of Byzantine generals may have communication problems when trying to agree on their next move.
The dilemma assumes that each general has its own army and that each is situated in different locations around the city they intend to attack. The generals need to agree on either attacking or retreating. It does not matter whether they attack or retreat, as long as all generals reach consensus, i.e., agree on a common decision in order to execute it. Each general has to decide: attack or retreat (yes or no); After the decision is made, it cannot be changed; All generals have to agree on the same decision and execute it in a synchronized manner.
The aforementioned communication problems are related to the fact that one general is only able to communicate with another through messages, which are forwarded by a courier. Consequently, the central challenge of the Byzantine Generals' Problem is that the messages can get somehow delayed, destroyed or lost.
In a few words, Byzantine fault tolerance (BFT) is the property of a system that is able to resist the class of failures derived from the Byzantine Generals' Problem. This means that a BFT system is able to continue operating even if some of the nodes fail or act maliciously.
https://academy.binance.com/en/articles/byzantine-fault-tolerance-explained

Concurrency:
The simultaneous occurrence of events or circumstances

Ethereum Virtual Machine (EVM):
The Ethereum Virtual Machine is a giant virtual machine that facilitates the deployment and execution of code. The EVM is often described as being Turing complete, which means that given enough time, memory, and necessary instructions, it can solve any computational task, no matter how complex.
The EVM allows developers to create decentralized applications on Ethereum. It also stores all the Ethereum accounts and smart contracts to simplify the retrieval of data and execution of the said contracts. Anyone can access the EVM from anywhere in the world through Ethereum nodes that lend computing power to the Ethereum network at a fee (gas).
https://medium.com/omniaprotocol/the-ethereum-virtual-machine-evm-what-is-it-and-how-does-it-work-e61fbe67996

Fully Homomorphic Encryption (FHE):
A form of encryption that permits users to perform computations on its encrypted data without first decrypting it. These resulting computations are left in an encrypted form which, when decrypted, result in an identical output to that produced had the operations been performed on the unencrypted data. Homomorphic encryption can be used for privacy-preserving outsourced storage and computation. This allows data to be encrypted and out-sourced to commercial cloud environments for processing, all while encrypted.
https://en.wikipedia.org/wiki/Homomorphic_encryption

Gossip Network:
A gossip protocol or epidemic protocol is a procedure or process of computer peer-to-peer communication that is based on the way epidemics spread.
https://en.wikipedia.org/wiki/Gossip_protocol

Key Managers:
A cryptographic key is defined as a string of data that is used to lock or unlock cryptographic functions. These functions include authentication, authorization and encryption.
Cryptographic key management generally refers to key management. It is basically defined as management of cryptographic keys that are used to deliver different purposes in a cryptographic network.
The basic cryptographic key management deals with the generation, exchange, storage, use, replacement and destruction of keys. The process involves cryptographic protocol design, key servers, user procedures, and other relevant protocols.
www.analyticssteps.com/blogs/what-cryptographic-key-management-and-how-it-done

Keystone:
An open sourced framework for building customizable TEE
https://arxiv.org/pdf/1907.10119.pdf

Modular:
Consisting of separate parts that, when combined, form a complete whole
https://dictionary.cambridge.org/us/dictionary/english/modular

MPC - Multiparty Computation:
Secure multi-party computation (also known as secure computation, multi-party computation (MPC) or privacy-preserving computation) is a subfield of cryptography with the goal of creating methods for parties to jointly compute a function over their inputs while keeping those inputs private. Unlike traditional cryptographic tasks, where cryptography assures security and integrity of communication or storage and the adversary is outside the system of participants (an eavesdropper on the sender and receiver), the cryptography in this model protects participants' privacy from each other.
https://en.wikipedia.org/wiki/Secure_multi-party_computation

Non-adaptive adversary:
The adversary can not take advantage of the information in the committee nodes to see if it controls the entire committee. It has to instead constantly have the node it has compromised return the same answer over and over again, but should the answer be correct, then it can deregister all the nodes in the system.
For more information see Appendix B in the whitepaper

ORAM (Oblivious Random Access Machine): *a compiler that transforms algorithms in such a way that the resulting algorithms preserve the input-output behavior of the original algorithm but the distribution of memory access pattern of the transformed algorithm is independent of the memory access pattern of the original algorithm.*
https://en.wikipedia.org/wiki/Oblivious_RAM

ParaTimes:
The Oasis Network is made up of two main layers: consensus layer and the ParaTime layer. These parts work together to create a powerful yet versatile blockchain. The Consensus Layer is a scalable, proof-of-stake consensus run by a decentralized set of validator nodes. ParaTimes connect to the Consensus Layer and periodically submit a record of their transactions to be stored by the validator nodes on the Ledger.
Each ParaTime is made up of set of compute nodes , which provide a computing environment for smart contracts that can be customized for a broad set of use cases. ParaTimes can be deployed by anyone on the Oasis Network.
This unique architecture allows transactions to be processed in parallel and for each ParaTime to be developed in isolation — allowing the Network to grow and adapt as the needs of the ecosystem change.
https://medium.com/oasis-protocol-project/the-oasis-eth-paratime-is-live-on-mainnet-33d8713ec870

Proof of Stake:
A modification of PoW (Proof of Work) as a means to solve its perceived dependency on energy consumption as a means to determine blockchain ordering.
Rather than rely on computers racing to generate the appropriate hash, the idea behind a PoS protocol is that participation is determined by ownership of the coin supply.
Using a set of factors determined by the protocol, the PoS algorithm pseudo-randomly elects a node (anyone who owns the coin) to propose the next block to the blockchain.
When a node gets elected, its role is to verify the validity of the transactions within the block, sign it and propose the block to the network for validation.
https://www.kraken.com/en-us/learn/proof-of-work-vs-proof-of-stake

Rate limiting:
A strategy for limiting network traffic. It puts a cap on how often someone can repeat an action within a certain timeframe – for instance, trying to log in to an account. Rate limiting can help stop certain kinds of malicious bot

activity. It can also reduce strain on web servers.
https://www.cloudflare.com/learning/bots/what-is-rate-limiting/

Replication:
Replication in computing involves sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

- *Computation replication, where the same computing task is executed many times. Computational tasks may be:*
    - *Replicated in space, where tasks are executed on separate devices*
    - *Replicated in time, where tasks are executed repeatedly on a single device*

https://en.wikipedia.org/wiki/Replication_(computing)

SGX (Software Guard Extension):
A trusted hardware that offers hardware-based memory encryption that isolates specific application code and data in memory. The SGX allows user-level code to allocate private regions of memory, called enclaves, which are designed to be protected from processes running at higher privilege levels.
https://en.wikipedia.org/wiki/Software_Guard_Extensions

Tendermint BFT consensus protocol:
an open-source blockchain protocol engine which developers can build upon with any programming language.
https://learn.bybit.com/blockchain/tendermint/

Verifiable Computing:
Verifiable computing (or verified computation or verified computing) enables a computer to offload the computation of some function, to other perhaps untrusted clients, while maintaining verifiable results. The other clients evaluate the function and return the result with a proof that the computation of the function was carried out correctly
en.wikipedia.org/wiki/Verifiable_computing

ZKP - Zero Knowledge Proof:
A zero-knowledge protocol is a method by which one party (the prover) can prove to another party (the verifier) that something is true, without revealing any information apart from the fact that this specific statement is true.
https://ethereum.org/en/zero-knowledge-proofs/