Summary on smart contract

1. Smart Contract

Smart contracts are an extension of traditional blockchains like Bitcoin. Unlike Bitcoin, which was designed primarily for peer-to-peer transactions, smart contracts enable programmable transactions and logic execution on the blockchain. They are stored publicly on the blockchain and can receive or transfer funds based on predefined conditions. To be effective, smart contracts must be designed with both **security** and **economic considerations** in mind, as counterparties are often incentivized to act in their own financial interest.

The motivation behind smart contracts comes from the limitations of Bitcoin's scripting language, which lacks user-friendliness and expressive power. Retrofitting Bitcoin's language to support complex operations would be cumbersome and costly. Instead, platforms like **Ethereum** were created to support smart contract programming directly through more expressive and developer-friendly languages such as **Solidity**.

A smart contract on Ethereum is a computer program that includes code, file storage, and an account balance. Once deployed, its code becomes immutable, though it can interact with users or other contracts through **entry points**, which function similarly to methods in traditional programming. The state of the contract is completely public. Executing a contract consumes a resource called **GAS**, and if the contract exceeds its GAS limit, execution is halted without refunding the consumed GAS.

Ethereum is currently the leading platform for smart contracts, supporting over 60 million accounts and 14.5 million contracts. These contracts are typically written in Solidity—a typed language inspired by C++, Python, and JavaScript—and compiled into **EVM (Ethereum Virtual Machine) bytecode** before being deployed via transactions. Contract data and code can be explored on platforms like Etherscan.io.

An analysis by Tonelli of 12,000 smart contracts found that software metrics such as size and complexity tend to show low variance, suggesting that smart contracts share many traits with traditional software systems. However, smart contracts are unique in their direct connection to blockchain activity and their visibility.

Smart contract usage indicators fall into two main categories: **blockchain interaction** and **developer interaction**. The first group includes metrics such as the number of transactions and transactions per day. For example, the **BAT token** contract has over 70 times the average number of transactions, and **EtherDelta** accounts for 20,000 times the average. Despite making up only 0.2% of contracts, these two account for over **61% of total transactions**, reflecting their high engagement levels. BAT's frequent transactions are explained by its link to the Brave browser, which rewards users for viewing ads.

The second category includes **developer activities**, such as the number of contract deployments, version updates, and code reuse. Analysis reveals that longer codebases tend

to be more complex, and that transaction frequency per day correlates moderately with size and complexity metrics, offering insights into how contract design affects usage.

2. Ekiden

The motivation behind Ekiden arises from a fundamental limitation of traditional smart contracts: they lack confidentiality. Since all data and computation are publicly visible on the blockchain, smart contracts cannot safely process sensitive information. While cryptographic approaches have been explored to address this issue, they often fall short in terms of performance. Trusted Execution Environments (TEEs) offer confidentiality and integrity guarantees, but naively combining TEEs with blockchain can introduce new vulnerabilities. Ekiden proposes a hybrid architecture that carefully integrates blockchain with TEEs to achieve both **privacy** and **performance**.

Ekiden's key innovation is the **separation of execution and consensus**. Smart contracts are executed **off-chain** inside TEEs (specifically Intel SGX enclaves), and the results of the execution—along with a cryptographic attestation—are published **on-chain**. This allows Ekiden to offload the expensive computation away from the main blockchain while still preserving integrity through attestations. As a result, the platform achieves **higher throughput** and protects the **confidentiality of data** during contract execution.

The technical challenges Ekiden addresses are substantial. First, **formal security modeling** is difficult due to the novel combination of TEEs and blockchain. Second, Ekiden must manage **TEE limitations**, such as availability failures (e.g., enclave crashes), side-channel attacks, and lack of trusted time sources. Solutions include making TEEs stateless (with persistent state stored on-chain), spatial and temporal separation of components, key rotation, and rejecting outdated information through blockchain validation. Third, **key management** is complex: since TEEs lack persistent storage, keys must be managed through a protocol that ensures confidentiality and forward secrecy using distributed key generation and pseudo-random functions.

Ekiden includes three main entities: **clients**, **compute nodes**, and **consensus nodes**. Clients initiate contracts and send them to compute nodes. Compute nodes execute contracts inside TEEs, generate encrypted new states and attestations, and send the results to consensus nodes for validation. Only when the consensus nodes accept the output and attestation is the result revealed to the client. This process ensures **atomic delivery**—ensuring that either both the client and blockchain receive valid output, or neither does.

Key components of Ekiden’s architecture include:

- **Proof of Publication**: Verifying that a message was truly published on-chain using timing-based confirmation without needing a trusted clock.

- **Key Management**: Short-lived keys are derived from long-term secrets shared across a distributed key management committee. This ensures confidentiality and limits the impact of a single key compromise.

- **Atomic Delivery Protocol**: Ensures that outputs are released to clients only if the corresponding state update is confirmed on the blockchain, preventing leakage of sensitive results in case of failure.

In summary, Ekiden enables rich, confidential smart contracts by combining blockchain consensus with off-chain execution in TEEs. It addresses major challenges in privacy, correctness, and secure key handling, making it a strong candidate for privacy-preserving decentralized applications.