Lifecycle of a Smart Contract

Table of Content

Smart contracts are one of the most fundamental building blocks of blockchain technology or distributed ledger technology. The term 'smart contract' was first introduced by computer scientist and cryptographer Nick Szabo in the 1990s. In his publication, Szabo defined smart contract as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises."

As blockchains gained in popularity, smart contracts began to flourish — especially with the emergence of Ethereum, one of the most popular blockchains that supports smart contracts. These self-executing contracts automatically enforce the rules and agreements encoded within them, making them ideal for a wide range of use cases, from supply chain management to financial instruments.

Smart contracts can enhance blockchain security in several ways. First, they can eliminate the need for third-party intermediaries, such as banks, lawyers, or escrow agents, who may be vulnerable to fraud, corruption, or human error. Second, they can ensure that the transactions and interactions on the blockchain are immutable, verifiable, and traceable, as they are recorded and validated by the network. Third, they can enable more advanced and customized security features, such as multi-signature verification, time locks, oracles, or encryption.

What Are Smart Contracts?

A smart contract is an agreement between two people or entities in the form of computer code programmed to execute automatically. A simple example could be in the case of life insurance. The policy terms would be encoded into the smart contract. In the event of a passing, the notarized death certificate would be provided as the input trigger for the smart contract to release the payment to the named beneficiaries.

Smart Contracts simply defined: Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met. Smart contracts are executed on blockchain, which means that the terms are stored in a distributed database and cannot be changed.

History of Smart Contracts

Smart contracts were first proposed in 1994 by Nick Szabo, an American computer scientist who invented a virtual currency called "Bit Gold" in 1998, 10 years before Bitcoin was introduced. In fact, Szabo is often rumoured to be the real Satoshi Nakamoto, the anonymous

Bitcoin inventor, which he has denied. Szabo defined smart contracts as computerized transaction protocols that execute the terms of a contract. He wanted to extend the functionality of electronic transaction methods, such as POS (point of sale), to the digital realm.

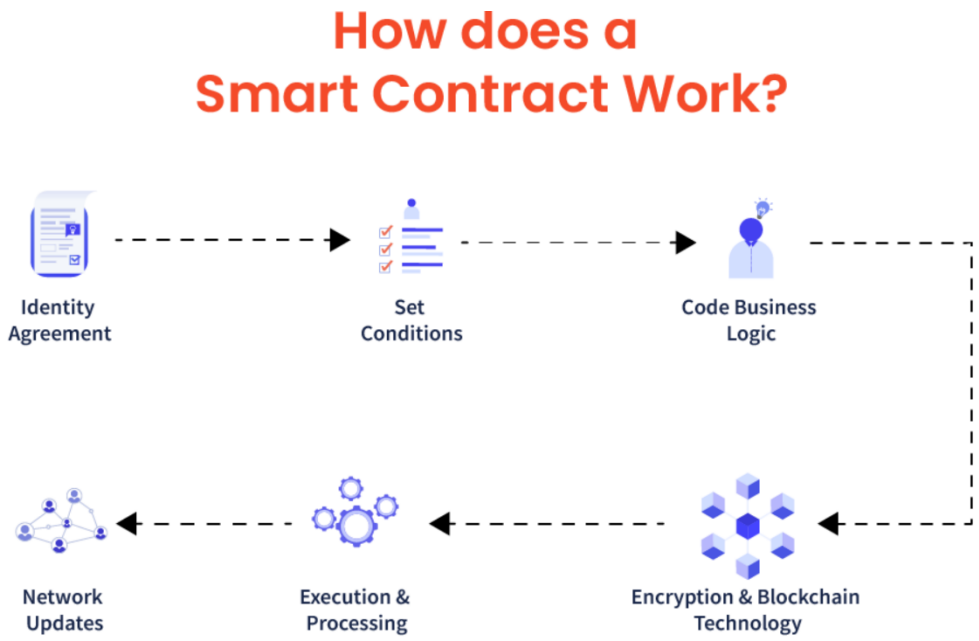
Szabo famously compared a smart contract to a vending machine. Imagine a machine that sells cans of soda for a quarter. If you put a dollar into the machine and select a soda, the machine is hardwired to either produce your drink and 75 cents in change, or (if your choice is sold out) to prompt you to make another selection or get your dollar back. This is an example of a simple smart contract. Just like a soda machine can automate a sale without a human intermediary, smart contracts can automate virtually any kind of exchange.

In his paper, Szabo also proposed the execution of a contract for synthetic assets, such as derivatives and bonds. Szabo wrote, "These new securities are formed by combining securities (such as bonds) and derivatives (options and futures) in a wide variety of ways. Very complex term structures for payments...can now be built into standardized contracts and traded with low transaction costs, due to computerized analysis of these complex term structures."

Many of Szabo's predictions in the paper came true in ways preceding blockchain technology. For example, derivatives trading is now mostly conducted through computer networks using complex term structures.

Smart contracts today also find their origin in Ricardian Contracts, a concept published in 1996 by Ian Grigg and Gary Howland as part of their work on the Ricardo payment system to transfer assets. Grigg saw Ricardian Contracts as a bridge between text contracts and code that had the following parameters: a single document that “is

i. a contract offered by an issuer to holders,
ii. for a valuable right held by holders, and managed by the issuer,
iii. easily readable by people (like a contract on paper),
iv. readable by programs (parsable like a database),
v. digitally signed,
vi. carries the keys and server information, and
vii. allied with a unique and secure identifier.”



- **Identify Agreement:** Multiple parties identify the cooperative opportunity and desired outcomes and agreements could include business processes, asset swaps, etc.
- **Set Conditions:** Smart contracts could be initiated by parties themselves or when certain conditions are met like financial market indices, events like GPS locations, etc.
- **Code Business Logic**: A computer program is written that will be executed automatically when the conditional parameters are met.
- **Encryption and Blockchain Technology**: Encryption provides secure authentication and transfer of messages between parties relating to smart contracts.
- **Execution and Processing**: In blockchain iteration, whenever consensus is reached between the parties regarding authentication and verification then the code is executed and the outcomes are memorialized for compliance and verification.
- **Network Updates**: After smart contracts are executed, all the nodes on the network update their ledger to reflect the new state. Once the record is posted and verified on the blockchain network, it cannot be modified, it is in append mode only.

How are Smart Contracts created on Ethereum?

Ethereum provides a decentralized virtual computer - the Ethereum Virtual Machine (EVM) - on which developers can build applications consisting of multiple smart contracts. Think of the EVM as a distributed global computer where all smart contracts are executed. (Ethereum is sometimes referred to as a "world computer.") Ethereum lets developers program their own smart contracts to define EVM instructions. The EVM executes a contract according to the rules the developer programmed. Smart contracts live in the form of bytecode within the decentralized database. This is the root of the innovation and disruptive potential of Ethereum. Contracts are written in Solidity, a Javascript-like language developed specifically for writing smart contracts.

The Ethereum white paper explains it:

The intent of Ethereum is to create an alternative protocol for building decentralized applications, providing a different set of tradeoffs that we believe will be very useful for a large class of decentralized applications […] Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions."

Paying with Gas

Whenever smart contract transactions occur within the Ethereum blockchain, it takes computational power to validate them across the network. To compensate for the time and energy it takes to perform those computations, transactions require a fee in the form of "gas" which is paid with ETH. The amount of gas depends on the amount of computation required to complete the transaction. This gives developers an incentive to create simple and efficient smart contracts. Moreover, the gas limit — the maximum amount of gas allotted to a particular transaction — means buggy code won't run indefinitely, costing an infinite amount of gas.

(Unspent gas is automatically refunded, but if the transaction runs out of gas, it's aborted and no gas is refunded.)

The Technology behind Smart Contracts

The Ethereum blockchain powers the technology underneath. This decentralised and distributed ledger securely records transactions and data. Smart contracts are self-executing computer programs that run on the Ethereum blockchain and enforce the terms of an agreement automatically. Developers write these contracts in a high-level programming language and compile them into low-level bytecode, which the Ethereum blockchain stores. The Ethereum Virtual Machine, a computer network that runs the Ethereum blockchain, executes the bytecode. When someone makes a transaction on the Ethereum blockchain, it triggers the smart contract to run and enforce the agreement's terms. The decentralised and distributed nature of the ledger ensures the security and transparency of the agreement's terms, as multiple computers store the transaction details, and anyone can audit them. By using smart contracts, individuals can automate various agreements and transactions, reducing the risk of fraud and the need for intermediaries.

Companies Using Ethereum Smart Contracts

Many companies have adopted Ethereum smart contracts to provide secure and efficient solutions for their business processes. Some of the companies using Ethereum smart contracts include:

- Microsoft: Microsoft has adopted Ethereum smart contracts to provide a secure and transparent platform for managing the supply chain of its products.
- JPMorgan Chase: JPMorgan Chase is using them to increase the efficiency and security of its cross-border payments.
- Accenture: Accenture uses them to provide secure and transparent solutions for its clients' supply chains.

Phases of Smart Contract

Development Phase:

In the developmental genesis of a smart contract, developers wield specialized tools akin to architects' instruments. They utilize Solidity, a dedicated programming language for Ethereum smart contracts, within sophisticated Integrated Development Environments (IDEs) like Remix or Truffle. These platforms act as the virtual drafting tables, allowing programmers to sculpt the contract's code and ensure its logic resonates with the envisioned functionality.

To fortify against potential vulnerabilities, developers rely on security analysis tools like MythX or Securify, akin to quality control inspectors ensuring the structural integrity of a building's blueprint. These tools serve as vigilant guardians, identifying and mitigating security loopholes before they metamorphose into critical issues, ensuring the solidity of the contract's foundation.

Deployment Phase:

As the Development Phase culminates, the smart contract ventures into the Deployment Phase, much like a newly constructed building being inaugurated. Ethereum clients such as MetaMask or Geth serve as the bridge, facilitating the deployment transaction and granting the contract its unique address on the chosen blockchain network. These tools act as the keys unlocking the gateway to the decentralized world, akin to the grand opening of a physical establishment.

The deployment process relies on Ethereum clients' functionalities, enabling developers to execute transactions and manage contracts seamlessly. Similar to a building's inauguration ceremony, this deployment marks the contract's entrance into the digital landscape, ready to materialize real-world applications—be it facilitating decentralized exchanges, managing supply chains, or encoding ownership rights as non-fungible tokens (NFTs).

Maintenance Phase:

In the Maintenance Phase, akin to the perpetual stewardship of a structure, developers employ a suite of tools to ensure the smart contract's ongoing reliability and adaptability. Bug tracking tools such as GitHub or Jira act as vigilant overseers, documenting and resolving reported issues—paralleling the maintenance crew identifying and rectifying structural faults in a building to ensure its longevity.

For seamless updates while preserving existing functionalities, developers utilize update implementation tools like OpenZeppelin. These tools provide the scaffolding necessary to execute upgrades, much like renovation crews maintaining and enhancing building features to align with evolving needs. Gas optimization tools like Solidity Gas Profiler analyze and refine the contract's code, akin to fine-tuning machinery to enhance efficiency and reduce operational costs, ensuring the contract remains viable and cost-effective.

Testing Phase:

The Testing Phase subjects smart contracts to rigorous evaluations, akin to stress-testing structures for durability and reliability. Developers employ a suite of testing tools—unit testing frameworks like Truffle or Hardhat dissect individual components, ensuring they operate flawlessly. Integration testing tools like Ganache or Waffle simulate real-world interactions, scrutinizing the contract's harmony—resembling the integration of different building systems for seamless functionality.

Engaging third-party security auditing services such as Quantstamp or Certik acts as expert inspectors, evaluating the contract's fortification against potential threats. These services conduct comprehensive analyses, probing for vulnerabilities and ensuring the contract's resilience against adversities. This phase epitomizes resilience testing, ensuring the contract emerges stronger and more reliable, aligning with real-life scenarios where robust systems endure challenges and emerge fortified.