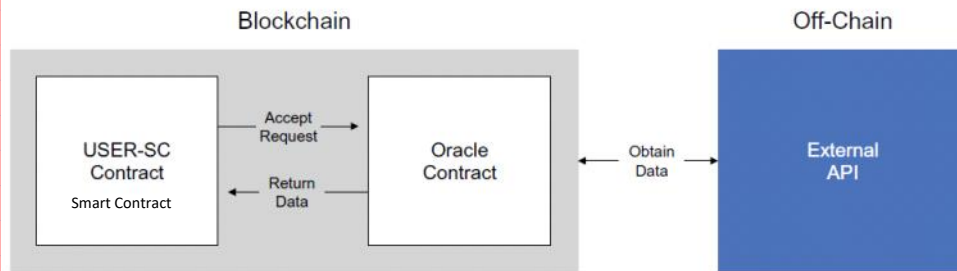


Oracle anatomy

4.4.1 Introduction

The network participants (nodes) validate and execute operations performed on the blockchain, such as smart contracts, but it is not uncommon for a smart contract to require data from external third parties. Given that blockchains cannot access data outside their networks, how is external data incorporated into the workflow? Here is where Oracles come in.



<code>pragma solidity ^0.5.1;</code>	
<code>contract Coin</code> <code>{</code>	<i># Coin toss contract.</i> <i>Allows two bettors to bet on a predefined amount.</i>
<code>uint256 amount;</code> <code>uint256 blockNumber;</code> <code>address payable[] bettors;</code>	<i># Contract owner</i>
<code>constructor(uint256 amount) public</code> <code>{</code> <code> amount = amount_;</code> <code>}</code>	<i># Creates the contract.</i> <i>@param amount_ the bet amount, in Wei.</i>
<code>function bet() payable public</code> <code>{</code> <code> require(msg.value == amount);</code> <code> require(bettors.length < 2);</code> <code> blockNumber = block.number + 1;</code> <code> bettors.push(msg.sender);</code> <code>}</code>	<i># Places a bet.</i>
<code>function toss()public</code> <code>{</code> <code> require(bettors.length == 2);</code> <code> require(blockNumber < block.number);</code> <code> uint256 winner = uint256(blockhash(block.number)) % 2;</code> <code> bettors[winner].transfer(address(this).balance);</code> <code>}</code>	<i># Tosses the coin and pays the winner.</i>
<code>}</code>	

Fig. 4.6 Coin toss example (solidity)

Oracles are agents on the blockchain that can verify events from the real world and then provide the corresponding information as data to the smart contracts (Fig. 4.7). Running on the blockchain, Oracles act as intermediaries between external data and smart contracts that also run on the blockchain [9]. Smart contracts usually specify only sequences like “If condition **C** has occurred, operation **O** will be executed.” To check whether the condition has been met, smart contracts often rely on external data. One example is sensors that measure real-world phenomena. For example, if you want to implement insurance through a smart contract

IoT

For example, if you want to implement insurance through a smart contract that pays direct compensation when certain temperatures are reached, an Oracle must provide the temperature information.

4.4.2 Smart Contract Integration

The addition of an Oracle occurs through multi-signature (**MultiSig**) contracts, which require that an invocation of a smart contract method be signed by multiple parties.

To improve trust in the external data a smart contract that an Oracle provides and to prevent the Oracle from being compromised by one party, data can be signed by multiple parties: In the case of a stock price Oracle, one can set up MultiSig so each stock price data item must be signed by three independent parties (e.g., Bloomberg, NASDAQ, and MSNBC).

4.4.3 Oracles and Security *POW; POS; POA.*

The basic principle of blockchain is that a consensus is reached through the involvement of multiple parties, eliminating the need to trust a single party.

However, if the data is provided by a central authority like a bank, the data-providing authority must be trusted.

4.4.4 Types of Oracles

Data sources for smart contract Oracles can be classified into five types:

- **Software Oracle:** Data is available online and is, ideally, provided from multiple independent sources, has a public record history (e.g., air traffic information, meteorological data), and is signed by multiple parties.

- **Hardware Oracle:** Data is from real-world measurements like RFID sensors in a supply chain site. Challenges with this type of data source include that there is usually only one source, there is no public record history, it is signed by only one party, and transmitting the information in a secure and provably immutable way is difficult.

From a technical perspective, Oracles can also be classified by their functional setup—that is, based on data flow (inbound vs. outbound) and whether they are consensus-based or not. This delineation is not mutually exclusive, so some data can be both inbound and outbound.

- **Inbound Oracles:** These Oracles provide a smart contract with information from the outside world. For example, a buy order is set to be executed as soon as the EUR-USD exchange rate falls below a certain limit.

- **Outbound Oracles:** These Oracles enable smart contracts to send data to the outside instead of just receiving it. For example, access to an area may be granted when a payment has been made on the blockchain (e.g., via a smart lock).

- **Consensus-based Oracles:** Several Oracles are combined so one does not have to rely on a single external source. These Oracles then form a consensus to make decisions. An example is a consensus Oracle that specifies that three out of five Oracles must agree before an operation is executed. Of course, it is also possible to apply scores to individual Oracles such that a source carries more weight if it appears to be more reliable than other sources.

$$\begin{aligned} C_{i1} \cdot C_{i2} \cdot C_{i3} &= \\ &= C_{e1} \cdot C_{e2} \\ &\quad \downarrow \\ i_1 + i_2 + i_3 &= \\ &= e_1 + e_2 \end{aligned}$$

4.4.5 Oracle Contract Example

The Oracle contract provides a simple working example of an Oracle smart contract written in the Solidity programming language (Fig. 4.8). In this example, the Oracle can provide the USD-EUR foreign exchange (FX) rate to other smart contracts on a blockchain: The contract specifies the owner (i.e., a single-signature Oracle), the timestamp input (to determine when the FX rate can be returned), and the FX rate itself, which other smart contracts can access.

<code>pragma solidity ^0.5.1;</code>	
<code>contract Oracle</code> <code>{</code>	<i># Oracle contract. A single-signature inbound Oracle for the USD-EUR exchange rate.</i>
<code>address payable public owner;</code>	<i># Oracle owner</i>
<code>uint8 public decimals;</code>	<i># USD-EUR exchange rate</i>
<code>uint256 public timestamp;</code>	<i># USD-EUR exchange rate timestamp</i>
<code>constructor() public</code> <code>{</code> <code>owner = msg.sender;</code> <code>}</code>	<i># Creates the contract</i>
<code>function set(uint8 decimals_, uint256 rate_) public</code> <code>{</code> <code>require(msg.sender == owner);</code> <code>require(rate > 0);</code> <code>decimals = decimals_;</code> <code>rate = rate_;</code> <code>timestamp = block.timestamp;</code> <code>}</code>	<i># Sets the new exchange rate</i> <i>* @param decimals_ the USD-EUR exchange rate decimal position</i> <i>* @param rate_ the USD- EUR exchange rate</i>
<code>}</code>	

Tokens

ERC-20

ERC-20 is the most popular Ethereum blockchain technical standard for tokens that are issued on Ethereum. It was proposed by Fabian Vogelsteller on November 19, 2015. ERC stands for Ethereum Request for Comment, and 20 is a unique ID number for the request to differentiate it from other standards. Currently, there are more than 65,000 ERC-20 tokens in existence, although many of these tokens have no market value.

RFC

The ERC-20 token defines the following common list of rules in the smart contract:

```
contract ERC20Interface {
function totalSupply() public view returns (uint);
function balanceOf(address tokenOwner) public view returns (uint balance);
function allowance(address tokenOwner, address spender) public view returns (uint remaining);
function transfer(address to, uint tokens) public returns (bool success);
function approve(address spender, uint tokens) public returns (bool success);
function transferFrom(address from, address to, uint tokens) public returns (bool success);
event Transfer(address indexed from, address indexed to, uint tokens);
event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

The relevant functions are listed as follows:

totalSupply(): Returns the total token supply.

balanceOf(): Gets the balance of the account for the given address.

allowance(): Returns the allowance amount from `_owner`.

transfer(): Transfers the balance from the owner's account to another account and must fire the transfer event.

transferFrom(): Sends the amount of tokens from address from to address to.

//The transferFrom() method is used to withdraw the workflow, allowing contracts to transfer tokens on your behalf.

approve(): Allows the spender to withdraw from your account with a certain amount.

Before the ERC token standard, the numerous initial coin offerings (**ICOs**) start-ups or Dapps had created their own tokens with many different standards.

After the release of the ERC-20 standard, things changed and have become much more streamlined.

The benefits of ERC-20 tokens include the following:

- 1.Reduces the risk of contract breaking
- 2.Reduces the complexity of token interactions
- 3.Uniform and quicker transactions
- 3.Confirms transactions more efficiently
- 4.Enhances token liquidity

However, ERC-20 tokens do not have regulatory built-in limitations by design, so there are no restrictions for transferring them. This kind of token is known as a utility token. When dealing with security regulations, token holders must apply for **know your customer (KYC)**; **anti-money laundering (AML)** verification processes; token trading is subject to federal security regulations and many additional constraints apply to these regulations.

Since the ERC-20 token is widely adopted in the industry, currently most of the security token ERC standard proposals are **ERC-20-compatible**, which means that potentially all **wallets** and **exchanges supporting ERC-20** will support these tokens as well.

 e. brokers

In ERC-20 standards, when a user trades these tokens, all of the token supply *can be treated* as the *same value*; it is interchangeable. The ERC-20 token is a **fungible token (FT)**. An FT is a core characteristic of cryptocurrencies.

It can be easily replaced by something identical and it is interchangeable with ease, like bitcoin.

On the other hand, a **non-fungible token (NFT)** is a special type of cryptographic token that has unique information or attributes. NFTs are thus irreplaceable or not interchangeable. For this purpose, the ERC-721 token was created as a standard.

Dieter Shirley, the creator of CryptoKitties, developed the term NFT.

Each token within the contract represents a different value.

Security token technical design overview.

Security tokens, like ERC-20 tokens with built-in regulatory restrictions,

protect investors' rights at the token level. It redefines the whole process through which companies or startups raise money. As of today, many different standards are being proposed and implemented in the marketplace. In this section, we'll look into the following Ethereum security token standards:

ERC-1400/ERC-1410

ST-20


R-Token

SRC-20

DS-Token

ERC-1404

ERC-884



Till this place