Lecture Notes

Smart Contract Development in Ethereum

Tokens in Ethereum

In this session, you will learn about the following:

- Difference between cryptocurrencies and tokens
- Motivation behind creating standards for tokens
- ERC 20 standard
- ERC 777 standard
- Asset digitisation and types of tokens
- ERC 721 standard

Tokens vs Cryptocurrency

Cryptocurrency can be seen as the native currency of any blockchain network. For example, Bitcoin's cryptocurrency is called Bitcoin or BTC and Ethereum's cryptocurrency is Ether.

- In Ethereum, the basic currency that is used to perform transactions is called Ether.
- You can create a sub-ecosystem within the outer blockchain network, which can have its own currency. This currency created in the sub-ecosystem can be called a token.
- Example: Suppose Amazon has given you a voucher worth ₹100. You can use this voucher to buy anything, but you can use it only on the Amazon website. This voucher can be considered a token within the Amazon ecosystem.

- Similarly, in a blockchain network like Ethereum, a token can be built on top of a cryptocurrency (like Ether) for a specific functionality or requirement.

Motivation Behind Standards

Try to visualise a situation where one application uses its defined token, and another uses a different definition. When the two applications interact with each other, there will be a need to convert each other's tokens to a common format.
This was about only two parties. What if there are hundreds of such applications? Imagine the conversion overhead if the applications want to interact with each other.
To overcome such situations, the Ethereum community created certain standards.
These standards are called ERC standards.
Let’s understand the need for ERC standards. They are as follows:

- ERC stands for Ethereum Request for Comments.
- These are application-level comments or standards defined for the token.
- Anyone can create an ERC. However, the ERC’s author must include a clear explanation of its standard.
- Applications can use this common ERC token and make it easier to interact with each other.
- The conditions of the standard can include anything; for example, the way it will interact with a smart contract.
- Some of the common ERC standards are ERC 20, ERC 721 and ERC 777. We will look at one or two of these in detail in the following segments.

ERC 20

ERC 20

- ERC 20 is an asset created on Ethereum and is written in Solidity.
- It is hosted on the Ethereum blockchain.
- ERC 20 tokens are stored and sent using Ethereum addresses and transactions.
- This standard has six functions – totalSupply, balanceOf, transfer, transferFrom, approve and allowance. We will learn these in detail soon.
- These functions can be used to transfer tokens to an address or a smart contract.

```
ERC 20 interface:

//core ERC20 functions

/* The totalSupply function is run only once at the
time of the deployment of the token. It defines the
total supply of tokens which  can be circulated at
the very start. The uint it returns can either be
funded from a wallet or by hardcoding a variable. */
 function totalSupply() constant returns (uint
totalSupply);


/* The balanceOf function accepts the owner address
and returns the balance of that address. */
 function balanceOf(address _owner) constant returns
(uint balance);

/* The transfer function is one of the main functions
of the ERC 20 interface. This function transfers
tokens from one account (_to) to another and returns
the status of the transfer as a boolean value. Actual
wallet owners can call this function to transfer
tokens */
 function transfer(address _to, uint _value) returns
(bool success);


/* The allowance function allows you to perform an
exchange between two accounts. The spender is the
entity that will ask for 'x' tokens from the owner.
*/
```

```
 function allowance(address _owner, address _spender)
constant        returns (uint remaining);


/* The approve function works in tandem with the
allowance function. It determines whether the spender
is approved to withdraw tokens from the owner or not.
It returns a boolean value indicating whether or not
the spender is approved. */
 function approve(address _spender, uint _value)
returns (bool success);


/* The transferFrom function transfers the value
amount of tokens from one account to another. It
returns a boolean value indicating whether the
transaction was successful or not. */
 function transferFrom(address _from, address _to,
uint _value) returns (bool success);



//Other than functions, there are certain events in
the standard.

/* This event is supposed to be emitted whenever an
approval is made. */
 event Approval(address indexed _owner, address
indexed _spender,    uint _value);


/* This event is supposed to be emitted whenever a
transfer is done. */
 event Transfer(address indexed _from, address
indexed _to, uint    _value);
```

ERC 777

What is the need for ERC 777?

Let's try to answer the following question:

What if only 1,000 of 5,000 tokens are sold as a result of a low demand for that token?

Or

What if there was a huge demand for an extra 2,000 tokens?

Such situations can be handled in the following ways:

- Low demand

    - This is when the sale of XYZ tokens is low; for example, only 1,000 out of 5,000 tokens are sold.
    - This can lower the value of that token.
    - To cater to this demand, you can cut down the supply of tokens by burning the existing tokens.
- High demand
    - Similarly, what do you do when the demand for XYZ tokens is raised from 5,000 to 9,000?
    - In order to fulfil this demand, ERC 777 provides a functionality of minting new tokens when their demand is high.

Let's go through the extra two functionalities of ERC 777 in detail.

```
ERC 777:

/* The burn function decreases the supply of the
tokens by a particular value that is taken as an
input to the function. It returns the status of the
operation as a boolean variable. */
function burn( uint256 _value) public returns ( bool
_success);

/* The mint function increases the total supply of
tokens by a particular value that is taken as an
input to the function. It returns the status of the
operation as a boolean variable. Tokens are minted to
a particular address that is also taken as an input.
*/
function mint( address _to, uint256 _value ) public
returns ( bool _success );

//The other functions are the same as the ERC 20
standard.
```

Fungible and Non-Fungible Tokens

Fungible tokens

- The first example of cryptocurrency was Bitcoin.
- Bitcoin was used as a virtual currency with features similar to the real currency.
- Fungible tokens are a way to tokenize cryptocurrencies on top of the blockchain network.
- In fact, fungibility is an essential feature of any cryptocurrency.
- Some of the examples of fungible token standards are ERC 20, ERC 777 and ERC 223.

Non-fungible token

- If we take out fungibility out of cryptocurrency, we can see that the addition of multiple features to a token is a very huge possibility.
- Non-fungible tokens are digital representations of some unique features like a real estate asset, voting rights or your digital identity like Aadhaar.
- CryptoKitties was the first implementation of non-fungible tokens. It is a game on Ethereum where people can collect and breed digital cats. Each cat's digital genetic material is stored on the blockchain. This is just one use case of non-fungible tokens.
- Non-fungible tokens can be used for many applications like digital know your customer (KYC) and supply chain tracking of any product.

The difference between fungible and non-fungible can be summarised as follows:

Fungible Tokens	**Non-Fungible Tokens**
• These are tokens with identical properties. • One token can be interchanged with another token of the same value. • Tokens are divisible into smaller amounts as long as they add up to the same	• These are tokens with unique properties. Even if two tokens are of the same type, they are unique. • No two tokens can be interchanged as they all have unique specifications. • As these tokens can

value. ● Example: ERC 20, ERC 777, etc.	represent an identity, they are not divisible into smaller units. ● Example: A certificate or a degree. A degree cannot be divided into a fraction of a degree, and each degree is unique. ● A very popular implementation of a non-fungible token can be CryptoKitties.

ERC 721

You learnt about the major differences between fungible and non-fungible tokens. Let's understand how asset tokenization is done on the blockchain network using the ERC 721 token standard.

- Every non-fungible token (like ERC 721) has a metadata associated with it which defines the token.
- This metadata is unique for each token. For example, suppose you have a land registry document set. Each document represents one token.
- Creating a unique set of tokens is also known as tokenizing assets on the blockchain network where each document can represent an asset with certain properties.
- There are many assets that you can create on a blockchain network like Ethereum. For example, a document verification application where each document represents a token and has a unique set of values against it. You can create games where each character can be represented as a token and has a set of functionalities associated with it.

ERC 721 is an example of a non-fungible token using which we can create assets on the Ethereum blockchain network.
The interface of ERC 721 looks like as follows:

ERC 721:

```
//Logging events

/* The Transfer event transfers a token from one
address to another as mentioned in the input
parameters. The _tokenId is the identity of the token
to be transferred as every non-fungible token is
unique. */
   event Transfer(address indexed _from, address
indexed _to, uint256 indexed _tokenId);

/* The Approval event provides approval on whether
the token can be sent from the owner to the approved
receiver. The id of the token is also taken as an
input parameter. */
    event Approval(address indexed _owner, address
indexed _approved, uint256 indexed _tokenId);


/* The ApprovalForAll event sets an approval for not
only one token id but for all the tokens from a
specific owner. This can be set by an operator as
mentioned in the input parameter. */
    event ApprovalForAll(address indexed _owner,
address indexed _operator, bool _approved);

// Methods

/* The balanceOf method returns the token balance of
any owner address. */
   function balanceOf(address _owner) external view
returns (uint256);

/* The ownerOf function returns the owner address of
a particular token id. */
    function ownerOf(uint256 _tokenId) external view
returns (address);

/* The safeTransferFrom function is used to transfer
a token from one address to another. The token id
```

```
mentions the particular token you want to transfer.
*/
    function transferFrom(address _from, address _to,
uint256 _tokenId) external payable;

/* The approve function approves whether the token
can be transferred or not. */
    function approve(address _approved, uint256
_tokenId) external payable;

/* Similar to the event ApprovalForAll, the
setApprovalForAll function sets the approval status
of a set of tokens owned by a particular address. */
    function setApprovalForAll(address _operator,
bool _approved) external;

/* The getApproved function returns the address of
the approved token. This token is sent as an input to
the function. It returns the address that has
approved of the token id */
    function getApproved(uint256 _tokenId) external
view returns (address);

/* The isApprovedForAll function returns a boolean
indicating whether an operator is approved by the
given owner or not. */
    function isApprovedForAll(address _owner, address
_operator) external view returns (bool);
```