# BlockChain 1: Installation of MetaMask and study spending Ether per transaction.

# BlockChain 2: Create your own wallet using Metamask for crypto transactions.

---

**Prerequisite:**

**1. What is Blockchain?**

A blockchain is a **distributed and decentralized digital ledger** that records transactions across a network of computers (nodes) in an immutable and transparent way. It consists of a chain of blocks, where each block contains a set of validated transactions and a cryptographic hash of the previous block.

**2. How does a blockchain work, step-by-step?**

1. **Transaction:** A user initiates a transaction (e.g., sending crypto).
2. **Broadcast:** The transaction is broadcast to a network of computers, known as nodes.
3. **Validation:** Nodes in the network check the transaction's validity (e.g., does the sender have enough funds?).
4. **Block Creation:** Valid transactions are bundled together into a new "block."
5. **Chain:** This new block is cryptographically linked to the previous block, creating a secure "chain."
6. **Distribution:** The newly added block is copied across the entire network, so every node has an identical, updated version of the ledger.

**3. What are the core properties of Blockchain?**

- **Decentralization**: No central authority; all nodes share a copy of the ledger.
- **Immutability**: Once data is recorded, it cannot be altered without consensus.
- **Transparency**: All transactions are publicly verifiable (based on blockchain type).
- **Security**: Uses cryptographic functions and consensus protocols for data integrity.

**4. What are hashing functions and their types?**

Hashing functions take input data and convert it into a fixed-size string of characters (the hash). Properties:

- **Deterministic**: Same input always yields the same hash.
- **Collision-resistant**: Unlikely for two different inputs to produce the same hash.
- **Irreversible**: Cannot derive input from output hash.

Examples:

- **SHA-256** (Used in Bitcoin)
- **Keccak-256** (Used in Ethereum)
- **RIPEMD-160**

**5. How does Blockchain achieve immutability?**

Blockchain uses **cryptographic hashes** to link blocks together. If someone tries to alter a block's data:

- The block's hash changes dramatically due to the **avalanche effect**.

- This breaks the link to the next block and every block after it.
- Fixing this would require recalculating the hash for **all subsequent blocks**, which is computationally infeasible without majority control over the network.

if someone tried to alter a transaction in Block 100, the hash of Block 100 would completely change. This new hash would no longer match the "previous hash" stored in Block 101, which would break the entire chain from that point forward. The rest of the network nodes would immediately reject this tampered chain as invalid.

**6. What is the Avalanche Effect?**

The avalanche effect refers to a property of cryptographic algorithms where a **small change in input** (e.g., modifying a single character) results in a **completely different output hash**. This is critical for blockchain security, ensuring tampering is easily detectable.

**7. What are Nodes or Validators in Blockchain?**

**Node:** A **node** is a computer or device participating in the blockchain network. It runs the blockchain's software (like **Geth** for Ethereum) and holds a full or partial copy of the ledger. Its job is to receive, validate, and broadcast transactions and blocks to other nodes.

**Validator:** A **validator** is a special type of node that has the authority to *create* new blocks and add them to the chain. In a Proof-of-Stake system (like modern Ethereum), validators are nodes that have staked (locked up) ETH as collateral to participate in this process. In Proof-of-Work systems (like Bitcoin), these are called "miners."

**8. What is a Nonce and how is it used?**

A **nonce** stands for "**Number used once**."

- Its primary use is in **Proof-of-Work (PoW)** mining.
- Miners must find a specific nonce value. When this nonce is combined with all the other data in a block and then hashed, the resulting hash must be below a certain target value (e.g., start with 10 zeros).
- Finding this nonce is a difficult trial-and-error process that requires a lot of computational power. The first miner to find the correct nonce "wins" the right to add their block to the chain and receive a reward.

**9. How do we connect to a Blockchain Network?**

Blockchain networks don't use standard web (Web2) protocols like HTTP by default. To interact:

- We use **JSON-RPC (JavaScript Object Notation – Remote Procedure Call)**, which allows sending commands to blockchain nodes programmatically.
- Tools like **MetaMask**, **Infura**, or **Alchemy** provide easy access through predefined endpoints instead of complex direct RPC connections.
- Default browsers cannot connect to blockchain without middleware like MetaMask or Web3 libraries.

**10. What are Endpoint Providers in Blockchain?**

Endpoint providers are services that let developers connect to blockchain nodes without running their own full node. Examples:

- **Infura**
- **Alchemy**
- **Ankr**
- **QuickNode**

They provide APIs and WebSocket endpoints to communicate with blockchain networks (mainnet and testnets).

**11. What is Asymmetric Key Cryptography?**

Asymmetric key cryptography (or public-key cryptography) uses **two mathematically related keys**: a public key and a private key. The public key can be shared openly, while the private key must be kept secret. This method is widely used in blockchain for secure transaction signing and wallet ownership.

**12. What are Public and Private Keys in Blockchain?**

- **Public Key**: Derived from the private key; acts as an address for receiving funds. It is safe to share.
- **Private Key**: A confidential alphanumeric string used to authorize transactions and prove ownership of assets in a blockchain account. It must never be shared or lost.

13. **What is the Role of the Private Key?**

The private key is the **only way** to access and control assets in a blockchain wallet. If lost, the wallet cannot be recovered; if stolen, your assets can be moved by the attacker. Blockchain systems do not have "password reset" options.

14. **What is a Web3 or Digital Wallet?**

A Web3 wallet is an application (software, browser extension, or hardware device) that:

- Stores and manages your **private keys** securely.
- Enables users to sign transactions, interact with decentralized apps (dApps), and manage crypto assets.
- Examples: MetaMask, Coinbase Wallet, Ledger.

**15. What is MetaMask?**

MetaMask is a popular **Ethereum-based Web3 wallet** available as a browser extension and mobile app. It:

- Stores private keys
- Lets users send/receive Ether and tokens
- Connects to dApps
- Works with mainnet and testnets

---

# Installation of MetaMask

1. **Open your browser** (Chrome, Firefox, Brave, etc.), **Go to the official MetaMask website** (metamask.io).
2. **Click on 'Download'**. **Choose the browser extension option**. **Add the extension to your browser**.

# Create your own wallet using Metamask

3. **Click the MetaMask icon** in the extensions area. **Click 'Get Started'**.
4. Select **'Create a new wallet'**. **Agree to the terms** and continue.
5. **Set a strong password** (this only protects the local browser wallet).
6. Click **'Secure My Wallet'**. MetaMask generates a **12-word Secret Recovery Phrase**. **Write down the recovery phrases** on paper. Do not store online.
7. **Click 'Next'**.
8. MetaMask asks you to **confirm the 12 words** in the correct order. **Select the words in sequence** and confirm.
9. You will see a message: **"Wallet Creation Successful"**. You are now shown the **wallet dashboard** with your account and balance.
10. Click on your **public address** to copy it for receiving Ethereum or tokens.
11. **Wallet setup is complete**.

---

**Prerequisite:**

**1. What is the Ethereum Mainnet?**

The Ethereum Mainnet is the **primary public blockchain network** where real ETH cryptocurrency and transactions exist. All transactions involve real value and incur actual costs.

**2. What is the Linea Mainnet?**

Linea Mainnet is a **Layer 2 scaling solution** built on top of Ethereum. Created by ConsenSys, it reduces transaction costs and improves speed while maintaining Ethereum-level security.

**3. What is a Test Network (Testnet) in Blockchain?**

A Testnet is a testing and educational environment. It is an exact copy of the Mainnet's features and rules, but it uses test tokens (like Sepolia ETH) that have no real-world value. These networks allow you to deploy contracts and test transactions without spending real money. Examples: Goerli, Sepolia, Linea Goerli

4. **Goerli and Sepolia. What is the difference?**

They are both test networks, but:

- **Goerli:** This was a popular testnet, but it is now **deprecated (no longer recommended)**. Your note about it costing money is correct. Its test tokens became scarce and, as a result, gained a small real-world value, which defeated its purpose as a free testing ground.
- **Sepolia:** This is the **current, recommended testnet** for application development. It is more stable, and its test tokens are easier to acquire for free, making it ideal for your practical exam.

**5. Can Ethereum Mainnet be used without buying Ether?**

No. All transactions on **Ethereum Mainnet**—even basic transfers or smart contract interactions—require real Ether. You must Buy Ether from an exchange.

**6. How do we get the free tokens for a test network?**

We get them from a **Faucet**.

**7. What is a Faucet in Blockchain?**

A faucet is a website or service that gives out small amounts of **free testnet tokens** (Ether, for example) so users can run transactions on test networks. Examples of Ethereum faucets:

- **Alchemy Sepolia Faucet**
- **QuickNode Faucet**
- **Infura Faucet**

**8. How much is 1 Ether in Wei?**

$$1 \text{ Ether (ETH)} = 10^{18} \text{ Wei}$$

Wei is the smallest unit of Ether, similar to how 1 Rupee = 100 Paise. Gas fees and balances are often denominated in Wei.

Example: 0.001 Ether = 1,000,000,000,000,000 Wei (1e15 Wei)

**9. What is Gas in Blockchain?**

**Gas** is the fee paid to validators for processing transactions on Ethereum. Expressed in **Gwei**, a subunit of Ether. Required for every operation, including sending tokens or deploying contracts. Higher gas = higher priority (faster processing).

# Study spending Ether per transaction

**Prepare MetaMask**

1. Open your browser.
2. Click the MetaMask extension icon.
3. Unlock MetaMask with your password.
4. Click the account avatar and choose **Settings** → **Advanced**.
5. Enable **"Show test networks"** if it's off.
6. Close settings.

**Switch to Sepolia**

7. Click the network dropdown at the top of MetaMask.
8. Select **Sepolia** from the network list.

**Create or pick receiver account**

**9.** If needed, click your account name → **Create Account.**
**10.** Enter a name and click **Create**.
**11.** Click the new account to switch to it.
**12.** Click the address (0x...) to copy the public address to clipboard.

**Send test Ether**

13. In MetaMask, switch to the **sender** account (the one that has Sepolia ETH).
14. Click **Send**.
15. In the "Add recipient" field paste the **receiver** address you copied earlier.
16. Enter the amount of Sepolia ETH to send (e.g., 0.1).
17. Click **Next**.

**Review gas and transaction details**

18. Look at the **Estimated gas fee** shown by MetaMask.
19. Confirm the **Total** (amount + gas).

**Confirm and submit**

20. Click **Confirm** to send the transaction.
21. MetaMask will show the transaction in the **Activity** list.

# BlockChain 3: Write a smart contract on a test network, for Bank account of a customer for following operations:

- **Deposit money**
- **Withdraw Money**
- **Show balance**

---

**1. What is Solidity?**

Solidity is a **high-level programming language** used to write smart contracts. It is specifically designed for the **Ethereum blockchain** and runs on the **Ethereum Virtual Machine (EVM)**. Syntax resembles JavaScript, C++, and Python.

**2. What is Remix IDE?**

It is a browser-based Integrated Development Environment for writing, compiling, and deploying Solidity smart contracts.

**3. Solidity Basics:**

**File Structure: Solidity Source Code**

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;


contract MyContract {

  // State variables, functions, constructor go here

}
```

- pragma: specifies the compiler version.
- contract: like a class; holds state and functions.
- SPDX-License-Identifier: declares the license (optional but recommended).

**Data Types in Solidity**

- **Basic Types**: uint (unsigned integer), int, bool, string, address, bytes

**State Variables**

- Stored permanently on the blockchain. Declared inside the contract but outside functions. Can be marked public, private, etc.

```
string public name;

uint256 private balance;
```

**Functions**

```
function functionName(parameters) visibility returns(type) {
  // logic
}
```

**Constructor**

- Special function that runs **once during deployment**.
- Usually used to initialize values.

**Special Keywords**

- msg.sender – address that called the function
- msg.value – amount of Ether sent with the transaction
- payable – allows function or address to receive Ether

```
function deposit() public payable {
   balance += msg.value;
}
```

**Sending Ether**

```
payable(msg.sender).transfer(amount); // sends Ether
```

**Error Handling**

- require(condition, "Error message")
  Reverts if condition is false.

**Example:**

```
require(balance > 0, "Insufficient balance");
```

---

**Code: (Bank.sol)**

```
// SPDX-License-Identifier: UNLICENSED

pragma solidity ^0.8.18;


contract SimpleBank {

  address public owner;          //variable to store the address of the contract owner.


  constructor() {                //It sets the 'owner' to the address that deployed the contract.

    owner = msg.sender;          //'msg.sender' is the address of the person deploying the contract.

  }


  function showBalance() public view returns (uint) {

    return address(this).balance;   // 'address(this).balance' refers to the balance of the contract itself.

  }


  function deposit() public payable {

    require(msg.sender == owner, "Only the owner can deposit.");

  }


  function withdraw(uint _amount) public {

    require(msg.sender == owner, "Only the owner can withdraw.");

    require(address(this).balance >= _amount, "Insufficient funds in contract.");

    payable(owner).transfer(_amount);

  }
```

1. **function showBalance():** Show the total balance of Ether held by this contract. return The current balance in Wei. This is a 'view' function, meaning it only reads data and costs no gas to call.

2. **function deposit():** Deposit Ether into the contract. This function is 'payable', which allows it to receive Ether. We add a security check to ensure only the owner can deposit. // Note: The Ether transfer is handled automatically because the function is 'payable' and the user sent Ether (msg.value) when calling it. No further code is needed to accept the Ether.

3. **function withdraw(uint _amount):** Withdraw a specific amount of Ether from the contract. Only the owner can withdraw, and only if the contract has enough funds. parameter: _amount - The amount of Ether (in Wei) to withdraw.
    - 'payable(owner)' converts the owner's address to a payable address.
    - '.transfer(_amount)' sends the specified amount to that address

## Executing the Code:

1. **Open Remix:** Go to remix.ethereum.org.
2. **Create File:** In the "File Explorers" sidebar, click the "New File" icon and name your file Bank.sol.
3. **Paste Code:** Paste the corrected code from above into the editor.
4. **Go to Compiler:** Click the "Solidity Compiler" tab on the left (it looks like a plugin icon).
    - Ensure the "Compiler" version matches the code (e.g., 0.8.18).
    - Click the blue "**Compile Bank.sol**" button.
    - A green checkmark will appear on the tab if it is successful.
5. **Go to Deploy:** Click the "Deploy & Run Transactions" tab (it looks like an Ethereum logo).
6. **Select Environment:**
    - At the top, change the "ENVIRONMENT" dropdown to **"Remix VM (Shanghai)"** (or any "Remix VM" option).
    - Your note is correct: this creates a **local, virtual blockchain** inside your browser. It provides you with several test accounts with 100 fake ETH each.
7. **Deploy Contract:**
    - Ensure your Bank contract is selected in the "CONTRACT" dropdown.
    - Click the orange "**Deploy**" button.
8. **Interact:**
    - Look at the bottom of the sidebar. You will see your contract under "Deployed Contracts." Click the small arrow to expand it.

## Interacting with Your Deployed Contract

You will now see the functions from your code as buttons.

**To Deposit Money:**

1. Near the top of the "Deploy" tab, find the **"VALUE"** field.
2. Enter a number, for example, 1000.
3. Change the unit dropdown next to it from "Ether" to **"Wei"**.
4. Now, click the orange deposit button in your deployed contract.
5. The transaction will appear in the console. You have just deposited 1000 Wei.

**To Show Balance:**

1. Click the blue showBalance button. It will display 1000 below the button, showing the current balance.

**To Withdraw Money:**

1. Click the orange withdraw button.
2. The transaction will succeed (because you are the accHolder).
3. Click showBalance again. The balance will now be 0.
4. If you check your "ACCOUNT" dropdown at the top of the sidebar, you will see your account's balance (which was 100 ETH) has increased slightly by the 1000 Wei you withdrew.