Assignment No: 01

Title: Crypto wallet

Objective:

- To study about crypto wallets.
- To setup a wallet on the browser.

Problem Statement:

To setup a crypto wallet

i) hosted wallets ii) self-custody wallet iii) hardware wallets (optional) and evaluate each of these

Theory / Procedure / Diagrams:

A Cryptocurrency wallet refers to a physical medium, device, service, or application that maintains private and/or public passwords for crypto transactions. In addition to the basic purpose of storing keys, it also makes the owner of the digital pseudonymous.

A crypto wallet is used to interact with a blockchain network. The three major types of crypto wallets are hardware, software, and paper wallets. Based on their work, they can be further classified as cold or hot wallets. Software-based wallets are more accessible and more convenient, whereas hardware ones are the most secure. Paper wallets are printed out on paper and are now unreliable and obsolete. Crypto wallets don't store the currency but act as a tool of interaction with blockchain, i.e., generating the necessary information to receive and send money via blockchain transactions.

The information comprises pairs of private and public keys. Based on these keys, an alphanumeric identifier called address is generated. In essence, this address specifies the location to which coins can be sent to the blockchain. The address can be shared to receive funds, but private keys are to be never disclosed. The private key can be used on any wallet for accessing the Cryptocurrency. If the private key is known, funds are accessible on any device. Also, coins are just transferred from one address to another, never leaving the blockchain.

1. Hosted Wallet:

A Hosted wallet is an online service where a user's Bitcoins are stored, and they can send and receive Bitcoins from this service. Coin base, Bit flyer, and Mt Gox are examples of hosted wallets. A hosted crypto wallet is a digital wallet in which your private keys are stored. In exchange, the wallet takes care of the backup and security of your funds. a hosted wallet as "an account-based software program for storing cryptographic keys controlled by an identifiable third party. These parties receive, store, and transmit Cryptocurrency transactions on behalf of their accountholders; the accountholder generally does not have access to the cryptographic keys

themselves. “In comparison to other crypto wallets, a hosted wallet is more similar to regular banking apps — you can always regain access to your assets, even if you have forgotten your password, mnemonic phrase or lost your phone. Other types of wallets do not provide this possibility. In the event that you lose access to your credentials or a device on which the wallet is installed, you will lose your funds.

2. Self-Custody Wallet:

Self-custodial crypto wallets provide you with direct access to public blockchain’s. The best wallets, like the Bitcoin.com Wallet, allow you customize the fees you pay to public blockchain miners and validators. This means, for example, that you can choose to pay less for transactions when you're not in a hurry (or more if you're in a rush!). Finally, because self-custodial wallets provide direct access to blockchain’s, they also enable you to interact with smart contracts. That means, for example, you can access decentralized finance products that enable you to earn passive income.

3. Hardware Wallets:

A hardware wallet is a special type of bitcoin wallet which stores the user's private keys in a secure hardware device.

They have major advantages over standard software wallets:

Private keys are often stored in a protected area of a microcontroller, and cannot be transferred out of the device in plaintext immune to computer viruses that steal from software wallets can be used securely and interactively, private keys never need to touch potentially-vulnerable software much of the time, the software is open source, allowing a user to validate the entire operation of the device hardware wallet could fail to protect your Bitcoin:

1. Malware swaps recipient Bitcoin addresses: a hardware wallet won't protect you from being tricked into sending Bitcoin to the wrong address. For example, malware on a PC could monitor for high value transactions and then swap out the recipient's authentic Bitcoin address for an address controlled by the attacker. When the stakes are high, multi factor (e.g., over the phone) confirmation of a recipient's Bitcoin address is recommended.

2. Insecure RNG (Random Number Generator): hardware wallets rely on the security of an RNG, often embedded in hardware, to generate your wallet's private keys securely. Unfortunately, it is notoriously difficult to verify the true randomness of the RNG. An insecure

RNG may create wallet keys that can later be recreated by an attacker, by generating psuedo-randomness that would seem statistically indistinguishable from true randomness yet still be predictable to an advanced attacker. An RNG may become insecure as a result of malicious weakening or an unintentional mistake. This failure mode is common to any wallet generation procedure in which the true randomness of the source of entropy being used can not be verified.

3. Imperfect implementation: the security of all computing devices relies on the quality of their implementation. Hardware wallets are no exception. Bugs at the software, firmware or hardware level may allow attackers to break into a hardware wallet and gain unauthorized access to secrets. Even if the design is perfect, proving the security of a hardware or software implementation is a very hard, mostly unsolved problem. To date, no wallet in existence is implemented using provably correct software.

Set up a wallet-Metamask:

Metamask is an self hosted wallet. MetaMask is a decentralized wallet with vast features and Web3 applications that make it a central hub for NFT and Web3 enthusiasts who desire to build decentralized applications on the Ethereum blockchain.

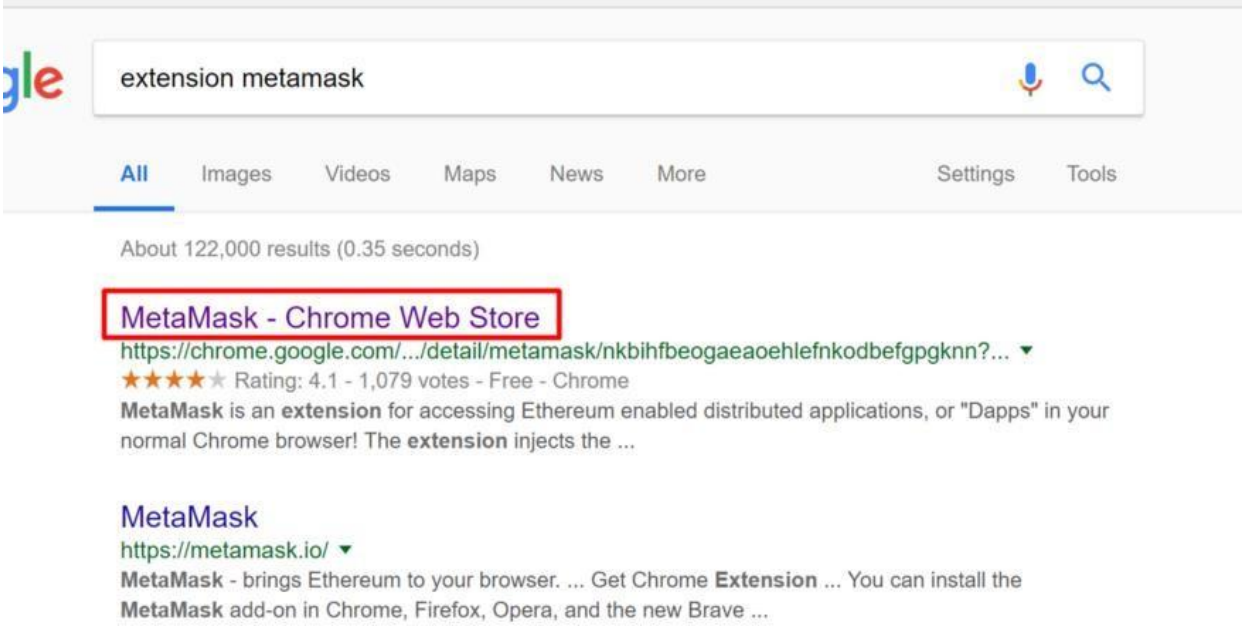
How to use MetaMask: A step by step guide

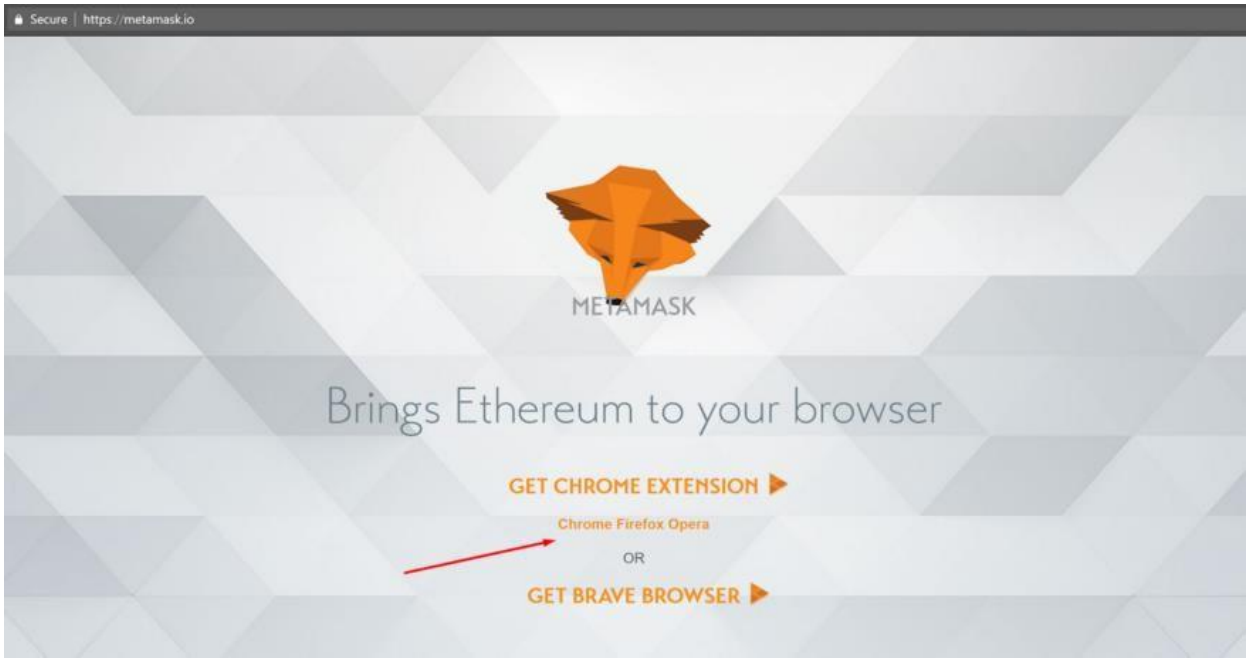
MetaMask is one of the most popular browser extensions that serves as a way of storing your Ethereum and other ERC-20 Tokens. The extension is free and secure, allowing web applications to read and interact with Ethereum's blockchain.

Step 1. Install MetaMask on your browser.

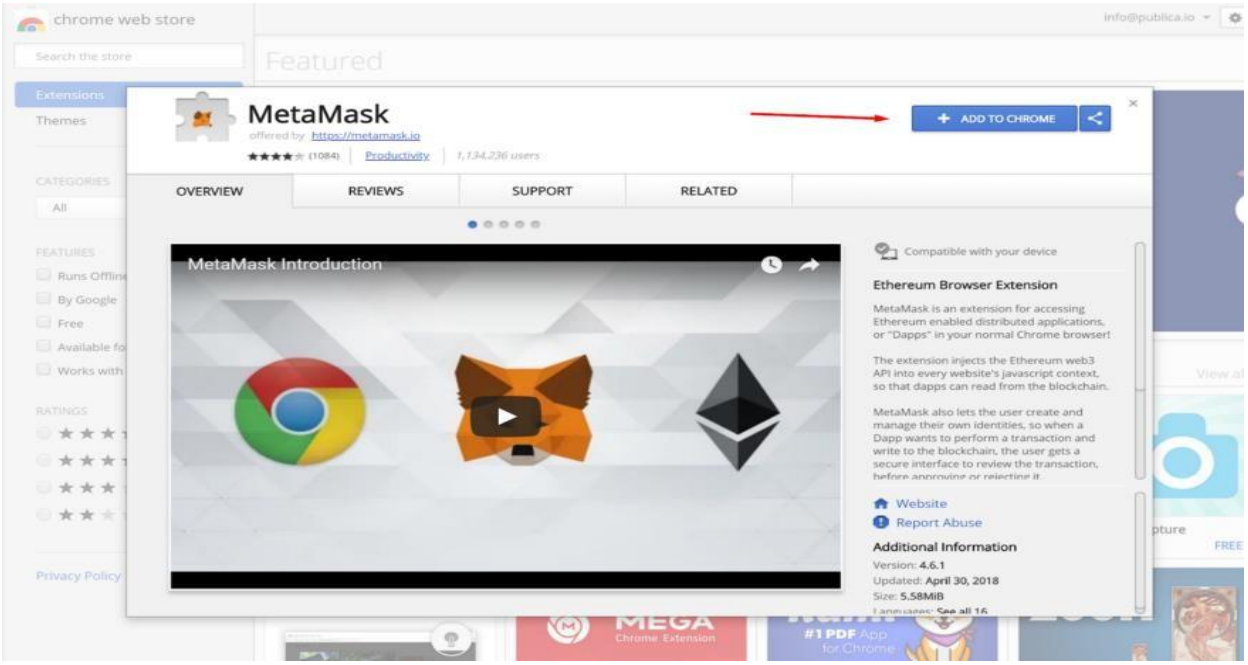
To create a new wallet, you have to install the extension first. Depending on your browser, there are different marketplaces to find it. Most browsers have MetaMask on their stores, so it's not that hard to see it, but either way, here they are Chrome, Firefox, and Opera.

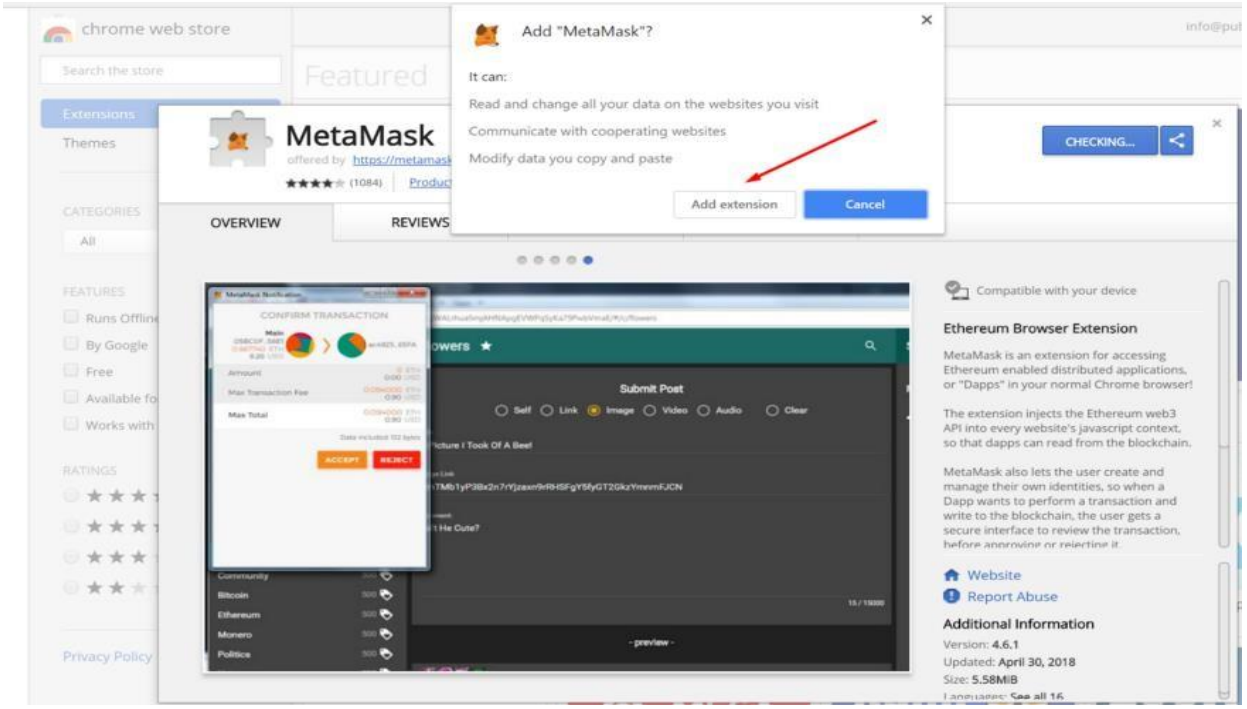
In this tutorial, I will be using Google Chrome as an example but the same steps apply to different browsers as well. If, for some reason, the links above don't work for you, just search MetaMask Extension on your favorite search engine, and you can find it there.





- Click on Install MetaMask as a Google Chrome extension.
- Click Add to Chrome.
- Click Add Extension.

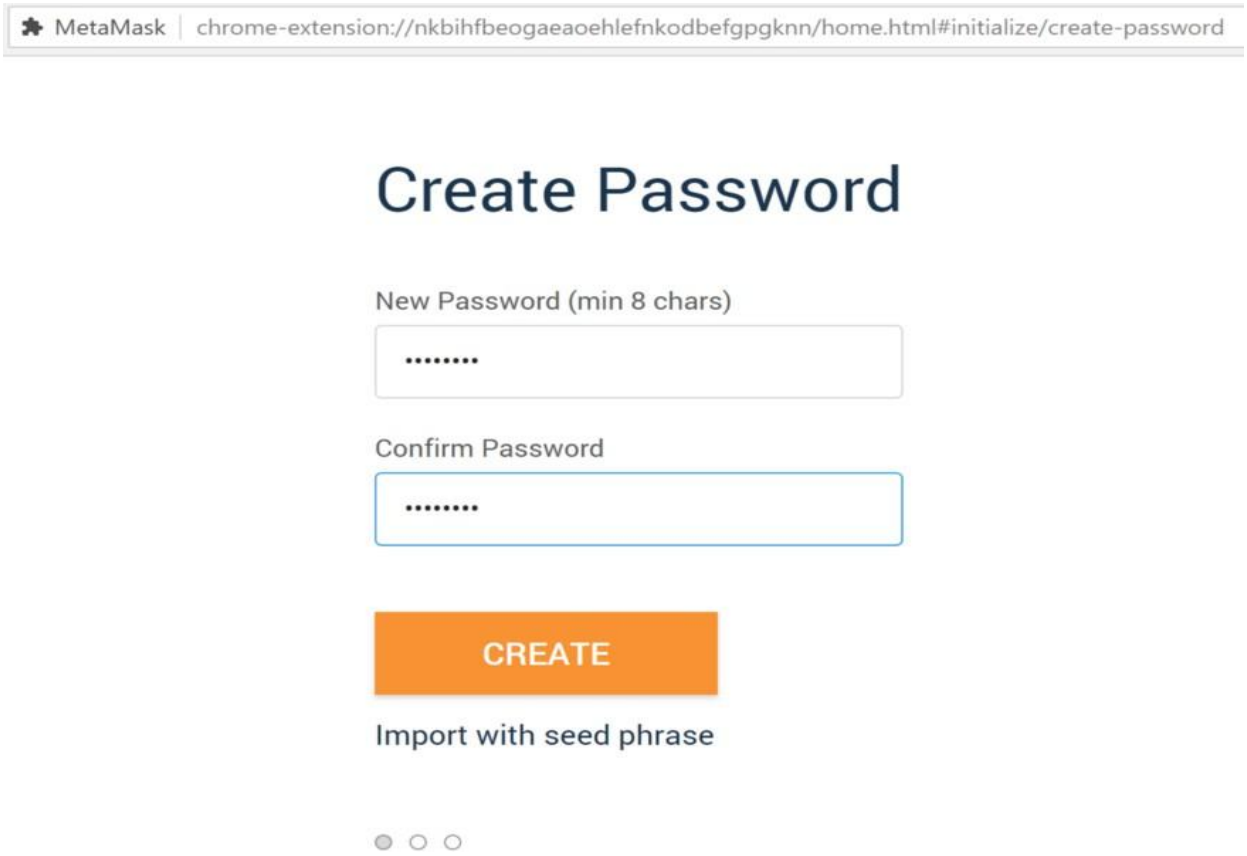




And it's as easy as that to install the extension on your browser, continue reading the next step to figure out how to create an account.

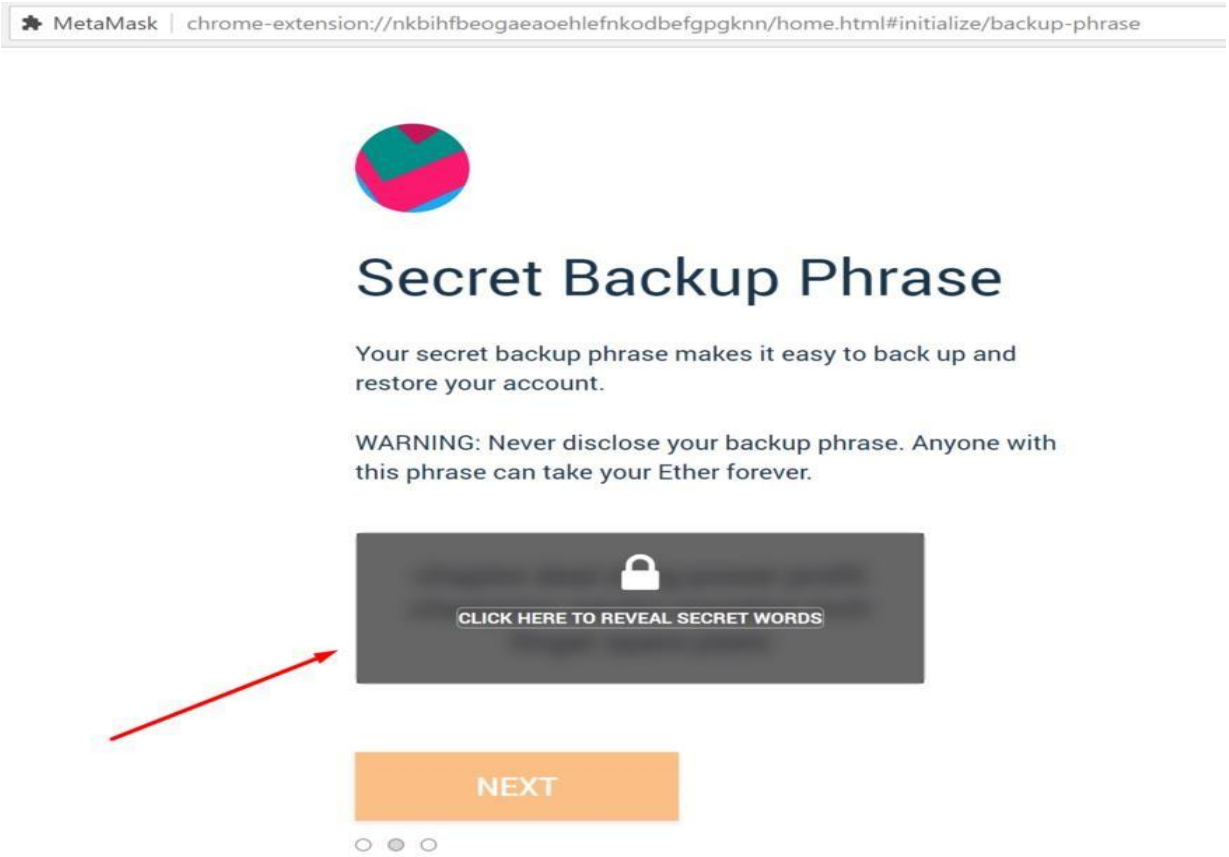
Step 2. Create an account.

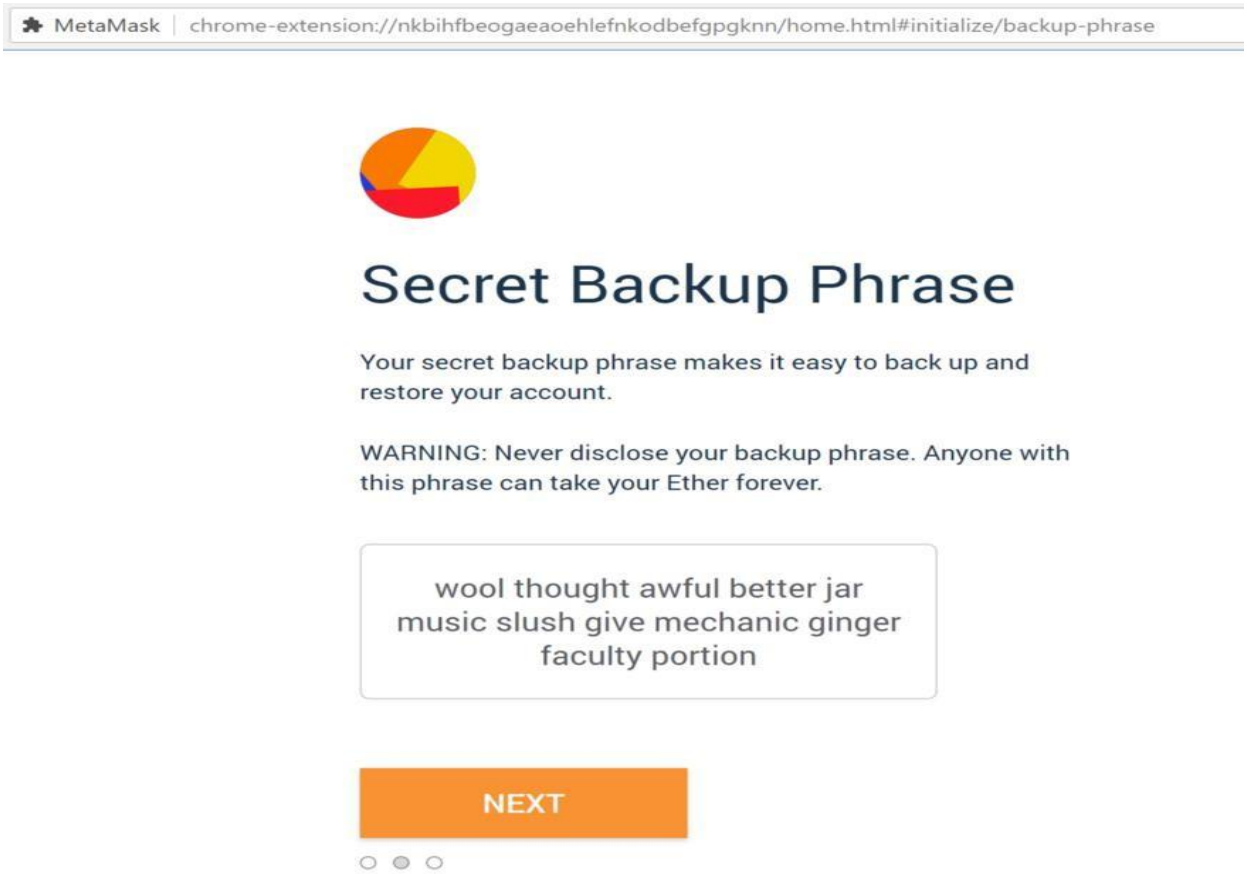
- Click on the extension icon in the upper right corner to open MetaMask.
- To install the latest version and be up to date, click Try it now.
- Click Continue.
- You will be prompted to create a new password. Click Create.



- Proceed by clicking Next and accept the Terms of Use.

Click Reveal Secret Words. There you will see a 12 words seed phrase. This is really important and usually not a good idea to store digitally, so take your time and write it down.





- Verify your secret phrase by selecting the previously generated phrase in order. Click Confirm.

And that's it; now you have created your MetaMask account successfully. A new Ethereum wallet address has just been created for you. It's waiting for you to deposit funds, and if you want to learn how to do that, look at the next step below.

Conclusion:

Thus we studied the different types of wallets and setup a wallet on our browser.

Assignment No: 01 -b

Title: Operations in a crypto wallet.

Objective:

To learn operations in a wallet

Problem Statement:

Understand the basic operations in the wallet on bitcoin such as 1) buy 2) sell 3) send 4) receive 5) exchange 6) mining.

Theory / Procedure / Diagrams / Circuits:

Bitcoin (BTC) is a Cryptocurrency, a virtual currency designed to act as money and a form of payment outside the control of any one person, group, or entity, thus removing the need for third-party involvement in financial transactions. It is rewarded to blockchain miners for the work done to verify transactions and can be purchased on several exchanges. Bitcoin uses blockchain technology to support peer-to-peer transactions between users on a decentralized network.

Transactions are authenticated through Bitcoin's proof-of-work consensus mechanism, which rewards Cryptocurrency miners for validating transactions.

How does Bitcoin work?

Each Bitcoin is a digital asset that can be stored at a Cryptocurrency exchange or in a digital wallet. Each individual coin represents the value of Bitcoin's current price, but you can also own partial shares of each coin. The smallest denomination of each Bitcoin is called a Satoshi, sharing its name with Bitcoin's creator. Each Satoshi is equivalent to a hundred millionth of one Bitcoin, so owning fractional shares of Bitcoin is quite common.

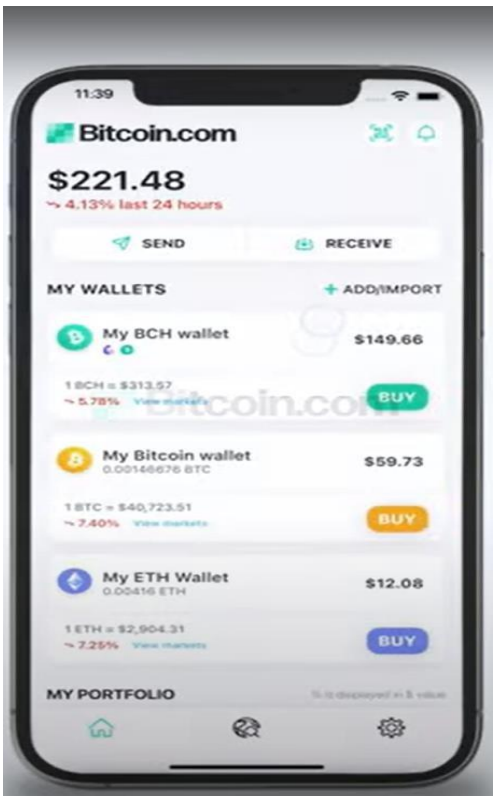
- **Blockchain:** Bitcoin is powered by open-source code known as blockchain, which creates a shared public history of transactions organized into "blocks" that are "chained" together to prevent tampering. This technology creates a permanent record of each transaction, and it provides a way for every Bitcoin user to operate with the same understanding of who owns what.
- **Private and public keys:** A Bitcoin wallet contains a public key and a private key, which work together to allow the owner to initiate and digitally sign transactions. This unlocks the central function of Bitcoin — securely transferring ownership from one user to another.
- **Bitcoin mining:** Users on the Bitcoin network verify transactions through a process known as mining, which is designed to confirm that new transactions are consistent with other

transactions that have been completed in the past. This ensures that you can't spend a Bitcoin you don't have, or that you have previously spent.

Basic Operations in the wallet on Bitcoin using Bitcoin wallet:

Receiving Bitcoin

1. Open your Bitcoin.com wallet app and tap the 'Receive' button at the top of the Home screen.
2. Choose which wallet you want to receive Bitcoin to. Make sure you select a (BCH) wallet if you are receiving Bitcoin Cash or a (BTC) wallet if you are receiving Bitcoin.
3. Your chosen wallet will generate an address that lets you receive coins. Copy this by tapping the QR code.
4. Provide this address to the sending party, or if you're in person, the sender can simply scan your wallet QR code with their device.



Sending Bitcoin

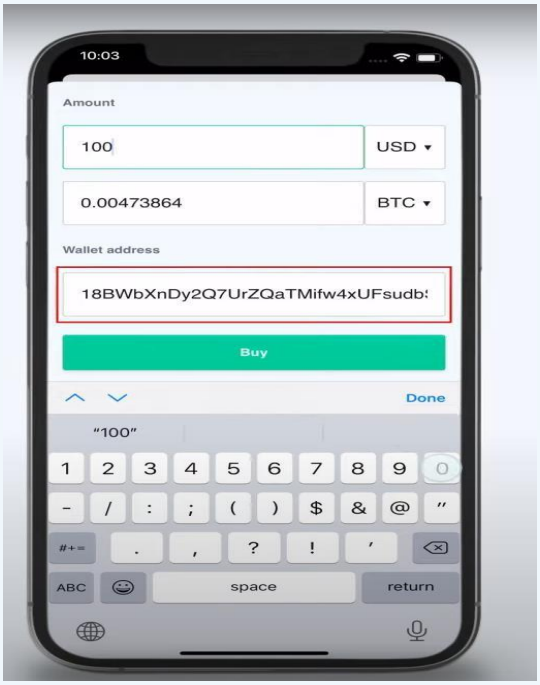
1. Open your Bitcoin.com wallet app and tap the 'Send' button at the top of the Home screen.
2. Copy and paste the recipient's wallet address into your own wallet app. If you're in person, select "Scan QR code" and simply scan it with your app.

3. Choose which wallet you want to send Bitcoin from. Make sure you select a BCH wallet if you want to send Bitcoin Cash or a BTC wallet if you want to send Bitcoin.
4. Enter how much you want to send and tap on ‘Continue’.

iii. Buying a bitcoin

Crypto wallets allow you to buy bitcoin conveniently from within the wallet app, and the **Bitcoin.com Wallet** is no exception. Importantly, the **Bitcoin.com Wallet** is fully non-custodial. This means you're always in complete control of your bitcoin. Here's the process for buying bitcoin using our app:

1. Open the **Bitcoin.com Wallet** app on your device.
2. Select Bitcoin (BTC) and tap the "Buy" button. Note: you can also buy other digital assets.
3. Follow the on-screen instructions to choose your preferred wallet for depositing. The **Bitcoin.com Wallet** actually consists of separate wallets for each digital asset we support (eg. BTC, BCH, etc.). Additionally, you can make as many individual wallets as you want, a feature that can help you to organize your funds. For example, you can make one Bitcoin wallet called *My BTC Savings* and another Bitcoin wallet called *Everyday BTC Spending*.
4. If it's your first purchase, you may be asked to verify your identity. Future purchases are completed in seconds!.Once complete, your purchase will proceed.



iv. selling a bitcoin

With this method, you can sell any amount of bitcoin (BTC) in your digital wallet directly to cash, which will then be deposited in your bank account. Here's how it works:

1. On the app's home screen, tap the "SELL" button.
2. If you haven't done so already, follow the instructions to connect your bank account.
3. Select the amount you'd like to sell. You can input the amount in either local currency terms or Bitcoin terms.
4. Confirm the transaction.

v. exchanging a Bitcoin:

Bitcoin exchange platforms match buyers with sellers. Like a traditional stock exchange, traders can opt to buy and sell Bitcoin by inputting either a market order or a limit order. When a market order is selected, the trader is authorizing the exchange to trade the coins for the best available price in the online marketplace. With a limit order set, the trader directs the exchange to trade coins for a price below the current ask or above the current bid, depending on whether they are buying or selling.

To transact in Bitcoin on an exchange, a user has to register with the exchange and go through a series of verification processes to authenticate their identity. Once the authentication is successful, an account is opened for the user who then has to transfer funds into this account before they can buy coins.

vi. mining a Bitcoin:

Bitcoin mining is the process by which new Bitcoins are entered into circulation. It is also the way the network confirms new transactions and is a critical component of the blockchain ledger's maintenance and development. "Mining" is performed using sophisticated hardware that solves an extremely complex computational math problem. The first computer to find the solution to the problem receives the next block of Bitcoins and the process begins again.

Cryptocurrency mining is painstaking, costly, and only sporadically rewarding. Nonetheless, mining has a magnetic appeal for many investors who are interested in Cryptocurrency because of the fact that miners receive rewards for their work with crypto tokens. This may be because entrepreneurial types see mining as pennies from heaven, like California gold prospectors in 1849. And if you are technologically inclined, why not do it?

The Bitcoin reward that miners receive is an incentive that motivates people to assist in the primary purpose of mining: to legitimize and monitor Bitcoin transactions, ensuring their validity. Because many users all over the world share these responsibilities, Bitcoin is a "decentralized" Cryptocurrency, or one that does not rely on any central authority like a central bank or government to oversee its regulation.

Input: (Test Cases / Data sets / Database Links)

Output: (Results / Visualization)

Expected sample Output

Conclusion:

Thus, we studied the basic operations of Bitcoin wallet.

Assignment No: 02-a

Title: Build and deploy a smart contract using hardhat.

Objective:

- Deploying a smart contract locally .
- To deploy a contract using test net and connect it to wallet

Problem Statement:

1) Create a local Ethereum network using Hardhat or any other tool, build a smart contract that lets you send a $ (wave) to your contract and keep track of the total # of waves. Compile it to run locally.

Requirements:

VS code, Nodejs

Theory / Procedure / Diagrams / Circuits:

Hardhat: Hardhat is an Ethereum development environment for professionals. It facilitates performing frequent tasks, such as running tests, automatically checking code for mistakes or interacting with a smart contract. Hardhat Runner is the main component you interact with when using Hardhat. It's a flexible and extensible task runner that helps you manage and automate the recurring tasks inherent to developing smart contracts and dApps.

Hardhat Runner is designed around the concepts of **tasks** and **plugins**. Every time you're running Hardhat from the command-line, you're running a task. For example, npx hardhat compile runs the built-in compile task. Tasks can call other tasks, allowing complex workflows to be defined. Users and plugins can override existing tasks, making those workflows customizable and extendable.

Check out the plugin list to use it with your existing tools. Built by the Nomic Foundation for the Ethereum community. Hardhat is a flexible and diverse JavaScript-based framework for Ethereum blockchain developers. hardhat comes built-in with Hardhat Network, a local Ethereum network node designed for development. It allows you to deploy your contracts, run your tests and debug your code, all within the confines of your local machine. It runs as either an in-process or stand-alone daemon, servicing JSON-RPC and WebSocket requests.By default, it mines a block with each transaction that it receives, in order and with no delay.

It's backed by the @ethereumjs/vm EVM implementation, the same one used by ganache, Remix and Ethereum Studio.

Procedure:

Creating a local Ethereum network and compiling it locally :

1. Install Node.js and npm (if you haven't already).
2. Install Hardhat

   To install it, you need to create an npm project by going to an empty folder, running npm init, and following its instructions. By running this command

   npm install --save-dev hardhat.
3. Initialize a new Hardhat project by running npx hardhat init.
4. Install the necessary dependencies for our project by running npm install ethers hardhat-deploy
5. Open the hardhat.config.js file and add the following lines of code to the networks section:

```yaml
module.exports = {
  networks: {
    hardhat: {
      chainId: 1337
    }
  }
};
```

7. Create a new file called WavePortal.sol in the contracts directory and add the following code:

```csharp
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract WavePortal {
    uint256 public totalWaves;

    function wave() public {
        totalWaves += 1;
    }
}
```

This smart contract has a totalWaves variable that keeps track of the number of waves sent to the contract, and a wave function that increments this variable by 1 every time it's called.

7. Compile the smart contract by running npx hardhat compile.

8. Create a new file called deploy.js in the scripts directory and add the following code:

```javascript
const { ethers } = require("hardhat");

async function main() {
  const WavePortal = await ethers.getContractFactory("WavePortal");
  const wavePortal = await WavePortal.deploy();

  console.log("WavePortal deployed to:", wavePortal.address);
}

main()
  .then(() => process.exit(0))
  .catch(error => {
    console.error(error);
    process.exit(1);
  });
```

This script deploys the WavePortal smart contract to the local network.

9. Deploy the smart contract by running npx hardhat run scripts/deploy.js --network hardhat.

10. Now we can interact with the contract using the Hardhat console. Run npx hardhat console to start the console, and then run the following commands to send a wave to the contract and check the current number of waves:

```javascript
const WavePortal = await ethers.getContractFactory("WavePortal");
const wavePortal = await WavePortal.attach("CONTRACT_ADDRESS_HERE");
await wavePortal.wave();
console.log(await wavePortal.totalWaves());
```

Input: (Test Cases / Data sets / Database Links)

Output: (Results / Visualization)

Expected sample Output

Conclusion: Thus we studied the use of hardhat tool and to run the contract locally.

Assignment No: 02-b

Title: Build and deploy a smart contract using hardhat on a web app.

Objective:

- Deploying a smart contract locally.
- To deploy a contract using test net and connect it to wallet.

Problem Statement:

Create a local Ethereum network using Hardhat or any other tool, build a smart contract that lets you send a 👋 (wave) to your contract and keep track of the total # of waves. onnect to any Ethereum wallet. Metamask. Deploy the contract with testnet. Connect your wallet with your web app. Call the deployed contract through your web app. Then store the wave messages from users in arrays using structs.

Requirements:

VS code, NodeJS

Theory / Procedure / Diagrams / Circuits:

Steps to follow to create an app web app to call the smart contract:

1. Create the smart contract: Write the smart contract code in Solidity programming language. You can use tools like Remix or Visual Studio Code with Solidity plugins to write, compile and debug your contract code.
2. Deploy the smart contract to a testnet: Deploy the smart contract to a testnet like Rinkeby, Ropsten or Kovan to test and validate its functionality. You can use a tool like Truffle or Remix to deploy your contract to a testnet.
3. Connect your wallet to your web app: In order to connect your wallet to your web app, you will need to use a Web3 provider like Metamask. Metamask is a browser extension that allows you to interact with the Ethereum network and your wallet from your web app. You can integrate Metamask by adding its script to your web app and calling its functions to interact with the Ethereum network.
4. Call the deployed contract through your web app: Once your wallet is connected to your web app, you can call the functions of the deployed contract using the Web3 provider. You will need

to use the contract address and ABI (Application Binary Interface) to interact with the contract functions.

Here's a high-level example of how you can connect your web app to a deployed smart contract on the Rinkeby testnet using Metamask:

1. Write the smart contract code in Solidity and compile it using Remix.
2. Deploy the smart contract to the Rinkeby testnet using Truffle.
3. Create a new web app and integrate the Metamask script by adding the following code to your HTML file:
4. Connect your wallet to the web app by calling `window.ethereum.request` function when the the user clicks a "Connect Wallet" button:
5. Call the contract functions by creating a new instance of the contract object using the contract address and ABI, and calling the desired function:

Smart Contract :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract WavePortal {
    struct Wave {
        address sender;
        string message;
        uint256 timestamp;
    }

    uint256 totalWaves;
    Wave[] waves;

    event NewWave(address indexed sender, string message, uint256 timestamp);

    function wave(string memory _message) public {
        totalWaves += 1;
        waves.push(Wave(msg.sender, _message, block.timestamp));
        emit NewWave(msg.sender, _message, block.timestamp);
    }

    function getTotalWaves() public view returns (uint256) {
```

```
        return totalWaves;
    }

    function getWaves() public view returns (Wave[]
        memory) { return waves;
    }
}
```

This contract has a struct called **Wave** which contains the **sender** address, the **message** string, and the **timestamp** of the wave. When a user sends a wave, the **wave** function is called, which increments the **totalWaves** counter, creates a new **Wave** struct with the sender's address, message, and timestamp, and adds it to the **waves** array. The **NewWave** event is also emitted with the same information.

The **getTotalWaves** function returns the total number of waves that have been sent, while the **getWaves** function returns an array of all the **Wave** structs that have been sent so far.

To deploy this contract on a testnet, you can use tools like Remix or Truffle. Once deployed, you can connect to the deployed contract using your Ethereum wallet (e.g., Metamask) and call the **wave** function through your web app to send a wave to the contract. You can also call the **getTotalWaves** and **getWaves** functions to retrieve information about the waves that have been sent.

Input: (Test Cases / Data sets / Database Links)

Output: (Results / Visualization)

Expected sample Output

Conclusion: Thus we studied the use of hardhat tool and to run the contract and called the contract using web app.