**Week 2**

**2.1 Public Key Cryptography**

* Uint8Array are used to represent bytes in browsers and code. They are efficient in terms of memory from normal arrays.
* Private keys are an array of 32 bytes. If we want to guess someone’s private key, the possibilities are 255^32. Each byte can at most represent 255.
* In hex, we’ll have twice the number of characters than ASCII. Because, hex takes 4 while ASCII takes 8 bits.
* Base64 takes 6 bits.
* Base58 is similar to Base64 but does not include visually similar characters like capital I and small l and capital O and 0.
* In hashing you can’t get original data. In encryption, you can get the original data by decryption.
* In blockchains, we use asymmetric encryption.
* A public key can be created for a private key. Using the private key, you can sign a message. The message can be decrypted using the public key which you have shared with other people. The other people then for sure know that the message was sent from the person who has the private key.
* For hashing, we use SHA-256. Foe encryption, we use RSA and other algorithms.
* Public key is like your bank account number which you can send to other people so that they can send you crypto.
* For sending crypto, you need to sign the transaction with your private key. The blockchain would then verify the message signature whether this transaction came from you or not using your public key.
* Anytime a block gets put on the blockchain then whoever made the block or was the miner gets some bitcoin.
* Every block on the blockchain has a bunch of transactions and every transaction has a signature.
* You can create multiple wallets using the same seed phrase.
* With every public key, there is an associated private key.
* Make sure the wallets you use are open source, so you can go through the codebase.
* In Uint8Array, if a number is bigger than 255 than it gets converted to an 8-bit number and the MSBs are discarded.
* Solana uses Base58 encoding.
* Base64 makes sure that the resultant string is a multiple of 4.If there are not enough characters, the resultant string gets padded from the right with “=”.
* Base58 is all characters from Base64 excluding 6 characters.
* In symmetric encryption, you encrypt and decrypt using the same private key.
* In asymmetric encryption, you sign using the private key and the miner decrypts it using the corresponding public key.
* In RSA, you multiply two prime numbers and get a very long number. The secret is the two prime numbers you multiplied, it’s very hard to find the two prime numbers and only way is to brute force.
* Hashing 🡪 SHA256, MD5. Symmetric Encryption 🡪 AES. Asymmetric Encryption 🡪 RSA, EdDSA (SOL), ECDSA (BTC, ETH).
* From private key, you can derive the public key. But from a public key, you cannot derive the private key.
* The @noble/ed25519 library generates private and public key as 32 bytes Uint8Array.
* The first block in the blockchain is called the genesis block.
* We as user sign the transaction using encryption. The miner verifies the message and uses hashing to find the nonce.
* The user first hashes the transaction, signs it using their private key and then sends it on the blockchain. Signature, the original message before hashing, and the public key are sent to a node on the blockchain.
* The last 32 bytes of the private key are just the 32 bytes of the public key.
* nacl is a very famous library for signing and verifying.
* Whenever you create a new wallet, you create a new branch from the seed phrase. This is called HD wallets.
* You now only need to keep the seed phrase and retrieve your wallets private/public keys. A single seed phrase can now be used to create all of your wallets.
* You can have a 12 word seed phrase or a 24 word see phrase.
* If you lose your seed phrase then you can’t generate your private keys for your public keys and your funds stay on the blockchain forever.
* The 12 words are your mnemonic. From the mnemonic, we generate the seed. Using the seed, we can then generate public and private key pairs.
* const words = generateMnemonic(mnemonicWords.length === 12 ? 128 : 256); If we give it 128, then it would generate a 12 word seed and if we give it 256, it would generate a 24 word seed.
* Using your seed and the derivation path, you can generate your public and private key.
* They allow users to recreate the same set of addresses and private keys from the seed across different wallets, ensuring interoperability and consistency. (for example if you ever want to port from Phantom to Backpack).
* Derivation paths have are like m/44/401. m denotes root directory, 44 purpose, 401 coin\_type.
* Derivation paths can be written in more than 1 way. There are different standards for writing derivation paths.
* If we’re creating a new platform, we need to make sure to support the same derivation paths that the folks at phantom and backup do so that we can let user transfer their wallets to our platform.
* Menmonic 🡪 seed
* Seed + path 🡪 derived seed 🡪 secret 🡪 public/private key
* The advantage of using mnemonic and seed is that now you don’t need to store private and public keys for all your wallets, rather you need to store only the mnemonic.