Cryptography

Public Key Encryption

Hashing

Overview

The cryptographic elements of the Seed blockchain are required to handle user validation, user authentication, user consent confirmation and the correctness of agreed upon states of the blockchain. Private and public addresses will be required for each user, where their private keys are used for identifying users. Cryptographic signatures will be sued to validate and authenticate users transactions on the network. Elliptic Curve Cryptography (ECC) will be used as our cryptographic approach of choice due to its efficient calculation time and its high security.

Private Key

The private key generated is a 256 bit key. This can base58check encoded into a Wallet Import Format (WIF) as needed. This key will need to be very secure, as it will act as the entry point into any user, and be required to sign all transactions.

In order to raise the security in generating private keys, our user interface will request the users give entropy. This entropy could be mouse movement changes, ten miscellaneous words or even user entered text.

Public Key:

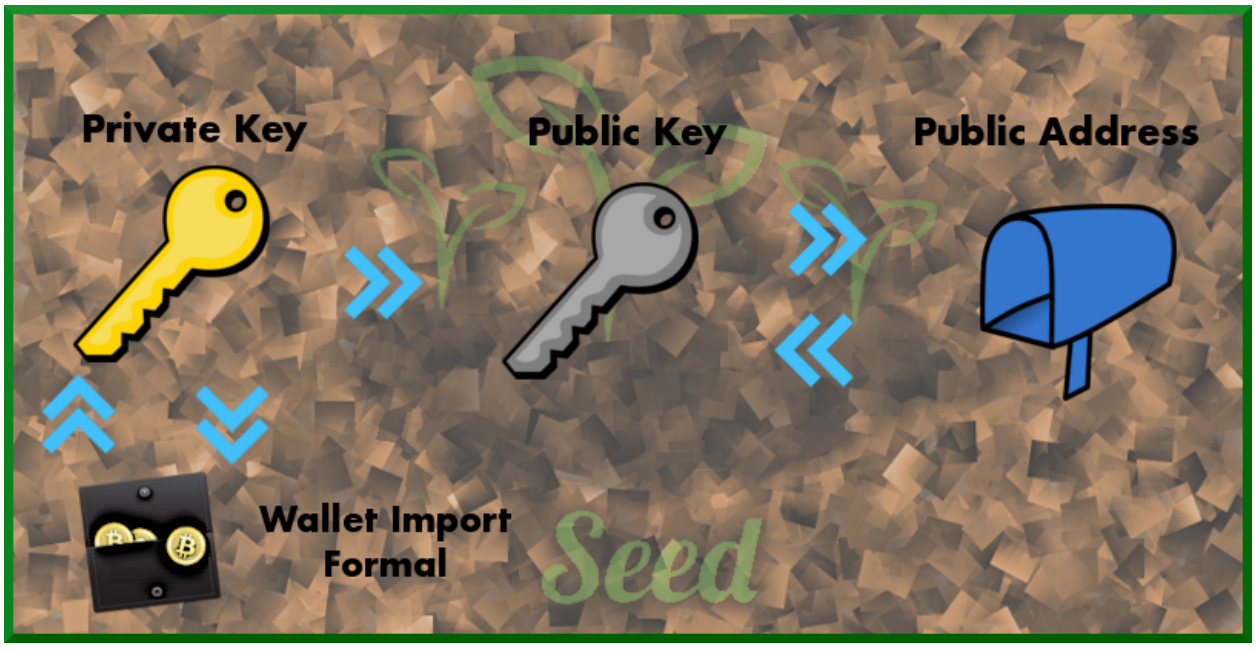
Each private key will generate one public key on our network, which is also 256 bits. More specifically, on the curve (we'll explain that after), a public key is an X coordinate that offers two possible Y coordinates (one positive, the other negative). Therefore, we can store the public key in 257 bits (256 bits for the X, then 1 bit for the sign to denote the positive or negative Y chosen).

The public key will be publicly available to all, however will generally be passed around in a encoded form known as a Public Address. This public key will be the unique identifier each users account is stored under, acting as the unique ID of each account.

Public Address

The Public Address is a base58 check encoded version of a users public key, which goes through the encoding with the network identifier appended to the end. What that effectively does is allow us to take a base64 encoded hexadecimal string and turn it into base58, a more human-readable format for the address. The reason it is more readable is base58 excludes certain characters that may cause confusion. Examples of omitted characters are a capital letter i, which can easily be mistaken for a lower-case L.
By appending the network identifier to the encoding, the same private key/public key combination can have a unique address for each network, allowing the same user to exist for both a private and public network with separate addresses in order to avoid sending the funds through the wrong network.
The example public address for the public key above, running on network “01” would be:

Key/Address/WIF Relationship



PrivateKeys can be encoded into Wallet Import Format (WIF) and decoded back to PrivateKeys.

PrivateKeys can be one-way hashed into PublicKeys.

PublicKeys can be encoded into a Public Address and decoded back into PublicKeys.

Transaction Signing

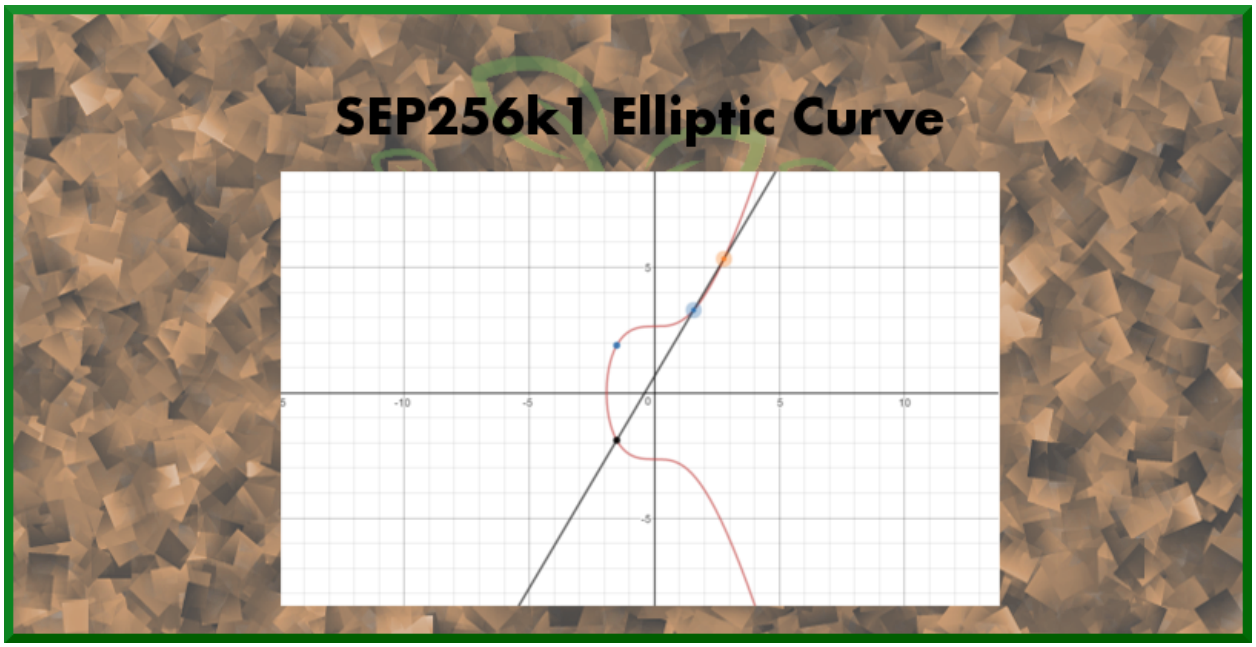
Transaction signing is where a private key and arbitrary data goes through a mathematical process in cryptography where that private key “signs” the data. When this occurs, any user can validate the signature via that users public key, allowing for the authorization of commands with provable consent, without users ever making their private key public.

Elliptic Curve Digital Signature Algorithm

The type of cryptography we decided to go with was Elliptic Curve Cryptography (ECC). This cryptography is generally very easy to recreate however very difficulty to break with a small amount of bits, making it a prime candidate for cryptocurrency cryptography.
The Elliptic Curve Digital Signature Algorithm (ECDSA) is an algorithm used with the ECC to allow for transaction signing. It it used by Bitcoin and Ethereum with great success, and after doing a further analysis, our conclusion was that this was still the most reliably route we could go during the design phase.

In the future, we are interested in pursuing a lattice cryptography alternative in order to future-proof the design. However, this will be a road map item, as we will fall back to ECDSA with the Secp256k1 parameters to build the curve that Bitcoin standardized for our cryptography.

Secp256k1 Curve Parameters



Secp256k1 refers to the parameters of the ECDSA curve and is defined in the Standards for Efficient Cryptography (SEC). Most elliptic curves are created with a random form, however Secp256k1 curve is created in such a way that its shape is fully deterministic and easy to compute. It is estimated to be roughly 30% faster to compute compared to other commonly used ECDSA curves.
Following this standard also proves that nothing extra was added to the curve by developers, while also allows more confidence to be had in the security of the curve in general.

Nested SHA256 Hashing

SHA-256 appears to be the safest hashing algorithm that we can use while minimizing computation time and the bits per hash. SHA-1 previously has been deemed not secure due to a vulnerability known as “The Birthday Attack”. This attack is believed to not be possible in SHA-2, however some experts still fear we may discover this attack in the future. However, specialists agree that, should such an attack be likely to occur, or should SHA-256 be proven to be less secure than originally anticipated, nesting SHA-256 hashes would raise security sufficiently over a simple single-pass hash.
Therefore, in all cases where hashing must be done, such as for hashing transactions, nested SHA-256 hashes will be executed.

Relevant Background Info

Background information that helped me decide on how to structure this design, whether to follow in Bitcoin & Ethereum's footsteps, as well as whether or not to try something different in terms of cryptographic design.

Development Discussion #1: Public Key Encryption

Development Discussion #2: Hashing Algorithms