How are deterministic wallets generated?

So lets take a random number of 128 bits which is 16 bytes which is 32 hexademicals. Always generate this from a cryptographically secure random number generator. Then, "Hash" it using a SHA256 hash algorithm which just "hashes" it into a 256 bit output.

Hashing just means that when you give a input, you get a seemingly random output but it is mathematically verified that if you give the same input again, you would get the same output every single time. Example: If you input "abcd" and it gives out 12345.....256 then you can be sure that if you input "abcd" for the 1000th time, it would still give u the same output it gave you the first time. Also it is very hard to find another input that gives out the same output, which makes it secure. Another important thing about hashes is that it is easy to get hash from a input, but it is very hard to get input from the hash back. So its like a one way thing.

Once you have the "hash" take the first 4 bits of it and add it to the original. Now split that 132 bits number(128 + 4) into 12 sections of 11 bits each. Now lets assign each section to a word, there is a predefined set of dictionary of 2048 words, which maps every single value possible from that 11 bit to a word. Since 2^11 is 2048, you can be sure that for every permutation of 11 bit that you have, there is a word. Example, 00000000000 may map to hello, 11111111111 maps to something else and so on. These mapped 12 words is the recovery code for your account. This makes it easier for recovery but the problem is if someone gets hold of the words, they can steal all of your funds in all of your accounts. So keep it VERY safe.

Now those bits(which were mapped to words) are added with a specific word "mnemonic" and a optional passphrase that you can add if you want to. Then the whole string (132bits+"mnemonic"+passphrase) goes through another "hashing" function which gives out a 512 bit output. This is done 2048 times to slow down brute force attacks if any attackers knows some of your words but not all of it. So it works like hash(hash(string)) 2048 times. A Lot Of Hashing.
So the 2048th hash is your seed phrase.

That seed phrase is used to generate your "master private key" which is the key that you use to "sign" your transactions. Signing your transactions means proving that the transaction indeed came from you and it wasn't someone else. It is just like signing in a cheque, which verifies that it was you who issued it.

To generate the master private key, you AGAIN put the seed phrase into the 512 bit hash and the left 256 bits of it is your private key and the right 256 bit is your master chain code. The private key is used to derive your public key and the chain code is there to provide entropy(randomness) for your child keys.

Now you can use the private key + chain code + index number where index number is just a 32bit number to know the index of the child key that we are deriving to derive a child key. the combined string("private key + chain code + index number") is again put into the 512bit hashing algorithm to get another hash which is split to private key and chain code like before (256 to the left and 256 to the right). This is the child key's private key and chain code for that index. You can do this to generate 4 billion child keys, and 4 billion grandchildren for each child and so on.

So, now that you have unlimited generation of private keys, you can create public keys from them which transform into valid addresses. Every single private key is linked to a public key using a formula: Public key = private Key x G where the G is always known number.

Since, there are so many keys to keep track of, we use a system called paths to do so. Lets call your master private key as m. So the 0th indexed child of it would be m/0 and the 1st indexed child of the child would be m/0/1 and so on. Now there is a specific way to use the keys which is:

```
m / purpose' / coin_type' / account' / change / address_index
```

where every single level that you derive has a specific purpose and it can be modified to suit your needs. You now have the tools to generate as many keys as you want for any chain that you want.