UNIVERSITY NAME

DOCTORAL THESIS

Hierarchical deterministic wallet

Author:
Daniele FORNARO

Supervisor:
Daniele MARAZZINA

A thesis submitted in fulfillment of the requirements for the degree of Mathematical Engeneering

in the

Research Group Name
Department or School Name

January 28, 2018

Declaration of Authorship

I, Daniele FORNARO, declare that this thesis titled, "Hierarchical deterministic wallet" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

UNIVERSITY NAME

Abstract

Faculty Name
Department or School Name

Mathematical Engeneering

Hierarchical deterministic wallet

by Daniele FORNARO

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Physical Constants

Speed of Light $c_0 = 2.997\,924\,58 \times 10^{8}\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

For/Dedicated to/To my…

Chapter 1

Elliptic Curve Geometry

1.1 Introduction

Bitcoin security is based on public and private key cryptograpy. The main concept is that it is simple to compute the public key, knowing the private, but it is infeasible to calculate the private key, knowing the public.

In order to obtain this result a particular Elliptic Curve is used.

1.2 Elliptic Curve over $\mathbb{F}_p$

A point Q, which coordinates are x and y, belong to an Elliptic Curve if and only if Q satisfies the following equation:

$$y^2 = x^3 + ax + b \quad \text{over } \mathbb{F}_p \tag{1.1}$$

Where $\mathbb{F}_p$ is the finite field defined over the set of integers modulo p and a and b are the coefficients of the curve.

We can rewrite the equation 1.1 in the following way:

$$y^2 = x^3 + ax + b \mod p \tag{1.2}$$

Figure 1.1 shows some examples of Elliptic Curve over $\mathbb{F}_p$ with $a = -7$ and $b = 10$

1.2.1 Operations

A point on the Elliptic Curve has some particular properties:

- Symmetry
- Point addition
- Scalar multiplication

Symmetry

For every point in the x axis exists two points in the y axis. Suppose that a point $P(x, y)$ belongs to the Elliptic Curve, then it must satisfy the equation 1.1. So it is easy to prove that the point $Q(x, p - y)$ belongs to the curve too.

Furthermore we have $P = -Q$, from the moment that $P + Q = 0$ (see addition below).

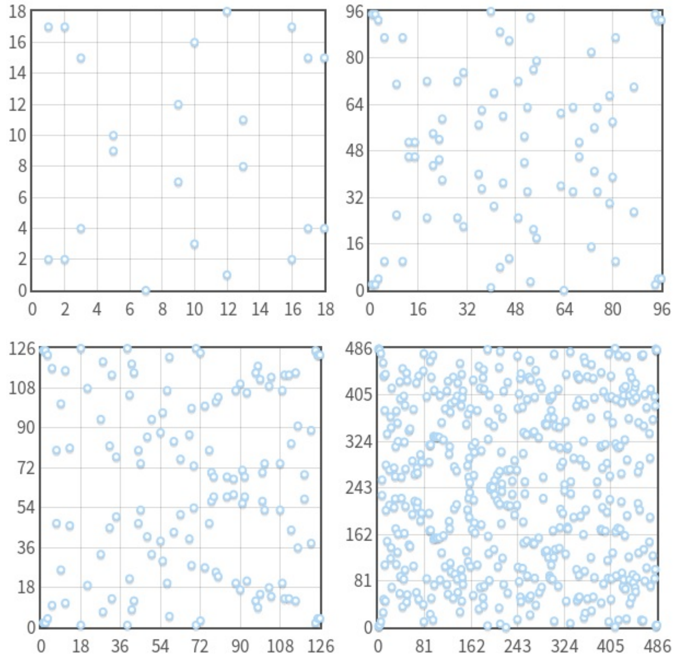

FIGURE 1.1: Points on the Elliptic Curve $y^2 = x^3 - 7x + 10 \mod p$, with $p = 19, 97, 127, 487$

Point addition

We need to change our definition of addition in order to make it works in $\mathbb{F}_p$. In this framework we claim that if three points are aligned over the finite field $\mathbb{F}_p$, then they have zero sum.
So $P + Q = R$ if and only if P, Q and $-R$ are aligned, in the sense shown in figure 1.2

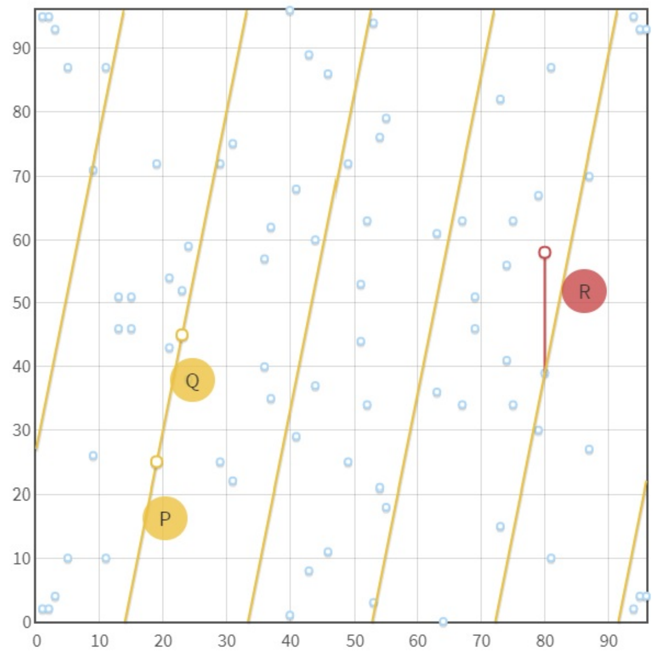

FIGURE 1.2: Elliptic Curve $y^2 = x^3 - 7x + 10 \mod 97$

The equations for calculating point additions are the follow:
Suppose that A and B belong to the Elliptic Curve.

$$A = (x_1, y_1) \quad B = (x_2, y_2)$$

Let's defined $A + B := (x_3, y_3)$
So we have:

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } x_1 \neq x_2 \\ \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{if } x_1 = x_2 \end{cases}$$

$$x_3 = s^2 - x_1 - x_2 \mod p$$
$$y_3 = s(x_1 - x_3) - y_1 \mod p$$

Scalar multiplication

Once defined the addition, any multiplication can be defined as:

$$nP = \underbrace{P + P + \cdots + P}_{n \text{ times}}$$

When n is a very large number can be difficult or even infeasible to compute nP in this way, but we can use the *double and add algorithm* in order to perform multiplication in $\mathcal{O}(\log n)$ steps.

1.2.2 Group order

An elliptic curve defined over a finite field is a group and so it has a finite number of points. This number is called order of the group.
If the prime order is a very large number, it is impossible to count all the point in that field, but there is an algorithm that allows to calculate the order of a group in a fast and efficient way: *Schoof's algorithm*.

Cyclic subgroups

Let's consider a generic point P, we have:

$$nP + mP = \underbrace{P + \cdots + P}_{n \text{ times}} + \underbrace{P + \cdots + P}_{m \text{ times}} = \underbrace{P + \cdots + P}_{n+m \text{ times}} = (n+m)P$$

So multiple of P are closed under addition and this is enough to prove that the set of the multiples of P is a cyclic subgroup of the group formed by the elliptic curve.

The point P is called **generator** of the cyclic subgroup.

Remark *The order of P is linked to the order of the elliptic curve by Lagrange's theorem, which states that the order of a subgroup is a divisor of the order of the parent group.*

Remark *If the order of the group is a prime number, all the point P generate a subgroup with the same order of the group.*

1.3 Bitcoin private-public key cryptography

1.3.1 Bitcoin Elliptic Curve

Bitcoin uses a specific Elliptic Curve defined over the finite field of the natural numbers, where $a = 0$ and $b = 7$.

The equation 1.1 becomes:

$$y^2 = x^3 + 7 \mod p \tag{1.3}$$

The *mod p* (modulo prime number) indicates that this curve is over a finite field of prime order $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

The *order* of this Elliptic Curve is a very large prime number, close to 2^{256}, but smaller then p.

Let's consider a particular point G, called generator, with:

$$x = 79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798$$
$$y = 483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8$$

From the moment that the order of the group is a prime number, the order of any subgroup is equal to the order of the entire group. In particular the order of the subgroup generated by G is equal to *order*.

Definition *A private key is a number choosen in the range beetwen 1 and* ***order****.*

Definition *A public key W is a point in the Bitcoin EC, derived from a private key k in the following way:*

$$W = k \cdot G \tag{1.4}$$

Where the multiplication between k and G is defined in the previous chapter.

This is a *one way* function, in the sense that computing the scalar multiplication, knowing the private key is simple, but make the reverse is infeasible.

Remark *It is infeasible to calculate a private key knowing the public key.*

1.3.2 Bitcoin keys rapresentation and addresses

In order to make it easy to store and recognise keys, some encods were designed.

A public key, a point in the EC, can be rapresented in two way: *uncompressed* or *compressed*.

Uncompressed public key

An uncompressed public key is rapresented in hexadecimal digits, and it is obtained simply concatenating the x coordinate with the y coordinate and adding 04 at the beginning, for a total of 130 hexadecimal digits.

Example of an uncompressed public key:
0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B235
22CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6

Compressed public key

A compressed public key is obtained simply taking the x coordinate and adding 02 at the begging if the y coordinate is even, 03 otherwise.
This is due to the *symmetry properties* of a point of the EC.

Example of a public key compressed:
0250863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B2352

WIF Private Key

WIF stands for wallet import format and is the standard way used to write down a private key.

- Add a version number (80 for Bitcoin) in front of the private key, in order to recognize quickly for what cryptocurrency that private key was used.
- Add 01 at the end of the private key if you want a WIF *compressed*, none if you want a WIF *uncompressed*. The difference between these two types is that from a *compressed* private key a *compressed* public key is expected and from a *uncompressed* private key a *uncompressed* public key is expected.
- Addd a checksum at the end, obtained applying the SHA256 function twice to the string previously obtained, take the first 4 bytes (8 hexadecimal digits) and put them at the end of the string.
- Compute the Base58Encode, obtaining a 52 digit string.

Example of private key WIF:
KwdMAjGmerYanjeui5SHS7JkmpZvVipYvB2LJGU1ZxJwYvP98617

Address

Among the Bitcoin transactions, one of the most used is a *Pay-to-PubkeyHash*, meaning that in the transaction you will not write directly the public key, but the hash of that public key.
The hash function used in this freamwork is the HASH160 function, applied to the *compressed* public key. This is an irreversible procedure, so you cannot obtain the public key from the public key hash.

In order to obtain a valid Bitcoin address, it is needed to encode the *PubkeyHash* in base58, adding first the version in front, the checksum at the end and then encode everything with Base58Encode, obtaining a 34 digit string.

Example of an Address:
1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2

Chapter 2

Wallet

A Bitcoin wallet is a structure used to store keys.

There are different type of wallet:

- Nondeterministic (*random*) Wallet
- Deterministic Wallet

Remark *Bitcoin wallets contains keys, not coins. Coins are in the Blockchain.*

2.1 Nondeterministic (*random*) Wallet

A nondeterministic wallet is the simplest type of wallet. Each Key is randomly and independently generated.

(i) Consider a *Discrete Uniform Random Variable*

$$X \sim \mathcal{U}(S)$$

Where S is the finite set of natural number in the range from 1 to *order*.

(ii) Take some realizations $k_1, k_2 ... k_n$ of X using enough entropy to make these numbesr (*private keys*) impossible to guess.

$$k_1 = X(\omega_1) \quad k_2 = X(\omega_2) \quad ... \quad k_n = X(\omega_n)$$

(iii) Go back to point (i) every time new *private keys* are needed.

With this procedure it is impossible to compute the *public key* without having already the *private key*.

Pros and Cons

Let's focus on the good and bad aspects of this wallet.

Random Wallet	
Pros	**Cons**
• Easy to implement	• Difficult to find real new entropy for every new *private key*. • Every time new *private keys* are needed, you need to make new back up. • Difficult to store or back up in a *non digital way*. Awkward to write it down all yours keys on a paper.

The use of *random wallet* is strongly discouraged for anything other than simple test. There are no good reason to use it.

2.2 Deterministic Wallets

A deterministic wallet is a more sophisticated one, in which every key is generated from a common "*seed*". This means that knowing the *seed* means also to know all the keys in the wallet.

There are different types of deterministic wallets, in this text we will analyze three main types:

- Deterministic Wallet *type 1*
- Deterministic Wallet *type 2*
- Deterministic Wallet *type 3*

These wallet are in increasing order of complexity.

2.2.1 Deterministic Wallet *type-1*

The Deterministic Wallet *type-1* is one of the simplest Wallet among the deterministic ones. Each key is generated adding a number in a sequential order to the *seed* and then computing an *hash* function such as the **SHA256** function.

Let's see how to generate the n^{th} private key:

(i) Generate a *seed* (only once), a random number from a *Discrete Uniform Random Variable*

$$seed = X(\omega) \qquad X \sim \mathcal{U}(S)$$

Where S is the finite set of natural number in the range from 1 to *order*.

(ii) Consider the numbers *seed* and n as strings and concatenate n to *seed*, obtaining a $value$

$$value = seed|n$$

(iii) Compute the SHA256 function to $value$ and obtain the n^{th} *private key*.

(iv) Go back to point (ii) every time new *private keys* are needed with $n = n + 1$.

With this procedure it is impossible to compute the *public key* without having already computed the *private key*.

Pros and Cons

Let's focus on the good and bad aspects of this wallet.

Deterministic Wallet type-1	
Pros	**Cons**
• In order to make a back up of the entire wallet it is needed to store the *seed* only. All *private keys* can be derived from it. • A single back up is needed. • The *seed* can be stored easily also in a *non digital way*, in a paper for example.	• Every time new *public keys* are needed, you need to use the *seed*, to compute new *private keys* and then derive the *public* ones. This could compromise the entire wallet, if the *seed* is used in a non safe environment. • There is only a *key* sequence. No way to distinguish the "purpose" of each *private key*.

The use of this type of wallet is not recommended for everyday use, but it could be used to store Bitcoin in a safe place: *cold wallet*.

2.2.2 Deterministic Wallet *type*-2

The Deterministic Wallet *type*-2 is more sophisticated. Each *private key* is generated in such a way that it is possible to compute the respective *public key* without knowing the *private*.

First let's introduce the necessary ingredients:

◇ **Master private key** (mp): a random number, generated from a *Discrete Uniform Random Variable*

$$mp = X(\omega) \qquad X \sim \mathcal{U}(S)$$

Where S is the finite set of natural number in the range from 1 to *order*.
The master private key must be take secret.

◇ **Master public key** (MP): a point on the EC, obtained from the mp:

$$MP = mp \cdot G$$

Where G is the *generator*.
This point can be consider non-secret.

◇ **Public random number** (r): a random number, generated from a *Discrete Uniform Random Variable*

$$r = X(\omega) \qquad X \sim \mathcal{U}(S)$$

This number can be consider non-secret.

Let's see how to generate the n^{th} private key: p_n

(i) Compute the SHA256 function to the concatenation of r with n, considered as string:

$$h_{n|r} = SHA256(n|r)$$

$h_{n|r}$ can be consider non-secret, from the moment that it is derived from non secret information.

(ii) Compute the n^{th} *private key* adding mp to $h_{n|r}$:

$$p_n = mp + h_{n|r} \qquad \mod (order)$$

In order to obtain the corresponding *public key* P_n, it is possible to compute the standard multiplication:

$$P_n = p_n \cdot G$$

It is also possible to compute P_n without knowing p_n, using only non-secret information: $h_{n|r}$ and MP.

(i) Compute V:

$$V = h_{n|r} \cdot G$$

V can be see as the *public key* of $h_{n|r}$ and can be consider non-secret.

(ii) Add MP to V:

$$P_n = MP + V$$

Where the sum in this contest is the one defined between two point in the EC.

It is easy to prove that P_n can be computed in these two way:

$$\begin{aligned} P_n &= p_n \cdot G \\ &= (mp + h_{n|r}) \cdot G \\ &= (mp \cdot G) + (h_{n|r} \cdot G) \\ &= MP + V \end{aligned}$$

Pros and Cons

Let's focus on the good and bad aspects of this wallet.

Deterministic Wallet type-2	
Pros	**Cons**
• In order to make a back up of the entire wallet it is needed to store the *master private key* and the random number *r*. All *private keys* can be derived from them. • A single back up is needed. • The *master private key* can be stored easily also in a *non digital way*, in a paper for example. • It is possible to derive a new *public key* using only non-secret information, with the procedure above.	• There is only a *key* sequence. No way to distinguish the "purpose" of each *private key*.

The *type-2 deterministic wallet* it is an improvement of the *type-1* because it has the same benefits (except for the need to back up two number instead of only one), but with a great advantage: it is possible to generate new addresses also in a non safe environment, having only r and MP.
A thief can only steal your privacy, because if MP and r are stolen, he is not able to make any Bitcoin transactions from your wallet, but he can see all the previous transactions and the total amount of Bitcoin stored in the wallet.

2.2.3 Deterministic Wallet *type-3*

Deterministic Wallet *type-3* is the most elaborate among the ones considered. Starting from a *seed* it is possible to obtain different *keys* in a hierarchical way, with a structure similar to a tree.

Let's see roughly how an this Wallet works:

(i) Generate a *seed*, a random number from a *Discrete Uniform Random Variable*, unique for each wallet.

$$seed = X(\omega) \qquad X \sim \mathcal{U}(S)$$

Where S is a finite set of natural number.

(ii) Generate a *master private key* from the *seed*, using a stretching function: PBKDF2.

(iii) From this *master private key* it is possible to generate 2^8 *private key* using a irreversible function: HASH512

(iv) Everyone of this *private key* "children" can derive 2^8 *private key* and all of these "grandchildren" can derive as many.

This procedure can produce a huge number of keys. They seem independent from an outside point of view: it is impossible to guess that two private-public key are derived from the same *seed*.

This particular type of Wallet is commonly known as **Hierarchical Deterministic Wallet**, one of the most used and widespread.

In the next chapter we will see in detail how it works.

Chapter 3

Hierarchical Deterministic Wallet

In this chapter we will see how an HD wallet works.

3.1 Elements

Let's focus on the main elements of the Wallet:

- ◇ Seed
- ◇ Extended keys

3.1.1 Seed

The entire Wallet is based on a *seed*.

It is a number taken from a *Discrete Uniform Random Variable*

$$seed = X(\omega) \qquad X \sim \mathcal{U}(S)$$

Where S is the finite set of natural number in the range from 1 to an arbitrary value. Obviously the greater the set from which the number can be extracted, the better it is for the security of the seed itself.

This is an example of seed expressed in hexadecimal format:
seed=fffcf9f6f3f0edeae7e4e1dedbd8d5d2cfccc9c6c3c0bdbab7b4b1aeaba8a5a29f9c999
693908d8a8784817e7b7875726f6c696663605d5a5754514e4b484542

3.1.2 Extended Key

An Extended Key is a sequence of bytes, encoded in base58. It contains all the information necessary for the derivation. When the derivation is made for the first time from the seed, the extended key is called master key.

Once it is decoded we will obtain exactly 78 bytes, with a specific meaning and order:

- ⊛ 4 bytes are used to specified the **version**.
- ⊛ 1 byte is used to specified the **depth** in the hierarchical tree: the extended key derived directly from the seed has $depth = 0$, its first children have $depth = 1$, grandchildren have $depth = 2$ and so on.

- 4 bytes are used for the **fingerprint**. It is a unique value that identify the parent. Compute the HASH160 function on the "parent" public key in a compressed form and then take the first 4 bytes, this is the fingerprint of the child:

$$fingerprint = HASH160(\text{parent public key})[0:4]$$

  Where $[0:4]$ is a python notation.
  For the master key the fingerprint is formed by 4 zeros bytes: $fingerprint = 0000000000$

- 4 bytes are used to specified the **index** of the child.
  For the master key the index is formed by 4 zeros bytes: $index = 0000000000$

- 32 bytes are used for the **chain code**. The chain code is used in order to introduce a sort of entropy in the children generation. We will see below how it works.

- 33 bytes are used for the **key**. It can be *private* or *public*.
  Public key is expressed in compact form, so the first byte is always 02 or 03. The first byte of the private key is always 00 in order to distinguish the key from the public one.

An extended key is called **Extended Private Key** if the lasts 33 bytes are used to specify the private key; it is called **Extended Public Key** if they are used to specify the public key.

For the Bitcoin mainnet it is used for the **version**: $0x0488ADE4$ for an extended private key, $0x0488B21E$ for an extended public key. When this bytes are encoded in base58, they returns *xprv* and *xpub* respectively.

3.2 From SEED to Master Private Key

In this section we will see in detail how it is possible to switch from a *seed* to a *master private key*.

First of all we need to convert the seed into a string of bytes, where the most significant bytes come first (big endian). In order to do so, we need to know how much long we want the string of bytes.

Let's see a practical example:

$$\begin{aligned}
byte_string_1 &= 00\ 00\ 00\ 01 \\
byte_string_2 &= 00\ 00\ 01 \\
byte_string_3 &= 00\ 01 \\
byte_string_4 &= 01
\end{aligned}$$

These 4 byte strings are obtained from the same seed: $seed = 1$ and the only different is the length of the string.

Remark *Different length of string produce different master private key, even if the seed is the same number.*

In python:

```
byte_string = seed.to_bytes(seed_bytes, 'big')
```

Where *seed* is a *int*, *seed_bytes* is the number of bytes that the *byte_string* should have.

It is essential to specify the length of the byte string, otherwise there will be obtained different wallets.

Once we obtain a string of bytes, we will compute the HMAC algorithm. The hash function used for HMAC is the SHA512 and the *key* is a particular string of bytes: *b"Bitcoin seed"*. In python the implementation is the follow:

```
from hashlib import sha512
from hmac import HMAC

hashValue = HMAC(b"Bitcoin seed", byte_string, sha512).digest()
```

Where *.digest()* is used in order to return a string of bytes.

```
from hashlib import sha512
from hmac import HMAC

hashValue = HMAC(b"Bitcoin seed", seed.to_bytes(seed_bytes, 'big'), sha512
    ).digest()
p_bytes = hashValue[:32]
p = int(p_bytes.hex(), 16) % order
p_bytes = b'\x00' + p.to_bytes(32, 'big')
chain_code = hashValue[32:]
```

3.3 Child Key derivation

Chapter 4

Child Key Derivation

4.1 Functional explanation

4.2 Normal derivation

4.2.1 Derive public child from public parent

4.2.2 Possible Risk

4.3 Hardened derivation

Chapter 5

Mnemonic to Seed

5.1 Functional explanation

5.2 BIP 39 derivation

5.2.1 Mnemonic generation

5.2.2 Seed derivation

5.3 Electrum derivation

5.3.1 Mnemonic generation

5.3.2 Seed derivation

5.4 BIP39 vs Electrum derivation

Chapter 6

How to use a HD Wallet

6.1 Multi-coin wallet BIP 44

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

`\hypersetup{urlcolor=red}`, or

`\hypersetup{citecolor=green}`, or

`\hypersetup{allcolor=blue}`.

If you want to completely hide the links, you can use:

`\hypersetup{allcolors=.}`, or even better:

`\hypersetup{hidelinks}`.

If you want to have obvious links in the PDF but not the printed text, use:

`\hypersetup{colorlinks=false}`.

Bibliography

Arnold, A. S. et al. (Mar. 1998). “A Simple Extended-Cavity Diode Laser”. In: *Review of Scientific Instruments* 69.3, pp. 1236–1239. URL: http://link.aip.org/link/?RSI/69/1236/1.

Hawthorn, C. J., K. P. Weber, and R. E. Scholten (Dec. 2001). “Littrow Configuration Tunable External Cavity Diode Laser with Fixed Direction Output Beam”. In: *Review of Scientific Instruments* 72.12, pp. 4477–4479. URL: http://link.aip.org/link/?RSI/72/4477/1.

Wieman, Carl E. and Leo Hollberg (Jan. 1991). “Using Diode Lasers for Atomic Physics”. In: *Review of Scientific Instruments* 62.1, pp. 1–20. URL: http://link.aip.org/link/?RSI/62/1/1.