KAPITEL 5

Wallets

Das Wort »Wallet« wird verwendet, um eine Reihe verschiedener Dinge bei Bitcoin zu beschreiben.

Aus einer übergeordneten Perspektive betrachtet, ist eine Wallet eine Anwendung, die als primäre Nutzerschnittstelle fungiert. Die Wallet kontrolliert den Zugriff auf das Geld des Nutzers, verwaltet Schlüssel und Adressen, hält den Kontostand nach und erzeugt und signiert Transaktionen.

Aus der etwas enger definierten Sicht eines Programmierers steht das Wort »Wallet« für die Datenstruktur, die zur Speicherung und Verwaltung des Schlüssels eines Benutzers verwendet wird.

In diesem Kapitel betrachten wir die zweite Sichtweise, in der Wallets Container für private Schlüssel sind, die üblicherweise in Form strukturierter Dateien oder einfacher Datenbanken implementiert werden.

Wallet-Technologie in der Übersicht

In diesem Abschnitt fassen wir die verschiedenen Technologien zusammen, die zum Aufbau benutzerfreundlicher, sicherer und flexibler Bitcoin-Wallets eingesetzt werden.

Ein gängiger Irrglaube in Bezug auf Bitcoin ist, dass Bitcoin-Wallets Bitcoins enthalten. Tatsächlich enthält die Wallet nur Schlüssel. Die »Coins« werden über das Bitcoin-Netzwerk in der Blockchain festgehalten. Nutzer kontrollieren die Coins im Netzwerk, indem sie Transaktionen mit den Schlüsseln in ihren Wallets signieren. So gesehen, ist eine Bitcoin-Wallet eher so etwas wie ein Schlüsselbund.

Bitcoin-Wallets enthalten Schlüssel, keine Coins. Jeder Nutzer besitzt eine Wallet, die diese Schlüssel enthält. Wallets sind »Schlüsselbunde« mit Paaren privater/öffentlicher Schlüssel (siehe »Private und öffentliche Schlüssel« auf Seite 59). Nutzer signieren Transaktionen mit Schlüsseln, wodurch sie den Besitz an den Transaktions-Outputs (ihren Coins) beweisen. Die Coins werden in der Blockchain in Form von Transaktions-Outputs (oft `vout` oder `txout` genannt) gespeichert.

Es gibt zwei wesentliche Arten von Wallets, die sich dadurch unterscheiden, ob ihre Schlüssel miteinander in Beziehung stehen oder nicht.

Der erste Typ ist die *nichtdeterministische Wallet*, bei der jeder Schlüssel unabhängig voneinander aus einer Zufallszahl generiert wird. Die Schlüssel stehen in keinerlei Beziehung zueinander. Diese Art Wallet ist als JBOK-Wallet bekannt, was für *Just a Bunch Of Keys* steht, zu Deutsch etwa »nur ein Haufen Schlüssel«.

Der zweite Typ ist die *deterministische Wallet*, bei der die Schlüssel aus einem einzelnen Master-Schlüssel abgeleitet werden, der als *Seed* bezeichnet wird. Alle Schlüssel dieses Wallet-Typs sind miteinander verwandt und können erneut generiert werden, wenn man den ursprünglichen Seed-Wert kennt. Es gibt eine Reihe verschiedener Methoden zur Schlüsselableitung, die bei deterministischen Wallets verwendet werden. Die am weitesten verbreitete Methode zur Ableitung von Schlüsseln verwendet eine baumartige Struktur und ist als *hierarchisch deterministische* oder kurz *HD-Wallet* bekannt.

Deterministische Wallets werden über einen sogenannten Seed-Wert initialisiert. Um ihre Verwendung zu vereinfachen, werden Seeds in Form englischer Wörter codiert, den *mnemonischen Codewörtern* (Mnemonic Code Words).

Die nächsten Abschnitte führen auf hohem Niveau in diese Technologien ein.

Nichtdeterministische (zufallsbasierte) Wallets

Bei der ersten Bitcoin-Wallet (die jetzt Bitcoin Core heißt) waren Wallets Ansammlungen zufällig generierter privater Schlüssel. Beispielsweise hat der Original-Bitcoin-Core-Client beim ersten Programmstart 100 zufällige private Schlüssel vorgeneriert und weitere Schlüssel bei Bedarf erzeugt. Jeder dieser Schlüssel wurde nur einmal verwendet. Solche Wallets wurden durch deterministische Wallets ersetzt, da man sie nur schwer verwalten, sichern und importieren kann. Der Nachteil zufälliger Schlüssel ist, dass man viele generieren und Kopien aller Schlüssel vorhalten muss, d.h., die Wallet muss häufig gesichert werden. Jeder einzelne Schlüssel muss gesichert werden, oder die mit ihm verknüpften Mittel sind unwiederbringlich verloren, wenn man nicht mehr auf die Wallet zugreifen kann. Das widerspricht dem Prinzip, Adressen nicht wiederzuverwenden. Die Wiederverwendung von Adressen schränkt die Privatsphäre ein, weil mehrere Transaktionen und Adressen miteinander verknüpft werden. Eine nichtdeterministische Typ-0-Wallet ist für eine Wallet eine schlechte Wahl, insbesondere wenn Sie Adressen nicht wiederverwenden wollen, weil Sie viele Schlüssel verwalten müssen, was wiederum bedeutet, dass Sie häufig Backups anlegen müssen. Zwar enthält der Bitcoin-Core-Client eine Typ-0-Wallet, von deren Nutzung wird von den Bitcoin-Core-Entwicklern aber abgeraten. Abbil-

dung 5-1 zeigt eine nichtdeterministische Wallet, die eine lose Sammlung zufälliger Schlüssel enthält.

Außer für einfache Tests sollten Sie nichtdeterministische Wallets nicht verwenden. Sie sind einfach zu umständlich in der Handhabung und Sicherung. Nutzen Sie stattdessen eine auf dem Industriestandard basierende *HD-Wallet* mit *mnemonischem Seed* für das Backup.

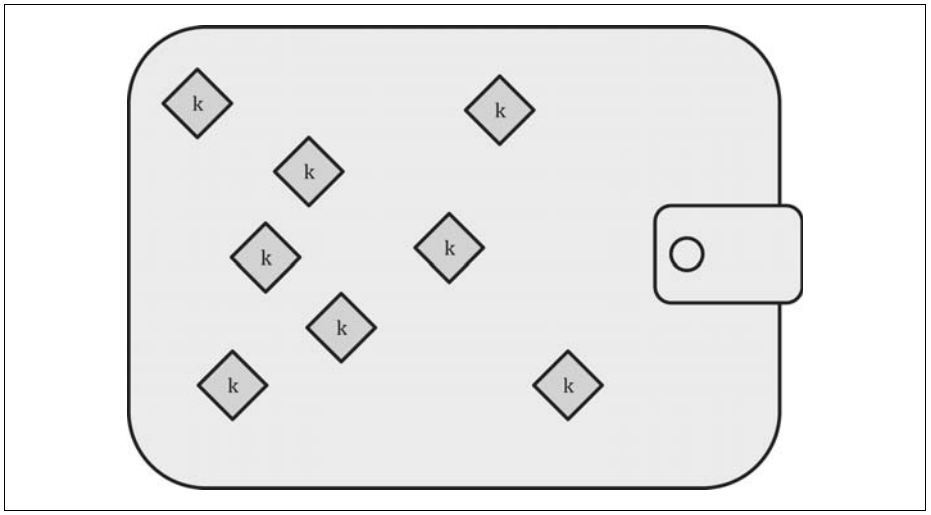

Abbildung 5-1: Nichtdeterministische (zufällige) Typ-0-Wallet: eine Sammlung zufällig generierter Schlüssel

Deterministische (Seed-basierte) Wallets

Bei deterministischen oder *Seed-basierten* Wallets werden die privaten Schlüssel von einem gemeinsamen Seed-Wert (engl. für »Saat«) abgeleitet. Dazu wird ein Einweg-Hash verwendet. Der Seed-Wert ist eine zufällig generierte Zahl, die mit anderen Daten kombiniert wird, etwa einem Index oder einem *Chain-Code* (siehe »HD-Wallets (BIP-32/BIP-44)« auf Seite 98), um die privaten Schlüssel abzuleiten. Bei einer deterministischen Wallet reicht der Seed aus, um alle daraus abgeleiteten Schlüssel wiederherzustellen, und darum genügt ein einmaliges Backup nach der Erstellung. Der Seed-Wert ist auch für den Wallet-Export und -Import ausreichend, was die einfache Migration der Schlüssel eines Nutzers zwischen verschiedenen Wallet-Implementierungen erlaubt. Abbildung 5-2 zeigt das logische Diagramm einer deterministischen Wallet.

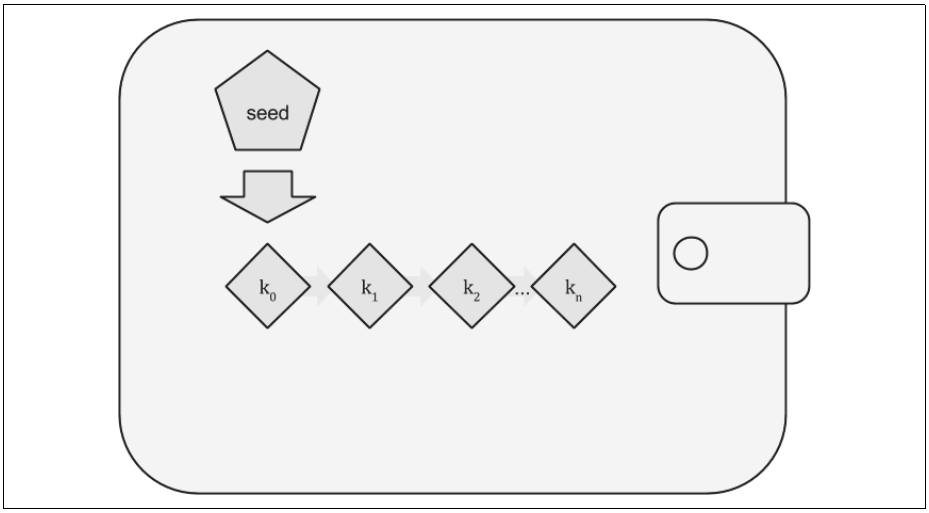

Abbildung 5-2: Deterministische (Seed-basierte) Wallet: eine deterministische Folge von Schlüsseln, abgeleitet aus dem Seed-Wert

HD-Wallets (BIP-32/BIP-44)

Deterministische Wallets wurden entwickelt, um die einfache Erzeugung vieler Schlüssel über einen einzelnen Seed-Wert zu vereinfachen. Die fortgeschrittenste Form der HD-Wallet ist durch den BIP-32-Standard definiert. HD-Wallets enthalten in einer Baumstruktur abgeleitete Schlüssel, bei denen der Parent-Schlüssel eine Folge von Child-Schlüsseln (»Kind-Schlüsseln«) ableiten kann, von denen jeder wiederum eine Folge von »Enkel-Schlüsseln« (engl. Grandchild) ableiten kann – und so weiter bis ins Unendliche. Diese Baumstruktur veranschaulicht Abbildung 5-3.

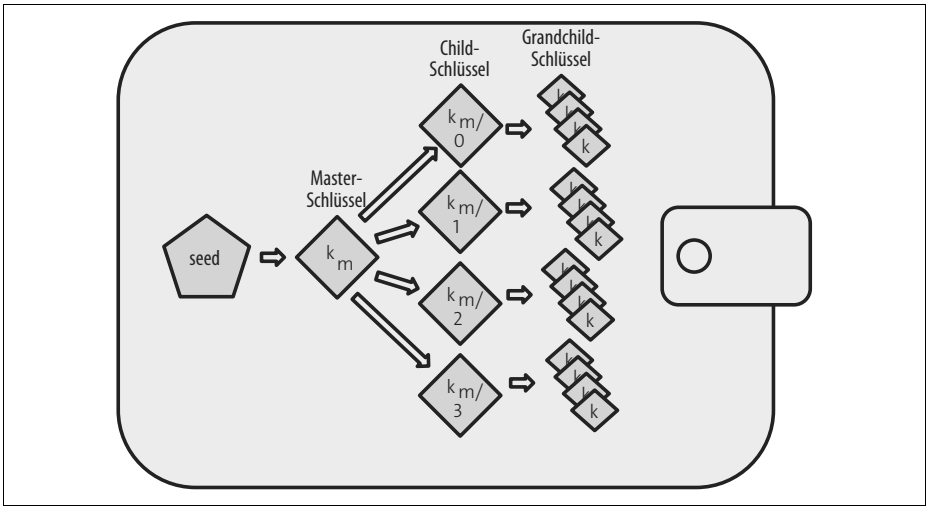

Abbildung 5-3: HD-Wallet (Typ 2): Baum von Schlüsseln, generiert aus einem einzigen Seed-Wert

HD-Wallets bieten gegenüber zufälligen (nichtdeterministischen) Schlüsseln zwei Vorteile. Zum einen kann die Baumstruktur genutzt werden, um eine zusätzliche organisatorische Bedeutung auszudrücken, etwa wenn ein bestimmter Zweig von Subschlüsseln für eingehende Zahlungen verwendet wird und ein anderer Zweig für dem Empfang des Wechselgelds bei ausgehenden Zahlungen. Diese Verzweigung von Schlüsseln kann auch in Unternehmen eingesetzt werden, indem etwa verschiedene Zweige für Abteilungen, Niederlassungen, spezielle Funktionen oder Rechnungslegungskategorien genutzt werden.

Der zweite Vorteil von HD-Wallets besteht darin, dass der Benutzer eine Reihe öffentlicher Schlüssel erzeugen kann, ohne auf die dazugehörigen privaten Schlüssel zugreifen zu müssen. HD-Wallets können daher auf unsicheren Servern oder in reinen Empfangssystemen verwendet werden, die für jede Transaktion einen anderen öffentlichen Schlüssel ausgeben. Die öffentlichen Schlüssel müssen nicht vorab hochgeladen oder berechnet werden, dennoch hat der Server keinen Zugriff auf die privaten Schlüssel, mit denen Guthaben eingelöst werden können.

Seeds und mnemonische Codes (BIP-39)

HD-Wallets sind ein sehr leistungsfähiger Mechanismus zur Verwaltung vieler Schlüssel und Adressen. Sie sind sogar noch nützlicher, wenn man sie mit einem standardisierten Verfahren kombiniert, bei dem Seeds aus einer Folge englischer Wörter erzeugt werden, die man einfach aufschreiben und über Wallets hinweg im- und exportieren kann. Dies wird als *Mnemonic* bezeichnet, und der Standard ist in BIP-39 definiert. Heutzutage verwenden die meisten Wallets (ebenso wie die Wallets anderer Kryptowährungen) diesen Standard und können mithilfe kompatibler Mnemonics Seeds für die (Rück-)Sicherung im- und exportieren.

Sehen wir uns das unter praktischen Gesichtspunkten an. Welcher der folgenden Seeds ist einfacher zu übertragen, auf Papier festzuhalten, fehlerfrei zu lesen oder an Wallets zu ex- und importieren?

```
0C1E24E5917779D297E14D45F14E1A1A

army van defense carry jealous true
garbage claim echo media make crunch
```

Die Wallet-Best-Practices

Während die Bitcoin-Wallet-Technologie gereift ist, haben sich verschiedene gängige Industriestandards entwickelt. Diese Standards sorgen dafür, dass Bitcoin-Wallets größtenteils kompatibel, einfach, sicher und flexibel sind. Diese gängigen Standards sind:

- mnemonische Codewörter, basierend auf BIP-39
- HD-Wallets, basierend BIP-32
- Mehrzweck-HD-Wallet-Strukturen, basierend auf BIP-43

- Wallets mit mehreren Währungen und Konten, basierend auf BIP-44

Diese Standards könnten sich ändern oder durch zukünftige Entwicklungen abgelöst werden, doch im Moment bilden sie einen Satz ineinandergreifender Technologien, der zum De-facto-Wallet-Standard für Bitcoin geworden ist.

Diese Standards wurden von einer Vielzahl von Software- und Hardware-Bitcoin-Wallets übernommen, wodurch all diese Wallets kompatibel sind. Ein Nutzer kann ein Mnemonic, das er auf einer dieser Wallets erzeugt hat, in eine andere Wallet importieren und alle Transaktionen, Schlüssel und Adressen wiederherstellen.

Einige Beispiele für Software-Wallets, die diese Standards unterstützen, sind (in alphabetischer Reihenfolge) Breadwallet, Copay, Multibit HD und Mycelium. Die diese Standards unterstützenden Hardware-Wallets sind unter anderem (in alphabetischer Reihenfolge) Keepkey, Ledger und Trezor.

Die folgenden Abschnitte sehen sich diese Technologien im Detail an.

Wenn Sie eine Bitcoin-Wallet implementieren, sollte diese auf einer HD-Wallet aufbauen, mit codierten Seed-Werten als Mnemonischer Code für Backups arbeiten sowie den Standards BIP-32, BIP-39, BIP-43 und BIP-44 folgen, die in den folgenden Abschnitten beschrieben werden.

Eine Bitcoin-Wallet verwenden

In »Bitcoin: Anwendungsfälle, Anwender und deren Geschichten« auf Seite 5 haben wir Gabriel kennengelernt, einen rührigen Teenager aus Rio de Janeiro, der in einem einfachen Webshop Bitcoin-Fanartikel wie T-Shirts, Kaffeebecher und Aufkleber verkauft.

Gabriel nutzt die Bitcoin-Hardware-Wallet Trezor (Abbildung 5-4), um seine Bitcoins sicher verwalten zu können. Der Trezor ist ein einfaches USB-Gerät mit zwei Buttons, der Schlüssel (in Form einer HD-Wallet) speichert und Transaktionen signiert. Trezor-Wallets implementieren alle in diesem Kapitel behandelten Industriestandards, d. h., Gabriel ist weder von einer proprietären Technologie noch von einem einzelnen Anbieter abhängig.

Abbildung 5-4: Trezor: eine Bitcoin-HD-Wallet als Hardware

Als Gabriel Trezor zum ersten Mal verwendete, generierte das Gerät ein Mnemonic und einen Seed-Wert über einen eingebauten Hardware-Zufallszahlengenerator. Während dieser Initialisierungsphase gab die Wallet eine nummerierte Folge von Wörtern (eines nach dem anderen) auf dem Display aus (siehe Abbildung 5-5).

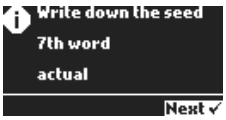

Abbildung 5-5: Trezor zeigt ein Mnemonic auf dem Display an.

Durch Aufschreiben dieser Mnemonics legt Gabriel eine Sicherung an (siehe Tabelle 5-1), die zur Wiederherstellung verwendet werden kann, falls der Trezor verloren oder defekt ist. Dieses Mnemonic kann zur Wiederherstellung mit einem neuen Trezor oder mit einer der vielen anderen kompatiblen Soft- und Hardware-Wallets verwendet werden. Beachten Sie, dass die Reihenfolge dieser Wörter wichtig ist, sie sind deshalb nummeriert. Gabriel muss jedes Wort in der angegebenen Reihenfolge festhalten, um die richtige Wortfolge zu erhalten.

Tabelle 5-1: Gabriels Papier-Backup des Mnemonic

1.	*army*	**7.**	*garbage*
2.	*van*	**8.**	*claim*
3.	*defense*	**9.**	*echo*
4.	*carry*	**10.**	*media*
5.	*jealous*	**11.**	*make*
6.	*true*	**12.**	*crunch*

Der Einfachheit halber zeigt Tabelle 5-1 ein 12-Wort-Mnemonic. Tatsächlich generieren die meisten Hardware-Wallets ein sichereres 24-Wort-Mnemonic. Die Verwendung des Mnemonic ist unabhängig von der Länge immer gleich.

Für seinen ersten Webshop hat Gabriel eine einzige Bitcoin-Adresse genutzt, die er mit seinem Trezor erzeugt hat. Diese Adresse wird von allen Kunden für alle Bestellungen verwendet. Wie wir sehen werden, hat dieser Ansatz einige Nachteile und lässt sich mit einer HD-Wallet verbessern.

Details der Wallet-Technologie

Sehen wir uns jedes dieser wichtigen Industriestandards im Detail an, die von so vielen Bitcoin-Wallets verwendet werden.

Mnemonische Codewörter (BIP-39)

Mnemonische Codewörter sind Wortfolgen, die eine Zufallszahl repräsentieren (codieren), die als Seed-Wert genutzt wird, um eine deterministische Wallet abzuleiten. Die Reihenfolge der Wörter reicht aus, um den Seed-Wert und daraus dann die Wallet und alle abgeleiteten Schlüssel wiederherzustellen. Eine Wallet-Anwendung, die deterministische Wallets mit mnemonischen Wörtern implementiert, präsentiert dem Nutzer eine Folge von 12 bis 24 Wörtern, wenn er eine Wallet anlegt. Diese Wortfolge ist das Wallet-Backup und kann zur Wiederherstellung der Wallet sowie aller Schlüssel verwendet werden. Das funktioniert mit jeder kompatiblen Wallet-Anwendung. Mnemonische Wörter machen es den Benutzern einfacher, die Wallets zu sichern, da sie im Gegensatz zu einer zufälligen Zahlenfolge besser gelesen und aufgeschrieben werden können.

Mnemonische Wörter werden häufig mit *Brainwallets* verwechselt, sind aber nicht identisch. Der wesentliche Unterschied besteht darin, dass eine Brainwallet (zu Deutsch »Gehirn-Wallet«) aus vom Nutzer gewählten Wörtern besteht, während mnemonische Wörter von der Wallet zufällig erzeugt werden. Dieser wesentliche Unterschied macht mnemonische Wörter wesentlich sicherer, weil Menschen eine schlechte Quelle für den »Zufall« darstellen.

Mnemonische Codes sind in BIP-39 (siehe Anhang C) definiert. Beachten Sie, dass BIP-39 eine Implementierung des Standards für mnemonische Codes ist. Es gibt einen anderen Standard (mit einem anderen Wortschatz), der von der Electrum-Wallet genutzt wird und älter ist als BIP-39. BIP-39 wurde von dem Unternehmen hinter der Trezor-Hardware-Wallet vorgeschlagen und ist mit Electrums Implementierung nicht kompatibel. Allerdings genießt BIP-39 breite Unterstützung durch die Industrie mit Dutzenden kompatiblen Implementierungen und kann daher als Defacto-Standard betrachtet werden.

BIP-39 definiert die Erzeugung eines mnemonischen Codes und eines Seed-Werts, den wir hier in neun Schritten erläutern wollen. Der Übersichtlichkeit halber wird der Prozess in zwei Teilen beschrieben: Die Schritte 1 bis 6 finden Sie in »Mnemonische Wörter erzeugen« auf Seite 102, die Schritte 7 bis 9 in »Vom Mnemonic zum Seed-Wert« auf Seite 104.

Mnemonische Wörter erzeugen

Mnemonische Wörter werden durch die Wallet automatisch generiert. Diese nutzt dazu einen standardisierten Prozess, der in BIP-39 definiert ist. Die Wallet startet mit einer Entropiequelle, fügt eine Prüfsumme hinzu und bildet die Entropie auf eine Wortliste ab:

1. Erzeuge eine zufällige Folge (Entropie) von 128 bis 256 Bits.
2. Erzeuge eine Prüfsumme der zufälligen Folge über einen SHA256-Hash der ersten Bits (Entropielänge/32).

3. Hänge die Prüfsumme an das Ende der Folge an.
4. Teile die Folge in Abschnitte von 11 Bits.
5. Bilde jeden 11-Bit-Wert auf ein Wort aus einem Wörterbuch mit 2.048 Wörtern ab.
6. Der mnemonische Code ist die Folge der Wörter.

Abbildung 5-6 zeigt, wie Entropie genutzt wird, um mnemonische Wörter zu erzeugen.

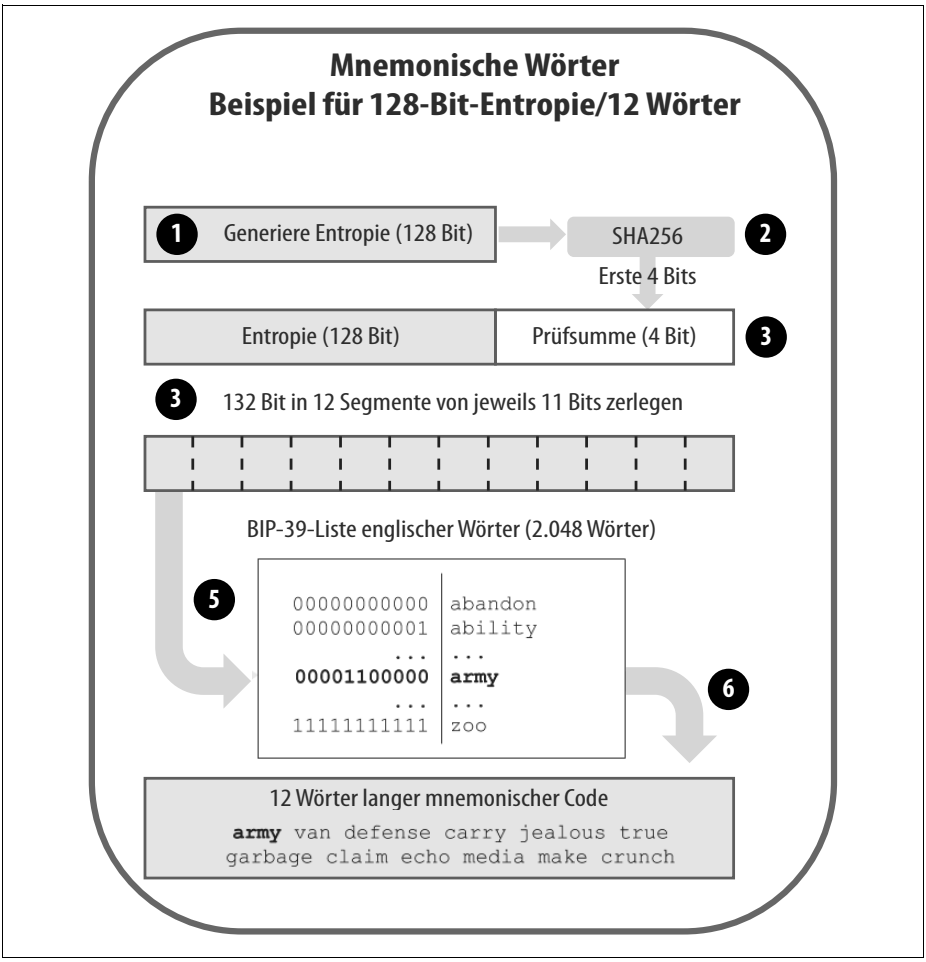

Abbildung 5-6: Entropie generieren und als mnemonische Wörter codieren

Tabelle 5-2 zeigt die Beziehung zwischen der Größe der Entropiedaten und der Länge mnemonischer Codes in Wörtern.

Tabelle 5-2: Mnemonische Codes: Entropie und Wortlänge

Entropie (Bits)	Prüfsumme (Bits)	Entropie + Prüfsumme (Bits)	Mnemonische Länge (Wörter)
128	4	132	12
160	5	165	15
192	6	198	18
224	7	231	21
256	8	264	24

Vom Mnemonic zum Seed-Wert

Die mnemonischen Wörter repräsentieren Entropie mit einer Länge zwischen 128 und 256 Bit. Die Entropie wird dann genutzt, um über die Funktion PBKDF2 den Seed-Wert auf 512 Bit zu strecken. Der so erzeugte Seed-Wert wird dann verwendet, um eine deterministische Wallet zu erzeugen und dessen Schlüssel abzuleiten.

Die den Schlüssel streckende Funktion verlangt zwei Parameter, das Mnemonic und einen sogenannten *Salt-Wert*. Die Aufgabe des Salt-Werts besteht darin, den Aufbau einer Lookup-Tabelle für Brute-Force-Angriffe zu erschweren. Im BIP-39-Standard dient der Salt-Wert einem weiteren Zweck: Er erlaubt die Einführung einer Passphrase, die als zusätzlicher Faktor den Seed-Wert schützt. Wir gehen darauf in »Optionale Passphrase bei BIP-39« auf Seite 106 genauer ein.

Der in den Schritten 7 bis 9 beschriebene Prozess knüpft nahtlos an die vorhin in »Mnemonische Wörter erzeugen« auf Seite 102 erläuterten Schritte an:

7. Der erste Parameter für die PBKDF2-Funktion ist das in Schritt 6 erzeugte Mnemonic.
8. Der zweite Parameter für die PBKDF2-Funktion ist ein Salt-Wert. Dieser besteht aus dem konstanten String `mnemonic` und einer vom Benutzer (optional) angegebenen Passphrase.
9. PBKDF2 »streckt« Mnemonic und Salt in 2.048 Hashing-Runden mit dem HMAC-SHA512-Algorithmus und erzeugt als Ergebnis einen 512-Bit-Wert. Dieser 512-Bit-Wert bildet dann den Seed-Wert.

Abbildung 5-7 zeigt ein zur Generierung des Seed-Werts verwendetes Mnemonic.

Die zur Streckung des Schlüssels verwendete Hashfunktion bietet einen sehr effektiven Schutz vor Brute-Force-Angriffen auf das Mnemonic bzw. die Passphrase. Das Ausprobieren von mehr als ein paar Tausend Passphrase-Mnemonic-Kombinationen ist sehr rechenintensiv, während die Zahl möglicher Seed-Werte riesig ist (2^{512}).

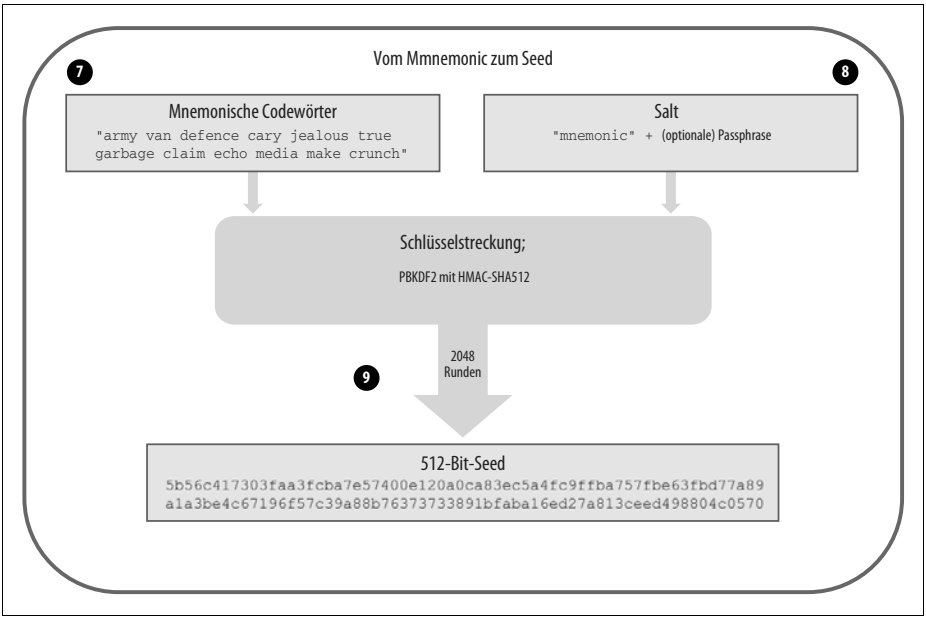

Abbildung 5-7: Vom Mnemonic zum Seed-Wert

Die Tabellen 5-3, 5-4 und 5-5 zeigen einige Beispiele mnemonischer Codes und den daraus ohne Passphrase erzeugten Seed-Werten.

Tabelle 5-3: Mnemonischer 128-Bit-Code, keine Passphrase, resultierender Seed-Wert

Entropie-Input (128 Bits)
`0c1e24e5917779d297e14d45f14e1a1a`
Mnemonic (12 Wörter)
`army van defense carry jealous true garbage claim echo media make crunch`
Passphrase
ohne
Seed (512 Bit)
`5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570`

Tabelle 5-4: Mnemonischer 128-Bit-Code mit Passphrase, resultierender Seed-Wert

Entropie-Input (128 Bit)
`0c1e24e5917779d297e14d45f14e1a1a`
Mnemonic (12 Wörter)
`army van defense carry jealous true garbage claim echo media make crunch`

Tabelle 5-4: Mnemonischer 128-Bit-Code mit Passphrase, resultierender Seed-Wert (Fortsetzung)

Passphrase
SuperDuperSecret
Seed (512 Bit)
3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0715861dc8a18358f80b79d49acf64142ae57037d1d54

Tabelle 5-5: Mnemonischer 256-Bit-Code ohne Passphrase, resultierender Seed-Wert

Entropie-Input (256 Bit)
2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c
Mnemonic (24 Wörter)
cake apple borrow silk endorse fitness top denial coil riot stay wolf luggage oxygen faint major edit measure invite love trap field dilemma oblige
Passphrase
ohne
Seed (512 Bit)
3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d062e55f1e0deaa082df8d487381379df848a6ad7e98798404

Optionale Passphrase bei BIP-39

Der BIP-39-Standard erlaubt bei der Ableitung des Seed-Werts die Verwendung einer optionalen Passphrase. Wird keine Passphrase verwendet, wird das Mnemonic mit einem Salt-Wert mit dem konstanten String `"mnemonic"` gestreckt, und ein spezifischer 512-Bit-Seed-Wert für jedes übergebene Mnemonic wird erzeugt. Wenn eine Passphrase verwendet wird, erzeugt die Funktion einen anderen Seed-Wert für das gleiche Mnemonic. Tatsächlich führt für ein einzelnes Mnemonic jede Passphrase zu einem anderen Seed-Wert. Grundsätzlich gibt es keine »falsche« Passphrase. Alle Passphrasen sind gültig, sie führen zu unterschiedlichen Seed-Werten und bilden so eine riesige Menge möglicher (nicht initialisierter) Wallets. Die Menge möglicher Wallets ist so groß (2^{512}), dass ein Brute-Force-Angriff praktisch unmöglich ist und auch die zufällige Auswahl kaum eine genutzte Wallet ergibt.

Es gibt keine »falschen« Passphrasen bei BIP-39. Jede Passphrase führt zu einer leeren Wallet (wenn sie noch nicht verwendet wurde).

Die optionale Passphrase führt zwei wichtige Features ein:

- Einen zweiten (auswendig gelernten) Faktor, der ein Mnemonic für sich genommen nutzlos macht. Das schützt Backups von Mnemonics vor einer Gefährdung durch Diebe.
- Eine Art »Fake-Wallet«, bei der die gewählte Passphrase zu einer Wallet mit nur geringen Mitteln führt, um einen Angreifer von der »echten« Wallet abzulenken, die den Großteil der Mittel enthält.

Allerdings ist es wichtig, zu bemerken, dass die Verwendung einer Passphrase auch mit dem Risiko des Verlusts behaftet ist:

- Ist der Besitzer der Wallet verstorben oder handlungsunfähig, ist der Seed-Wert nutzlos, und alle in der Wallet gespeicherten Mittel sind für immer verloren.
- Bewahrt der Eigentümer die Passphrase hingegen am gleichen Ort auf wie den Seed-Wert, macht er den eigentlichen Zweck eines zweiten Faktors zunichte.

Zwar sind Passphrasen sehr nützlich, doch sie sollten nur in Kombination mit einem sorgfältig geplanten Backup/Recovery-Prozess eingesetzt werden. Dabei sollte auch das mögliche Ableben des Eigentümers berücksichtigt und eine Nutzung des Vermögens durch die Erben ermöglicht werden.

Mit mnemonischen Codes arbeiten

BIP-39 ist in Form einer Bibliothek für viele verschiedene Programmiersprachen implementiert:

python-mnemonic (https://github.com/trezor/python-mnemonic)
: Die Referenzimplementierung des Standards durch das SatoshiLabs-Team (das BIP-39 vorgeschlagen hat) in Python.

bitcoinjs/bip39 (https://github.com/bitcoinjs/bip39)
: Eine Implementierung von BIP-39 für das beliebte bitcoinJS-Framework in JavaScript.

libbitcoin/mnemonic (https://github.com/libbitcoin/libbitcoin/blob/master/src/wallet/mnemonic.cpp)
: Eine Implementierung von BIP-39 als Teil des beliebten Libbitcoin-Frameworks in C++.

Es gibt auch einen als eigenständige Webseite implementierten BIP-39-Generator, der für Tests und Experimente sehr nützlich ist. Abbildung 5-8 zeigt diese Website, die Mnemonics, Seed-Werte und erweiterte private Schlüssel erzeugt.

Abbildung 5-8: Ein BIP-39-Generator als eigenständige Website

Die Seite (*https://iancoleman.github.io/bip39/*) kann offline in einem Browser genutzt werden, ist aber auch online verfügbar.

Eine HD-Wallet aus dem Seed-Wert erzeugen

HD-Wallets werden aus einem einzigen *Stamm-Seed-Wert (Root Seed)* erzeugt, einer 128, 256 oder 512 Bit langen Zufallszahl. Üblicherweise wird dieser Seed-Wert, wie im vorigen Abschnitt beschrieben, aus einem Mnemonic erzeugt.

Jeder Schlüssel in der HD-Wallet wird deterministisch aus dem Root Seed abgeleitet. Das macht es möglich, die gesamte HD-Wallet aus diesem Seed-Wert auf jeder kompatiblen HD-Wallet wiederherzustellen. Auf diese Weise können Sie HD-Wallets mit Tausenden oder sogar Millionen von Schlüsseln sichern, wiederherstellen sowie exportieren/importieren, indem Sie nur das Mnemonic übertragen, aus dem der Root Seed abgeleitet ist.

Der Prozess, aus dem die Master-Schlüssel und der Master-Chain-Code für eine HD-Wallet erzeugt werden, ist in Abbildung 5-9 zu sehen.

Der Root Seed wird an den HMAC-SHA512-Algorithmus übergeben, und der daraus resultierende Hashwert wird genutzt, um einen *privaten Master-Schlüssel* (m) und einen *Master-Chaincode* (c) zu erzeugen.

Der private Master-Schlüssel (m) erzeugt dann den entsprechenden öffentlichen Master-Schlüssel (M) über die übliche Multiplikation elliptischer Kurven $m * G$, die wir aus »Öffentliche Schlüssel« auf Seite 62 kennen.

Der Chaincode (c) wird genutzt, um Entropie für die Funktion bereitzustellen, die Child-Schlüssel aus Parent-Schlüsseln erzeugt (was im nächsten Abschnitt beschrieben wird).

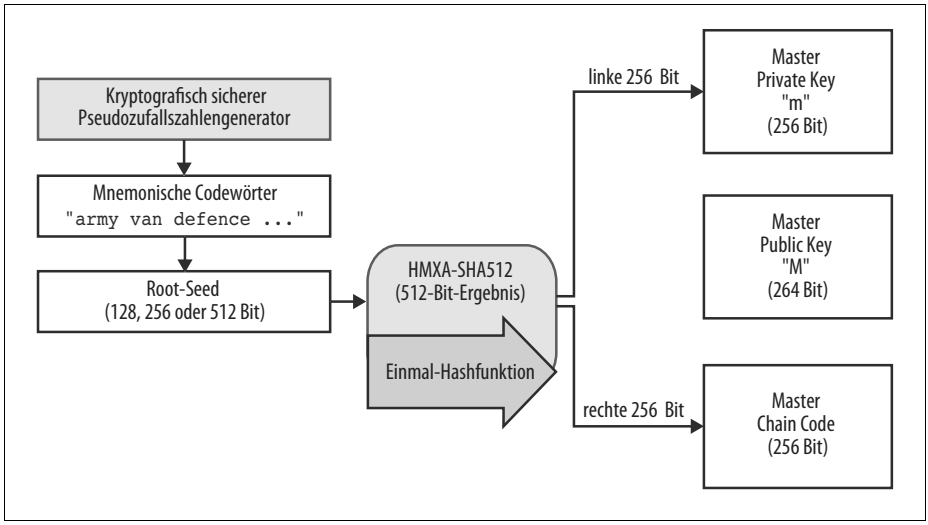

Abbildung 5-9: Master-Schlüssel und Chain-Code aus dem Root Seed erzeugen

Ableitung privater Child-Schlüssel

HD-Wallets verwenden eine Funktion zur Ableitung von Child-Schlüsseln aus Parent-Schlüsseln (*Child Key Derivation*, CKD).

Die CKD-Funktion basiert auf einem Einweg-Hash, der Folgendes kombiniert;

- einen privaten oder öffentlichen Parent-Schlüssel (einen nicht komprimierte, ECDSA-Schlüssel),
- einen als Chaincode (256 Bit) bezeichneten Seed-Wert,
- einen Index (32 Bit).

Der Chaincode führt deterministische zufällige Daten in den Prozess ein, sodass es nicht reicht, den Index und einen Child-Schlüssel zu kennen, um andere Child-Schlüssel ableiten zu können. Wer den Child-Schlüssel kennt, kann dessen Geschwister nicht ableiten, solange er nicht auch den Chaincode kennt. Der initiale Chaincode (am Stamm des Baums) wird aus dem Seed-Wert erzeugt, während nachfolgende Child-Chaincodes aus dem Parent-Chaincode abgeleitet werden.

Diese drei Elemente (Parent-Schlüssel, Chaincode und Index) werden kombiniert in einer Hashfunktion zur Generierung von Child-Schlüsseln verwendet.

Der öffentliche Parent-Schlüssel sowie Chaincode und Index werden kombiniert, und über den HMAC-SHA512-Algorithmus wird daraus ein 512-Bit-Hashwert erzeugt. Dieser 512-Bit-Hash wird in zwei Teile von jeweils 256 Bits halbiert. Die 256 Bits der rechten Hälfte des erzeugten Hash werden zum Chaincode für das

Child. Die linke Hälfte und der Index werden zum privaten Parent-Schlüssel addiert, um den privaten Child-Schlüssel zu erzeugen. In Abbildung 5-10 ist dieser Vorgang zu sehen, wobei der Index 0 verwendet wird, um den ersten Child (nach Index) des Parents zu erzeugen.

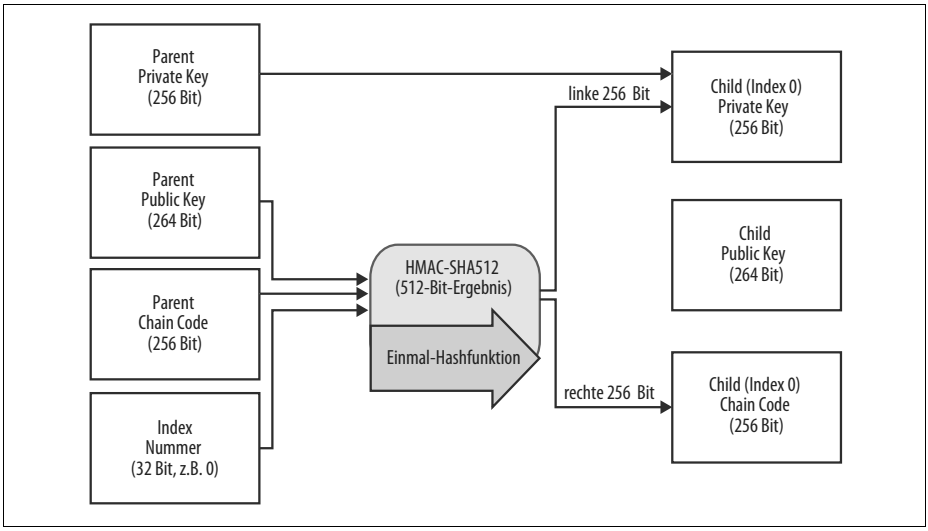

Abbildung 5-10: Privaten Parent-Schlüssel erweitern, um einen privaten Child-Schlüssel zu erzeugen

Durch Änderung des Index können wir den Parent erweitern und nacheinander weitere Children, z.B. Child 0, Child 1, Child 2 etc. Jeder Parent-Schlüssel kann 2.147.483.647 (2^{31}) Children besitzen. (2^{31} ist die Hälfte des gesamten Wertebereichs von 2^{32}. Die andere Hälfte ist für eine spezielle Art der Ableitung reserviert, über die wir später noch sprechen werden.)

Wiederholt man diesen Prozess eine Stufe tiefer, wird jedes Child zu einem Parent und kann eigene Children mit einer unendlichen Reihe von Nachfahren erzeugen.

Abgeleitete Child-Schlüssel verwenden

Private Child-Schlüssel sind von nichtdeterministischen (zufälligen) Schlüsseln nicht zu unterscheiden. Da die Ableitungsfunktion eine Einwegfunktion ist, kann der Child-Schlüssel nicht verwendet werden, um den Parent-Schlüssel zu ermitteln. Der Child-Schlüssel kann ebenfalls nicht dazu genutzt werden, irgendwelche Geschwisterschlüssel zu finden. Wenn Sie den n-ten Child-Schlüssel kennen, können Sie dessen Geschwister wie etwa Child n–1 oder Child n+1 oder eines der anderen Children dieser Sequenz nicht ermitteln. Nur der Parent-Schlüssel und der Chaincode können alle Children ableiten. Ohne Chaincode kann der Child-Schlüssel auch nicht verwendet werden, um irgendwelche »Enkel« abzuleiten. Sie benötigen sowohl den privaten Child-Schlüssel als auch den Child-Chaincode, um einen neuen Zweig zu eröffnen und »Enkel« abzuleiten.

Wozu kann also der private Child-Schlüssel allein verwendet werden? Er kann genutzt werden, um einen öffentlichen Schlüssel und eine Bitcoin-Adresse zu erzeugen. Dann kann er eingesetzt werden, um Transaktionen zu signieren und so über die in dieser Adresse gespeicherten Mittel zu verfügen.

Ein privater Child-Schlüssel, der dazugehörige öffentliche Schlüssel und die Bitcoin-Adresse sind von zufällig erzeugten Schlüsseln und Adressen nicht zu unterscheiden. Die Tatsache, dass sie Teile einer Folge sind, ist außerhalb der HD-Wallet-Funktion, die sie erzeugt hat, nicht zu ersehen. Einmal erzeugt, arbeitet er wie ein »normaler« Schlüssel.

Erweiterte Schlüssel

Wie Sie vorhin gesehen haben, kann die Funktion zur Schlüsselableitung verwendet werden, um Children auf jeder Ebene des Baums zu erzeugen. Dazu verwenden wir drei Inputs: einen Schlüssel, einen Chaincode und den Index des gewünschten Childs. Die beiden grundlegenden Zutaten sind der Schlüssel und der Chaincode. Kombiniert man die beiden, erhält man einen sogenannten *erweiterten Schlüssel*. Man kann sich diesen erweiterten Schlüssel auch als »erweiterbaren« Schlüssel vorstellen, da aus einem solchen Schlüssel Child-Schlüssel abgeleitet werden können.

Erweiterte Schlüssel sind einfach die Verkettung des 256-Bit-Schlüssels und des 256-Bit-Chaincodes in einer 512-Bit-Sequenz, und so werden sie auch gespeichert und dargestellt. Es gibt zwei Arten erweiterter Schlüssel. Ein erweiterter privater Schlüssel ist die Kombination aus einem privaten Schlüssel und einem Chaincode. Dieser kann verwendet werden, um private Child-Schlüssel abzuleiten (und daraus öffentliche Child-Schlüssel). Ein erweiterter öffentlicher Schlüssel besteht aus einem öffentlichen Schlüssel und einem Chaincode. Aus diesem lassen sich öffentliche Child-Schlüssel (nur öffentliche) erzeugen, wie es in »Einen öffentlichen Schlüssel generieren« auf Seite 65 beschrieben wird.

Stellen Sie sich einen erweiterten Schlüssel als Wurzel eines Zweigs in der Baumstruktur der HD-Wallet vor. Von der Wurzel des Stamms können Sie den ganzen Rest ableiten. Der erweiterte private Schlüssel kann einen vollständigen Stamm erzeugen, während der erweiterte öffentliche Schlüssel lediglich einen Zweig mit öffentlichen Schlüsseln erzeugen kann.

Ein erweiterter Schlüssel besteht aus einem privaten oder öffentlichen Schlüssel sowie einem Chaincode. Ein erweiterter Schlüssel kann Child-Schlüssel erzeugen und so einen eigenen Zweig in der Baumstruktur anlegen. Wer den erweiterten Schlüssel kennt, hat Zugriff auf den gesamten Zweig.

Erweiterte Schlüssel werden in Base58Check codiert, um sie einfach zwischen unterschiedlichen BIP-32-kompatiblen Wallets im- und exportieren zu können. Die Base58Check-Codierung für erweiterte Schlüssel verwendet eine spezielle Versions-

nummer, die zu den Präfixen `xprv` und `xpub` in den Base58-codierten Zeichen führt, wodurch sie leicht zu erkennen sind. Da die erweiterten Schlüssel aus 512 oder 513 Bit bestehen, sind sie auch wesentlich länger als die anderen Base58Check-codierten Strings, die wir bisher gesehen haben.

Hier ein Beispiel für einen erweiterten *privaten* Schlüssel, codiert in Base58Check:

```
xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VMNjGr67Lctvy5P8oyaYAL9CAWrUE9
i6GoNMKUga5biW6Hx4tws2six3b9c
```

Hier der dazugehörige erweiterte *öffentliche* Schlüssel, ebenfalls in Base58Check codiert:

```
xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgypeQbBs2UAR6KECeeMVKZBPLrtJ
unSDMstweyLXhRgPxdp14sk9tJPW9
```

Ableitung öffentlicher Child-Schlüssel

Wie bereits erwähnt, ist eine sehr nützliche Charakteristik von HD-Wallets die Fähigkeit, öffentliche Child-Schlüssel aus öffentlichen Parent-Schlüsseln abzuleiten, ohne den privaten Schlüssel zu besitzen. Das bietet uns zwei Möglichkeiten, einen öffentlichen Child-Schlüssel abzuleiten: entweder aus dem privaten Child-Schlüssel oder direkt aus dem öffentlichen Parent-Schlüssel.

Ein erweiterter öffentlicher Schlüssel kann also verwendet werden, um alle öffentlichen Schlüssel (und nur die öffentlichen Schlüssel) in diesem Zweig der Struktur der HD-Wallet abzuleiten.

Diese Verknüpfung kann genutzt werden, um Deployments mit sehr sicheren, rein öffentlichen Schlüsseln aufzubauen, bei denen ein Server oder eine Anwendung nur einen erweiterten öffentlichen Schlüssel und keinerlei private Schlüssel enthält. Diese Form des Deployments kann eine unendliche Zahl von öffentlichen Schlüsseln und Bitcoin-Adressen erzeugen, ohne auf das an diese Adressen gesendete Geld zugreifen zu können. Gleichzeitig kann auf einem anderen (sichereren) Server der erweiterte private Schlüssel alle passenden privaten Schlüssel ableiten, mit denen eine Transaktion signiert und Geld eingelöst werden kann.

Eine typische Anwendung dieser Lösung ist die Installation eines erweiterten öffentlichen Schlüssels auf einem Webserver mit einer E-Commerce-Anwendung. Der Webserver kann neue Bitcoin-Adressen für jede Transaktion mithilfe des öffentlichen Schlüssels erzeugen (etwa für den Warenkorb eines Kunden). Auf dem Webserver selbst liegen keine privaten Schlüssel, die man stehlen könnte. Ohne HD-Wallets müsste man schon im Vorfeld Tausende von Bitcoin-Adressen auf einem separaten sicheren Server erzeugen und auf den E-Commerce-Server hochladen. Dieser Ansatz ist umständlich und verlangt eine regelmäßige Pflege, damit dem E-Commerce-Server die Schlüssel nicht ausgehen.

Eine weitere gängige Anwendung dieser Lösung bilden Cold Storage oder Hardware-Wallets. In diesem Szenario kann der erweiterte private Schlüssel auf einer Paper- oder Hardware-Wallet (wie etwa einem Trezor) gespeichert werden, wäh-

rend die erweiterten privaten Schlüssel online vorgehalten werden. Der Benutzer kann beliebige »Empfängeradressen« erzeugen, während die privaten Schlüssel offline sicher aufbewahrt werden. Um die Mittel auszugeben, kann der Nutzer den erweiterten privaten Schlüssel auf einem offline liegenden Bitcoin-Client oder einer Hardware-Wallet verwenden. Abbildung 5-11 verdeutlicht den Mechanismus, bei dem aus einem Parent-Schlüssel öffentliche Child-Schlüssel abgeleitet werden.

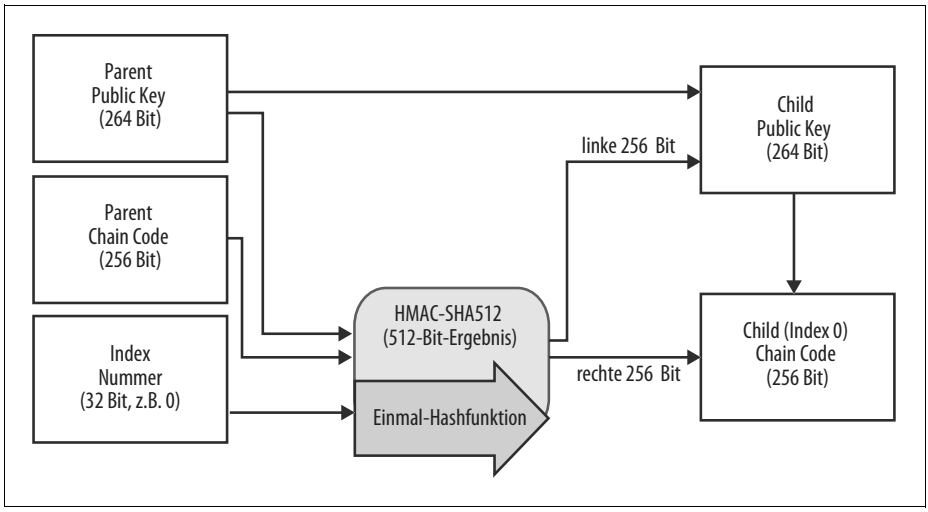

Abbildung 5-11: Öffentlichen Parent-Schlüssel erweitern, um öffentliche Child-Schlüssel zu erzeugen

Einen erweiterten öffentlichen Schlüssel in einem Webshop nutzen

Kehren wir zu Gabriels Webshop zurück und sehen wir uns an, wie HD-Wallets eingesetzt werden.

Für Gabriel war der Webshop zuerst nur ein Hobby und basierte auf einer einfachen gehosteten WordPress-Seite. Der Shop war mit nur wenigen Seiten und einer einzelnen Bitcoin-Adresse sehr einfach aufgebaut.

Gabriel nutzte die erste von seinem Trezor erzeugte Bitcoin-Adresse für seinen Shop. Auf diese Weise würden alle eingehenden Zahlungen an eine Adresse gehen, die durch die Hardware-Wallet kontrolliert wird.

Kunden können Bestellungen über ein Formular aufgeben und Zahlungen an die von Gabriel veröffentlichte Bitcoin-Adresse senden. Er erhält daraufhin eine E-Mail mit den Daten, die er zur Abwicklung des Auftrags benötigt. Bei nur wenigen Aufträgen pro Woche hat das System ausreichend gut funktioniert.

Doch der Shop wurde schnell erfolgreich, und es gingen viele Bestellungen von der lokalen Community ein. Schon bald wurde Gabriel mit Bestellungen über-

schwemmt, und es wurde zunehmend schwerer, die Bestellungen mit den Zahlungen abzugleichen, insbesondere wenn zeitnah mehrere Bestellungen mit den gleichen Beträgen eingingen.

Die HD-Wallet bot mit der Fähigkeit, öffentliche Child-Schlüssel abzuleiten, ohne die privaten Schlüssel kennen zu müssen, hier eine wesentlich bessere Lösung an. Gabriel kann einen erweiterten öffentlichen Schlüssel (xpub) auf seine Website laden und für jede Bestellung eine eindeutige Adresse ableiten. Er kann die Beträge über seinen Trezor einlösen, der auf die Website geladene xpub kann nur Adressen generieren und Zahlungen empfangen. Diese Möglichkeit von HD-Wallets ist ein tolles Sicherheits-Feature. Gabriels Website enthält keine privaten Schlüssel und stellt daher keine besonders hohen Sicherheitsanforderungen.

Zum Export des xpub verwendet Gabriel eine webbasierte Software im Zusammenspiel mit seiner Trezor-Wallet. Der Trezor muss angeschlossen sein, um die öffentlichen Schlüssel exportieren zu können. Beachten Sie, dass Hardware-Wallets niemals private Schlüssel exportieren – diese verbleiben immer auf dem Gerät. Abbildung 5-12 zeigt die Webschnittstelle, die Gabriel zum Export des xpub verwendet.

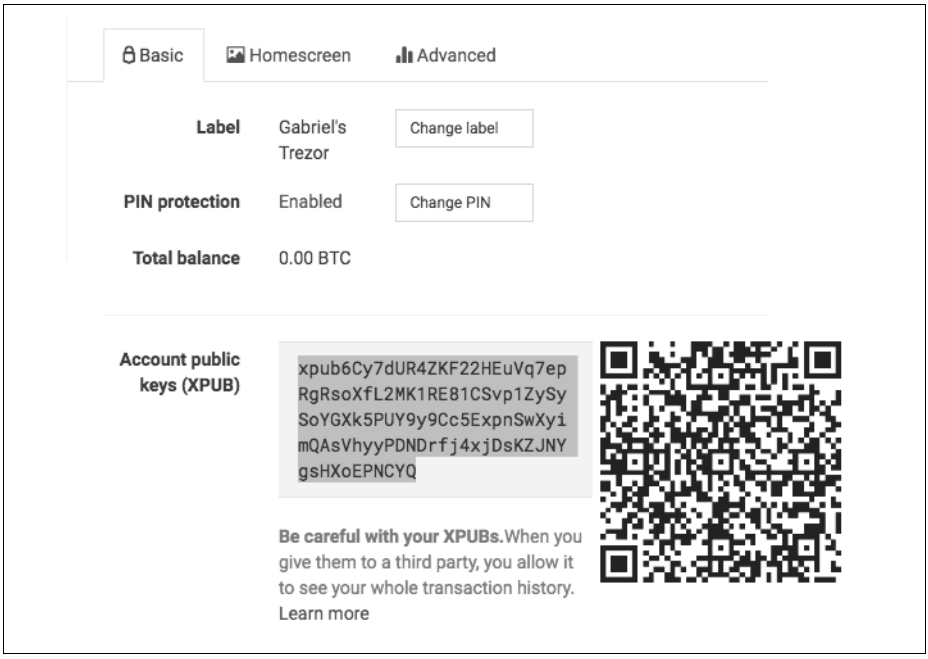

Abbildung 5-12: Export eines xpub aus einer Trezor-Wallet

Gabriel kopiert den xpub in die Bitcoin-Software seines Webshops. Er verwendet *Mycelium Gear*, ein Open-Source-Webshop-Plug-in für eine Vielzahl von Webhosting- und Content-Plattformen. Mycelium Gear verwendet den xpub, um eine eindeutige Adresse für jeden Kauf zu erzeugen.

Gehärtete Ableitung von Child-Schlüsseln

Die Fähigkeit, einen Zweig öffentlicher Schlüssel aus einem xpub abzuleiten, ist sehr nützlich, aber es gibt ein potenzielles Risiko. Durch den Zugriff auf einen xpub gelangen Sie nicht an die privaten Child-Schlüssel. Doch weil der xpub den Chaincode enthält, können Sie, falls der private Schlüssel irgendwie durchsickert, alle anderen privaten Child-Schlüssel ableiten. Ein einziger durchgesickerter privater Child-Schlüssel legt zusammen mit einem Parent-Chaincode alle privaten Schlüssel aller Children offen. Schlimmer noch: Mit dem privaten Child-Schlüssel und einem Parent-Chaincode können Sie den privaten Parent-Schlüssel ableiten.

Um diesem Risiko entgegenzuwirken, verwenden HD-Wallets eine alternative Ableitungsfunktion, die als *gehärtete Ableitung* bezeichnet wird. Diese »bricht« die Beziehung zwischen öffentlichem Parent-Schlüssel und Child-Chaincode auf. Die gehärtete Ableitungsfunktion verwendet den privaten Parent-Schlüssel anstelle des öffentlichen Parent-Schlüssels, um den Child-Chaincode abzuleiten. Auf diese Weise wird eine »Firewall« in der Parent/Child-Sequenz erzeugt, da der Chaincode nicht mehr genutzt werden kann, um einen privaten Parent- oder Child-Schlüssel abzuleiten. Die gehärtete Ableitungsfunktion ist nahezu identisch mit der normalen Ableitung des privaten Child-Schlüssels. Anstelle des öffentlichen Parent-Schlüssels wird aber der private Parent-Schlüssel als Input für die Hashfunktion verwendet (siehe Abbildung 5-13).

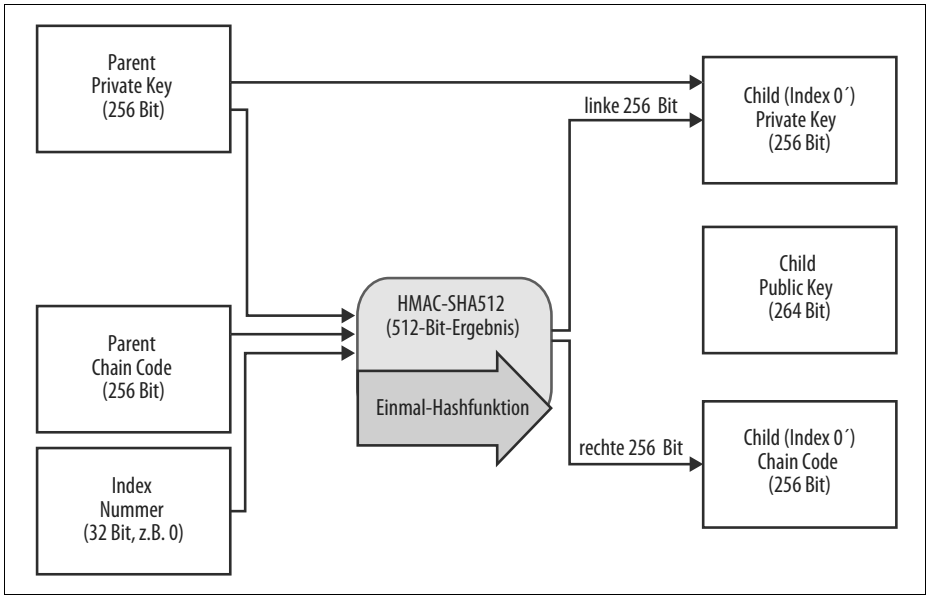

Abbildung 5-13: Gehärtete Ableitung eines Child-Schlüssels, vermeidet den öffentlichen Parent-Schlüssel

Wird die gehärtete Ableitungsfunktion verwendet, unterscheiden sich der resultierende private Child-Schlüssel und der Chaincode völlig von dem Ergebnis der nor-

malen Ableitungsfunktion. Der resultierende »Zweig« von Schlüsseln kann zur Erzeugung erweiterter öffentlicher Schlüssel genutzt werden, die nicht angreifbar sind, weil der darin enthaltene Chaincode nicht ausgenutzt werden kann, um private Schlüssel offenzulegen. Die gehärtete Ableitung wird daher genutzt, um an der Stelle eine »Lücke« zu schaffen, an der erweiterte Schlüssel verwendet werden.

Wenn Sie die Bequemlichkeit eines xpub zur Ableitung von »Zweigen« mit öffentlichen Schlüsseln verwenden wollen, ohne sich selbst der Gefahr eines durchgesickerten Chaincodes aussetzen zu wollen, sollten Sie diesen nicht von einem »normalen«, sondern von einem gehärteten Parent ableiten. Es hat sich bewährt, den ersten Child-Schlüssel des Master-Schlüssels über eine gehärtete Ableitung zu erzeugen, um eine Gefährdung des Master-Schlüssels zu verhindern.

Indizes für normale und gehärtete Ableitungen

Der Index, der für die Ableitungsfunktion verwendet wird, ist ein 32-Bit-Integerwert. Um einfach zwischen der normalen und der gehärteten Ableitungsfunktion unterscheiden zu können, wird der Index in zwei Wertebereiche geteilt. Indizes zwischen 0 und 2^{31}–1 (0x0 bis 0x7FFFFFFF) werden ausschließlich für die normale Ableitung verwendet, die Indizes zwischen 2^{31} und 2^{32}–1 (0x80000000 bis 0xFFFFFFFF) nur für die gehärtete Ableitung. Bei Werten kleiner als 2^{31} handelt es sich um einen normalen Child-Schlüssel, bei Indizes größer oder gleich 2^{31} um gehärtete Child-Schlüssel.

Um den Index besser lesen und darstellen zu können, beginnt die Darstellung der Indizes gehärterer Children bei null, wird aber zusätzlich mit einem Hochkomma versehen. Der erste normale Child-Schlüssel wird also als 0 dargestellt, während das erste gehärtete Child (Index 0x80000000) als 0' ausgegeben wird. Dementsprechend wird der zweite gehärtete Schlüssel mit dem Index 0x80000001 als 1' dargestellt werden und so weiter. Ein HD-Wallet-Index von i' bedeutet also 2^{31}+i.

HD-Wallet-Schlüsselbezeichner (Pfad)

Die Schlüssel in einer Wallet werden über einen »Pfad« identifiziert, wobei jede Stufe durch ein Slash (/) getrennt wird (siehe Tabelle 5-6). Vom Master-Schlüssel abgeleitete private Schlüssel beginnen mit einem `m`. Aus dem öffentlichen Master-Schlüssel abgeleitete öffentliche Schlüssel beginnen mit einem `M`. Der erste private Child-Schlüssel des privaten Master-Schlüssels ist also `m/0`. Der erste öffentliche Child-Schlüssel ist `M/0`. Der »Enkel« des ersten Childs ist dann `m/0/1` und so weiter.

Die »Herkunft« eines Schlüssels wird von rechts nach links gelesen, bis man den Master-Schlüssel erreicht, aus dem er abgeleitet ist. Der Bezeichner `m/x/y/z` beschreibt beispielsweise das z-te Child des Schlüssels `m/x/y`, der das y-te Child des Schlüssels `m/x` ist, der wiederum das x-te Child von `m` darstellt.

Tabelle 5-6: HD-Wallet-Pfadbeispiel

HD-Pfad	Beschriebener Schlüssel
m/0	Der erste (0) private Child-Schlüssel des privaten Master-Schlüssels (m).
m/0/0	Erster privater Enkel-Schlüssel des ersten Schlüssels (m/0).
m/0'/0	Erster normaler Enkel des ersten gehärteten Child-Schlüssels (m/0').
m/1/0	Erster privater Enkel-Schlüssel des zweiten Child-Schlüssels (m/1).
M/23/17/0/0	Erster öffentlicher Ururenkel des ersten Urenkels des 18. Urenkels des 24. Child-Schlüssels.

Durch die Baumstruktur der HD-Wallet navigieren

Die Baumstruktur der HD-Wallet bietet eine enorme Flexibilität. Jeder erweiterte Parent-Schlüssel kann vier Milliarden Child-Schlüssel besitzen, jeweils zwei Milliarden normale und zwei Milliarden gehärtete Child-Schlüssel. Jeder dieser Child-Schlüssel kann weitere vier Milliarden Child-Schlüssel erzeugen und so weiter. Der Baum kann beliebig tief sein mit einer unendlichen Anzahl von Nachkommen. Bei all der Flexibilität wird es aber recht schwierig, durch den Baum zu navigieren. Besonders schwer ist der Transfer von HD-Wallets zwischen Implementierungen, da die interne Organisation in Verzweigungen und Unterverzweigungen schier endlos ist.

Zwei BIPs bieten eine Lösung für diese Komplexität, indem sie Standards für die Struktur der HD-Wallet-Bäume empfehlen. BIP-43 empfiehlt die Verwendung des ersten gehärteten Child-Index als speziellen Bezeichner, der den »Zweck« der Baumstruktur verdeutlicht. Basierend auf BIP-43 soll eine HD-Wallet nur einen Zweig der ersten Ebene verwenden, bei dem der Indexwert die Struktur und den Namensraum für den Rest des Baums festlegt, indem er dessen »Zweck« definiert. Zum Beispiel dient eine HD-Wallet, die ausschließlich den Zweig m/i'/ verwendet, einem bestimmten Zweck, der durch `i` identifiziert wird.

Auf dieser Spezifikation aufbauend, schlägt BIP-44 eine Multi-Account-Struktur mit der »Zweck«-Nummer `44'` unter BIP-43 vor. Alle der BIP-44-Struktur folgenden HD-Wallets können daran erkannt werden, dass sie nur einen Zweig des Baums, nämlich m/44'/, nutzen.

BIP-44 legt die Struktur mit fünf vordefinierten Baumebenen fest:

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

Der »Zweck« (`purpose`) der ersten Ebene ist immer `44'`. Der »Cointyp« (`coin_type`) der zweiten Ebene legt die Kryptowährung fest. Das erlaubt Mehrwährungs-Wallets, bei denen jede Währung einen eigenen Teilbaum unterhalb der zweiten Ebene besitzt. Momentan sind drei Währungen definiert: Bitcoin ist m/44'/0', Bitcoin Testnet ist m/44'/1' und Litecoin ist m/44'/2'.

Die dritte Ebene des Baums ist das Konto (`account`). Darüber kann der Nutzer seine Wallet in separate, logische Unterkonten für buchhalterische oder organisatori-

sche Zwecke aufteilen. Eine HD-Wallet könnte beispielsweise zwei Bitcoin-Konten enthalten: m/44'/0'/0' und m/44'/0'/1'. Jedes Konto bildet den Stamm eines eigenen Teilbaums.

Auf der vierten Ebene, change, besitzt eine HD-Wallet zwei Teilbäume. Einer dient der Erzeugung von Empfängeradressen, während der andere für Change-Adressen, also »Wechselgeldadressen« gedacht ist. Während die vorangegangenen Ebenen die gehärtete Ableitung nutzen, verwendet dieses Level die normale Ableitung. Das erlaubt dieser Ebene des Baums, erweiterte öffentliche Schlüssel zu exportieren, die in ungeschützten Umgebungen eingesetzt werden können. Verwendbare Adressen werden von der HD-Wallet als Child-Elemente des vierten Levels erzeugt, was die fünfte Ebene des Baums zum »Adressindex« macht. Zum Beispiel wäre M/44'/0'/0'/0/2 die dritte Empfängeradresse für Bitcoin-Zahlungen des primären Kontos. Tabelle 5-7 zeigt einige weitere Beispiele.

Tabelle 5-7: Beispiele der BIP-44-HD-Wallet-Struktur

HD-Pfad	Beschriebener Schlüssel
M/44'/0'/0'/0/2	Dritter empfangender öffentlicher Schlüssel des primären Bitcoin-Kontos.
M/44'/0'/3'/1/14	15. öffentlicher Schlüssel für Change-Adressen des vierten Bitcoin-Kontos.
m/44'/2'/0'/0/1	Zweiter privater Schlüssel im Litecoin-Hauptkonto zum Signieren von Transaktionen.