

ブロックチェーン公開講座 第2回

ビットコインその1

芝野恭平

東京大学大学院工学系研究科技術経営戦略学専攻

ブロックチェーンイノベーション寄付講座

特任研究員

shibano@tmi.t.u-tokyo.ac.jp

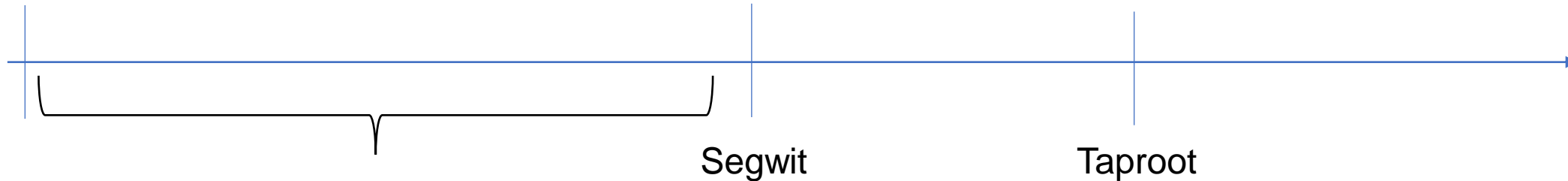


ビットコインプロトコルの主なアップグレード

2009年1月

2017年8月

2021年11月



Taproot



- ・ マークル化抽象構文木 (MAST)
- ・ シュノア署名

講義ではこの初期のビットコインプロトコルを取り扱います.

→ より基本的なブロックチェーンの原理を学ぶことが目的.

教材

- Mastering Bitcoin (second edition)
 - https://github.com/bitcoinbook/bitcoinbook/tree/second_edition_print3
- ビットコインとブロックチェーン:暗号通貨を支える技術 (Mastering Bitcoin日本語訳)
 - <https://www.amazon.co.jp/dp/B072JLL66R>

 README  License

Mastering Bitcoin

Mastering Bitcoin is a technical book that explains what Bitcoin is and how it works.

This repository contains the complete text of three editions of the book Mastering Bitcoin as published in paperback and ebook formats by O'Reilly Media. Different editions of this book are covered by different op licenses (see [LICENSE](#)). The three editions are:

- The [first edition, second print](#), published in December 2014
- The [second edition, third print](#), published in March 2018
- The [third edition, first print](#), published in December 2023

Reading This Book

To read this book *for free*, see [BOOK.md](#). Click on each of the chapters to read in your browser.

Please note that some of the links in each chapter do not work when reading the book on Github. This is because those links are intended for the print and ebook editions of the book and only work when all the chapters are rendered together. Unfortunately, Github does not have the ability to render the complete book at once.



ビットコインの仕組み



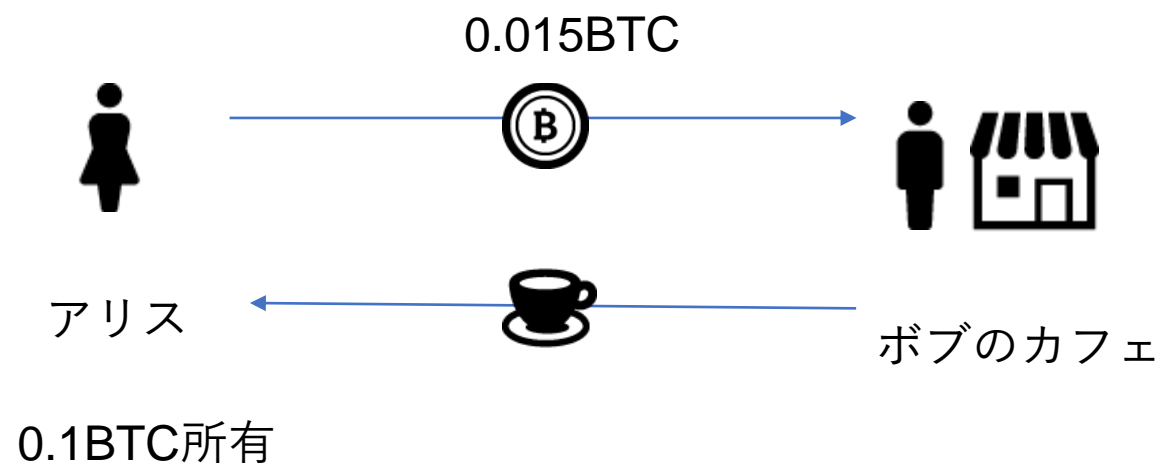
概要

- アリスがボブへ送金する例を考える.
- このときビットコインはシステムの的にどのように動作して送金を実現しているのか概観する.
 - ウォレットと鍵
 - トランザクション作成
 - ビットコインネットワークにブロードキャスト
 - マイニングによるブロック生成, 伝搬



コーヒー代金の支払いをビットコインで行う

- アリスはボブのカフェでコーヒーを注文した.
- アリスはもともと0.1BTC持っていた.
- コーヒーの代金として0.015BTCをボブに支払った.



ハッシュ関数とは

- 一方向関数
- 順方向は一瞬で値が出力される
- 逆方向は求めることが困難.
- ハッシュ関数はいくつも形式がある
 - MD5, SHA1, SHA256, . . .
- SHA256がよく使用される.
 - 暗号学的ハッシュ関数
 - 現像計算困難性 . . . ハッシュ値から入力値を求めるのが困難
 - 第二現像計算困難性 . . . ある入力値が与えられたとき, 同じハッシュ値を持つ異なる入力値を求めるのが困難
 - 強衝突耐性 . . . 同じハッシュ値となる2つの異なる入力値を求めるのが困難

```
shibano@user-pc:~/git/doc_blockchain$ time echo -n 'test' | shasum -a 256
9f86d081884c7d859a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08 -
real    0m0.048s
user    0m0.031s
sys     0m0.016s
```

あいいうえお



fdb481ea956fdb654afcc327cff9b626966b2abdabc3f3e6dbcb1667a888ed9a

あいいうえこ



59d9b0d8d3e1262bd220ebc58989569c064b48df22c28dfcc3fd2186b484af34

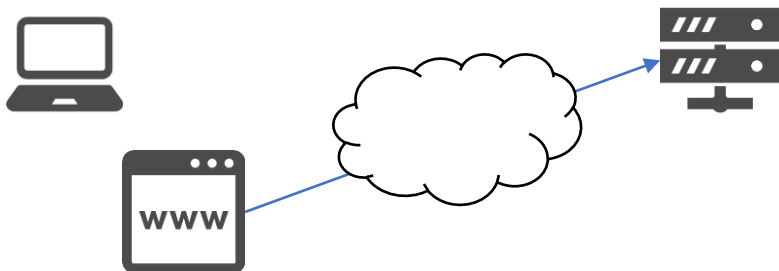
あいうおえ



a762518a8b8eff0175e1e0e9493594db95b23aba9f04dae7f52621c1dafddea9

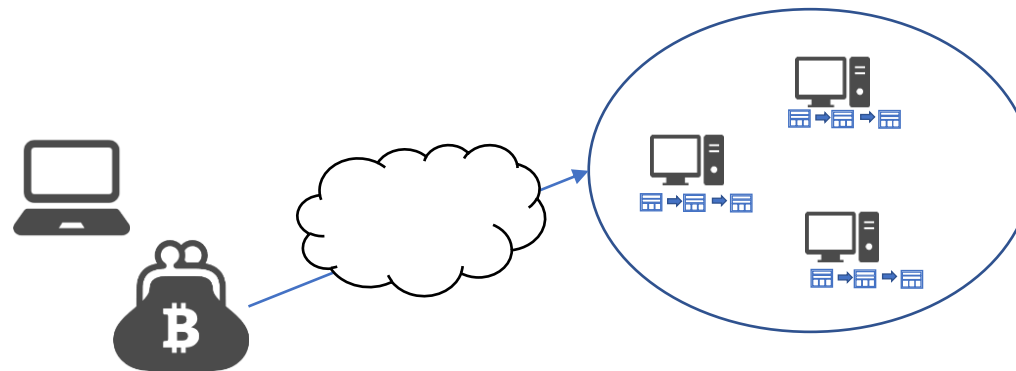
一般的なWebシステムとビットコインシステム

Webのシステム



ブラウザを使って、HTTPリクエストを送信するとWebサーバーはレスポンスを返す。

ビットコインのシステム



ウォレットソフトウェアを使ってトランザクションを生成して、ビットコインネットワークにブロードキャストする。しばらく待つと、ブロックチェーンに取り込まれる。

ビットコイン概観

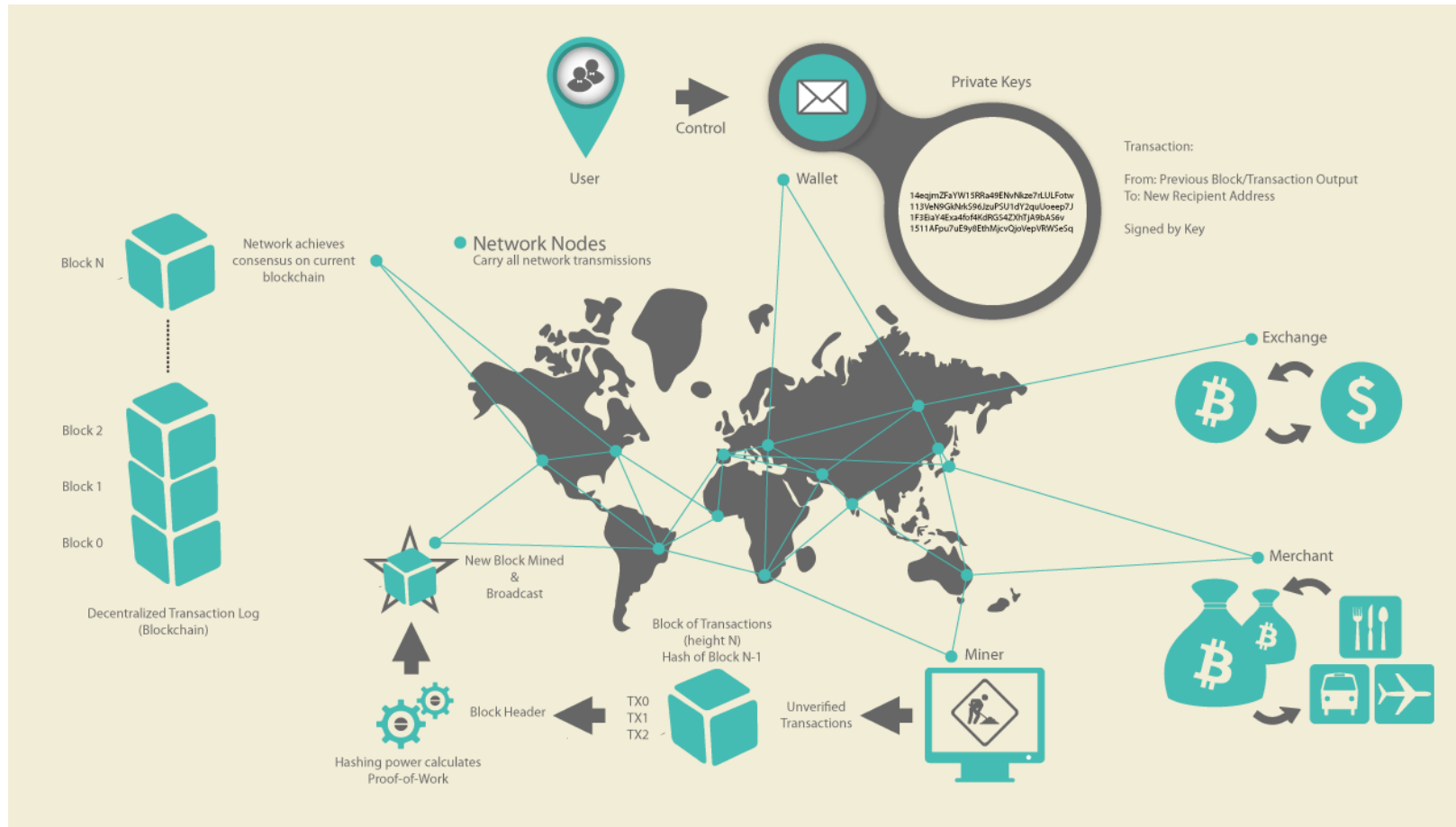
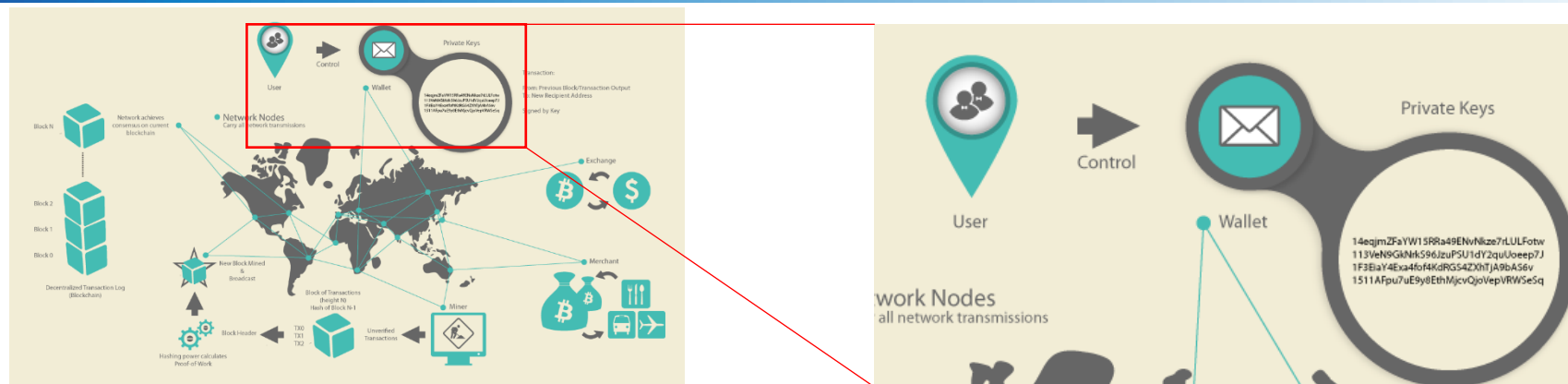


Figure 1. Bitcoin overview

https://github.com/bitcoinbook/bitcoinbook/blob/second_edition_print3/ch02.asciidoc

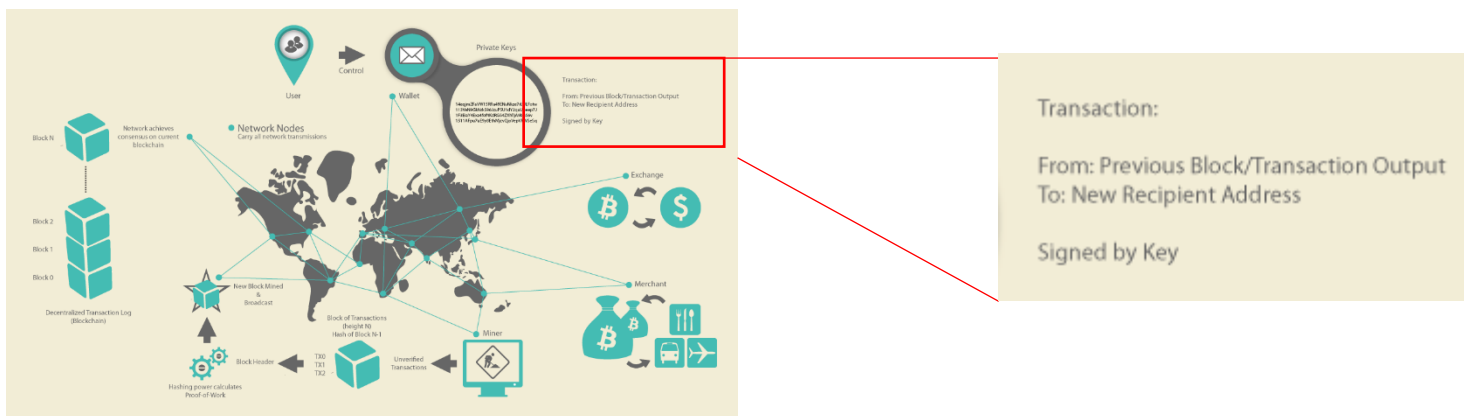
ユーザーとウォレット，秘密鍵



- ウォレットの利用
 - Bitcoinの保有者は，自身のアドレスと，それに紐づく秘密鍵，それらを管理するウォレットを使用します。
- アドレス
 - ・・・ビットコインを利用するための口座番号のようなもの．ユーザーはEメールをだれかのメールアドレスに送信するのと同様に，ほかの人が持っているアドレスに送金します。
- 秘密鍵
 - ・・・自身のアドレスから送金を行うために必要。
- ウォレット
 - ・・・自身のアドレスの残高を確認したり，秘密鍵を管理したり，送金を行うことができるソフトウェア。

「アリスは，自身の秘密鍵が管理されているウォレットを使ってボブのアドレス宛に0.015BTC送金するトランザクションを生成する」

トランザクション



ウォレットにて作られたトランザクションは、ビットコインネットワークに送信される。

トランザクションには

- どのトランザクションから
- どのアドレスに
- いくら

送金するかという情報を含み、送金元の電子署名も含む。

アリスからボブへの送金トランザクション

ビットコインはUTXO (Unspent Transaction Output)モデルを採用しており、未使用のトランザクションから新しいトランザクションを作る。

- アドレスごとに残高が記録されているわけではない
- アドレスからではなく未使用のトランザクションから作る

アリスへ送金されたトランザクション

作成するトランザクション

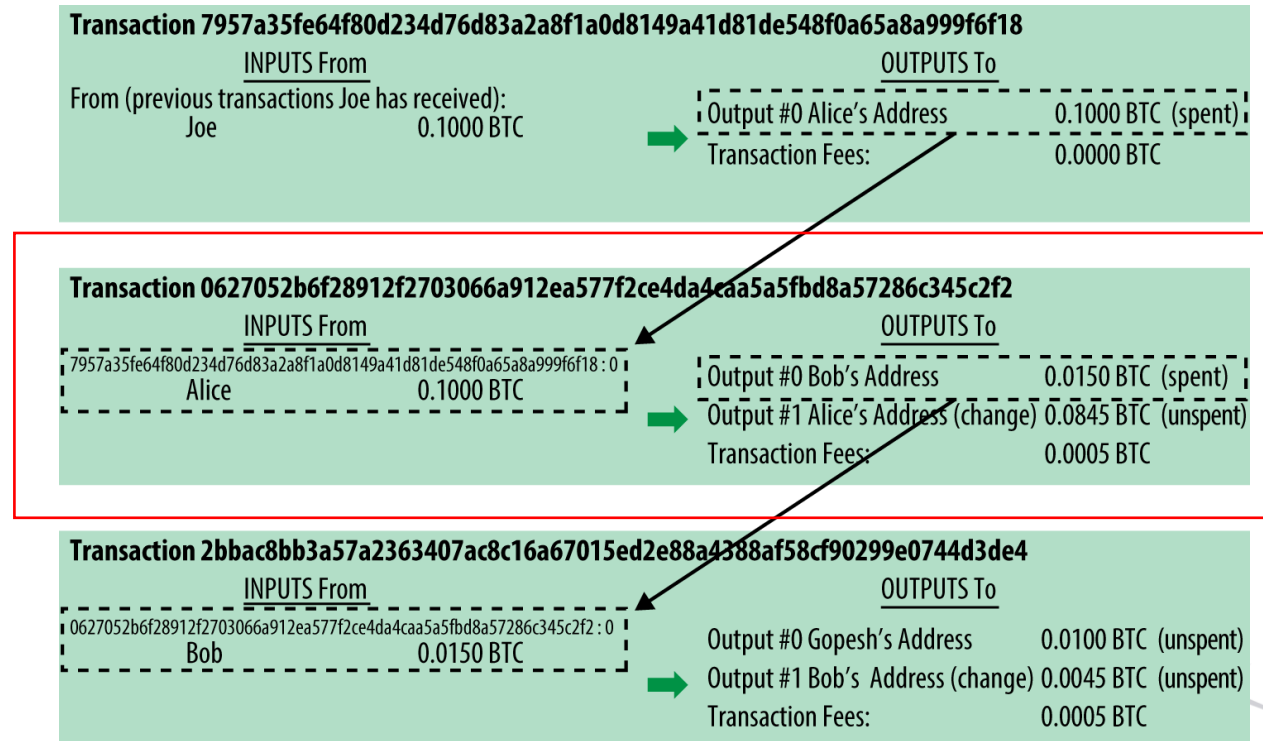


Figure 4. A chain of transactions, where the output of one transaction is the input of the next transaction

トランザクションの形式

- 一般的なトランザクション
 - 1つのアドレスから1つのアドレスに送金
 - 元のアドレスへのおつり送金が含まれる
- 集約型トランザクション
 - 複数のインプットを集めて1つのアウトプットにまとめる
 - おつりとして受け取った小額トランザクションをまとめるときに使われる
- 分配型トランザクション
 - 1つのインプットから複数のアウトプットに分ける
 - 給与の支払いなどに使われる

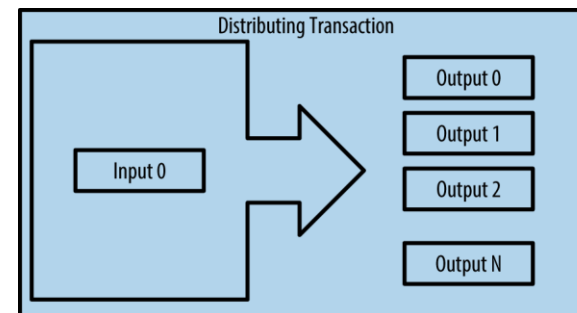
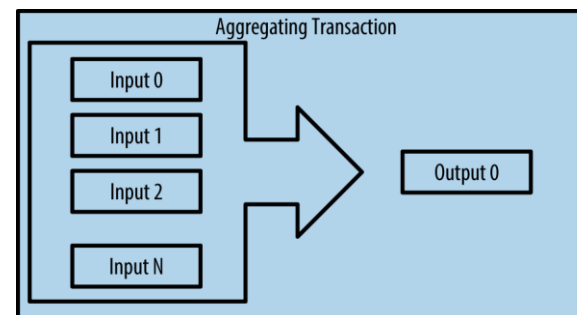
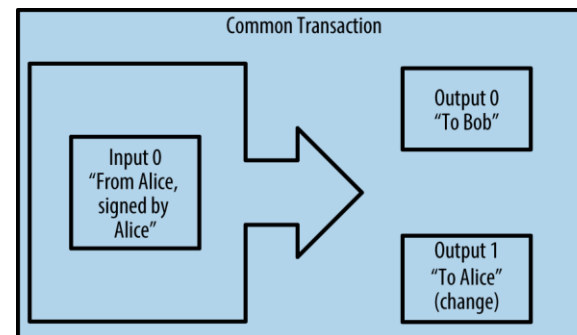


Figure 5. Most common transaction
Figure 6. Transaction aggregating funds
Figure 7. Transaction distributing funds

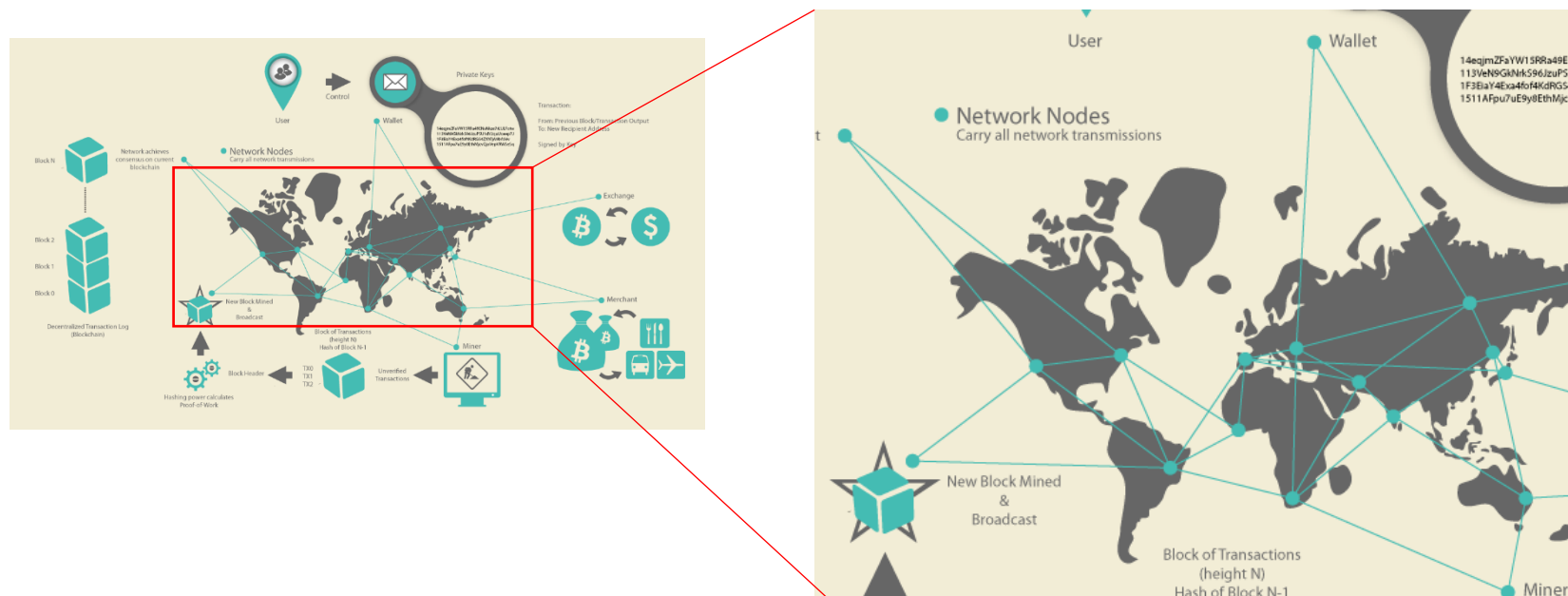
正しいインプットをどのように得るか

- ほとんどのウォレットはブロックチェーンデータとは別に、UTXO (Unspent Transaction Output, 未使用トランザクションアウトプット) を保持するデータベースを内部に持っている
 - ブロックチェーンのデータをもとにDBを生成
 - そうしないと最初から最後までブロック(80万超え, 2024-04時点)に含まれるTxすべてをチェックしないといけなくなる.
- フルインデックスウォレットの場合：
 - フルノード (全ブロックチェーンデータを持っている)
 - ブロックチェーンデータより全アドレスに対するUTXOデータベースを生成可能
 - フルノードなので、データ量が非常に大きくなる.
 - データ量は600GB強.
- 軽量ウォレットの場合：
 - 持ち主のUTXOデータベースのみ保持
 - ビットコインネットワークに問い合わせで生成する
 - フルノードに問い合わせる

アウトプットの作成

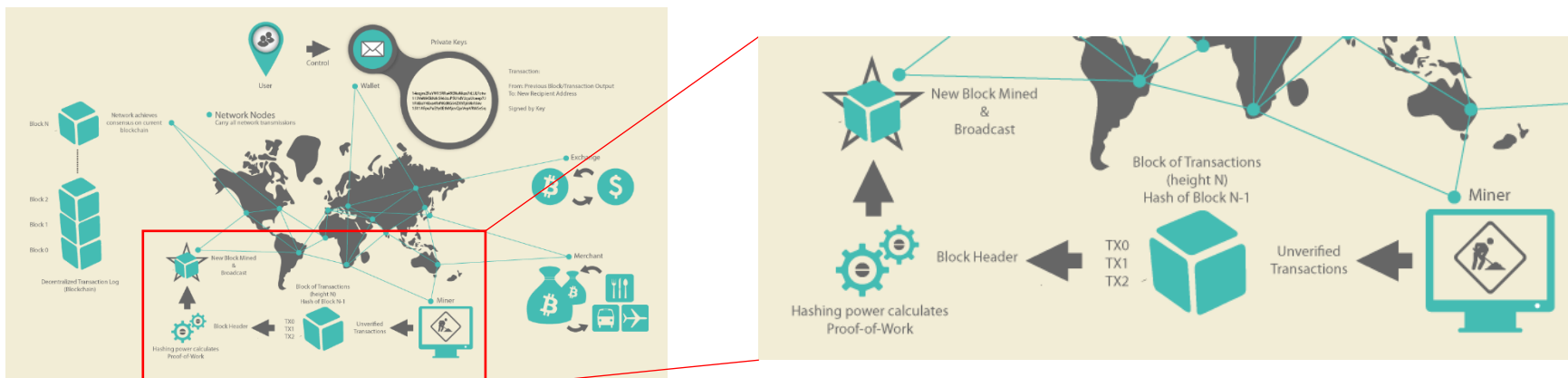
- アウトプットは2つ
 - ボブのアドレスへ0.015BTC
 - アリスのアドレスへ0.0845BTC
 - (手数料0.005BTC)
- トランザクションアウトプットはスクリプトの形で作成されている。
 - スクリプトはその資金を使用するための解除条件
 - 誰でも任意のアドレスに関するスクリプトを生成できる.
 - ボブへの送金アウトプット
 - ボブのアドレスに対応する秘密鍵からの署名を作れる人であれば誰でもこのアウトプットから送金可能.
 - 実際は, ボブだけが秘密鍵を持っているので, アウトプットに対する署名を作成できる.
 - アリスへの送金アウトプット
 - 同様にアリスのみが自分の秘密鍵を用いてそこから送金するトランザクションを作成できる

トランザクションをビットコインネットワークへ送信, 伝搬



- 世界中にあるビットコインノードの一部に送信する
- そのノードはインターネットを通じて別のノードにそれを転送する
- 数秒以内に大半のノードに到達する
- このときはまだブロックには入っていない。
 - トランザクションプール内に存在

ブロックの生成・マイニング



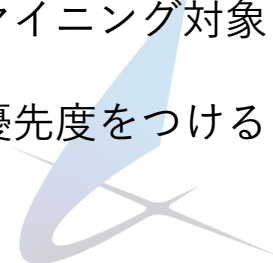
- トランザクションがブロックに記録されるにはマイニングが必要
 - ブロックにはトランザクションが複数記録される。
 - 世界にあるどこかのノードでマイニングに成功し、ブロック内に該当のトランザクションが取り込まれる必要がある。
- PoW (Proof of Work)
 - トランザクションを含むブロックに十分な計算量がつぎ込まれた場合のみそのトランザクションが承認される。
 - 「計算」とは（正確ではないが）乱数を生成して、ある条件の値が出るまで繰り返し乱数を生成し続けるというもの。
 - すごく目の大きいサイコロを振り続けるイメージ
 - 具体的にはある条件に合致するハッシュ値が出る元のインプットを探す
 - マイニングに成功すると報酬がもらえる
 - 2024/4現在マイニング報酬は6.25BTC（そろそろ半減期が来て、3.125 BTCに変更される）
 - 4年に1回報酬は半額になる
 - 手持ちのパソコンで誰でもマイニングはできる
 - マイニングには膨大な計算が必要だが、検証は一瞬で可能（ハッシュを利用）

マイニングの目的

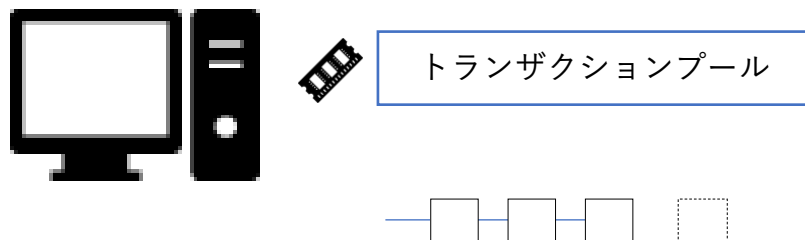
- マイニングは、それぞれのブロックの中に新しいビットコインを作り出します。これは、あたかも中央銀行が新しいお金を印刷するようなものです。ブロックごとに作り出されるビットコインの量は決められており、時間とともに減少していきます。
- マイニングは、信用を作り出します。マイニングは、「トランザクションを含むブロックに十分な計算量がつぎ込まれた場合にのみ、このトランザクションが承認される」ことを保証することによって、信用を作り出します。より多くのブロックがあるということは、より多くの計算量を要したことを意味し、したがって、より多くの信用を得ていることを意味するのです。

ビットコインとブロックチェーン:暗号通貨を支える技術P.27より引用

- いくつも正しいTxがある中で、一定の計算量をつぎ込んでまで次のブロックに含めたい、というマイナーの意思を表現。
- Txを作る側から考えると、手数料を多めに設定することで多くのマイナーにマイニング対象のブロック中に取り込んでもらえることが期待される。
- つまり、「正しいTx」すべてをブロックに取り込むのが無理だから取り込む優先度をつけるのがPoWとも言える。



トランザクションとマイニング

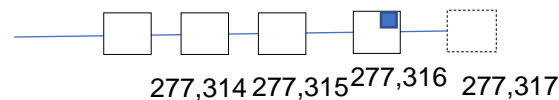
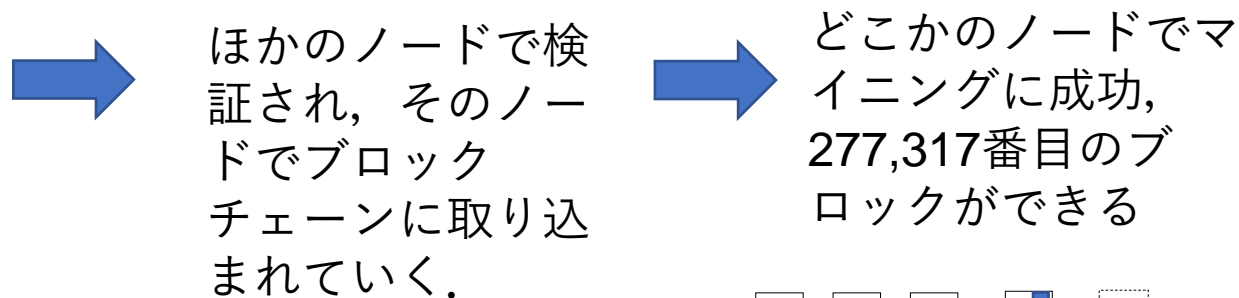
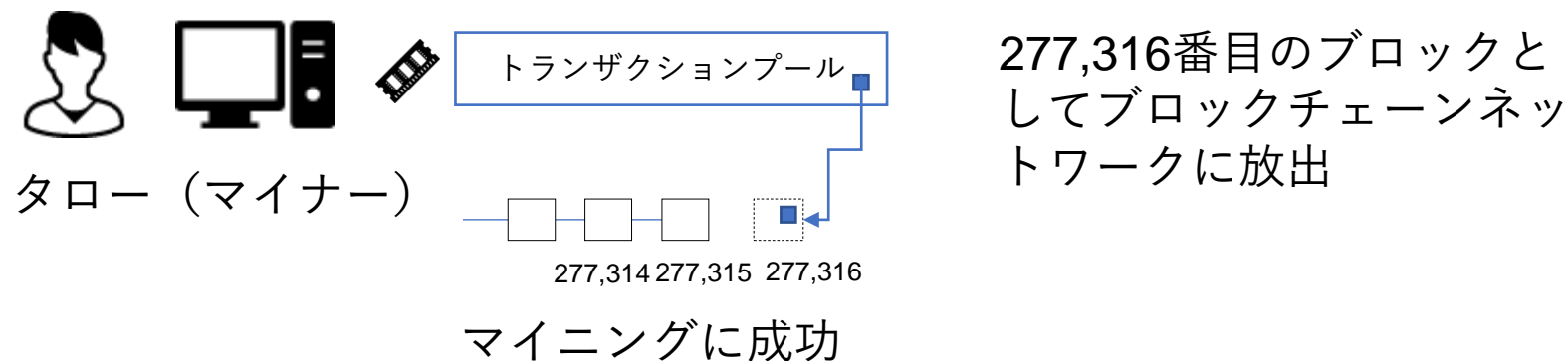


ウォレットから送信されたトランザクションはトランザクションプールに入る。(ノードのメモリ内のみで管理されており、まだブロックチェーンには記録されていない)

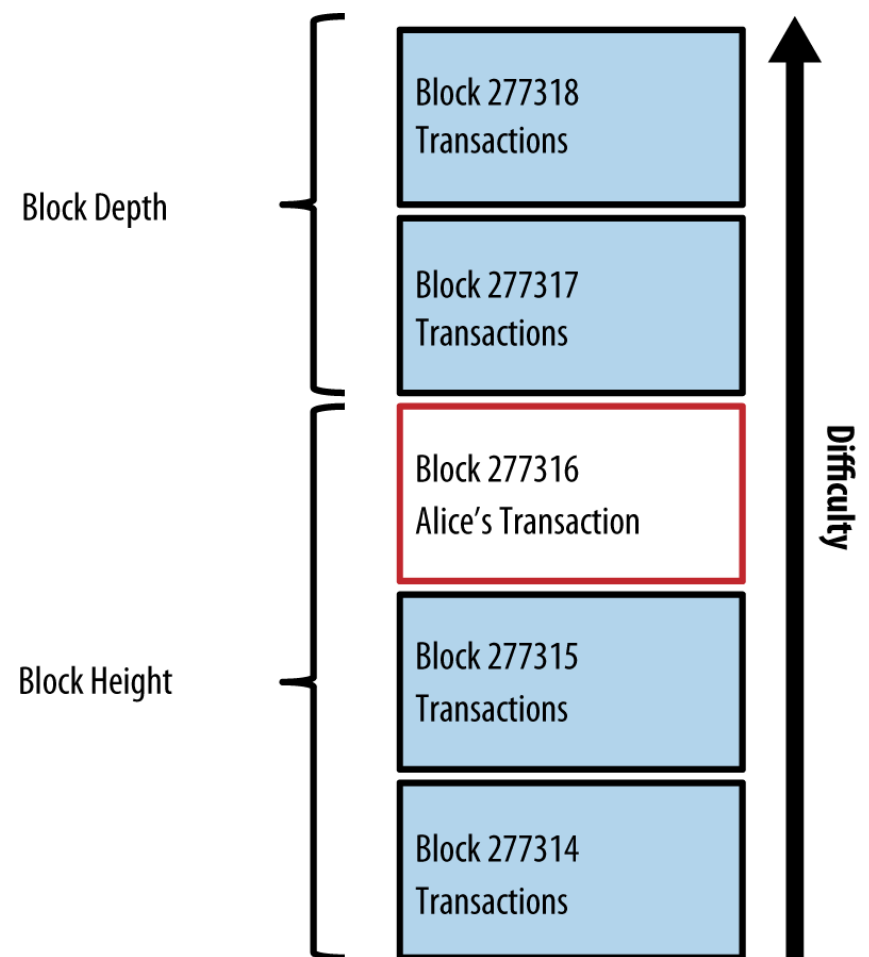
トランザクションプールの中から、手数料が高いトランザクションをいくつか選び、ブロックサイズ1MBに入るだけのトランザクションを詰め込み、マイニングを行う。

マイニングに成功したら、新しいブロックが作られる。作られたブロックは、即時にほかのノードに送信される。新しいブロックを受け取ったノードは、検証を行い、問題なければ新しいブロックとしてブロックチェーンに取り込む。

アリスのトランザクションがブロックに取り込まれた



ブロック高・深度・ファイナリティ



ブロック高

→ 初期ブロックからのブロック数

→ block #277316 : 277,316

ブロック深度

→ 最新ブロックからのブロック数

→ block #277316 : 3

深さが深ければ深いブロックほど翻る可能性が低くなる。

→ ブロック取り消しのためにはより多くの計算が必要

慣例的に6個のブロックがひるがえることはないと言われている

→ ファイナリティ

Figure 9. Alice's transaction included in block #277316

トランザクションのその後

- アリスからボブに支払われた0.015BTCのトランザクション
 - タローのマイニングによりブロックに取り込まれブロックチェーンの一部になった
- ボブは晴れてその0.015BTCをほかの人への支払いに使用できる。
 - トランザクションが取り込まれたブロックが正しいことをボブはウォレットで検証できる。
 - ゼロ番目のブロックから順次正しさの検証ができる。

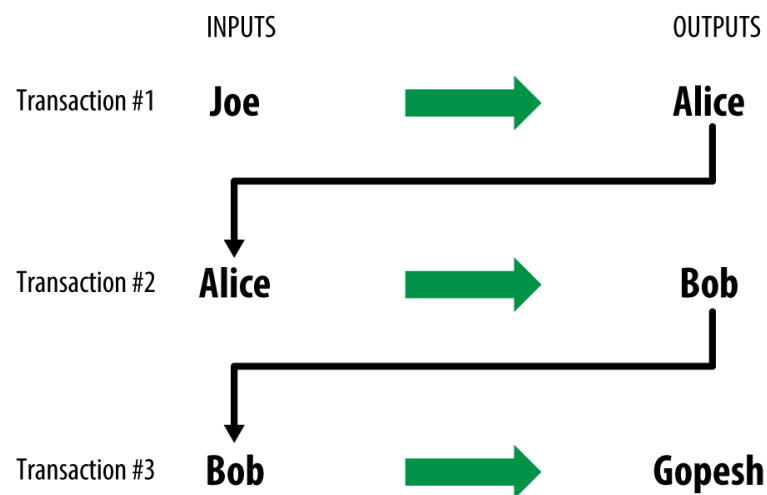


Figure 10. Alice's transaction as part of a transaction chain from Joe to Gopesh

アリスからボブへの送金トランザクションをブロックチェーンで確認

Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK (0.1 BTC - Output)



1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA

- (Unspent)

0.015 BTC

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK -

(Unspent)

0.0845 BTC

97 Confirmations

0.0995 BTC

Summary

Size 258 (bytes)

Received Time 2013-12-27 23:03:05

Included In 277316 (2013-12-27 23:11:54 +9
Blocks minutes)

Inputs and Outputs

Total Input 0.1 BTC

Total Output 0.0995 BTC

Fees 0.0005 BTC

Estimated BTC Transacted 0.015 BTC

<https://www.blockchain.com/btc/tx/0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2>

Figure 8. Alice's transaction to Bob's Cafe

ブロックチェーンに記録されているかどうかでその送金が完了していることを確認できる。
全世界の人に公開され、半永久的に記録される。





まとめ

- アリスがボブへ送金する例で，ビットコインのシステムの流れを概観した
 - ウォレットと鍵
 - トランザクション作成
 - ビットコインネットワークにブロードキャスト
 - マイニングによるブロック生成，伝搬



補足) Bitcoin Core

- Bitcoin coreとはビットコインクライアントのリファレンス実装
 - <http://bitcoin.org/> からDLできる
 - <https://github.com/bitcoin/bitcoin> にソースコードが公開されている.
 - C++で実装されている
- Bitcoin Coreはビットコインシステムのリファレンス実装.
 - つまり, 各機能 (ウォレット, トランザクション, ブロック検証, P2Pネットワーク) の実装をどのようにするか, というお手本になっている



補足) bitcoin explorer

- Bitcoin explorer
 - ビットコイン用の様々なコマンドラインツール
 - bxコマンド
 - ビットコインで使われているハッシュ計算をしたり, 鍵のフォーマットをしたりできる
 - libbitcoinのプロジェクト中にある
 - ビットコインクライアント用の様々なライブラリ
- インストールしておくと便利
- <https://github.com/libbitcoin/libbitcoin-explorer>
- 以下は/home/shibano/bin/libbitcoin以下にインストールするコマンドです(Ubuntu 22.04). 自分の環境で適宜置き換えてください.

```
sudo apt-get install build-essential autoconf automake libtool pkg-config git
git clone https://github.com/libbitcoin/libbitcoin-explorer.git
cd libbitcoin-explorer/
git checkout -b version3 origin/version3
./install.sh --prefix=/home/shibano/bin/libbitcoin --build-boost --build-zmq --disable-shared
```



鍵，アドレス，ウォレット



概要

- 公開鍵暗号方式
- 秘密鍵, 公開鍵, アドレス
- 公開鍵, 秘密鍵
 - 公開鍵
 - 秘密鍵 → 銀行ATMのPINコードやハンコのようなイメージ
 - 自分以外誰にも知られてはいけない
 - ※ あくまでイメージです
- 秘密鍵の作り方
- 公開鍵の計算の仕方
- アドレスの公開鍵からの求め方
- 秘密鍵・公開鍵のフォーマット
- ウォレット
 - 秘密鍵を管理する
 - 鍵の収納, 生成など



公開鍵暗号方式について

- HTTPS通信（SSL/TLS）でも使用されている。
- 代表的なアルゴリズムには、ECDSAに加えてRSAがある。
- ECDSA（Elliptic Curve Digital Signature Algorithm：楕円曲線デジタル署名アルゴリズム）がBitcoinでは採用されている。
- 公開鍵暗号方式とは
 - 秘密鍵と公開鍵のペアを一意に生成できる。
 - 公開鍵は誰にでも渡して問題ない
 - 署名ができる
 - 秘密鍵を持っている人しか生成できない署名文字列を生成
 - その秘密鍵に対応する公開鍵で署名を検証できる。
- ビットコイン所有者はトランザクションに公開鍵と署名を記載する
 - ビットコインネットワークのすべての参加者はその公開鍵と署名をもって検証できる。

秘密鍵, 公開鍵, アドレスの関係

- Bitcoinでは各参加者の口座番号を「アドレス」と表現する.
- アドレスとは27から34文字の文字列からなる.
- 秘密鍵は乱数で生成する.
- 楕円曲線をもとに秘密鍵から公開鍵が生成され, その公開鍵からハッシュ関数を通じてアドレスが導出される.

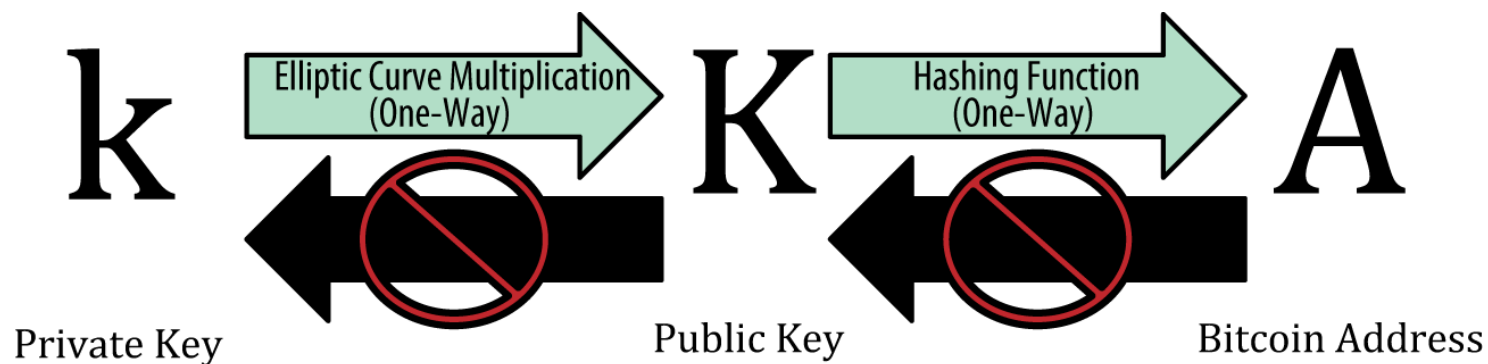


Figure 1. Private key, public key, and bitcoin address

[https://github.com/bitcoinbook/bitcoinbook/blob/second edition_print3/ch04.asciidoc](https://github.com/bitcoinbook/bitcoinbook/blob/second%20edition_print3/ch04.asciidoc)

秘密鍵の生成

- 秘密鍵は自分だけしかもってない（持ってはいけない）鍵
- これを持っている人だけが対応するアドレスから送金可能
- 乱数により生成される.
- バックアップが非常に重要になる一方, 流出しないように守る必要もある.
 - 秘密鍵は同じものを再生成することは不可能.
- 256bitの数
 - 正確には楕円曲線secp256k1の位数として定義される定数 p に対して $p-1$ より小さい数
 - $p = 1.158 * 10^{77}$ で, 2^{256} よりわずかに小さい
 - p でmodを取る
- 乱数で生成すると誰かほかの人の秘密鍵とかぶるのでは？
 - 1から 2^{256} までの数字の中から一つを選ぶ
 - 参考) $136 * 2^{256}$ 乗 → エディトン数 (宇宙に存在する全陽子数)
 - <https://qiita.com/zaburo/items/074995ae8cb81d506222>
- コンピュータで生成するときには注意
 - 疑似乱数を用いて生成すると結構簡単に類推されてしまう.
 - CSPRNG (cryptographically secure pseudorandom number generator) と呼ばれるような実装を用いることが望ましい



秘密鍵と公開鍵について

- 秘密鍵と公開鍵の関係
 - 秘密鍵はスカラー（正の整数）です.
 - 一方，公開鍵は楕円曲線の点 (x, y) です.
 - x も y も正の整数です.
- ということをこれから話します.



公開鍵

- 秘密鍵から導出
- 楕円曲線暗号を使用して導出される.
- NIST策定のsecp256k1という標準で定義された楕円曲線と定数を使っている.
 - NIST: National Institute of Standards and Technology アメリカ国立標準技術研究所

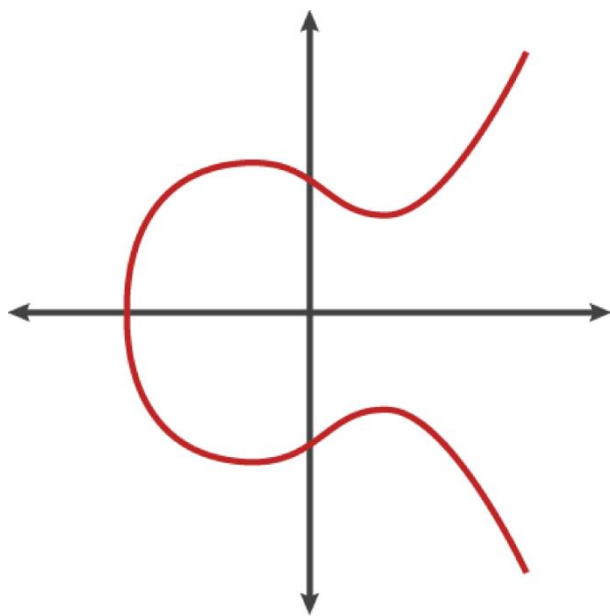


Figure 2. An elliptic curve

$$y^2 \bmod p = (x^3 + 7) \bmod p$$

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

: 位数と呼ばれる. 素数. 2^{256} よりわずかに小さい数

または,

$$y = (x^3 + 7) \text{ over } (\mathbf{F}_p)$$

有限体 \mathbf{F}_p 上で定義されている曲線.
と表現される.

※ この曲線は, 実数ではなく素数位数の有限体上で定義されているため, 2次元にちりばめられたドットパターンのように見える.

例：位数が素数17である楕円曲線

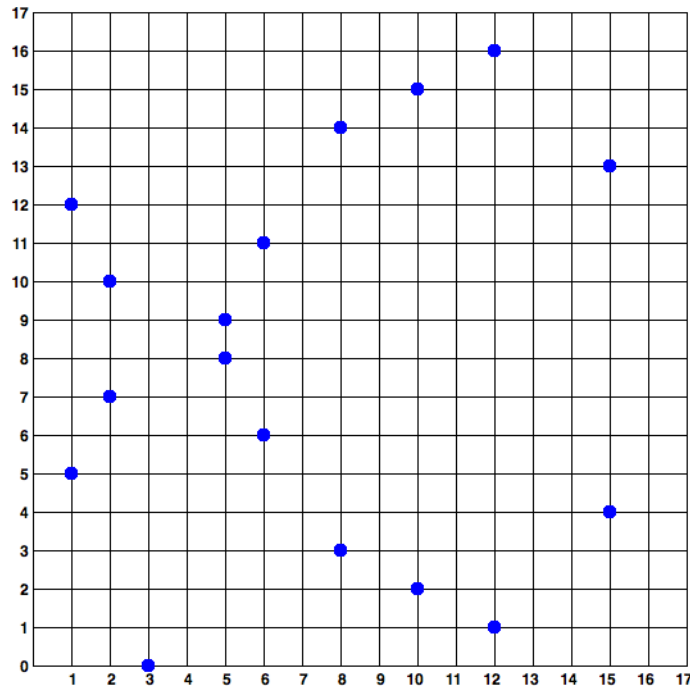


Figure 3. Elliptic curve cryptography: visualizing an elliptic curve over $F(p)$, with $p=17$

$$y^2 \bmod 17 = (x^3 + 7) \bmod 17$$

$x=1$ のとき,
右辺 = $(1+7) \bmod 17$
= 8

$$y^2 \bmod 17 = 8$$

となる y は

$$y = 5, 12$$

$$\text{※) } 25 \bmod 17 = 8$$

$$144 \bmod 17 = 8, (17 \cdot 8 = 136)$$

ビットコインに使われるsecp256k1楕円曲線は、これよりもっと大きいグリッド上にもっと複雑に書かれたドットパターン

例：secp256k1曲線上の点の例

どんな点がsecp256k1の点か？

以下の点Pはsecp256k1上の点.

P = (55066263022277343669578718895168534326250603453777594175500187360389116729240,
32670510020758816978083085130507043184471273380659243275938904335757337482424)

$$y^2 \bmod p = (x^3 + 7) \bmod p$$

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

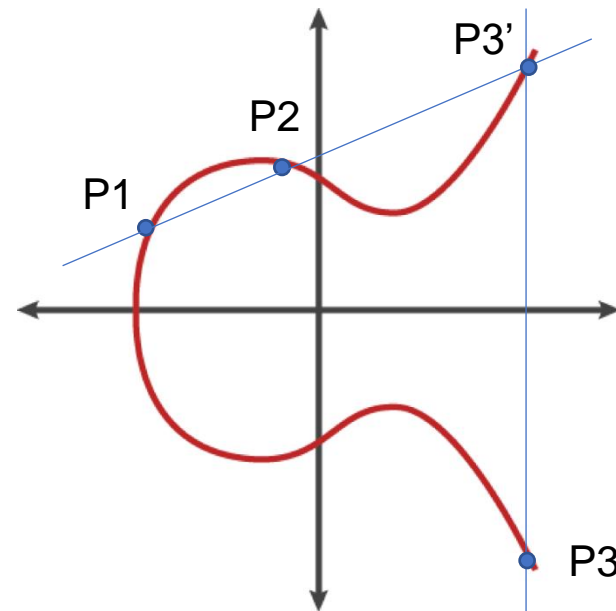
Pythonで計算して確認してみる.

```
shibano@DESKTOP-940THE0:~/git/bitcoin$ /usr/bin/python3.7
Python 3.7.5 (default, Nov 7 2019, 10:50:52)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
>>> y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
>>> p = 2**256 - 2**32 - 2**9 - 2**8 - 2**7 - 2**6 - 2**4 - 1
>>> (x**3 + 7 - y**2)%p
0
>>> p
115792089237316195423570985008687907853269984665640564039457584007908834671663
```

楕円曲線上での演算

- 楕円曲線上の演算として、ゼロ元、加法、正整数倍が定義されている。
- 0
 - 無限遠点と呼ばれる
- +
- 加法
 - $P_3 = P_1 + P_2$ の求め方：
 - P_1 と P_2 を通る直線が P_1 と P_2 とは別に交わる点 P_3'
 - P_3' をx軸に対して対象な点が P_3
- 同じ値の加法
 - $P_1 + P_1$ は？
 - P_1 での楕円曲線の接線を直線として、それと交わる点が P_3' とする。
- 正の整数倍
 - その数だけ加法を繰り返すこと



公開鍵の生成

- 楕円曲線上で、あらかじめ決められた生成元 G (generator point) を秘密鍵 k 倍することで、公開鍵 K を得られる。

$$K = k * G$$

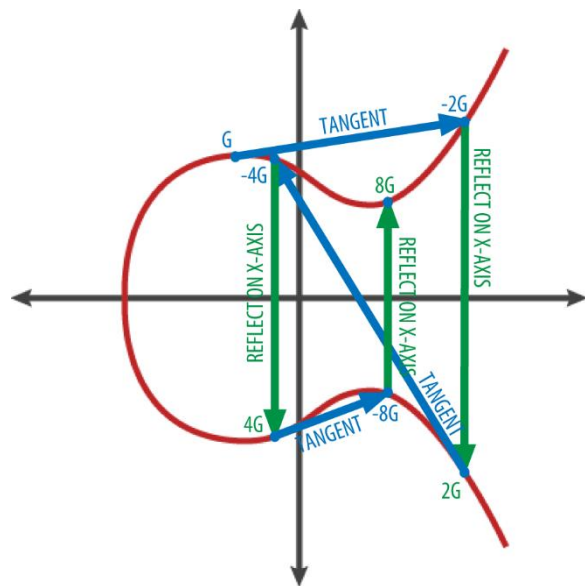
K : 公開鍵

k : 秘密鍵, スカラー

G : 生成元

- 公開鍵 K は (x,y) 座標である。

$G=(0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798, 0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8)$



秘密鍵から公開鍵を計算することは可能。
大きい数であるが正のスカラー倍。

逆に公開鍵をもとに秘密鍵を求めることは現実的には不可能（楕円曲線上の離散対数問題，ECDLP）。

Figure 4. Elliptic curve cryptography: visualizing the multiplication of a point G by an integer k on an elliptic curve

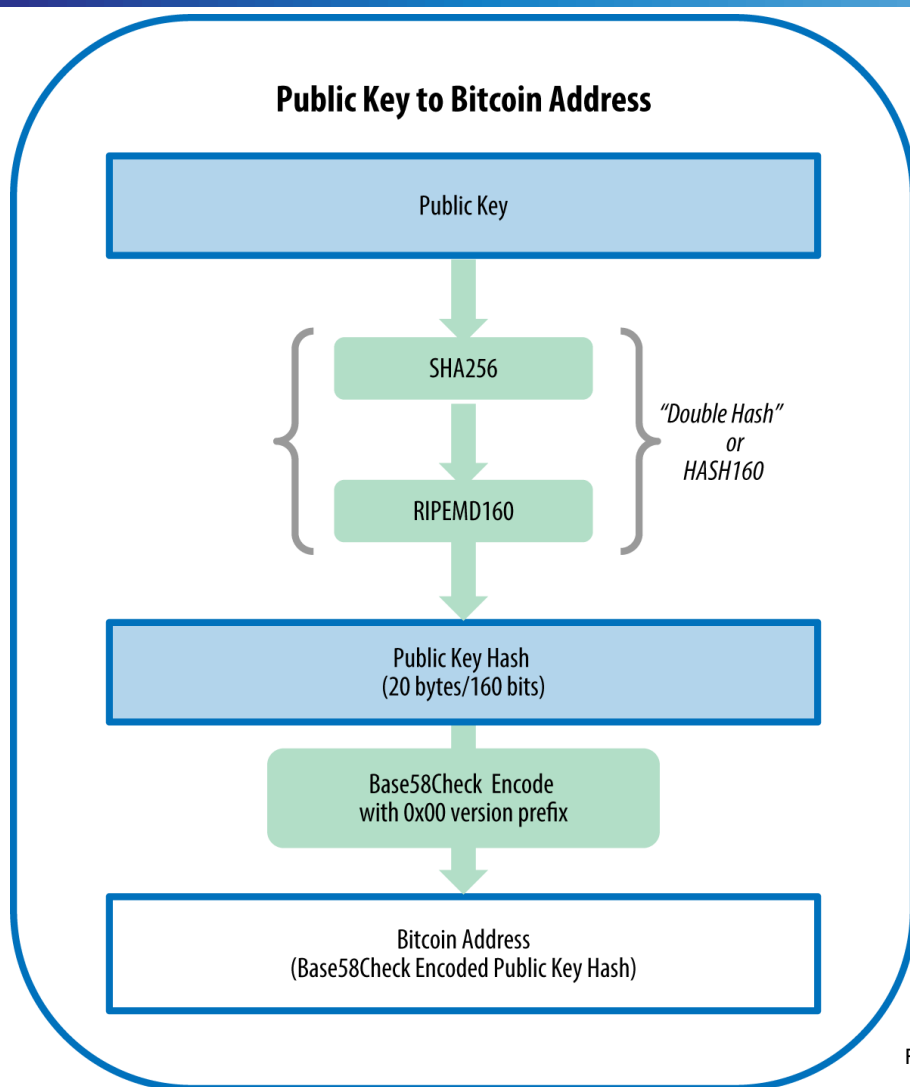
ビットコインアドレス

アドレスは公開鍵から作られる.

アドレスの例：

1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy

公開鍵からアドレスを生成する流れ



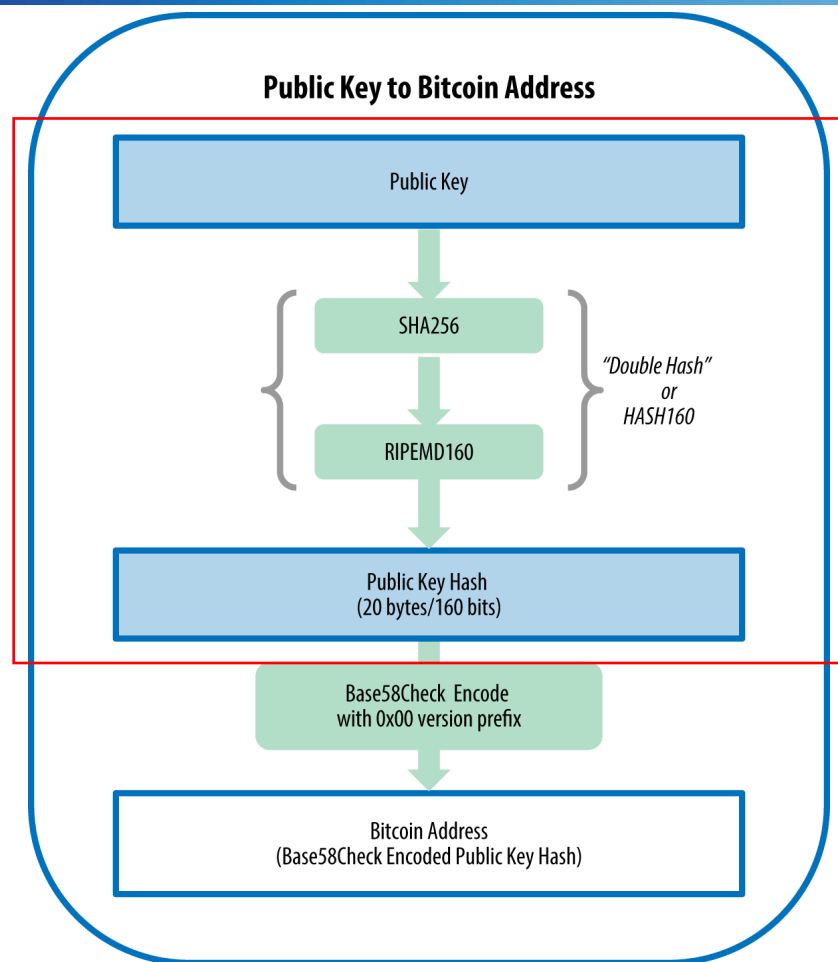
公開鍵からアドレスのもととなるデータを作成し、文字列での表現（エンコード）をします。

文字列で表現できないとあらゆる情報システムで取り扱うことが面倒なためです。
→ メールでの記述など。

ライプエムディー160

Figure 5. Public key to bitcoin address: conversion of a public key into a bitcoin address

公開鍵からアドレスを生成する流れ



公開鍵をSHA256とRIPEMD160という2つのハッシュ関数でハッシュ化する。
ここで入力する公開鍵は後述するフォーマットで表現されたもの。

$H1 = \text{SHA256 (Public Key)}$
:256bit (32Bytes)

$H2 = \text{RIPEMD160}(H1)$
: 160bit (20Bytes)

Figure 5. Public key to bitcoin address: conversion of a public key into a bitcoin address

公開鍵からアドレスを生成する流れ

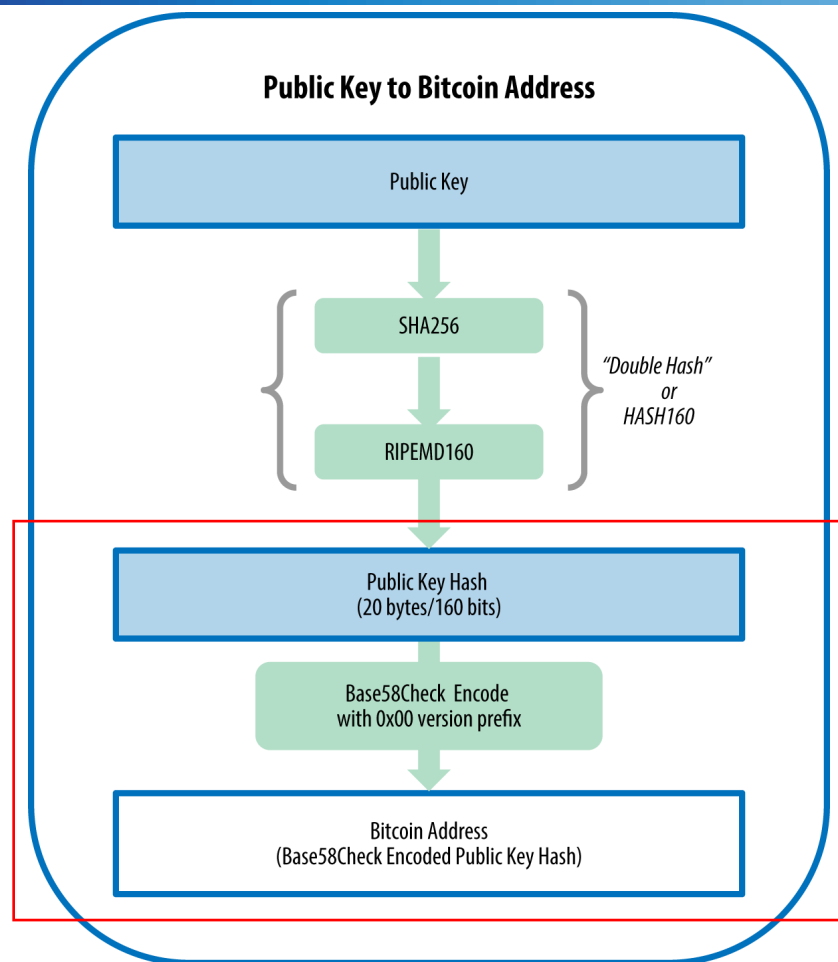


Figure 5. Public key to bitcoin address: conversion of a public key into a bitcoin address

Base58Checkエンコード

エンコード方式とは

- データをアルファベットや数字を組み合わせた文字列で表現する。
可逆的。

- 参考：Base64

Eメールで添付データを表現するのに使用されている。

26個のアルファベット小文字, 26個の大文字, 10個の数字, 「+」 「/」 の2個の記号

表現できる文字の種類が大きければ多いほど短く表記できる。


エンコード方式：Base64の例

- 例えば画像をBase64形式で表現してみる.

Base64エンコーダー

公開日: 2015/08/21 | 更新日: 2017/07/26

画像をbase64エンコードするツールです。ローカル環境で処理するため、画像を当サーバーにアップロードする必要がありません。

| 画像 | Data URI |
|--|---|
|  | <pre>KsoOpTnNemUHcCjS4hloLHVJ12GmhquwS1f3dNKy7CL7IRPnhsj7eoJaNfr9BsuFT6Kg21vJ2VV DabZ4kJrDR74TZpbVJaaUoL+r1VG0FrAcy89BCsUlftmVh3xBC9ImO53T87Og5EYhMHTP4rdF3 7vdfFW3lnYoXQSk7W+cwQA3luJ5deePY9P/prf/UnNxzQAPCx Dz7xzxDi+73DEZhPex9CJRpyyl fAfFRRiSs7R3TkfXgDag3w7pmcchdjYCasialjrjnEEOc5Va5MXPl6KPvu9CPU/5E7MLb05yveE9g YCbQfY559oHSOKYejK9WVDhHxLne6YJ3KZU7AHw6dvFe792XN8ebTRpn54J/Ydjb+2ECJ3D9 yriZTvr1cD+Dv5Bzqc7Rmio+6zv/8Lyda57nOXThnAsxOuee854OasWjoYsn43b8K8fIDINuq4yr 3rsrAO11fX+xlOxKym8E+Bk4d1+e80fj0D3snGNyfq/M837wdFSYDnwMnwvePZjG+ZO+CyvO 9bAwf51r3SPg9JxTchSOus4PaU57jqifUznpAu2TDzWXMscYuFZ+N53z67nUGvvuxRj9XXOqV2 P0p1LJz3bO35Nr3bjoX0QqrysVQxizhm5es51oBD3S8lbqvUE5L6LwZfSOG/J0UTOHcXg7siZX Yzhak5zgneXHfk7K5cQvR8z01xz5n41nEXFX1cufcn1jn6lqMyn51Q+7oB7GZyosgOXnl04lvA95 Pwq53QxOJ/l+dkRHxauEyo6F3x13n19PN6cGVbDAPDVObMPwSU46nPKxQFXQhcP0jw/985 f+MOff7W4fe3x2uO1x2uP1x6vPV57vPZ47fHa47XHjT/+P3I98UJs+NpCAAAAAEIFTKSuQmCC</pre> |

<https://lab.syncer.jp/Tool/Base64-encode/>

https://www.irasutoya.com/2020/12/blog-post_0.html

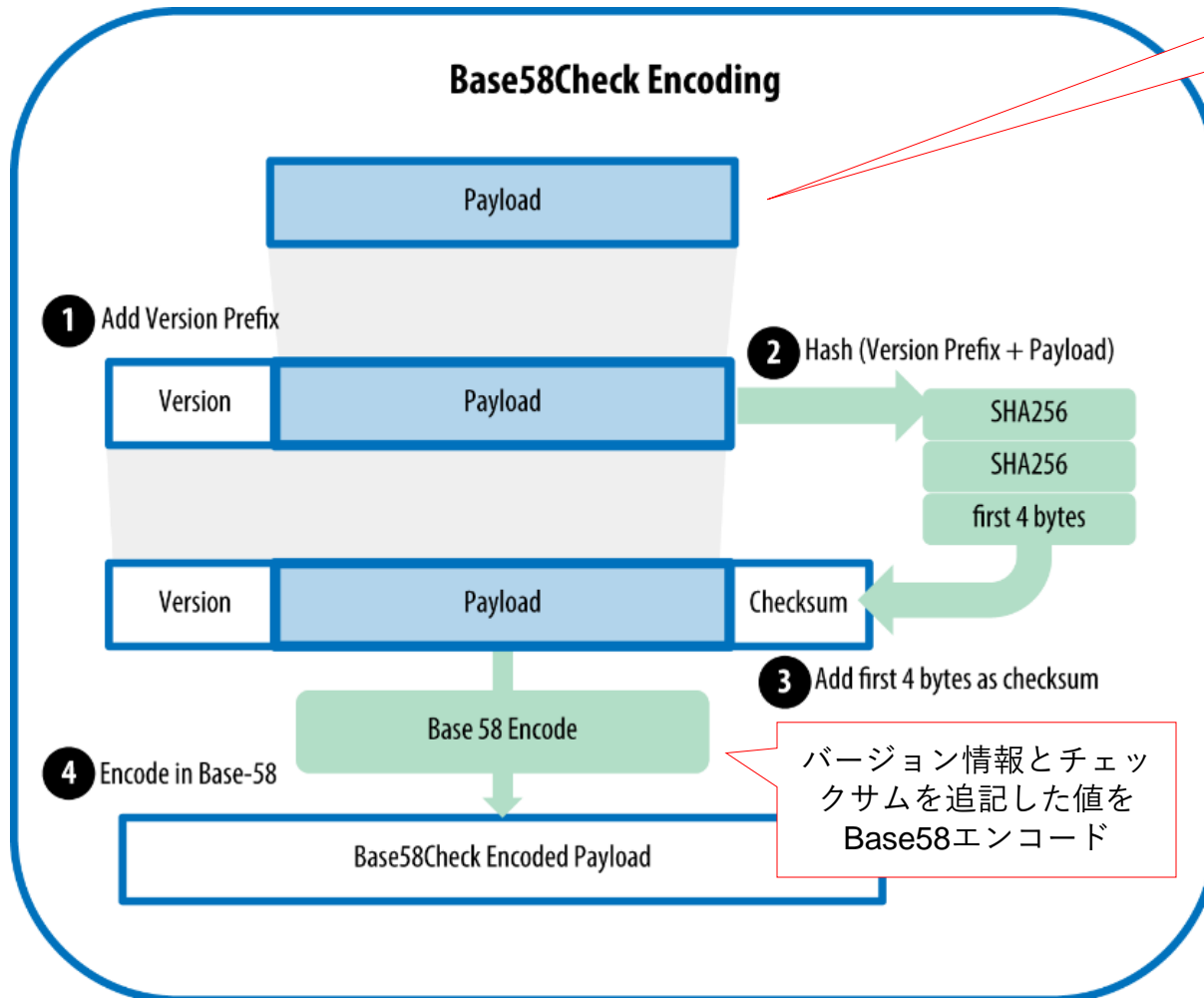
Base58, Base58Check

- Base58
 - ビットコインで使うために開発されたテキストベースのエンコード形式
 - バイナリデータ（バイト列）をテキストで表現.
 - 58文字で表現.
 - 1文字で0から57までの58個の値を表現できる.
 - Base64に比べ, 見間違いやすい6文字が省かれている
 - 0（ゼロ）, O（大文字オー）, l（小文字エル）, I（大文字アイ）, 「+」, 「/」
 - Base58エンコーダ・デコーダを使うとどんな値がどんな文字列に変換されるのかわかる.
 - <https://appdevtools.com/base58-encoder-decoder>
- Base58Check
 - Base58にバージョン情報と, チェックサムが組み込まれたエンコード形式
 - チェックサムとは
 - エンコードしようとしているデータの最後に追加される4バイトの値
 - 元データのハッシュから生成
 - 転写間違いやタイプミスの検出・防止に使用される



Base58Check Encoding

アドレスの場合は、
公開鍵のハッシュ
(20バイト=160bit)



| Type | Version prefix (hex) | Base58 result prefix |
|------------------------------|----------------------|----------------------|
| Bitcoin Address | 0x00 | 1 |
| Pay-to-Script-Hash Address | 0x05 | 3 |
| Bitcoin Testnet Address | 0x6F | m or n |
| Private Key WIF | 0x80 | 5, K, or L |
| BIP-38 Encrypted Private Key | 0x0142 | 6P |
| BIP-32 Extended Public Key | 0x0488B21E | xpub |

Table 1. Base58Check version prefix and encoded result examples

- バージョン情報:
アドレスの種類について一目で分かるように
- チェックサム:
転記ミスを検出できるように
- Base58:
人が読みやすいように

Figure 6. Base58Check encoding: a Base58, versioned, and checksummed format for unambiguously encoding bitcoin data

秘密鍵のフォーマット

- 秘密鍵も公開鍵も，いくつかの異なる形式で表現される。
- 見た目が異なっていたとしてもすべて同じ秘密鍵の値（楕円曲線におけるスカラー）を表現している。
- 人々が間違えずに容易に読み転写できることを目的。
 - 加えてソフトウェアでも形式を間違えないように認識できるように
- 秘密鍵はこれです！と渡されたときにどの形式なのかは文字列の最初の文字で判別できるようになっているものもある。

WIFはWallet
Import Format

ここでの圧縮
(compressed) の
意味は，秘密鍵では
なく公開鍵のデータ
容量が少なくなるこ
うこと（後述）。
圧縮された公開鍵に
紐づく秘密鍵。
ペイロード部の最後
に0x01を追記。

| Type | Prefix | Description |
|----------------|--------|---|
| Raw | None | 32 bytes = 256bit |
| Hex | None | 64 hexadecimal digits |
| WIF | 5 | Base58Check encoding: Base58 with version prefix of 128 (0x80)- and 32-bit checksum |
| WIF-compressed | K or L | As above, with added suffix 0x01 before encoding |

Table 2. Private key representations (encoding formats)

1バイトは8bit.
256bitは32バイト。

16進数2文字(00～ff)で1バイト

32バイトの値は16進数表記で2文
字ずつのペアが32個分
→ 64文字

秘密鍵のフォーマット（具体的な値の例）

- Hexと，WIF，WIF-compressedの値が相互に変換できることを確認していきます。

| Format | Private key |
|----------------|--|
| Hex | 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd |
| WIF | 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn |
| WIF-compressed | KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ |

Table 3. Example: Same key, different formats

WIF，WIF-compressed形式からHexに変換して値が一致することを確認

```
$ bx wif-to-ec 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

$ bx wif-to-ec KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
```



秘密鍵のフォーマット：Base58Checkから16進数へのデコード

- Base58check-decode オプションを使用するとバージョン情報とchecksumの値もわかる。

WIF形式の秘密鍵

```
$ bx base58check-decode
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
wrapper
{
  checksum 4286807748
  payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd
  version 128
}
```

バージョン情報：128 =
0x80はWIF形式ということ

圧縮WIF形式の秘密鍵

```
$ bx base58check-decode
KxFC1jmwWCoACiCAWZ3eXa96mBM...sTYzGmf6YwgdGWZgawvrtJ
wrapper
{
  checksum 2339607926
  payload 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01
  version 128
}
```

圧縮された秘密鍵は最後に
01がついている

秘密鍵のフォーマット ：16進数からBase58Checkエンコードをして確認してみる

- bxでBase58check-encode オプションを使用

WIF形式の秘密鍵

```
$ bx base58check-encode 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd --version 128  
5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbkeyhfsYB1Jcn
```

WIFの秘密鍵は最初が「5」

WIF形式のバージョン情報：128を指定

圧縮WIF形式の秘密鍵

```
$ bx base58check-encode 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 --version 128  
KxFC1jmwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawrtJ
```

圧縮WIFの秘密鍵は最初が「K」



公開鍵のフォーマット：圧縮されていない公開鍵

- 公開鍵
 - 圧縮されていない公開鍵と圧縮された公開鍵の2つの表現がある
 - 公開鍵は楕円曲線上の点(x, y)である.

圧縮されていない公開鍵

04は「圧縮されていない公開鍵」
の意味

- プレフィックス0x04 (1バイト) に, 256[bit]ずつx座標y座標と続いて表現
 - 520[bit] (33バイト) = 8[bit] (1バイト) + 256[bit] (32バイト) + 256[bit] (32バイト)

公開鍵の座標は秘密鍵を用いて求められる.

x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

この場合公開鍵は520[bit]

04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

と表現される.

\$ bx ec-to-public -u 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd

04f028892bad7ed57d2fb57bf33081d5cfcf6f9ed3d3d7f159c2e2fff579dc341a07cf33da18bd734c600b96a72bbc4749d5141c90ec8ac328ae52ddfe2e505bdb

公開鍵のフォーマット：圧縮された公開鍵

- 送金時には、トランザクション中に公開鍵を含める必要がある。
- 圧縮されていない公開鍵は520bit(65バイト)と大きく、できればもっと小さくしてブロックに詰め込みたい。
- 圧縮された公開鍵は264bit(33バイト)で表現される。
 - 圧縮された公開鍵を使用している人は、トランザクションサイズが小さくなるので手数料も安くすむ
 - プレフィックス 8bit(1バイト) + x座標256bit(32バイト)
 - 楕円曲線上のy座標は式を解くことで導出できるためxさえわかっているならばyは省略可能

$$y^2 \bmod p = (x^3 + 7) \bmod p$$
 - プレフィックスが02または03
 - 方程式のyの解は2つ
 - y座標は偶数または奇数になる
 - 位数pは素数（奇数）であり、マイナスを取った値は偶奇が逆転
 - プレフィックス02：偶数，03：奇数 と表現する

04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A07CF33DA18BD734C
600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
= 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

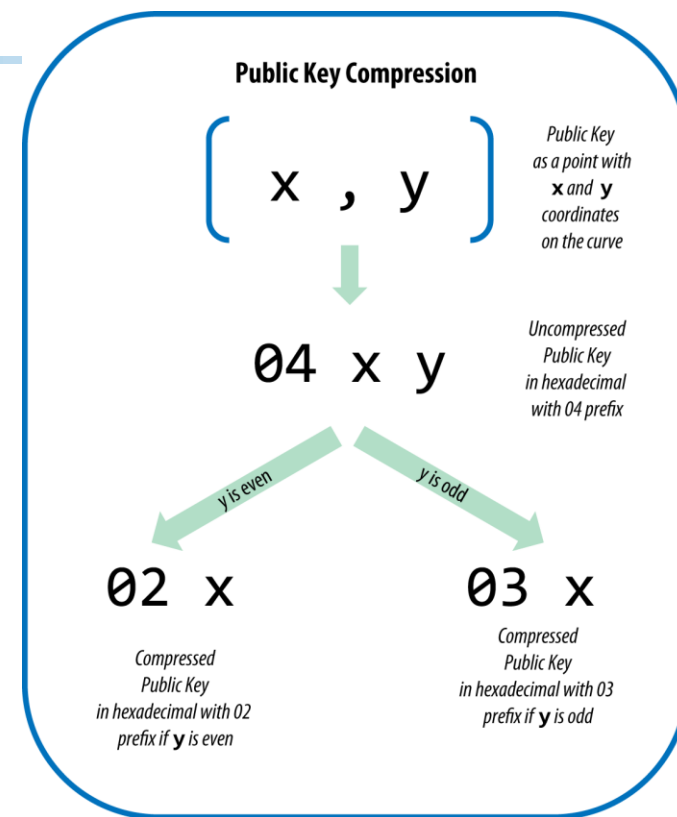
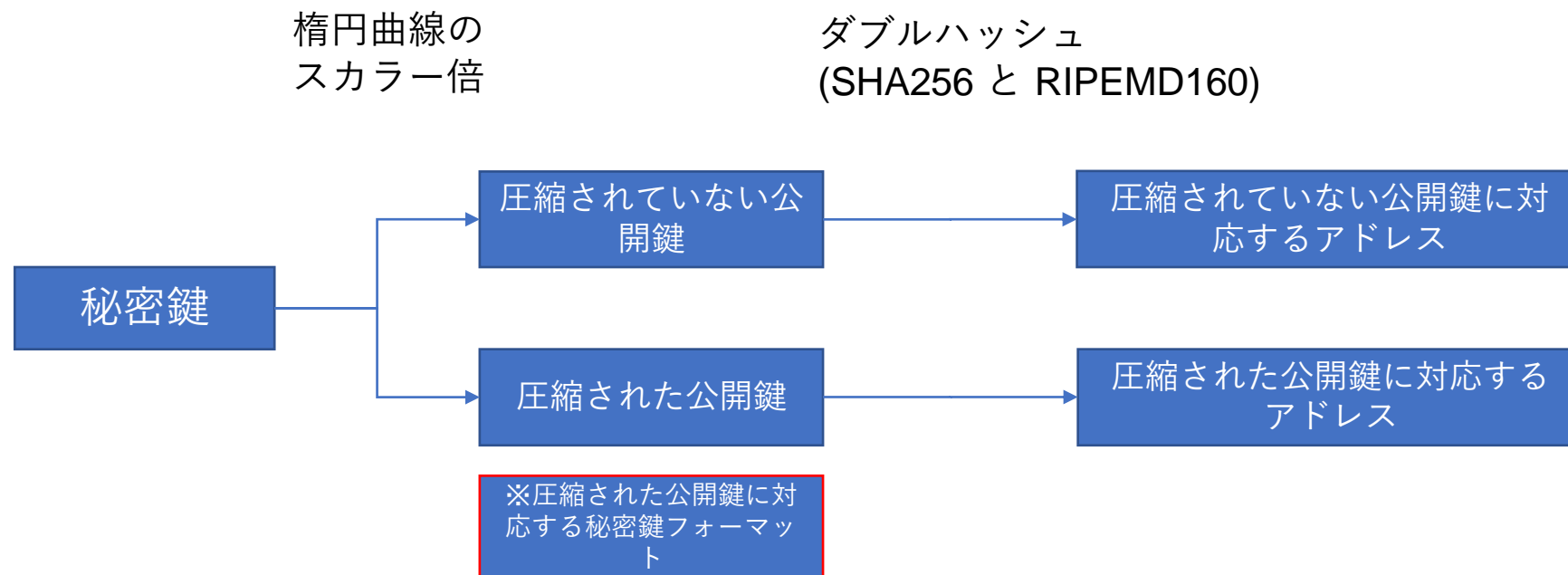


Figure 7. Public key compression

\$ bx ec-to-public
1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc3
2c8ffc6a526aedd
03f028892bad7ed57d2fb57bf33081d5cfcf6f9ed3d3d7f159c2
e2fff579dc341a

圧縮されていない・圧縮された公開鍵に対応する秘密鍵，アドレスについて



※ 圧縮された公開鍵の仕組みは比較的新しいもので，ウォレットによっては対応していないものがある。

圧縮されていない・圧縮された公開鍵に対応する秘密鍵，アドレスの具体例

| | 秘密鍵 (HEX) | 秘密鍵 (WIF) | 公開鍵 | アドレス |
|------------------------------|--|--|--|--|
| WIF (非圧縮の公開鍵) | 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd | 5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2Jpb nkeyhfsYB1Jcn prefixが「5」 | 04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB | 1424C2F4bC9JidNjjTUZC bUxv6Sa1Mt62x |
| WIF compressed (圧縮された公開鍵) | 1e99423a4ed27608a15a2616a2b0e9e52ced330ac530edcc32c8ffc6a526aedd01 | KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ prefixが「K」または「L」 | 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A | 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy |

公開鍵からアドレスを導出するコマンド

```
$ bx ec-to-address
04F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB
1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x
$ bx ec-to-address 03F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

ウォレット

- ウォレットは秘密鍵を管理し，トランザクションが生成できるツール
 - ビットコイン自体が入っているわけではない。
 - 有効な送金トランザクションを作成するためには秘密鍵で署名を行う必要がある
- ビットコインのアドレスはプライバシー保護のために再利用は避けたい
- 同じアドレスを使い続けると多くのトランザクションに結びつき，プライバシーの低下に結びつく。
 - 例：毎日〇時頃に同じコンビニで利用している，など。
- 複数の秘密鍵を生成・管理する方法で，2種類のウォレットがある
 - 非決定性（ランダム）ウォレット
 - 決定性ウォレット → こちらを中心に紹介

非決定性（ランダム）ウォレット

- 秘密鍵を1つずつ乱数で生成するもの
- Type-0非決定性(non-deterministic)ウォレット, ランダムウォレットと呼ばれる
- Bitcoin Coreでは初回起動のときに100個のランダムな秘密鍵が生成される.
- アドレスごとに1つずつの秘密鍵を保持する必要がある.
 - バックアップする際はすべての秘密鍵を保管する必要がある
- このウォレットの使用はBitcoin Core開発者からは推奨されていない.

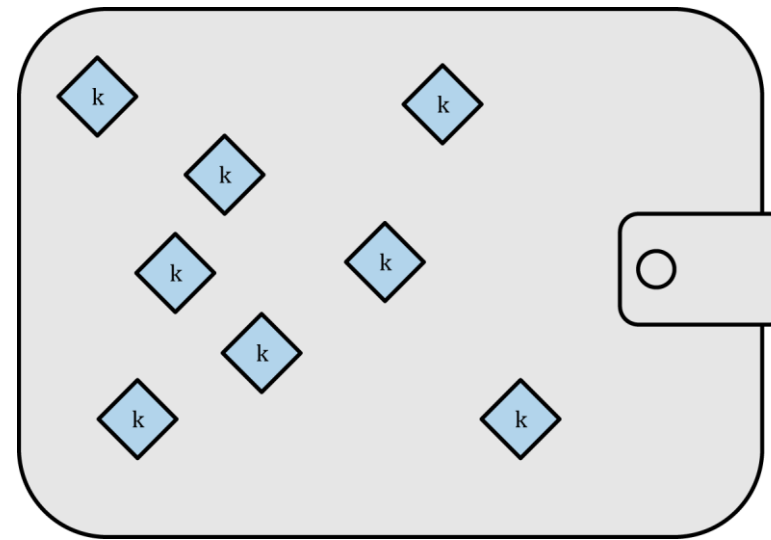


Figure 1. Type-0 nondeterministic (random) wallet: a collection of randomly generated keys
https://github.com/bitcoinbook/bitcoinbook/blob/second_edition_print3/ch05.asciidoc

k:鍵, Key

決定性 (deterministic, seeded) ウォレット

- 決定性(deterministic)ウォレット, Seededウォレットは1つの共通のシードからハッシュ関数を用いて導出される秘密鍵が入ったウォレット.
- シード:
 - ランダムに生成される数値
- シード値と, インデックス番号, 「chain code」というデータと組み合わせて複数の秘密鍵を生成できる (HD Wallet, 後述)
- シード値さえあれば, 複数の秘密鍵を復元することができる.
- バックアップはシードのみすればOK
 - 厳密には鍵の生成順序も
- 同じ仕組みのウォレットであれば, ウォレット間でシード値を移動すれば再利用可

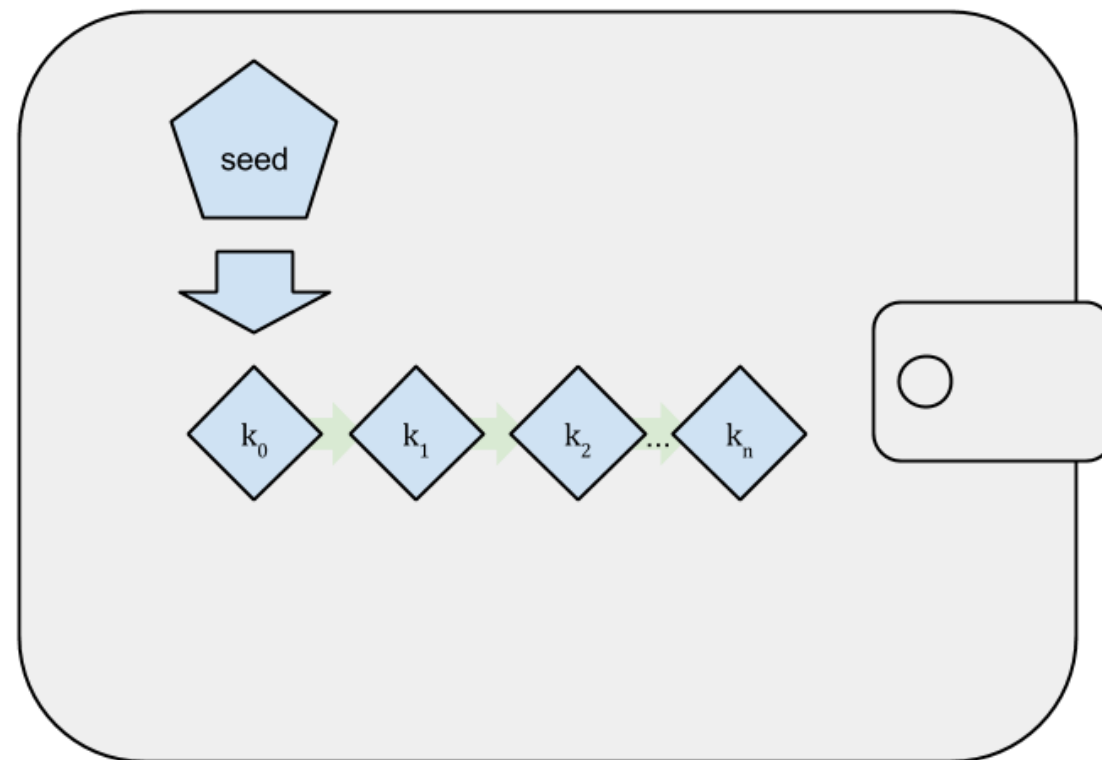


Figure 2. Type-1 deterministic (seeded) wallet: a deterministic sequence of keys derived from a seed

ニーモニックコードワード (Mnemonic Code Words)

- ニーモニックコードはシードを表現する英単語列

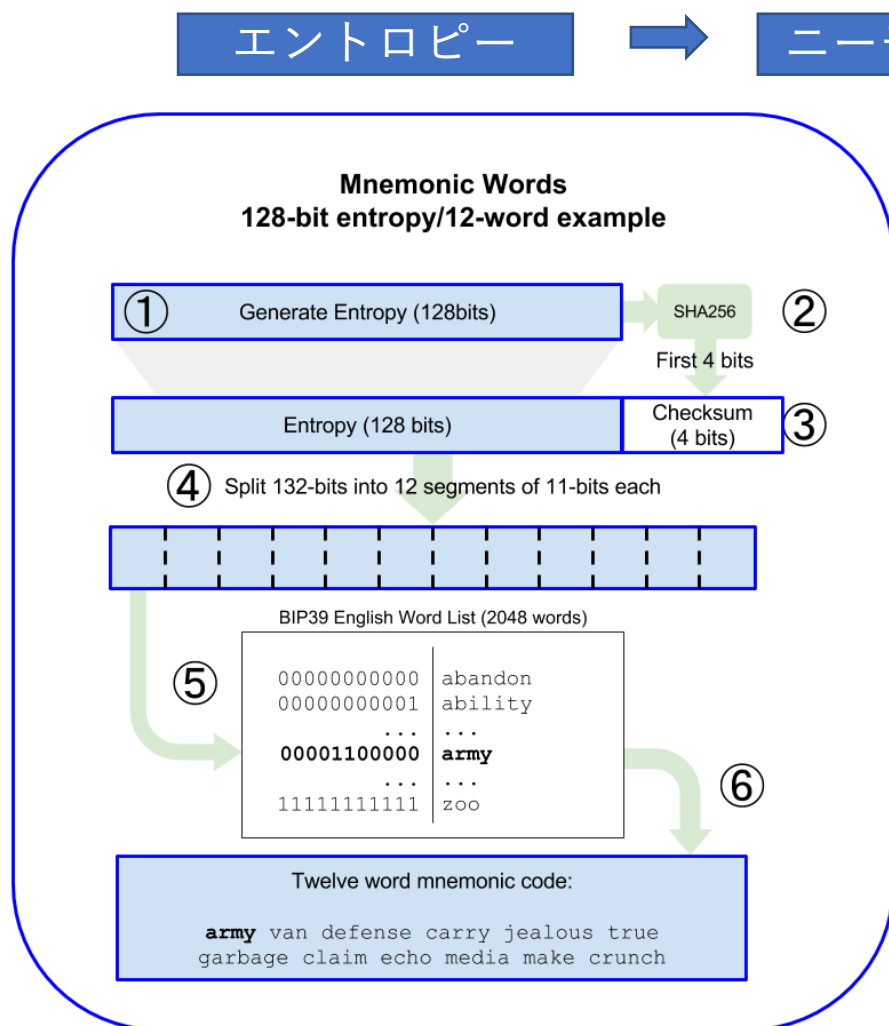
例 12 単語：

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

- 12から24個の英単語の列で表現される.
 - 読みやすく正確に転写できる.
 - 単なる乱数では転記ミスが発生することが多い.
 - 128ビットから256ビットのランダムな値 (エントロピー) を表現.
 - エントロピーと同じ自由度で, ニーモニックワードの単語組み合わせが表現される.
 - BIP-39(Bitcoin Improvement Proposal)で定義されている



ニーモニックコードの導出



- 128bitエントロピー/ 12単語 の例

- ① ランダムな128ビット列（エントロピー）を生成
- ② SHA256でハッシュ化，最初の4ビットを取り出す
- ③ エントロピーの最後にチェックサムとして追記
- ④ 132ビットを11ビットずつの12個に分割
- ⑤ もともと存在している2048wordsからなる英単語リストで11ビットそれぞれに対応する単語を取得

11ビットは $2^{11} = 2048$ 通り

- ⑥ 12単語で構成されるニーモニックコードが得られる



Figure 6. Generating entropy and encoding as mnemonic words

ニーモニックコードからSeed値(512bit)を得る

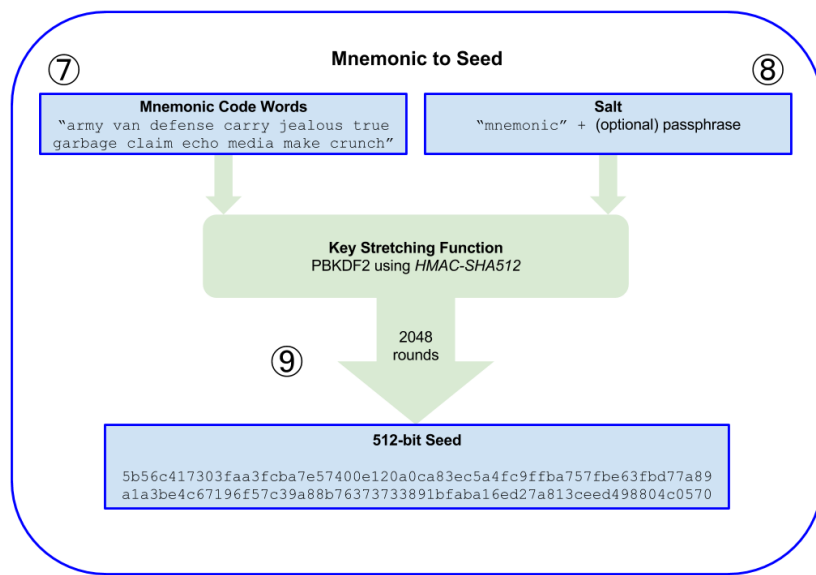
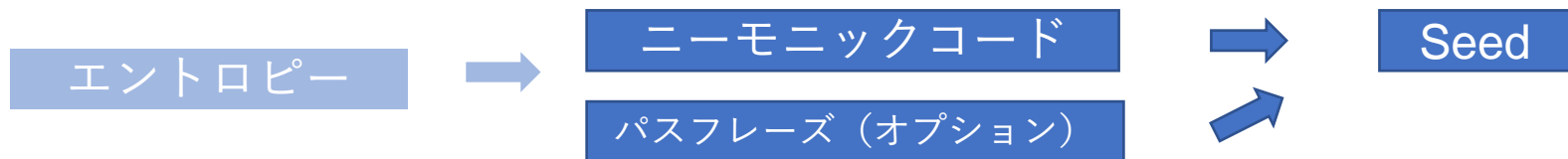


Figure 7. From mnemonic to seed

覚えやすいということは流出しやすい。
ニーモニックの一部が流出してしまったとき
でも簡単にはSeed値を復元できなくしている

⑦ ニーモニックコードを並べる

⑧ Salt値を加える. ここでは任意のパスフレーズを入れることができる.

⑨ Key Stretching Functionを用いてSeed値 (512bit) を生成

※ Key Stretching FunctionはPBKDF2 using HMAC-SHA512
平たく言うと, ハッシュ化を2048回繰り返している

たくさんハッシュ関数を通す理由はブルートフォースアタック対策

(パスフレーズに違う値を何度も何度も入れて元の入力値を探しだすこと.

ハッシュ計算回数を2048回にしておくことで, ブルートフォースにかかる時間が1回の場合にくらべて2048倍になる.

ニーモニックコードとSeed値の例

| | |
|---------------------------------|--|
| Entropy input (128 bits) | 0c1e24e5917779d297e14d45f14e1a1a |
| Mnemonic (12 words) | army van defense carry jealous true garbage claim echo media make crunch |
| Passphrase | (none) |
| Seed (512 bits) | 5b56c417303faa3fcba7e57400e120a0ca83ec5a4fc9ffba757fbe63fbd77a89a1a3be4c67196f57c39a88b76373733891bfaba16ed27a813ceed498804c0570 |

Table 3. 128-bit entropy mnemonic code, no passphrase, resulting seed

| | |
|---------------------------------|--|
| Entropy input (128 bits) | 0c1e24e5917779d297e14d45f14e1a1a |
| Mnemonic (12 words) | army van defense carry jealous true garbage claim echo media make crunch |
| Passphrase | SuperDuperSecret |
| Seed (512 bits) | 3b5df16df2157104cfdd22830162a5e170c0161653e3afe6c88defeefb0818c793dbb28ab3ab091897d0715861dc8a18358f80b79d49acf64142ae57037d1d54 |

Table 4. 128-bit entropy mnemonic code, with passphrase, resulting seed

| | |
|---------------------------------|---|
| Entropy input (256 bits) | 2041546864449caff939d32d574753fe684d3c947c3346713dd8423e74abcf8c |
| Mnemonic (24 words) | 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 | (none) |
| Seed (512 bits) | 3269bce2674acbd188d4f120072b13b088a0ecf87c6e4cae41657a0bb78f5315b33b3a04356e53d062e55f1e0deaa082df8d487381379df848a6ad7e98798404 |

Table 5. 256-bit entropy mnemonic code, no passphrase, resulting seed

階層的決定性ウォレット, HDウォレット (BIP-32, BIP-44)

- 決定性ウォレットの一つ
- 階層的決定性(hierarchical deterministic)ウォレットと呼ばれ, HDウォレットと略される.
- 鍵がツリー構造をなしている
 - 親鍵が子鍵群を生成し, それぞれの子鍵が孫鍵群を生成
 - ここでの鍵とは秘密鍵

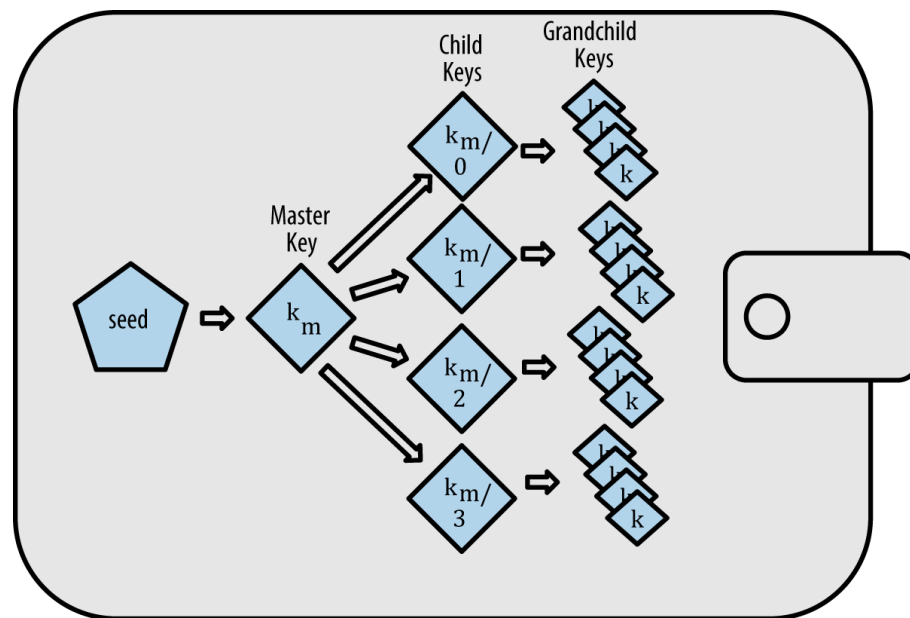


Figure 3. Type-2 HD wallet: a tree of keys generated from a single seed

HDウォレットの利点

- 用途別に階層構造でアドレスを割り当て
 - ツリー構造の、とあるブランチを支払いの受け取りに使ったり
 - 別のブランチを支払いの際のおつりの受け取りに使ったりと用途を分けることができる.
 - また、例えば企業が利用する際に、部や課などの部門ごと、特定の機能ごと、口座ごとにブランチを割り当てることができる.
- 秘密鍵へのアクセスなしで公開鍵を複数生成できる
 - 安全でないサーバーや受取用のサーバーでウォレット運用を想定するケースでも使用できる
 - 公開鍵に関わる情報のみで、複数の公開鍵を運用・管理できる.
 - サーバーは秘密鍵の保持が不要

シードからマスター鍵生成

- シード512bit
 - ニーモニックコード+パスワードをハッシュ化した512bit
- このシードさえあればHDウォレット全体を再生成できる
- マスター鍵の生成の仕方は以下：

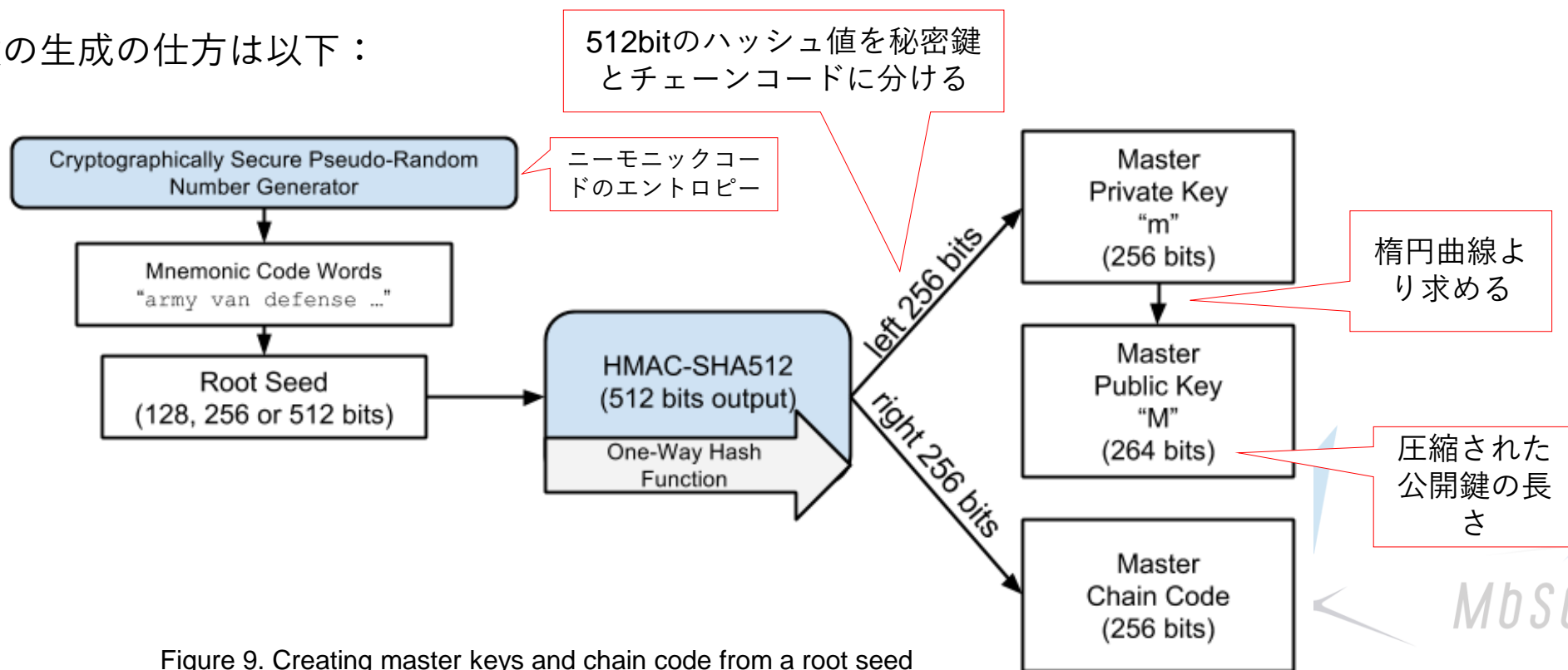


Figure 9. Creating master keys and chain code from a root seed

子秘密鍵の導出

- 子秘密鍵は親がいくつも生成可能.
- 子の秘密鍵は親の秘密鍵, チェーンコード, インデックスから導出される
 - インデックス: 子0, 子1, 子2, ... という子の順番.
 - インデックスを変えることで約20億個もの子鍵を生成できる (4バイト正整数値の最大値の半分)

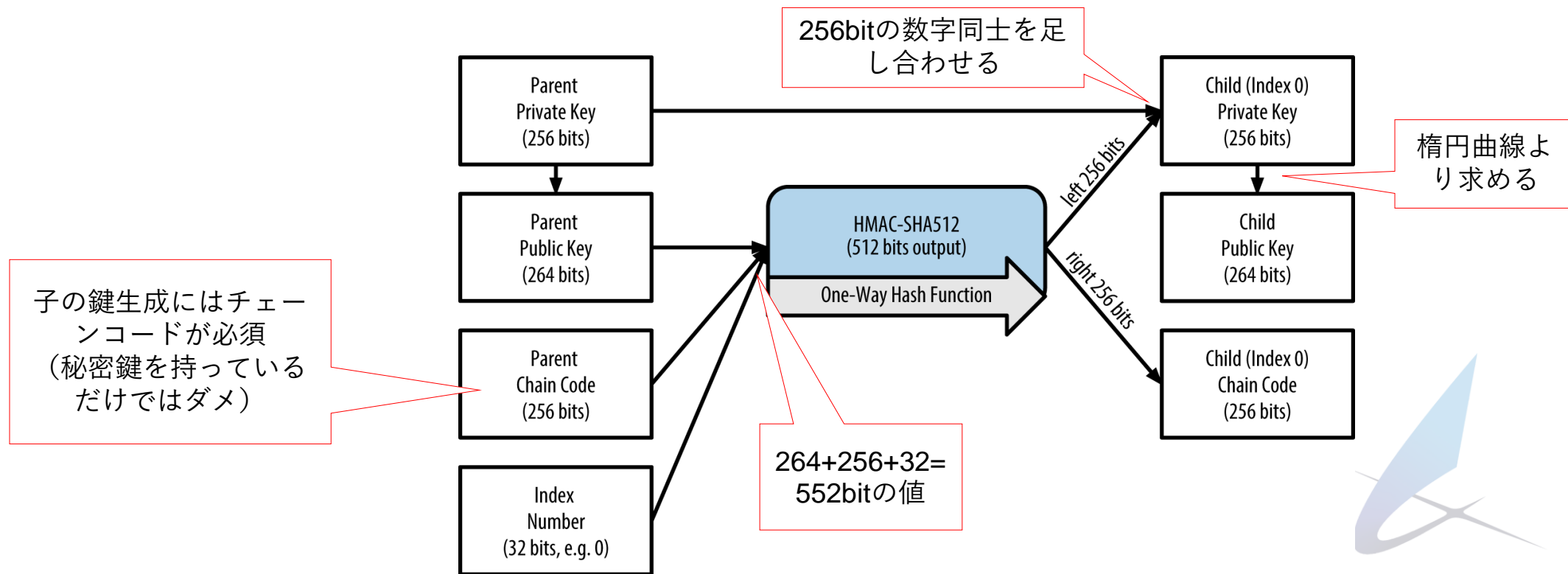


Figure 10. Extending a parent private key to create a child private key

導出された子鍵の使用

- 子秘密鍵
 - 非決定性ウォレットにおける鍵と見分けはつかない
 - 子秘密鍵を持っても親，兄弟の秘密鍵は導けない
 - 唯一，親秘密鍵とチェーンコードを用いて子（自分）の秘密鍵を導出できる
- 子秘密鍵の用途
 - そのアドレスからの支払いの署名
 - 子公開鍵＋アドレスの導出

※ こんな複雑な処理をしなくとも，例えばSHA256を繰り返し計算することにして，繰り返し数で子・孫・ひ孫・・・を計算するということでも階層型で秘密鍵を導出することは可能では？という疑問がわく．

→ チェーンコードを導入することで，途中の秘密鍵のみが万が一流出してしまった場合においてもチェーンコードの同時に流出しない限り，子以下の秘密鍵の導出はできない．

→ 次ページで紹介する，拡張公開鍵のみで子の公開鍵の導出は実現できない．

拡張鍵

- 鍵 + チェインコードを拡張鍵と呼ぶ。
- 拡張鍵は2種類：
 - 秘密鍵 + チェーンコード
 - 子の秘密鍵の生成が可能，もちろん公開鍵も生成可能
 - $256\text{bit} + 256\text{bit} = 512\text{bit}$ の長さ
 - これが流出すると，子以下すべての秘密鍵が復元できてしまうので注意。
 - 公開鍵 + チェーンコード
 - 同じく512bit
 - これを用いることで，秘密鍵なしで子の公開鍵の生成が可能



拡張公開鍵：親公開鍵から子の公開鍵の導出

- 拡張公開鍵で子の公開鍵を導出

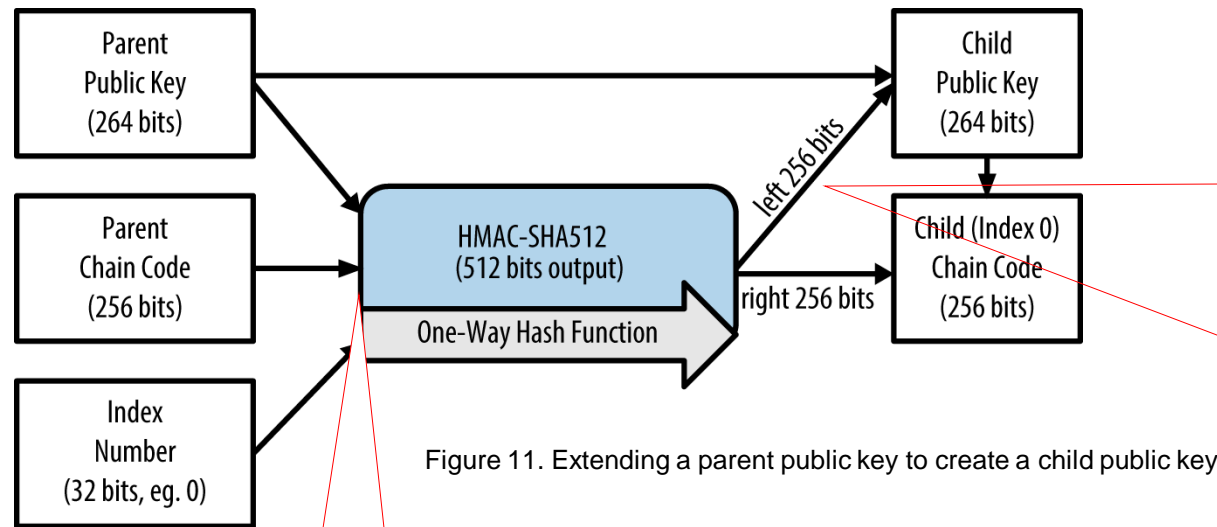


Figure 11. Extending a parent public key to create a child public key

264+256+32
=552bitの値
(子秘密鍵の導出
と同じ値)

$$K_{parent} = k_{parent} * G$$

$$K_{child} = k_{child} * G$$

$$k_{child} = k_{parent} + h_{left}$$

$$K_{child} = (k_{parent} + h_{left}) * G$$

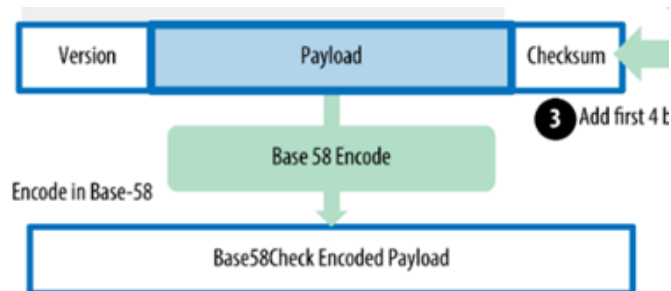
$$K_{child} = k_{parent} * G + h_{left} * G$$

$$K_{child} = K_{parent} + h_{left} * G$$

Gをh_left倍した点と、親公開鍵の点を楕円曲線上で足し合わせた点の子の公開鍵。

拡張鍵のエンコード

- 拡張鍵はBase58Checkでエンコードされる



- 異なる互換ウォレットの間でエクスポートインポートが可能

- プレフィックス

- 拡張秘密鍵：xprv
- 拡張公開鍵：xpub

拡張秘密鍵の例：

xprv9tyUQV64JT5qs3RSTJkXCWKMyUgoQp7F3hA1xzG6ZGu6u6Q9VM
NjGr67Lctvy5P8oyaYAL9CAWrUE9i6GoNMKUga5biW6Hx4twS2six3b9c

拡張公開鍵の例：

xpub67xpozcx8pe95XVuZLHXZeG6XWXHpGq6Qv5cmNfi7cS5mtjJ2tgype
QbBs2UAR6KECeeMVKZBPLrtJunSDMstweyLXhRgPxdp14sk9tJPW9

子公開鍵を導出できる利点

- 拡張公開鍵の強力な利点として、秘密鍵を使うことなく親の公開鍵から子の公開鍵を導出できる
- 信頼できないサーバーには拡張公開鍵のみを配置することで、際限なく公開鍵+アドレスの生成ができる
 - 秘密鍵が流出する危険性がない
 - そのアドレスへ送金されたお金は引き出せない
- 安全なサーバーに拡張秘密鍵を保持しておけば、対応した秘密鍵の生成ができ、上記のお金を引き出せる
 - どの鍵をどれに使うのか導出順(Path, 次ページ)は共有の必要あり.
- 応用例：Eコマースウェブサーバー
 - ウェブサーバーに拡張公開鍵のみを配置
 - 顧客のショッピングカートごとにアドレスを生成

HDウォレット鍵識別子(path)

- HDウォレットにある鍵はpathの命名規則を使って一意に指定される.
- ツリーの階層をスラッシュ (/) で区切って表現される.
- マスター秘密鍵から得られた秘密鍵は「m」から始まる
 - m/0は最初の子秘密鍵
- マスター公開鍵から得られた公開鍵は「M」から始まる
 - M/0 は最初の子公開鍵

鍵導出に「強化」という仕組みもありますがこの講義では割愛しています.

| HD path | Key described |
|-------------|--|
| m/0 | The first (0) child private key from the master private key (m) |
| m/0/0 | The first (0) child private key from the first child (m/0) |
| m/0'/0 | The first (0) normal child from the first <i>hardened</i> child (m/0') |
| m/1/0 | The first (0) child private key from the second child (m/1) |
| M/23/17/0/0 | The first (0) child public key from the first child (M/23/17/0) from the 18th child (M/23/17) from the 24th child (M/23) |

Table 6. HD wallet path examples

まとめ

- 公開鍵暗号方式
- 秘密鍵の作り方
 - 安全な乱数で生成
 - 256bitの数
- 公開鍵の秘密鍵からの求め方
 - 楕円曲線暗号を使用
- アドレスの公開鍵からの求め方
 - ダブルハッシュ + Base58Check
- 鍵のフォーマット
 - 公開鍵はそのままと長いので圧縮された公開鍵が生成できる.
- ウォレット
 - 秘密鍵の入れ物で, ビットコイン自体が入っているわけではない
 - 非決定性ウォレットと決定性ウォレットがある.
 - 決定性ウォレットでは, 秘密鍵なしで公開鍵の子を導出可能な仕組みも存在する.



- 本スライドの著作権は、東京大学ブロックチェーンイノベーション寄付講座に帰属しています。 自己の学習用途以外の使用、無断転載・改変等は禁止します。
- ただし、
 - Mastering Bitcoin 2nd edition
https://github.com/bitcoinbook/bitcoinbook/tree/second_edition_print3
 - ビットコインとブロックチェーン --- 暗号通貨を支える技術 （著：Andreas M. Antonopoulos
訳：今井崇也，鳩貝 淳一郎）
- を使用した部分の使用のみ，CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0/deed.ja> のライセンスを適用するものとします。