ビットコインをきっかけに学ぶ暗号技術入門

赤根大吾[†1]

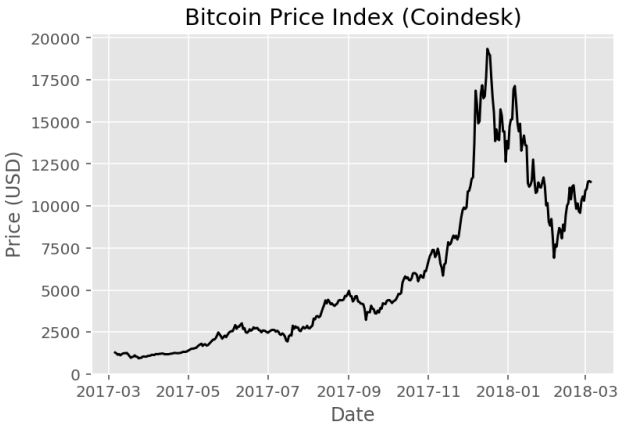
概要：ビットコインをはじめとする暗号通貨は、デジタル署名や一方向ハッシュ関数など、コンピュータを使った暗号技術によって支えられています。何かと話題の多い暗号通貨を題材にして、暗号技術を学んでみてはいかがでしょうか。基本的な近代暗号の用語から、ビットコインで使われている楕円曲線暗号、特に、秘密鍵から公開鍵を導くスカラー倍算について詳しく解説します。

キーワード：暗号通貨、暗号技術、ビットコイン、楕円曲線暗号、有限体、スカラー倍算、ECDSA、ECDH、公開鍵暗号、ブロックチェーン、ハッシュ関数

1. 暗号通貨の「暗号」とは

1.1 2017 年は暗号通貨元年？

2017 年はビットコインをはじめとする暗号通貨が大変話題になった一年でした。特に、日本円や米ドルなどの法定通貨（fiat currency）に対する価値の上昇はめざましく、年始には 1000 ドル程度だった 1BTC の価格は 12 月には一時 20000 ドル近くまで上昇しました。所有する暗号通貨の価格上昇により資産が億の単位になった「億り人」などのバズワードがニュースを賑わせ、改正資金決済法の施行や、暗号通貨の売買による収益を税法上どのように取り扱うかなど、法律面での整備も進みました。



1.2 「暗号通貨」か「仮想通貨」か

日本では「仮想通貨」という呼称が一般的ですが、英語圏では、「crypto currency＝暗号通貨」と呼ばれます。本稿でも「暗号通貨」を使いたいと思います。その方が、Suica など発行・管理を行う主体が存在する既存の「電子マネー」に対して、ビットコインをはじめとする非中央集権型の貨幣システムの新規性を区別しやすいと思うからです。

1.3 暗号通貨の技術を理解するメリット

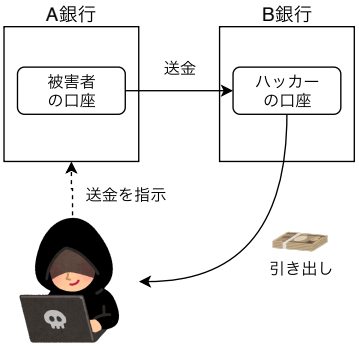
暗号通貨を所有する人が、そこで使われている技術を理解するのは当然のことでしょう。「Gox」やa、「セルフ Gox」bを防ぐため、正しい知識を身につけておく必要があります。また、暗号通貨に興味がない人でも、暗号通貨に使われている技術を学ぶ意味はあると考えます。暗号通貨は、コンピュータを用いた近代的な暗号技術に支えられており、その殆どは 2009 年の暗号通貨の登場以前から、毎日の暮らしの中で使われているものだからです。[1] 暗号通貨そのものが、今後どれだけ普及するかはわかりませんが、暗号通貨で使われている暗号技術そのものは、コンピュータにおける基礎知識なのです。

1.4 Coincheck 事件に学ぶブロックチェーン技術の概要

2018 年 1 月に Coincheck 社から 580 億円相当の NEM(XEM) cが流出するという事件がありました。このような暗号通貨の資金流出問題と、旧来のインターネットバンキングのハッキング被害との違いを見て見ましょう。インターネットバンキングのハッキングでは、大まかには以下の流れで被害が発生します。

1. ハッカーは、パスワードを不正に入手、またはマルウェアで振込の宛先を自分の口座に書き換えるなどして、自分の口座への送金指示する
2. 自分の口座から現金を引き出す

図 1: インターネットバンキングのハッキング被害



一方、暗号通貨では「口座」の代わりに「ウォレット＝財布」で資産を管理します。ウォレットには「アドレス」と、それに対応する「秘密鍵」が存在します。簡単に特徴を書

†1 (株)デジタルフィールド　東京都羽村市

a 取引所に預けた暗号通貨が消失することを、2014 年に発生した Mt.Gox 事件になぞらえ、「Gox する」と言われます。

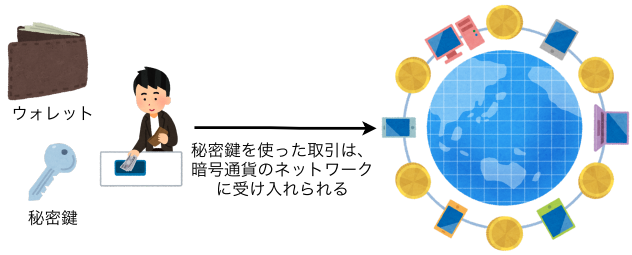
b 機器の故障や取り扱いの不備など、自分の過失によって暗号通貨を失うことを「セルフ Gox」と呼びます。

c NEM（ネム）の技術によって実現される代表的な暗号通貨とその単位を、XEM（ゼム）といいます。

き出すと以下のようになります。

- 「ウォレット」には「アドレス」と「秘密鍵」が存在する
- 「アドレス」と「秘密鍵」はペアである
- 「アドレス」は世界中に公開して良い
- 「秘密鍵」は絶対に公開してはいけない
- 「秘密鍵」を知っていれば、「アドレス」の残高を自由に使える

「秘密鍵」は「アドレス」を使った取引を通算した残高の正しい所有者の証明となり、「秘密鍵」を使った取引は世界に繋がった暗号通貨のネットワークに受け入れられます。



アーシュラ・K・ル＝グウィンの「ゲド戦記」には、対象を意のままに操ることができる「まことの名」が登場します。[2] これを喩えに使うのであれば、「通り名＝ハイタカ」が「アドレス」、知られてはいけない「まことの名＝ゲド」が「秘密鍵」に対応します。

Coincheck の場合、今回の事件で引き出された XEM のアドレスは、

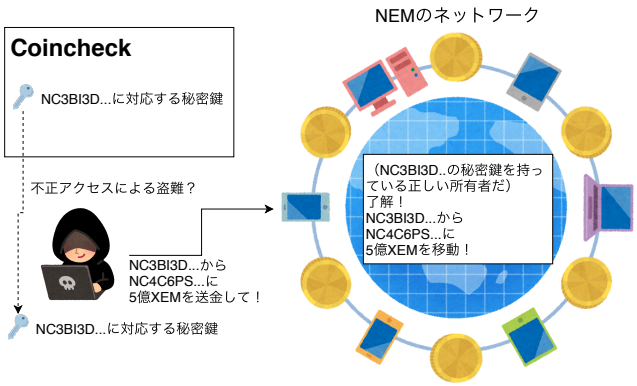
NC3BI3DNMR2PGEOOMP2NKXQ
GSAKMS7GYRKVA5CSZ

であるとわかっています。一方、秘密鍵は、私たちが知ることはできません。本来 Coincheck だけが知っているはずですが、流出がハッカーの仕業だとすると、ハッカーが何かしらの手段で秘密鍵を入手したことになります。

また、送金先のアドレスは

NC4C6PSUW5CLTDT5SXAGJDQJ
GZNESKFK5MCN77OG

でした。



ハッカーは「NC3BI3D...」の秘密鍵を用いて、「正しい」所有者として、「NC3BI3D...」から「NC4C6PS...」へ約 5 億 XEM の送金を行ったことになります。暗号通貨の世界では、秘密鍵の持ち主はアドレスの所有者の証明なので、この取引は正常に完了します。また、この取引は世界中に分散した NEM のネットワークを構成するコンピュータ群に記録されるため、取り消すことはできません。

さて、ここでこのような疑問を持つ方はいないでしょうか。

「暗号通貨というけれど、どこに暗号が使われているのだろう。暗号って、何かを暗号化して誰かから読めなくするように、秘密を守るためにあるのではないの？」

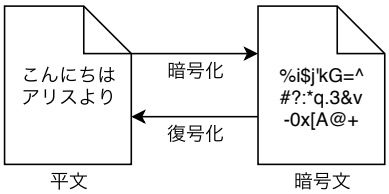
この疑問に答えるためには、少しコンピュータで使う暗号技術についての説明が必要です。

2. コンピュータで使う暗号技術

このセクションでは、コンピュータで使う暗号技術を代表して、「共通鍵暗号」「公開鍵暗号」「一方向ハッシュ関数」「デジタル署名」の 4 つを解説します。ただし、その技術をどのように実現しているか、深くは扱いません。興味のある方は「暗号技術入門」[3]「暗号がわかる本」[4]、「現代暗号入門」[5]などを参考にしてください。

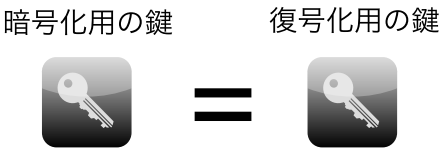
2.1 秘密を守る暗号技術

暗号といえば、秘密を守るためにあります。メールの添付ファイルを zip ファイルで圧縮する際に暗号化する、というのが一番身近な例でしょうか。ここで、用語を定義します。**暗号化**されていない素のドキュメントを**平文**(ひらぶん)、暗号化したドキュメントを**暗号文**と言います。また、暗号文を平文に戻すことを**復号化**と言います。



2.1.1 共通鍵暗号

共通鍵暗号は、「暗号化するための鍵」と、「復号化する（＝暗号を解く）ための鍵」が同じ暗号です。



zip ファイルを暗号化してメールに添付する場合、暗号化する時のパスワードと、復号化するためのパスワードは同じですが、このパスワードが「共通鍵」となります。

共通鍵暗号として、最もよく使われている方式として AES (Advanced Encyption Standard) があります。無線 LAN の設定などで目にしたことがある人も多いのではないでし

ようか。

図 2: 共通鍵暗号でメッセージを送信

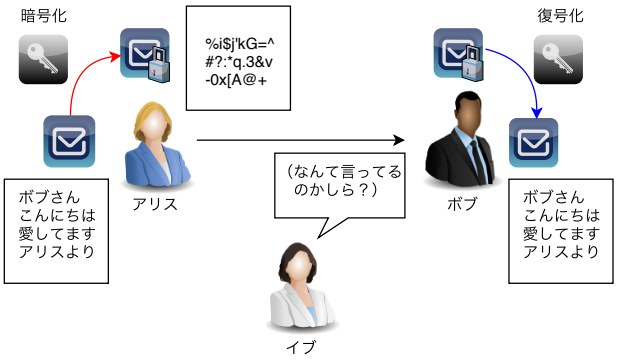


図 2 では、アリスが共通鍵暗号を用いてボブにメッセージを送信しています。イブが盗聴していますが、メッセージは暗号化されているのでイブには内容が理解できません。

2.1.2 鍵配送問題

これでアリスの秘密は守られました。めでたしめでたし…、とはなりません。共通鍵暗号には「鍵配送問題」が付きまといます。

図 3: 鍵配送問題

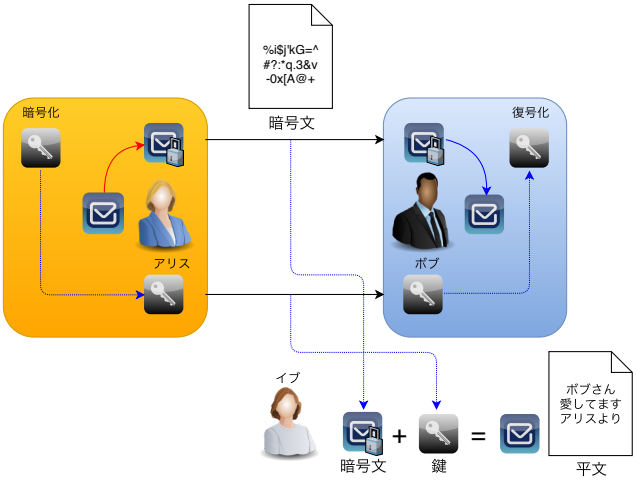


図 3 でボブは共通鍵を使って復号化していますが、アリスはボブにどのように鍵を渡せばよいでしょうか。イブが通信内容を盗聴できるのであれば、アリスからボブに鍵を送る時に、イブは鍵を入手できることになります。そして、暗号文もイブの手によって復号化されてしまいます。

日本のビジネスの現場では、パスワードで暗号化したメールを添付して、その前後のメールで鍵であるパスワードを送付するという慣習が見られますが、セキュリティ的にはあまり意味がないと言えるでしょう。d

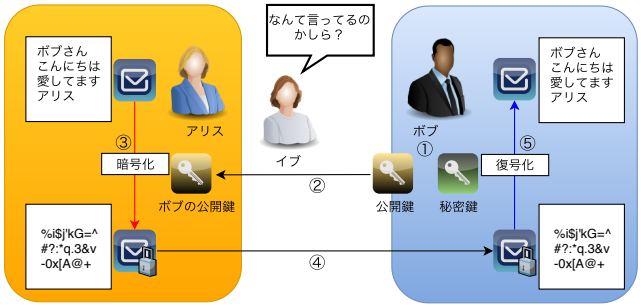
2.1.3 公開鍵暗号

鍵配送問題を解決する手段の一つとして、公開鍵暗号があります。先ほどの共通鍵暗号では一つの鍵を暗号化と復号化で使いましたが、公開鍵暗号では「公開鍵」と「秘密鍵」の 2 つが登場します。

- ペアとなる「公開鍵」と「秘密鍵」がある
- 「公開鍵」は持ち主以外に知られても良い
- 「秘密鍵」は持ち主以外に知られてはいけない
- 「公開鍵」を使って暗号化する
- 「秘密鍵」を使って復号化する

公開鍵暗号を使ってアリスがボブにメッセージを送る手順を示します。

図 4: 公開鍵暗号でメッセージを送信



1. ボブは自身の「公開鍵」と「秘密鍵」のペアを作成する
2. ボブはアリスに「公開鍵」を送付する
3. アリスは「ボブの公開鍵」を使ってメッセージ（平文）を暗号化する
4. アリスは暗号化したメッセージ（暗号文）をボブに送付する
5. ボブは受け取った暗号文を、「ボブの秘密鍵」で復号化し、平文のメッセージを得る

盗み見しているイブは「ボブの公開鍵」と「暗号化されたメッセージ」を入手できますが、メッセージを復号化するのに必要な「秘密鍵」はないので、メッセージの内容を知ることはできません。

公開鍵暗号方式として、RSA、ElGamal、楕円 ElGamal などが挙げられます。PGP（GPG）や S/MIME など、公開鍵暗号方式を使ってメールや添付ファイルを暗号化する手段も提供されています。

2.2 ごまかしを防ぐ暗号技術

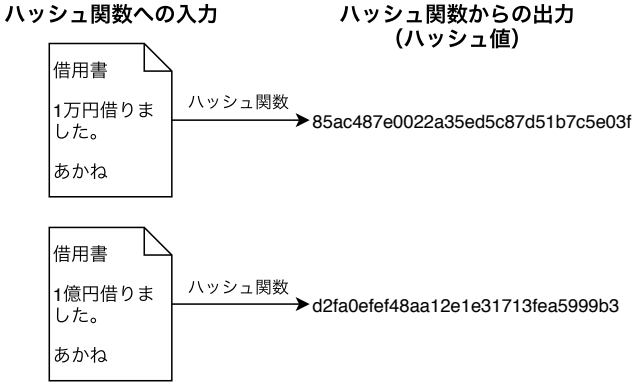
ここまでは秘密を守るための暗号技術を見てきましたが、少し違う目的のための暗号技術を紹介します。一言でいう

d　パスワードは暗号文とは別の手段で受け渡しを行うことが推奨されます。内閣官房情報セキュリティセンターの「庁舎内におけるクライアント PC 利用手順」では「保護に用いたパスワードについては、あらかじめ受信者と合意した文字列を用いるかあるいは、電子メールで送信せずに電話などの別手段を用いて伝達すること」とあります。

と、「ごまかし」を防ぐために暗号技術を使う方法です。

2.2.1 一方向ハッシュ関数

一方向ハッシュ関数は、入力された内容に応じて、決められたルールに従った出力を返す関数で、内容に変更があったかを判断するために使われます。ハッシュ関数の出力を「ハッシュ値」とも呼びます。次の例は、MD5 という一方向ハッシュ関数に、2 つのテキストファイルを入力し、その出力結果を比較したものです。



上の文書では「一万円」となっている箇所が、下の文書では「一億円」となっています。ただの 1 文字しか違いがないのに、ハッシュ値は大きく異なります。

この例では、内容が簡単だったため、目視でも違いがわかりました。これが 1 万文字を超えるような契約書が 2 部あり、その 2 部に違いがあるか、急いで確認しないといけない場合を想像すると、ありがたみがわかるでしょう。それぞれのデータを一方向ハッシュ関数に入力し、同じハッシュ値が出力されたら、内容は同じだと判断できます。

重要なことをまとめると：

- ハッシュ関数は入力が同じであれば同じハッシュ値を返す
- ほんの少しでも異なる入力であれば、（例外が無視できる確率で）異なるハッシュ値を返す

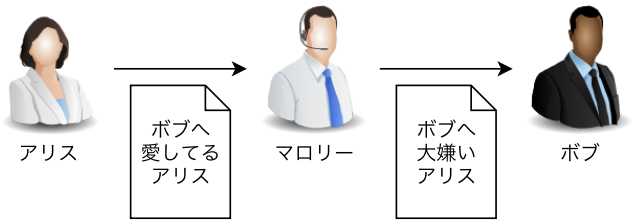
ハッシュ関数は、ビットコインのマイニング[e]にも使われています。簡単にいうと、「それまでの取引の履歴」＋「ランダムな文字」をハッシュ関数の入力として、あるルールを満たすハッシュ値が得られた場合にマイニングの成功となり、ブロックチェーンに新しいブロックが追加され、マイナー[f]に報酬が支払われます。

ビットコインでは SHA256 というハッシュ関数が使われています。

2.2.2 デジタル署名

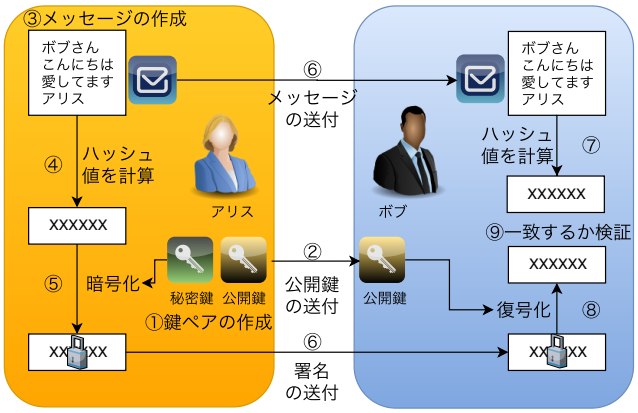
デジタル署名は、メッセージの内容が改ざんされていないと保証するための手段です。「公開鍵暗号技術」と「ハッシュ関数」を使います。

デジタル署名の必要性を認識するために、メッセージの内容を盗み見するだけだったイブではなく、メッセージの内容を改ざんできるマロリーに登場してもらいましょう。アリスはボブに「愛してる」というメッセージを送ったのに、マロリーが「大嫌い」と書き換えてしまいました。



デジタル署名を使えば、このような状況でも、受け取ったメッセージが、本当にアリスが書いたものかどうかを判断できます。デジタル署名にもいくつかの方法が存在しますが、ここでは RSA をデジタル署名に使った場合の手順を紹介します。先ほど、公開鍵暗号によるメッセージ送付の例では「公開鍵」で暗号化したメッセージを「秘密鍵」で復号化しました。RSA は「秘密鍵」で暗号化したものを「公開鍵」で復号化するという逆の使い方も可能で、その特性を署名に利用しています。

図 5: RSA を用いたデジタル署名



1. アリスは自身の「公開鍵」と「秘密鍵」を作成する
2. アリスは「公開鍵」をボブに送付する（この公開鍵の受け渡しは確実に行えたと仮定する）
3. アリスはボブ宛てのメッセージを作成する
4. 作成したメッセージのハッシュ値を計算する
5. 作成したハッシュ値を、アリスの秘密鍵で暗号化する。これを「デジタル署名」という
6. アリスは、「メッセージ」と「デジタル署名」をボブに送付する
7. ボブは受け取ったメッセージからハッシュ値を計算する
8. ボブは受け取ったデジタル署名を「アリスの公開鍵」で復号化する
9. ボブは「メッセージから計算したハッシュ値」と「デ

e ビットコインの取引を分散台帳（ブロックチェーン）に記録する作業です。マイニングに成功すると報酬として、一定量の新規発行されるビットコインと、送金者がトランザクションに含めた手数料を入手できます。

f マイニングを行う人のことです。

ジタル署名を復号化して得られたハッシュ値」とを比較する。一致した場合は、メッセージは改ざんされていない、一致しない場合は、メッセージが改ざんされた、と判断する

公開鍵暗号で暗号化したときと同様、秘密鍵は持ち主以外に知られてはいけません。アリスの公開鍵で復号化できるデジタル署名を作成できるのは、公開鍵に対応する秘密鍵を持ったアリスだけなので、9 の「ハッシュ値」の比較が一致するのであれば、アリスがメッセージを作成した時点から、メッセージは全く同じ内容であると言えます。

デジタル署名によって、秘密鍵の持ち主は、メッセージの内容に責任を持つことができます。逆にメッセージの内容が誰かによって改ざんされたものだと言い逃れはできません。

2.2.3 ハッシュ値だけでは、何故ダメか

メッセージを改ざんされたことはハッシュ値を比較すればわかるのだから、メッセージとハッシュ値を常にセットで送付すれば、ボブは改ざんに気づけるのでは？と考えるかもしれません。しかし、それではダメなのです。アリスやボブがハッシュ値を計算できるのと同様に、マロリーもハッシュ値を計算できるため、マロリーが「改ざんしたメッセージ」と「改ざんしたメッセージから計算したハッシュ値」を一緒に渡せば、ボブは改ざんに気づけません。

2.2.4 ECDSA による署名の作成と検証のデモ

ビットコインでも使われている ECDSA (Elliptic Curve Digital Signature Algorithm)というデジタル署名のアルゴリズムがあります。Elliptic Curve というのは「楕円曲線」という意味です。この ECDSA を使って、ボブに届いたメッセージが、アリスから送られたものかを確かめる手順を、Python のプログラムで示します。なお、本稿で使用するプログラムは Jupyter notebook 形式で github に公開しています。

https://github.com/dgakane/citp-report2017/

```
from ecdsa import SigningKey, VerifyingKey,¥
    SECP256k1, BadSignatureError
sk = SigningKey.generate(curve=SECP256k1)
```

アリスは作成した秘密鍵を使って、メッセージに署名をします。

```
msg_alice="""ボブへ
愛してる
アリス"""
vk = sk.get_verifying_key()
signature = sk.sign(msg_alice.encode())
```

ここで、sk が秘密鍵、vk が公開鍵、signature が署名となります。ボブのところに、アリス名義の、全く逆の内容のメッセージが 2 通届いたとします。このうちのどちらをアリスが書いたものか（署名したものか)、公開鍵（vk）と署名（signature）で検証します。

```
msg_mallory="""ボブへ
大嫌い
アリス"""

# マロリーのメッセージを検証
print("============")
print(msg_mallory)
print("============")
try:
    vk.verify(signature, msg_mallory.encode())
except BadSignatureError:
    print("↑アリスが署名したものではない")
else:
    print("↑アリスが署名したもの")

# アリスのメッセージを検証
print("============")
print(msg_alice)
print("============")
try:
    vk.verify(signature, msg_alice.encode())
except BadSignatureError:
    print("↑アリスが署名したものではない")
else:
    print("↑アリスが署名したもの")
```

このプログラムの実行結果は、以下のようになります。

```
============
ボブへ
大嫌い
アリス
============
↑アリスが署名したものではない
============
ボブへ
愛してる
アリス
============
↑アリスが署名したもの
```

どうやら、どちらがアリスが書いたメッセージなのか、判別できたようです。

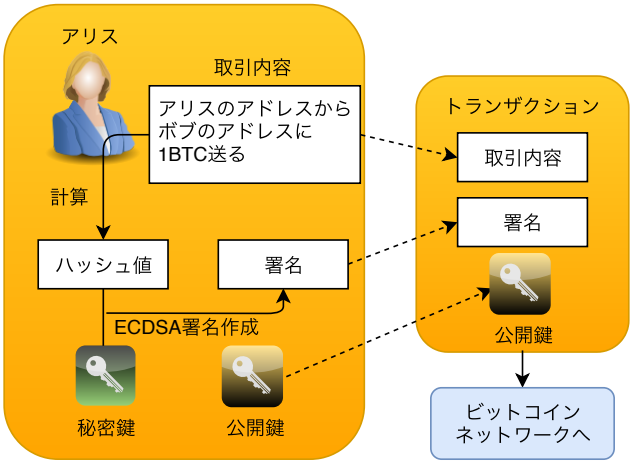
2.3　ビットコインのトランザクション（取引）の仕組み

アリスがボブにビットコインを使って送金するという例で、これまで紹介した暗号技術が暗号通貨にどのように使われているかを見てみましょう。

1. アリスは「アリスからボブに 1BTC を送金」という取引内容のハッシュ値を計算する

2. 「ハッシュ値」と「アリスの秘密鍵」を用いて、ECDSA で署名を作成
3. 「取引内容」＋「署名」＋「公開鍵」からなるトランザクションを作成。ビットコインネットワークに送信する

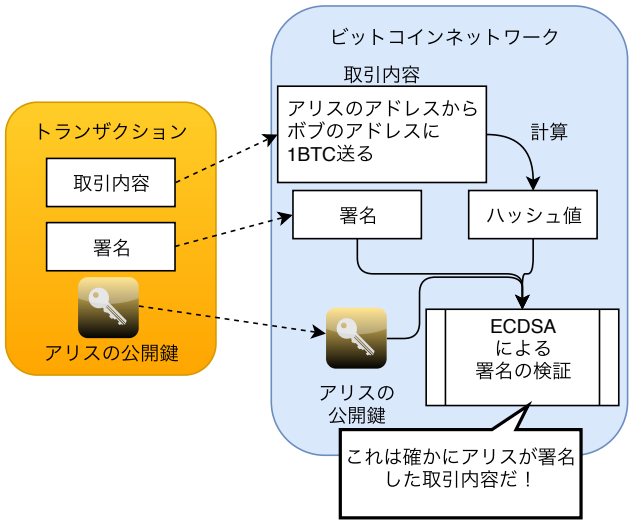
図 6: トランザクションの作成



アリスからのトランザクションを受け取ったビットコインネットワークでは、トランザクションが本当にアリスから送信されたものかを検証します。

1. トランザクションに含まれる「取引内容」から、「ハッシュ値」を計算する
2. 計算した「ハッシュ値」と、トランザクションに含まれる「署名」「アリスの公開鍵」を ECDSA に従い検証する

図 7: トランザクションの検証



このように検証が完了したトランザクションは、ビットコインネットワークを構成する他のノード（コンピュータ）に伝播し、アリスから送信された正当な取引としてブロックチェーンに織り込まれることになります。

ノードが検証している項目はもっと多岐に渡りますが、さらに詳しく知りたい方は「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」[6]を参照してください。

2.4 実は日常的に使っている暗号技術

ここまで見てきた暗号技術は、普段あまり意識しないのですが、実は日常的に使っているものです。下は、著者の Mac の Firefox で、Google にアクセスした際に、どのような暗号を使っているかを表示したダイアログです。

「TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256」という文字列が読み取れます。詳しい説明は省きますが、Web ブラウザと Web サーバとの通信の中で、いくつかの暗号技術を組み合わせて使っていることを示しています。

目的	方式
鍵交換に使っている方式は	ECDHE g
署名に使っている方式は	ECDSA
暗号化に使っている方式は	AES 128 GCM
メッセージ認証に使っている方式は	SHA256

共通鍵暗号で挙げた AES、ビットコインも使っている署名のアルゴリズム ECDSA やハッシュ関数 SHA256 が使われていることがわかります。

日常的に使われている暗号技術と、ビットコインなどの暗号通貨が繋がっていることが、おわかりいただけたでしょうか。

3. 楕円曲線暗号入門

さて、ここまで登場してきた「アドレス」「公開鍵」「秘密鍵」「ECDSA」などをより深く理解するためには、楕円曲線暗号について解説が必要です。正確には有限体上で定義された楕円曲線暗号といい、数学的な内容も出てきますが、プログラムによるスニペットとグラフによる可視化で、できるだけイメージしやすい説明に挑戦します。

g ECDHE に関しては、セクション 3 の最後にアルゴリズムを紹介します。

3.1 ビットコインで使われる楕円曲線

3.1.1 「アドレス」「公開鍵」「秘密鍵」の関係

「アドレス」「公開鍵」「秘密鍵」が具体的にどのようなものか例を見てみましょう。

ビットコインの**アドレス**は、以下のように 1 で始まる文字列です。

12noHBPojYCYtEPaXGeLAY8ExyXJRC7uq8

公開鍵は（あとで説明しますが）平面上の点なので、x 座標と y 座標を表す 2 つの値で構成されます。[h]

(x, y)=

(59657349624863415277140746483024327429760811273444096062005870676035102190933,

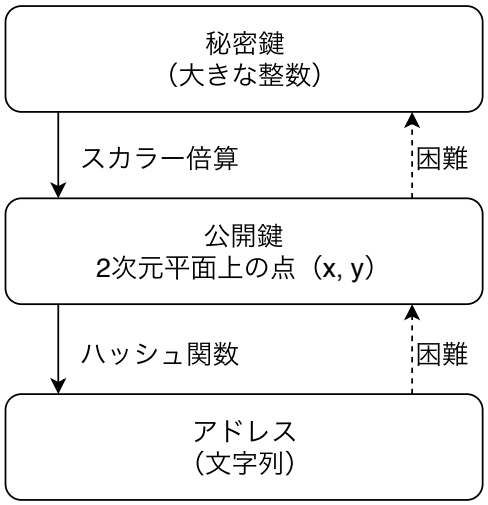
57361002836192819049676076242901405112088254166527378116283440525442954582659)

秘密鍵は一つの値で、大きな整数です。

57625604009098861386593689532106037116306023484397133537612206883757739966481

公開鍵は、秘密鍵から「スカラー倍算」という計算で作られます。逆に、公開鍵から秘密鍵を求めることは困難です。

アドレスは、公開鍵から「ハッシュ関数」で作られます。アドレスから公開鍵を求めることは困難です。図にまとめます。



ハッシュ関数は、先ほどのセクションで紹介したものです。そして、新たに「スカラー倍算」というものが出てきました。**このセクションは、「スカラー倍算」の説明をゴールとします**。

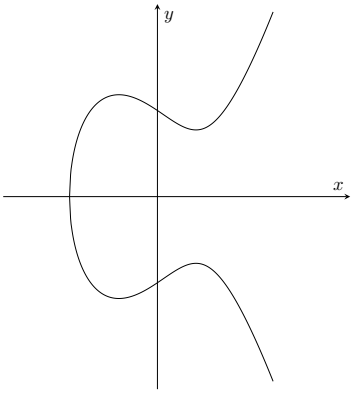
3.2 実数平面上の楕円曲線

楕円曲線は以下のような方程式で与えられます

$$y^2 = x^3 + ax + b$$

グラフで描画すると以下のようになります。

図 8: 楕円曲線



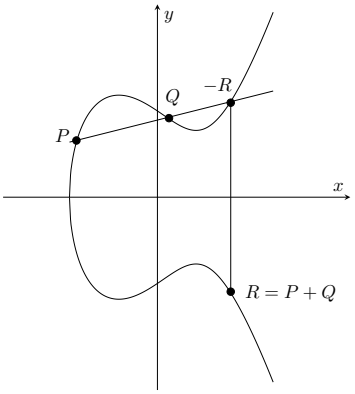
3.2.1 楕円曲線上の点の加法（実数バージョン）

この楕円曲線上で、以下のように点と点の加法（足し算）を定義します。

楕円曲線E上の 2 点の和$P+Q=R$を以下のように定義する:

- PとQを通る直線が、再びEと交差する点を$-R$
- $-R$と x 軸に関して対称な点をRとする

図 9: 楕円曲線上の加法（$P \neq Q$の場合）



この定義に従い、具体的な座標の計算も可能です。

$P=(x_P, y_P)$、$Q=(x_Q, y_Q)$、$R=(x_R, y_R)$と書くとき、

$$x_R = \lambda^2 - x_P - x_Q$$

$$y_R = \lambda(x_P - x_R) - y_P$$

ここでλは、点P, Qを通る直線の傾きで、

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

とする。

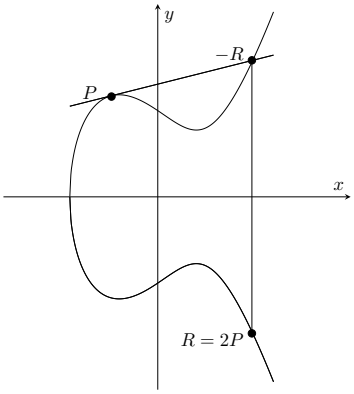
PとQがどんどん近づいて、一つの点となった場合($P=Q$)は、楕円曲線上の点の 2 倍と考えることができます。

楕円曲線E上の点Pの 2 倍$P+P=2P=R$を以下のように定義する:

- EのPにおける接線が、再びEと交差する点を$-R$
- $-R$と x 軸に関して対称な点をRとする

h ビットコインの公開鍵は、02 または 03 から始まる「圧縮された形式」の文字列で表現されることもあります。どちらにしても、平面上の点であることに変わりはありません。

図 10: 楕円曲線上の加法（$P = Q$の場合）



この場合も、以下のように座標を求めることができます

$P = (x_P, y_P)$、$R = (x_R, y_R)$と書くとき、

$$x_R = \lambda^2 - 2x_P$$

$$y_R = \lambda(x_P - x_R) - y_P$$

ここでλは、点P, Qを通る直線の傾きで、

$$\lambda = \frac{3x^2 + a}{2y}$$

とする。

3.3 コンピュータで楕円曲線を扱う

ここまで楕円曲線を我々にとって普通の数＝実数上で考えてきました。グラフに書いたり、慣れ親しんだ方法で座標を計算したりできるのでイメージしやすいのですが、実際にコンピュータで楕円曲線を扱う場合と大きな乖離があります。実数には、無限に大きな数が存在したり、どれだけ小さな数と数との間にも無限の数が存在するといった特徴があり、有限の桁数（ビット）しか持たないコンピュータには扱いづらいのです。コンピュータでは実数上ではなく、有限体(ゆうげんたい)上で定義された楕円曲線を使います。

3.3.1 体（Field）とは？

体(たい)とは、四則演算（足し算、引き算、かけ算、割り算）が行える集合です。その集合の要素を使って四則演算を行えば、演算の結果もまたその集合に含まれます。この性質を「四則演算で閉じている」と言います。例えば、実数の集合ℝは体となります。「実数」同士で四則演算を行った結果は必ず「実数」になるからです。

$$1 + 2 = 3 \cdots 3 \text{ は実数}$$

$$1 - 2 = -1 \cdots -1 \text{ は実数}$$

$$1 \times 2 = 2 \cdots 2 \text{ は実数}$$

$$1 \div 2 = 0.5 \cdots 0.5 \text{ は実数}$$

「四則演算が行えるなんて当たり前では？」と思うかもしれません。「四則演算が行えない」例としては、整数全体の集合ℤでは、その要素同士での割り算の結果が整数にならないことがあります。

$$1 \div 2 = 0.5 \cdots 0.5 \text{ は整数ではない}$$

そのため、整数全体の集合ℤは体ではありません。

3.3.2 有限体（Finite Field）とは？

有限体は、有限の要素からなり、四則演算で閉じている集合です。要素の数が無限に存在するℝなどと異なり、コンピュータでの演算が容易です。口で言うのは簡単ですが、本当にそのような集合は存在するのでしょうか。pを素数として、0から$p-1$までの整数で構成される集合$\mathbb{F}_p = \{0, 1, \cdots, p-1\}$に、私たちが普段使っているものとは、少し違った四則演算を定義して、有限体とすることができます。pが小さい場合で例を作ってみます。$p = 5$の場合、つまり$\mathbb{F}_5$での四則演算を見ていきます。

3.3.3 有限体$\mathbb{F}_5$の足し算

基本的には普通の足し算と同じですが、計算の結果 5 を超えた場合は、その結果を 5 で割った余りを答えとします。$a + b$の結果を以下にまとめます。

a＼b	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Python の関数として書くと、このようになります。

```
def add(a, b): # F5 での足し算
    return (a + b) % 5
```

引き算は、このコード例で b がマイナスになった場合ですので、割愛します。

3.3.4 有限体$\mathbb{F}_5$のかけ算

かけ算も同様に、「普通のかけ算」の結果を 5 で割った余りとなります。

a＼b	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

```
def multiply(a, b): # F5 でのかけ算
    return (a * b) % 5
```

3.3.5 有限体$\mathbb{F}_5$の割り算

割り算に関しては、少し工夫が必要です。$1 \div 2$など、通常の割り算では結果が整数にならず、そのままでは計算結果が$\mathbb{F}_5$に含まれません。ここで少し割り算の意味を考えてみると、「割り算」は「かけて 1 になる数＝逆数」をかけていると言えます。例えば、「普通の」かけ算や割り算で考える

と 2 の逆数は$\frac{1}{2}$となります。$\mathbb{F}_5$の世界で「かけて 1 になる組み合わせ」をかけ算の表の中ではグレーの網かけで表しました。これによると、2 の逆数は 3 となります。なので、$\mathbb{F}_5$の世界では、$1 \div 2 = 1 \times 3 = 3$となります。かけ算の表で、かけたら 1 になる数を探しながら、割り算の表を作れます。

b

a ÷	0	1	2	3	4
0	-	0	0	0	0
1	-	1	3	2	4
2	-	2	1	4	3
3	-	3	4	1	2
4	-	4	2	3	1

しかし、いちいちかけ算の表を確認しないと割り算ができないのでしょうか。$p = 5$の場合なので、なんとかできますが、pの値が大きくなった場合には大変そうです。ここは数学の力を借りることにします。「ユークリッドの互除法」や「フェルマーの小定理」を利用して、「かけて 1 になる数＝逆数」を求めることができます。[7]

フェルマーの小定理より
任意の素数pと、0 ではない$n \in \mathbb{Z}_p$について

$$n^{p-1} = 1 \bmod p$$

が成り立つ。ここから

$$n^{p-2} = n^{-1} \bmod p$$

と変形でき、nの逆数「n^{-1}」は、「n^{p-2}のpの剰余」と等しい

次のコードでは、フェルマーの小定理を使って逆数を求めています。Python でべき乗を行う pow 関数は第 3 引数を受け取ることができ、pow(x, y, z)は x の y 乗に対する z の剰余を返します。

```
def inv(n, p): # 剰余 p の世界で逆数を求める関数
    return pow(n, p-2, p) # フェルマーの小定理より
def div(a, b): # F5 での割り算
    return (a * inv(b, 5)) % 5
```

このように、$\mathbb{F}_5$に（少し変わった）四則演算を定義することができました。一般に、pが素数の場合は、$\mathbb{F}_p$は有限体となり、素体とも言われます。

3.3.6 有限体$\mathbb{F}_p$上の楕円曲線

有限体$\mathbb{F}_p$での楕円曲線を以下のように定義します。

$y^2 \equiv x^3 + ax + b \pmod p$を満たす 2 次元平面$\mathbb{F}_p \times \mathbb{F}_p$上の点$(x, y)$

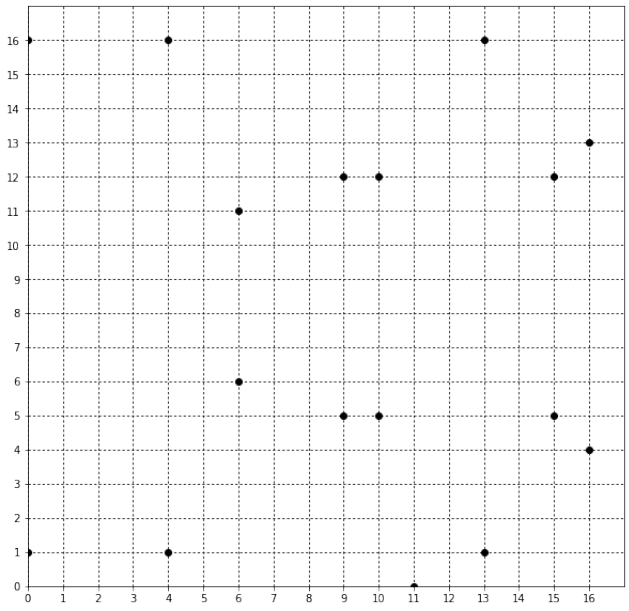
$s \equiv t \pmod p$と言うのは、「sをpで割った余りと、tをpで割った余りは等しい」という意味です。$0 \equiv t - s \pmod p$と変形すれば、「$t - s$をpで割った余りは0である」と読み替えられます。なので、有限体$\mathbb{F}_p$での楕円曲線は、「$x^3 + ax + b - y^2$をpで割った余りが0となるような(x, y)の集合」となります。式や文章だと分かりにくいですが、コードとグラフで確認すると意外と簡単です。$\mathbb{F}_{17}$上での$y^2 = x^3 + x + 1$をグラフにプロットする Python のプログラムを以下に示します。

```
def plot_ec(a, b, p):
    import matplotlib.pyplot as plt
    xlist = []
    ylist = []
    for x in range(p):
        for y in range(p):
            if((x**3 + a * x + b - y**2) % p == 0):
            # 方程式を満たす x,y の組をリストに格納
                xlist.append(x)
                ylist.append(y)
    #以下は表示のため
    plt.figure(figsize=(10,10))
    plt.axis([0,p,0,p])
    if(p < 55):
        point_style = 'o'
        plt.grid(which='major',linestyle=':'
                 , color="black")
        plt.yticks( [i for i in range(p)] )
        plt.xticks( [i for i in range(p)] )
    else:
        point_style = '.'
    plt.plot(xlist, ylist, point_style , color="black")
    plt.show()

plot_ec(1, 1, 17)
```

実行結果はこうなります。

図 11: $\mathbb{F}_{17}$上での楕円曲線$y^2 = x^3 + x + 1$

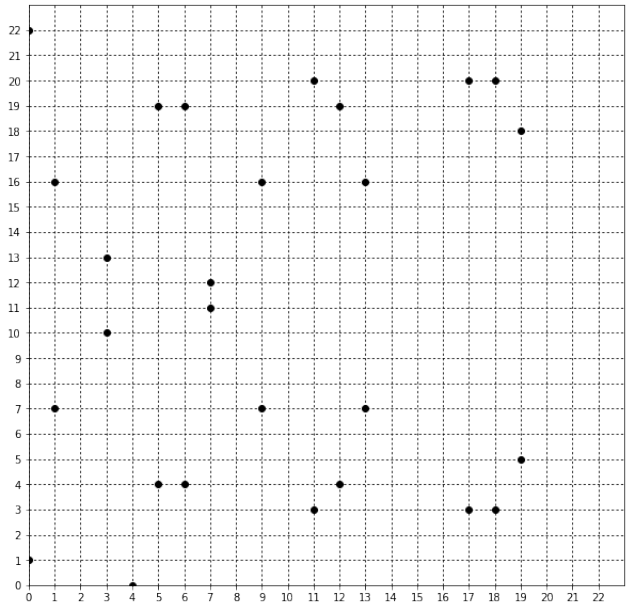


plot_ec の第 3 引数を変更することで、異なるpでの楕円曲

線をみることができます。pをもう少し大きくして見ましょう。$p=23$の場合：

```
plot_ec(1, 1, 23)
```

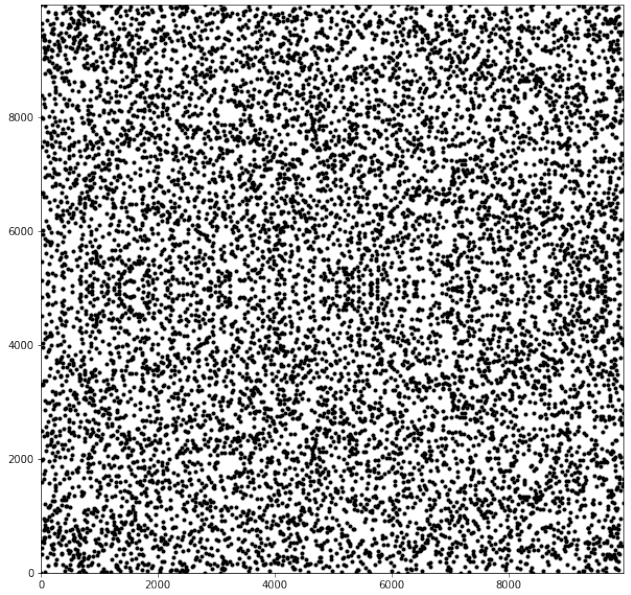
図 12: $\mathbb{F}_{23}$上での楕円曲線$y^2=x^3+x+1$



$p=9973$の場合：

```
plot_ec(1, 1, 9973)
```

図 13: $\mathbb{F}_{9973}$上での楕円曲線$y^2=x^3+x+1$



いかがでしょうか。楕円曲線と言う割に、楕円でも曲線でもありません。よく観察すると以下のような特徴があります。

- とびとびの値を取っている
- $y=0$上の点を除いて、$y=\frac{p}{2}$を挟んで線対称に点が分布している
- 対称な点同士は奇数と偶数に分かれている

3.3.7 楕円曲線の点の加法（有限体$\mathbb{F}_p$バージョン）

実数の時と同様に、楕円曲線上の点の加法を定義します。

$\mathbb{F}_p$上の楕円曲線$y^2=x^3+ax+b$上の点P,Qの和$R=P+Q$を以下のように定める:

$P=(x_P,y_P)$、$Q=(x_Q,y_Q)$、$R=(x_R,y_R)$と書くとき、

$$x_R=\lambda^2-x_P-x_Q$$

$$y_R=\lambda(x_P-x_R)-y_P$$

ここでλは、

$$\begin{cases} P\neq Q \text{ の時: } \lambda=\dfrac{y_Q-y_P}{x_Q-x_P} \\ P=Q\text{の時: } \lambda=\dfrac{3x_P^2+a}{2y} \end{cases}$$

とする。[i]

式自体は実数の時と同じですが、ここでの四則演算は全て$\mathbb{F}_p$における四則演算だということに注意が必要です。分数＝割り算は、普通の割り算ではなく、有限体における逆数のかけ算です。

3.3.8 具体的にやってみる

ここで具体的に、楕円曲線上の点の足し算を行なって見ます。$(x,y)=(0,1)$は$y^2=x^3+x+1$上の点です。この点をGと名付けます。$\mathbb{F}_{23}$上で$(0,1)+(0,1)=G+G=2G$を計算します。

$(0,1)+(0,1)=G+G=2G$の計算：

まずλを計算する。

同じ点同士の足し算なので、「$P=Q$の時」に該当する。

$$\lambda=\frac{3x_P^2+a}{2y}=\frac{3\cdot 0^2+1}{2\cdot 1}=\frac{1}{2}=2^{-1}=2^{23-2}\bmod 23=12$$

（逆数の計算にはフェルマーの小定理を利用した。）

$$x_R=\lambda^2-x_P-x_Q=12^2-0-0$$

23を超える計算結果は、23で割った余りなので、

$$=144\bmod 23=6$$

$$y_R=\lambda(x_P-x_R)-y_P=12(0-6)-1$$

$$=12\cdot 17-1$$

$$=203\,mod\,23=19$$

$$\therefore 2G=(0,1)+(0,1)=(6,19)$$

これらの計算を ec_double という関数にまとめてプログラムにすると以下のようになります。

```
# F_p 上の y^2=x^3+ax+b での p, a, b
p, a, b = 23, 1, 1
G=(0,1)
def inv(n, p):
    return pow(n, p-2, p)
def ec_double(A):
    l = (((3 * A[0] **2) + a) * inv(2*A[1], p)) % p
    x = (l ** 2 - A[0] - A[0]) % p
    y = (l * (A[0] - x) - A[1]) % p
```

i 不完全な定義です。無限遠点に関する定義を省略しています。

```
    return x, y
G2=ec_double(G)
print(G2) # (6, 19)
```

このようにして計算してできた$2G = (6,19)$にもう一度Gを足して$3G$を作って見ましょう。今度は異なる点の足し算なので、ec_add という関数を作ります。

```
def ec_add(A, B):
    l = ((B[1] - A[1]) * inv(B[0] - A[0], p)) % p
    x = (l ** 2 - B[0] - A[0]) % p
    y = (l * (A[0] - x) - A[1]) % p
    return x, y
G3 = ec_add(G2, G)
print(G3) # (3, 13)
```

できた点に繰り返しGを足していくと、次から次へと楕円曲線上の点が現れます。自作の関数でも良いのですが、Python の ecdsa ライブラリを使って同じことをしてみます。すでに自作関数を使って計算した$2G$と$3G$の検算にもなります。

```
from ecdsa.ellipticcurve import CurveFp, Point
# F_p 上の y^2=x^3+ax+b の意味での p, a, b
p, a, b = 23, 1, 1
c = CurveFp(p,a,b)
# G を c 上の点(0,1)とする
G = Point(c, 0, 1)
current = G
for i in range(1,28):
    print("{}G:¥t{}".format(i,current))
    current = current + G
```

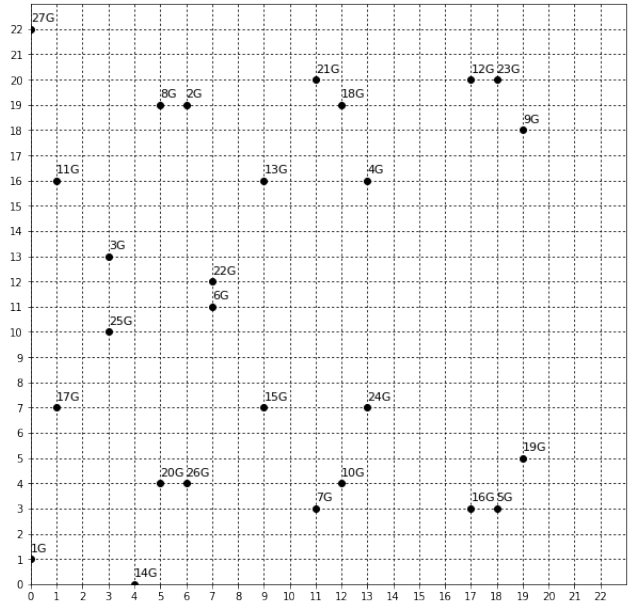
このプログラムの実行結果は以下のようになります。

表 1: Gを足し合わせた結果

```
1G:     (0,1)
2G:     (6,19)
3G:     (3,13)
    (略)
23G:    (18,20)
24G:    (13,7)
25G:    (3,10)
26G:    (6,4)
27G:    (0,22)
```

これを座標にプロットすると図 14 のようになります。図 12 と同じグラフが得られました。

図 14: $\mathbb{F}_{23}$上での楕円曲線$y^2 = x^3 + x + 1$



3.3.9 楕円曲線上の点のスカラー倍

このように点Gをd回足して得られる点を**スカラー倍点**といい、スカラー倍点を求める計算を**スカラー倍算**と言います。

スカラー倍算

$$dG = \underbrace{G + \cdots + G}_{d個}$$

dGの座標を具体的に求めるための方法として、定義通りに、GからスタートしてGを$d-1$回足していく素朴な方法があります。

スカラー倍算（素朴な方法）

$$dG = \underbrace{G + \cdots + G}_{d個}$$

$$= G + \underbrace{G + \cdots + G}_{d-1\ 個}$$

$$= 2G + \underbrace{G + \cdots + G}_{d-2\ 個}$$

$$\vdots$$

$$= (d-1)G + G$$

このように逐次計算を行うのではなく、もっと直接的に求めることはできないのでしょうか。

3.3.10 スカラー倍算の高速化（バイナリ法）

直接的、とは言えないかもしれませんが、もっと効率よく計算するアルゴリズムとして**バイナリ法**を紹介します。以下は、スカラー倍算を用いて楕円曲線上の点Pをd倍する手順です。

スカラー倍算（バイナリ法）

- スカラーdの２進数表現を$(1, d_{n-2}, \cdots, d_1, d_0)$とする
  例: d = 129の場合(1,0,0,0,0,0,0,1)
- 開始する点をPとする

1. $Q \leftarrow P$
2. $i = n-2, n-3, \cdots, 1, 0$に対して以下を行う
   2-1: $Q \leftarrow 2 \times Q$
   2-2: $d_i = 1$ならば$Q \leftarrow Q + P$
3. Qを返す

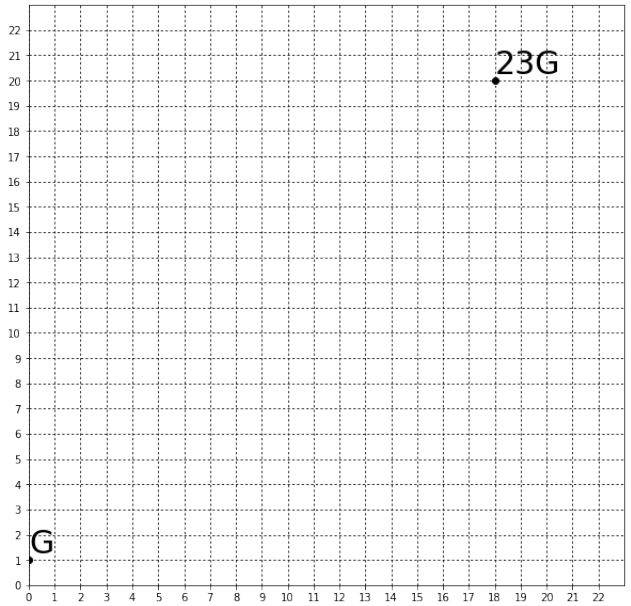
これはプログラムで見た方がわかりやすいかもしれません。ここで使っている関数 ec_double と ec_add は、3.3.8 で作成したものです。

```
def binary_method(P, d):
    Q = P
    bin_str = format(d, 'b') # d の二進数表現文字列
    for bit in bin_str[1:]:
        Q = ec_double(Q)
        if bit == "1": Q = ec_add(Q, P)
    return Q
```

この関数を使って$23G$を計算してみましょう。表 1 ならびに図 14 から、$23G$が(18,20)であることがわかっています。binary_method を使っても同じ結果が得られます。

```
G23 = binary_method(G,23)
print(G23) # (18, 20)
```

図 15: $23G$をいきなり求める



3.3.11 スカラー倍算に必要な計算量

スカラー倍算の方法として、順番に点を足し合わせていく「素朴な方法」と、効率的な「バイナリ法」を紹介しました。しかし、バイナリ法は本当に効率的なのでしょうか？この 2 つを比べると、どのくらい計算量に違いがあるのでしょうか。計算量を式として表してみます。

「素朴な方法」の計算量：

Gから$2G$を計算…ec_double を使用
$2G$から$3G$を計算…ec_add を使用
$3G$から$4G$を計算…ec_add を使用
：
$(d-1)G$からdGを計算…ec_add を使用

つまり、dG まで至るまでに、
ec_double を 1 回、ec_add を$(d-2)$回使う。
ec_double と ec_add の計算量が同程度とすると、

$$\textit{素朴な方法の計算量} = d - 1$$

バイナリ法は、dを二進数で表現した桁ごとに手順 2-1 と手順 2-2 を行なっています。また、ある数nを二進数で表した場合の桁数は約$\log_2 n$になることを利用します。

バイナリ法の計算量：

手順 2-1 の ec_double は必ず実行される

手順 2-2 の ec_add は$\frac{1}{2}$の確率で実行される

ec_double と ec_add の計算量が同程度とすると、
$\textit{バイナリ法の計算量} = (1 + 0.5) \times$「$d$の二進数での桁数」と書けます。ここで

$$d\text{の二進数での桁数} \cong \log_2 d$$

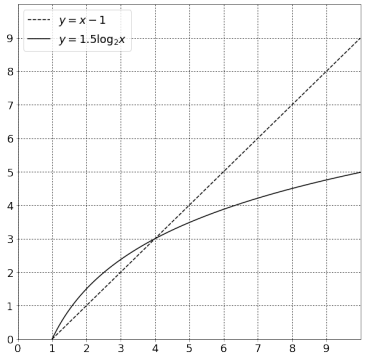
なので、

$$\textit{バイナリ法の計算量} \cong 1.5 \log_2 d$$

※≅は「おおよそ等しい」

$d-1$と$1.5\log_2 d$では、どちらが大きいのでしょうか？グラフを書いて確認します。最初のうちこそいい勝負に見えますが…

図 16: 計算量の比較（10 まで）



dが大きくなるに従って、徐々に差が大きくなって来ます。

図 17: 計算量の比較（100 まで）

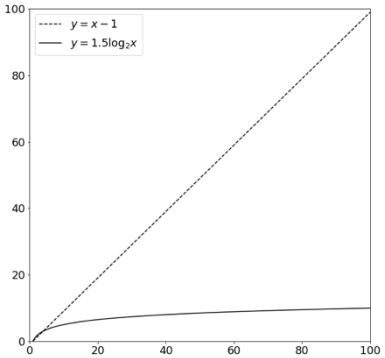
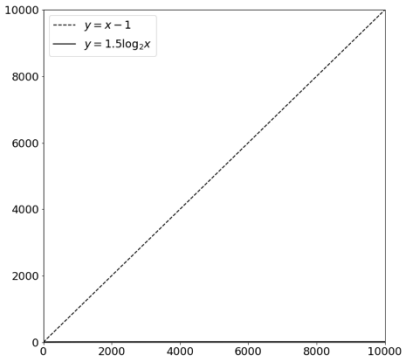


図 18: 計算量の比較（10000 まで）



「素朴な方法」での計算量は一定の割合で増加し続けるのに比べ、「バイナリ法」では増加の仕方が減っていき、dが十分に大きな数になると「素朴な方法」と比べて無視できる計算量であることがわかります。

3.4 楕円曲線暗号における「鍵」

実は、楕円曲線暗号における「秘密鍵」と「公開鍵」は、それぞれdと、dGになります。

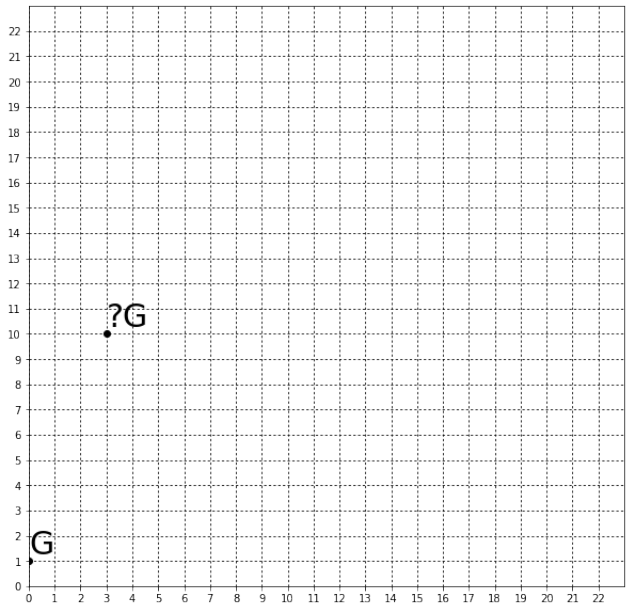
秘密鍵	公開鍵
d	dG

以下のような情報をあなたが知っていたとします。

知っている情報	例
有限体$\mathbb{F}_p$の標数p	$p = 23$
楕円曲線の方程式	$y^2 = x^3 + x + 1$
ベースポイントG	$G = (0,1)$
公開鍵dG	$dG = (3,10)$

さて、ここで$dG = (3,10)$はGを何回か（d回）足し合わせたものであることはわかっています。このとき、dの値がわかりますか？少し考えてみてください。

図 19: $(3,10)$はGを何倍したもの？



実は、私たちは$\mathbb{F}_{23}$上の$y^2 = x^3 + x + 1$の点を、図 14 で全て計算済みです。そのため、図を見比べることでdが 25 であることを知ることができます。ですが、それは点を一つずつ足していった「素朴な方法」で求めたものでした。$?G = (3,10)$となるような?を知るためには、Gを足してできる点を一つ一つ、しらみつぶしに試していくしかないのです。このように、

- 秘密鍵（$= d$）を知っていれば、公開鍵（$= dG$）を計算するのは（計算量的に）容易
- 一方、秘密鍵を知らない攻撃者が公開鍵から秘密鍵を求めようとしても、総当たり、この場合は順番に試していくしか方法がない

という性質を、楕円曲線暗号は利用しています。スカラー倍算によって、秘密鍵から公開鍵を求めることはできるのに、公開鍵から秘密鍵を求めることはできないというのは、ここで挙げた計算量の違いによるものです。

3.5 楕円曲線上の点の位数

ところで、楕円曲線上の点Pを足し合わせていくと、いつかPとx座標が等しい点が出現します。これを$-P$とし、$-P$にさらにPを足した点を無限遠点$\mathcal{O}$と定義します。すると、楕円曲線上の点の足し算は以下の様に巡回します。

$$\underbrace{P \to 2P \to 3P \to \cdots \to -P \to \mathcal{O}}_{r\text{個}} \to P \to \cdots$$

この様に$rP = \mathcal{O}$となるような最小の自然数を「Pの点位数」と言います。楕円曲線暗号で使う場合、点位数rが素数となるようなベースポイントGを選びます。

ここまで見てきた、「有限体での四則演算」や「スカラー倍算」「点位数」がわかれば、楕円曲線を使った暗号方式（楕円曲線 ElGamal）や、署名（ECDSA）のアルゴリズムを自分で計算してみることが可能になります。これらのアルゴリズムについて、詳しくは「楕円曲線暗号入門」[8]や「暗号理論と楕円曲線」[9]などをご参照ください。

3.6 ビットコインに使われる楕円曲線のパラメータ

ビットコインは米国国立標準技術研究所（NIST）が策定した secp256k1 という楕円曲線と定数を使用しています。以下の情報は、誰でも知ることが可能な公開情報です。

有限体$\mathbb{F}_p$の標数p:

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

楕円曲線の方程式:

$$y^2 = x^3 + 7$$

スカラー倍のベースポイント:

$$G = (G_x, G_y)$$

G_x =55066263022277343669578718895168534326250603453777594175500187360389116729240

G_y =32670510020758816978083085130507043184471273380659243275938904335757337482424

ベースポイントGの点位数r:

r =1157920892373161954235709850086879078528375642790749043826051631415181614943 37

秘密鍵は、1 からrの範囲で、ランダムに選ばれます。このrは、10^{77}くらいの値です。「観測可能な宇宙に存在する素粒子の数」が、10^{80}くらいと言われていますので、とても大きな選択肢の中から秘密鍵を選んでいるわけです。公開鍵dGに対応する秘密鍵dを、しらみつぶしに探すことがいかに無謀か、おわかりいただけるでしょうか。

3.7 楕円曲線 Diffie-Hellman 鍵共有

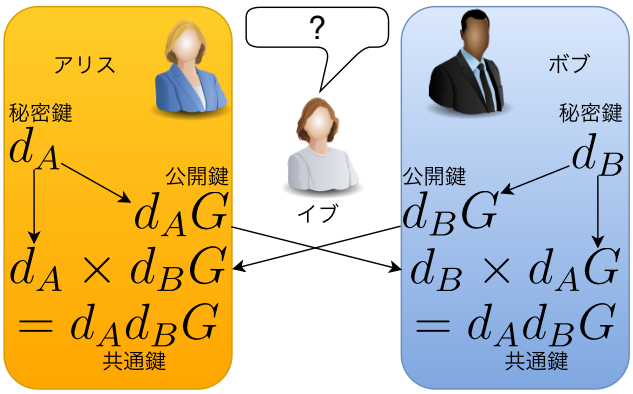
最後に、スカラー倍算を使ったアルゴリズムの例として、ここでは楕円曲線を使った鍵共有の方式、ECDH (Elliptic-curve Diffie–Hellman)について紹介します。

「2.1.2 鍵配送問題」では、盗聴者がいる場合、共通鍵暗号を使おうにも、鍵が盗聴者に知られてしまう問題を取り上げ、その解決策の一つとして公開鍵暗号を紹介しました。ここで、もう一つの解決策として、盗聴者のいるところでも、楕円曲線を使って安全に鍵を受け渡しできる方法を紹介します。

楕円曲線 Diffie-Hellman (ECDH)
有限体$\mathbb{F}_p$、楕円曲線の方程式、ベースポイントG、Gの点位数rはアリス、ボブ、イブに既知とする。

1. アリスは$1 < d_A < r$となるd_Aを選び、秘密鍵とする。またd_AGを公開鍵とする。
2. ボブは$1 < d_B < r$となるd_Bを選び、秘密鍵とする。またd_BGを公開鍵とする。
3. アリスとボブはお互いの公開鍵を交換する。公開鍵は盗聴されても構わない。
4. アリスはボブから入手したd_BGと自分の秘密鍵d_Aで、d_Ad_BGを計算する。
5. ボブはアリスから入手したd_AGと自分の秘密鍵d_Bで、d_Ad_BGを計算する。
6. d_Ad_BGをお互いの共通鍵として使う。[j]

図 20: 楕円曲線 Diffie-Hellman



イブは公開鍵であるd_AGやd_BGを入手することはできますが、公開鍵から秘密鍵であるd_Aやd_Bを知ることはできません。一方、アリスとボブは自身の秘密鍵と相手の公開鍵からd_Ad_BGを計算することが可能です。このd_Ad_BGを共通鍵として、安心して使うことができます。

4. 社会とつながる暗号技術

2018 年 3 月 8 日の参議院予算委員会、浅田均議員の質問の中で、公文書の管理にハッシュ関数やブロックチェーンを活用してはどうかという趣旨の発言がありました。

国会は約 1 年にわたり「森友学園問題」に揺れており、国有地の売却についての「決裁文書」に、「国会に提出されたもの」と「詳細が記された売却契約当時のもの」が存在する—つまり決裁文書の改ざんの疑惑で、佐川国税庁長官の辞任に発展しました。

浅田議員の発言、「公文書の管理にハッシュ関数やブロックチェーンを利用すること」のメリットを著者なりに解釈すると、

- ハッシュ値により、「決裁文書」が決裁時点から変更されていないことを保証できる
- ブロックチェーンに記録することで、決済文書をハッシュ値ごと「なかったこと」にできない

ということだと思います。[k]

暗号通貨には少しいかがわしいイメージがつきまとっていますが、使われている技術そのものは、民主的な国家を維持する上で必要不可欠な「国民の知る権利」や「正しい情報公開」のために使われるかもしれません。[10]

暗号通貨以外へのブロックチェーン技術の活用の取り組みは世界中で行われており、特にエストニアでは、登記や納税など、多くの公的サービスでブロックチェーンが利用されています。

暗号通貨に対しては、投機の対象として価格のボラティリティ（変化の激しさ）に一喜一憂するか、いかがわしいものだと無視するか、どちらかの立場となることが多いと感じています。しかし、そこで使われている暗号技術をできるだけ正確に、冷静に知ることは、間違いなく暗号通貨以外のところでも役に立つことでしょう。本稿がそのきっかけになれば、幸いです。

j d_Ad_BGはGのスカラー倍であり、Gと同様に平面上の点です。通常、d_Ad_BGのx座標が共通鍵として使われます。

k 改ざん防止、削除防止のためにブロックチェーンも使えますが、ブロックチェーンでないと実現できないわけではありません。タイムスタンプ、ヒステリシス署名など、既存の技術が存在します。ただし、改ざんや削除を防ぐ原理は共通であると考えます。

謝辞

本稿の作成にあたり、「知の発信」SIG のメンバをはじめとする CITP フォーラムの方々、またデジタルフィールドの同僚にレビューをいただきました。特に青木伸輔さん（中電 CTI）、服部智明さん（NEC）から多くのアドバイスを頂戴しました。貴重な時間を割いていただき、本当にありがとうございました。ただし、本稿に残った誤りとレビュー結果の未反映は、全て赤根の責任です。

参考文献・資料

[1]木下宏揚. サトシ・ナカモトは現代のマルコーニか？. 情報処理学会 連続セミナー2016 第 6 回：「フィンテック～ブロックチェインの理解と応用～」資料. 情報処理学会 (2016)
http://www.ipsj.or.jp/event/seminar/2016/program06.html

[2]アーシュラ・K. ル=グウィン (著), 清水真砂子 (翻訳). 影との戦い ゲド戦記 I, 岩波書店 (1999)
https://www.iwanami.co.jp/book/b260752.html

[3]結城浩. 暗号技術入門 第 3 版 秘密の国のアリス. SB クリエイティブ株式会社 (2015)
http://www.hyuki.com/cr/

[4]神保雅一, イオタゼミ. なるほどナットク！ 暗号がわかる本. オーム社 (2004)
http://shop.ohmsha.co.jp/shopdetail/000000002099/

[5]神永正博. 現代暗号入門　いかにして秘密は守られるのか(ブルーバックス). 講談社 (2017)
http://bookclub.kodansha.co.jp/product?isbn=9784065020357

[6]アンドレアス・M・アントノプロス（著）, 今井崇也,鳩貝淳一郎（訳）. 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. NTT 出版 (2016)
http://www.nttpub.co.jp/search/books/detail/100002391

[7]萩田真理子. 暗号のための代数入門. サイエンス社. p87 (2010)
http://www.saiensu.co.jp/?page=book_details&ISBN=ISBN978-4-7819-1268-4&YEAR=2010

[8]伊豆哲也. 楕円曲線暗号入門. (2013)
https://researchmap.jp/mulzrkzae-42427/

[9]辻井重男, 笠原正雄(編著), 有田正剛, 境隆一,只木孝太郎, 趙晋輝, 松尾和人(共著). 暗号理論と楕円曲線, 森北出版 (2008)
http://www.morikita.co.jp/books/book/2213

[10]野口悠紀雄. 森友問題の公文書改ざんはブロックチェーン技術で防げる. ダイヤモンド・オンライン (2018)
http://diamond.jp/articles/-/163327

著者紹介

赤根大吾（認定番号：14000022）
（株）デジタルフィールド
取締役
情報処理安全確保支援士
（登録番号 第 002400 号）

ソフトウェア開発インフラの構築、継続的インテグレーションの導入に従事。高度情報技術者（テクニカルエンジニア（情報セキュリティ）、ネットワークスペシャリスト、プロジェクトマネージャ、IT ストラテジスト）。TOEIC 885 (2015 Apr)。@dgakane on Twitter