暗号について

概念的なことを学びます。具体的なことは各自調べてみて下さい。暗号技術入門超おすすめ。

前置き

今日やりたいことは以下です。

- 暗号について
- 共通鍵暗号
- 公開鍵暗号
- ハッシュ関数
- メッセージ認証コード
- 電子署名
- SSL証明書

おまけで時間があればブロックチェーンとJWTについて話します。
OAuth2の説明までは時間的に厳しそう・・・

暗号

ルールに則って元のデータ（平文）をよくわからないデータ（暗号文）にすること
データは文章だったり、数字だったり、バイナリだったりと何でも良いです（というかコンピューター上は全部二進数だし）

例１:

```
ニイタカヤマノボレ
↓
戦闘行動を開始せよ


トラ・トラ・トラ
↓
ワレ奇襲ニ成功セリ
```

例２：

```
TAICHI
↓
UBJDIJ


こんにちは
↓
はちにんこ
```

例１と例２は少し違った方法で暗号化しています。分かりますか？

PCで使われる暗号化は例２のほうです。

```
TAICHI
↓
UBJDIJ
```

これはアルファベットを後ろへ1文字ずらしています。いわゆるシーザー暗号ってやつです。

```
TAICHI
↓
VCKEJK
```

これはアルファベットを後ろへ2文字ずらしています。

こんな単純は暗号化ですが、現代の暗号において大切なルール（アルゴリズム）と鍵（パスワード）が含まれています。

この暗号化のルールは「アルファベットを後ろへずらす」です。
暗号化の時に使う鍵はここでいう、1文字なのか2文字なのかです。
シーザー暗号はルールも鍵も単純なので簡単に破れますが実際にはルールも鍵ももっと複雑です。
ルールは秘密にする場合もありますが、インターネットの世界で使われている暗号化はルールが公開されています。

そして、この鍵を相手も自分も持っていないと解けない暗号を共通鍵暗号といいます。

共通鍵暗号

これはもの凄くイメージしやすいと思います。同じ鍵（パスワード）を使って暗号化と複合化をします。

zipで暗号化してメールでパスワードをおくるあれですｗ
これは現実世界でも上図のように同じ鍵を２人が持っていればその２人だけが開けられますよね。
つーか、何を当たり前の事を言っているんだと。

インターネットの世界で使われる共通鍵暗号は色々ありますが、有名なモノは「DES」、「AES」です。
おそらくこの二つの名前を知っていれば困る事はないです。

DES

45年も前にアメリカで作られた暗号化の規格。
20年以上も前に1日かからずに突破出来てしまっている。
広く使われているし、枯れた技術だし、これしか対応していないモノもあるしってことでなかなかなくならない。
なのでタダのDESと互換性のある、3回暗号化するトリプルDESってのが使われているけど3回も計算するのでチョット遅い。
なんでタダのDESと互換性があるかは暗号技術入門読んで下さい。

AES

DESやばいよねってことで20年前につくられた規格。
アメリカのNISTってところが次世代暗号をオープンに募集。５つ残った最終候補のうちの一つ。
強固で速い。みんな大好きパスワード付きzipでも使われています。

自転車の番号合わせて開ける鍵。
本体が暗号化のアルゴリズム、数字が鍵。数字が長ければ長いほど強固になるのと同じく鍵長が長ければ強固。

ただこいつの問題は**鍵（パスワード）をどうやって相手にわたすのか**ってこと。
`暗号化してパスワード渡せばいいやん！` とか `メールで送ればいいじゃん！` ってのは無しです・・・・

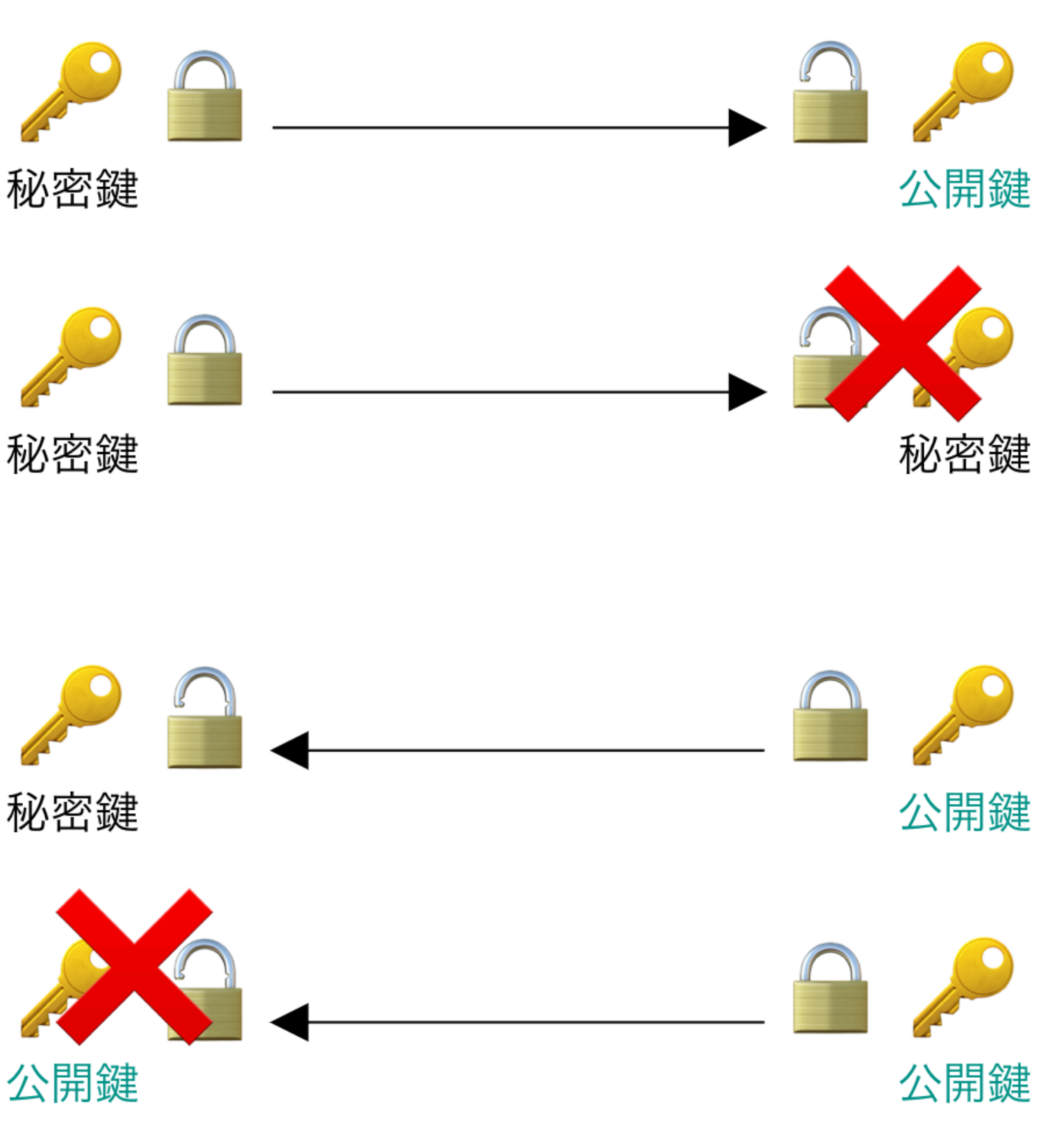
共通鍵ハンズオン

```
openssl enc --help
echo "Hello, World" > plain.txt
openssl enc -e -des -in plain.txt -out des.txt
openssl enc -d -des -in des.txt -out plain2.txt
```

`-des` を `-aes128` 等に変更すれば使用する暗号化方式を変えられます。

公開鍵暗号

こいつがよく分からんのはわかる。



鍵が二つあります。片方の鍵で暗号化されたものはもう一方でしか復号できない。逆もしかり。
魔法のような超便利な暗号です。
鍵の受け渡し問題が解決しちゃいます。何でかの説明は口頭でします。

この公開鍵暗号も幾つか種類がありますが一番有名なのがRSA暗号です

RSA暗号

証明までですると大変なので、こういう性質を使ってやっているんだって説明をします。
中学生の数学で分かることしか使わないでできます。（対数が分かると少し楽ですが）

素数 `p` , `q` があって

$$N = p \times q \quad (p, q は素数)$$

とします。つまり `N` は二つの素数の積です。
次に `p -1` と `q - 1` の最小公倍数 `L` を計算します。

$$L = lcm(p-1)(q-1) \quad (L は p-1, q-1 の最小公倍数)$$

ちなみに最小公倍数は一番小さい共通する倍数です。
例えば4と６があったとしたらそれぞれの倍数は

$$\begin{aligned} &4 : 4, 8, 12, 16, 20, 24, ... \\ &6 : 6, 12, 18, 24, 30, ... \end{aligned}$$

となり、この中だと12と24が共通の倍数、つまり公倍数です。なので最小公倍数は12になります。
次に `E` を求めます。こいつは一言で表すと

```
1より大きくてLより小さいLと互いに素の数
```

となってよく分からんので説明すると、つまり、EとLの最大公約数は1だってことです。

$$\begin{aligned} &1 < E < L \\ &gcd(E, L) = 1 \quad (E と L の最大公約数は 1) \end{aligned}$$

次にDを求めます。ソロソロ何やってんだっていわれそうですがもう少し。
Dは下記を満たします。

$$\begin{aligned} &1 < D < L \\ &E \times D \bmod L = 1 \quad (E と L の積を L で割った余りが１) \end{aligned}$$

で、結局なんなんだというと、一定の法則に従って `N, L, E, D` を二つの素数 `p, q` から求めました。
`L` は `E, D` を計算するために使っただけなので暗号化に使うのは `N, E, D` だけです。
この３つの数には次の性質があります。

$$\begin{aligned} &a^E \bmod N = b \\ &b^D \bmod N = a \end{aligned}$$

aを平文, bを暗号文とすると

$$\begin{aligned} &平文^E \bmod N = 暗号文 \\ &暗号文^D \bmod N = 平文 \end{aligned}$$

つまり、一定のルールで出した数字を使うとEとNを使って変換したものはDとNを使って戻せるということです。
実際にやってみると

$$\begin{aligned} &p = 5, q = 11 \\ &N = 5 \times 11 = 55 \\ &L = lcm(p-1)(q-1) = lcm(4, 10) = 20 \\ &E = 13 \quad (互いに素の数は 3, 7, 9, 11, 13, 17, 19 とあるのでその中の一つ) \\ &D = 17 \quad (20 で割った余りが 1 の 13 の倍数は 221 なので 221 \div 13 = 17) \end{aligned}$$

暗号化したいaを7とすると

$$\begin{aligned} &a = 7 \\ &b = a^E \bmod N = 7^{13} \bmod 55 = 2 \\ &a = 2^{17} \bmod 55 = 7 \end{aligned}$$

となって、暗号化復号化出来ました。つまり、公開鍵暗号でいう公開鍵は `E` 、秘密鍵は `D` ってことです！(もちろん暗号化、復号化に `N` も必要ですが)
つまり、 `D` を秘密にしてもっておけば `E` で変換したものは `D` でしか元に戻せないってことです！

なんかモニョります。これって相互に変換出来ているだけだよねって。
問題二つあります。
ひとつめが、 `N` から `p, q` が求められればあとは `E` と使って `D` が計算できるじゃんってのと
ふたつめが、下記の方程式をとけばいいだけじゃんと。

$$暗号文 = x^E \bmod N$$

ひとつめは素因数分解って難しいよねってことで担保されています。
つまり、メチャクチャ大きな二つの素数をかけた数字を二つの素数に戻すのって今存在するアルゴリズムやスパコンでも無理なんです。
ちなみに600桁くらいの数字を使ってます！ｗ

ふたつめはこの方程式は離散対数問題に帰結されます。離散対数問題が何かは興味があったら調べて貰ってこれも、今はまだ現実的な時間では解けないっていうことで担保されています。

つまり、アルゴリズムが開発されたり計算速度が今よりもっと速くなったら使い物にならないってことです。

RSAハンズオン

```
openssl genrsa 2024 > secret.key
openssl rsa -pubout -in secret.key -out public.key

openssl rsa -in secret.key -text #秘密鍵の中身を確認
openssl rsa -pubin -in public.key -text #公開鍵の中身を確認

openssl rsautl -encrypt -pubin -inkey public.key -in plain.txt -out rsa.txt #暗号化
openssl rsautl -decrypt -inkey secret.key -in rsa.txt -out plain3.txt #復号化
```

どっちでも大丈夫

$$\begin{aligned} &平文^E \bmod N = 暗号文 \\ &暗号文^D \bmod N = 平文 \end{aligned}$$

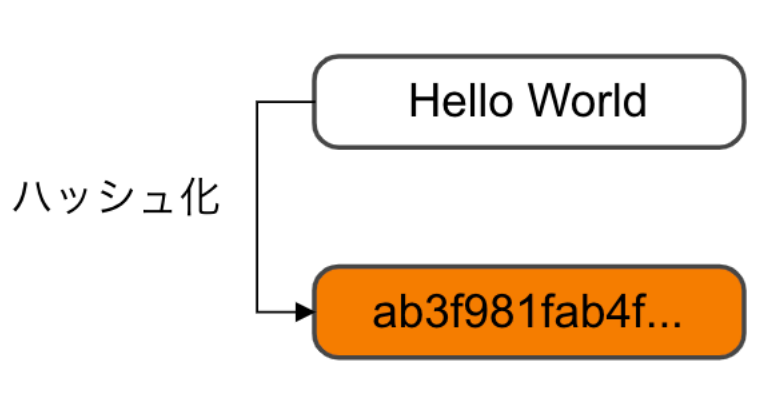
上記の式をみると相互に変換できていますよね。つまり、秘密鍵で暗号化したら公開鍵で復号できるし、公開鍵で暗号化したら秘密鍵で復号化できるってことです。ｓ

ハッシュ関数

データを入れたら固定の長さの数字にしてくれる関数。しかも違うデータをいれれば別の数字にしてくれる。
つまり

```
AAAAA -> 1
ABABA -> 8
AAAAB -> 9
```

一桁の数だと10通りにしかならないが、例えばmd5というハッシュ関数は128ビット、つまり 42億 x 42億 x 42億 x 42億 通りある。
ただ、世の中のデータの数は無限なのでできる限り同じにならないようにしているが違うデータから同じ数字が出ることがある。
これを「衝突」といいます。この辺りに興味があれば暗号技術入門読んで下さい。ここでは違うデータからは違う数字がでる関数とだけ認識を。
つまり、**データの指紋って考えてると分かりやすいです**



ハッシュ関数ハンズオン

```
md5 ファイル名
shasum -a 256 ファイル名
```

色々な使い方があるけれど、今回関係ありそうなのはデータの変更の検出に使えます。

Aファイルのハッシュ値が `c943cccdaf73f8d7e37b623ebc8292ae` だとします。
それを数日後にまたハッシュ値を取ると `b92a826a967d86242d7bc75590dbdf0b` になっていたとします。
そうするとAファイルは変更されていることになるわけです。
これを応用するとファイルの改竄の検出に使えます。それが次のメッセージ認証コードMACです。

メッセージ認証コード

そろそろお腹いっぱいになってきたかもですがまだSSLのSの字もでてきていません。
メッセージ認証コード（Message Authentication Code）、MACです・・・。
コンピュータの世界にはMACが少なくとも３つあります。MacとMACアドレスとこのMACです。
これは分かってしまえば大したことないんです。考えた人すげーと思います。

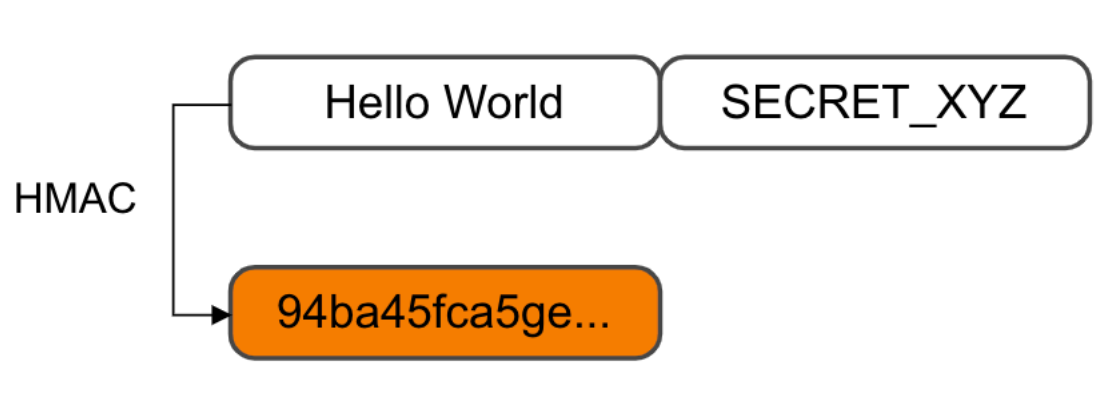
メッセージ認証コードの実現方法

幾つかありますがハッシュを使ったものを覚えておけば大丈夫だと思います。というかそれしか使ったこと無い・・・。ハッシュを使ったMAC、詰まりHMACです。

仕組み

`Hello World` という文字列をAさんがBさんに送りたいです。その際にAさん、Cさん共に `SECRET_XYZ` という文字列（シークレットキー）を共有しておきます。

Aさんは md5(`Hello World` + `SECRET_XYZ`) の様に文字列を繋げてハッシュ値 `H` を取ります。
この `H` と一緒に `Hello World` を Bさんに送ります。
Bさんは `Hello World` と自分が持っている `SECRET_XYZ` を使ってハッシュ値を作り `H` と比較します。同じならデータは改竄されていないことになります。



HMAC ハンズオン

```
cat plain.txt | openssl dgst -sha256 -hmac "hogehoge"
```

問題

AさんがBさんを経由してCさんへ何かデータを送ります。Bさんも中身を確認する必要があります。
その際にデータの改竄を防ぎたいです。どうしたらベストでしょうか？
（これはJWTがまさにやっていることです）

1. 平文をそのまま

超アウト。改竄し放題！

2. 暗号化して送る

登場人物がAさんとCさんだけなら⭕です。お互いがお互いしか知らないパスワードで暗号化して送れば大丈夫です。

3. ハッシュ関数を使う

あらかじめAさんがデータのハッシュ値を取っておいて、Cさんに渡しておきます。
一見⭕なようですがこのハッシュ値を問題無く送れるのなら最初からデータを送ればいいじゃないかって・・・。

4. メッセージ認証コードをつかう

概ねOKです。概ね！

そう、概ねなんです。幾つか問題がのこっていますが、その一つはHMACにつかうシークレットキーをどうやって相手に送るか問題です。あらかじめ共有出来るならそれはそれでOKです。
もう一つは、同じメッセージを相手も自分も持っている問題です。受け取った相手が改竄しないなんて保証はないですよね？・・・・ｗ（否認問題）

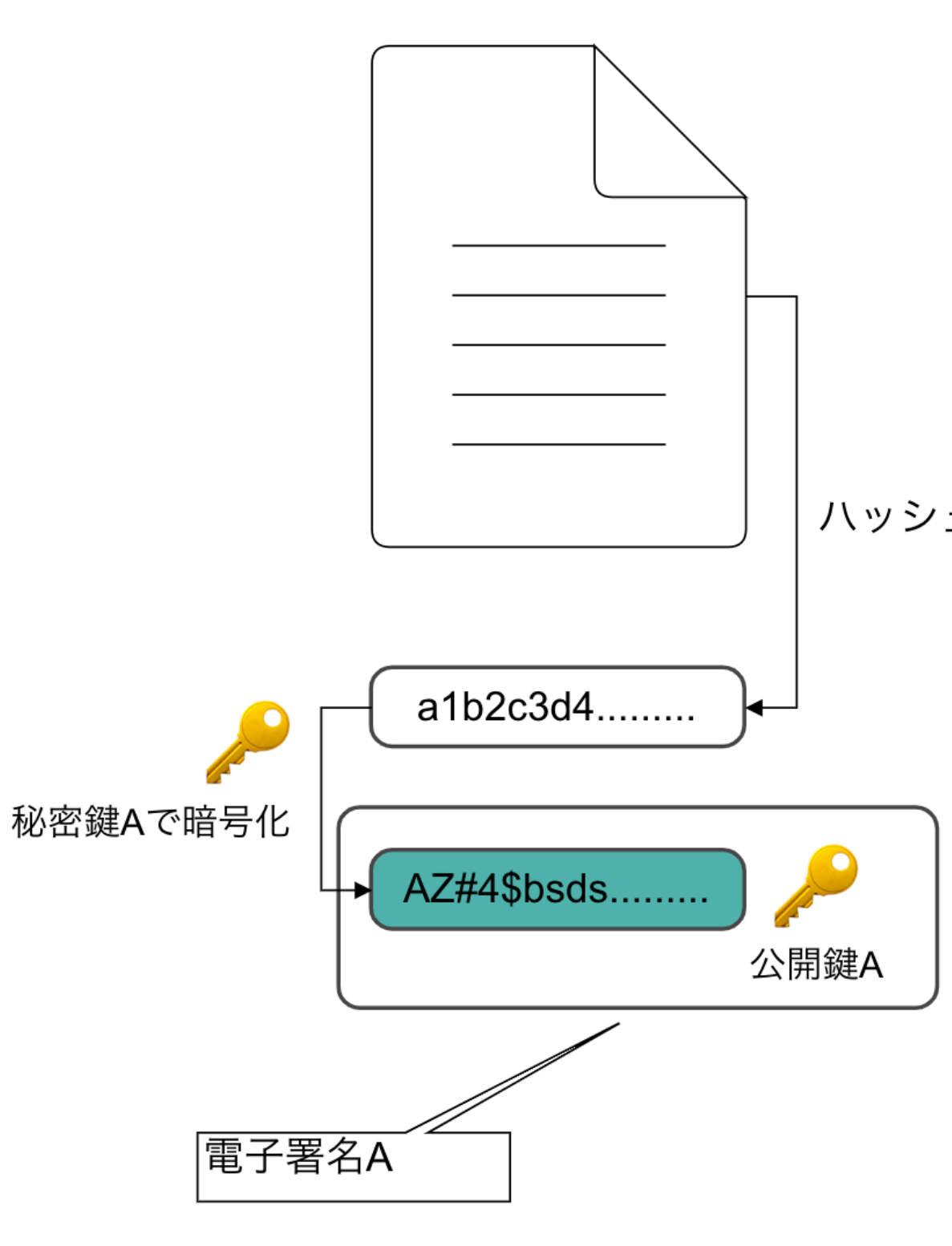
そこで、電子署名が出てきます！

電子署名

メッセージ認証コードの問題は鍵の受け渡しと、鍵を持っている相手の改竄やねつ造を否認できない古都でした。これを解決するには公開鍵をつかってお互いが違う鍵を持っていればいいんです。

Aさんはデータを自分の秘密鍵で暗号化して公開鍵を一緒に送ります。これだけです。
そうすると、そのメッセージはその公開鍵と対になる秘密鍵を持った人が作ったデータだという事になります。
ただ、公開鍵は処理の負荷が大きすぎるので大抵はデータのハッシュ値を暗号化して送ります。
これが電子署名の正体です。
e-taxでマイナンバーカード使うときに電子署名つけるますが、これはマイナンバーカードの中に秘密鍵が入っていて申告書のデータのハッシュ値をその秘密鍵で暗号化して税務署に公開鍵と一緒に渡しているんだと思います。多分。

で、これでおしまいと思いきやまだ問題が・・・・。
AさんからBさんのデータに電子署名をつけたのは良いけど、この署名本当にAさんのなの？って問題です。ここでやっと証明書が出てきます。



証明書

SSL証明書含めて証明書ってなにかっていうと実は上記の電子署名につけた公開鍵につけた電子署名です。なんか再帰的ですよね。まさに再帰的な仕組みです。

AさんからBさんにデータを渡すときに電子署名をつけました。電子署名はハッシュを暗号化したものとそれを復号化出来る公開鍵です。
Bさんの問題はその公開鍵が本当にAさんのものなのかということです。もしかしたらCさんが作ってAさんの名前をかたっているだけかもしれません。

どうするかというと、第三者に証明してもらうしかありません。人は自分の事を自分で証明出来ないんです。

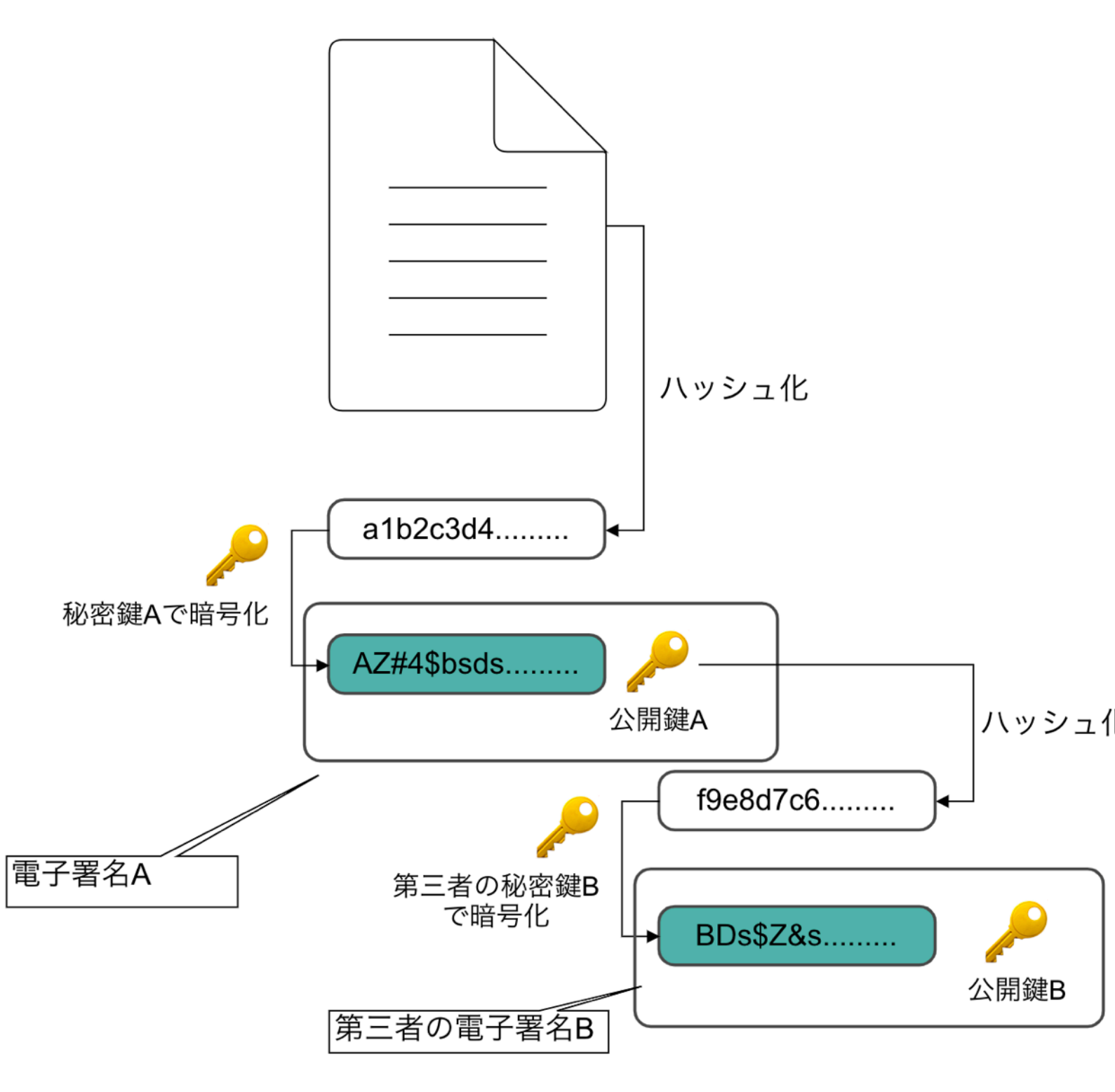
なので、第三者に証明してもらうために、Aさんの使っている公開鍵はAさんのものですよと第三者に電子署名をつけて貰います。電子署名が着いているって事はその公開鍵は第三者以外の人が証明していないってことです。

・・・・・・・。

？？？？？？？。

なんか変ですよねｗ

その第三者の電子署名が第三者のものだって証明は？



最後に

暗号化だけでなく、低レイヤーのことを学ぶには論理演算ビット演算を多少知っていると理解が深まります。
特に暗号化はxor使いまくりだし。