

# 公開鍵暗号1

光成滋生

last update: 2025/10/20

## 目的

- 共通鍵暗号やMAC, AEADに必要な秘密鍵を共有する方法を学ぶ
- 相手が本当の相手であることを検証するための署名やその応用, PKIを学ぶ
- 実際の署名アルゴリズムを理解する

# 目次

## 用語一覧

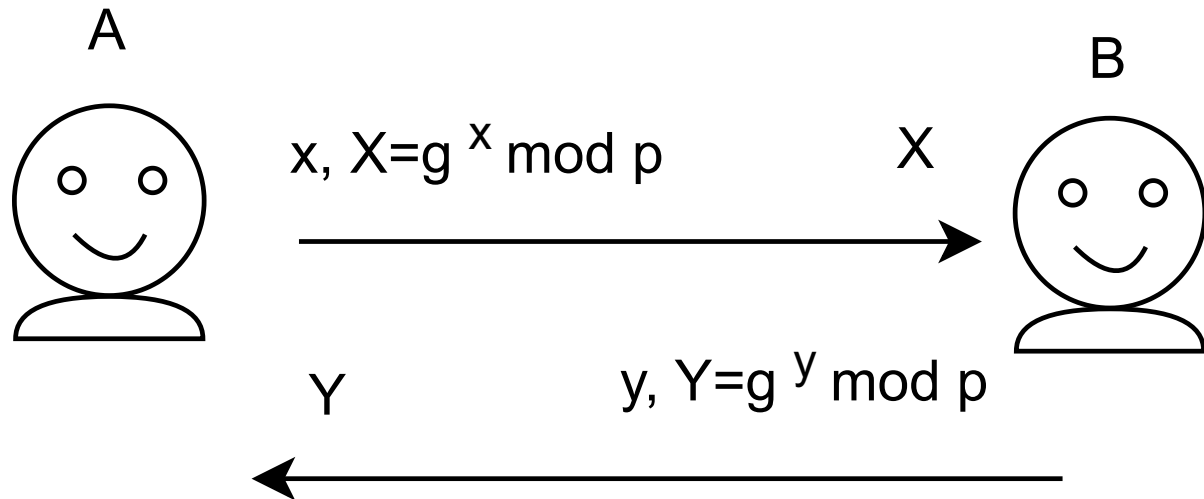
- 公開鍵暗号, DH鍵共有, DLP, DHP
- 楕円曲線, ECDH鍵共有, 群, スカラー倍算
- 有限体, Euclidの互除法, 逆元
- 二項定理, Fermatの小定理
- 加法公式, 射影座標
- 能動的な攻撃者, AitM攻撃
- 署名, sEUF-CMA
- ECDSA, 安全でない乱数の危険性
- RSASSA-PKCS1-v1\_5, RSASSA-PSS
- sshの公開鍵認証, PKI

# DH (Diffie-Hellman) 鍵共有

## 盗聴者のいる通信経路で安全に通信する方法

### AとBの間の鍵共有

- $p$ : 素数,  $g$ : 0でない整数 ( $0 < g < p$ ) を選び  $p$  と  $g$  を共有する (public)
- A は  $x \in [1, p - 1]$  をランダムに選んで  $X := g^x \bmod p$  ( $p$  で割った余り) を B に送る
- B は  $y \in [1, p - 1]$  をランダムに選んで  $Y := g^y \bmod p$  を A に送る
- A は  $s_1 = Y^x \bmod p$ , B は  $s_2 = X^y \bmod p$  を計算する



# これでうまくいくのか？

$$s_1 = s_2 = s$$

- 整数  $a, b$  に対して  $(a \bmod p) \times (b \bmod p) \bmod p = (a \times b) \bmod p$
- よって  $a^b \bmod p = (a \bmod p)^b \bmod p$
- $s_1 = Y^x \bmod p = (g^y \bmod p)^x \bmod p = (g^y)^x \bmod p = g^{xy} \bmod p = s_2$ 
  - 以降、煩雑なので  $\bmod p$  は省略する

## 安全なのか？

- 盗聴者は  $p, g, X, Y$  を盗聴できる
- 盗聴者はこれらの情報から  $s$  を計算できるか？
- $X = g^x$  が分かっているのだから、 $g, g^2, g^3, \dots$  と計算して  $X = g^x$  となる  $x$  を探せばよい
- そうすれば  $Y^x = g^{xy} = s$  も計算できる
  - $p$  が十分大きいと、 $x$  を見つけるのが困難であることが知られている

# 離散対数問題 DLP (Discrete Logarithm Problem)

## 定式化

- $p, g, X$  が与えられたとき、 $X = g^x \bmod p$  となる  $x$  を求める問題
  - $y = a^x$  のとき  $x = \log_a y$  を底  $a$  について  $y$  の対数と呼んだ
  - 今回はその整数版なので離散対数という

## DLP を解くコスト

- 2025年現在, 大きな $p$ について一般数体篩法 GNFS (General Number Field Sieve) が最良の手法
- 計算コストは  $L_p[1/3, (64/9)^{1/3}]$ 
  - $L_p[a, c] := \exp((c + o(1))(\log p)^a (\log \log p)^{1-a})$ : 準指数時間
  - $f(x)$  が  $o(g(n))$  であるとは  $\lim_{x \rightarrow \infty} f(x)/g(x) = 0$  であること
  - $L_p[1, c] \sim p^c$  ( $\log p$  の指数時間),  $L_p[0, c] \sim (\log p)^c$  ( $\log p$  の多項式時間)
- 例えば  $p = 2^{2048}$  で  $2^{116}$ ,  $p = 2^{3000}$  で  $2^{137}$  程度
  - 3000bit の素数を使うと128bitセキュリティの安全性があるとみなせる

# DH問題

## DLP が困難だけでOKか？

- $x, y$  を求められなくても直接  $s$  を求める方法があるかもしれない

## DHP (Diffie-Hellman Problem)

- $p, g, X = g^x \bmod p, Y = g^y \bmod p$  が与えられたとき  $s = g^{xy} \bmod p$  を求めよ

## DLPとDHPの関係

- 先程見たようにDLPが解ければDHPも解ける
  - 問題としてはDHPの方が易しい
  - 2025年現在, DHPを解く最良の手法はDLPを解くこと
  - DHPがDLPより真に易しいか・同じぐらいの難しさかは不明

## 安全性の言い換え

- DHPが困難ならDH鍵共有は安全

# PQC (Post-Quantum Cryptography)

## 量子計算機に対しても安全な暗号技術

- 耐量子計算機暗号
- $\log p$  の多項式時間でDLPを解く量子アルゴリズムが知られている
- 量子計算機が実用化されるとDH鍵共有は安全でなくなる
- 量子計算機に安全な鍵共有方法が必要

## ML-KEM (Module-Lattice Key Encapsulation Mechanism)

- [FIPS 203](#): 2024年NISTで標準化された
- ブラウザ Chrome, Edge, Firefoxなどで利用可能になってきている
  - [Cloudflare Research](#)
- 詳細は講義の後半で



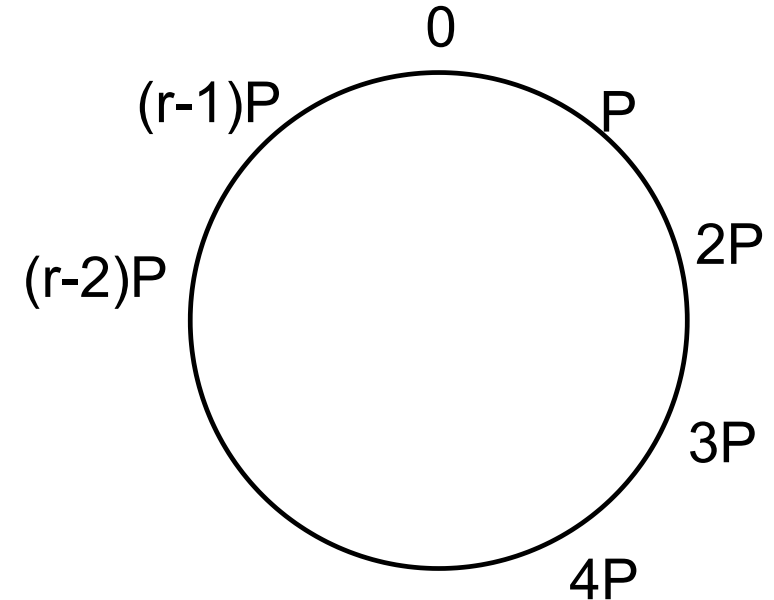
# 楕円曲線

## 楕円曲線 EC (Elliptic Curve) とは

- 「楕円」でも「曲線」でも無い「楕円曲線」という数学用語
  - イメージ的には「複素曲線」で浮輪の表面のようなもの
  - 代数的な定義は後述

## 楕円曲線の点集合

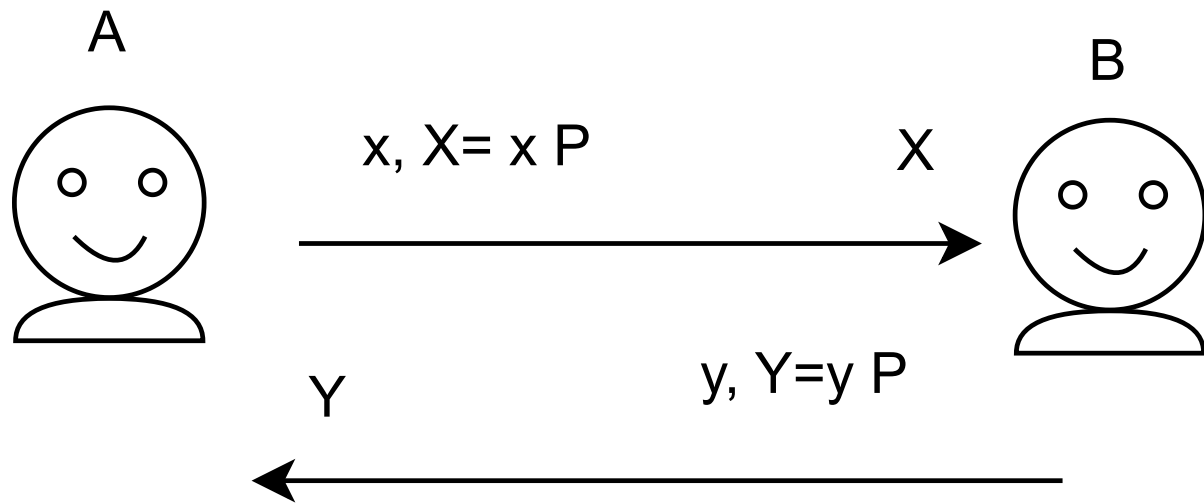
- $G$  を  $\{0, P, 2P, \dots, (r-1)P\}$  という形で表せる楕円曲線の  $r$  個の点集合とする
  - $G$  には  $aP \pm bP := ((a \pm b) \bmod r)P$  という加減算が定義されている
  - $(aP + bP) + cP = aP + (bP + cP) = (a + b + c)P$
  - $aP := P + P + \dots + P$  ( $a$  個の  $P$  の和) ,  $a(bP) = (ab)P$
  - $0$  は整数の  $0$  に相当する点,  $-aP := (r - a)P$  とする



# ECDH鍵共有

## DH鍵共有の類推

- A は  $x \in [1, r - 1]$  をランダムに選んで  $X := xP$  を B に送る
- B は  $y \in [1, r - 1]$  をランダムに選んで  $Y := yP$  を A に送る
- A は  $s_1 = xY = xyP$ , B は  $s_2 = yX = yxP = xyP$  を計算する



## DH鍵共有との比較

- $g^x, g^y$  の代わりに  $xP, yP$ .  $g^{xy}$  の代わりに  $xyP$

# ECDH鍵共有の安全性

## DLPとDHPの楕円曲線版

- ECDLP:  $P, X = xP$  が与えられたとき  $x$  を求めよ
- ECDHP:  $P, X = xP, Y = yP$  が与えられたとき  $xyP$  を求めよ

## ECDHPの難しさ

- 2025年現在, ECDHPを解く最良の手法はECDLPを解くこと
  - DHPと同様にECDHPがECDLPより真に易しいか・同じぐらいの難しさかは不明
- DH鍵共有が3000bitで128bitセキュリティだったのに対して  
ECDH鍵共有は256bitの小さい鍵で同じセキュリティレベル

## ECDHの量子計算機に対する安全性

- $n$  bitのECDHは  $O(n^3)$  の多項式時間で解けるので安全ではないが  
TLS1.3で標準的に使われている鍵共有手法

# 抽象的な定義

## DHとECDHの共通点

- どちらも群構造を利用している

## 群とは集合 $G$ が次の性質を満たすもの

- $f : G \times G \rightarrow G$  が存在し,  $f(a, b)$  を  $a \cdot b$  と書く ( $ab$  と書くことも)
  - 結合則:  $(ab)c = a(bc)$  for  $\forall a, b, c \in G$
  - 単位元:  $e \in G$  が存在して  $ea = ae = a$  ( $e = 1$  と書くことがある)
  - 逆元:  $\forall a \in G$  に対して  $a^{-1} \in G$  が存在して  $aa^{-1} = a^{-1}a = e$

## 可換群

- 群  $G$  について  $\forall a, b \in G$  に対して  $ab = ba$  が成り立つ
- このとき  $ab$  を  $a + b$  と書き, 単位元を  $e = 0$ , 逆元を  $-a$  と書くことがある (加法群という)

# 群の例

**整数の集合  $G := \mathbb{Z}$**

- $a, b \in G$  に対して  $f(a, b) := a + b$ , 0が単位元の可換群

**0以外の有理数の集合  $G := \mathbb{Q}^*$**

- $a, b \in G$  に対して  $f(a, b) := ab$ , 1が単位元の可換群

**2行2列の実数行列で逆行列を持つ行列の集合  $G := GL_2(\mathbb{R})$**

- $a, b \in G$  に対して  $f(a, b) := ab$  (行列の積), 単位行列が単位元の非可換な群

**$p$  で割った余りで考える集合  $G := \langle g \rangle := \{1, g^1, g^2, \dots, g^{r-1}\}$**

- $a, b \in G$  に対して  $f(a, b) := ab$ , 1が単位元の可換群

**楕円曲線の点の部分集合  $G := \{0, P, \dots, (r-1)P\}$ : 可換群**

# 巡回群

## 群 $G$ が巡回群であるとは

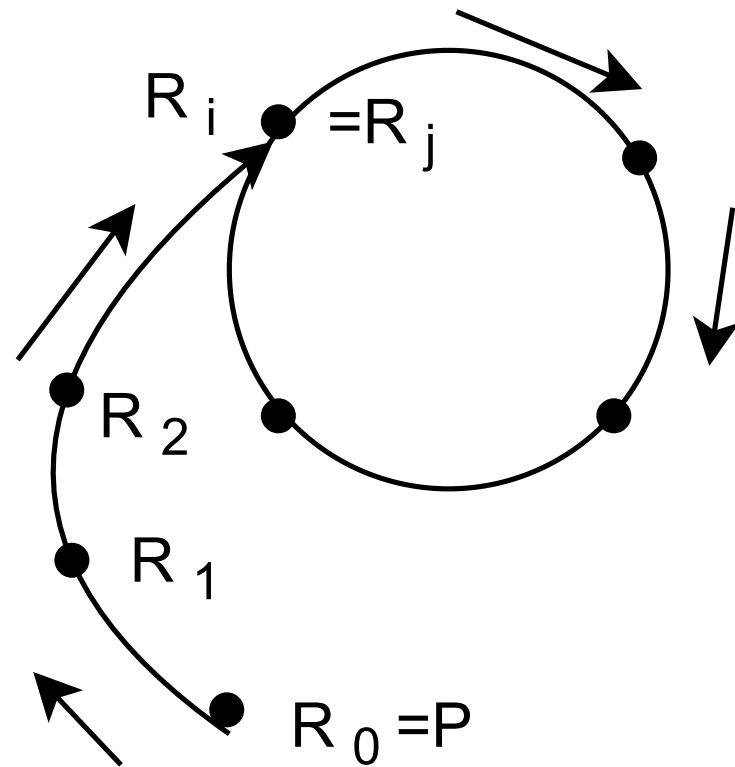
- $g \in G$  に対して  $\langle g \rangle := \{e, g, g^2, g^3, \dots\}$  を  $g$  で生成される部分群（巡回群）という
  - $\langle e \rangle = \{e\}$ .
- $g^r = e$  となる  $r$  が存在すれば  $\langle g \rangle = \{e, g, g^2, \dots, g^{r-1}\}$  となり有限巡回群という
  - $r$  を  $\langle g \rangle$  の位数といい  $r = |\langle g \rangle|$  と書く
- 巡回群は可換群である
  - $a, b \in \langle g \rangle$  に対して  $a = g^x, b = g^y$  と書けて  $ab = g^{x+y} = g^{y+x} = ba$

## 有限巡回群 $G = \langle g \rangle$ に関するDH鍵共有

- $x, y \in [1, r-1]$  をランダムに選んで  $X = g^x, Y = g^y$  を交換し  $g^{xy}$  を共有する
- DH鍵共有とECDHの鍵共有は  $f(a, b)$  を  $ab$  と書くか  $a + b$  と書くかの違いだけ
- 実際論文によっては楕円曲線の演算を  $g^a$  の形で書いてあるものもある（講義では  $+$  を使う）

## 位数 $r$ の加法巡回群 $G = \langle P \rangle$ のDLPに対する一般的な攻撃

- $Q = aP \in G$  となる  $a$  を見つけたい
- $m$ : 分割数を固定し, 整数  $(u_i, v_i)$  をランダムに選び  
 $M_i := u_i P + v_i Q$  とする (事前計算)  
 $g(R) := R_x \bmod m$  とする
- $R_0 := P, R_{i+1} := R_i + M_{g(R_i)}$  と順次求める
  - $R_i = a_i P + b_i Q$  の形
- $R_i = R_j$  となる  $i < j$  が見つかったとする (衝突)
  - $R_i = a_i P + b_i Q = a_j P + b_j Q = R_j$
  - $Q = (a_i - a_j)/(b_j - b_i)P \# b_i = b_j$  ならやり直し
- 衝突するのに必要なステップ数は  $O(\sqrt{r})$
- 今 (2025年) のところこれより効率よくECDLPを解く方法は知られていない
- $r$  bitのECDLPの安全性は  $r/2$  bit



# 楕円曲線のためにしばらく数学の準備

## 合同

- 整数  $m(> 0)$ ,  $a, b$  に対して  $a \equiv b \pmod{m}$  とは  $(a - b) \bmod m = 0$  のときをいう
  - $m$  で割った余りが同じ. このとき  $a$  と  $b$  は  $m$  を法として合同という
  - 注意: 「 $a \bmod m$ 」は  $a$  を  $m$  で割った余り (0以上  $m$  未満の整数)

## 整数環

- 整数  $m(> 0)$  に対して集合  $\mathbb{Z}/m\mathbb{Z} := \{0, 1, 2, \dots, m - 1\}$  とする ( $\mathbb{Z}/m$  と略記する)
- 加減算:  $a \pm b := (a \pm b) \bmod m$ . 乗法:  $ab := (ab) \bmod m$  for  $a, b \in \mathbb{Z}/m$ 
  - 加算と乗算は可換:  $a + b = b + a, ab = ba$
  - 加法に関する単位元 0:  $a + 0 = a$ , 乗法に関する単位元 1:  $a \times 1 = a$
  - 結合法則:  $(a + b) + c = a + (b + c), (ab)c = a(bc)$
  - 分配法則:  $a(b + c) = ab + ac$
- これらを満たすものを可換環という



# 除算の定義

## 思考

- $\mathbb{Z}/m \ni a, b (\neq 0)$  に対して除算  $a/b$  を定義したいが  $a/b$  は整数にならない場合がある
- まず逆数  $1/a$  を考える

## 逆数とは

- $ax = 1$  となる  $x$  が  $a$  の逆数
- 整数環でも  $ab \equiv 1 \pmod{m}$  となる  $b$  があれば  $b$  を  $a$  の逆数と呼んでよいのでは
- 例えば  $m = 7$  のとき  $3 \times 5 \equiv 1 \pmod{7}$  なので  $3$  の逆数  $1/3 = 5$  とする
- 例えば  $m = 6$  のとき  $3 \times x \not\equiv 1 \pmod{6}$  なので  $3$  の逆数は存在しない
  - $3 \times 1 \equiv 3, 3 \times 2 \equiv 0, 3 \times 3 \equiv 3, 3 \times 4 \equiv 0, 3 \times 5 \equiv 3$

## 素数 $p$ に対する整数環

- $\mathbb{F}_p := \mathbb{Z}/p = \{0, 1, 2, \dots, p-1\}$
- **定理:**  $\forall a \in \mathbb{F}_p^* := \mathbb{F}_p \setminus \{0\}$  に対して  $a$  の逆数が存在する
  - つまり  $ab \equiv 1 \pmod{p}$  となる  $b \in \mathbb{F}_p^*$  が存在する
- $\mathbb{F}_p$  は四則演算ができる
  - 逆元の存在:  $a \in \mathbb{F}_p^*$  に対して  $a^{-1} \in \mathbb{F}_p^*$  が存在して  $aa^{-1} = 1$
  - 問: 逆元は存在すればただ一つしかないことを示せ
- このとき  $\mathbb{F}_p$  を有限体という
  - 無限集合だけど四則演算が出来るものを体という
  - 有理数の集合  $\mathbb{Q}$ , 実数の集合  $\mathbb{R}$ , 複素数の集合  $\mathbb{C}$  も体

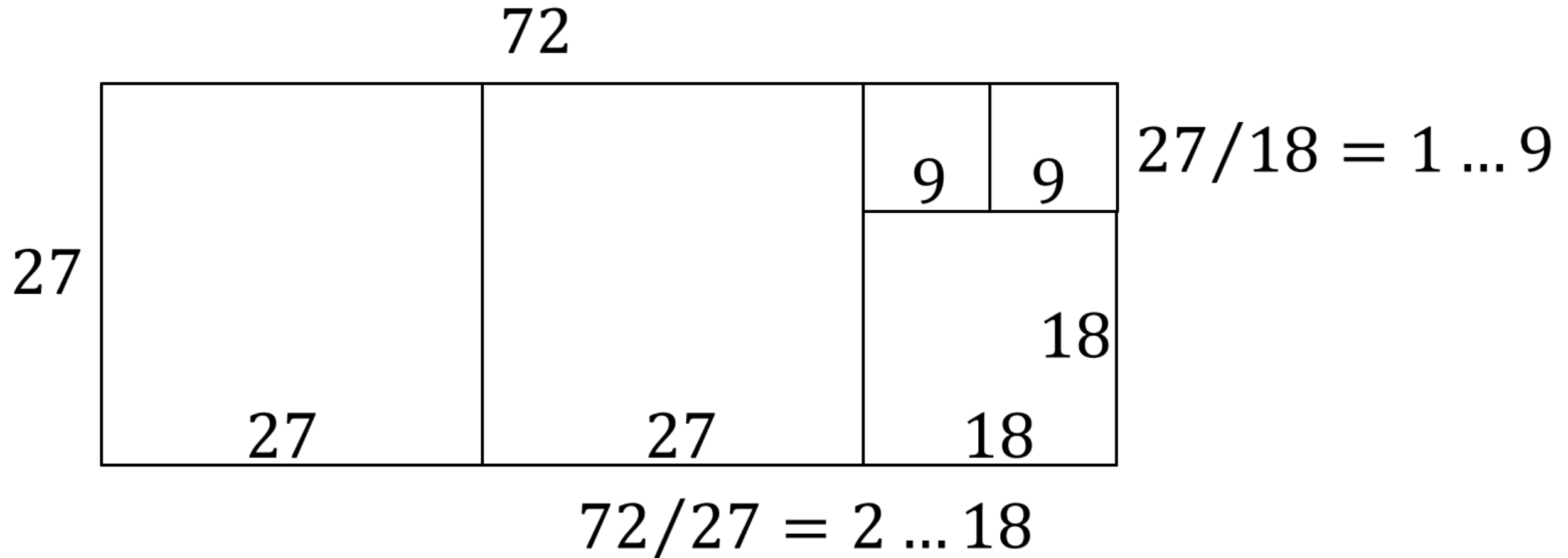
# Euclidの互除法

## 最大公約数 $\gcd(a, b)$ の性質

- $\gcd(a, 0) = a, \gcd(a, b) = \gcd(b, a)$ .
- $\gcd(a, b) = \gcd(a - b, b)$ 
  - $c := \gcd(a, b), c' := \gcd(a - b, b)$  とすると  $a = ca', b = cb'$  と書けて  $a - b = c(a' - b')$ . よって  $c$  は  $a - b$  と  $b$  の公約数となり,  $c'$  の最大性から  $c \leq c'$ .  
同様に  $a - b = c'x, b = c'y$  と書くと  $a = c'(x + y)$ . よって  $c'$  は  $a$  と  $b$  の公約数となり,  $c$  の最大性から  $c' \leq c$ . よって  $c = c'$ .
- $b \neq 0$  のとき  $\gcd(a, b) = \gcd(a \bmod b, b)$ .
  - $a$  を  $b$  で割った余りになるまで  $a - b$  に置き換えることを繰り返す
- $c = \gcd(a, b)$  に対して  $b = 0$  なら  $c = a$  で終了
  - $b > 0$  なら  $r_0 := a \bmod b$  として  $c = \gcd(r_0, b) = \gcd(b, r_0)$ .  $r_0 = 0$  なら終了
  - $r_0 > 0$  なら  $r_1 := b \bmod r_0$  として  $c = \gcd(r_1, r_0)$ .  $r_1 = 0$  なら終了
  - これを繰り返すといずれ終了し  $c$  が求まる

# $\gcd(72, 27)$ の例

72と27の長方形を書く



- $\gcd(72, 27) = \gcd(18, 27) = \gcd(27, 18) = \gcd(9, 18) = \gcd(18, 9) = 9$

# 拡張Euclidの互除法

$a, b$  に対して  $ax + by = \gcd(a, b)$  となる整数  $x, y$  が存在し求められる

- 先程の例で4回で終わったとする
  - $r_0 := a \bmod b: a = q_0b + r_0.$
  - $r_1 := b \bmod r_0: b = q_1r_0 + r_1.$
  - $r_2 := r_0 \bmod r_1: r_0 = q_2r_1 + r_2.$
  - $r_3 := r_1 \bmod r_2 = 0: r_1 = q_3r_2 + r_3.$
- これを順に代入すると
- $$\begin{aligned}\gcd(a, b) &= \gcd(r_0, r_1) = r_2 = r_0 - q_2r_1 = r_0 - q_2(b - q_1r_0) \\ &= (1 + q_1q_2)r_0 + b(-q_2) = (1 + q_1q_2)(a - bq_0) + b(-q_2) \\ &= a(1 + q_1q_2) + b(-q_0q_1q_2 - q_2) = ax + by \text{ となる } (x, y) \text{ が求まった}\end{aligned}$$

# 有限体の逆元

## 拡張Euclidの互除法を用いて逆元を求める

- $\forall a \in \mathbb{F}_p^*$  に対して  $a$  と  $p$  は互いに素である ( $p$  は素数だから)
  - つまり  $\gcd(a, p) = 1$
- 互除法を用いて  $ax + py = \gcd(a, p) = 1$  となる整数  $x, y$  が存在する
- これは  $ax \equiv 1 \pmod{p}$ , つまり  $x \bmod p$  は  $a$  の逆数

## $\mathbb{Z}/m$ が有限体になる条件

- $m$  が合成数  $m = uv$  ( $u, v > 1$ ) のとき,  $uv \equiv 0 \pmod{m}$  なので  $u$  の逆数は存在しない
- よって  $\mathbb{Z}/m$  が有限体になる必要十分条件は  $m$  が素数である

## 別の方法も紹介する

# 二項定理

## 組合せ（組み合わせ）の数

- $n$  個の中から  $k$  個選ぶ組合せ  $\binom{n}{k} := \frac{n!}{k!(n-k)!}$ . 例  $\binom{5}{2} = \frac{5!}{2!3!} = 10$
- $(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}$  が成り立つ
  - 問題: 数学的帰納法を使って証明せよ
  - 例:  $(x + y)^5 = x^5 + 5x^4y + 10x^3y^2 + 10x^2y^3 + 5xy^4 + y^5$
- 素数  $p$  と  $0 < k < p$  に対して  $\binom{p}{k}$  は  $p$  の倍数
  - $\binom{p}{k} = \frac{p!}{k!(p-k)!}$  の分子に  $p$  があり, 分母に  $p$  がない
- $(x + y)^p \equiv x^p + y^p \pmod{p}$ 
  - $(x + y)^p = \sum_{k=0}^p \binom{p}{k} x^k y^{p-k} \equiv x^p + y^p \pmod{p}$
- $x^p \equiv x \pmod{p}$ 
  - $x^p \equiv (1 + (x - 1))^p \equiv 1 + (x - 1)^p \equiv 1 + \dots + 1 \equiv x \pmod{p}$

# Fermatの小定理

素数  $p$  と  $a \in \mathbb{F}_p^*$  に対して  $a^{p-1} \equiv 1 \pmod{p}$

- $a^p \equiv a \pmod{p}$  より  $a(a^{p-1} - 1) \equiv 0 \pmod{p}$
- $a$  と  $p$  は互いに素なので  $a^{p-1} - 1 \equiv 0 \pmod{p}$ .

## 逆元の計算

- Fermatの小定理より  $a \in \mathbb{F}_p^*$  に対して  $a(a^{p-2}) \equiv 1 \pmod{p}$ .
- これは  $a^{-1} = a^{p-2} \pmod{p}$  を意味するので逆元を計算できる

## 注意

- $a = 0$  のときは  $a^{p-2} = 0$  になってしまう（そもそも逆元は存在しない）



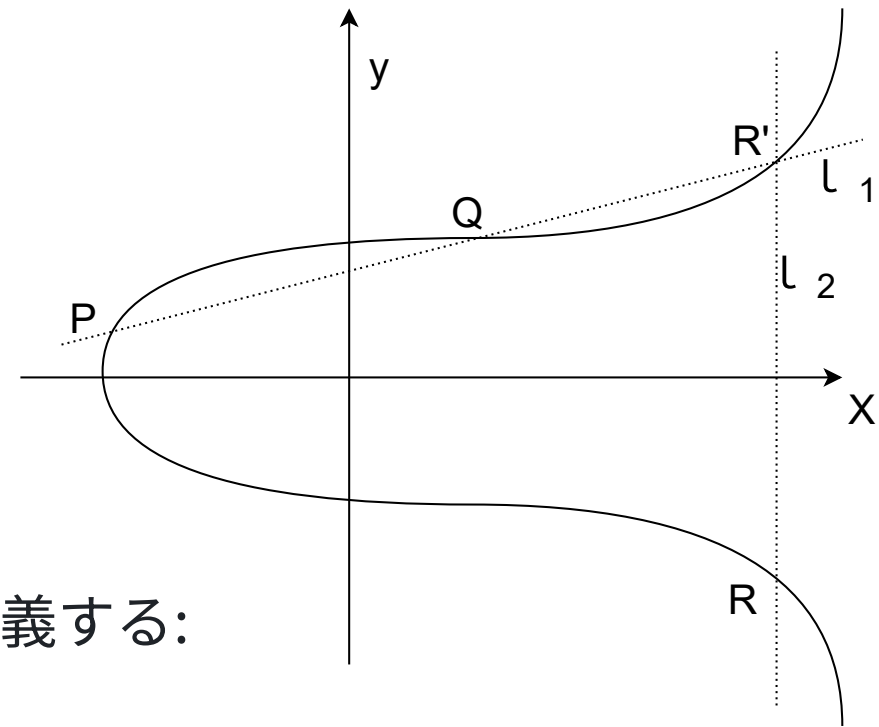
# 楕円曲線の定義

体  $K$  上の (Weierstrass型) 楕円曲線  $E/K$  とは

- $E(K) := \{(x, y) \in K^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$ 
  - 条件:  $a, b \in K$  で  $4a^3 + 27b^2 \neq 0$  ( $x^3 + ax + b = 0$  が重解を持たない)

## 実数上のグラフ

- $E$  の定義方程式  $y^2 = x^3 + ax + b$  のグラフと  $O$  の集合
- $x$  軸に対して対象
- $O$  をシンボリックに無限遠点とする ( $(0, 0)$  ではない)



## 楕円曲線に加算を定義する

- 一般の  $P, Q \in E(K)$  について  $P + Q \in E(K)$  を次で定義する:  
  $P$  と  $Q$  を通る直線  $l_1$  と  $E(K)$  の交点  $R'$  をとり,  
  $R'$  の  $y$  座標の符号を反転させた点を  $R' := P + Q$  とする

# 定義概要

## 特別な点や関係のとき

- $P + O := O + P := P$  for  $\forall P \in E(K)$ ,  $-O := O$
- $P = (x, y)$  について  $-P := (x, -y)$
- $P = Q$  のとき  $P$  における接線を考える
- $Q = -P$  のとき直線は  $y$  軸に並行となり  $P + (-P) := O$  とする

## $G := E(K)$ が群であることの確認

- 単位元と逆元は定義から自明
- 結合則:  $(P + Q) + R = P + (Q + R)$  for  $\forall P, Q, R \in E(K)$ 
  - グラフのデモ: [desmos](#)
  - 実はこれを証明するのはとても大変（普通は代数幾何や複素解析の知識を利用）
  - 初等的な証明: K. Nuida, [An Elementary Linear-Algebraic Proof without Computer-Aided Arguments for the Group Law on Elliptic Curves](#), 2020

# 加法公式の導出

$P = (x_1, y_1), Q = (x_2, y_2), R = (x_3, y_3)$  とする

- 直線  $l_1 : PQ$  の傾き  $\lambda := (y_2 - y_1)/(x_2 - x_1)$  ( $x_1 \neq x_2$  のとき)
- $y = \lambda(x - x_1) + y_1$  を  $y^2 = x^3 + ax + b$  に代入整理
- $x^3 - \lambda^2 x^2 + (a + 2\lambda(\lambda x_1 - y_1))x + b - (\lambda x_1 - y_1)^2 = 0$
- 3次方程式の解と係数の関係:  $x_1 + x_2 + x_3 = \lambda^2$  より  $x_3 = \lambda^2 - x_1 - x_2$
- これを  $l_1$  に代入して  $R'$  の  $y$  座標が求まるので  $y_3 = -\lambda(x_3 - x_1) - y_1$

$x_1 = x_2$  のとき

- $P = Q$  なら  $P$  における接線  $l_1$  の傾きは定義方程式を微分して  $2yy' = 3x^2 + a$  より  $\lambda = (3x_1^2 + a)/(2y_1)$ . 後は上記と同様
- $P \neq Q$  なら  $P + Q = R = O$  と定義した

# 楕円曲線の演算の定義まとめ

## 単位元

- 任意の  $P \in E(K)$  に対して  $P + O = O + P = P$

## 逆元

- $-O := O$   
任意の  $P := (x, y) \in E(K) \setminus \{O\}$  に対して  $-P := (x, -y)$ ,  $P + (-P) = O$

## 加法

- $O$  でない任意の  $P := (x_1, y_1)$ ,  $Q := (x_2, y_2) \neq -P$  に対して  
 $x_3 := \lambda^2 - (x_1 + x_2)$ ,  $y_3 := -\lambda(x_3 - x_1) - y_1$  とすると  $P + Q = (x_3, y_3)$   
ただし

$$\lambda := \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & (x_1 \neq x_2) \\ \frac{3x_1^2 + a}{2y_1} & (x_1 = x_2) \end{cases}$$

## 楕円曲線の群の演算の定義

- $x_1 = x_2$  かつ  $y_1 = 0$  のときはどうなるか

## mod $p$ のDH鍵共有

- $p$  を大きな素数,  $P \in [1, p - 1]$  をランダムに固定・共有し  
   $a \in [1, p - 1]$  をランダムに選び  $aP$  を相手に送る方式のDH鍵共有は安全か

# 能動的な攻撃者

## ECDHで鍵共有を行い共通鍵暗号で暗号化

- 受動的な攻撃者（盗聴者）に対しては安全
- 能動的な攻撃者（改竄者）に対しては安全ではない

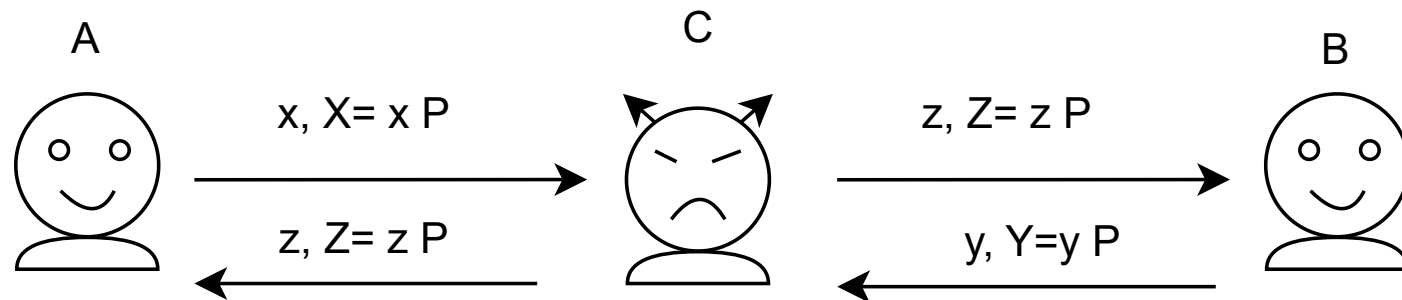
## 敵対的中間者攻撃 AitM (Adversary in the Middle) 攻撃

- A と B の間に改竄者が入り込んで盗聴・改竄を行う攻撃
- 以前は MitM (Man in the Middle) 攻撃
- フィッシング詐欺などのアプリケーションレベルの攻撃も含まれる

# ECDH鍵共有へのAitM攻撃

## A と B の間に改竄者 C が入り込む

- A は  $x$  を選び  $X = xP$  を B に送ったつもり
  - C は  $X$  を盗聴し、代わりに  $z$  を選び  $Z = zP$  を B に送る
  - B は A から  $X$  を受け取ったつもりだが実は  $Z$  を受け取っている
- B は  $y$  を選び  $Y = yP$  を A に送ったつもり
  - C は  $Y$  を盗聴し、代わりに  $Z = zP$  を A に送る
  - A は B から  $Y$  を受け取ったつもりだが実は  $Z$  を受け取っている



- A は  $s_A = xZ = xzP$ , B は  $s_B = yZ = yzP$  を秘密鍵として使う
- C は  $s_A = zX, s_B = zY$  の両方を取得できる

## 署名とは

- ペンや判子などの物理的な署名をデジタル上で実現する暗号技術
  - デジタル署名とも

## 物理的な署名との違い

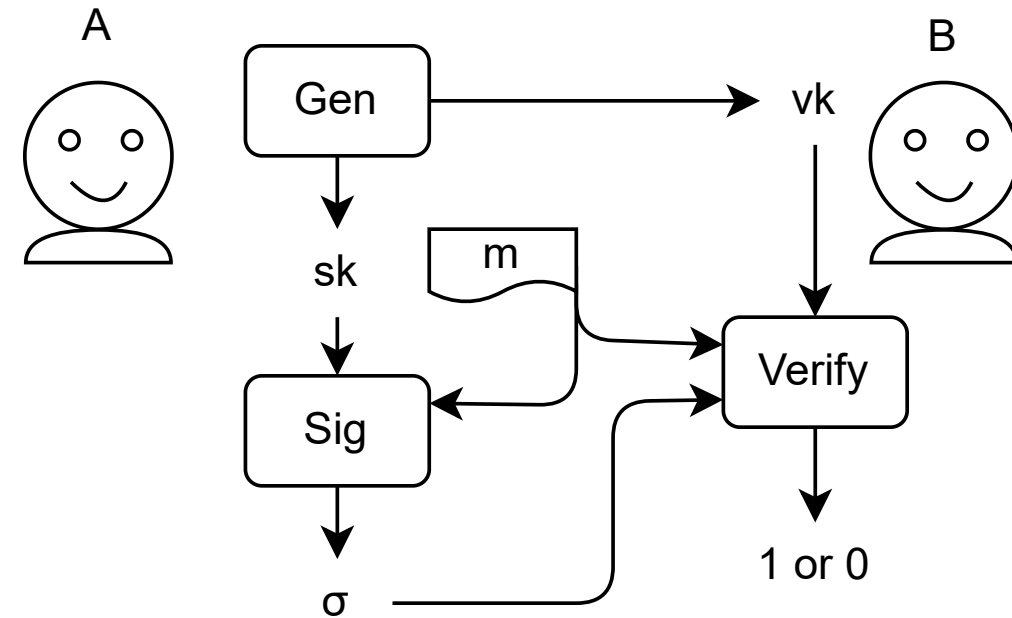
- コンピュータ上ではデジタルな契約書は簡単にコピーできてしまう
- それだけでは本物か偽物か区別ができない
- 署名者だけが作成できる仕組みを実現する必要性



# 署名の定義

## 次の三つ組のアルゴリズム

- 鍵生成  $1^\lambda$  を入力とし署名鍵  $sk$  と検証鍵  $vk$  を出力
  - $Gen(1^\lambda) = (sk, vk)$
  - メッセージ空間  $\mathcal{M}$  も決まる
- 署名生成  $sk$  と  $m \in \mathcal{M}$  から署名  $\sigma$  を出力
  - $Sig(sk, m) = \sigma$
  - 暗に  $vk$  を入力することもある
- 検証  $(vk, m, \sigma)$  を入力し正当なら1, 不正なら0を出力
  - $Ver(vk, m, \sigma) \in \{0, 1\}$



## 用語

- 署名鍵  $sk$  は秘密にするので秘密鍵, 検証鍵  $vk$  は他人に見せるので公開鍵ともいう
- 署名アルゴリズム・署名生成・生成された署名を全部「署名」ということが多い

# 署名の性質

## 正当性

- $Ver(vk, m, Sig(sk, m)) = 1$ .
  - 正しい署名鍵を使って署名すれば正しいと判定される

## 偽造できない

- 署名鍵を持っている人しか正しい署名を作れない
- 偽のメッセージに対する署名を作れない
- 否認できない
  - MACは  $sk = vk$  だったのでAとBの両方が正当なMAC値を生成できる
    - 互いに「相手がMAC値を作った」と主張できる
  - 署名はAしか署名できないのでAは署名したことを否認できない
  - MACの上位互換
    - ただし一般的にMACの方が高速なので用途に応じて使い分ける

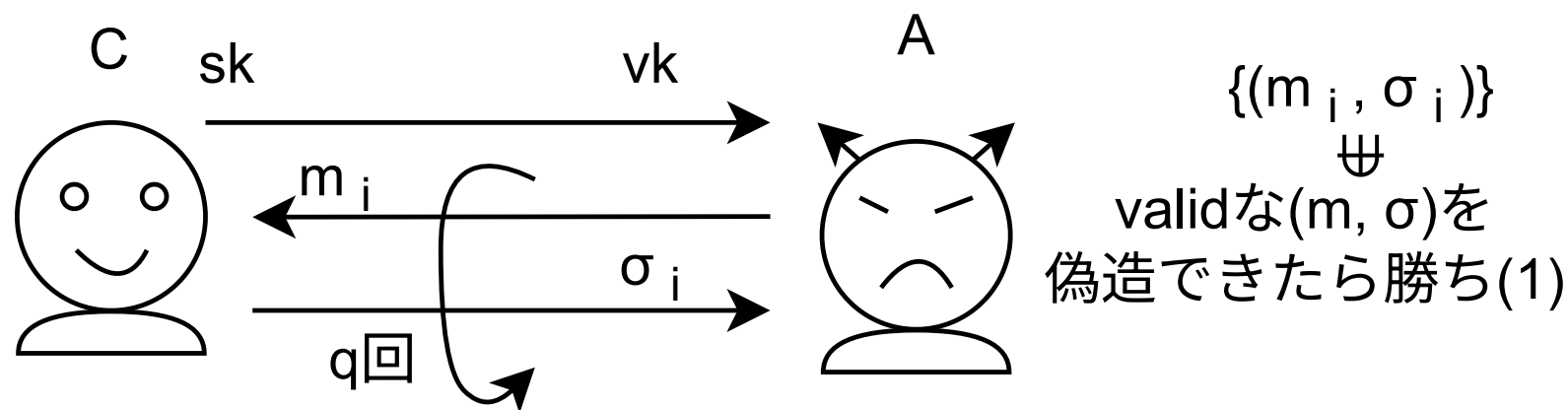
# 署名の強存在的偽造不可能性 sEUF-CMA

## 実験 $Exp_{\mathcal{A}}(\lambda)$

- 挑戦者  $\mathcal{C}$  は  $Gen(1^\lambda) = (sk, vk)$  を実行し  $vk$  を攻撃者  $\mathcal{A}$  に渡す
- $\mathcal{A}$  は  $q$  回  $m_i$  を選び  $\mathcal{C}$  に依頼して  $\sigma_i = Sig(sk, m_i)$  を得る
- $Ver(vk, m, \sigma) = 1$  なる  $(m, \sigma) \notin \{(m_i, \sigma_i)\}_{i=1}^q$  を作れたら1, それ以外は0を出力

## 安全性の定義

- どんなPPT攻撃者  $\mathcal{A}$  に対しても優位性  $Adv_{\mathcal{A}}(\lambda) := Pr[Exp_{\mathcal{A}}(\lambda) = 1] < negl(\lambda)$  のとき sEUF-CMA安全という



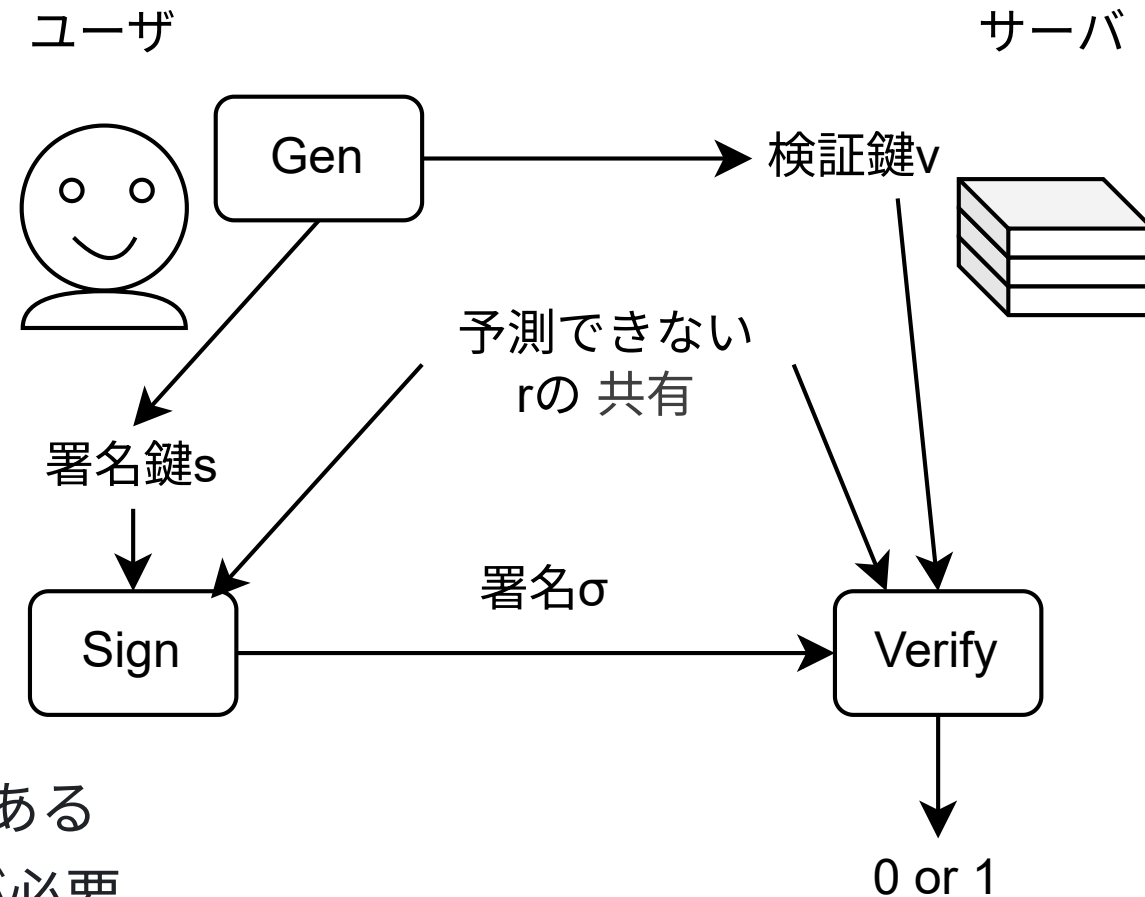
# sshの公開鍵認証

## 署名を用いて本人確認する仕組み

- $Gen$  で署名鍵  $s$  と検証鍵  $v$  を生成
  - $v$  をサーバに登録する
- 認証
  - ECDH鍵共有等を元に予測できない  $r$  を生成
  - $\sigma = Sign(s, r)$  をサーバに送信
  - サーバは  $Verify(v, r, \sigma)$  を確認

## 注意

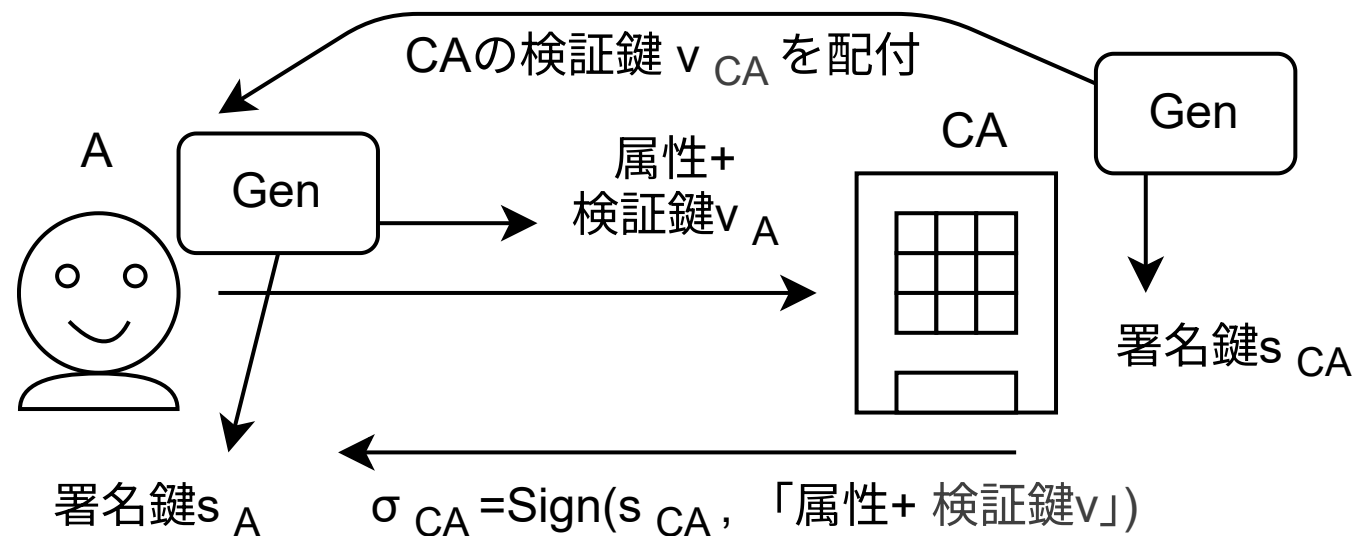
- 検証鍵をサーバに登録するのは慎重にする必要がある
  - 送信には能動的攻撃者に対しても安全な経路が必要
- $r$  は毎回異なる値でなければならない
  - リプレイ攻撃（盗聴者が通信を盗聴して再利用する）を防ぐため



# 認証局 CA (Certification Authority)

## 本人確認しその属性と検証鍵（公開鍵）との対応を保証する機関

- CAは鍵生成して署名鍵  $s_{CA}$  は厳重管理. 検証鍵  $v_{CA}$  を広く公開
- ユーザ（人とは限らない）  $A$  からの申請により検証鍵  $v_A$  と属性  $ID_A$  を確認
- それらの情報に署名鍵  $s_{CA}$  で署名して  $\sigma_{CA}$  を発行
- ユーザの属性がドメインなどのサーバの情報するときサーバ証明書という
- CAは証明書の発行・管理・検証・失効などを行う



# 公開鍵基盤 PKI (Public Key Infrastructure)

## CAの相互認証

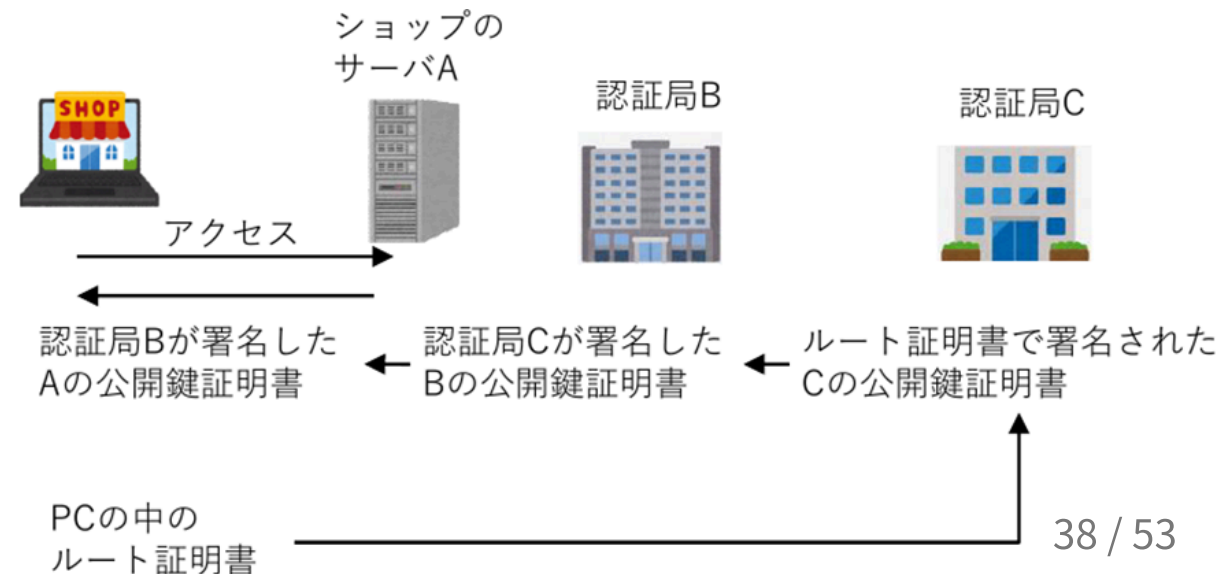
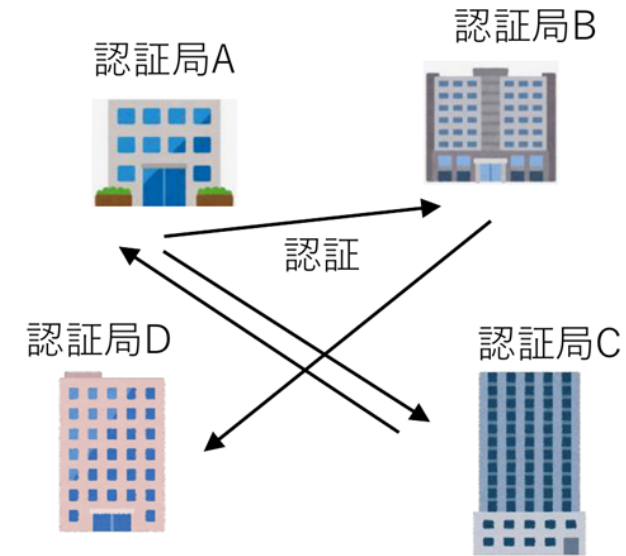
- CAが一つでは権限や責任が集中する
- 複数のCAが相互に認証することで分散化・冗長化を実現

## ルートCA

- 自分で自身を認証したもの
- 信頼の起点（トラストアンカー）

## ルート証明書

- ルートCAが発行する自己署名
- 通常OSやブラウザに組み込まれている



# ECDSA (Elliptic Curve Digital Signature Algorithm)

## 楕円曲線を用いた署名

- $G := \langle P \rangle = \{O, P, \dots, (r-1)P\}$ : 楕円曲線  $E(\mathbb{F}_p)$  の有限巡回群,  $H$ : ハッシュ関数
- $Gen(1^\lambda)$ :  $s \xleftarrow{U} \mathbb{F}_r^*$  を署名鍵,  $S := sP$  を検証鍵とする
- $Sig(s, m)$ :  $k \xleftarrow{U} \mathbb{F}_r^*$  を選び  $R := kP$  の  $x$  座標を  $t$  として ( $t = (kP)_x$  と書くことにする)  
 $\sigma := (t, (H(m) + st)/k \bmod r)$  を署名とする
- $Ver(S, m, \sigma = (t, u))$ :  
 $P' := (H(m)P + tS)/u$  の  $x$  座標が  $t$  ならば1, それ以外は0を出力
  - 分母が0になる場合はやり直す
  - 正当性:  $P' = (H(m)P + tsP)/((H(m) + st)/k) = kP = R$  なので  $t = P'_x$

## 注意

- 署名時の乱数は予測できてはならない
- 2010年PlayStation 3の署名は乱数を固定値にしていたため秘密鍵が漏洩した

# 固定乱数を利用したときのECDSAの危険性

署名  $Sign(s, m_i; k_i) = \sigma_i = (t_i, u_i)$  ( $i = 1, 2$ ) を考える

- $u_1 = (H(m_1) + st_1)/k_1$
- $u_2 = (H(m_2) + st_2)/k_2$ ;  $u_i, H(m_i), t_i$  は既知,  $s$  は未知
  - $k_1, k_2$  が未知でもその差  $k_2 - k_1 = d$  が既知とすると変数2個で方程式2個なので解ける
- $k_1 u_1 - st_1 = H(m_1)$
- $k_1 u_2 - st_2 = H(m_2) - du_2$
- 特に  $k_1 = k_2$  ( $d = 0$ ) のとき解ける. 乱数が単調増加 (e.g.,  $d = 1$ ) していても解ける

乱数の代わりに決定的アルゴリズムを用いる手法 [RFC-6979](#)

- $k$  を秘密鍵  $s$  とメッセージのハッシュ値  $h = H(m)$  から決定的に生成する
- $V := 0x0101\dots01, K := 0x0000\dots00$
- $K = HMAC_K(V || 0x00 || s || h), V = HMAC_K(V)$
- $K = HMAC_K(V || 0x01 || s || h), V = HMAC_K(V), \dots$  (省略) により  $k$  を出力



# 楕円曲線の実装の話

## スカラー倍算 $nP$ の計算

- $nP = P + \dots + P$  ( $n$  個の  $P$  の和) を逐次的に計算すると  $O(n)$  かかる
- $n \approx r \approx 2^{256}$  なら不可能

## バイナリ法

- $n$  を2進数で表現する  $n = \sum_{i=0}^{L-1} n_i 2^i$ . ( $n_i \in \{0, 1\}$ ,  $L \approx \log_2 r \approx 256$ )
- $2^i P$  ( $i = 0, \dots, L-1$ ) を計算する (2倍していくだけなので  $L-1$  回でOK)
- $nP = \sum_{i=0}^{L-1} n_i 2^i P$  なので  $n_i = 1$  となる  $i$  だけ  $2^i P$  を加算する

## 演算コスト

- 2倍算(D:  $2P$ ):  $L-1$  回
- 加算(A:  $P + Q$ ):  $n$  はランダムなので平均的に半分ぐらいが  $n_i = 1$ . よって  $L/2$  回
- 合計  $(L-1)D + (L/2)A$

# ウィンドウ法

## バイナリ法の改善

- $w$  bitずつまとめて処理する
- 事前に  $P, 2P, 3P, \dots, (2^w - 1)P$  を計算しておく
- $n$  を  $2^w$  進数展開する
$$n = \sum_{i=0}^{\lceil L/w \rceil - 1} n_i (2^w)^i \quad (n_i \in [0, 2^w - 1])$$
- $Q = 0$  を初期値として  $Q = 2^w Q + n_i P$  を繰り返す
  - $n_i P$  はテーブル参照

## 演算コスト

- 2倍算:  $L - 1$  回 (バイナリ法と同じ)
- 加算:  $(2^w - 1) + \lceil L/w \rceil$  回
  - $L = 256, w = 4$  ならウィンドウの方がコストが小さい

# 射影座標

## 有限体の除算を避けることによる高速化

- $(x, y)$  の形 (アフィン座標) では演算のたびに  $\lambda$  の計算で除算が必要
  - 除算を避けたい: 分数  $a/b + c/d = (ad + bc)/(bd)$  を分子・分母のペアとして表現する
  - $x = X/Z, y = Y/Z$  として  $[X : Y : Z]$  の形 (射影座標) で表現する
  - $X' = cX, Y' = cY, Z' = cZ$  ( $c \neq 0$ ) のときも  $x = X'/Z', y = Y'/Z'$  なので同じ点
  - $[0 : 0 : 0]$  は除外する
- 方程式  $y^2 = x^3 + ax + b$  は  $Y^2Z = X^3 + aXZ^2 + bZ^3$  の形(D)

## 無限遠点

- アフィン座標では  $O$  という無限遠点が別途表現する必要があった
- $[X : Y : Z] = [0 : 1 : 0]$  は(D)を満たすのでこれを無限遠点として統一的に扱う

# 射影座標による加法公式

$P = [X_1 : Y_1 : Z_1], Q = [X_2 : Y_2 : Z_2]$  とする

- $x_1 \neq x_2$  とする  
$$\lambda = (y_2 - y_1)/(x_2 - x_1) = (Y_2 Z_1 - Y_1 Z_2)/(X_2 Z_1 - X_1 Z_2) =: S/T$$
- $X_3 = \lambda^2 - x_1 - x_2 = (Y_2 Z_1 - Y_1 Z_2)^2 / (X_2 Z_1 - X_1 Z_2)^2 - X_1/Z_1 - X_2/Z_2$   
$$X_3 = S^2/T^2 - (X_1/Z_1 + X_2/Z_2) = (S^2 Z_1 Z_2 - (X_1 Z_2 + X_2 Z_1)) / (T^2 Z_1 Z_2)$$
  
 $Y_3$  も同様に計算できる
- アフィン座標に比べて乗算回数は多少増えるが除算が不要になるので高速化できる
- ECDHなどを射影座標のまま計算し, 最後に一度だけアフィン座標に変換する

# サイドチャネル攻撃

## 暗号装置が動作中の挙動を観察して解読する攻撃

- 電流・時間・音声・電磁波などを利用
- 電力解析: 消費電力の変化から内部状態を推測
- タイミング攻撃: 処理時間の違いから内部状態を推測

## 楕円曲線のスカラー倍算

- バイナリ法は加算するしないは秘密鍵のビットに対応 (情報が漏洩する可能性)

## EUCLEAK攻撃 (2024)

- YubicoのYubiKey 5/FIDO2デバイスへの攻撃報告
- InfineonライブラリのECDSAの実装に問題
  - 逆数計算のための拡張Euclid互除法への攻撃
  - 攻撃条件が高いため直ちに脅威にはならない

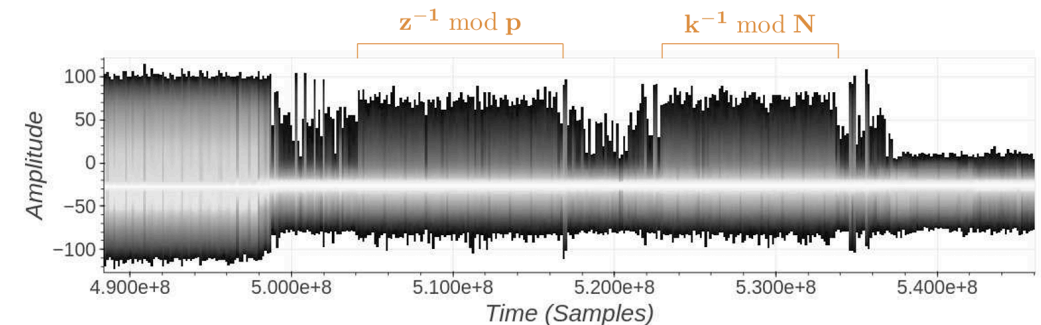
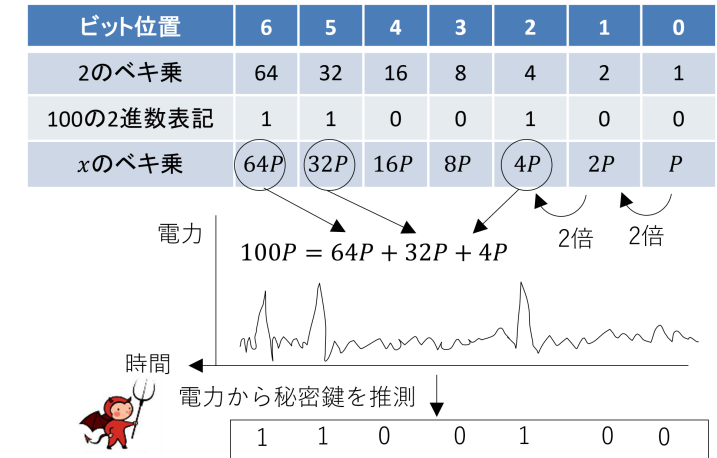


Figure 1.8: Feitian A22 JavaCard - EM Traces - ECDSA Signature -  $(r, s)$  Computation

## ed25519曲線

- twisted Edwards型楕円曲線
  - $-x^2 + y^2 = 1 + dx^2y^2, p = 2^{255} - 19, d = -121665/121666 \in \mathbb{F}_p$
  - $O = (0, 1), -P = (-x, y)$

## 完全加法公式

- $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$  に対して  $P_3 = (x_3, y_3) = P_1 + P_2$  は
$$x_3 = (x_1y_2 + y_1x_2)/(1 + dx_1x_2y_1y_2),$$
$$y_3 = (y_1y_2 + x_1x_2)/(1 - dx_1x_2y_1y_2)$$
  - $P_1 + (0, 1) = (x_1, y_1), P_1 + (-P_1) = (0, 1)$  が成り立つことを確認できる
  - Wierstrass型と違って場合分けが不要

# Curve25519曲線

## Montgomery型楕円曲線

- $v^2 = u^3 + Au^2 + u, p = 2^{255} - 19, A = 486662$
- $O$  は無限遠点,  $P = (u, v)$  に対して  $-P = (u, -v)$
- ed25519とCurve25519は互いに変換できる
  - $u = (1 + y)/(1 - y), v = (\sqrt{A + 2})(u/x)$  とするとed25519型になる
  - $d = -(A - 2)/(A + 2)$

## 加法公式

- Weierstrass型と同様に場合分けが必要
- しかしスカラー倍算は  $x$  座標だけで計算できる方法がある (Montgomery ladder)
  - ECDH鍵共有では  $x$  座標だけ利用するので効率がよい

# RSA (Rivest, Shamir, Adleman) の準備

## 歴史

- 1977年公開. ただし1970年代初頭にCocksたちが先に考えていた (CESGの発表)

## RSA関数の生成

- RSAの落とし戸つき一方向性関数 (trapdoor one-way function):  $f_e$ 
  - $p, q$  を素数,  $n := pq$ , 整数  $a$  に対して  $f_a(x) := x^a \bmod n$  とする
  - $d, e \in [1, n - 1]$  を  $de \equiv 1 \pmod{(p - 1)(q - 1)}$  となる整数とする
    - $e = 65537$  とすることが多い
  - $f_e(x) = x^e \bmod n$  をRSAの落とし戸つき一方向性関数という
  - $(n, e)$  を公開情報,  $(p, q, d)$  を秘密情報とする

## $f_e$ の性質

- $\forall x \in [0, n - 1]$  に対して  $f_d(f_e(x)) = f_e(f_d(x)) = x$ . すなわち  $f_d(x) = f_e^{-1}(x)$
- $f_e : [0, n - 1] \rightarrow [0, n - 1]$  は一方向性置換 (全単射) でもある



# RSA関数が全単射であること

$x^{ed} \equiv x \pmod{n}$  を示す

- Fermatの小定理より  $p, q$  のどちらでも割れない整数  $x$  に対して  
 $x^{p-1} - 1$  は  $p$  で割れ,  $x^{q-1} - 1$  は  $q$  で割れる
  - よって  $x^{(p-1)(q-1)} - 1 = (x^{p-1})^{q-1} - 1 = (x^{q-1})^{p-1} - 1$  は  $p$  でも  $q$  でも割れる
  - $p$  と  $q$  は互いに素なので  $x^{(p-1)(q-1)} - 1$  は  $n = pq$  で割れる:  $x^{(p-1)(q-1)} \equiv 1 \pmod{n}$
  - $de \equiv 1 \pmod{(p-1)(q-1)}$  より, ある整数  $a$  を使って  $de = 1 + a(p-1)(q-1)$  とすると  $x^{ed} = x^{1+a(p-1)(q-1)} = x(x^{(p-1)(q-1)})^a \equiv x \pmod{n}$
- $x$  が  $p$  で割れるとき  $x^{ed} \equiv 0 \equiv x \pmod{p}$ 
  - $x$  が  $q$  でも割れるならそれは  $x = 0$  なので  $x^{ed} = 0 \equiv x \pmod{n}$
  - $x$  が  $q$  で割れないとすると上記と同様に  $x^{ed} \equiv x \pmod{q}$  なので  $x^{ed} \equiv x \pmod{n}$

## 素因数分解 IF (Integer Factoring)

- $n = pq$  が与えられたとき  $p, q$  を求める問題
- 2025年現在, 一般数体篩法 GNFS (General Number Field Sieve) が最良の手法 (DLPと同じ)
- 計算コストは  $L_n[1/3, (64/9)^{1/3}]$  で
  - 例えば  $n = 2^{2048}$  で  $2^{116}$ ,  $n = 2^{3000}$  で  $2^{137}$  程度

## RSA問題

- $n, e, y := f_e(x)$  が与えられたとき  $x$  を求める問題
- RSA仮定: 十分大きな  $p, q$  に対してRSA問題はIFと同等に難しい

## $n$ から $p, q$ が求まれば

- $(p-1)(q-1)$  が求まるので拡張Euclidの互除法で  $e$  の逆元  $d$  が求まる
- IFが解ければRSA問題も解ける / 逆は未解決問題

## 広く使われているRSAを使った署名の一つ **RFC-8017**

- encode関数:  $encode(m) := 0x00 | 0x01 | 0xf \dots f | 0x00 | T | H(m)$ 
  - $H(m)$  の先頭に固定値を連結する.  $T$  は固定バイト列
- $Gen(1^\lambda)$ : RSA関数を生成し  $d$  を署名鍵,  $(n, e)$  を検証鍵とする
- $Sign(d, m)$ :
  - $m' := encode(m)$  として  $\sigma := f_d(m') = m'^d \bmod n$  を署名とする
  - $\sigma = H(m)^d \bmod n$  ではない
- $Ver((n, e), m, \sigma)$ :
  - $m'' := f_e(\sigma) = \sigma^e \bmod n$  と  $m' = encode(m)$  を求めて  $m'' = m'$  なら1, それ以外は0を返す

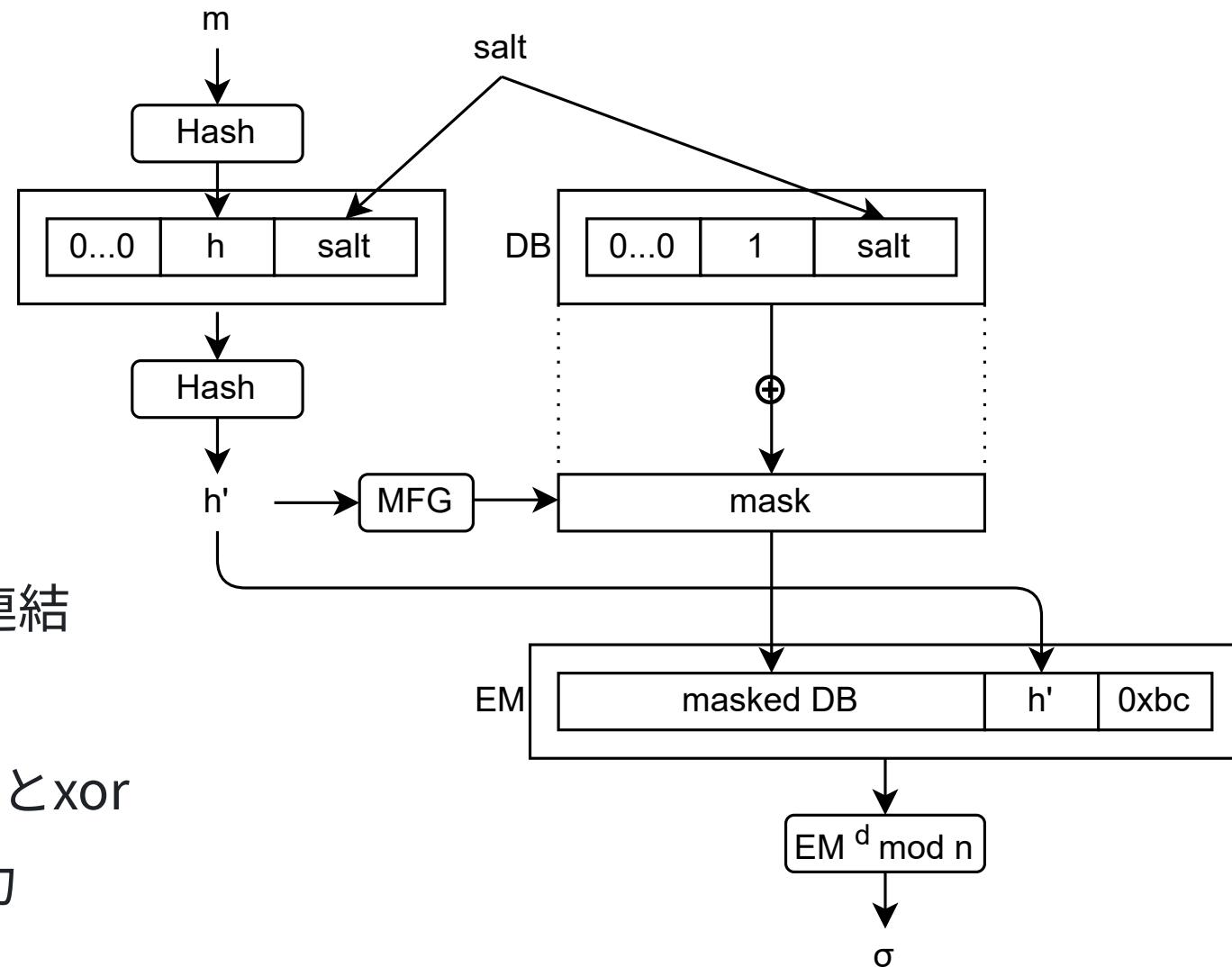
## 正当性の確認

- $m'' = f_e(f_d(m')) = m'$  が成り立つ

# RSASSA-PSS (Probabilistic Signature Scheme)

## より安全なRSAを使った署名方式

- saltを入力可能にすることで同じ  $m$  でも異なる署名を生成できる
- MGF (Mask Generation Function): PRF
  - $\text{count}:=0, T:=\text{" "}$
  - $T := T | \text{Hash}(\text{seed} | \text{count}), \text{count}++$
- 署名
  - $m$  から  $h = H(m)$  を求めて salt と連結
  - もう一度HashしてMGFでmaskを生成
  - maskとsaltを連結してDBを作りmaskとxor
  - それからEMを作りRSA関数で  $\sigma$  を出力



# RSASSA-PSSの検証

$Ver(e, m, \sigma)$

- $EM = \sigma^d \bmod n$  を求め  
masked DBと $h'$ を取り出す
- $h'$  からMFGでmaskを生成し  
masked DBとxorしてmaskを復元
- maskからsaltを取り出し  
 $h'' = H(0..0|h|salt)$  を計算
- $h' = h''$  ならvalid

## 特徴

- saltは途中で復元される
- $\sigma$  だけからでは  $h$  を復元できない
- RSASSA-PKCS1-v1\_5と異なり安全性証明がある

