2023/10/18 ver 1

SecCap 2023
「最新情報セキュリティ理論と応用-補助資料-」
「先進情報セキュリティとアルゴリズム-補助資料-」

宮地 充子

大阪大学大学院

北陸先端科学技術大学院大学

Email: ta-seccap-22@crypto-cybersec.comm.eng.osaka-u.ac.jp

概 要

本講義では，暗号の基本原理，手法, 安全性証明，計算効率の習得を通じて，情報セキュリティを支える数理的な考え方及びその応用手法について理解することを目的とします．さらにブロックチェーンや携帯端末等を含めて，次世代の公開鍵暗号として脚光を浴びている楕円曲線暗号についても解説します．

本講義は板書形式です．板書の速度が頭で理解するのに最適な時間だからです．なお，板書内容は基本的に頭に記憶することが必須です．講義内容は講義と演習の両方の組み合わせで初めて定着しますので，演習をしっかり行うようにしてください．演習は技術計算ソフトウェア Python を用いて実装します．

本書では，講義後の課題を列挙します．課題は問，演習，解析の 3 種類からなります．全ての課題は本講義用に開発された学習システム moodle を用いて提出，返却されます．なお，講義の応用で知っていてほしい内容については，特問，特演習，特解析としています．できる人は挑戦してください．

https://seccap.comm.eng.osaka-u.ac.jp/

- 事前問 (Problem) と事前演習 (Implementation): 講義の理解を深めるために，事前に実施します．moodle で回答を入力するようにしてください．
- 問 (Problem): 講義の理解を深めます．ノートに解答を記載，あるいは Latex で記載して，pdf に変換して提出．
- 演習 (Implementation): 実装により，理論と実装の間のギャップを理解します．Python を用いて Python に標準搭載の関数も利用しながら作成．関数の作成は必ず，本テキスト *N* 章掲載の API と同じ関数を定義してください．入力変数，出力変数，利用するライブラリも記載していますので，確認しましょう．演算速度の測定は指定された回数 (10^4 回等）実施し，平均値を求めるようにして下さい．提出物は，Python のソースファイルと演習で要求された実行結果のファイルです．なお，正しく動いているか確認するために，実行結果は moodle で解答を入力できるようになっている演習もあります．この場合，実行結果が正しいことを確認してから，ソースのファイルを提出してください．ソースファイルと実行結果のファイルを 1 つのディレクトリに保存して，zip で圧縮し，以下のファイル名で提出して下さい．
- 解析 (Experiment): 実験結果と解析のペアで構成．実験結果後，統計処理などをする際に問を利用します．実験データは excel 等を用いて電子的に作成，実験結果は自分なりに解析する．提出物は Python のソース，実験データのエクセルファイル，実験結果を含む解析レポートとなります．これらを 1 つのディレクトリに保存して，zip で圧縮し，以下のファイル名で提出して下さい．解析は人により異なります．

大学名頭文字_名前_演習番号 (大学名頭文字：JAIST → J, NAIST → N, 阪大 → OE, 阪大 pro-sec → OP, 京大 → K, 例: JAIST の宮地が第一回の問を提出する場合: J_miyaji_P1 例: JAIST の宮地が演習 1.1 を提出する場合: J_miyaji_I1.1

実装課題の中で利用する数値例は O 章に，実装する関数 API は N 章記載の API に沿ってください．標準搭載関数で利用する関数も N 章に記載されています．なお，作成した関数は P 章記載のテストデータと比較して動作確認を行うとともに，本講義用で構築された moodle システムにある演習検証を実行してください．実装課題はこれらの動作確認を終わらせてから提出してください．

全講義を終えて，現代暗号理論の基本技術が習得できるように講義は構成されています．途中，挫けずに進めてください．（最後になると簡単になります）ではインターネットの安心・安全を支えるセキュリティ技術を理論，演習，実装の中で習得していきましょう．

教科書：“代数学から学ぶ暗号理論”（日本評論社） 2,3,7-9,13 章
講義日程及び時間: 10/13, 11/10, 12/1, 12/22, 1/12 (18:30-20:00)

1. 講義 1. 暗号基盤理論（離散数学）の紹介． (教科書 2, 3 章)
   修得知識：知識単位 （群，環，体，有限体，有限体上の演算，位数，指数，Lagrange の定理，べき乗演算）
   実装アルゴリズム （公開鍵暗号・ディジタル署名実装に必須のアルゴリズム）バイナリ法，ユークリッドの互除法，拡張ユークリッドの互除法，素数判定法
2. 講義 2+3. 楕円曲線の紹介． (教科書 4, 5 章)
   修得知識：知識単位 （楕円曲線の性質，座標系，加法公式，加算連鎖）
   実装アルゴリズム （楕円曲線暗号実装に必須のアルゴリズム）楕円曲線の性質，座標系，加法公式，加算連鎖
3. 講義 4. 楕円曲線暗号 (教科書 7 章)
   修得知識：知識単位 （楕円曲線暗号の安全性，特徴，他の公開鍵暗号との違い）
   実装アルゴリズム （楕円曲線暗号）
4. 講義 5. ハイブリッド暗号 (教科書 7 章)
   修得知識：知識単位 （共通鍵暗号の仕組み，安全性，ハイブリッド暗号）
   実装アルゴリズム （RC4 鍵生成，疑似乱数生成）

本講義はセキュリティ・暗号関係の研究者・技術者を目指す方の育成を目標としていますが，セキュリティ・暗号に興味をもつ受講生用の取り組みも設定します．講義は同一ですが，課題については受講生の目的・レベルに応じて取り組んでください．なお，単位取得の条件は basic コースの課題の提出です．

セキュリティスペシャリスト実装コース セキュリティ・暗号関係の技術者を目指す方のコースです．全ての事前課題と全ての演習と演習に付随する解析に取り組みましょう．問については，下記を取り組みましょう．

- 講義 1 問 1.2, 1.3,
- 講義 2 問 2.2, 2.3, 2.4, 2.5,
- 講義 3 問 3.1, 3.2, 3.4
- 講義 4 問 4.1,
- 講義 5 問 5.1, 5.3

セキュリティスペシャリスト理論コース セキュリティ・暗号関係の設計を目指す方のコースです．全ての事前課題と全ての問と問及び実装した演習に付随する解析に取り組みましょう．演習については，下記を取り組みましょう．

- 講義 1 演習 1.2, 1.3 1.4,
- 講義 2 演習 2.1, 2.2,
- 講義 3 演習 3.2, 3.3.
- 講義 4 演習 4.3, 4.4
- 講義 5 演習 5.1, 5.3, 5.5

standard コース セキュリティ・暗号関係の知識習得を目指す方の標準的なコースです．全ての事前課題に加えて，以下を実施します．

- 講義 1 問 1.2, 1.3,
  演習 1.2, 1.3, 1.4,
- 講義 2 問 2.1, 2.3, 2.5,
  演習 2.2,
- 講義 3 問 3.1, 3.2, 3.4
  演習 3.2, 3.3.
- 講義 4 問 4.1, 4.2,
  演習 4.2, 4.3,4.4
- 講義 5 問 5.1,5.3
  演習 5.1, 5.3, 5.5

basic コース セキュリティ・暗号関係の最低限の知識習得を目指す方のコースです．全ての事前課題に加えて，以下を実施します．

- 講義 1 問 1.2
  演習 1.4, (演習 1.4 にフェルマー法を適用して，逆元にも利用します．)
- 講義 2 問 2.3, 2.5,
  演習 2.2,
- 講義 3 問 3.1
  演習 3.3. (加法公式は affine 座標を利用します．)
- 講義 4 問 4.1,
  演習 4.2, 4.4
- 講義 5 問 5.1,5.3
  プログラム演習: 演習 5.1, 5.5

目 次

1 講義 1 暗号基盤理論 (教科書 2, 3, 13 章)

コラム 1

アルゴリズムの計算量の評価は重要です．本講義ではアルゴリズムの評価の方法も考えます．一般にアルゴリズムは同じ操作を繰り返すという構造をもち，このようなアルゴリズムの計算量を評価するには下記の考えが有効です．

1 回の操作に必要な計算量 × 繰り返し回数

計算量の単位に n ビットの法乗算 $M(n)$ を用います．n ビットの法乗算と t ビッドの法乗算の計算量，あるいは，法逆元との換算については，以下の換算式を利用します．
(a)$M(n)$ を n ビットの 1 法乗算の計算量とするとき $M(t) = \left(\frac{t}{n}\right)^2 M(n)$.
(b)$I(n)$ を n ビットの法逆元の計算量とするとき $I(n) = CM(n)$，ここで C の値は計算機環境により異なります．本講義では $C = 11$ とします．
また，入力 a と 入力 b に対して，商 q と余り r を求めるアルゴリズム $a = bq + r$ の計算量を，入力 a の 2 進の長さ $\mathrm{len}(a) = n$ を用いて，$R(n)$ と表します．実際のアルゴリズムでは厳密な評価や比較が難しいこともあります．そのような場合，一つの指標に置き換えて比較します．アルゴリズムの課題を解きながら，計算量の評価・比較の方法も習得しましょう．

Lesson 1

It is important for us to estimate computational complexity of algorithms. In this lecture, we also learn how to estimate the computational complexity of algorithm. Generally, an algorithm has a structure to repeat one procedure. We can evaluate the complexity of such an algorithm in the below.

computational complexity of one procedure × number of repeats

We also use an n-bit modulo multiplication, $M(n)$, to estimate a computational complexity. A relation between a t-bit modulo multiplication and an n-bit modulo multiplication, or between an n-bit modulo inversion $I(n)$ and an n-bit modulo multiplication are defined as follows:
(a)$M(t) = \left(\frac{t}{n}\right)^2 M(n)$.
(b)$I(n) = CM(n)$, where c is set to 11 usually.
We also need a computation for given n-bit input a and m-bit input b to output a quationt q and a residue r, which is denoted by $R(n)$. In a real algorithm, it is not easy to evalutate the algorithm. In that case, we may estimate the algorithm with the above characteristic. Let's learn how to estimate the computational complexity of algorithm.

1.1 群・環・体 (教科書 2.4-2.6 章，3.8 章)

例 1 $\mathbb{Z}/3\mathbb{Z}$ における和と積.

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

例 2 $\mathbb{Z}/4\mathbb{Z}$ における和と積.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

問 1.1 剰余環 $\mathbb{Z}/m\mathbb{Z}$ において以下を示せ.

$$\mathbb{Z}/m\mathbb{Z} \text{ が体になる.} \iff m \text{ が素数}$$

問 1.2 $\mathbb{Z}/43\mathbb{Z}$ の乗法群 $U(\mathbb{Z}/43\mathbb{Z})$ 上で位数 7 となる元を求めるアルゴリズムを次のステップを参考にして答えよ，また，そのアルゴリズムの計算量をステップ数で評価する.
(0) 42 を素因数分解せよ.
(1) $U(\mathbb{Z}/43\mathbb{Z}) \ni \forall a$ が位数 *7* になる確率を求めよ.
(2) $U(\mathbb{Z}/43\mathbb{Z}) \ni \forall a$ が位数 *14* になる確率を求めよ.

(3) $U(\mathbb{Z}/43\mathbb{Z}) \ni a$ が位数 *14* の時，a から位数 *7* の元を求める方法を述べよ．
(4) $U(\mathbb{Z}/43\mathbb{Z}) \ni \forall a$ が位数 *21* になる確率を求めよ．
(5) $U(\mathbb{Z}/43\mathbb{Z}) \ni a$ が位数 *21* の時，a から位数 *7* の元を求める方法を述べよ．
(6) $U(\mathbb{Z}/43\mathbb{Z}) \ni \forall a$ が位数 *42* になる確率を求めよ．
(7) $U(\mathbb{Z}/43\mathbb{Z}) \ni a$ が位数 *42* の時，a から位数 *7* の元を求める方法を述べよ．
(8) $U(\mathbb{Z}/43\mathbb{Z}) \ni \forall a$ が位数 *6* になる確率を求めよ．
(9) $U(\mathbb{Z}/43\mathbb{Z}) \ni a$ が位数 *6* の時，a^k は任意の $\mathbb{Z} \ni k > 0$ に対して位数 *7* にならないことを証明せよ．
(10) (1)-(9) を用いて，位数 *7* の元を求めるアルゴリズムを記載し，位数 *7* の元を求めるアルゴリズムの計算量を乗算回数の期待値で評価せよ．

1.2 剰余環上の除算 (教科書 2.6, 3.3 章)

アルゴリズム 1 (ユークリッドの互除法)
入力：整数 a，$b(a \geq b)$
出力：$\gcd(a, b) = d$
(1) $a_0 = a$，$a_1 = b$，$i = 1$ とする．
(2) $a_i = 0$ ならば，$d = a_{i-1}$ を出力し，終了．
(3) $a_i \neq 0$ のとき，$a_{i-1} = a_i q_i + a_{i+1}$，$0 \leq a_{i+1} < |a_i|$，$i = i + 1$ とし，(2) へ．

事前演習 1 バイナリ表記された *2* 元に対して，ユークリッドの互除法を実装し，最大公約数を求めよ．
(1) $gcd(111101, 101)$
(2) $gcd(111101, 111011)$
(3) $gcd(111011001, 110111101)$
(4) $gcd(111011001, 111011)$

演習 1.1 ユークリッドの互除法を実装し，次の数の最大公約数を求めよ．
(1) $gcd(1234567, 234578)$
(2) $gcd(11111111, 22222222)$
(3) $gcd(p_1 - 1, q_1 - 1)$ *(表 O.9)*
(4) $gcd(p_2 - 1, q_2 - 1)$ *(表 O.10)*
(5) $gcd(p_3 - 1, q_3 - 1)$ *(表 O.11)*

アルゴリズム 2 (拡張ユークリッドの互除法)
入力：整数 a と $b(a > b)$
出力：$\gcd(a, b) = d$ なる d と $ax + by = d$ なる整数 x，y
(1) $a_0 = a$，$a_1 = b$ とする．
(2) $x_0 = 1$，$x_1 = 0$ とする．
(3) $y_0 = 0$，$y_1 = 1$ とする．
(4) $i = 1$ とする．
(5) $a_i = 0$ ならば，$d = a_{i-1}$，$x = x_{i-1}$，$y = y_{i-1}$ を出力し，終了．
(6) $a_{i-1} = a_i q_i + a_{i+1}$，$0 \leq a_{i+1} < a_i$ により，a_{i+1} と q_i を定める．
(7) $x_{i+1} = x_{i-1} - q_i x_i$ とする．
(8) $y_{i+1} = y_{i-1} - q_i y_i$ とする．
(9) $i = i + 1$ として，(5) へ．

演習 1.2 拡張ユークリッドの互除法を実装して，以下の数に対して，$ax + by = 1$ となる整数 x, y を求めよ．
(1) $a = 23$, $b = 17$
(2) $a = 13$, $b = 8$

演習 1.3 *[有限体上の逆元]* 拡張ユークリッドの互除法を応用して，以下の $\mathbb{F}_p$ 上の逆元を求めよ．
(1) 表 *O.3* にある p_1, g_1 を用いて $g_1^{-1} \pmod{p_1}$ を求めよ．
(2) 表 *O.3* にある p_1, g_3 を用いて $g_3^{-1} \pmod{p_1}$ を求めよ．
(3) 上記 *2* つの元の逆元を求めるアルゴリズムの計算量を比較する．第一回のコラムの記述に沿って評価する．つまり，*1* 回の操作に必要な計算量×繰り返し回数 で評価する．繰り返し回数は実験的に求め，計算量については $R(n)$ を用いて評価する．

特演習 1 実行回数は 10^4 回．拡張ユークリッドの互除法を応用して，以下の $\mathbb{F}_p$ 上の逆元を求めよ．
(1) 表 *O.3* にある p_1 上の逆元の演算時間を考えるために，*1024* ビットの数 g を 10^4 個生成しよう．この問題では，*1024* ビット目（最上位ビット）は *1* とする．後の演習でも利用するので，$\texttt{Table1}_g$ 保管すること．
(2) (1) で生成した 10^4 個の $g \in \texttt{Table1}_g$ を用いて，$g^{-1} \pmod{p_1}$ を求めて，平均時間と標準偏差を求めよ．
(3) ステップ数 *(ループ (5)-(8) の実行回数)* をカウントして求めて，ステップ数の平均回数と標準偏差を求めよ．アルゴリズムの理論的計算量をステップ数で換算してみよう．厳密には，各ステップ i 毎の $R(len(a_i))$ の n_i の大きさの減少率も考慮して，理論的計算量のステップ数の評価に用いる．

(4) 表 *O.3* にある p_1 上の逆元の演算時間を考えるために，*512* ビットの数 g を 10^4 個生成しよう．この問題では，*512* ビット目（最上位ビット）は *1* とする．後の演習でも利用するので，$\texttt{Table2}_g$ 保管すること
(5) (4) で生成した 10^4 個の $g \in \texttt{Table2}_g$ を用いて，$g^{-1} \pmod{p_1}$ を求めて，平均時間と標準偏差を求めよ．
(6) ステップ数 *(*ループの実行回数*)* をカウントして求めて，平均回数と標準偏差を求めよ．アルゴリズムの理論的計算量をステップ数で換算してみよう．厳密には，各ステップ i 毎の $R(len(a_{i-1}))$ の $len(a_{i-1})$ の減少率も考慮して，理論的計算量のステップ数の評価に用いる．

特解析 1 特演習 *1* の結果から以下の観点について解析してみよう．利用した表はベースポイント g の大きさ[1] が異なる．なお解析の際に利用した演習の結果を明記するとともに，実測値，理論的計算量の観点から議論すること．
(1) 特演習 *1* のように素数が同じでベースポイントのビット長が異なるとする．ベースポイントのビット長と逆元を求める計算量の関係を求めよ．*256* ビット*, 768* ビットの数 g を 10^4 個とり，同様の実験をしてみよう．平均値と分散を求めて，その関係を考えてみよう．この問題では，*256, 768* ビット目（最上位ビット）はそれぞれ *1* とする．
(2) 最大の計算量となるベースポイントの条件を予測してみよう．

1.3 べき乗演算 (教科書 13.3 章)

基本的に断りのない限り，各ビットに 1 が立つ確率は 1/2 とする．また k が n ビットの数とは，$k = \sum_{i=0}^{n-1} k_i 2^i$ $(k_i \in \{0,1\})$ を意味し，セキュリティの分野では仮に $k_{n-1} = 0$ であっても，n ビットと呼ぶ．

アルゴリズム 3 (法 p 上のバイナリ法 1) $\texttt{ModBinary1}(k, y, p)$
入力： 正整数 $k = \sum_{i=0}^{n-1} k_i 2^i$ $(k_i = 0,1)$ と g, p
出力： $y = g^k \bmod p$
$\texttt{ModBinary1}(k, g, p)$
1. $y = 1$.
2. for $i = n-1, \cdots 0$, do {
   if $k_i = 1$, then $y = \texttt{Mod}(\texttt{Mod}(y^2, p) \cdot g, p)$.
   else $y = \texttt{Mod}(y^2, p)$ }
3. Output y

問 1.3 *(1)* バイナリ法において何回の乗算 *(*法乗算*)* と *2* 乗算 *(*法 *2* 乗算*)*[2] が必要になるか求めよ．ただし各ビットに *1* が立つ確率は *1/2* とする．

1. $k = 15$ の場合
2. 160-*bit* 有限体 $\mathbb{F}_p$ 上のランダムな 160-*bit* k を用いる場合．
3. 1024-*bit* 有限体 $\mathbb{F}_p$ 上のランダムな 160-*bit* k を用いる場合．
4. 1024-*bit* 有限体 $\mathbb{F}_p$ 上のランダムな 1024-*bit* k を用いる場合．

(2) 上記の *2-4* の場合の計算量を乗算と *2* 乗算の計算量で評価し比較せよ．但し，以下を仮定する．

- 160 ビット*:* 1024 ビット = 1 : 6
- n ビットの乗算の計算量 *:* mn ビット の乗算の計算量 = $1 : m^2$
- 160 ビットの乗算の計算量の単位を M_{160} として評価
- 加算，減算の計算量は無視
- 乗算の計算量 M *: 2* 乗算の計算量 $S = 1 : 0.8$*,* 乗算の計算量 M *:* 逆元の計算量 $I = 1 : 11$*,*

演習 1.4 バイナリ法を実装し，以下の $\mathbb{F}_p$ 上のべき乗演算の結果を求めよ．
(1) 表 *O.3* にある p_1, g_1, k_1 を用いて $g_1^{k_1} \pmod{p_1}$ を求めよ．
(2) 表 *O.5* にある p_3, g_3, k_3 を用いて $g_3^{k_3} \pmod{p_3}$ を求めよ．
(3) 表 *O.1* にある p_4, g_4, k_4 を用いて $g_4^{k_4} \pmod{p_4}$ を求めよ．
(4) 上記 *3* つの実行速度を測定し，問 *1.3* の理論値と比較せよ．*(*10^4 回実施し，平均値と標準偏差を求める．*)*

事前演習 2 次の *2* 元に対して，$a^{b-2} \pmod{b}$ を求めよ．ここで，b は素数である．
(1) $(a, b) = (101, 111101)$
(2) $(a, b) = (111011, 111101)$
(3) $(a, b) = (1101111011, 111011001)$
(4) $(a, b) = (111011, 111011001)$

[1] 大きさとは一般にビットサイズである．

[2] n-bit の有限体 $\mathbb{F}_p$ 上の乗算 (法乗算) を実装する際には，n-bit の乗算 (出力結果は $2n$ bits) と $2n$-bit の整数 p を法にした値を求める法演算，すなわち，法乗算が必要である．なお，楕円曲線上の加法公式では実行速度の観点から，乗算の度に法演算を行うとは限らない．このため，楕円曲線上の加法公式では乗算と法演算を分けて考える．また有限体上の逆元 y とは，$\mathbb{F}_p$ の元 x に対して，$xy \equiv 1 \pmod{p}$ かつ $\mathbb{F}_p \ni y$ を求めることを意味する．

1.4 拡張べき乗演算 (教科書 13.3 章)(オプション)

この章はオプションなので，興味のある人は実施してください．2 つの元のべき乗演算 $g_1^{k_1} g_2^{k_2} \pmod{p_1}$ を効率よく求める方法を考えよう．複数回計算する同じ乗算を 1 度の実行で再利用することで，演算回数が削減できる．

特問 1 *2* つの元のべき乗演算の積 $g_1^k g_2^h \pmod{p_1}$ を求めるアルゴリズムと計算量について考える．このアルゴリズムは離散対数問題に基づく署名方式などでは必須の演算となる．計算量はこれまで通り乗算の回数で考える．ここでは，$\mathbb{F}_{p_1}$ 上の乗算 *(*法乗算*)* の回数でカウントし，*2* 乗算 *(*法 *2* 乗算*)* の計算量は乗算と同じとする．
(1)$g_1^k \pmod{p_1}$ と $g_2^h \pmod{p_1}$ を独立に求めて，その積から $g_1^k g_2^h \pmod{p_1}$ を求める計算量を求めよ．
(2)$g_1^k \pmod{p_1}$ と $g_2^h \pmod{p_1}$ を独立に求めるのではなく，$k = \sum_{i=0}^{n-1} k_i 2^i$ と $h = \sum_{i=0}^{n-1} h_i 2^i$ の *2* 進展開を同時にスキャンして求める方法がある．これを拡張バイナリ法とよぶ．以下のステップで，拡張バイナリ法のアルゴリズムを構成せよ．

1. テーブル作成：ループで利用する元を求める．
2. ループ：$i = n-1$ *to 0* で (k_i, h_i) を同時にスキャンし，*2* 乗算と乗算の繰り返しでべき乗演算を求める．

(3) 上記拡張バイナリ法を用いて *2* つの元のべき乗演算の積を求める計算量を求めよ．計算量はテーブルを構成する計算量とテーブルを用いたべき乗演算の計算量の総和である．

特演習 2 拡張バイナリ法を実装し，以下の $\mathbb{F}_p$ 上のべき乗演算の結果を求めよ．
(1) 表 *O.3* にある p_1, g_1, k_1, g_2, k_2 を用いて $g_1^{k_1} g_2^{k_2} \pmod{p_1}$ を求めよ．

拡張バイナリ法の主要なアイデアは，再利用できる乗算をテーブルとして事前に作成し，計算量を下げることにある．同様の考え方で，1 つのべき演算の高速化手法が色々提案されている．乗算の再利用をどのように実現するのか? ここでは，拡張バイナリ法を 1 つのべき乗演算に適用して，計算量を減らすことを考えよう．

特問 2 *(1)* 拡張バイナリ法を用いて，べき乗演算アルゴリズム *(*入力 *g,k* に対し，$g^k \pmod{p}$ を出力*)* を記述せよ．k のビット長を n とする．簡単のために，$n = 2m$ とし，下記のステップで行うとする．

1. べき乗 $k = \sum_{i=0}^{n-1} k_i 2^i$ を *2* つに分割する[3]．$h_1 = \sum_{i=0}^{m-1} k_i 2^i$, $2^m h_2 = \sum_{i=m}^{n-1} k_i 2^i = 2^m \sum_{i=0}^{n-1-m} k_{i+m} 2^i$
2. $g^k = g^{h_1} g_1^{h_2} \pmod{p}$ とし，具体的に g_1 を求める．
3. $g^k = g^{h_1} g_1^{h_2} \pmod{p}$ に拡張バイナリ法を適用する．*(*テーブル作成とループの *2* つのステップからなる．*)*

(2) 上記アルゴリズムを用いたべき乗演算アルゴリズム（ステップ *2*，*3*）の計算量を乗算回数，*2* 乗算の回数で評価せよ．ステップ *3* では拡張バイナリ法を適用する際に作成するテーブルの作成に掛かる計算量とループの計算量についても分けて評価しよう．

1.5 Fermat の小定理 (教科書 2.4-2.6 章，教科書 3.1 章, 3.8 章)

ここでは，Fermat の小定理を紹介し，暗号への応用について考える．

定理 1 (フェルマ (Fermat) の小定理) p を素数，r を p と互いに素な整数とするとき，

$$r^{p-1} \equiv 1 \pmod{p}$$

が成り立つ．

Fermart の小定理を応用することで，逆元を求めることもできる．つまり， $a^{p-1} = a \cdot a^{p-2} \equiv 1 \pmod{p}$ を利用する．

特演習 3 *Fermart* の小定理を応用して，以下の $\mathbb{F}_p$ 上の逆元を求めよ．
(1) 表 *O.3* にある p_1 上の逆元の演算時間を考えるために，特演習 *1* で生成した 10^4 個の *1024* ビットの数 $g \in \texttt{Table1}_g$ を用いて，$g^{-1} \pmod{p_1}$ を求めて，平均時間と標準偏差を求めよ．
(2) ステップ数 *(*ループの実行回数*)* をカウントして求めて，平均回数と標準偏差を求めよ．さらに，各ステップ毎に必要となる計算量を $M(1024)$ を単位として求めよ．次に，*1* 回の操作に必要な計算量 × 繰り返し回数 でアルゴリズムの理論的計算量を換算してみよう．
(3) 表 *O.3* にある p_1 上の逆元の演算時間を考えるために，特演習 *1* で生成した 10^4 個の *512* ビットの数 g を用いて，$g^{-1} \pmod{p_1}$ を求めて，平均時間と標準偏差を求めよ．
(4) ステップ数 *(*ループの実行回数*)* をカウントして求めて，平均回数と標準偏差を求めよ．さらに，各ステップ毎に必要となる計算量を $M(1024)$ を単位として求めよ．次に，*1* 回の操作に必要な計算量 × 繰り返し回数 でアルゴリズムの理論的計算量を換算してみよう．

特解析 2 演習 *3* の結果から以下の観点について解析してみよう．利用した表は素数 p およびベースポイント g の大きさが異なる．なお解析の際に利用した演習の結果を明記するとともに，実測値，理論的計算量の観点から議論すること．
(1) 素数の大きさが同じでベースポイントの大きさが異なるときに，逆元を求める計算量の関係はどのようになるか．
(2) 特解析 *1* と比較してみよう．

[3] このステップは考え方であり実際的な計算量には影響を与えない．つまり，整数値の入力 k に対して，その数のビット表現は自然に計算機の中に格納されてあり，その値は入手可能である．

2　講義 2 楕円曲線 1 (教科書 4，5 章)

コラム 2

楕円曲線は整数論の分野で研究されている分野です．研究の対象としての重要性のみならず，数多くの数学の定理を実感できるとても興味深い分野です．講義は 2 回に分けて行います．今日は 3 次曲線である楕円曲線にどのように加法, 写像，準同型写像を定義するのか学習し，楕円曲線の魅力を見ていきしょう．

Lesson 2

An elliptic curve is a field studied in the number theory. It is very important for not only research but also educational objects. Because an elliptic curve includes many important mathematical facts. We have two lectures in elliptic curves. The first lecture includes how to define addtion over elliptic curve, then, understands a rational map and a homomorphism map (isogeny). We hope that everybody are impressed by a mistery of elliptic curves.

2.1　楕円曲線の性質

楕円曲線について簡単に述べる．$\boldsymbol{K}$ を標数 5 以上の体とする．K 上の楕円曲線 $E/\boldsymbol{K}$ とは $a, b \in \boldsymbol{K}$(体) に対して，以下の **Weierstrass 標準形**

$$\boldsymbol{E} : y^2 = x^3 + ax + b \tag{1}$$

で与えられる非特異な 3 次曲線と無限遠点 $\boldsymbol{O} = (\infty, \infty)$ のことである．ここで，$D = 4a^3 + 27b^2$ を**判別式**といい，非特異とは $D \neq 0$ であることを意味し，判別式が 0 であるとは，**特異点**をもつことを意味する．標数が 2 または 3 の場合の楕円曲線の標準形は異なる．無限遠点は，(1) において $x \to \infty$ のとき $y \to \infty$ と考えて，$\boldsymbol{E}$ の点と考える．また，

$$j = \frac{4 \cdot 1728a^3}{D}$$

を j**-不変量**という．j-不変量について以下が成り立つ．

定理 2 *(1) 2* つの楕円曲線 $\boldsymbol{E}$ 及び $\boldsymbol{E}'$ が同型である[4].
$\Longleftrightarrow$ *2* つの楕円曲線 $\boldsymbol{E}$ 及び $\boldsymbol{E}'$ が同じ j-不変量をもつ.
(2) $\boldsymbol{K} \ni \forall j_0$ に対して，$\boldsymbol{K}$ 上の楕円曲線で j-不変量 j_0 をもつ楕円曲線 $\boldsymbol{E}_{j_0}$ が存在する．

楕円曲線の $\boldsymbol{K}$-有理点の集合を，

$$\boldsymbol{E}(\boldsymbol{K}) = \{(x, y) \in \boldsymbol{K}^2 \mid y^2 = x^3 + ax + b\} \cup \{\boldsymbol{O}\}$$

とする．楕円曲線のパラメータ a, b を含む体 $\boldsymbol{K}$ を楕円曲線の定義体と呼ぶ.

問 2.1 体 K 上の楕円曲線 E/K から，j-不変量を対応させる写像は定理 2 から全射写像になる．しかし，単射にはならない．単射にならないことを，次の $\mathbb{Q}$ 上の楕円曲線 $E_{1,1} : y^2 = x^3 + x + 1$ と同じ j 不変量をもつ $E_{1,1}$ と異なる楕円曲線　$E_{a,b} : y^2 = x^3 + ax + b$ を構築することで示せ．但し，このような楕円曲線は無数に作れるため，b が最小の正整数となる異なる楕円曲線を答えよ.

2.2　加算公式

楕円曲線 (1) の座標系を **affine 座標系**と呼ぶ．楕円曲線には $\boldsymbol{O}$ が零元になるような加法が定義できる．楕円曲線上の 2 点 $A = (x_1, y_1)$ と $B = (x_2, y_2)$ に対し，$A + B$ を，A と B を結ぶ直線と楕円曲線とのもう一つの交点の x 軸に対称な点と定義する．楕円曲線の加算の利点は，幾何学的に定義された加算が有理式で書き下せる点である．実際，$A \neq B$ に対して，$C = (x_3, y_3) = A + B$ は，以下の式で計算できる．

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2, \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \tag{2}$$

ここで $\lambda = (y_2 - y_1)/(x_2 - x_1)$ である．$A = B$ のとき，$C = (x_3, y_3) = 2A$ は，以下の式で計算できる．

$$\begin{aligned} x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \tag{3}$$

ここで $\lambda = (3x_1^2 + a)/(2y_1)$ である．(2)，(3) から，楕円曲線の係数に依存するのは，2 倍算のみで加算は依存しないことがわかる．

[4]同型写像が $\boldsymbol{K}$ 上定義されるとき，$\boldsymbol{K}$-同型と呼び，このとき $\boldsymbol{E}(\boldsymbol{K}) = \boldsymbol{E}'(\boldsymbol{K})$ となる．

この加算により楕円曲線 $\boldsymbol{E}$ は，$\boldsymbol{O}$ を零元にもつ可換群[5]になる．また $\boldsymbol{E} \ni P = (x_1, y_1)$ に対して，逆元は $-P = (x_1, -y_1)$ で与えられる．特に，$\boldsymbol{E}(\boldsymbol{K})$ もこの加算に関して群になり，$\boldsymbol{K}$ が有限体 $\mathbb{F}_p$ のとき，$\boldsymbol{E}(\mathbb{F}_p)$ は有限可換群になる．本章以降，$n \in \mathbb{Z}$ に対して，

$$nP = \begin{cases} P + \cdots + P(n \text{ 回の加算}) & n > 0 \\ -n(-P) & n < 0 \\ \boldsymbol{O} & n = 0 \end{cases}$$

と表す．加算公式を実行する際の計算量は乗算 M，2 乗算 S と逆元 I の計算回数で評価されることが多い．この際，2 倍や 8 倍などの小さな数との積や加算，減算の計算回数はこれらに比べて小さい計算量で実現できるため，無視してよい．また乗算と 2 乗算は一般に 2 乗算の方が小さい計算量で実現できるためそれらの回数を区別することが多い．

問 2.2 有理数体上のアファイン加法公式の性能を評価する．
(1) アファイン加法公式の 加算と *2* 倍算公式で乗算 *M, 2* 乗算 *S* と逆元 *I* を何回行う必要があるか求めよ．但し，*2,3,4* などの小さい数の積は無視する．
(2) アファイン加法公式を実装するのに必要なメモリ量（変数の個数）を求めよ．途中変数に利用するメモリもカウントしなるべく利用するメモリが小さくなるように公式を記述する．

演習 2.1 有理数体上のアファイン加法公式を実装しよう．実装した楕円曲線の加法公式を以下の例で確かめてみよう．楕円曲線 $\boldsymbol{E}/\mathbb{Q}$ を

$$\boldsymbol{E} : y^2 = x^3 + x + 1$$

とする．$P = (0, 1)$ に対して，以下の各点を求めよ．
(1) $2P, 3P$ を求めよ．（各座標の元は通分，つまり分母分子で共通因子を持たない分数（つまり，有理数）で表示して，分子，分母で答えよ．）
(2) kP $(k = 4, \cdots, 18)$ を求めよ．（各座標の元は通分，つまり分母分子で共通因子を持たない分数（つまり，有理数）で表示して，分子，分母で答えよ．）

解析 2.1 演習 *(2.1)* のべき演算結果において以下の問に答えよ．
(1) kP $(k = 1, \cdots, 18)$ の x, y 座標の分母の素因数を求めよ．
(2) (1) を用いて，k と分母の素因数の関係に特徴的なことを説明せよ．

2.3 写像

楕円曲線間の写像である同種写像を定義し，その性質について述べる．

定義 1 楕円曲線 $\boldsymbol{E}$ から楕円曲線 $\boldsymbol{E}'$ への写像

$$\phi : \boldsymbol{E} \longrightarrow \boldsymbol{E}'$$

が**同種写像** *(isogeny)* であるとは，以下の *2* 条件を満たすことである．
(1) ϕ は**有理写像** *(morphism)*，すなわち，ある有理式 $F(x, y), G(x, y)$ を用いて，$\phi((x, y)) = (F(x, y), G(x, y))$ と表される．
(2) $\phi(\boldsymbol{O}) = \boldsymbol{O}$
同種写像が全単射であるとき**同型写像**とよび，$\boldsymbol{E}$ と $\boldsymbol{E}'$ を**同型**とよぶ．

解析 2.2 体 $\mathbb{Q}$ 上の楕円曲線 $E/\mathbb{Q} : y^2 = x^3 + ax^2 + bx$ $(\mathbb{Z} \ni a, b \neq 0)$ 上の [2] 倍写像を考える．このとき，以下を答えよ．
(1)

$$\begin{aligned} [2] : E &\longrightarrow E \\ P &\longmapsto 2P \\ (x, y) &\longmapsto \left(\frac{f(x, y)}{g(x, y)}, \frac{h(x, y)}{r(x, y)} \right) \end{aligned}$$

が同種写像となることを有理多項式 f, g, h, r を求めることで示せ．
(2) [2] の核 $Ker([2])$ の群の元を求めよ．

2.4 有限体上の楕円曲線

楕円曲線の定義体として有限体 $\mathbb{F}_p$ $(p \geq 5)$ を考える．このとき楕円曲線 $\boldsymbol{E}/\mathbb{F}_p$ 上の楕円曲線の加算については，体 $\boldsymbol{K}$ を有限体 $\mathbb{F}_p$ と置き換えて考えるとよい．つまり，加算公式においては，(2)，(3) の計算を $\mathbb{F}_p$ 上行うことになる．この結果，有限体上の楕円曲線 $\boldsymbol{E}/\mathbb{F}_p$ の $\mathbb{F}_p$-有理点の集合，

$$\boldsymbol{E}(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p^2 \mid y^2 = x^3 + ax + b\} \cup \{\boldsymbol{O}\}$$

は，有限可換群になる．

[5]加群とも呼ぶ

問 2.3 定理 *2* より，$\mathbb{F}_5$ 上には $\mathbb{F}_5 \ni \forall j$ を j-不変量に持つ楕円曲線 $E_{a,b} : y^2 = x^3 + ax + b (a, b \in \mathbb{F}_5)$ が存在する．そこで，$\mathbb{F}_5 \ni \forall j$ に対して，それを j-不変量にもつ楕円曲線の式を各 j に対して構成することを考える．
(1) $j = 0$ については，$\boldsymbol{E} : y^2 = x^3 + 1$ 以外の楕円曲線をすべて挙げよ．
(2) $j = 3$ については，$\boldsymbol{E} : y^2 = x^3 + x$ 以外の楕円曲線をすべて挙げよ．
(3) $j \neq 0, 3$ の時は楕円曲線を以下の式を利用して，$\mathbb{F}_5 \ni j$ に対して j 不変量が j になる楕円曲線 $\boldsymbol{E}/\mathbb{F}_5$ を構築しよう．但し，複数ある場合は，$\frac{3j}{3-j}$ が $\mathbb{F}_5$ で最小となる楕円曲線を答えよ．

$$\begin{aligned} \boldsymbol{E}_{0,1} &: y^2 = x^3 + 1, \\ \boldsymbol{E}_{1,0} &: y^2 = x^3 + x, \\ \boldsymbol{E}_j &: y^2 = x^3 + \tfrac{3j}{3-j}x + \tfrac{2j}{3-j}. \end{aligned}$$

例 3 $\mathbb{F}_5$ 上の楕円曲線

$$\boldsymbol{E} : y^2 = x^3 + 1$$

の元の個数を求めるには，$x = 0, 1, 2, 3, 4$ に対して， *2* 次方程式 $y^2 = x^3 + 1 \pmod 5$ となる y の個数を求めるとよい．*2* 次方程式の根の個数は *0,1,2* である．数学的に言い換えると，$x^3 + 1 \pmod 5$ が *5* を法として平方数（これを平方剰余という）であるかどうか調べることと同じになる．これは以下のようにかける．

x	0	1	2	3	4
$x^3 + 1 \pmod 5$	1	2	4	3	0
y	± 1	−	± 2	−	0

よって，

$$\boldsymbol{E}(\mathbb{F}_5) = \{(x, y) | (0, \pm 1), (2, \pm 2), (4, 0)\} \cup \{\mathcal{O}\}$$

となり，$\#\boldsymbol{E}(\mathbb{F}_5) = 6$ より $tr(\phi) = 6 - 6 = 0$ となる．これは後述の *Hasse* の定理を満たす．また $(2, 2)$ の位数が *6* になることより，群の構造は $\boldsymbol{E}(\mathbb{F}_5) = \langle (2, 2) \rangle \cong \mathbb{Z}/6\mathbb{Z}$ となる．つまり (2.2) で $\boldsymbol{E}(\mathbb{F}_5)$ の元がすべて生成できる．

問 2.4 問 *2.3* で $\boldsymbol{E}/\mathbb{F}_5$ で j-不変量 $j = 0$ となる $\boldsymbol{E} : y^2 = x^3 + 1$ 以外の楕円曲線をすべて列挙した．例 *3* に従って，j-不変量 $j = 0$ となるすべての楕円曲線の元の個数 $\#\boldsymbol{E}_{0,b}(\mathbb{F}_5)$ を求め，$(b, \#\boldsymbol{E}_{0,b}(\mathbb{F}_5))$ を答えよ．

問 2.5 有限体上の楕円曲線の *affine* 加法公式性能を求めよう．
(1) アファイン加法公式の加算と *2* 倍算公式で $|p|$ ビットの乗算 *M, 2* 乗算 *S* と逆元 *I* 及び法演算 *(mod p)* を何回行う必要があるか求めよ．但し，小さい数の積は無視する．
(2) 必要なメモリ量（変数の個数）を求めよ．途中変数に利用するメモリもカウントする．なるべく利用するメモリが小さくなるように公式を記述する．
(3) (2) のメモリで実装できる公式を具体的に記述する．

演習 2.2 問 *2.5* のメモリ量と計算回数で有限体上の楕円曲線 $E/\mathbb{F}_p$ $(p \geq 5)$ のアファイン加法公式を実装しよう．実装した楕円曲線暗号の加法公式を以下の事例で確かめてみよう．$\mathbb{F}_{13}$ 上の楕円曲線 $\boldsymbol{E}_1/\mathbb{F}_{13}$ と $\boldsymbol{E}_2/\mathbb{F}_{13}$ を

$$\begin{aligned} \boldsymbol{E}_1 &: y^2 = x^3 + x + 1, \quad \boldsymbol{E}_1(\mathbb{F}_{13}) \ni P_1 = (0, 1) \\ \boldsymbol{E}_2 &: y^2 = x^3 + x - 1, \quad \boldsymbol{E}_2(\mathbb{F}_{13}) \ni P_2 = (1, 1) \end{aligned}$$

とする．$P_1 = (0, 1)$ 及び $P_2 = (1, 1)$ に対して，アファイン座標の加法公式を用いて，以下の各点を求めよ．
(1) $2P_1, 3P_1$ を求めよ．
(2) kP_1 $(k = 4, 5)$ を求めよ
(3) $2P_2, 3P_2$ を求めよ．
(4) kP_2 $(k = 4, 5)$ を求めよ

3 講義 3 楕円曲線 2 (教科書 4，5 章)

3.1 有限体上の楕円曲線と Hasse の定理

コラム 3

物事をローカルとグローバルに考えることがあります．ここではグローバルとは有理数体 $\mathbb{Q}$ 上で考えることを意味し，ローカルとは有限体 $\mathbb{F}_p$ 上で考えることを意味します．一般にグローバルに成り立つことはローカルで成り立ちますが，その逆は必ずしも成り立ちません．楕円曲線を用いて，数学におけるグローバルとローカルの概念について考えましょう．また，楕円曲線をもちいて，写像，核，有限体の部分群について，学びます．

Lesson 3

We sometimes investigate a fact from the global or local point of view. In this lecture, the rational or finite field corresponds to the global or local object, respectively. Generally, a fact becomes true in the global field, then it becomes true in the local field. However, the opposite may not be true. By using elliptic curves over a global or local field, let us investigate an idea of global and local from the mathematical sense. Furthermore, using elliptic curves, learn about maps, kernel, and subgroups of finite fields.

定理 3 (Hasse の定理) $\boldsymbol{E}/\mathbb{F}_p$ を p 個の元をもつ有限体 $\mathbb{F}_p$ 上の楕円曲線とする．このとき，

$$|tr(\phi) = \#\boldsymbol{E}(\mathbb{F}_p) - p - 1| \leq 2\sqrt{p}$$

である．

例 4 $\mathbb{F}_5$ 上の楕円曲線 $\boldsymbol{E}_1/\mathbb{F}_5$ と $\boldsymbol{E}_4/\mathbb{F}_5$ を

$$\begin{aligned} \boldsymbol{E}_1 &: y^2 = x^3 + 1, \quad \boldsymbol{E}_1(\mathbb{F}_5) \\ \boldsymbol{E}_4 &: y^2 = x^3 + 4, \quad \boldsymbol{E}_4(\mathbb{F}_5) \end{aligned}$$

を考える．$\boldsymbol{E}_1/\mathbb{F}_5$ と $\boldsymbol{E}_4/\mathbb{F}_5$ の j-不変量はともに 0 のため，2 つの楕円曲線は同型になる．同型写像は以下で与えられる．

$$\begin{aligned} \phi : \boldsymbol{E}_1(\mathbb{F}_5) &\longrightarrow \boldsymbol{E}_4(\mathbb{F}_5) \\ (x, y) &\longmapsto (4x, 2y) \end{aligned}$$

ローカルで同型になる楕円曲線だが，グローバル $\mathbb{Q}$ ではどうだろうか? 2 つの楕円曲線はどちらも $\mathbb{Q}$ 上の楕円曲線 $\boldsymbol{E}_1/\mathbb{Q}$ と $\boldsymbol{E}_4/\mathbb{Q}$ と考えることができる．j-不変量はともに 0 なので，同型になり，同型写像は以下で与えられる．

$$\begin{aligned} \psi : \boldsymbol{E}_1 &\longrightarrow \boldsymbol{E}_4 \\ (x, y) &\longmapsto ((\sqrt[3]{2})^2 x, 2y) \end{aligned}$$

ψ は $\mathbb{Q}$ 上定義されないため，$\boldsymbol{E}_1/\mathbb{Q}$ と $\boldsymbol{E}_4/\mathbb{Q}$ は $\mathbb{Q}$ 上同型にはならない．

定義 2 $\mathbb{Q}$ 上の楕円曲線 $\boldsymbol{E}_1/\mathbb{Q}$ のツイストとは，$\mathbb{Q}$ 上の楕円曲線 $\boldsymbol{E}_2/\mathbb{Q}$ で，$\boldsymbol{E}_1$ と $\overline{\mathbb{Q}}$-同型な楕円曲線である．但し， $\mathbb{Q}$ 上同型な楕円曲線は同一視する．（なお，$\overline{\mathbb{Q}}$ は $\mathbb{Q}$ の代数的閉包と呼ばれる体を表す．直感的には $\mathbb{Q}$ の代数的閉体 $\mathbb{C}$ と考えると良い．）

問 3.1 例 4 の 2 つの楕円曲線について，以下の問に答えよ．
(1) $\boldsymbol{E}_1(\mathbb{F}_5) \ni P_1 = (2, 2)$ に対して，$\phi(P_1)$ を求めよ．
(2) $\#\boldsymbol{E}_4(\mathbb{F}_5)$ を例 3 に沿って求めよ．
(3) $2\phi(P_1)$ と $\phi(2P_1)$ を求めて，$\phi(2P_1) = 2\phi(P_1)$ となることを確認せよ．（準同形写像となることの表れである．

特問 3 例 4 の 2 つの楕円曲線について，ϕ が全単射になることを $\phi^{-1} : \boldsymbol{E}_4 \longrightarrow \boldsymbol{E}_1$ を求めることで示せ．

問 3.2 $\mathbb{F}_5$ 上の楕円曲線 $\boldsymbol{E}_1/\mathbb{F}_5$ と $\boldsymbol{E}_2/\mathbb{F}_5$ を

$$\begin{aligned} \boldsymbol{E}_1 &: y^2 = x^3 + x + 1, \quad \boldsymbol{E}_1(\mathbb{F}_5) \ni P_1 = (0, 1) \\ \boldsymbol{E}_2 &: y^2 = x^3 + x - 1, \quad \boldsymbol{E}_2(\mathbb{F}_5) \ni P_2 = (1, 1) \end{aligned}$$

とする．このとき，以下の問いに答えよ．プログラムは利用せずに，手で解いてみましょう．
(1) $\boldsymbol{E}_1/\mathbb{F}_5$ と $\boldsymbol{E}_2/\mathbb{F}_5$ の j-不変量を求めよ．
(2) $\#\boldsymbol{E}_1(\mathbb{F}_5)$ と $\#\boldsymbol{E}_2(\mathbb{F}_5)$ を求めよ．
(3) P_1 と P_2 の位数を求めよ．
(4) $\boldsymbol{E}_1$ から $\boldsymbol{E}_2$ への同型写像を求めよ，

$$\begin{aligned} \psi : \boldsymbol{E}_1 &\longrightarrow \boldsymbol{E}_2 \\ (x, y) &\longmapsto (tx, sy) \end{aligned}$$

(5) （*4*）を用いて $\psi(0, 1)$ を求めよ．

定義 3 写像 $f : G \longrightarrow G'$ が**全射**であるとは，

$$G' \ni \forall y \text{ に対して, } f(x) = y \text{ となる } x \in G \text{ が存在する}$$

写像であり，**単射**であるとは，

$$x_1, x_2 \in G \text{ が } f(x_1) = f(x_2) \Longrightarrow x_1 = x_2 \text{となる}$$

写像である．写像 f が全射かつ単射であるとき，**全単射**であるという．f の**像** *(Image)* Imf を

$$Imf = \{f(a) \mid a \in G\}$$

と表し，f の**核** *(kernel)* $Kerf$ を

$$Kerf = \{a \in G \mid f(a) = 1\}$$

で表す．

特問 4 $\mathbb{F}_5$ 上の楕円曲線 $\boldsymbol{E}_1/\mathbb{F}_5$ と $\boldsymbol{E}_2/\mathbb{F}_5$ と *2* つの同種写像を考える．ここで，$r = a^2 - 4b \neq 0$ である．

$$\boldsymbol{E}_1 \quad : y^2 = x^3 + ax^2 + bx,\ \boldsymbol{E}_2 \quad : y^2 = x^3 - 2ax^2 + rx,$$

$$\begin{aligned} \psi : \boldsymbol{E}_1 \quad &\longrightarrow \boldsymbol{E}_2, \qquad \hat{\psi} : \boldsymbol{E}_2 \quad \longrightarrow \boldsymbol{E}_1 \\ (x, y) \quad &\longmapsto \left(\tfrac{y^2}{x^2}, \tfrac{y(b-x^2)}{x^2}\right), \quad (x, y) \longmapsto \left(\frac{y^2}{4x^2}, \frac{y(r-x^2)}{8x^2}\right) \end{aligned}$$

(1) ψ の核 $ker(\psi)$ を求めよ．具体的に点の x 座標，y 座標を求めよ．
(2) $\hat{\psi}$ の核 $ker(\hat{\psi})$ を求めよ．具体的に点の x 座標，y 座標を求めよ．
(3) $\hat{\psi} \circ \psi$ の写像を求めよ．
(4) $\hat{\psi} \circ \psi$ の写像の核 $ker(\hat{\psi} \circ \psi)$ となる具体的な楕円曲線の点 *(*x 座標，y 座標*)* を求めよ．

3.2 Jacobian 座標系

affine 座標系は加算公式の度に逆元計算が必要になる．逆元計算は乗算に比べて時間がかかるので，逆元計算を利用しない座標系 **Jacobian 座標系**が利用されることがある．Jacobian 座標系の楕円曲線は，(1) を $(x, y) = (X/Z^2, Y/Z^3)$ と変換することにより与えられる．

$$E_J : Y^2 = X^3 + aXZ^4 + bZ^6. \tag{4}$$

Jacobian 座標は射影空間上の点を表すため，厳密には点の同値類を表していることになる．つまり，以下の同値関係を満たす場合 $(Z, Z_1 \neq 0$ のとき$)$，同じ点を表すことになる．

$$(X, Y, Z) \sim (X_1, Y_1, Z_1) \Longleftrightarrow (X/Z^2, Y/Z^3) = (X_1/{Z_1}^2, Y_1/{Z_1}^3)$$

Jacobian 座標系では $\boldsymbol{O} = (1, 1, 0)$ となり，加法公式は，$P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$，$P + Q = R = (X_3, Y_3, Z_3)$ に対して，下記で与えられる．

注意 1 楕円曲線の加算，*2* 倍算の計算量とメモリ量を削減するには，再利用できる演算はメモリに保管して，同じ計算を *2* 回しないこと，また，利用しない値が保管されたメモリは開放し，他の必要な値に利用する．例えば，加算で必要な Z_1^3 は $Z_1^3 = (Z_1^2)Z_1$ とすでに計算した Z_1^2 を再利用するとよい．楕円曲線暗号は加算公式で実現されるので，例えば，乗算総数 $8M$ が $7M$ になると 87.5% の削減になる．また，一般に二乗算は乗算より高速に実現できる．このため，$2Z_1Z_2 = (Z_1 + Z_2)^2 - Z_1^2 - Z_2^2$ を用いることにより乗算の計算を二乗算の計算に置き換え，さらに，既に計算済みの Z_1^2 や Z_2^2 を用いることで，さらなる計算の効率化を図ることもよく行われる．

• 加算公式 ($P \neq \pm Q$**)**

$$X_3 = R^2 - J_1 - 2J_2,\ Y_3 = R(J_2 - X_3) - 2S_1J_1,\ Z_3 = ((Z_1 + Z_2)^2 - Z_1^2 - Z_2^2)H. \tag{5}$$

ここで $U_1 = X_1Z_2^2$, $U_2 = X_2Z_1^2$, $S_1 = Y_1Z_2^3$, $S_2 = Y_2Z_1^3$, $H = U_2 - U_1$, $R = 2(S_2 - S_1)$, $I = (2H)^2$, $J_1 = IH$, $J_2 = IU_1$ である．

• 2 倍算公式 ($R = 2P$**)**

$$X_3 = M^2 - 2S,\ Y_3 = M(S - X_3) - 8Y_1^4,\ Z_3 = (Y_1 + Z_1)^2 - Y_1^2 - Z_1^2 (= 2Y_1Z_1) \tag{6}$$

ここで $S = 2((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4)$, $M = 3X_1^2 + aZ_1^4$ である．

問 3.3 有理数体上の *Jacobian* 加法公式の性能を評価する．
(1)Jacobian 加法公式 の 加算と *2* 倍算公式で乗算 *M, 2* 乗算 *S* と逆元 *I* を何回行う必要があるか求めよ．但し，小さい数の積は無視する．
(2)Jacobian 加法公式を実現するのに必要なメモリ量（変数の個数）を求めよ．途中変数に利用するメモリもカウントする．
(3) (3) のメモリで実装できる公式を具体的に記述する．

演習 3.1 有理数体上の *Jacobian* 加法公式を実装しよう．演習 *2.1* の楕円曲線 $\boldsymbol{E}/\mathbb{Q}$ 及び P を用いて，*Jacobian* 座標系で以下の各点を求めよ．ここで，$P = (0, 1, 1)$ と考えるとよい．
(1) $2P$, $3P$ を求めよ．（各座標の元 X, Y, Z は整数値であるが，$\forall t \in \mathbb{Z}$ に対して，$(X, Y, Z) = (Xt^2, Yt^3, Zt)$ となることから，$|Z|$ が最小となる値 (X, Y, Z) を代表元として，求めよ．*)*

特演習 4 演習 2.1 の楕円曲線 $\boldsymbol{E}/\mathbb{Q}$ 及び P を用いて，Jacobian 座標系で以下の各点を求めよ．ここで，$P=(0,1,1)$ と考えるとよい．
(1) kP $(k=4,5)$ を求めよ．（各座標の元 X,Y,Z は整数値であるが，$|Z|$ が最小となる値 (X,Y,Z) を表示せよ．）
(2) $kP=(X_k,Y_k,Z_k)$ $(k=1,\cdots,5)$ の大きさを座標 X_k,Y_k,Z_k の最大ビットサイズで表せ．
(3) (2) で求めた点を Affine 座標系に変換し，演習 2.1 の結果と一致することを確かめよ．

特解析 3 特演習 *4* の結果から，以下の観点について解析してみよう．
(1) 実験結果を用いて，k の増加率と $kP=(X_k,Y_k,Z_k)$　の増加率の関係を予想しよう．
(2) 加法公式から予想の正しさを議論してみよう，

問 3.4 有限体上の *Jacobian* 加法公式の性能を求めよう．
(1) Jacobian 加法公式 の 加算と *2* 倍算公式で乗算 *M, 2* 乗算 *S* と逆元 *I* 及び法演算 *(mod p)* を何回行う必要があるか求めよ．但し，小さい数の積は無視する．
(2) 必要なメモリ量（変数の個数）を求めよ．途中変数に利用するメモリもカウントする．なるべく利用するメモリが小さくなるように公式を記述する．
(3) (2) のメモリで実装できる公式を具体的に記述する．

解析 3.1 問 *(2.5)* と *(3.4)* の結果を用いて，*affine* と *Jacobian* 加法公式をメモリ量，計算量で比較し，計算機性能に応じてどの座標を選ぶとよいか解析しよう．

有理数体上の加法公式の実装では，アファイン公式の場合は通分の処理，Jacobian 公式の場合は共通因子の処理という，計算結果の最適化の処理が必要になるが，有限体上での演算ではそのような処理が不要になることに注意したい．

演習 3.2 問 *3.4* のメモリ量と計算回数で有限体上の楕円曲線 $E/\mathbb{F}_p$ $(p\geq 5)$ の *Jacobian* 加法公式を実装する．次に，$\mathbb{F}_{13}$ 上の楕円曲線 $\boldsymbol{E}_1/\mathbb{F}_{13}$ と $\boldsymbol{E}_2/\mathbb{F}_{13}$ を

$$\begin{aligned}\boldsymbol{E}_1 &: y^2=x^3+x+1, \quad \boldsymbol{E}_1(\mathbb{F}_{13})\ni P_1=(0,1)\\ \boldsymbol{E}_2 &: y^2=x^3+x-1, \quad \boldsymbol{E}_2(\mathbb{F}_{13})\ni P_2=(1,1)\end{aligned}$$

とする．$P_1=(0,1)$ 及び $P_2=(1,1)$ に対して，$P_1=(0,1,1)$ 及び $P_2=(1,1,1)$ と考えて，*Jacobi* 座標の加法公式を用いて以下の各点を求めよ．
(1) $2P_1, 3P_1$ を求めよ．
(2) kP_1 $(k=4,5)$ を求めよ
(3) $2P_2, 3P_2$ を求めよ．
(4) kP_2 $(k=4,5)$ を求めよ

3.3 楕円曲線上のスカラー演算

楕円曲線暗号の演算の主要部分は楕円曲線の元のスカラー倍算，kG の計算となる．スカラー倍算の計算には，有限体の乗法群 $\mathbb{F}_p^*$ 上で紹介したバイナリ法を楕円曲線上の演算に置き換えたアルゴリズムが利用できる．
次の演習では，NIST[**?**] で標準化され，世界的に広く利用されている楕円曲線を用いて実装してみよう．

演習 3.3 楕円曲線 $\boldsymbol{E}/\mathbb{F}_p$ *(*$p=256$ ビット*)* 上のスカラ倍算の結果を求めよ．$E: y^2=x^3-3x+b$ とし，ベースポイントは $G=(g_x,g_y)$，$n=\#E(\mathbb{F}_p)$ で,kG を求めよ．k,n は *10* 進数で記載，他は *16* 進数で記載した．

$$\begin{aligned}p &= 2^{256}-2^{224}+2^{192}+2^{96}-1.\\ n &= 115792089210356248762697446949407573529996955224135760342422259061068512044369\\ b &= 5ac635d8\ aa3a93e7\ b3ebbd55\ 769886bc\ 651d06b0\ cc53b0f6\ 3bce3c3e\ 27d2604b\\ g_x &= 6b17d1f2\ e12c4247\ f8bce6e5\ 63a440f2\ 77037d81\ 2deb33a0\ f4a13945\ d898c296\\ g_y &= 4fe342e2\ fe1a7f9b\ 8ee7eb4a\ 7c0f9e16\ 2bce3357\ 6b315ece\ cbb64068\ 37bf51f5\\ k &= 172\ 35091\ 96654\ 51459\ 12345\ 17144\ 99640\ 83306\ 22345\ 44321\end{aligned}$$

特問 5 では，安全性のレベルが同じ場合の有限体上のべき演算と楕円曲線上のスカラー倍算の計算量を比較してみよう．

特問 5 *(1)* 256 ビットの有限体 $\mathbb{F}_p$ 上の楕円曲線 E で定義される *ECDLP* と 2048 ビットの有限体 $\mathbb{F}_p$ 上定義される *DLP* の安全性は同じである．このとき，*それぞれ*の暗号の基本演算 *ECDLP kG (k* は 256 *bits*，$G=(x_G,y_G)$ は 256 *bits) DLP* $g^k \pmod p$ *(k* は 256 *bits, p* は 2048 *bits)* の速度の比の理論値を求めよ．但し，以下を仮定する．

- 256 ビット*:* 2048 ビット = 1 : 8
- n ビットの乗算の計算量 *:* mn ビット の乗算の計算量 = $1:m^2$
- 256 ビットの乗算の計算量の単位を M_{256} として評価
- 加算，減算の計算量は無視する．

- 乗算の計算量 M : 2 乗算の計算量 $S = 1 : 0.8$
- 乗算の計算量 M : 除算の計算量 $I = 1 : 11$

1. アファイン座標系を利用する場合
2. *Jacobian* 座標系を利用する場合

(2) アファイン座標系と *Jacobian* 座標系を比較する．*256 bits* の体でアファイン座標系の方が *Jacobian* 座標系より高速になる除算と乗算の計算量の比 I/M を求めよ．

特演習 5 *affine* 座標と *Jacobian* 座標で実装し，演習 *(4.3)* の実行時間を比較する．特問 *5* の結果と比較しよう．

4 講義 4 楕円曲線暗号 (教科書 5，6 章)

コラム 4

楕円曲線暗号は 1986 年に Miller と Koblitz の 2 人の研究者によって独立に提案されました．発表年度は違いますが，それぞれの研究が独立に評価されるのは査読付き論文として投稿していることが大きな要因です．同様に，楕円曲線に基づく ID ベース暗号があります．こちらは笠原先生チームと Boneh-Franklin が独立に提案しましたが，笠原先生チームは国際学会に投稿しておらず，Boneh-Franklin が発明したと思っている人が多いと思います．世界は広く，日本はその一部です．皆様も，アイデアの権利を守り，優先権を主張するにはどうすべきかを心がけてください．

Lesson 4

Elliptic curve cryptosystems were proposed by Koblitz and Miller, independently. Although, those papers have been published in different year, we respect two researchers results. This is because they have exactly published their reviewed paper. On the other hand, ID-based encryption scheme (IBE) was proposed by Kasahara group and Boneh-Franklin. However, many of us thinks that Boneh-Franklin has proposed the IBE since Kasahara group has not published their results as the international conference. Japan is just one country in the world. Let us think how to protect our idea.

4.1 楕円曲線暗号

DLP に基づく方式は，全て**楕円曲線上の離散対数問題** (ECDLP) に基づく方式に変換することができる．変換に関する対応は表 4.1 のようになる．

表 4.1: 有限体上の暗号と楕円曲線暗号の対応

	有限体 $\mathbb{F}_p$ 上の暗号	楕円曲線 $\boldsymbol{E}/\mathbb{F}_p$ 上の暗号
安全性	DLP	ECDLP
利用する群	有限体の乗法群 $\mathbb{F}_p^*$	楕円曲線の有理点の集合 $\boldsymbol{E}(\mathbb{F}_p)$
演算	乗法	加法
ベースポイント	$\mathbb{F}_p^* \ni g$	$\boldsymbol{E}(\mathbb{F}_p) \ni G$
ベースポイントの位数	$\mathrm{ord}(g) = l$	$\mathrm{ord}(G) = l$

次章から具体的に変換して構築された方式について，紹介するが，まずはじめに，ECDLP について定義する．

定義 4 (ECDLP) 有限体上の楕円曲線 $\boldsymbol{E}(\mathbb{F}_p)$ の元 Y, G に対して

$$Y = xG = G + \cdots + G(\ G \text{ の } x \text{ 回の和})$$

となる x が存在するならばその x を求めよ．

楕円曲線上の ElGamal 暗号について記載する．

【ユーザの鍵生成】ユーザ B は，次のように公開鍵と秘密鍵のペアを生成する．
1. B は，楕円曲線 $\boldsymbol{E}/\mathbb{F}_p$ (p は素数) と位数 ℓ のベースポイント $\boldsymbol{G} \in \boldsymbol{E}(\mathbb{F}_p)$ を生成する．ここで位数 ℓ は素数とする．
2. 乱数 $x \in \mathbb{Z}_\ell$ を生成し，これを秘密鍵とし，$\boldsymbol{E}(\mathbb{F}_p)$ 上で

$$Y = x\boldsymbol{G}$$

を計算する．
〈 公開鍵 〉$\boldsymbol{E}/\mathbb{F}_p$, $\boldsymbol{G}$, Y
〈 秘密鍵 〉x

【暗号化】 ユーザ A が平文 $m \in \mathbb{Z}_{|p|}$ を暗号化して B に送るとする．
1. 乱数 $r \in \mathbb{Z}_\ell^*$ を生成し，

$$U = r\boldsymbol{G} = (u_x, u_y) \tag{7}$$

を計算する．
2. B の公開鍵 Y を用いて

$$\begin{aligned} V &= rY = (v_x, v_y) \\ c &= v_x \oplus m, \end{aligned} \tag{8}$$

を計算し，暗号文 (U, c) を B に送信する．ここで，$\oplus$ はビット毎の排他的論理和である．

【復号】 暗号文 (U, c) を受信した B は以下のようにして m を復号する．

$$\begin{aligned} V &= xU = (v_x, v_y) \\ m &= v_x \oplus c. \end{aligned}$$

平文 m を暗号化する際に, (8) では排他的論理和を用いた．オリジナルの論文では，m を楕円曲線の元 $M \in \boldsymbol{E}(\mathbb{F}_p)$ に写像して暗号化を行う．すなわち，

$$C = rY + M.$$

しかし，平文 m を楕円曲線の元に対応させるのが非効率であることから，記述の方法が利用されることが多い．

楕円 ElGamal 暗号の暗号化に用いる乱数 r の値は，暗号化のたびに異なる必要がある. 問 4.1 はその解読を表す.

問 4.1 同じ r を異なる暗号化に利用した場合，一組の暗号文と平文のペアから，すべての暗号文が復号できることを示す．つまり，既知平文攻撃で完全解読できる．

演習 4.1 問 *4.1* を用いて，例 *3* で公開鍵 $Y = (4, 0)$, 平文 *2* の暗号文 $\{(4, 0), 6\}$ を入手したとき，暗号文 $\{(4, 0), 7\}$ を復号してみよう．

問 4.2 楕円 *ElGamal* 暗号は，適応的選択暗号文攻撃で完全解読できることを示せ．但し，復号オラクルに聞く暗号文は *1* 回とする．

演習 4.2 小さい例で楕円曲線暗号を実装してみよう．例 *3* の楕円曲線 $E/\mathbb{F}_5$ とベースポイント $G = (2, 2)$ をシステムパラメータとする．
(1) 秘密鍵 $x = 3$ を用いた公開鍵 $Y = xG$ を求めよ．
(2) 平文 $m = 4$ を $r = 3$ で暗号化した結果を求めよ．また復号できることを確かめよ．

有限体の元を楕円曲線に埋め込んで，加法準同型をもつ楕円曲線暗号を構築してみよう．まずは下記の問で，有限体の元を楕円曲線に埋め込む方法を考えよう．

特問 6 加法準同型をもつ楕円曲線暗号を作るには，有限体の元を楕円曲線の点に埋め込む関数を作る必要がある．例 *3* より，$\#\boldsymbol{E}(\mathbb{F}_5) = 6$ であることより，少なくともすべての $\mathbb{F}_5$ の元を異なる点に移すことはできる．有限体 $\mathbb{F}_5 = \{0, 1, 2, 3, 4\}$ の元を楕円曲線 $\boldsymbol{E}(\mathbb{F}_5)$ に埋め込む写像 $\phi : \mathbb{F}_5 \longrightarrow \boldsymbol{E}(\mathbb{F}_5)$ を考えよう．ただし，以下の条件を考慮すること．
(1) 写像は単射であること．
(2) 写像はアルゴリズムとして記載. つまり，$\mathbb{F}_5 = \{0, 1, 2, 3, 4\} \ni x$ に対して，$M_x \in \boldsymbol{E}(\mathbb{F}_5)$ をどのように決定するかを記述する．
(3) どの写像も加法準同型にならないことを証明せよ．

次の演習では，NIST[**?**] で標準化され，世界的に広く利用されている楕円曲線を用いて実装してみよう．

演習 4.3 以下の有限体と楕円曲線を用いる．なお，楕円曲線は $E : y^2 = x^3 - 3x + b$ とし，ベースポイントは $G = (g_x, g_y)$，$n = \#E(\mathbb{F}_p)$ である．n は *10* 進数で記載，他は *16* 進数で記載した．

$$\begin{aligned} p &= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1. \\ n &= 115792089210356248762697446949407573529996955224135760342422259061068512044369 \\ b &= 5ac635d8\ aa3a93e7\ b3ebbd55\ 769886bc\ 651d06b0\ cc53b0f6\ 3bce3c3e\ 27d2604b \\ g_x &= 6b17d1f2\ e12c4247\ f8bce6e5\ 63a440f2\ 77037d81\ 2deb33a0\ f4a13945\ d898c296 \\ g_y &= 4fe342e2\ fe1a7f9b\ 8ee7eb4a\ 7c0f9e16\ 2bce3357\ 6b315ece\ cbb64068\ 37bf51f5 \end{aligned}$$

(0) 秘密鍵 $x = 105792089210356248762697446949407573529996955224135760342422259061068512044368$ を用いた公開鍵を求めよ．
(1) 次の平文 m*(*255 ビット*)* を r *(*255 ビット*)* で暗号化した結果を求めよ．
$r = 39500472352324671173807819617973955890358426519716096396971791569113162974965$

m = 5006421507588603899595423797631392968450266947792064885507177554531680094119 4 (2) 暗号文 (U, c) を復号した結果を求めよ．ここで $U = (u_x, u_y)$

$$
\begin{aligned}
c &= 4f00\ 705dd\ 95006\ 812e0\ 792a8\ 73d6f\ b739f\ 35a17\ b22e2\ 853c1\ bc361\ eb245\ 9e34c \\
u_x &= 5c3ea2171\ 01cc6\ e824a\ c29b0\ 747ff\ 5900e\ 3bfdd\ 0327f\ 37f6f\ b1aed\ 7fd15\ 91786 \\
u_y &= 967ad485\ 4635a\ abb35\ 4f348\ b9942\ 0d540\ 415a9\ a18a8\ 94c32\ dfbe3\ 28a99\ d9cc0
\end{aligned}
$$

演習 4.4 演習 *4.3* の楕円曲線を用いて以下を実施しよう．
(1) 自分の秘密鍵を生成し，その公開鍵を求めて，公開鍵掲示板に掲載する．
(2) 掲示板に掲載された担当の *TA* の公開鍵を用いて，自分の好きな食堂のランチメニューを暗号化して提出しよう．担当は以下の通りである．大阪 *ProA, NAIST,* 阪大工は *Kaiming,* 大阪 *ProB, JAISTC* は *Mathieu*，京大は奥村，大阪宮地研 *A*，*JAISTB* は加藤，大阪宮地研 *B*，*JAISTA* は新井，なお，データは *16* 進に変換して提出し，*256* ビットで暗号化できる文字数[6]に注意すること．*TA* は正しく復号できれば掲示板に *OK* を入れよう．

5 講義 5 データ秘匿と完全性 (教科書 7 章)

コラム 5

オンラインショッピング時には，店舗に対して，ユーザのデータは暗号化されて送付され，店舗は受信した内容が改ざんされていないことが検証できます．これを実現するのが TLS です．これまで学習した公開鍵暗号，ディジタル署名，共通鍵暗号を用いて，TLS は実装されます．この TLS を支える仕組みが公開鍵証明書であり，公開鍵証明書は公開鍵に対するセンターの署名となります．安全性の攻撃モデルで選択平文攻撃を学びました．公開鍵証明書は機械が自動的に署名を生成するため，選択平文攻撃の脅威にさらされます．本講義では，ディジタル署名を学習後，データ秘匿とディジタル署名の組み合わせによるハイブリッド暗号について学習します．

Lesson 5

Such a function is achieved by using TLS. TLS can be constructed by using a public key encryption, a digital signature, and a secret key encryption that you have learned in this lecture. The mechanism that supports this TLS is a public key certificate, which is a signature on the public key. One attack model is the chosen message attack. Since public key certificates are automatically signed by a machine, they are vulnerable to chosen message attacks. In this lecture, we see how to generate a signature, a combination of encryption and signature, which achives a hybrid encryption.

5.1 楕円曲線署名

楕円曲線上の DSA 署名について記載する．

【ユーザの鍵生成】 ユーザ A は，次のように公開鍵と秘密鍵のペアを生成する．
1. 楕円曲線 $\boldsymbol{E}/\mathbb{F}_p$ (p は素数) と素数位数 ℓ のベースポイント $G \in \boldsymbol{E}(\mathbb{F}_p)$ を生成する．
2. 乱数 $x \in \mathbb{Z}_\ell^*$ を生成して秘密鍵とし，$\boldsymbol{E}(\mathbb{F}_p)$ 上で

$$Y = xG$$

を計算する．
3. 一方向性ハッシュ関数

$$H : \{0,1\}^* \to \{0,1\}^{|\ell|-1}$$

を全ユーザに公開する．
⟨ システムパラメータ ⟩ $\boldsymbol{E}/\mathbb{F}_p,\ G,\ \ell, H$
⟨ 公開鍵 ⟩ Y
⟨ 秘密鍵 ⟩ x
【署名生成】 ユーザ A はメッセージ $m \in \{0,1\}^*$ に以下のように署名する．
1. メッセージ m に対して，そのハッシュ値 $m' = H(m)$ を生成する．
2. 乱数 $r \in \mathbb{Z}_\ell^*$ を生成し，

$$
\begin{aligned}
U &= rG = (u_x, u_y) \\
u &= u_x \bmod \ell
\end{aligned}
\tag{9}
$$

[6] 1 アルファベットは 1 バイト，日本語全角 1 文字 (UTF-8) は 3 バイトに相当します．

を計算する．ここで $u = 0$ ならば乱数 r を取り直す．
3. 秘密鍵 x を用いて，

$$v = r^{-1}(m' + xu) \bmod \ell \tag{10}$$

を計算する．ここで $v = 0$ ならば乱数 r を取り直す．
m に対する署名は，$(u, v) \in \mathbb{Z}_\ell^* \times \mathbb{Z}_\ell^*$ である．

【署名検証】 A の m の署名 (u, v) を以下のように検証する．
1. $(u, v) \in \mathbb{Z}_\ell^* \times \mathbb{Z}_\ell^*$ でなければ，署名を拒絶する．
2. 公開鍵 y を用いて，

$$\begin{aligned} m' &= H(m) \\ d &= 1/v \bmod \ell \\ U' &= m'dG + udY = (u'_x, u'_y) \end{aligned}$$

を求める．
3. $u \equiv u'_x \pmod{\ell}$ ならば OK をそうでなければ NG を出力する．

問 5.1 ディジタル署名の重要な応用に公開鍵証明書がある．公開鍵証明書とはユーザの公開鍵が正しいお墨付きを与えるもので，センターがセンターの秘密鍵を用いてユーザの公開鍵にディジタル署名を施すものである．ユーザが楕円曲線暗号を用いる場合，ユーザの公開鍵 Y にセンターが楕円曲線上の *DSA* 署名を施し，これを公開鍵証明書とすることができる．公開鍵証明書はユーザの数だけ必要になるため，そのサイズの削減は重要な要素である．演習 *4.3* の楕円曲線をもちいて，公開鍵を作成する場合，1 ユーザの公開鍵証明書の管理に必要なメモリ量（ビットサイズ）を求めよ．G の位数は n であることを用いる．

演習 5.1 楕円曲線 $E/\mathbb{F}_5 : y^2 = x^3 + 2x + 1$ とベースポイント $G = (1, 2)$ を利用する．なお，ここではハッシュ関数は利用しないで考える．
(1) 秘密鍵 $x = 2$ を用いた公開鍵 $Y = xG$ を求めよ．
(2) (1) の鍵ペアを認証局の公開鍵と秘密鍵とする，このとき，ユーザ A の秘密鍵 $x_a = 3$, 公開鍵 $Y_a = x_a(1, 2)$ に対して，公開鍵証明書を生成してみよう．具体的には公開鍵 Y_a の x 座標に署名を施した結果を求める．ただし，$r = 5$ を用いるとする．

演習 5.2 演習 *4.3* の楕円曲線 $E/\mathbb{F}_p$ とベースポイント G を利用する．なお，ここではハッシュ関数は *shake256* を利用する．
(1) G の位数が n になることを示せ．
(2) 秘密鍵
x = 113 25678 91234 56789 12345 67891 23456 78912 34567 89123
を用いた公開鍵 $Y = xG$ を利用する場合を考える．
次の平文 m(256 ビット) を r (256 ビット) で署名生成した結果を求めよ．また検証できることを確かめよ．
m = 98 52161 57341 97693 18382 97635 53364 23817 19970 77546 10313 71670 81630 93268 24438 65550
r = 66 54081 52789 85542 44937 56268 14020 80961 80036 08689 79896 12658 76446 33693 67349 67749

5.2 OAEP (Optimal Asymmetric Encryption Padding) の基本概念

既存の RSA 暗号，Rabin 暗号，ElGamal 暗号は，適応的選択暗号文攻撃のもとで識別不可能性を満たさない．RSA-OAEP 暗号 [**?**] は適応的選択暗号文攻撃に対して安全に RSA 暗号を変換した方式である．**OAEP**(Optimal Asymmetric Encryption Padding) は RSA 暗号に限らず，他の方式のパディング方法にも利用できるので，ここでは一般的に記載する．

【初期設定】 公開鍵暗号を $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ とし，Enc の定義域のサイズを k バイト，平文空間のサイズを $(k >) k_2$ バイト，0 パディングのサイズを $k_1 (> 0)$ バイトとする．k_0 は $k = k_0 + k_1 + k_2$ を満たすように設定する．またハッシュ関数を公開する．

$$\begin{aligned} G: &\quad \{0,1\}^{8k_0} &&\longrightarrow \{0,1\}^{8(k_2+k_1)} \\ H: &\quad \{0,1\}^{8(k_2+k_1)} &&\longrightarrow \{0,1\}^{8k_0} \end{aligned}$$

【ユーザの鍵生成】 ユーザ B は公開鍵暗号 $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ を用いて，秘密鍵 SK と公開鍵 PK を生成する．

【暗号化】 ユーザ A が平文 $m \in \{0,1\}^{8k_2}$ を暗号化して B に送るとする．
1. 乱数 $r \in \{0,1\}^{8k_0}$ に対して，

$$\begin{aligned} s &= G(r) \oplus m \,\|\, (0^{8k_1}) \\ t &= H(s) \oplus r \\ w &= s \,\|\, t \\ \mathsf{cipher} &= \mathsf{Enc}(PK, w) \end{aligned} \tag{11}$$

を計算する．ここで，$m \,\|\, (0^{8k_1}), s \,\|\, t$ はそれぞれ $0^{8k_1}, t$ が下位ビットである．
2. cipher を暗号文として B に送信する．

【復号】
1. B は自分の秘密鍵を用いて[7],

$$\begin{aligned} w &= \mathsf{Dec}(SK, \mathsf{cipher}) \\ s &= [w]_{k_0}^{k-1} \\ t &= [w]_0^{k_0-1} \\ r &= H(s) \oplus t \\ z &= G(r) \oplus s \\ m &= [z]_{k_1}^{k_1+k_2-1} \\ \mathsf{chk} &= [z]_0^{k_1-1} \end{aligned} \tag{12}$$

を計算する．ここで $[w]_{k_0}^{k-1}$ は w の k_0 から $k-1$ バイト目，$[w]_0^{k_0-1}$ は w の 0 から k_0-1 バイト目を意味する．
2. $\mathsf{chk} = 0^{8k_1}$ ならば，m を復号文として出力し，それ以外の場合，エラーを出力する．

演習 5.3 *OAEP* に楕円 *ElGamal* 暗号を用いる関数を構成しよう．
(1) $k_0 = 8, k_1 = 8, k_2 = 16$ として *128bit* のデータを *OAEP* パディングを行い，楕円 *ElGamal* 暗号・復号を行う関数を作成する．
(2) 表 *O.8* の 鍵 *K (*128 ビット*)* を暗号化した結果を求めよ．また復号できることを確かめよ．なお，楕円エルガマル暗号では 公開鍵，秘密鍵, 乱数は表 *Q.2* の データを用いる．（演習 *4.3* 参考）また *OAEP* 用の乱数は表 *Q.3* の r_{64} のデータを用いる．

5.3 OAEP の一般化

5.2 章では OAEP の基本概念について説明した．実際に RSA-OAEP 実用化する際には，ハッシュ関数や m のパディング手法など厳密に規格化されている **[?]**．また，一般に利用されるハッシュ関数は SHA1 などに見られるように出力のサイズが固定値のため，OAEP のように出力サイズが可変となる関数を既存のハッシュ関数から構築する必要がある．これが **MGF**(Mask Generation Function) と呼ばれる関数である．

IETF の RFC3447**[?]** においては，既存のハッシュ関数の出力バイト[8]数を k_0, 平文 m のバイトサイズを k_2，RSA の法 n のバイトサイズを k, 0 パディング列 PS のバイトサイズを k_1 とし，以下のように平文にパディングを設定する．ここで $k = 2k_0 + k_1 + k_2 + 2$ である．ここでは規格化されている RSA-OAEP の簡易版を記載する．詳細は RFC3447**[?]** を参照されたい．

【初期設定】

1. 公開鍵暗号を (Gen, Enc, Dec) とし，Enc の定義域のサイズを k バイト，平文空間のサイズを $k > k_2$ バイト，0 パディングのサイズを $k_1 > 0$ バイトとする．
2. **MGF** (message generation function) を以下で定める．

$$\begin{aligned} G_{\mathtt{MGF}}: &\quad \{0,1\}^{8k_0} &&\longrightarrow \{0,1\}^{8(k_0+k_1+k_2+1)} \\ H_{\mathtt{MGF}}: &\quad \{0,1\}^{8(k_0+k_1+k_2+1)} &&\longrightarrow \{0,1\}^{8k_0} \end{aligned}$$

3. 利用するハッシュ関数で決まる固定値を lHASH とする．SHA1 の場合，下記となる．

$$\mathrm{lHASH} = 1245845410931227995499360226027473197403882391305$$

4. 0 パディング列 PS を以下とする．PS は平文 m のサイズにより $\perp$(0 バイト) になることもある．

$$\mathrm{PS} = (0\mathrm{x})00^{k_1}$$

【暗号化】 ユーザ A が平文 $m \in \{0,1\}^{8k_2}$ を暗号化して B に送るとする．
1. 乱数 $r \in \{0,1\}^{8k_0}$ に対して，平文 m のパディング w を以下のように計算し，暗号関数に入力する．

$$\begin{aligned} \mathrm{Pad}(m) &= \mathrm{lHASH} \,\|\, \mathrm{PS} \,\|\, (0\mathrm{x})01 \,\|\, m \\ s &= G_{\mathtt{MGF}}(r) \oplus \mathrm{Pad}(m)\ (r \in \{0,1\}^{8k_0}) \\ t &= H_{\mathtt{MGF}}(s) \oplus r \\ w &= (0\mathrm{x})00 \,\|\, t \,\|\, s \\ \mathsf{cipher} &= \mathsf{Enc}(PK, w) \end{aligned} \tag{13}$$

(14)

[7] ここではビッグエンディアンの表記で記載する．
[8] 8 ビットのことであるが，オクテット (octet) ともいう．

【復号】
1. B は自分の秘密鍵を用いて[9],

$$
\begin{aligned}
w &= \mathsf{Dec}(SK, \mathsf{cipher}) && (15)\\
s &= [w]_{k_0+1}^{k-1}\\
t &= [w]_{1}^{k_0}\\
r &= H_{\mathtt{MGF}}(s) \oplus t\\
z &= G_{\mathtt{MGF}}(r) \oplus s\\
m &= [z]_{k_0+k_1+1}^{k_0+k_1+k_2} && (16)
\end{aligned}
$$

を計算する．ここで $[w]_{k_0+1}^{k-1}$ は w の k_0+1 から $k-1$ バイト，$[w]_1^{k_0}$ は w の 1 から k_0 バイトを意味する．
2. 以下の条件をチェックし，OK なら m を復号文として出力し，それ以外の場合，エラーを出力する．

- (15) の w の 最上位 1 バイトが 0 である．
- (16) が Pad(m) のフォーマット (13) を満たしている．

5.4 ChaCha20

ストリーム暗号の一例として，Daniel J. Bernstein により 2008 年に開発された ChaCha を紹介する．ChaCha は 32 ビットを 1 ワードとし，ワード単位による算術加算，左ローテートシフト，排他的論理和の 3 つの演算に基づく擬似乱数生成アルゴリズムからなる．ChaCha の初期内部状態 $X^{(0)}$ は 8 ワードの秘密鍵: $k_0, \ldots, k_7$，3 ワードの Nonce: v_0, v_1, v_2，1 ワードのカウンタ値: t_0(初期カウンタ値は 1)，4 ワードの固定値: c_0 = 0x61707865, c_1 = 0x3320646e, c_2 = 0x79622d32, c_3 = 0x6b206574 の合計 16 ワードを 4×4 の行列式として，以下のように設定する．

$$
X^{(0)} = \begin{pmatrix} x_0^{(0)} & x_1^{(0)} & x_2^{(0)} & x_3^{(0)} \\ x_4^{(0)} & x_5^{(0)} & x_6^{(0)} & x_7^{(0)} \\ x_8^{(0)} & x_9^{(0)} & x_{10}^{(0)} & x_{11}^{(0)} \\ x_{12}^{(0)} & x_{13}^{(0)} & x_{14}^{(0)} & x_{15}^{(0)} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & v_0 & v_1 & v_2 \end{pmatrix}
$$

擬似乱数生成アルゴリズムでは 4 ワードのベクトル $(x_a^{(r)}, x_b^{(r)}, x_c^{(r)}, x_d^{(r)})$ を一単位として ChaCha の内部状態を更新する．ここで，更新の最小単位を**ラウンド** r，更新するための関数を **1/4 ラウンド (QuarterRound) 関数**と呼び，以下の演算により内部状態を更新する．

$$
\begin{cases}
x_{a'}^{(r)} = x_a^{(r)} + x_b^{(r)}; & x_{d'}^{(r)} = x_d^{(r)} \oplus x_{a'}^{(r)}; & x_{d''}^{(r)} = x_{d'}^{(r)} \lll 16;\\
x_{c'}^{(r)} = x_c^{(r)} + x_{d''}^{(r)}; & x_{b'}^{(r)} = x_b^{(r)} \oplus x_{c'}^{(r)}; & x_{b''}^{(r)} = x_{b'}^{(r)} \lll 12;\\
x_a^{(r+1)} = x_{a'}^{(r)} + x_{b''}^{(r)}; & x_{d'''}^{(r)} = x_{d''}^{(r)} \oplus x_a^{(r+1)}; & x_d^{(r+1)} = x_{d'''}^{(r)} \lll 8;\\
x_c^{(r+1)} = x_{c'}^{(r)} + x_d^{(r+1)}; & x_{b'''}^{(r)} = x_{b''}^{(r)} \oplus x_c^{(r+1)}; & x_b^{(r+1)} = x_{b'''}^{(r)} \lll 7;
\end{cases}
$$

QuarterRound 関数は行列式の列単位で演算を行う列ラウンドと行列式の対角単位で演算を行う対角ラウンドを交互に 10 回繰り返す．つまり，列ラウンドは (x_0, x_4, x_8, x_{12}), (x_1, x_5, x_9, x_{13}), $(x_2, x_6, x_{10}, x_{14})$, $(x_3, x_7, x_{11}, x_{15})$ の 4 組のベクトルを一単位として，対角ラウンドは $(x_0, x_5, x_{10}, x_{15})$, $(x_1, x_6, x_{11}, x_{12})$, (x_2, x_7, x_8, x_{13}), (x_3, x_4, x_9, x_{14}) の 4 組のベクトルを一単位として内部状態を更新する．ChaCha における 2 ラウンドの状態遷移図は図 1 のとおりである．ChaCha はラウンドを 20 回繰り返すことから **ChaCha20** と呼ばれる．20 ラウンド後の内部状態 $X^{(20)}$ が生成されると，初期内部状態 $X^{(0)}$ とのワード単位による算術加算 $X^{(0)} + X^{(20)}$ により 16 ワードの擬似乱数系列 (キーストリーム) を出力する．

ChaCha20 による平文の暗号化は平文を 16 ワードのブロックに分割し，それぞれのブロックに対してカウンタ値をインクリメントしながら対応するキーストリームブロックを生成する．生成したキーストリームブロックと平文ブロックとの排他的論理和をとることで暗号文を生成する．ChaCha による平文の暗号化は図 2 のとおりである．

擬似乱数を生成し，1 ビットごとにデータの暗号化を行うストリーム暗号と異なり，64 bits, 128 bits というブロックごとにデータの暗号化を行うののがブロック暗号である．ブロック暗号では暗号化するデータのビット数に合わせて，ブロック暗号の実行回数や出力ビット数を決定する必要がある．これがブロック暗号のモードと呼ばれる処理である．

問 5.2 *128* ビット暗号 Enc を利用し *32* ビット毎に暗号化を行う *CFB* モードを考える．つまり，*128* ビットの暗号文のうち *32* ビットを平文とのマスクに利用し，その *32* ビットを次の入力にフィードバックする方法である．このとき，*1* ビットの暗号文の誤りが復号の際に何ビットの平文の誤りに拡大するか考えよ．

[9] ここではビッグエンディアンの表記で記載する．

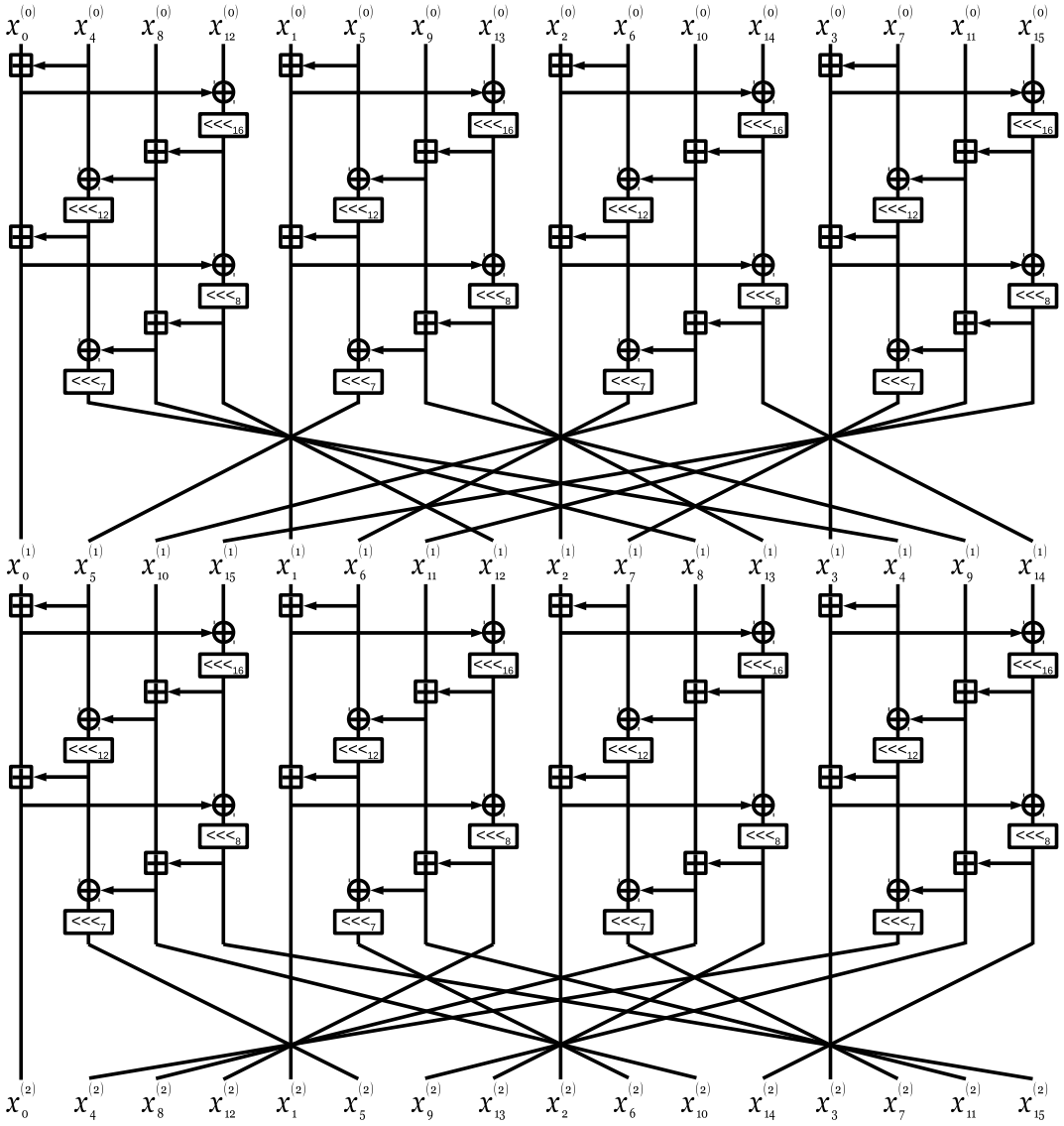


図 1: ChaCha における 2 ラウンドの状態遷移図

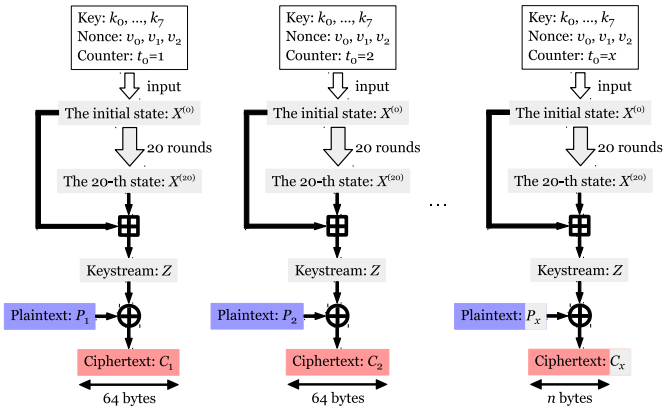


図 2: ChaCha による平文の暗号化

5.5 ハイブリッド暗号

SSL 通信等では鍵共有に公開鍵暗号を利用し，公開鍵暗号で共有した鍵を用いてデータの暗号化を行う．公開鍵暗号で共通鍵暗号の鍵共有を行う際にどのように OAEP を用いるのかを考えてみよう．

問 5.3 *128* ビット共通鍵暗号の鍵 *K* を公開鍵暗号を用いて暗号化して送付することを考える．このとき，*256* ビットの有限体上の楕円曲線暗号に *OAEP* 変換を行って，*K* を暗号化するとする．楕円曲線暗号と *OAEP* 変換を組み合わせた方式で必要となるハッシュ関数　G と H の定義域と値域をどのように設定するとよいか具体的に記載せよ．なお，ここでは *5.2* 章に記載した *OAEP* 変換を想定する．

また，演習 5.4 で実際に秘匿通信を実装してみよう．

演習 5.4 公開鍵 *PK* と平文 *M* の入力に対し，*128* ビット乱数 *K* を生成し，*OAEP* で鍵 *K* にパディングを付加し，楕円 *ElGamal* 暗号を用いて *K* を暗号化し，同時に *K* を用いて *ChaCha* で平文 *M* を暗号化する関数を構成しよう．なお，*OAEP* によるパディングは演習 *5.3* と同じ k_0, k_1, k_2 の設定とする． 暗号結果は *16* 進数バイト列で変換後ファイルに保存すること．
(1) 128 ビットの乱数 *K* を生成し，*K* を *OAEP* パディングを用いて楕円 *ElGamal* 暗号で暗号化する．楕円 *ElGamal* 暗号の公開鍵，秘密鍵，乱数については，演習 *5.3* と同一のデータを利用する．また *OAEP* 用の乱数は表 *Q.3* の r_{64} のデータを用いる．
(2) (1) で生成した *K* を *ChaCha* の鍵として，次の文章 *(*公開鍵暗号が最初に提案された論文のアブストラクト*)* を暗号化してみよう．
"Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing."
(3) (2) の暗号文を秘密鍵を用いて復元できることを確かめよ．

演習 5.5 楕円 *ElGamal* 暗号と *ECDSA* 署名の自分の公開鍵と秘密鍵を生成し，実際に暗号・復号を体験しよう．暗号文や必要な公開鍵データは *moodlle* に掲載されている．
(1) 楕円 *ElGamal* 暗号及び *ECDSA* 署名の鍵生成関数を用いて，自分の暗号化用の公開鍵と署名検証用の公開鍵 *(*整数値*)* を作成し，そのデータを *moodle* 掲示板に提出しよう．またハッシュ関数は *shake256* を用い，楕円曲線は演習 *4.3* を用いる．
(2) (1) を提出後，各自の *(1)* の鍵を用いたハイブリッド暗号の結果，つまり，暗号文と暗号化に用いた鍵を各自の公開鍵で暗号化した暗号化鍵データ *(*整数値*)* が *moodle* に掲載されている．なお，楕円 *ElGamal* 暗号は *OAEP* によるパディングを行い，パラメータは演習 *5.3* と同じ k_0, k_1, k_2 の設定とする．暗号文は今回の取り組みに関する内容である． 暗号化鍵データを秘密鍵で復号し，次に暗号文を復号して平文を確認しよう．各自への *TA* からの暗号文はそれぞれ該当する *TA* が掲示板に掲載する．
(3) (2) で復号した平文で記載された内容に沿って文章を作成しよう．次に作成した文章を各自の担当 *TA* 向けのハイブリッド暗号で暗号化し，各自の秘密鍵で署名を生成して提出する．各担当の公開鍵ファイルのデータは掲示板に掲載されている．

- 大阪 *ProA, NAIST,* 阪大工は 新井
- 大阪 *ProB, JAISTC* は *Kaiming*,
- 京大は *Mathieu*,
- 大阪宮地研 *A*，*JAISTB* は奥村,
- 大阪宮地研 *B*，*JAISTA* は加藤

作成した文章の暗号化を行い，暗号文 *(16* 進数*)* と署名 *(*整数*)* を提出しよう．
提出課題ファイルは *(1)(3)* である．*(1)* の楕円 *ElGamal* 暗号の公開鍵 *(ECDSA* 署名の公開鍵*)* は掲示板に掲載すること
(3) の暗号文と署名のファイルはそれぞれ，大学名頭文字_名前_*ECElG,* 大学名頭文字_名前_*ECDSA,* 大学名頭文字_名前_*Cipher,* のファイル名で提出すること．

参考文献

A　はじめに

Python を使用するにあたって最低限必要と思われる Python の使い方を説明する．

B　基本操作

Python を使用するにあたって最低限必要と思われる機能及び利用する組み込み関数を紹介する．ここでは，windows 10 上で Anaconda を利用する前提で説明する．

B.1　Windows 上での Python の利用について

Python は，デフォルトでは Windows にインストールされておらず，別途入手してインストールする必要がある．本章では，「Anaconda 5.0」(https://www.continuum.io/) をインストールすることを想定している．Anaconda は修正 BSD ライセンスで頒布されており無償利用が可能である．Anaconda は Python ディストリビューションの一つで，以下のようなライブラリのほかに，Python プログラムの記述と実行，メモの作成などをブラウザ上で行なう「Jupyter Notebook」，統合開発環境の「Spyder」などが自動的にインストールされる利点がある．

- NumPy：数値データの作成と操作に関するライブラリ
- SciPy：科学技術計算に関するライブラリ
- SymPy：代数計算に関するライブラリ

以降では，Python のバージョンは 3.6 として説明する．Python を利用するには以下ような方法がある．

スクリプト実行: Perl や Ruby と同様の使い方であり，テキストエディタを用いて「***.py」というファイルを作成し，コマンドプロンプトで実行する．[スタート] - [Anaconda3] - [Anaconda Prompt] でコマンドプロンプトを起動し，「python ***.py」を入力して実行する．コンパイルは不要である．

対話モード: Python を対話的 (Interactive) に実行するシェルである．対話モードは，[スタート] - [Anaconda3] - [Anaconda Prompt] でコマンドプロンプトを起動し，″python″とタイプしてリターンキーを押下することによって開始する．また，″exit()″または″quit()″とタイプしてリターンキーを押下することによって終了することができる．対話モードでは一行ずつ入力して出力を得ることができる．エディタを使用しないため，簡易的な計算を行うのに向いている．

Jupyter Notebook: ウェブブラウザ上で Python を実行・表示できるツールである．ノートのように扱うことができ，式と答えが表示された形式のまま保存できる利点がある．ただし，これで保存されたファイルはノートブック形式であり，「***.ipynb」となる．このファイルは Jupyter Notebook 以外では実行できない．

B.2　Anaconda のインストール

Anaconda のインストール方法は https://docs.continuum.io/anaconda/install/windows に記載されている．Anaconda 4.4 時点での手順を以下に示す．

(1) Anaconda を新規にインストールする

1. https://www.continuum.io/downloads からインストーラをダウンロードする．ここでは，Python 3.6 バージョンをダウンロードする．
2. ダウンロードしたインストーラを起動する．
3. Next をクリックする．
4. ライセンスを確認し，″I Agree″を押す．
5. インストール対象ユーザーを選ぶ．ここでは，″Just Me″ を選択して ″Next″ を押す．
6. インストールフォルダを選択して ″Next″ を押す．デフォルト設定でよい．
7. Python の環境変数の設定，Python 3.6 をデフォルトの Python として登録するかを問われるが，デフォルト設定で良い．
8. ″Install″ を押すとインストールが始まる．
9. ″Finish″ を押し，インストーラを閉じる．

(2) 既にインストールしている Anaconda に Python3 系を導入する

1. 「スタート」 -「Anaconda3」 -「Anaconda Prompt」からコマンドプロンプトを起動する．
2. 「conda create -n py36 python=3.6 anaconda」と入力して，Enter を押す．
3. 「Proceed ([y]—n)?」と聞かれるので ″y″ を押す．
4. インストール完了後,「activate py36」と入力して Enter を押すと，Python3.6 環境が利用できる．
5. 元のバージョンの環境に戻したい場合は,「deactivate」と打ち込んで非アクティブ化することができる．
6. 詳細は https://conda.io/docs/using/envs.html#share-an-environment を参照．

B.3 Python プログラムの実行方法

ここでは，プログラムファイルの実行と対話実行，Jupyter Notebook の 3 種類について解説する．

(1) Python プログラムファイルの実行

1. プログラムをテキストエディタ等で作成し，拡張子 ″.py″ で保存する．ここでは，作成したプログラムのパスを ″C:\work\test.py″ と仮定する．
2. 「スタート」 -「Anaconda3」 -「Anaconda Prompt」からコマンドプロンプトを起動する．
3. コマンドプロンプト上で「cd /d C:\work」と入力して Enter を押すと，作業フォルダへ移動できる．
4. 「python test.py」と入力し Enter を押すとプログラムが実行される．

(2) Python プログラムの対話実行

1. 「スタート」 -「Anaconda3」 -「Anaconda Prompt」からコマンドプロンプトを起動する．
2. 「python」と入力し Enter を押すと，Python 対話モードが開始する．
3. 実行したい Python のステートメントを記述して Enter キーを押すと実行される．
4. 「exit()」または「quit()」と入力し Enter を押すと，Python 対話モードを終了する．

(3) Jupyter Notebook での実行

1. 「スタート」 -「Anaconda3」 -「Jupyter Notebook」から Jupyter Notebook を起動する．
2. 「New」 -「Python3」から新規ファイルを作成する
3. 実行したい Python のステートメントを記述して Shift キーを押しながら Enter キーを押すと実行される．
4. Python プログラムを中断するには i を二度押しする．
5. 保存は「File」 -「Save and Checkpoint」，ファイル名の変更は「File」 -「Rename...」で行う．
6. 保存したファイルを開くには,「File」 -「Open」でファイルを選択することにより行う．

C 基本事項

C.1 コメント

一行コメント： #から行末までがコメントアウトされる．

```
print("abc") # コメント
```

複数行コメント： 3 重クオート文字列で囲うと，その内部がコメントアウトされる．

```
'''
コメント
'''
```

C.2 変数

Python3 では「変数名 = 値」で変数を作成することができる．変数の値は変更することができる．変数はどのような型でも初期化・代入することができる．

```
a = 1        # 変数 a を 1 で初期化
b = 2        # 変数 b を 2 で初期化
c = a + b    # 変数 c に a + b の結果を代入
print(c)     # c の中身を表示
```

C.3 型

Python3 における代表的な組み込み型を列挙する．

型名	書式例	説明
整数型 (int)	1	多倍長の整数型．桁あふれを意識する必要がない．
浮動小数点型 (float)	1.0	指数表現による実数表現．丸め誤差などに注意が必要
文字列型 (str)	'abc'	Python3 では Unicode 文字列で日本語を扱える．
バイト型 (bytes)	b'abc'	ASCII 文字列.
ブール型 (bool)	True	True または False.
None 型 (NoneType)	None	空の状態を示す.

現在の変数がどの型であるかを確認するには type() 関数を用いる．

```
a = 1          # 整数型(int)
b = 1.0        # 浮動小数点型(float)
c = 'あいう'   # Unicode 文字列
d = "あいう"   # ダブルクォーテーションで囲っても同じ
e = b'abc'     # ASCII 文字列
f = True       # ブール型
g = None       # None 型

type(a)        # class 'int'
```

C.4 print 関数

プログラム実行中にコンソール画面に文字列を表示させるには print() を用いる．また，文字列中に別の文字列を挿入するには，format() を用いる．次の print() 関数の呼び出しでは，いずれも "Hello, world."を表示する．

```
s1 = "Hello, world."
s2 = "Hello, {}.".format("world")
a = "Hello"
b = "world"
s3 = "{}, {}.".format(a, b)
print("Hello, world.")
print(s1)
print(s2)
print(s3)
```

format() は数値に対しても有効である．次のプログラムを実行すると 「2 times 3 equal 6」と表示される．

```
print("{} times {} equal {}".format(2,3,6))
```

D　数値型

ここでは，Python3 にある数値型について解説する．

型名	書式例	説明
整数型 (int)	1	多倍長の整数型．桁あふれを意識する必要がない．
浮動小数点型 (float)	1.0	指数表現による実数表現．丸め誤差などに注意が必要．
複素数型 (complex)	1.0 + 2.0j	実部・虚部は float 型．
有理数型 (fractions.Fraction)	fractions.Fraction(2, 5)	分子・分母に int を持つ．

D.1　基本的な演算

四則演算・商・剰余・べき乗は以下の記号が対応する．Python3 では，整数同士の除算において，"/"による結果は浮動小数点，"//"による結果は整数となるため注意が必要である．

```
a + b          # 加算
a - b          # 減算
a * b          # 乗算
a / b          # 除算
a // b         # a を b で割った商
a % b          # a を b で割った余り
a ** b         # a の b 乗
```

D.2　ビット演算

int 型に対して以下のビット演算子が用意されている．

```
~a      # ビット反転
a & b   # 論理積(AND)
a | b   # 論理和(OR)
a ^ b   # 排他的論理和(XOR)
a << b # b ビット左シフト
a >> b # b ビット右シフト
```

D.3　有理数の扱い

有理数を扱うクラスとして fractions.Fraction が用意されている．

```
from fractions import Fraction
import math
a = Fraction(3, 4)     # a = 3/4
b = Fraction(16, -10) # b = -8/5 自動的に通分される
a + b                  # -> Fraction(-17, 20)
Fraction(math.pi)      # -> Fraction(884279719003555, 281474976710656) 実数は分数に近似されるので注意
Fraction(a, b)         # -> Fraction(-32, 15) 分子・分母に分数を入れてもOK
```

D.4　法演算の扱い

整数の法演算は，演算後に % 演算子で余りを求めるとよい．べき乗は pow 関数を使うこともできる．

```
a, b, m = 6, 8, 7
(a * b) % m      # a + b mod m
pow(a, b, m)     # (a ** b) % m
```

高速化が必要な場合は，gmpy2 (http://gmpy2.readthedocs.io/en/latest/index.html) を利用する．本ライブラリは GNU Multi-precision Library の Python ラッパーである．

E　文字列型

ここでは Python3 の文字列型について解説する．

型名	書式例	説明
文字列型 (str)	'abc'	Python3 では Unicode 文字列で日本語を扱える．
バイト列型 (bytes)	b'abc'	ASCII 文字列.

E.1　文字列操作

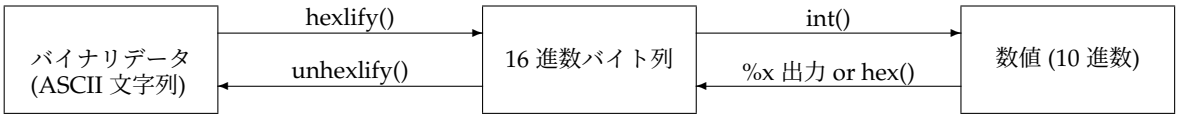
代表的な文字列操作を列挙する．[] 演算子による部分文字列の切り出しをスライシングと呼ぶ．

```
a + b       # 文字列 a と 文字列 b を連結
a * n       # 文字列 a を n 回繰り返し
a[0]        # 文字列 a の最初の文字を取り出す
a[-1]       # 文字列 a の最後の文字を取り出す
a[n]        # 文字列 a の中の n 番目の文字を取り出す
a[-n]       # 文字列 a 後ろから n 番目の文字を取り出す
a[n:m]      # 文字列 a の中の n 番目から m-1 番目までの文字列を取り出す
a[n:]       # 文字列 a の中の n 番目から最後までの文字列を取り出す
a[:m]       # 文字列 a の中の 0 番目から m-1 番目までの文字列を取り出す
a[n:m:s]    # 文字列 a の中の n 番目から m-1 番目までの文字列を s 個飛ばしで取り出す
```

E.2　文字列・数値型の相互変換

Python では，ビット演算以外の演算は 10 進数の数値で行う必要がある．しかし，乱数生成関数である urandom(n) の出力はバイナリデータで得られる．さらに，バイト列をバイナリデータとして内部では扱う．例えば,「a = b'Python'」と書いた場合，変数 a には b'Python' というバイト列がバイナリデータとして格納されることになる．そのため，これらバイナリデータを 10 進数の数値に変換した後に演算を行わなければならない．バイナリデータを 10 進数の数値に変換するためには，下図の通り，まずバイナリデータを 16 進数のバイト列に変換する（バイナリデータは 16 進数のバイト列とバイト単位で対応）．その後，16 進数のバイト列を 10 進数の数値に変換する．

16 進数バイト列は単なる中継の役割ではなく，主なメリットが 2 つある．1 つは，バイナリデータをバイト単位で可視化できる点であり，もう 1 つはバイト列として保存する際に 10 進数に比べてかなり短くできる点である．もちろんバイナリデータで保存するのが最もデータサイズを短くできるが，テキストデータの方が扱いやすいという場合がある．



これらの変換のサンプルコードを以下に示す．

```
import os, binascii
a0 = os.urandom(10)          # バイナリデータ
a1 = binascii.hexlify(a0)    # バイナリデータ → 16 進バイト列
a2 = int(a1, 16)             # 16 進バイト列 → 整数型
a3 = b"%x" % a2              # 整数型 → 16 進バイト列
a4 = binascii.unhexlify(a3) # 16 進バイト列 → バイナリデータ
```

16 進数バイト列というのはあくまでもオリジナルのデータをバイト毎に可視化しているバイト列に過ぎないということに注意する．例えば b'python' というバイト列（内部ではバイナリデータとして扱われる）は，16 進数バイト列では「707974686f6e」であり，10 進数の数値では「123666946355054」である．つまり，オリジナルのデータである'Python' から見ると，16 進数バイト列は単なる 16 進数で表現されたバイト列であり，これをオリジナルデータのバイト列として関数に入力してはいけない．x='Python'

の x にはバイナリデータが入っていることを考えると，オリジナルデータのバイト列というのは下図の一番左に位置するバイナリデータを指すと思って差し支えない．

E.3 文字列 (2/8/10/16 進数) から整数への変換

int() により文字列を整数に変換できる．第一引数はバイト列 (bytes) でも構わない．第二引数は基数を指定する．第二引数を省略すると 10 進数に変換される．

```
int('14')       # => 14  ( 10 進数 )
int('101', 2)   # => 5   (  2 進数 )
int('101', 8)   # => 65  (  8 進数 )
int('ff', 16)   # => 255 ( 16 進数 )
```

E.4 整数から文字列 (2/8/10/16 進数) への変換

整数から文字列への変換には下記のような方法がある．バイト列へ直したい場合は，encode() 関数を利用すれば良い．

```
a = 255

# 10 進数
"%d" % a         # => '255'
format(a, 'd')   # => '255'

#  2 進数
bin(a)           # => '0b11111111'
format(a, 'b')   # => '11111111'

#  8 進数
oct(a)           # => '0o377'
"%o" % a         # => '377'
format(a, 'o')   # => '377'

# 16 進数
hex(a)           # => '0xff'
"%x" % a         # => 'ff'
format(a, 'x')   # => 'ff'
```

E.5 バイナリデータと 16 進バイト列の相互変換

hexlify() はバイナリデータを 16 進バイト列に変換し，逆に unhexlify() は 16 進バイト列をバイナリデータに変換する．

```
import binascii
v1 = binascii.hexlify(b'Python')
if (len(v1) % 2 == 0):
    v2 = binascii.unhexlify(v1)
else:
    v2 = binascii.unhexlify('0' + v1)
print(v1) # => b'507974686f6e'
print(v2) # => b'Python'
```

バイナリデータと 16 進数バイト列はバイト単位で対応するため，unhexlify() の入力は偶数個の 16 進数バイト列でなければならない．そこで，上記サンプルコードに示されているように，奇数個の場合は 16 進数バイト列としての 0 をパディングする．

E.6 バイト列 (bytes) と Unicode 文字列 (str) の相互変換

バイト列 (bytes) と Unicode 文字列 (str) を相互変換するには, encode() および decode() を用いる．

```
a = 'あいうえお'
b = a.encode() # Unicode 文字列 → バイト列
c = b.decode() # バイト列 → Unicode 文字列
print(c)       # => 'あいうえお'
```

関数によっては引数や戻り値がバイト列 (bytes) か Unicode 文字列 (str) に限定される場合がある．型は type() 関数を用いて確認することができる．

F 制御文

F.1 ブール演算

Python は and, or, not のブール演算を持つ．

```
a and b   # 論理積
a or b    # 論理和
not a     # 否定
```

F.2 if 文

C 言語と比較して注意点が 3 箇所ある．1 つは各条件文の最後にコロン「：」が必要なこと，2 つ目は elif となっていること，さらに条件文が真のときに実行される範囲はインデントされている範囲だけである．if-elif-else の分岐処理のサンプルコードを以下に記す．

```
n = 0
if n < 0:
    print('n < 0')
elif n == 0:
    print('n == 0')
else:
    print('n > 0')
```

結果：n==0

F.3 for 文

通常のカウンタを取る for 文のサンプルコードを以下に記す．インデントされている範囲だけ繰り返されることに注意する．range 関数は range オブジェクトを作成する関数であり，range(5) は 0≤x<5 となる整数 x を順々に生成する．

```
a = 0
for i in range(5):
    a = a + i
print(a)  # => 10
```

F.4 while 文

while 文のサンプルコードを以下に記す．for 文と同様，インデントされている範囲だけ繰り返されることに注意する．

```
a = 4
b = 0
while a > 0:
    b = b + a
    a = a - 1
print(b)  # => 10
```

G コンテナ

G.1 リスト (list)

リストとは任意のオブジェクトのシーケンスであり，その要素は 0 から始まる整数で番号付けされる．リストを作るには [ ] を使い，リストにはどのような値も持たせることができる．例えば enc=[‘RSA’,[‘AES’,‘DES’,‘RC4’]] のように書く．このとき，enc[0] が RSA，enc[1] が [‘AES’,‘DES’,‘RC4’] である．さらに，enc[1][2] は RC4 である．リストのサイズは len で調べることができる．例えば，len(enc) は 2 である．

```
# リストの作成
a = []                          # 空リストの作成
b = ['RSA','AES','DES']         # 複数要素からなるリスト作成
c = ['RSA',['AES','DES','RC4']] # リストの入れ子
d = [0, 1, 2, 3, 4, 5]

# 非破壊的操作 -> リストは変更されない
len(b)      # リストの要素数
a + b       # リスト a と リスト b を連結
a * n       # リスト a を n 回繰り返し
a[0]        # リスト a の最初の要素を取り出す
a[-1]       # リスト a の最後の要素を取り出す
a[n]        # リスト a の中の n 番目の要素を取り出す
a[-n]       # リスト a 後ろから n 番目の要素を取り出す
a[n:m]      # リスト a の中の n 番目から m-1 番目までのリストを取り出す
a[n:]       # リスト a の中の n 番目から最後までのリストを取り出す
a[:m]       # リスト a の中の 0 番目から m-1 番目までのリストを取り出す
a[n:m:s]    # リスト a の中の n 番目から m-1 番目までのリストを s個とばしで取り出す
b = a[:]    # リスト a をコピーして b へ代入する
sorted(a)   # リスト a をソートして返す

# 破壊的操作 -> リストが変更される
a[n] = 'RC4'      # n 番目の要素を置き換える
first = d.pop(n)  # n 番目の要素を取り出し
d.insert(m, x)    # 任意の要素 x を m 番目の要素の前に追加する
last = d.pop()    # 末尾を取り出す
d.append(x)       # 末尾に要素 x を追加
d.extend([0, 1])  # 末尾にリストを追加
a.sort()          # リストをソートする
```

G.2 タプル (tuple)

(...) で要素を並べたものをタプル (tuple) とよぶ．タプルはリストとほぼ同じような操作が可能だが，要素を変更できない点が異なる．

```
# タプルの作成
a = (1, 2, 3, 4)
b = (10,)          # 要素が1つのときはカンマが必要

# 非破壊的操作 -> タプルは変更されない
len(b)       # タプルの要素数
a + b        # タプル a と タプル b を連結
a * n        # タプル a を n 回繰り返し
a[n]         # タプル a の中の n 番目の要素を取り出す
a[n:m]       # タプル a の中の n 番目から m-1 番目までのタプルを取り出す
a[n:]        # タプル a の中の n 番目から最後までのタプルを取り出す
a[:m]        # タプル a の中の 0 番目から m-1 番目までのタプルを取り出す
a[n:m:s]     # タプル a の中の n 番目から m-1 番目までのタプルを s個とばしで取り出す
sorted(a)    # タプル a をソートして返す (戻り値はリスト)

# 破壊的操作はNG
a[2] = 60          # エラー

# リストとの相互変換
list((1, 2, 3))    # -> [1, 2, 3]
tuple([1, 2, 3])   # -> (1, 2, 3)
```

G.3 辞書 (dict)

... は、辞書 (dict) と呼ばれるキーと値の組を持つ．辞書はハッシュテーブルで実装されているため，各要素への高速なランダムアクセスが可能である．

```
# 辞書の作成
d = {'RSA': 10, 'AES': 20, 'DES': 30}

# 各要素へのアクセス
d1 = d['RSA']
d2 = d['AES']
d3 = d['DES']

# 要素の有無チェック
if 'RSA' in d:
    print("key=RSA is included")

# 要素の追加・削除
d['RC4'] = 40 # ('RC4', 40) の追加
del d['RSA']  # ('RSA', 10) の削除


# for 文によるイテレーション
for k, v in d.items():
    print(k, v)

for k in d.keys():
    print(k, v[k])
```

G.4 セット (set)

セット (set) は重複の無いリストを扱う．セット同士の減算・OR・AND・XOR 操作が可能である．

```
a = set([1, 2, 3])
b = set([3, 4, 5])

print(a)           #=> {1, 2, 3}
print(b)           #=> {3, 4, 5}
print(a - b)       #=> {1, 2}
print(a | b)       #=> {1, 2, 3, 4, 5}
print(a & b)       #=> {3}
print(a ^ b)       #=> {1, 2, 4, 5}
print(1 in a)      #=> True
a.add(4)
print(a)           #=> {1, 2, 3, 4}
```

G.5 リスト内包表記

リスト内包表記を用いることにより，リスト操作を簡素に記述することができる．リスト内包表記は for 文よりも 2 倍程度高速に動作することが多い．

```
a = [1, 2, 3]
print([x * 2 for x in a])                          #=> [2, 4, 6]
print([x * 2 for x in a if x == 3])                #=> [6]
print([[x, x * 2] for x in a])                     #=> [[1, 2], [2, 4], [3, 6]]
print([(x, x * 2) for x in a])                     #=> [(1, 2), (2, 4), (3, 6)]

b = [4, 5, 6]
print([x * y for x in a for y in b])               #=> [4, 5, 6, 8, 10, 12, 12, 15, 18]
print([a[i] * b[i] for i in range(len(a))])        #=> [4, 10, 18]
```

H 関数

H.1 関数の作成

他のプログラミング言語と同様に，Python においても自身で関数を定義できる．関数を定義するには def を使用する．ただし，関数本体のインデントは必須であり，関数の範囲を意味する．次の関数は，2 つの入力を入れ替える関数例である．

```
def ex(a, b):
    return (b, a)
```

H.2 関数の呼び出し

下記は関数の呼び出し例である．

```
print(ex(1, 3)) # => (3,1)
```

Python では変数の定義を明示的にしないが，変数は自作関数の内部/外部で区別される．また，変数の型は何を代入するかによって自動的に決まる．異なる型の値を代入すれば，その都度変数の型が変わる．

I ライブラリの利用

I.1 モジュールとパッケージ

Python のライブラリは，モジュールやパッケージと呼ばれる単位で管理されている．

たとえば，プロセッサ時刻を求める関数 clock() は time モジュールに含まれる．このため，clock() を使用する際には time モジュールをあらかじめ import して呼び出す．

```
import time
time.clock()
```

chi2 は scipy パッケージの stats モジュールに含まれるクラスである．呼び出しパスは scipy.stats.chi2() である．呼び出し方は複数あるが，下記サンプルコードの下へいくほど名前空間が汚れるため注意が必要である．

```
# import [パッケージ.]モジュール
import scipy.stats
a = scipy.stats.chi2(1.0)

# from パッケージ import モジュール
from scipy import stats
b = stats.chi2(1.0)

# from [パッケージ.]モジュール import 識別子
from scipy.stats import chi2
c = chi2(1.0)

# from パッケージ import *
from scipy.stats import *   # 全ての識別子がimportされる
d = chi2(1.0)
```

I.2 ライブラリ一覧

以下は本稿で利用するライブラリの一覧である．

モジュール	クラス / メソッド
組み込み関数	min, max, sum
math	floor, ceil, exp, log, pow, sqrt, sin, cos, tan
sympy	gcd, lcm, invert, factorint, divisors, prime, primepi, nextprime, prevprime, isprime, randprime
binascii	hexlify, unhexlify
os	urandom
random	seed, randrange, getrandbits
hashlib	sha1, sha224, sha256, sha384, sha512, shake_128, shake_256
pickle	dump, load
csv	writer, reader,
numpy	mean, var, std
scipy.stats	chi2.ppf
time	clock, perf_counter()

以下にサンプルコードを記す．関数 `gcd()` を使用する際には `sympy.gcd()` と書く．ただし，import は一度でよい．

```
import sympy
sympy.gcd(6, 9) # => 3
```

I.3 組み込み関数

組み込み関数から主要なものを列挙する．

```
min(1, 2, 3) # 最小値, 引数はリストでも可.
max(1, 2, 3) # 最大値, 引数はリストでも可.
sum(1, 2, 3) # 総和
```

I.4 math ライブラリ

math ライブラリから主要な関数を列挙する.

```
import math
math.pi           # 円周率 π
math.e            # 自然対数の底 e
math.floor(x)     # x 以下の最大の整数
math.ceil(x)      # x 以上の最小の整数
math.exp(x)       # e**5
math.log(x)       # eを底とするlog
math.pow(x, y)    # x ** y
math.sqrt(x)      # xの正の平方根
math.sin(x)       # sin(x)
math.cos(x)       # cos(x)
math.tan(x)       # tan(x)
```

I.5 sympy による整数の扱い

sympy による整数の扱いについて紹介する.

```
import sympy,random
sympy.gcd(a, b)     # a, b の最大公約数
sympy.lcm(a, b)     # a, b の最小公倍数
sympy.invert(a, b)  # a mod b の逆元
sympy.factorint(a)  # a の素因数分解
sympy.divisors(a)   # a の約数
sympy.prime(a)      # a 番目の素数
sympy.primepi(a)    # a より小さい素数の個数
sympy.nextprime(a)  # a より大きい最小の素数
sympy.prevprime(a)  # a より小さい最大の素数
sympy.isprime(a)    # a の素数判定

random.seed(x)        # 乱数シードをセット
sympy.randprime(a, b) # [a, b) 間のランダムな素数を生成
```

I.6 sympy による多項式の扱い

sympy による多項式の扱いについて紹介する.

```
import sympy
x, y = sympy.symbols('x y') # 変数宣言
sympy.expand((x + y)**3)    # 式の展開
sympy.factor(x**3 + 3*x*y**2 + 3*x**2*y + y**3)    # 式の展開
sympy.gcd(x**2 + x, x**2)   # 最大公約数
sympy.lcm(x**2 + x, x**2)   # 最小公倍数
```

I.7 sympy による級数和の扱い

sympy による級数和の扱いについて紹介する．

```
import sympy
n = sympy.symbols('n') # 変数宣言
A = sympy.summation(1/(2**n), (n, 0, sympy.oo)) # a_n = 1/(2**n) なる級数の和 (0 <= n <= ∞)
print(A)    # => 2
```

I.8 乱数生成

```
os.urandom(n)
random.seed(x)
```

os.urandom(n) は，暗号の使用に適している疑似乱数を生成する関数であり，n バイト（n は整数）からなる乱数をバイナリデータで返す．この関数は，Windows API である CryptGenRandom を内部的に呼び出す．本関数は，環境ノイズなどから生成されるエントロピープールを利用して擬似乱数を生成し，乱数のシードとしても利用可能である．os.urandom() を使用する際には文頭で「os」をインポートしておく．

random.seed(x) は乱数生成器を初期化する関数であり，整数 x を乱数のシードにセットする．random.randrange(a,b) は random.seed(x) でセットされたシードを元に a 以上 b 未満の疑似乱数を生成する．

このサンプルコードは，urandom() により 20 バイトの擬似乱数を生成して seed として登録し，random.randrange によって 0 から 99 までの乱数を 10000 個生成する．

```
import os,random
myseed = os.urandom(20)
random.seed(myseed)
a = [random.randrange(0, 100) for i in range(10000)]
```

I.9 ハッシュ生成

```
hashlib.sha1(s).hexdigest()
hashlib.sha224(s).hexdigest()
hashlib.sha256(s).hexdigest()
hashlib.sha384(s).hexdigest()
hashlib.sha512(s).hexdigest()
hashlib.shake_128(s).hexdigest(length)
hashlib.shake_256(s).hexdigest(length)
```

sha1(s).hexdigest() は，入力である任意長のバイト列 s（入力のバイナリデータに相当）に対して，160 ビットの 16 進数文字列を出力するハッシュ関数 SHA1 であり，sha224(s).hexdigest() は，入力である任意長のバイト列 s に対して，224 ビットの 16 進数文字列を出力するハッシュ関数 SHA224 である．その他，256 ビット，384 ビット，及び 512 ビットの 16 進数文字列を出力するハッシュ関数も用意されている．また，可変長の戻り値を返す shake_128(s).hexdigest(length)，shake_256(s).hexdigest(length) が用意されている．これらの関数を利用するには hashlib を import しておく必要がある．

以下にハッシュ値を生成する 2 種類のサンプルコードを示す．1 つ目のサンプルコードは，バイト列 b′Python′ のハッシュ値を導出するものである．b′Python′ は内部ではバイナリデータとして扱われる．

```
import hashlib
v = hashlib.sha1(b'Python').hexdigest()
print(v) # '6e3604888c4b4ec08e2837913d012fe2834ffa83'
```

次のサンプルコードは，10 進数の数値とメッセージをビット連結したもののハッシュ値を導出するものである．数値もメッセージもパソコン内部ではビット列として扱われるのでビット連結できることは自然である．ただし，値の右側に最下位ビット (LSB) があることに注意する．

```
import hashlib
u = 123456789
m = b'Python'
tmp = b'%x' %u
if (len(tmp)%2 == 0):
    s1 = binascii.unhexlify(tmp)
else:
    s1 = binascii.unhexlify(b'0'+tmp)
s = s1 + m
v = hashlib.sha1(s).hexdigest()
print(v) # '9be4fd318b9d7d6a6104769ef012618d9f45bda7'
```

I.10 統計処理

平均・分散・標準偏差などは下記のように求められる．

```
import numpy
a = [1,2,3,4,5]
numpy.mean(a)  # 平均
numpy.var(a)   # 分散
numpy.std(a)   # 標準偏差
```

chi2.ppf(1-α,n) は，自由度 n，有意水準 α のカイ自乗統計量を導出する関数である．以下に，α=0.005，n=10 のときのカイ自乗統計量を導出するサンプルコードを示す．ただし，この関数の import 方法が他と異なることに注意する．

```
import scipy.stats
scipy.stats.chi2.ppf(1-0.005, 10) # => 25.188179571971173
```

I.11 時間計測

時間計測には time.clock() を利用する．最初にこの関数が呼び出されてからの経過時間を秒で返す．この関数は Windows API である QueryPerformanceCounter() に基づき，その精度は通常 1 マイクロ秒以下である．次に pow() の演算に要するの時間の測定の悪い例と良い例を示す．測定誤差を減らすために，1000 回の処理の平均を取ることにしている．当然入力値は同じ値にしないので，ここでは乱数生成関数を利用する．まずは悪い例を示す．この悪い例では，pow() の演算時間だけでなく，os.random() の演算時間も含まれてしまう点が悪い点である．

```
# 悪い例
import time,os,binascii
p = 184908779667576050572947262326548513689
t0 = time.clock()
for i in range(1000):
    a = int(binascii.hexlify(os.urandom(20)), 16)
    pow(2, a, p)
t1 = time.clock() - t0
print(t1/1000)
```

次に良い例を示す．ここでは，a の 1000 個のデータを予めメモリに用意しておき，純粋に pow() の演算時間だけを計算している．もちろん，メモリアクセス等の時間を要するが乱数生成関数の演算に要する時間に比べたら無視できるほど小さい．

```
# 良い例
import time,os,binascii
p = 184908779667576050572947262326548513689
a = [ ]
for i in range(1000):
    a.append(int(binascii.hexlify(os.urandom(20)), 16))
t0 = time.clock()
for i in range(1000):
```

```
    pow(2, a[i], p)
t1 = time.clock() - t0
print(t1/1000)
```

なお, time.clock() は Python 3.3 以降, 非推奨とされており, 代わりに time.perf_counter() を使用すると良い.

J ファイル入出力

J.1 基本

ファイルを扱うには open() によってファイルハンドルを取得する．open() 関数の第一引数はファイルパス，第二引数はオープンモードである．オープンモードには読み取りなら 'r'，書き込みなら 'w'，追加書き込みなら 'a' を指定する．また，'b' はバイナリモードである．同時に利用可能なファイルハンドル数には上限があるため，ファイルハンドルは利用後に速やかに close() する．read(), wirte() は一行ずつ読み書きする場合に適している．

```
x = ['aaa', 'bbb', 'ccc']

f = open('test.txt','w')
for row in x:
    f.write(row + '\n')
f.close()

y = []
f = open('test.txt','r')
for row in f:
    y.append(row.strip())
f.close()

print(y) # => ['aaa', 'bbb', 'ccc']
```

readlines() を利用すると，一度にすべての行を読み込むことが可能である．また，with as 構文を用いると自動的に close() が呼ばれるため，資源解放忘れがなくなる．

```
x = ['aaa', 'bbb', 'ccc']

with open('test.txt','w') as f:
    f.write('\n'.join(x))

with open('test.txt','r') as f:
    y = [row.strip() for row in f.readlines()]

print(y)  # => ['aaa', 'bbb', 'ccc']
```

J.2 pickle

以下の dump() と load() をペアで使用することによってデータ型を意識することなくファイルを入出力することができる．pickle はデータ型が入れ子になっている場合も再帰的に機能する．以下のサンプルコードは，データ x をファイル data.txt に書き出し，ファイル data.txt から y へ読み込む．data.txt が存在しない場合は新たにファイルが作成され，存在する場合は上書きされる．pickle を import しておく必要がある．

```
import pickle
x = [1, 2]
with open('data.txt', 'wb') as f:
    pickle.dump(x, f)

with open('data.txt', 'rb') as f:
```

```
    y = pickle.load(f)
print(y) # => [1, 2]
```

J.3 CSV ファイル入出力

大量の演算データを CSV 形式でファイル出力したり，ファイルから入力したりできると便利である．

```
import csv

with open('file.csv', 'w', newline='') as f:
  writer = csv.writer(f)
  writer.writerow([1, 'RSA', 'A'])
  writer.writerow([2, 'AES', 'S'])
  writer.writerow([3, 'RC4', 'S'])

with open('file.csv', 'r', newline='') as f:
    reader = csv.reader(f)
    a = [row for row in reader]

print(a) # => [['1', 'RSA', 'A'], ['2', 'AES', 'S'], ['3', 'RC4', 'S']]
```

K グラフ描画

基本的なグラフ描画は次のように書く．このサンプルコードは sin(x) のグラフを描くコードである．`plt.axis` は XY 座標の範囲を指定する関数で，下記では $0 \leq x < 5, -1.5 \leq y < 1.5$ が指定される．そして，`plt.grid` は目盛りを付ける関数である．また，`np.arange` は x の値を設定する関数であり，下記では 0 から 5 未満まで 0.01 ずつ増加する値がリスト形式で x にセットされる．つまり，y にはリスト形式で `sin(x)` の出力が格納される．さらに，`plt.plot` 及び `plt.show` で xy 座標にリストの組をプロットする．

```
import numpy as np
import matplotlib.pyplot as plt
plt.axis([0.0, 5.0, -1.5, 1.5])
plt.grid(True)
x = np.arange(0, 5, 0.01)
y = np.sin(x)
plt.title('y = sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x, y)
plt.show()
```

楕円曲線を描画するには，等高線を描く `contour` 関数を使う工夫が必要である．このサンプルコードは楕円曲線 $y^2 = x^3 + x + 1$ のグラフを描くコードである．ここでは，x と y の両方に値を設定し，`meshgrid` を使って 2 次元のリストに変換し，新たな変数 z を導入して $z = y^2 - x^3 - x - 1$ をプロットする．ただし，`contour` 関数の第 4 引数を [0] にすることで，$z = 0$ のときのグラフだけを描くことができる．

```
import numpy as np
import matplotlib.pyplot as plt
plt.grid(True)
x = np.arange(-1,1,0.01)
y = np.arange(-2,2,0.01)
(X,Y) = np.meshgrid(x,y)
Z = Y**2-X**3-X-1
plt.title('Y**2 = X**3 + X + 1')
plt.xlabel('X')
plt.xlabel('Y')
```

```
plt.contour(X,Y,Z,[0])
plt.show()
```

画像ファイルとして保存するには，show() の代わりに savefig() を呼ぶ．

```
import numpy as np
import matplotlib.pyplot as plt
plt.grid(True)
x = np.arange(-1,1,0.01)
y = np.arange(-2,2,0.01)
(X,Y) = np.meshgrid(x,y)
Z = Y**2-X**3-X-1
plt.contour(X,Y,Z,[0])
plt.savefig("graph.jpg") # png, epsなども可
```

L Python2 から 3 での変更点

Python2 から 3 でのバージョンアップ変更点で重要なものを掲載する．

L.1 print が文から関数に変更

Python3 から print 文は関数に変更された．このため，引数は括弧 () で括らなければならない．

```
# Python2
print "Hello"

# Python3
print("Hello")
```

L.2 a / b の仕様変更

int 型同士の除算 (/) は Python2 では int 型を返す演算子だったが，Python3 では float 型を返す演算子に変更された．Python3 で int 型を返したいときは，// 演算子を使う．

```
# Python2
3 / 2   # => 1

# Python3
3 / 2   # => 1.5
3 // 2  # => 1
```

L.3 文字列型が Unicode に

Python2 では文字列型はバイト列であったが，Python3 では文字列型が Unicode に変更された．Python3 でバイト列を扱いたいときは，バイト型 (bytes) を使う．

```
# Python2
"normal"          # Python2 での文字列型はバイト列
type("normal")    # <type 'str'>
u"unicode"        # Python2 でUnicodeを使うときは，文字リテラルの前に u を付与
type(u"unicode")  # <type 'unicode'>
```

```
# Python3
"normal"           # Python3 では文字列型が Unicode に
type("normal")     # <type 'str'>
b"bytedata"        # Python3 でバイト列を扱いたいときは，文字リテラルの前に b を付与
type(b"bytedata") # <type 'bytes'>
"str".encode()     # b'str'
b"bytes".decode() # b'bytes'
```

L.4 range() 関数の仕様変更

Python2 では range() 関数はリストを戻り値としていたため，長い反復処理で大きなリストを生成してパフォーマンスを低下させる欠点があった．このため，Python2 では反復毎に数字を生成する xrange() 関数が用意されていた．Python3 では，range() が xrange() と同様の処理を行うように変更され xrange() は廃止されている．

```
# Python2
for i in range(5):  # [0,1,2,3,4] が生成される
  print(i)

for i in xrange(5): # 反復毎に i へ 0 から 4 が順に代入される
  print(i)

# Python3
for i in range(5):  # 反復毎に i へ 0 から 4 が順に代入される
  print(i)

list(range(5))      # => [0,1,2,3,4]

for i in xrange(5): # 廃止されたのでエラー
  print(i)
```

M ライブラリ API

M.1 ライブラリ API 一覧

M.2 ライブラリ API 詳細

Algorithm 1 累乗計算 pow(x, y[, z])

Input: x, y, z (int 型)

Output: x の y 乗を返す. z があれば，x の y 乗に対する z の剰余を返す.

Object: int

Algorithm 2 累乗計算 divmod(x, y)

Input: x, y (int 型)

Output: x, y の商と余り $(x//y, x\%y)$ を返す.

Object: int

表 M.1: ライブラリ API 一覧

パッケージ	関数	内容
標準	pow(x, y, z)	累乗計算
標準	divmod(x, y)	整数の商と余り $(x//y, x\%y)$ を返す
標準	int.bit_length()	n の 2 進数表現のビット数を返す
標準	int.from_bytes(s, order)	与えられたバイト列の整数表現を返す.
標準	int.to_bytes(n, order)	整数を表すバイト列を返す.
標準	(str).encode()	Unicode 文字列をバイト列に変換する
標準	(byte).decode()	バイト列を Unicode 文字列に変換する
標準	bin()	整数を先頭に 0b がついた 2 進文字列に変換
binascii	binascii.hexlify(s)	ASCII バイト列を 16 進数を表すバイト列に変換する
binascii	binascii.unhexlify(s)	16 進数を表すバイト列を ASCII バイト列に変換する
hashlib	hashlib.shake_128(m)	バイト列 m を入力とする SHAKE128 のオブジェクトを返す
hashlib	hash.shake_128.hexdigest(n)	shake_128 オブジェクトによる出力を返す.
os	os.urandom(n)	長さ n のランダムなバイト列を生成する.
random	random.randrange(a, b)	[a, b) に含まれる乱数を返す.
random	random.seed(a, version)	乱数関数を初期化する. デフォルトは a=None, version=2
sympy	sympy.factorint(n)	n の素因数を返す.
time	time.perf_counter()	パフォーマンスカウンターの値 (小数点以下がミリ秒) を返す.
math	math.ceil()	入力 (浮動小数点) の「天井」 (入力以上の最小の整数) を返す.
numpy	numpy.mean()	入力データの算術平均を返す.
numpy	numpy.var()	入力データの分散を返す.
collections	collections.Counter()	入力 (リスト, タプル) の各要素とその要素の出現回数を返す.
scipy.special	scipy.special.perm(n, k)	順列 nPk の計算
scipy.special	scipy.special.comb(n, k)	組み合わせの数 nCk

Algorithm 3 ビット長出力 int.bit_length()

Output: 整数のビット数 (int 型)

Object: int

Algorithm 4 バイト列を整数へ変換 int.from_bytes(s, order)

Input: s: バイト列 (bytes 型), order: バイトオーダー ('big' or 'little')

Output: 整数

Remark: Python3.2 から追加.

Algorithm 5 整数をバイト列へ変換 int.to_bytes(n, order)

Input: n: 整数 (int 型), order: バイトオーダー ('big' or 'little')

Output: バイト列 (bytes 型)

Remark: Python3.2 から追加.

Algorithm 6 Unicode 文字列のエンコード (str).encode()

Input: str: 入力文字列 (Unicode 型)

Output: バイト列 (bytes 型)

Package: 標準

Algorithm 7 バイト列のデコード (bytes).decode()

Input: bytes: 入力バイト列 (byte 型)

Output: 文字列 (Unicode 型)

Package: 標準

Algorithm 8 整数の 2 進文字列への変換 bin()

Input: 整数

Output: 入力の 2 進文字列 (先頭に 0b がついている)

Package: 標準

Algorithm 9 ASCII バイト列を 16 進数バイト列へ変換 binascii.hexlify(s)

Input: s: ASCII バイト列 (bytes 型)

Output: 16 進数表現バイト列 (bytes 型)

Package: binascii

Algorithm 10 16 進数バイト列を ASCII バイト列へ変換 binascii.unhexlify(s)

Input: s: 16 進数表現バイト列 (bytes 型)

Output: ASCII バイト列 (bytes 型)

Package: binascii

Algorithm 11 Shake128 によるハッシュ値 hashlib.shake_128(m).hexdigest()

Input: m : バイト列 (bytes 型)

Output: m のハッシュ値 (bytes 型, 16 進数表示)

Package: hashlib

Algorithm 12 ハッシュアルゴリズム SHAKE128 のオブジェクト生成 hashlib.shake_128(m)

Input: m : バイト列 (bytes 型)

Output: SHAKE128 アルゴリズムを扱うオブジェクト (hash 型)

Package: hashlib

Algorithm 13 ランダムなバイト列を生成 os.urandom(n)

Input: n: バイト数 (int 型)

Output: ランダムな長さ n のバイト列 (bytes 型)

Package: os

Remark: エントロピープールを利用するため，乱数初期化子は持たない．Unix の/dev/urandom に相当する．

Algorithm 14 [a, b) の範囲内にある乱数を生成 random.randrange(a, b)

Input: a, b: 乱数の生成範囲 (int 型)

Output: 乱数 (int 型)

Package: random

Remark: random.seed で初期化される．

Algorithm 15 乱数関数の初期化 random.seed(a=None, version=2)

Input: a: 乱数シード (int or str or bytes 型), version: 乱数生成アルゴリズムのバージョン (int 型)

Package: random

Remark: random.randrange, random.getrandbits を初期化する．

Algorithm 16 入力 (整数) の素因数分解を行う．sympy.factorint(n)

Input: n：整数 (int 型)

Output: n の素因数分解

Package: sympy

Algorithm 17 時間計測 time.perf_counter()

Input: なし

Output: パフォーマンスカウンターの値 (小数点以下がミリ秒)

Package: time

Algorithm 18 浮動小数の天井 math.ceil()

Input: 浮動小数

Output: 入力の「天井」 (入力以上の最小の整数).

Package: math

Algorithm 19 算術平均 numpy.mean()

Input: データのリスト

Output: データの平均

Package: numpy

Algorithm 20 標本分散 numpy.var()

Input: データのリスト

Output: データの分散

Package: numpy

Algorithm 21 リスト中の要素の出現回数 collections.Counter()

Input: データのリスト

Output: 入力の各要素とその要素の出現回数

Package: collections

Algorithm 22 順列の計算 scipy.special.perm(n, k)

Input: 整数 n, k ($n \geq k$)

Output: 順列 ${}_nP_k$

Package: scipy.special

Algorithm 23 組み合わせの数の計算 scipy.special.comb(n, k)

Input: 整数 n, k ($n \geq k$)

Output: 順列 ${}_nC_k$

Package: scipy.special

N　作成関数 API

演習で作成する関数は他の演習で作成した関数を用いる必要があります．この際，関数名や関数の入出力変数を統一すると，共同作業がスムースに進みます．なお，参考関数として利用する Python の標準関数を記載します．それ以外の Python の標準関数は利用しないで，自分でアルゴリズムを考えて実装してください．また，自作した関数を利用する場合は，自作関数として記載していますので，自作関数を利用するようにしてください．なお，文字列・配列・ビットに関連した関数や制御関数については，明示的に指定されていなくても使用することができます．

N.1　第 1 回

Algorithm 24 ユークリッドの互除法 euclid(a, b)

Input: 整数: a, b

Output: 整数: a, b の最大公約数

Algorithm 25 拡張ユークリッドの互除法 ex_euclid(a, b)

Input: 整数: a, b

Output: リスト: $[d = ax + by, x, y]$

Algorithm 26 逆元 (拡張ユークリッドの互除法を利用) inv(a, n)

Input: 整数: a, n (n は素数と合成数のどちらも取りうる)

Output: 整数: $a^{-1} \bmod n$

自作関数：ex_euclid

Algorithm 27 拡張ユークリッドの互除法のループ回数計測 exEuclidExp(a, b)

Input: 整数: a, b

Output: リスト: $[d = ax + by, x, y, counter]$ (counter はループの実行回数)

Algorithm 28 逆元計算のループ回数計測 (拡張ユークリッドを利用) invExp(a, n)

Input: 整数: a, n (n は素数と合成数のどちらも取りうる)

Output: 整数: $a^{-1} \bmod n$, counter (counter はループの実行回数)

自作関数：exEuclidExp

Algorithm 29 法 n 上のバイナリ法 mod_binary(g, k, n)

Input: 整数: g, k, n (n は素数と合成数のどちらも取りうる)

Output: $g^k \bmod n$

参考関数：bin

Algorithm 30 法 n 上の拡張バイナリ法 ex_mod_binary(g_1, g_2, k_1, k_2, n)

Input: 整数: k_1, k_2, g_1, g_2, n (n は素数と合成数のどちらも取りうる)

Output: ${g_1}^{k_1} {g_2}^{k_2} \bmod n$

参考関数：bin, zip

Algorithm 31 フェルマーの小定理を利用した逆元計算 inv2(a, p)

Input: 整数: a, p (p は素数)

Output: 整数: $a^{-1} \bmod p$

自作関数：mod_bynary

Algorithm 32 法 n 上のバイナリ法でのループ回数計測 modBinaryExp(g, k, n)

Input: 整数: g, k, n (n は素数と合成数のどちらも取りうる)

Output: [$g^k \bmod n$, counter] (counter はループ回数)

参考関数：bin

Algorithm 33 逆元計算のループ回数計測 (フェルマーの小定理を利用) invExp2(a, n)

Input: 整数: a, n (n は素数と合成数のどちらも取りうる)

Output: 整数: $a^{-1} \bmod n$, counter (counter はループの実行回数)

自作関数：modBinaryExp

N.2 第 2 回

Algorithm 34 $\mathbb{Q}$ 上アファイン座標系の楕円曲線加算 aff_ec_add(x_1, y_1, x_2, y_2)

Input: 加算する 2 点の x, y 座標：$(x_1, y_1), (x_2, y_2)$ $((x_1, y_1) \neq (x_2, y_2))$

Output: 加算結果の x, y 座標：(x_3, y_3)

関連演習: 演習 2.1

Algorithm 35 $\mathbb{Q}$ 上アファイン座標系の楕円曲線 2 倍算 aff_ec_dbl(x_1, y_1, a)

Input: 2 倍する点の x, y 座標：(x_1, y_1), 楕円パラメータ (整数)：a

Output: 2 倍した結果の x, y 座標：(x_2, y_2)

関連演習: 演習 2.1

Algorithm 36 $\mathbb{Q}$ 上アファイン座標系の楕円曲線バイナリ法 aff_ec_exp(a, x_0, y_0, k)

Input: 楕円パラメータ (整数)：a, ベースポイント P の x, y 座標：(x_0, y_0), スカラ (整数)：k

Output: スカラ倍演算結果の x, y 座標：$(x_k, y_k) = kP$

参考関数: bin, len

自作関数: aff_ec_dbl, aff_ec_add

関連演習: 演習 2.1

Algorithm 37 有限体上アファイン座標系の楕円曲線加算 mod_aff_ec_add(x_1, y_1, x_2, y_2, p)

Input: 加算する 2 点の x, y 座標 (整数)：$(x_1, y_1), (x_2, y_2)$, 法：p $((x_1, y_1) \neq (x_2, y_2))$

Output: 加算結果の x, y 座標：$(x_3, y_3) = P + Q \bmod p$

自作関数: inv

関連演習: 演習 2.2

Algorithm 38 有限体上アファイン座標系の楕円曲線 2 倍算 mod_aff_ec_dbl(x_1, y_1, a, p)

Input: 2 倍する点の x, y 座標 (整数)：(x_1, y_1), 楕円パラメータ (整数)：a, 法：p

Output: 2 倍した結果の x, y 座標：$(x_2, y_2) = 2P \bmod p$

自作関数: inv

関連演習: 演習 2.2

Algorithm 39 有限体上アファイン座標系の楕円曲線バイナリ法 mod_aff_ec_exp(a, x_0, y_0, k, p)

Input: 楕円パラメータ (整数)：a, ベースポイント P の x, y 座標：(x_0, y_0), スカラ (整数)：k, 法：p

Output: スカラ倍演算結果の x, y 座標：$(x_k, y_k) = kP \bmod p$

参考関数: bin, len

自作関数: mod_aff_ec_dbl, mod_aff_ec_add

関連演習: 演習 2.2

N.3 第 3 回

Algorithm 40 $\mathbb{Q}$ 上の Jacobian 座標系の楕円曲線加算 jac_ec_add$(x_1, y_1, z_1, x_2, y_2, z_2)$

Input: 加算する 2 点の x, y, z 座標：(x_1, y_1, z_1), (x_2, y_2, z_2)

Output: 加算結果の x, y, z 座標：(x_3, y_3, z_3)

関連演習: 演習 3.1, 特演習 4

Algorithm 41 $\mathbb{Q}$ 上の Jacobian 座標系の楕円曲線 2 倍算 jac_ec_dbl(x_1, y_1, z_1, a)

Input: 2 倍する点の x, y, z 座標：(x_1, y_1, z_1), 楕円パラメータ (整数)：a

Output: 2 倍した結果の x, y, z 座標：(x_2, y_2, z_2)

関連演習: 演習 3.1, 特演習 4

Algorithm 42 $\mathbb{Q}$ 上の Jacobian 座標系の楕円曲線バイナリ法 jac_ec_exp(a, x_0, y_0, z_0, k)

Input: 楕円パラメータ (整数)：a, ベースポイントの x, y, z 座標：(x_0, y_0, z_0), スカラ (整数)：k

Output: スカラ倍演算結果の x, y, z 座標：$(x_k, y_k, z_k) = kP$

参考関数: bin, len

自作関数: jac_ec_dbl, jac_ec_add

関連演習: 演習 3.1, 特演習 4

Algorithm 43 有限体上の Jacobian 座標系の楕円曲線加算 mod_jac_ec_add$(x_1, y_1, z_1, x_2, y_2, z_2, p)$

Input: 加算する 2 点（整数）(x_1, y_1, z_1), (x_2, y_2, z_2) と法 p

Output: 整数 $(x_3, y_3, z_3) = P + Q \bmod p$

関連演習: 演習 3.2, 特演習 5

Algorithm 44 有限体上の Jacobian 座標系の楕円曲線 2 倍算 mod_jac_ec_dbl(x_1, y_1, z_1, a, p)

Input: 2 倍する点（整数）(x_1, y_1, z_1), 整数楕円パラメータ a と法 p

Output: 整数 $(x_2, y_2, z_2) = 2P \bmod p$

関連演習: 演習 3.2, 特演習 5

Algorithm 45 (参考) 有限体上の Jacobian 座標系の楕円曲線バイナリ法 mod_jac_ec_exp(a, x_0, y_0, z_0, k, p)

Input: 整数楕円パラメータ a, 整数ベースポイント (x_0, y_0, z_0), 整数 k と法 p

Output: 整数 $(x_k, y_k, z_k) = kP \bmod p$

参考関数: bin, len

自作関数: mod_jac_ec_dbl, mod_jac_ec_add

関連演習: 演習 3.2, 特演習 5

Algorithm 46 (参考) Jacobian 座標から Affine 座標系への座標変換 jac_to_affin(X, Y, Z)

Input: Jacobian 座標系で表現された点：$P = (X, Y, Z)$

Output: P の Affine 座標系での表現：(x, y)

関連演習: 特演習 4

Algorithm 47 有限体上アファイン座標系の楕円曲線加算 mod_aff_ec_add(x_1, y_1, x_2, y_2, p)

Input: 加算する 2 点の x, y 座標 (整数)：(x_1, y_1), (x_2, y_2), 法：p ($(x_1, y_1) \neq (x_2, y_2)$)

Output: 加算結果の x, y 座標：$(x_3, y_3) = P + Q \bmod p$

自作関数: inv

関連演習: 演習 3.3, 特演習 5

Algorithm 48 有限体上アファイン座標系の楕円曲線 2 倍算 mod_aff_ec_dbl(x_1, y_1, a, p)

Input: 2 倍する点の x, y 座標 (整数)：(x_1, y_1), 楕円パラメータ (整数)：a, 法：p

Output: 2 倍した結果の x, y 座標：$(x_2, y_2) = 2P \bmod p$

自作関数: inv

関連演習: 演習 3.3, 特演習 5

Algorithm 49 有限体上アファイン座標系の楕円曲線バイナリ法 mod_aff_ec_exp(a, x_0, y_0, k, p)

Input: 楕円パラメータ (整数)：a, ベースポイント P の x, y 座標：(x_0, y_0), スカラ (整数)：k, 法：p

Output: スカラ倍演算結果の x, y 座標：$(x_k, y_k) = kP \bmod p$

参考関数: bin, len

自作関数: mod_aff_ec_dbl, mod_aff_ec_add

関連演習: 演習 3.3, 特演習 5

N.4 第 4 回

Algorithm 50 共通乱数の楕円 ElGamal 暗号の解読 ec_elgamal_attack($(U, c, m), (U', c')$)

Input: 既知の平文と暗号文の組 (U, c, m) ((U, c) が暗号文，m はその平文), 解読対象の暗号文 (U', c')

Output: (U', c') の平文 m'

参考関数:

自作関数:

関連演習: 演習 4.1

Algorithm 51 楕円 ElGamal 暗号の鍵生成関数 ec_elgamal_key_gen(a, x_0, y_0, p, x)

Input: 楕円パラメータ：a, ベースポイントの x, y 座標：(x_0, y_0), 法：p, 秘密鍵 (整数)：x

Output: 公開鍵：$Y = x(x_0, y_0)$

参考関数:

自作関数: mod_aff_ec_exp

関連演習: 演習 4.2, 4.3, 4.4

Algorithm 52 楕円 ElGamal 暗号の暗号化関数 ec_elgamal_enc($m, a, x_0, y_0, pubx, puby, p, r$)

Input: 平文 (整数)：m, 楕円パラメータ：a, ベースポイントの x, y 座標：(x_0, y_0), 公開鍵：$(pubx, puby) \in E(\mathbb{F}_p)$, 法：$p$, 乱数 (整数)：$r$

Output: 暗号文：$C = [U, c]$ なる配列，ここで $U = [x_u, y_u]$ (U を x_u, y_u の配列) とする。すなわち、$C = [[x_u, y_u], c]$ とする。)

参考関数: ^ (XOR の計算)

自作関数: mod_aff_ec_exp

関連演習: 演習 4.2, 4.3, 4.4

Algorithm 53 楕円 ElGamal 暗号の復号関数 ec_elgamal_dec(C, a, x, p)

Input: 暗号文：$[U, c]$ ($U = [x_u, y_u]$), 楕円パラメータ (整数)：a, 秘密鍵：x, 法：p

Output: 平文 (整数)：m

参考関数: ^ (XOR の計算)

自作関数: mod_aff_ec_exp

関連演習: 演習 4.2, 4.3, 4.4

Algorithm 54 16 進数を 10 進数に変換 hex_to_decimal(h)

Input: 16 進数 h

Output: h を 10 進数に変換した数

参考関数: int(·, 16)

関連演習: 演習 4.3

N.5 第 5 回

Algorithm 55 ECDSA 署名-公開鍵生成 ec_dsa_key_gen(G, x, a, p)

Input: ベースポイント $G = [g_x, g_y]$, 秘密鍵：x, 楕円パラメータ (整数)：a, 法：p

Output: 公開鍵 $Y = xG = [Y_x, Y_y]$

参考関数: mod_aff_ec_exp

関連演習: 演習 5.1, 5.2, 5.5

Algorithm 56 Shake 256 のハッシュ値 (整数値) の計算 shake256(m, ℓ)

Input: 文字列： m, 出力のビット長：ℓ

Output: shake256 にょる m のハッシュ値 (ℓ ビット整数)

関連演習: 演習 5.2, 5.5

※この関数はこちらから提供します

Algorithm 57 ECDSA 署名-署名生成 ec_dsa_sign(m, x, G, ℓ, r, a, p)

Input: 文字列： m, 秘密鍵：x, ベースポイント $G = [g_x, g_y]$, ベースポイントの位数:ℓ, 乱数 (整数): r, 楕円パラメータ (整数)：a, 法：p

Output: 配列（整数）： $[u, v]$ s.t. $u = u_x \bmod \ell$, $v = r^{-1}(m + xu) \bmod \ell$

参考関数: 自作関数: mod_aff_ec_exp, inv, shake256

関連演習: 演習 5.1, 5.2, 5.5

Algorithm 58 ECDSA 署名-署名検証 ec_dsa_verify(signature, m, Y, G, ℓ, a, p)

Input: 署名配列: signature=[u, v], 文字列： m, 公開鍵: $Y = [Y_x, Y_y]$, ベースポイント $G = [g_x, g_y]$, ベースポイントの位数:ℓ, 楕円パラメータ (整数)：a, 法：p

Output: 署名が正しければ True を, そうでない場合 False を返す

参考関数: 自作関数: inv, mod_aff_ec_exp, mod_aff_ec_add, shake256

関連演習: 演習 5.2, 5.5

Algorithm 59 MGF(Message Generation Function) mgf(D, oLen)

Input: データ (16 進バイト列): D, 出力バイト長: oLen

Output: D の MGF 出力値 (16 進文字列)

関連演習：演習 5.3, 5.4, 5.5

※この関数はこちらから提供します

Algorithm 60 補助関数 xor(x, y)

Input: 16 進数バイト列 x, y

Output: 16 進数バイト列 $x \oplus y$

参考関数 binascii.unhexlify, binascii.hexlify, zip

関連演習：演習 5.3

O 数値例

演習問題利用する数値例を列挙する.

表 O.13: 実験 **??**

$p^{1024}_{205,1}$	=	493595662 2392582906 0159543226 4097389883 0219276387 0876133442543
$p^{1024}_{205,2}$	=	456808137 5831629151 5717355536 5566850489 5496119860 5929142386251
$p^{1024}_{205,3}$	=	506122852 3243889890 9959304006 4579176710 1990872376 8699682562461

$$
\begin{aligned}
p_{205,4}^{1024} &= 464070881\,5961179432\,5372916676\,9152885541\,1813776945\,0524712275639\\
p_{204,5}^{1024} &= 183708392\,8689162312\,8623749268\,8259016032\,6233453942\,4149602250757\\
q_{204}^{1024} &= 203918313\,8486605892\,0559002137\,1223440022\,4409438062\,2706168647991\\
p_{128,1}^{1024} &= 327727013\,0697280848\,8944386996\,3828059711\\
p_{128,2}^{1024} &= 327252369\,8083701070\,2068872026\,5095547587\\
p_{128,3}^{1024} &= 307454904\,9880630415\,6742678423\,2785496057\\
p_{128,4}^{1024} &= 319924500\,4586458424\,6505038090\,3060228263\\
p_{128,5}^{1024} &= 302452415\,7870358694\,8842693281\,9841899867\\
p_{128,6}^{1024} &= 338758195\,2055110602\,9709400454\,1911629483\\
p_{128,7}^{1024} &= 287458879\,5521621931\,8417952737\,8191280963\\
p_{128,8}^{1024} &= 305456423\,3327921932\,5853935801\,9720643289\\
q_{128}^{1024} &= 332279858\,4304493661\,1686731274\,3835330019\\
p_{102,1}^{1024} &= 495993670\,4085803091\,909934580737\\
p_{102,2}^{1024} &= 469814670\,4762017122\,018325831239\\
p_{102,3}^{1024} &= 486667865\,5871057795\,337652859779\\
p_{102,4}^{1024} &= 489797520\,2654528612\,188886803549\\
p_{102,5}^{1024} &= 446464625\,6452858241\,550294992549\\
p_{102,6}^{1024} &= 475909429\,5450630354\,542422115239\\
p_{103,7}^{1024} &= 973019323\,5597482959\,391696091919\\
p_{103,8}^{1024} &= 101118188\,9708315373\,4384206604517\\
p_{103,9}^{1024} &= 949832607\,6865856811\,759823202011\\
p_{103,10}^{1024} &= 985114119\,5354528562\,161235301321\\
q_{106}^{1024} &= 746930625\,8946365431\,6918233318301\\
p_{64,1}^{1024} &= 1793527218\,8932513967\\
p_{64,2}^{1024} &= 1800312522\,9117196033\\
p_{64,3}^{1024} &= 1827421461\,7505544791\\
p_{64,4}^{1024} &= 1751576479\,3748280049\\
p_{64,5}^{1024} &= 1811950497\,7260936187\\
p_{64,6}^{1024} &= 1821399703\,9822583309\\
p_{64,7}^{1024} &= 1618704703\,5808728619\\
p_{64,8}^{1024} &= 1728773677\,3492607021\\
p_{64,9}^{1024} &= 1836897792\,5200211593\\
p_{64,10}^{1024} &= 1776317044\,3295043023\\
p_{64,11}^{1024} &= 1774898105\,1160024049\\
p_{64,12}^{1024} &= 1747302870\,7383089513\\
p_{64,13}^{1024} &= 1716872894\,0836252187\\
p_{64,14}^{1024} &= 1746710983\,3925754251\\
p_{64,15}^{1024} &= 1773343856\,6473322513\\
p_{64,16}^{1024} &= 1763074430\,9897329279\\
q_{64}^{1024} &= 1690799673\,5427429109\\
p_{102,1}^{512} &= 499986983\,1519335186\,491039887011\\
p_{102,2}^{512} &= 434391841\,2294302841\,333376743907\\
p_{102,3}^{512} &= 454007051\,3713951418\,173066378121\\
p_{103,4}^{512} &= 923279523\,4423487512\,704957441133\\
p_{103,5}^{512} &= 738245583\,9101146879\,478406938167\\
q_{104}^{512} &= 192491032\,0067926071\,2302169662261\\
p_{64,1}^{512} &= 1800561438\,4815733387\\
p_{64,2}^{512} &= 1651409709\,4544156203\\
p_{64,3}^{512} &= 1701721994\,2406764657\\
p_{64,4}^{512} &= 1737059845\,9556484827\\
p_{64,5}^{512} &= 1822958800\,2006510251\\
p_{64,6}^{512} &= 1634346051\,7761035033\\
p_{64,7}^{512} &= 1658670383\,1293716201\\
p_{64,8}^{512} &= 1835786494\,4608595657\\
q_{64}^{512} &= 1754844339\,2621680141\\
p_{51,1}^{512} &= 2230411555\,766393\\
p_{51,2}^{512} &= 2245702431\,005177\\
p_{51,3}^{512} &= 2246104575\,278881
\end{aligned}
$$

$p_{51,4}^{512}$	=	1968007279 343447
$p_{51,5}^{512}$	=	2248619464 978387
$p_{51,6}^{512}$	=	2249329656 268613
$p_{51,7}^{512}$	=	2024614374 484313
$p_{51,8}^{512}$	=	2056690612 038619
$p_{52,9}^{512}$	=	4146564166 452341
$p_{52,10}^{512}$	=	4472076290 677579
q_{53}^{512}	=	8729924371 238659
$p_{32,1}^{512}$	=	4262526571
$p_{32,2}^{512}$	=	4243343609
$p_{32,3}^{512}$	=	4268376751
$p_{32,4}^{512}$	=	4138507397
$p_{32,5}^{512}$	=	4275976433
$p_{32,6}^{512}$	=	4184037401
$p_{32,7}^{512}$	=	4273997501
$p_{32,8}^{512}$	=	4111394753
$p_{32,9}^{512}$	=	4231168379
$p_{32,10}^{512}$	=	3826886339
$p_{32,11}^{512}$	=	4211145737
$p_{32,12}^{512}$	=	4236261749
$p_{32,13}^{512}$	=	4276081549
$p_{32,14}^{512}$	=	4031143681
$p_{32,15}^{512}$	=	4071011653
$p_{32,16}^{512}$	=	4198142857
q_{32}^{512}	=	4062055801

Algorithm 61 楕円 ElGamal-OAEP 暗号-暗号化

ec_elgamal_oaep_enc($m, k_0, k_1, k_2, a, x_0, y_0, pubx, puby, p, r_1, r_2$)

Input: メッセージ (16 進バイト列):$m < \ell$ (ℓ はベースポイントの位数), OAEP 乱数バイト長 k_0, パディングバイト長 k_1, メッセージバイト長: k_2, 曲線パラメータ: a, ベースポイント: $G = (x_0, y_0)$, 公開鍵: $Y = (pubx, puby)$, 法: p, 楕円 ElGamal 暗号用乱数: $r_1 < \ell$, OAEP 用乱数 (k_0 バイト 16 進バイト列): r_2

Output: 暗号文: $w = (\mathcal{U}, c)$ (w の構成は ec_elgamal_enc と同じ) パディング後の楕円 ElGamal 暗号で暗号化する平文は $k_0 + k_1 + k_2$ バイトになるようにパディングする．)

参考関数： encode(), mgf, int

自作関数： ec_elgamal_enc, xor

関連演習：演習 5.3, 5.4, 5.5

Algorithm 62 楕円 ElGamal-OAEP 暗号-復号 ec_elgamal_oaep_dec($C, k_0, k_1, k_2, a, x, p$)

Input: 暗号文: $C = (\mathcal{U}, c)$, OAEP 乱数バイト長 k_0, パディングバイト長 k_1, メッセージバイト長: k_2, 曲線パラメータ:a, 秘密鍵: x, 法 p

Output: 復号成功時は復号文 (16 進文字列)m を返す．復号失敗時は None を返す

参考関数： mgf, hex, encode(), decode()

自作関数： ec_elgamal_dec, xor

関連演習：演習 5.3, 5.4, 5.5

Algorithm 63 ChaCha20 暗号化 chacha20enc(K, m)

Input: 鍵 (16 進バイト列, 128bit or 256bit)：K, メッセージ (バイト列)：m

Output: 暗号文 (バイト列)：C

関連演習：演習 5.4, 5.5

※この関数はこちらから提供します

Algorithm 64 ハイブリッド暗号-暗号化 (楕円) ec_elg_hybrid_enc($K, m, k_0, k_1, k_2, a, x_0, y_0, pubx, puby, p, r_1, r_2$)

Input: 鍵 (16 進バイト列)：K, 平文 (通常の文字列)：m, OAEP 乱数バイト長: k_0, パディングバイト長: k_1, メッセージバイト長 k_2, 楕円パラメータ (整数)：a, ベースポイントの x, y 座標：(x_0, y_0), ECElGamal 公開鍵 (整数の配列)：$(pubx, puby) \in E(\mathbb{F}_p)$, 法：$p$, 乱数 (整数)：$r_1$(ElGamal 用の乱数), r_2(OAEP 用の乱数, 16 進バイト列)

Output: K の暗号文 (整数): $C_K = (U, c)$，ここで $\mathcal{U} = (x_u, y_u)$, m の暗号文 (16 進文字列): C_m

参考関数： binascii.hexlify, decode(), encode()

自作関数： chacha20enc, ec_elgamal_oaep_enc

関連演習: 演習 5.4, 5.5

Algorithm 65 ハイブリッド暗号-復号 (楕円) ec_elg_hybrid_dec($C_K, C_m, k_0, k_1, k_2, a, x, p$)

Input: K の暗号文 (整数)：$C_K = (U, c)$，ここで $U = (x_u, y_u)$, m の暗号文 (16 進文字列)：C_m, OAEP 乱数バイト長: k_0, パディングバイト長: k_1, メッセージバイト長 k_2, 楕円パラメータ (整数)：a, ECElGamal 秘密鍵 (整数)：x, 法：p

Output: 復号成功時は ChaCha の秘密鍵と復号文のリスト $[K, m]$ (K は 16 進バイト列，m は通常の文字列) を返す．復号失敗時は None を返す

参考関数： binascii.unhexlify, encode(), decode()

自作関数： chacha20enc, ec_elgamal_oaep_dec

関連演習: 演習 5.4, 5.5

表 O.1: 160 ビット素数

p_4	=	73075081 8665451459 5239617144 9964083306 2084344321
g_4	=	2
k_4	=	17235091 9665451459 1234517144 9964083306 2234544321

表 O.2: 157 ビット整数

m_1	=	123 75081 11111 51459 12345 17144 99640 33333 55555 44444
r_1	=	123 45678 91234 56789 12345 67891 23456 78912 34567 89123
r_2	=	113 25678 91234 56789 12345 67891 23456 78912 34567 89123

表 O.3: 1,024 ビット素体

p_1	=	179769313	4862315907	7083915679	3787453197	8602960487	5601170644	4423684197	
		1802161585	1936894783	3795864925	5415021805	6548598050	3646440548	1992391000	5079287700
		3355816639	2295531362	3907650873	5759914822	5748625750	0742530207	7447712589	5509579377
		7842444242	6617334727	6292993876	6870920560	6050270810	8429076929	3201912819	4467627007
ℓ_1	=	89884656	7431157953	8541957839	6893726598	9301480243	7800585322	2211842098	
		5901080792	5968447391	6897932462	7707510902	8274299025	1823220274	0996195500	2539643850
		1677908319	6147765681	1953825436	7879957411	2874312875	0371265103	8723856294	7754789688
		8921222121	3308667363	8146496938	3435460280	3025135405	4214538464	6600956409	7233813503
g_1	=	2							
g_2	=	3							
g_3	=	$p_1 - 4$							
k_1	=	2137858233	0649381417	4696927539	8479291412	7976226092			
		0485325804	8108487164	4119000600	1415349290	3789795134	2787125074	0540083892	2914995687
		6406489134	1265294738	0207374900	6458927006	5399155390	0954617037	8468358492	9753269110
		9084168706	9786303767	0868022176	5537411507	2258330322	9140472735	5023569089	8518749239
k_2	=	5653423359	3039407425	4433017009	9880725338	4559379100			
		7410819211	5937645050	5499824740	0741425627	8367541365	1201024417	3844247923	5536228759
		0802671976	5311636330	8565847410	7994638347	8718740305	4790304127	7983163637	4101856231
		5240667128	4971720906	9816526027	8072896856	6018316270	2665463088	5659143409	3339991506

表 O.4: 1,018 ビットの例

p_2	=	2403928	9861879309	5361017201	6670766295	5798221226	1067999550	3781601699	
		5079150364	3645921628	6515591744	7700825530	1416701179	8966618620	3003394616	3311796796
		7831306501	2883107655	0085578605	5919750577	9586831880	4289389056	9477596130	2883489548
		4201026323	3263427292	3093086705	5336725221	5242913291	4704765012	7249634477	3349897661
ℓ_2	=	235602549	7647385004	6586536675	5585567600	1569676039			
g_2	=	223430576	4178904874	2226942690	6195161767	8328806534	2384853944	5139969853	
		9878215171	3855299851	8623144538	1932618038	7501745856	9632252569	2516975899	9729335545
		4814586334	5982301876	4919047828	9169251991	3262751174	3981332237	7796421797	5243365085
		8754451978	6674834759	3695654408	7470953611	5867164313	8122477155	5933732847	0627806180
k_2	=	40845473	3839582623	3244734395	0544815575	2423456693			

P テストデータ

演習で作成する関数の動作確認用のデータです．演習で作成した関数はまず，以下のデータと合致するか確認してください．

表 O.5: 1,024 ビット素数 p_3, ベースポイント $\mathbb{F}_p \ni g_3(\mathrm{ord}(g_3) = \ell)$, 1,023 ビット乱数 ℓ_3, k_3,r_3 と OAEP 用乱数

p_3	=	135873444 2648880208 2605577702 2778894442 0892880418 2342167852 9552507472
		9079492807 6003157447 1230038836 2344936751 7410681042 9518809553 3153288359 4890525616
		7722940609 8100610136 0991271594 7691038097 6831217259 8134172440 4727297150 3688764137
		5029257447 6036157404 1615027169 1039282461 2195565959 7223865183 7205433613 2155360047
g_3	=	2
ℓ_3	=	67936722 1324440104 1302788851 1389447221 0446440209 1171083926 4776253736
		4539746403 8001578723 5615019418 1172468375 8705340521 4759404776 6576644179 7445262808
		3861470304 9050305068 0495635797 3845519048 8415608629 9067086220 2363648575 1844382068
		7514628723 8018078702 0807513584 5519641230 6097782979 8611932591 8602716806 6077680023
k_3	=	57936722 1324440104 1302788851 1389447221 0446440209 1171083926 4776253736
		4539746403 8001578723 5615019418 1172468375 8705340521 4759404776 6576644179 7445262808
		3861470304 9050305068 0495635797 3845519048 8415608629 9067086220 2363648575 1844382068
		7514628723 8018078702 0807513584 5519641230 6097782979 8611932591 8602716806 6077680023
r_3	=	47936722 1324440104 1302788851 1389447221 0446440209 1171083926 4776253736
		4539746403 8001578723 5615019418 1172468375 8705340521 4759404776 6576644179 7445262808
		3861470304 9050305068 0495635797 3845519048 8415608629 9067086220 2363648575 1844382068
		7514628723 8018078702 0807513584 5519641230 6097782979 8611932591 8602716806 6077680023
r_{oaep_512}	=	84629 4561031082 0848364283 6464927423 0275240543 9834618616 0752812353 2454141256
		1247803792 8634961254 2153256413 5746078706 8970698750 8744563522 7490097466 782894686
r_{oaep_160}	=	78825572461472101619059116246394405469665090789 9

表 O.6: 1,024 ビット素数 p_4, 160 ビット位数のベースポイント $\mathbb{F}_p \ni g_4(\mathrm{ord}(g_4) = \ell_4)$, 158, 160 ビット乱数 k_4,r_4)

p_4	=	141108755 3329747116 0681521826 3958123381 1845882120 6101844813 6404826965
		8894330794 5378916621 8230378522 2285640581 2786036719 0611065605 3750255462 5753148936
		9344062782 5218069782 1880894009 1447658298 3518536032 3706998059 7505163602 4730956156
		7099846439 1197300372 9331477720 0949382303 7167642459 3784527310 9255717090 9406945309
g_4	=	79207621 7877600382 3576323926 9746451281 5520975586 2576344005 0213854787
		2406330846 6725739742 1010854631 6235969173 6492935768 1934505810 8657967082 6832189488
		4518347711 0927089585 9682955591 8931536779 2520597630 8332008486 7242870421 4841371963
		6524425738 8686997557 1374550464 4699780995 3054632950 1856786371 3795563229 9024284915
ℓ_4	=	136211592 3099293242 3699222613 0521234356 1846087883
k_4	=	31647783 2003765415 2477353792 6352571815 0288987267
r_4	=	131647783 2003765415 2477353792 6352571815 0288987267

表 O.7: 小さい有限体の例 ($\mathbb{F}_p$, $\ell \mid (p-1)$, $ord(g) = \ell$)

	$\mathbb{F}_p$	ℓ	g	h
(1)	983	491	2	981
(2)	1187	593	3	1184
(3)	9987	4939	2	5
(4)	10079	5039	3	11

(べき乗算) 演習 1.4, : p : 1024bit 素数, $q = (p-1)/2$: 1023bit 素数, $k \in [1, q]$

表 O.8: 128 ビット整数

K_1(Decimal)	=	184 221027 78699 61103 64334 85937 28750 60126
K_1(Hex)	=	8a 97 ad eb da 4b c6 5e 07 2c 25 a8 70 dd 93 9e

- p = 1 6172 2843 4731 0795 1878 1965 2301 0278 7808 9812 9304 8648 7693 9285 1319 9489 2003 4541 6630 4728 6149 4261 6153 8957 9993 6946 2947 8786 8572 6549 2519 7743 3635 2325 7076 9590 4138 9911 4723 4072 4603 5258 7039 7715 7115 6154 9962 9292 1342 2994 5032 8755 2800 1617 8818 1774 6715 1989 9908 0848 5445 2802 5379 0904 5880 8020 8105 5347 7605 9442 2746 7751 5598 9001 6491 8419 2227
- q = 8086 1421 7365 5397 5939 0982 6150 5139 3904 4906 4652 4324 3846 9642 5659 9744 6001 7270 8315 2364 3074 7130 8076 9478 9996 8473 1473 9393 4286 3274 6259 8871 6817 6162 8538 4795 2069 4955 7361 7036 2301 7629 3519 8857 8557 8077 4981 4646 0671 1497 2516 4377 6400 0808 9409 0887 3357 5994 9954 0424 2722 6401 2689 5452 2940 4010 4052 7673 8802 9721 1373 3875 7799 4500 8245 9209 6113
- g = 3
- k = 10 4571 8704 5037 5446 0826 0300 9247 8128 3536 0745 0058 8985 4016 0509 0290 7977 8978 1827 0556 5319 3714 3909 2550 6161 3097 3670 9929 3941 1661 3083 3827 3198 5660 1144 6487 7867 3546 1877 1349 3977 1924 7348 8299 8510 8204 6399 0623 8414 4311 4889 5630 6610 8294 2534 7290 7226 0408 4630 0068 9528 8051 6877 1990 5030 9111 1158 3719 1715 0969 4344 4779 2040 9435 4942 9418 3981
- $g^q \bmod p = 1$
- $g^k \bmod p$ = 6860 4621 4128 4475 2900 7754 9546 8089 0419 2606 7214 1901 5868 8634 3977 2491 7254 4902 5245 1024 3317 5133 4422 6544 1957 4139 5994 2010 8286 1887 3286 0160 5282 9148 6753 5222 4607 9097 2165 3422 8589 4946 8773 0126 7774 2870 0925 4615 3505 8405 6594 2421 2897 9651 9355 7233 4995 9685 8509 3272 0501 2483 1922 5542 6083 7080 8623 3334 9544 6586 2453 1840 0265 5446 7343 2182 1794

演習 1.2 (拡張ユークリッドの互除法) : p : 1024bit 素数, $a \in [1, p]$, $x, y \in \mathbb{Z}$ s.t. $ax + py = 1$

- p = 1 3587 3444 2648 8802 0826 0557 7702 2778 8944 4208 9288 0418 2342 1678 5295 5250 7472 9079 4928 0760 0315 7447 1230 0388 3623 4493 6751 7410 6810 4295 1880 9553 3153 2883 5948 9052 5616 7722 9406 0981 0061 0136 0991 2715 9476 9103 8097 6831 2172 5981 3417 2440 4727 2971 5036 8876 4137 5029 2574 4760 3615 7404 1615 0271 6910 3928 2461 2195 5659 5972 2386 5183 7205 4336 1321 5536 0047
- a = 6713 3154 0796 5730 2823 6948 9926 0332 0052 0715 2011 2103 4639 8405 3173 0534 7604 3698 6831 0127 5989 0346 4948 5654 7874 9762 5976 7670 2846 1558 6934 7399 4611 7443 7115 7541 8167 5473 7152 1263 3208 5508 1560 6312 4662 5936 5382 5342 0239 0266 3842 1665 8059 4617 2670 1799 8621 8207 3437 9121 1754 4587 0277 7423 3472 7209 9964 4933 4420 4352 1554 4869 9894 7615 5622 2891 4576
- x = 4136 5361 7575 9588 7529 8657 8634 9595 2112 3856 3490 0194 8803 1242 5318 6918 1102 3075 6357 7808 0689 4165 2337 0079 0345 6444 8978 5388 8655 0998 4047 2840 5025 8008 7528 8510 1999 4257 9088 8380 5202 2752 7227 3531 0700 5111 4618 1596 8615 4081 9797 7339 3291 7020 6019 3134 3130 7257 1268 7521 9777 8200 7469 2566 8019 8443 4352 3958 3689 3678 5316 8887 1259 5806 2859 3581 5686
- y = -2043 8042 3228 2252 2661 5588 0729 7356 1954 6917 2801 8434 6172 6071 7421 2341 1210 1321 3211 2542 6799 1874 5191 3517 7336 5280 2843 3329 0575 8599 5443 6193 3533 4390 8551 9851 6269 6112 9068 3992 0346 8982 6466 4559 5144 5176 6467 4484 7750 7018 0528 6971 9153 5999 6300 5376 6508 2671 3782 6803 7451 1704 1762 9703 6698 3072 5766 0591 9207 4062 8729 7840 0429 7005 1982 1629 8705

なお上記の解 (x, y) に対し, $x' = x - pt$, $y = y + at$ $(t \in \mathbb{Z})$ も解になります.

(RSA 暗号) :

- e = 65537
- d = 1314 4330 4793 9274 2815 1332 2215 1110 8704 8800 6794 7255 3856 1657 4001 1065 4335 8350 5126 9502 8610 5916 8266 4665 9981 4975 9408 7235 0366 8627 3602 9419 4220 3190 2814 0588 3458 2135 8616 7270 2832 5663 4412 3191 3951 9923 8284 5289 4484 7397 4065 2028 8349 2432 5486 9474 6970 3072 7830 5986 9110 9663 0577 7683 5942 6701 4527 7055 0634 5806 0568 6885 5006 9304 5782 5343 83513
- p(512 bits) = 1206 2671 6082 2359 2785 5576 1048 8590 6100 8357 3423 8736 6140 1669 6310 1629 7538 2086 8976 8836 8639 0733 7656 7629 6141 5197 6813 3657 6681 3326 0302 5235 6115 3684 2905 1500 5877 3339083
- q(512 bits) = 1257 5048 0619 9176 2258 1325 1545 6708 8066 2668 8217 7513 6218 9646 1515 4315 9731 4091 0180 3505 6460 9356 3622 4797 3606 1395 5798 7683 1444 7862 1639 0463 8882 4332 3252 3292 3770 8577867

表 O.9: 512 ビットの素数の組 - 1 (p_1, q_1:512-bit, $n_1 = p_1 \times q_1$,e_1: 32-bit, $gcd(a_1, n_1) = 1$, $|k_1| = |n_1|, k_1 < n_1$)

p_1	=	1061381230 1896775148 8299426371 0298546241 2287794185 3877384672 4576275961
		8509602736 6051956410 5416621657 1968217361 0615476408 9255703974 0729186347 7131247892 44883
q_1	=	1157303566 3327195853 8442388435 4286933387 6114938989 4754990017 0830535765
		0732178608 3018482881 9928489991 4248138795 2864065457 8560196897 1231714755 3772482655 09893
n_1	=	1228340282 9371229672 0762660347 5421461646 5209348167 0201947376 3249930095
		5769193300 3435610085 9768987586 8733831669 2229094216 9535836110 2860006897 5461290111
		5941975224 8671316105 8298860890 9985552715 9045423477 4552183738 8939563882 4495933403
		4615659391 1239817569 4572958404 0032409298 3089026564 0578775821 4276110040 436127519
e_{1-1}	=	576545171
d_{1-1}	=	3240119253 6282284985 2937640680 1561754108 8107068689 9129710462 9180360828
		1162104458 5516888062 8895806434 8844691177 5283356386 5361311664 1193673091 5287548583
		9148644717 9012634626 9963262730 4171809163 9197392417 5886127177 0064850130 8653623794
		2258743958 0303085709 8539375261 9828361163 7092685818 0336861819 3220918317 49480363
e_{1-2}	=	4170652133
d_{1-2}	=	4400877548 0361668477 6773407914 8433938575 8776325274 4162519273 4300666321
		9106026995 1887543716 4590262996 6377504884 3670717399 1430338401 4636764437 7666294961
		3111452313 0465800850 7271486013 1440747661 6710751735 2605176547 9187252258 8693284457
		7558783953 8673210507 4595941943 5464851966 3281747671 7970541520 5792541224 47719913
a_1	=	5517136991 0051437990 4126907809 7691886400 4594400468 0579369956 2639916993
		5459645309 2936382738 3901143322 0903844032 9782230806 5100196316 5983148138 9038664650
		7034968771 7707483416 9602476092 5769897674 8740151772 0715057633 3584463725 8758436116
		1658474605 4957823806 8993246164 2739285921 0459388248 9289102848 3858898273 48866771
k_1	=	4790803212 4466708640 5082821139 6166041485 1281601799 7379478442 0229758404
		2029548314 4737351219 2239347898 9236470550 2874127684 5024428026 2453424885 2004107095
		2254994809 2068728231 9494673491 3741339098 4868117642 2885271685 2146572363 2643962930
		7682906468 8036543959 7335446193 2808904382 8142255591 0552145824 5050864185 30684390

表 O.10: 512 ビットの素数の組 - 2 (p_2, q_2:512-bit, $n_2 = p_2 \times q_2$, e_2:64-bit, $gcd(a_2, n_2) = 1$, $|k_2| = |n_2|, k_2 < n_2$)

p_2	=	1185308576 3133505990 2862012693 6392113175 5361282207 7718064542 1682390646 3045956110 4040185181 1887198856 1332197770 0151754580 2509656687 5663947404 2315970486 49687
q_2	=	1324147987 9786700069 5376730134 0228371346 1303895018 6387250437 1572833335 1182535991 7916987304 5490567956 9343041660 9104011044 9999896454 1968688709 3605287712 69943
n_2	=	1569523966 4591850295 5193834718 7763885068 5739921301 9211056913 5670254477 8362395342 4981098100 3203717970 3739071352 7200065108 0485043023 4788150574 2461209041 7644696580 5210049306 3095556772 1591369643 3784788243 3273567579 1585286253 0141525314 5239724411 4433931271 4812961056 4195773940 5631844445 8695333074 1471865748 319457841
e_1	=	16906953396398285955
d_1	=	7243120132 7459355910 4718233668 1310111949 9609639791 1006981852 1891729612 3904894886 3282836141 9263587738 2223140871 2979119342 9476334875 0948968199 9208855556 6997541879 1136760295 4918028434 9351662954 6984571593 4583725830 1381051407 1426513902 7719788715 2694674835 9902661782 3666098202 9008713958 6496712874 3435198316 095145
a_2	=	8273172872 5194908289 2236176663 6745746952 8377629992 3444868922 2799947380 8545734393 6014816382 1388375508 2805006377 0654397886 6124357334 0349538848 2362613359 2304202545 9102737381 9491870149 9926161168 4851439601 1676206647 5445546988 0526500359 5439269501 5391883077 9325957499 2519364080 1344009225 7831969264 5061721961 99532825
k_2	=	1510611613 3679978202 7909948821 5310586749 0950348922 5543744191 8572391793 0120874747 9918027528 6680672639 8883021452 9226358772 3560597448 1819755657 0073572539 5153114171 7413494388 4071414804 6233931768 3191431043 6415866358 5331414086 2052956067 9609405497 0175524650 6679069530 0301408503 5728199079 1590484047 2327542139 70906224
r_2	=	1113587397 5794446838 4163550742 2549378044 1326065243 8479685343 4545266502 4892484614 0194858368 0152878868 2775275081 7988170103 5410791946 6010736321 8574957085 62039

- n(1024 bits) = 1516 8867 5229 4351 4454 5762 6307 6551 3843 8409 0531 8881 4264 3341 0993 3178 2940 7927 7610 5818 0842 6192 4822 6698 6356 8833 9185 9396 4183 7133 8494 7350 9537 1904 9387 6558 1690 4550 7678 5281 8023 3325 1445 8438 2935 8013 4439 8779 8968 7034 2021 4994 2241 1253 8038 7654 9156 1264 5588 1007 4903 3611 6762 8966 4911 1675 3399 8425 5409 2820 3432 8119 9277 1620 6914 8998 75961
- *message*(1020 bits) = 9380 0268 1455 1100 0395 1680 5740 3795 8517 3246 9117 6630 2786 1995 5375 3827 8130 2089 7087 6198 1337 1981 7277 4031 4623 2740 5830 0468 4100 7656 3368 6228 4802 8671 2159 1758 3366 1085 1174 7268 6236 3297 6213 3266 0034 9018 3723 5938 1406 7167 0583 7687 6777 3503 1671 0176 7574 8670 5474 3880 6960 5213 0236 5172 8708 6261 5423 8459 3797 2545 6067 7419 9244 0521 3408 8576 557
- *cipher* = 1239 6790 8186 5578 9642 5405 9623 5486 6844 0982 3142 1600 8174 4953 5819 6363 7038 6369 1701 2916 6369 4436 9885 3804 5422 6106 9823 1039 1218 7564 9076 0912 3506 8030 6321 1324 0401 7484 2792 7534 2191 9960 7275 1468 8574 6587 0013 0143 2634 4452 7712 9764 4751 4197 5020 8630 9446 7198 5582 8540 5642 9300 4223 4783 2474 1826 9158 6957 7453 0854 3340 9322 2649 0751 7184 1991 15712

(RSA-OAEP 暗号) :

- n(1024 bits) = 1319 9664 9081 9883 0981 5009 4122 3160 6409 9988 7200 8467 2203 5670 4480 6582 0632 9986 0177 4142 5592 7395 9878 4901 1474 9026 2698 2832 6520 2147 5938 1792 6551 9984 5793 6217 7299 8404 3905 4838 0689 8514 0623 3864 9654 3388 2904 5552 6885 8728 5851 6219 4605 3376 3923 1268 0578 7956 9268 2905 5995 9042 2046 7205 8771 0762 9271 3074 0460 4424 3853 3124 0538 4889 8103 7901 24491
- e = 17
- d = 1164 6763 1542 9308 6160 1478 8931 4553 5067 6460 6353 6882 8414 9120 9835 8748 8793 8222 9568 3066 9640 6525 8716 3148 0713 1493 7674 9558 2223 7189 0533 6875 8722 3515 8053 1956 8205 7415 6366 8437 3315 6436 1630 9716 4007 9679 0490 0300 7752 2365 8035 4323 3292 3992 4506 4743 9719 6947 3468 3045 3671 4979 0102 1988 1003 3962 3586 1837 0829 4418 9542 5705 7285 2387 4962 1070 52993
- r(160 bits) = 976172171425244805716717114288565309992579413391

表 O.11: 512 ビットの素数の組 - 3 (p_3, q_3:512-bit, $n_3 = p_3 \times q_3$, e_3:96-bit, $gcd(a_3, n_3) = 1$, $|k_3| = |n_3|, k_3 < n_3$)

p_3	=	9599724472 0774591709 3840929535 1348996733 2868455933 8072367828 7015797869 3765829946 9871755715 9063112480 4878040576 6720230511 2706638969 5431443647 0919347304 8347
q_3	=	1215903644 9739250060 6096309888 3622687320 0838844190 0239984788 8102552041 1437687569 8709068905 7956108167 0068532247 4247654605 6472174913 0863519839 4434242804 86653
n_3	=	1167233997 6344370570 8568459695 4891354662 7640956374 5323740425 7136259107 5486777858 4330574304 2743079604 8925073072 9776954865 8802927423 4069501543 2095205032 8146497566 5678480630 4411866553 7806278220 5022188868 5962926806 0292775049 8387055511 2017587086 2496314002 4150305756 4836383137 7331056181 5698080034 2118804904 557212591
e_3	=	394203512154731469664702999 31
d_3	=	1393788582 3096711794 5389439208 3928985141 9878930943 6919363645 3686076511 5415925578 7954214246 2456048851 5326679482 8769294798 3386170094 6333089661 0365306575 7028177682 1642299039 2185974956 8500878836 6679109934 8267920306 0551049688 3010641915 8921960476 8948555056 8282679006 4888550118 6860360638 8727010900 6453180955 69480383
a_3	=	8767142442 1652388950 6426896438 9619900644 5185897064 6380740588 9616808859 9903264970 8622796868 4417061150 6580163120 4547948483 9313426559 8192406101 5335286113 5763140223 4248276942 3026303535 6103843587 6081389801 0277191736 2219512572 2971702016 0175850000 5347742292 5129871008 0476404456 6077962563 3162226643 5475737467 43342095
k_3	=	8775639933 8693425772 3588635722 2170288853 0279473215 3150830915 4201012623 1959240646 1540908092 6736780704 6443318462 9189948399 7539515433 7145392059 2734621419 0589329251 9252249862 1356874120 8948106741 1690879810 9795740849 4921032742 8566805387 3689318129 5101313209 0781006976 4941637729 7019783074 2959418297 8513273171 579104
r_3	=	9252603648 9950447750 1920424993 8037126999 6940321772 1673727433 0948174216 3968887187 9856045244 6856381282 8779931444 2168464249 3763696467 3275215265 8852560434 6551

表 O.12: 256 ビット，341 ビット，342 ビットの素数

$p_{256,1}$	=	9941433401 5203114960 9526738843 3997810424 3180737243 6577343921 1785285298 5243877
$p_{256,2}$	=	11188162371 5119180788 5222912270 5022208805 8587137547 7054079219 3728086967 85333197
$p_{256,3}$	=	1024251801 4062364432 9684971746 6346801594 2762407940 7752936443 1626609164 33842869
$p_{256,4}$	=	9699716940 1433342338 0020821998 2175684873 2698444814 9706418755 6601377921 5176849
$p_{256,5}$	=	1121860325 9323072578 9036535291 0752951628 9143173589 0576369178 5637290233 96691719
$p_{256,6}$	=	9859465706 2062438738 9997689888 8105295949 9441773971 2724620644 1568882854 3936849
$p_{341,1}$	=	3891959031 0264267341 3700056810 8937900354 1022208879 8978305407 4296698666 3269387607 6331389547 1080407684 987
$p_{341,2}$	=	4427097703 0367742933 4184751354 2455648045 4499812140 3253376000 5166058967 8024194060 9945892334 5179373991 057
$p_{342,3}$	=	8177691406 2846101635 6186958356 4507224208 5812741772 4007701629 2017078599 5286020516 0221865297 6276439249 321
$p_{341,3}$	=	3921014898 8611599345 1163009455 0579065995 8724470558 5260627058 0083365821 3687560719 0150346213 9246480070 829
$p_{342,2}$	=	7269804724 8172173272 9811908562 0946690548 2882694064 6216307713 4902269530 0339558444 9252380964 1760407629 981

表 O.14: 1024 ビット RSA 公開鍵と暗号文

e	=	3
$n_{1,1024}$	=	1246424028 1595120124 2747411048 4548817765 5482865688 7198056462 2526090307 4883138802 6169367886 0579097227 2349519097 8994958671 2501040015 0427991456 9506822012 5866775284 7150923689 3066329423 9894595245 0596032798 7094888333 2207128158 6978694758 1477630705 5697445656 0883416827 6981860601 4379983270 1591647584 5782191229 582548671
$n_{2,1024}$	=	1237863469 0741288296 4224724042 5419819269 8133253860 0075571606 9588423881 8598472030 2656988531 5509973507 7662798128 7953491799 1585529667 1584860366 3159475522 4552690063 9500214460 5702485087 4617781911 7802710430 3037933642 0260049069 4377244926 5720354131 4818561339 1009125141 4421228217 3278068615 6155013399 4126667410 810563519
$n_{3,1024}$	=	1064418998 9059955334 7630116702 4849141577 1282674064 9823955954 4338235351 3660437238 7646213947 4028499878 3804836338 0853826216 7484206192 1832754289 4536428751 7842431103 4086734609 3680931513 8345730774 2663068860 6825876526 6265585620 8539941172 9642335117 5599110065 0502643328 4012784100 5067880507 6279630607 3185853145 846863439
c_1	=	2100950624 9788720465 1237831931 4715382676 7557898188 5699760993 1718957066 3494538606 8910263847 5923880354 1740356375 2411464035 0716587886 1319008822 2120540910 3570740523 2499036650 4845919640 7190262181 1364677555 5623045502 1232905215 0787503325 7137330502 2074867401 0826106297 8156877512 2929190962 6926911763 3392467558 21527505
c_2	=	4556366548 5889359707 4708385715 5607402658 7199909856 3901415900 6675693178 1719919662 5451177626 4689372837 1510284791 5497932211 4334870964 3853818922 4419713601 0555631921 3962946763 3268925800 6266106198 1902736550 1020905726 5425967389 1898635555 7002192580 8597370512 1245127078 2318433520 4110576710 4333960048 9978366319 74780954
c_3	=	6839735635 7072083459 8789613303 2691362866 1403009598 6242426683 4581313167 5542210793 5495017335 0823604850 0748483820 5378347211 6780014827 8125841553 2899328876 2973217361 1418702231 4133204950 7480764848 6904063885 7260414209 6207092226 3405917155 4458251955 4279510671 7051600255 0166966933 2096655158 5876408888 2784613834 36211255

- *message*(128 bits) = 282081456775998934689133617587168685129
- *cipher* = 1287 0107 4370 3005 4101 4829 3473 8564 5297 5385 3252 4014 8798 3564 6041 8427 8059 2622 7601 7913 5505 2449 2861 9141 9379 9268 2181 8698 2531 0459 4013 9614 8196 5788 5439 9251 3353 9694 0841 4762 9392 2557 4343 5568 7615 8351 8349 8057 7594 8578 8756 7504 8670 6621 3670 3602 0679 0489 8189 4026 0911 0471 9058 3254 6285 8524 4432 0269 5189 6761 2457 4483 7776 0141 0937 9749 7173

(Schnorr 署名) :

- p = 1358 7344 4264 8880 2082 6055 7770 2277 8894 4420 8928 8041 8234 2167 8529 5525 0747 2907 9492 8076 0031 5744 7123 0038 8362 3449 3675 1741 0681 0429 5188 0955 3315 3288 3594 8905 2561 6772 2940 6098 1006 1013 6099 1271 5947 6910 3809 7683 1217 2598 1341 7244 0472 7297 1503 6887 6413 7502 9257 4476 0361 5740 4161 5027 1691 0392 8246 1219 5565 9597 2238 6518 3720 5433 6132 1553 60047
- g = 2
- l = 6793 6722 1324 4401 0413 0278 8851 1389 4472 2104 4644 0209 1171 0839 2647 7625 3736 4539 7464 0380 0157 8723 5615 0194 1811 7246 8375 8705 3405 2147 5940 4776 6576 6441 7974 4526 2808 3861 4703 0490 5030 5068 0495 6357 9738 4551 9048 8415 6086 2990 6708 6220 2363 6485 7518 4438 2068 7514 6287 2380 1807 8702 0807 5135 8455 1964 1230 6097 7829 7986 1193 2591 8602 7168 0660 7768 0023
- x = 1125
- y = 8150 9569 2124 5132 5670 0454 7633 8272 6116 5996 1628 1801 8583 3136 0347 1907 8973 7253 4802 3475 4607 1254 4902 9027 0092 1566 1023 1775 4236 0489 8637 9459 8178 1073 4296 3027 4360 3052 7214 2442 8910 0510 5234 7722 9399 2396 7574 6985 2512 7210 8023 2954 0138 8850 9840 7681 0976 7087 4261 1327 0442 1027 1095 9097 4255 4546 0461 8404 1450 5047 6326 3176 7559 9120 4561 2148 0925

表 O.15: p : 1024 ビット素体 ($(p-1)/2$: 素数) と 128,256,512,768,1024 ビットの元 (その 1)

p_1	=	8988465674 3115795386 4652595394 5123668089 8848947115 3286367150 4057886633 7902750481
		5663542386 6120376801 0560056939 9356966788 2939488440 7208311246 4237153197 3706218888
		3946712432 7426381511 0980062304 7059726541 4760425028 8441907534 1171231440 7369565552
		7041361858 1675255342 2931491199 7362296923 9858152417 6781648121 13740223
$g_{1,128}$	=	2552117751 9070384759 7530955573 826162347
$g_{1,256}$	=	8684406692 7987146567 6782387565 1593088995 2488499230 4230295931 8800593484 7271147
$g_{1,512}$	=	6703903964 9712985497 8701249910 2923063739 6829102961 9668886178 0721860882 0150368892
		8049017446 5278875284 8300246170 0109663569 3909502908 4762428253 2031682019 4359
$g_{1,768}$	=	1164388569 2255317013 6173461634 6876916442 6645128375 2245835428 9028519538 2145147960
		7255608874 9218584432 4878640038 6897578427 0449277426 4638343550 2693722031 0959401380
		1431063138 0433259038 8996081885 3196698010 0327809929 8133116068 7893683471 79
$g_{1,1024}$	=	8988465674 3115795386 4652595394 5123668089 8848947115 3286367150 4057886633 7902750869
		6958773138 3843755525 5432172565 5745109003 3367405855 9993074255 9302280579 0866981827
		4596257237 0038316637 2993914345 7307393904 9634469093 1157838013 4754021809 5687344708
		8871699796 1490511695 5592352172 3079247292 1099216175 3478101745 73488207

表 O.16: p : 1024 ビット素体 ($(p-1)/2$: 素数) と 128, 256, 512, 768, 1024 ビットの元 (その 2)

p_2	=	1348269851 1467369307 9697889309 1768550213 4827342067 2992955072 5608682995 0685412572
		2349531357 9918056520 1584008540 9903545018 2440923266 1081246686 9635572979 6055932833
		2592006864 9113957226 6647009345 7058958981 2214063754 3266286130 1175684716 1105434832
		9056204278 7251288301 3439723679 9604344538 5978722862 6517247218 169050179
$g_{2,128}$	=	1701411834 6046923176 8580791863 303232283
$g_{2,256}$	=	5789604461 8658097711 7854925043 4395392697 5274699741 2204831921 6661138833 3043903
$g_{2,512}$	=	1173183193 8699772462 1272718734 3011536154 4445093018 3442055081 1626325654 3526314469
		3967908773 2709146407 0213786563 8210285644 8170028802 4492743180 6255034292 94719
$g_{2,768}$	=	7762590461 5035446757 4489744231 2512776284 4300855834 8305569526 0190130254 7634320185
		0973369147 0114228130 0024207119 4693015968 9063824154 3513075716 5865401550 6642323711
		1859962478 7279773807 6833618935 6207375228 6041172683 8663514724 2081213868 3
$g_{2,1024}$	=	8988465674 3115795386 4652595394 5123668089 8848947115 3286367150 4057886633 7902751257
		8254003890 1567134250 0304288191 2133251218 3795323271 2777837265 4367407960 8026403985
		7315859444 1654511513 5187181774 0075695447 8584579380 1517624120 6572782104 9693276738
		7003840586 1449063025 6550426820 4737525528 1030836284 0296372329 25182547

- r = 4382 4410 6346 5217 0727 5070 0228 1711 538602
- *message* = "This is an example of Schnorr signature."
- e = 107008017532587363415974788040473219337168884659 9
- v = 120384019724165166284035101762605122454543166396247 7

表 O.17: $n = pq$: 1024 ビット剰余環 ($p, q, (p-1)/2, (q-1)/2$ は素数) と 128,256,512,768,1024 ビットの元

p_1	=	1173183193 8699772462 1272718734 3011536154 4445093018 3442055081 1626325654 3526314353 6047016400 1089604049 9228777875 9131752944 9703464746 0453248703 1294428803 95983
q_1	=	1089384394 3078360143 4038953110 4224997857 6984729231 3196194003 9367302393 3274434756 9186515228 6726060903 4998150884 7765199163 1867502978 4706588081 4773398176 73279
n_1	=	1278047463 0661777156 5130290907 6572271556 5305084667 9607905329 1983230755 7420547334 0977159933 0963991076 4001508096 1471068715 2105458512 6649931755 3508720223 7375851935 9024330520 1232849812 3855614623 9684904495 5738773699 7331687379 3628315859 6706058528 2374756062 8380733094 0868908971 4058628589 4958057351 3082585825 238038257
a_{128}	=	2552117751 9070384763 4424443721 245264803
a_{256}	=	5789604461 8658097718 0625942397 3063469081 1064122948 8868993314 1554399978 6647923
a_{512}	=	6703903964 9712985497 8701249910 2923063779 0849164925 9116807405 9761961025 6288419690 1976063991 1946823578 2342718002 7423083692 5875051909 7304113745 3916897497 7699
a_{768}	=	7762590461 5035446757 4489744231 2512776284 4300855834 8305569526 0190130254 7634318844 3165439204 4143232389 7524386534 8565576005 0919864165 3661947476 2243781134 3956493414 9795345709 2907345638 6239242514 8345903233 7150007697 9077070121 1247732673 9
a_{1024}	=	8988465674 3115795386 4652595394 5123668089 8848947115 3286367150 4057886633 7902751257 8254003890 1567134250 0304288191 2133251218 3795323271 2777837265 4367407960 8025063204 9385916847 0658771263 5366597161 2635731633 4624590395 0389383758 4952365836 3862980532 2387071148 9020893966 2174006034 3265530638 9865850325 3851769245 89179087

Q 数値例 (楕円曲線)

演習問題に利用する数値例を列挙する.

表 Q.1: 160 ビット上の楕円曲線

p	=	80181938 5093403524 9050147795 4289294831 0645897957
a	=	80181938 5093403524 9050147795 4289294831 0645897954
b	=	23756723 3982590907 1668366836 5552239880 4119025399
g_x	=	2
g_y	=	6799365 0243915087 0056418043 5573536471 6595191047
ℓ	=	80181938 5093403524 9050156749 8657352984 4218487823
x	=	32458174 1857968069 9492994192 0044936372 6684074707
y_x(公開鍵)	=	75504173 0279269073 5170715474 5985483058 4155909604
y_y	=	17949735 9686951748 6300514777 7378859451 8104529673
r	=	47405419 8185012694 8173556040 4166191195 1037771404

表 Q.2: NIST-256 楕円曲線の秘密鍵，公開鍵，乱数

x(秘密鍵)	=	2707776 3127661076 5074772615 1299710344 7780673270 7969708492 5547739929 2372780398
y_x(公開鍵)	=	8454681 2694622314 2346456520 2281597025 8631217784 1355992870 4964864498 3010139605
y_y	=	6207476 7048292746 1797472358 9868698042 2491711536 3933406028 6887757774 8039401698
r_{256}	=	9494296 3219160478 6456164568 4742594911 8405414797 8018139631 4526502220 4180576071

表 Q.3: 16 ビット整数

r_1(Decimal)	=	64123
r_1(Hex)	=	fa7b
r_2(Decimal)	=	58419
r_2(Hex)	=	e433
r_{64}(Decimal)	=	14454 42163 20457 86768
r_{64}(Hex)	=	c 8986b e43bc fa690

表 Q.4: 160 ビット上の楕円曲線

p	=	146150162 4496790265 1454485899 2078549371 7258890819
a	=	0
b	=	3
g_x	=	140061329 4960275999 6389273992 8043668197 5015170626
g_y	=	52512599 2262199324 1000231658 5044230492 6193669571
ℓ	=	146150162 4496790265 1454473809 9497118849 9300027613
x(秘密鍵)	=	72694763 2739520350 8591436465 6412457684 5992007282
y_x(公開鍵)	=	54357942 2704623691 1185830752 9615567255 2537111459
y_y	=	132024754 0342186013 9660010544 1342390825 5769832172
r_1	=	47405419 8185012694 8173556040 4166191195 1037771404

R　テストデータ (楕円曲線)

演習で作成する関数の動作確認用のデータです．演習で作成した関数はまず，以下のデータと合致するか確認してください．

演習 2.1 (ℚ 上のアファイン加法)：$E : y^2 = x^3 + x + 3, a = 1, P = (-1, 1)$

- $2P = (6, -15), \lambda = 2$
- $4P = 2(2P) = (1081/900, -65771/27000), \lambda = -109/30$
- $3P = 2P + P = (11/49, 617/343), \lambda = -16/7$
- $4P = 3P + P, \lambda = 137/210$

演習 2.4 (ℚ 上の Jacobian 加法)：$E : y^2 = x^3 + 4x + 6, a = 4, P = (-1, 1, 1)$.
Z が非負最小となる $(X, Y, Z) \in \mathbb{Z}^3$ を最終的な計算結果とします．

- $2P = (57, -435, 2)$
- $3P = P + 2P = (-3856, 34805696, 244) = (-241, 543839, 61)$

同様に $4P$ の計算結果は:

- $4P = 2(2P) = (9969121, 39023065169, -1740)$
- $4P = 3P + P = (593521587856, 566879638727986496, -424560) = (9969121, -39023065169, 1740)$

演習 2.6 ($\mathbb{F}_p$ 上のアファイン加法)：$p = 257, E : y^2 = x^3 + 4x + 6, a = 4, P = (256, 1)$

- $2P = (207, 42), \lambda = 152$
- $4P = 2(2P) = (142, 25), \lambda = 175$
- $3P = 2P + P = (184, 201), \lambda = 167$
- $4P = 3P + P, \lambda = 140$