2023/07/07-ver5

第 23 回 セキュリティPBL 特論 I/先進セキュリティPBL -離散対数問題と関連暗号技術-

大阪大学大学院 工学研究科

宮地 充子

Email:miyaji@comm.eng.osaka-u.ac.jp

修得知識： 離散数学 (素数, 体，有限体, 平方剰余,
Python: 図表示，ヤコビ記号，有限体の演算，合同式，平方根,
テキスト: **「代数学から学ぶ暗号理論」（日本評論社）** 2-7-10,13 章

はじめに

本 PBL 演習の前身であるサマースクールは 2000 年に第一回を開催し，2016 年に第 16 回目の開催を行いました．その後，2017 年度から，全国の大学院生を受け入れる PBL 演習に変化し，2018 年度から社会人，大学院生，学部生を受け入れるダイバーシティの高い PBL に発展してます．2023 年は $23 = 1 * 23$ 回目 (素数回) になります．これまでの延べ参加人数は 378 名を数えます．本演習ですが，毎年新しい課題を取り入れ，毎年受講しても新たなことが学べる演習となっています．今年受講される皆さんも数学の楽しさ，セキュリティの面白さを体感して頂ければと思います．今年は flipping を導入し，事前課題を解くことで，より理解の高い演習を実施できると考えています．なお，サマースクールは以下の目的のために，宮地研究室が独自に行ってきた公開講座です．

- 広く一般の方々に情報セキュリティの研究や技術の一部を紹介することで，セキュリティの重要性を伝えること．
- 情報セキュリティ分野の研究や開発に携わる技術者・研究者の人材育成．
- 数学がどのようにセキュリティに応用されるかを教育することで,「数学は利用できる科学（技術)」であることを伝えること．
- 広く一般の方々に当研究室の研究内容を伝えると共に研究室メンバのレベルアップを図ること．

これからも PBL として開催しますので，コメント，サポートをよろしくお願いします．

本講座の目的は数学の理論及び暗号の理論を演習を通じて学習するということが目的です．このため，理論の説明後，テキストに記載された演習を行いますが，テキストには演習が若干多めに記載されています．各演習で必須のものを記載しますので，それ以外は終わった人が解く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．実装する関数 API は E 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 F 章に記載されています．なお，作成した関数は F 章に実装したプログラムの正しさを検証するサンプルデータを掲載しています．各自，実装後，サンプルデータでプログラムの正しさを確認してください．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．

- 解析 (analysis): 実験結果と理論値から考察します，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．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，提出して下さい．解析には方式提案の課題も含まれます．方式提案については，word や tex などを利用して作成してください．

  python ソースの提出は jupyter notebook のファイルを提出してください．ファイル名は，以下に沿ってください．解析の提出物は，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．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．なお,「問」「演習」「解析」（プログラムの結果）の提出物のファイル名の演習番号のアルファベットはそれぞれ P，I，A の頭文字をつけて下さい．

  大学名頭文字_名前_演習番号 (大学名頭文字：JAIST → J, ProSec → P, 阪大工学部通信コース/工学研究科情通 → OEC, 阪大工学部電子コース/工学研究科量子 → OEE, 阪大工学部情シスコース/情報科学研究科 → OI, 石川高専 → IK,)
  例: 阪大工学部の宮地が第一回の問を提出する場合: OE_miyaji_P1

また，演習コースをレベルから advanced, standard, biginner の 3 コースに分けいます． 講義は全て行いますが，演習は各受講者のスキル及び好みに応じて進めるこが可能です．

advanced コース 演習 1.1, 演習 1.2, 解析 $1^{\dagger}$, 演習 1.3 → 演習 2.1, 解析 $4^{\dagger}$, 演習 $3.3^{\dagger}$, 解析 7，問 $2.2^{\dagger}$
→ 演習 $3.1^{\dagger}$，演習 $3.2^{\dagger}$, 解析 $6^{\dagger}$ → 解析 5 → 問 4.1 → 問 5.1, 演習 $5.1^{\dagger}$, 解析 8
→ 演習 6.1，6.2，演習 6.3, 演習 6.4, 解析 9 → チーム対抗戦

Standard コース 演習 1.1, 演習 1.3, 解析 4, → 演習 $3.3^{\dagger}$, 解析 7, 問 $2.2^{\dagger}$ → 演習 $3.1^{\dagger}$，演習 $3.2^{\dagger}$, 解析 6
→ 問 4.1 → 問 5.1, 問 5.2, 演習 $5.1^{\dagger}$, 解析 $8^{\dagger}$ → 演習 6.1, 演習 6.2 → (チーム対抗戦)

beginner コース 演習 1.1, 演習 1.3 → 演習 2.1，問 $2.2^{\dagger}$ → 問 3.1，演習 $3.2^{\dagger}$
→ 問 5.1, 演習 $5.1^{\dagger}$ → 演習 6.1, 6.2 → (チーム対抗戦)

グループプレゼン（解析 1(ECD の上限），解析 4(Fermat 法の上限），解析 6(準同型），解析 7（DH 鍵共有法），解析 8（署名）から 2 つ＋対抗戦

目 次

1 講義 1(教科書 2，3 章)

本講義の目的の 1 つは有限体の基本演算である乗法とその逆元演算の除法について理解することです．特に，逆元演算は暗号の重要な構成要素です．暗号解読の一つであるサイドチャネル攻撃は入力ごとに計算量が変わる事を利用します．逆元演算を用いて，入力ごとに計算量が違う事例を学習します．暗号では n ビットは n 個の 0,1 の列を意味し，$0 = \underbrace{0\cdots0}_{160}$ も 160 ビットになります．安全な暗号とは，どのような n ビットの秘密情報に対しても，その秘密情報の解読に掛かる時間が n ビットの全数探索に相当することを意味します．

1.1 群・環・体

集合 G が群であるとは，次の 4 つの条件を満たされていることである．

(G0) G の任意の 2 元 a, b に対して，$ab \in G$ が定義されている．

(G1) (結合律) $(ab)c = a(bc)$

(G2) (単位元の存在) $1 \in G$ で，任意の $a \in G$ に対して，$a1 = 1a = a$ となるものが存在する

(G3) (逆元の存在) 各 $a \in G$ について，$aa^{-1} = a^{-1}a = 1$ となる $a^{-1} \in G$ が存在する．

上述の 4 つの条件を群の公理とよぶ．群 G において，

(G4) (可換律) $ab = ba$

が満たされているとき，G は可換群，またはアーベル群とよぶ．

群 G の元の個数 $\#G$ を G の位数 (order) という．また，群 G の元 a をとるとき， a のべき全体のなす集合を $\langle a\rangle$ と表す． $\langle a\rangle$ は，明らかに G の部分群になる．

$$\langle a\rangle = \{a^k | k \in \mathbb{Z}\}$$

部分群 $\langle a\rangle$ の位数を a の位数という．a の位数は，$a^k = 1$ となる最小の正整数 k （または，無限大) に等しい．

加法と乗法という二つの演算の定義された集合 R が環であるとは，R が次の 3 つの条件をみたすことである．

(R1) R の加法について可換群になる．(これを加群とよぶ)

(R2) (乗法の結合法則) $(ab)c = a(bc)$

(R3) (分配法則)

$$\begin{cases} a(b+c) &= ab + ac \\ (a+b)c &= ac + bc \end{cases}$$

(R4) (単位元の存在) R の 0 と異なる元 1 で，R の任意の元 x に対して，$1x = x1 = x$ を満たすものが存在する．

集合 R が上の性質に加えて，
(R5) (乗法の交換法則) $ab = ba$
を満たすとき，R は**可換環**とよぶ．

例 1 整数の集合 $\mathbb{Z}$ は可換環である．

可換環 R において，零元 0 以外の任意の元が可逆元であるとき，R は体であるという．有理数全体の集合 $\mathbb{Q}$, 実数全体の集合 $\mathbb{R}$, 複素数全体の集合 $\mathbb{C}$ は，すべて体である

1.2 有限体の演算

正の整数 m を一つとる．整数 a, b の差 $a-b$ が，この整数 m の倍数であるとき，a と b は法 m に関して合同であるといい，記号で，

$$a \equiv b \pmod{m}$$

と表す．この式を合同式という．任意の整数 a を m で割ったときの余りを r とすると，$0 \le r \le m-1$ であるから，m を法とした剰余類は m 個存在する．

例 2 3 を法とした剰余類は，以下の 3 個である．

$$\begin{aligned}
\overline{0} &= 3\mathbb{Z} &&= \{\cdots, -3, 0, 3, \cdots\} \\
\overline{1} &= 1+3\mathbb{Z} &&= \{\cdots, -2, 1, 4, \cdots\} \\
\overline{2} &= 2+3\mathbb{Z} &&= \{\cdots, -1, 2, 5, \cdots\}
\end{aligned}$$

各剰余類から選んだ以下の代表元の集合を用いて，

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, \cdots, m-1\}$$

と表すことにする．このとき，$\mathbb{Z}/m\mathbb{Z}$ は，以下の演算で環になる．$\mathbb{Z}/m\mathbb{Z} \ni a, b$ に対して，

$$\begin{aligned}
a \cdot b &\equiv ab \pmod{m} \\
a + b &\equiv a + b \pmod{m}
\end{aligned}$$

と定義する．このとき，乗法に関する単位元は，$1 + m\mathbb{Z}$ となり，加法に関する零元は，$m\mathbb{Z}$ となる．

例 3 $\mathbb{Z}/3\mathbb{Z}$ における和と積.

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

$\times$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

例 4 $\mathbb{Z}/4\mathbb{Z}$ 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

$\times$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

事前問 1 例 *3, 4* を自分で計算して確かめよ．

事前問 2 $\mathbb{Z}/3\mathbb{Z}$ において *2* の逆元を求めよ．

事前問 3 $\mathbb{Z}/13\mathbb{Z}$ について，和と積の表を作成せよ．

1.3 有限体上の除算

整数 c が整数 a の約数であり，かつ，整数 b の約数でもあるとき，c を a と b の公約数とよぶ．a, b の約数はそれぞれ有限個であるから，公約数 c も有限個である．その中で最大の正数 d を最大公約数とよぶ．記号では，$d = \gcd(a, b)$ と表す．gcd は，greatest common divisor の略である．とくに，$\gcd(a, b) = 1$ のとき，a と b は互いに素であるという．2 つの整数 a, b の最大公約数 $d = \gcd(a, b)$ について，以下が成り立つ．

補題 1

$$\gcd(a, b) = \gcd(r, b) \ (a = bq + r \text{ となる } b > r \geq 0 \text{ と } q \in \mathbb{Z}) \tag{1}$$

$$\gcd(a, b) = \gcd(a - b, b) \tag{2}$$

2 つの整数 a, b の最大公約数 d は，補題 1 の式 (1) を用いる方法と式 (2) がある．前者の方法がユークリッドの互除法であり，以下のようにして求められる．

アルゴリズム 1 (ユークリッドの互除法)
入力：2 正数 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．

事前問 4 次の数の最大公約数を求めよ．
(1) $gcd(1234567, 234578)$
(2) $gcd(11111111, 33332222)$

ユークリッドの互除法を応用した拡張ユークリッドの互除法から以下の補題が導かれる．

補題 2 整数 a と整数 b の最大公約数を d とするとき，

$$ax + by = d$$

をみたす整数 x, y が存在する．

アルゴリズム 2 (拡張ユークリッドの互除法)
入力：2 正数 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) $a_0 = a$，$a_1 = b$，$i = 1$ 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．

事前問 5 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) $a = 13,\ b = 8$

有限体の逆元は上述の拡張ユークリッドの互除法を用いて求めることができる．

補題 3 $\gcd(a,m)=1$ のとき，次の合同式をみたす整数 x が存在する．

$$ax \equiv 1 \pmod{m}.$$

事前問 6 合同式 $3X \equiv 1 \pmod{20}$ を解いてみよう．（つまり $\mathbb{Z}/20\mathbb{Z}$ 上の 3 の逆元を求めることになる．）

演習 1.1 *[拡張 ECD を用いた有限体上の逆元]* 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 *1* にある p_1, g_1 を用いて $g_1^{-1} \pmod{p_1}$ を求めよ．
(2) 表 *1* にある p_1, g_2 を用いて $g_2^{-1} \pmod{p_1}$ を求めよ．
(3) 表 *1* にある p_1, g_3 を用いて $g_3^{-1} \pmod{p_1}$ を求めよ．
(4) 表 *1* にある p_1, g_4 を用いて $g_4^{-1} \pmod{p_1}$ を求めよ．

2 つの整数 a, b の最大公約数 d は，補題 1 の式 (2) を用いると剰余を求めずに最大公約数が以下のように求められる．

アルゴリズム 3 (ユークリッドの互除法 2)
入力：2 正数 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)[1] $a_i \neq 0$ のとき，$a_{i+1} = |a_i - a_{i-1}|$，$a_i = \mathtt{min}(a_{i-1}, a_{i+1})$，$a_{i+1} = \mathtt{max}(a_{i-1}, a_{i+1})$，$i = i+1$ とし，(2) へ．

さらに，逆元をハード実装する際には，2 で割ることはシフトで実現できる．このため最大公約数の 2 の因子部分を別に扱うとさらにコンパクトにかつ高速に実装できる．

アルゴリズム 4 (バイナリユークリッドの互除法)
入力：2 正数 a，$b(a \geq b)$
出力：$\gcd(a, b) = d$
(1)[初期設定] $a_0 = a$，$a_1 = b$,$k = 0$，$i = 1$ とする．
(2) $a_i = 0$ ならば，$d = a_{i-1}2^k$ を出力し，終了．
(3) $a_i = 0 \pmod 2$ かつ $a_{i-1} = 0 \pmod 2$ のとき，$a_i \neq 0 \pmod 2$ あるいは $a_{i-1} \neq 0 \pmod 2$ になるまで，以下を繰り返す．$a_i = a_i/2$，$a_{i-1} = a_{i-1}/2$，$k = k+1$． (4) へ．
(4) $a_{i-1} = \mathtt{min}(a_{i-1}, a_i)$，$a_i = \mathtt{max}(a_{i-1}, a_{i+1})$，
(5) $a_{i-1} = 0$ ならば，$d = a_i 2^k$ を出力し，終了．
(6) $a_{i+1} = |a_i - a_{i-1}|$，
(7) $a_{i+1} = a_{i+1}/2^j$(a_{i+1} が奇数になるまで割る[2])
(8)a_i= min(a_i+1, a_i-1),a_i+1= max(a_i+1, a_i-1),
(9)i=i+1とし, (5) へ.
a_i= min(a_i-1, a_i+1),a_i+1= max(a_i-1, a_i+1),i=i+1とし, (4) へ.

演習 1.2 バイナリユークリッドの互除法を用いた拡張バイナリユーグリッドの互除法のアルゴリズムを作成し，実装しよう．また演習 *1.1* で値を確かめよう．

解析 1 *[拡張 ECD を用いた有限体上の逆元]* 拡張ユークリッドの互除法による逆元演算のステップ数 *(ループの実行回数)* が最大になる入力の特徴を実験的に予想する．予想するにはグラフなどの見える化をすることが重要である．小さい事例で，*CSV* ファイルにデータを保存し，グラフで特徴を抽出し，実際に利用される大きさで予想しよう．
(1) $p = 2^{12} - 77$ とする．$\mathbb{F}_p \ni \forall a$ に対して，a^{-1} を求め．ステップ数 *(ループの実行回数)* をカウントして求めよ．各

[1] 大小比較を高速に実現できることが望ましい．
[2] 2 で割る操作は 1 ビット右シフトで実現すると高速である．

値のステップ数をグラフなどで見える化し，どのような値が最大ステップ数になるか議論せよ．最大ステップ数と最小ステップ数，平均値，分散値を求めよ．
(2) $p = 2^{11} + 5$ として，*(1)* と同じことを実施せよ．
(3) どのような　a の逆元を求める際にステップ数が大きくなるか予想しよう．
(4)$p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ は素数である[3]．*(1), (2)* から最大ステップ数となる a を予想し，実際にステップ数を求めよう．
*(*各自，あるいはチームごとにステップ数が大きくなる元を予測し，*TA* バウンドと比較しよう.*)*

解析 2 *[*バイナリ *ECD* を用いた有限体上の逆元*]* バイナリ拡張ユークリッドの互除法による逆元演算のステップ数*(*ループの実行回数*)* が最大になる入力の特徴を実験的に予想する．予想するにはグラフなどの見える化をすることが重要である．小さい事例で，*CSV* ファイルにデータを保存し，グラフで特徴を抽出し，実際に利用される大きさで予想しよう．
(1) $p = 2^{12} - 77$ とする．$\mathbb{F}_p \ni \forall a$ に対して，a^{-1} を求め．ステップ数 *(*ループの実行回数*)* をカウントして求めよ．各値のステップ数をグラフなどで見える化し，どのような値が最大ステップ数になるか議論せよ．最大ステップ数と最小ステップ数，平均値，分散値を求めよ．
(2) $p = 2^{11} + 5$ として，*(1)* と同じことを実施せよ．
(3) どのような　a の逆元を求める際にステップ数が大きくなるか予想しよう．
(4)$p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ は素数である[4]．*(1), (2)* から最大ステップ数となる a を予想し，実際にステップ数を求めよう．
*(*各自，あるいはチームごとにステップ数が大きくなる元を予測し，*TA* バウンドと比較しよう.*)*
(5) 解析 *1* の最大ステップ数の元とバイナリ拡張ユークリッドの互除法の最大ステップ数の元に違いがあるか議論せよ．

1.4　べき演算

後に紹介する ElGamal 暗号の実行時間の主要部分はべき演算の実行時間となる．ここでは，べき演算を求める最も基本的なアルゴリズムであるバイナリ法について紹介する．暗号では，160 ビットの数には 0 も 0 が 160 個並ぶとして含める．このため，指数が 0 でも正しく動くことが重要である．

アルゴリズム 5 (バイナリ法) `binary(`g`,` k`)`
入力： 正整数 $k = \sum_{i=0}^{n-1} k_i 2^i \ (k_i = 0, 1)$ と g と k
出力： $y = g^k$
`Binary1(`k`,` g`)`
`1.` $y = 1$.
`2.` `for` $i = n-1, \cdots 0$, `do`
`if` $k_i = 1$, `then` $y = y^2 * g$.
`else` $y = y^2$
`next` i
`3.` `Output` y

事前演習 1 以下の数に対して，a^b を求めよ．
(1) $a = 23,\ b = 17$
(2) $a = 13,\ b = 8$
(3) $a = 5,\ b = 31$
(4) $a = 7,\ b = 41$

[3]NIST が規定した標準的に利用される楕円曲線の素体となる．
[4]NIST が規定した標準的に利用される楕円曲線の素体となる．

ElGamal 暗号などでバイナリ法を利用する際には，g, k の大きさがそれぞれ 1024 ビット，160 ビット以上となるため，アルゴリズム 5 のままでは変数 y のビット長が大きくなり，実用上動かない．アルゴリズム E.1.4 では，定義体 $\mathbb{F}_p$ を入力とし，y のビット長が定義体の 3 倍長より小さくなるように考慮したアルゴリズムである．このとき，法演算 Mod の計算時間もかかるので，法演算の回数を極力少なくすることも重要で，それが “定義体の 3 倍長より小さい” という条件となっている．

アルゴリズム 6 (法 p 上のバイナリ法) `mod_binary`$(g,\ k, n,\ p)$
入力： 正整数 $k = \sum_{i=0}^{n-1} k_i 2^i\ (k_i = 0, 1)$，$g, k$ のビットサイズ n，p，
出力： $y = g^k \bmod p$
`mod_binary`$(k,\ g,\ n,\ p)$
1. $y = 1$.
2. for $i = n-1, \cdots 0$, do
   if $k_i = 1$, then $y =$ Mod(Mod(y^2, p)$\cdot g$, p).
   else $y =$ Mod(y^2, p)
   next i
3. Output y

演習 1.3 *(1)* $2^{1234567890} \pmod{97}$ を求めよ．
(2) $2^0 \pmod{97}$ を求めよ[5]．

乱数生成の精度は暗号の安全性に大きな影響を与える．実際，ランダムに生成した秘密鍵がある確率で一致すると鍵が解読される．python には複数の乱数生成関数が提供されている．`secrets.randbelow`(n), `random.randint`(a, b) の出力する乱数の精度を簡単に統計データで確認する．具体的には，生成される乱数列の集合の相関係数とハミング重みで行う．理想的な乱数生成であれば，生成された乱数列の集合は互いに相関がないことが予想される．一方，ハミング重みとは乱数列の初期値を x_0 とするとき，i 番目に生成される乱数 x_i と x_0 とのハミング距離（2 進データでの距離）を d_i とする．理想的な乱数であれば，ハミング距離はビット列に対して一様に分布することが予想される．相関係数とハミング距離を用いて，乱数の検定をしてみよう．

解析 3 `secrets.randbelow`(n), `random.randint`(a, b) で生成される乱数の集合の相関係数を計算し，乱数の精度を比較する．10^3 個の乱数 k を *10* 回生成し，10^3 個の乱数の集合を *10* セット生成しよう．その後，*10* セットの相関係数とハミング距離を求める．それぞれの乱数生成関数がランダムな乱数を生成するか検定しよう．
(1) 64-*bit* のランダムな $k \in \{0, 1\}^{64}$.
(2) 128-*bit* のランダムな $k \in \{0, 1\}^{128}$
(3) 160-*bit* のランダムな $k \in \{0, 1\}^{160}$

2 講義 2(教科書 3, 7.8 章)

本講義の目的は計算量の理論的評価を理解することです．暗号は法 p 上の演算で実行されます．これらの計算量を理論的に評価することが暗号の評価に繋がります．暗号では 128 ビットの法乗算や 1024 ビットの法乗算，さらには二乗算，逆元が利用されますが，ビットや演算の違いを理論的に評価することが重要です．評価には以下を利用することが一般的です．

- 160 ビットの乗算の計算量の単位を M_{160} として評価
- 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
- 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 $I = 1 : 11$,

[5] 0 乗が正しく計算できることを確認するための課題である．

2.1 Fermat 法の応用によるサイドチャネル攻撃の回避

演習 1.1 では拡張ユークリッドを用いた逆元のアルゴリズムが元により実行速度が変わることを確認した．ここでは，Fermat の小定理を用いた逆元の実装方法を考え，その実行速度について考察してみよう．

-Fermat の小定理-

素数 p に対し p と互いに素な任意の自然数 a は，$a^{p-1} \equiv 1 \pmod{p}$ を満たす．これを **Fermat の小定理**という．Fermat の小定理の応用の一つに擬素数判定テストがある．n が与えられた正整数としよう．このとき，$1 \le a \le n-1$ かつ $\gcd(a,n) = 1$ となる a に対して，$a^{n-1} \equiv 1 \pmod{n}$ を満たすかどうか判定するテストを繰り返すことで，確率的に素数判定を行う．これを**フェルマー法**という．フェルマー法は素数は必ず素数と判定するが，合成数も素数と確率的に判定することがある．このように確率的に素数であるということを判定する方法は確率的素数判定法と呼ばれる．

Fermat の小定理から，以下の式が成り立つ．

$$\begin{aligned} a^{p-1} &\equiv 1 \pmod{p} \\ a^{-1} &\equiv a^{p-2} \pmod{p} \end{aligned} \tag{3}$$

上述の (3) を用いて逆元を求めるアルゴリズムを構築しよう．

アルゴリズム 7 (逆元 1) `Inverse2(`p`,` g`)`
入力： 素数 p と g
出力： $y = g^{p-2} \pmod{p}$
`Inverse(`p`,` g`)`
`1.` $y = g$`.`
`2.` $y =$ `ModBinary1(`$p-2$`,` g`,` p`)`
`3.` `Output` y

演習 2.1 *[Fermat の小定理を用いた有限体上の逆元] Fermat* 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 *1* にある p_1, g_1 を用いて $g_1^{-1} \pmod{p_1}$ を求めよ．
(2) 表 *1* にある p_1, g_2 を用いて $g_2^{-1} \pmod{p_1}$ を求めよ．
(3) 表 *1* にある p_1, g_3 を用いて $g_3^{-1} \pmod{p_1}$ を求めよ．
(4) 表 *1* にある p_1, g_4 を用いて $g_4^{-1} \pmod{p_1}$ を求めよ．
(5) 上記 *4* つの計算量を，1024 ビットの乗算の回数で比較する．但し，乗算の回数ではコラムを考慮する．
(6) 上記 *4* つのステップ数 *(ModBinary1* におけるループ数*)* をカウントして求めよ．(ステップ数による見積もりはかなり雑な見積もりになることを知りましょう.)

拡張ユークリッドを用いた逆元の演算は，元により演算時間が異なり，サイドチャネル攻撃の課題が存在した．一方，Fermat 法を用いた逆元演算のアルゴリズムでは，一定時間の演算を実現することがわかる．

解析 4 *[Fermat 法を用いた有限体上の逆元] Fermat* の小定理を応用した逆元演算の乗算回数が入力によりどのように変わるかを実験する．
(1) $p = 2^{12} - 77$ とする．$\mathbb{F}_p \ni \forall a$ に対して，a^{-1} を求め, 12 ビットの乗算の回数で比較する．最大乗算回数と最小回数，平均値，分散値を求めよ．
(2) $p = p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ は素数である．`secrets.randbelow(`$p-1$`)+1,` `random.randint(`$1, p-1$`)` でランダムに *256* ビットの数 a を 10^4 個生成し，それぞれの逆元 a^{-1} を求めて，逆元にかかる時間の実測値の平均値，分散を求めよう．
(3) 解析 *2* と比較する．
(4) *2* つの乱数生成で違いがあるか議論せよ．

演習 2.2 バイナリ法を用いて，法 *4* で *3* と合同な素数 p に対して，$\mathbb{F}_p^* \ni a$ の平方根が存在するなら，それを求めるアルゴリズムを作成せよ．

2.2 乱数の精度

コラム-誕生日パラドックス-

誕生日パラドックスとは「同じ誕生日の人がいる確率が 50 %を超えるには何人集まればよいか?」という問題の解が予想に反し小さいことをいう．実際に確認してみよう．n 人の中に一人も同じ誕生日の人がいない確率は $p(365,n) = \frac{364}{365} \cdot \frac{363}{365} \cdot \frac{362}{365} \cdots \frac{365-n+1}{365} = \frac{365!}{365^n(365-n)!}$．よって n 人の中で同じ誕生日の人が少なくとも 2 人いる場合の確率 $\overline{p(365,n)}$ は

$$\overline{p(365,n)} = 1 - \frac{365!}{365^n(365-n)!}$$

となる．一般に一様多数にある M 種類の異なるデータから n 個のデータを一様に入手した際に同じデータが少なくとも 1 つ存在する確率は下記で与えられる．

$$\overline{p_1(M,n)} = 1 - \frac{M!}{M^n(M-n)!}$$

さらに，この理論値は Talor 展開を用いて以下のように近似できる．

$$\overline{p_2(M,n)} = 1 - exp[-\frac{n(n-1)}{2M}]$$

問 2.1 *(1)* バースデーパラドックスのコラムに *2* 通りの理論値 $\overline{p_1(M,n)}$, $\overline{p_2(M,n)}$ を記載した．その収束の速度の違いを見てみよう．*365* 日の誕生日 $M = 365$, $n = [1,100]$ を用いて，$\overline{p_1(M,n)}$ と $\overline{p_2(M,n)}$ を図示しよう．
(2) $\overline{p_1(M,n)}$，$\overline{p_2(M,n)}$ の誤差はどのようになるか？
(3) 2 つの理論値をどのように使い分けるとよいか議論しよう．

解析 5 本問題は理論値と実験値の違いを評価する方法を学ぶ問題である．実験値を用いて理論値をどのように評価するとよいのか学習しよう．*10* 種類のデータから衝突を考える問題を取り上げる．問 *3.1* の体とベースポイント，$m = 4$ を用いて，*1* から *10* までの乱数 r をランダムに生成し，$c = 4 \cdot y^r \pmod{23}$ *(*r を用いた暗号文の一部*)* を複数回求めて，ファイルに出力するプログラムを構築する．但し，乱数は毎回生成する．
(1-1) 同じ暗号文が出てくるまで暗号化を行う操作を *100* 回行い，各回で一致する暗号文が出現するまでに必要となった暗号化の回数を求める．次に，出現回数の最小値 n_{min} と最大値 n_{max} を出現回数の下限と上限とし，$[n_{min}, n_{max}]$ の範囲で，各出現回数が起こった回数，次に出現確率を求め，グラフにプロットする．つぎにグラフの結果からわかることをまとめよ．
(1-2) 種類の数が *10* の場合に，$[n_{min}, n_{max}]$ の範囲で各出現回数が起こる確率をバースデーパラドックスのコラムの $\overline{p_1(M,n)}$，$\overline{p_2(M,n)}$ を用いて求めて，図示しよう．
(1-3) (1-1) の実験値と *(1-2)* の理論値との差をグラフを用いて議論しよう．
（オプション）*(3)* 問は小さい数なので，データに偏りが起こる可能性がある．そこで，例えば，$p = 2^{31} - 1$ に対して，同様に $g = 1090433$ とし，g^k を計算し，実験をしてみよう．なお，g の位数は 331 である．k は乱数生成関数（`secret.choise` や `random.randint`）で生成しよう．

問 2.2 今回の *PBL* の参加者から誕生日が一致する *2* 名をオンラインで求める方法を考えよう．オンラインの学生向けの質問に対して，複数人の学生が答えた場合，*1* つの質問に対して，*1* 名の学生の回答のみを見てよいとする．何回の質問で同じ誕生日の人が求められるか予想してみよう．つまり，同じ誕生日のペアが見つかるまで繰り返す質問の回数の期待値を求めよ．(期待値は方法に依存します.）実際にやってみて，近かった値からポイントを付与します．なお，当日は，実際に *TA* が全員参加型のアルゴリズムで誕生日コリジョンを求めて，各自の予想と合っているかを確認します．*TA* の回答までに提出しましょう．

3 講義 3(教科書 8 章)

プライバシーにはレベルがあります．名前を暗号化すると，暗号文からは本人がわかりません．しかし，それだけでは十分ではないこともあります．つまり，同じ暗号文が同一人物を表すとしたらどうでしょう？この概念が linkability です．本講義では，公開鍵暗号を理解し，さらに，暗号の一つの機能である準同形性を応用して，linkability を防ぐ方法，また，linkability の問題点を理解しましょう．なお，ElGamal 暗号は乗法準同形を満たします．

3.1 ElGamal 暗号

【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. 有限体 $\mathbb{F}_p$ (p は素数 p) と位数 l のベースポイント $g \in \mathbb{F}_p$ を生成する．
2. 乱数 $x \in \mathbb{Z}_l$ を生成して秘密鍵とし，

$$y = g^x \pmod{p}$$

を計算する．

〈公開鍵〉$p,\ g,\ y$

〈秘密鍵〉x

【暗号化】 ユーザ A が平文 $m \in \mathbb{Z}_n$ を暗号化して B に送るとする．

1. 乱数 $r \in \mathbb{Z}_l$ を生成し，

$$u = g^r \pmod{p} \tag{4}$$

を計算する．

2. B の公開鍵 y を用いて

$$c = y^r m \pmod{p} \tag{5}$$

を計算し，暗号文 $(u, c) \in \mathbb{F}_p \times \mathbb{F}_p$ を B に送信する．

【復号】 B は，自分の秘密鍵を用いて，

$$m = c/u^x \pmod{p}$$

を計算し，m を復号する．

問 3.1 素数 $p = 23$ による有限体 $\mathbb{F}_p$ 及びベースポイント $g = 2$ とするとき，以下の問に答えよ．

(1) 2 の位数を求めたい．$2^\ell = 1 \pmod{23}$　となる $\ell > 1$ を一つずつ計算する方法もあるが，理論的に高々*2* 回のべき乗演算で求める方法を考えてみよう[6]*.*

(2) 秘密鍵に *2* を用いた *ElGamal* 暗号の公開鍵を作れ*.*

(3) (2) の鍵を用いて，$m = 4$ を $r = 3$ を用いて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．

(4) 2 を用いて，原始根 *(*位数が *22* となる元*)* を *1* 回の乗算で求めよ[7]*.*

演習 3.1 表 *2* の例 $\mathbb{F}_{p_4}, g_4 \in \mathbb{F}_{p_4}, g_4$ の位数 ℓ_4 を用いる．このとき，有限体 $\mathbb{F}_p$ を用いて，設問に沿って *ElGamal* 暗号を実現せよ．なお，実装においては暗号化に用いる乱数は入力として与える．

(1) 秘密鍵 d_b = 172 35091 96654 51459 12345 17144 99640 83306 22345 44321 に対する公開鍵 p_b を求めよ．

(2) 公開鍵を用いて，平文 m と乱数 r, m = 123 75081 11111 51459 12345 17144 99640 33333 55555 44444

r = 123 45678 91234 56789 12345 67891 23456 78912 34567 89123

を用いて，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．

演習 3.2 演習 *3.1* の有限体を利用する．

(1) $\ell_4 - 1$ までの乱数 r をランダムに生成し，自分の秘密鍵を生成，その後，公開鍵　$g^r \pmod{p}$ を作成し，*moodle* に提出する．また作成した公開鍵を公開鍵掲示板に保管しよう．

(2) TA の公開鍵を用いて，本日のお昼ご飯を暗号化して，掲示板に記載しよう．なお，データは *16* 進に変換して提出すること．*1024* ビットで暗号化できる文字数[8]までお昼ご飯を表すこと．*TA* は正しく復号できれば掲示板に受講生の鍵を用いて暗号化して感想を入れよう．

(3) 受講者は秘密鍵で復号し，*TA* のメッセージを入手しよう．入手できたら，掲示板に *OK* を記載して下さい．

[6] この問は，$\mathbb{F}_p$ の元の個数と位数の関係を理解するためである．

[7] この問は，元の位数の関係を理解するためである．

[8] 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．

解析 6 *ElGamal* 暗号は乗法準同形暗号である．乗法準同形暗号であることを用いて，セキュアなアプリケーションを考えて，設計してみよう．設計とは，*ElGamal* 暗号の秘密鍵をもつユーザと公開鍵で暗号化するユーザを明記し，公開鍵が掲載されるサーバはどこにあるかまで明確にすること．また，*ElGamal* 暗号を導入することで，導入する前と比べて，何が守られるのかを明記する．参考までに *1* つの事例を掲載する．例えば，図書館での本の管理に，書籍情報を *RFID* のタグの中に入れて，本の貸し出しを管理するシステムの提案を考えよう．*RFID* のデータは外から容易に入手できるのが特徴である．借りた本は持ち歩くので，*RFID* のタグに本の名前をそのまま書いておくと，どのような本を借りたのか，図書館以外の人からも容易に入手できる．これを防ぐ方法を考える．この際，本の名前の秘匿だけでなく，*linkability* の観点からのプライバシ強化も必要である．一方，本を返却する際には，どの本が貸し出されたのかの情報を *RFID* から入手できる必要もある．プライバシ情報を保護するという観点で，貸し出し中の本の *RFID* から入手される情報からプライバシを保護しつつ，本の返却が正しくできる必要がある．これらを満たした *ElGamal* 暗号を導入した図書館管理システムを考え，図書館に提案する．思いつかない人は上記事例で，どの場面に暗号機器を設定し，鍵の設定をどのようにするか具体的に考えよ．

3.2 DH 鍵共有法

DH 鍵共有法を紹介する．DH 鍵共有法は公開のネットワークを用いて鍵を共有する方法である．

【システム設定】 $\mathbb{F}_p$ を有限体とし，$g \in \mathbb{F}_p$ を位数が大きな素数 ℓ の元 (ベースポイント) とする．また $\mathbb{F}_p$ 及び g はシステム内で公開する．

【鍵共有】

1. A は，乱数 $x_a \in \mathbb{Z}_\ell^*$ を用いて，$y_a = g^{x_a} \bmod p$ を計算し，B に送信する．y_a は A の公開鍵になる．
2. B は，乱数 $x_b \in \mathbb{Z}_\ell^*$ を用いて，$y_b = g^{x_b} \bmod p$ を計算し，A に送信する．y_b は B の公開鍵になる．
3. A は，受信した y_b 及び x_a を用いて，
$$K_{a,b} = y_b^{x_a} \bmod p$$
を計算し，$K_{a,b} = g^{x_a x_b} \bmod p$ を共有鍵とする．
4. B は，受信した y_a 及び x_b を用いて，
$$K_{a,b} = y_a^{x_b} \bmod p$$
を計算し，$K_{a,b} = g^{x_a x_b} \bmod p$ を共有鍵とする．

演習 3.3 *(1) moodle* からチームの *TA* の公開鍵と *ElGamal* 暗号の章で構築した自分の秘密鍵を用いて，*DH* 共有鍵 K を作成し，ファイルに出力するプログラムを作成しよう．
(2) 各自の生徒番号 *ID* を共有鍵と同じ長さになるようにする．乱数 r を生成し，$ID' = r \,\|\, 0000 \,\|\, ID$ とする．次に，排他的論理和で暗号化 $ID' \oplus K$ し，$ID' \oplus K$ を公開鍵掲示板に提出する．
(3) （*FYI*) 他のデータを暗号化する際には，文字コードは *B* 章の `hexlify` を利用してください．
(4) TA は *(3)* を復号して，復号できた *ID* を確認したら，公開鍵掲示板に掲載する．但しデータは *16* 進に変換して記載すること．

解析 7 *DH* 鍵共有法で共有した鍵の検証について考える．
1. 演習 *3.3* では，*TA* と鍵を共有し，正しく共有できたことを検証した．しかし，検証方法はあまり，安全とは言えない．この検証方法のどこに問題があるのか，議論せよ．
2. DH 鍵共有法で正しく秘密が共有できたことを，公開の *NW*（掲示板利用可）で，検証する方法を考えよう．このとき，セキュリティポリシー（どの部分が秘密として第三者からみえないと考えるか）を明確にする．さらに，できれば，実際に掲示板を利用して，構築したアルゴリズムで *TA* と共有鍵が正しいことを検証しよう．

4 講義 4

4.1 離散対数問題解読

G を乗法に関する有限巡回群とする．離散対数問題とは G の生成元 g と $G \ni h = g^x$ から，離散対数である整数 x を求める問題である．

$$(g, h = g^x) \to x$$

離散対数問題の困難性はこれまでに紹介した DH 鍵共有手法や ElGamal 暗号の安全性の根拠となっている．本章では離散対数問題の解読法について学習し，実際に暗号の解読を試みる．

G の位数を n とする．ある G の元の 2 通りの表現方法（しばしば “衝突” と呼ばれる）を用いて、離散対数問題が解ける．すなわち $a_i \neq a_j, b_i \neq b_j$ を満たす整数 a_i, a_j, b_i, b_j について，衝突

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$$

が成立する場合、$g^{a_i - a_j} h^{b_i - b_j} = 1_G$ と変形し、$x = (a_j - a_i)/(b_i - b_j) \bmod n$．以降では衝突を発見する手法とする Pollard の ρ 法を解説する．

4.1.1 ρ 法

Pollard はある関数 $f : G \to G$ を繰り返し計算することで G の元をランダムに移動させ，衝突を発見する手法を提案した．ある初期値 $x \in G$ について

$$x_i := \underbrace{f \circ f \circ \cdots \circ f}_{i \text{ 回}}(x_0)$$

と定義する．G は有限であり，ある $i < j$ について $x_i = x_j$ が成立する．したがって x_i の列は図 4 が示すようにギリシャ文字の ρ と似た形となる．この特徴から ρ 法と名付けられた．実際，Pollard は関数 f を次のように定義した．

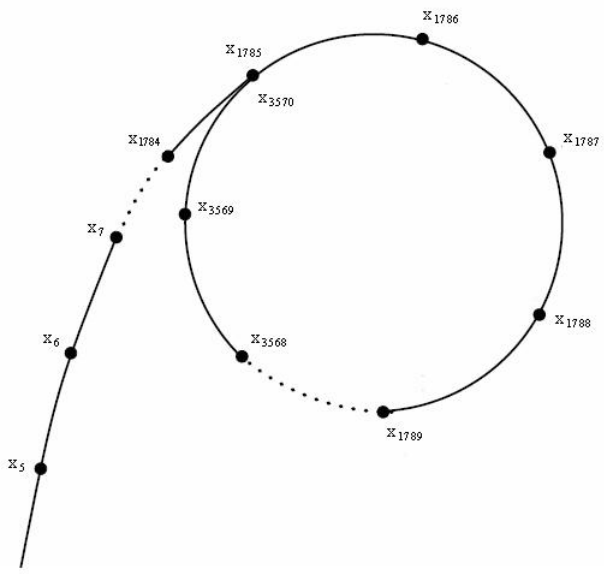


図 1: ρ 法において生成される x_i の列

$$f(x) = \begin{cases} hx \text{ if } x \in S_1, \\ x^2 \text{ if } x \in S_2, \\ gx \text{ if } x \in S_3 \end{cases}$$

ここで S_1, S_2, S_3 は共通の要素を持たず $S_1 \cup S_2 \cup S_3 = G$ を満たす集合とする．ある整数 a_0, b_0 について初期値 $x_0 = g^{a_0} h^{b_0}$ とすると，$x_i = g^{a_i} h^{b_i}$ を満たす整数 a_i, b_i を容易に把握しながら G の元を生成できる．したがって $x_i = x_j$ となるとき，衝突 $g^{a_i} h^{b_i} = g^{a_j} g^{b_j}$ が発見できる．

各 $x_i \in G$ を一様ランダムに生成すると仮定すれば，誕生日のパラドックスと同様な議論から $O(\sqrt{n})$ 個の $\{x_i\}$ 中に衝突があることが期待できる．すなわち ρ 法は $O(\sqrt{n})$ の計算量で離散対数問題を解くことができる．

問 4.1 *(1)* G を $(\mathbb{Z}/101\mathbb{Z})^\times$ とし，その生成元 $g = 2, h = 3 \in G$ とする．また関数 f を以下で定義する．

$$f(x) = \begin{cases} hx \ if \ x \in S_1 = \{x \in G \mid x \equiv 1 \mod 3\}, \\ x^2 \ if \ x \in S_2 = \{x \in G \mid x \equiv 2 \mod 3\}, \\ gx \ if \ x \in S_3 = \{x \in G \mid x \equiv 0 \mod 3\} \end{cases}$$

このとき $g^t \equiv h \mod 101$ を満たす整数 t を求めたい．初期値 $a_0 = 1, b_0 = 0$ を用いた ρ 法に従って，次の表の続きを完成させることで衝突を発見し t を求めよ．まずこの表の $i = 2$ までのステップについて説明する．

- $i = 0$ のとき
  $a_0 = 1, b_0 = 0$ より，x_0 は

  $$x_0 = g^1 \cdot h^0 = 2^1 \cdot 3^0 = 2$$

  のように更新される．次に更新した x_1 を $f(x_0)$ のように変換するために x_0 の範囲を決定する．

  $$x_0 \equiv 2 \mod 3$$

  となる．この x_0 の値から $f(x_0) = x_0^2$ を実行する．

- $i = 1$ のとき

  $$x_1 = x_0^2 = (2^1 \cdot 3^0)^2 = 2^2 \cdot 3^0 = 4$$

  と x_1 は更新されるので，$i = 1$ のとき，表のように $a_1 = 2, b_1 = 0$ となる．さらに $i = 0$ と同様に更新した x_1 を $f(x_1)$ のように変換するために x_1 の範囲を決定する．

  $$x_1 \equiv 1 \mod 3$$

  となる．この x_1 の値から $f(x_1) = hx_1$ を実行する．

- $i = 2$ のとき

  $$x_2 = hx_1 = 3 \cdot (2^2 \cdot 3^0) = 2^2 \cdot 3^1 = 12$$

  と x_2 は更新されるので，$i = 2$ のとき，表のように $a_2 = 2, b_2 = 1$ となる．以上が $i = 2$ までのステップである．これと同様なステップを繰り返していくことで，$g^t \equiv h \mod 101$ を満たす整数 t を求めよ．

(2) (1) において異なる初期値 $x'_0 = g^{a'_0} h^{b'_0}, (a'_0, b'_0) \neq (1, 0)$ を用いた場合、ステップ数がどのようになるか確かめよ．

(3) (1) において $h = 7$ としたとき，$g^t \equiv h \mod 101$ を満たす整数 t を求めよ．ただし初期値や繰り返し回数に注目すること．

4.1.2 衝突発見のための工夫

関数 f の構成　関数 f の作り方については問 4.1 の限りではない．例えば同様な更新式を用いたまま $S_1 = \{x \in G \mid 0 \leq x \leq 33\}, S_2 = \{x \in G \mid 34 \leq x \leq 67\}, S_3 = \{x \in G \mid 68 \leq x \leq 100\}$ としてもよい．ただし x^2 の計算が行われる S_2 には単位元 1_G が含まれないように注意する．なぜなら $x_i = 1_G$ であったとき，以降すべての $j > i$ について $x_j = 1_G$ が成立し，新たな元が生成されないためである．

ステップ数 i	a_i	b_i	$x_i = 2^{a_i}3^{b_i} \mod 101$
0	1	0	2
1	2	0	4
2	2	1	12
3	3	1	24
		⋮	

Distinguished point を用いたメモリ使用量の削減　Rivest は ρ 法のメモリ使用量を削減するために，$\{x_i\}$ のうちある条件を満たす元のみを記録し比較する手法を提案した．この条件を満たす元は distinguished point と呼ばれ，また確認が容易である条件が用いられた．例えば $\{x_i\}$ のうち d の倍数である元のみを保持するとき，保持する元の数はおよそ $1/d$ となりメモリ使用量を削減できる．一方で衝突発見のために生成する x_i の数は率直な ρ 法よりもおよそ d 個多くなる．なぜなら衝突を発見できるのは図 4 における循環部（円）のうち，d の倍数である元を 2 度目に生成した時となるからである．

λ 法による並列化　ρ 法における x_i の生成は容易に並列化可能である．例えば初期値 (a_0, b_0) から元 x_i を生成することと並行して，異なる初期値 (a_0', b_0') から元 x_i' を生成する場合を考える．両者が同じ関数 f を用いるとき，ある i, j について $x_i = x_j'$ が成立した場合以降すべての $k \geq 0$ について $x_{i+k} = x_{j+k}'$ が成立する．すなわち両者が 1 度同じ元を訪れた後に生成する列は等しくなる．一方の軌跡が他方の軌跡の中間地点に合流すると考えたときギリシャ文字の λ のように見えることから以上のような ρ 法の並列化計算は λ 法と呼ばれる．当然 λ 法は $N > 2$ 種類の初期値に関しても同様に動作する．

Floyd の循環検出法　列 $\{x_i\}$ における衝突は Floyd の循環検出法を用いても発見できる．Floyd の循環検出法はウサギとカメのアルゴリズムとも呼ばれる手法で，$x_{2i}, x_{2(i+1)} = f(f(x_{2i}))$（ウサギ）と $x_i, x_{i+1} = f(x_i)$（カメ）の 2 値比較を繰り返すことで列内の衝突を検出する．率直な ρ 法のように衝突発見までの x_i の値をすべて保持する必要がなく，メモリ使用量が非常に小さいという特徴がある．次に Floyd の循環検出法の計算量を解析する．図 4 において循環部（円）の長さを o，それ以外の長さを ℓ とおく．ℓ 回目の 2 値比較時にはカメ x_ℓ が循環部の開始位置に，ウサギ $x_{2\ell}$ がその ℓ ステップ先に存在する．ウサギは循環部を回り続けるため，このときウサギはカメの $o - (\ell \mod o)$，すなわち高々o ステップ後方にいることが分かる．ウサギは i が増えるごとにカメとの距離を 1 ずつ縮めていくことから全体で高々$\ell + o$ 回の 2 値比較によって衝突を検出できる．E.4.1 章の議論から $\ell + o$ の大きさは $O(\sqrt{n})$ であり，Floyd の循環検出法の計算量は $O(\sqrt{n})$ となる．なお x_i, x_{2i} を元に $x_{i+1}, x_{2(i+1)}$ を得るためには関数 f を 3 度計算する必要がある．よって Floyd の循環検出法の計算時間は率直な ρ 法よりも一般に大きくなる．なお，Floyd の循環検出法には λ 法による並列化を適用できないことに注意する．

問 4.2 G を $(\mathbb{Z}/47\mathbb{Z})^\times$ とし，$g = 16, h = 32 \in G$ とする．また関数 f は問 *4.1* と同じく定義されている．$g^{23} \mod 47 = 1$ を利用し、$(a_i = 1, b_i = 0)$ とし、$(a_{2i} = 1, b_{2i} = 1)$ とし、*Floyd* 法で解いてください。

4.1.3　チーム対抗課題

班のメンバーで協力して以下の五つの課題に取り組もう．見つかった x, 計算時間と繰り返し回数, 初期条件と関数 f をチームプレゼンで報告してください．その後，moodle に提出しましょう．

課題 1（5 点） 次の式を満たす整数 x を求めよ．

$$397628281^x = 404852643 \mod 554860571$$

$$Hint: 397628281^{5044187} = 1 \mod 554860571$$

課題 2（5 点） 次の式を満たす整数 x を求めよ．

$$383929062152^x = 329819161243 \mod 1023417167557.$$

$$\textit{Hint} : 383929062152^{1740505387} = 1 \mod 1023417167557.$$

課題 3（20 点） 次の式を満たす整数 x を求めよ．

$$472974174173350^x = 391980987166049 \mod 497264225202533.$$

$$\textit{Hint} : 472974174173350^{124316056300633} = 1 \mod 497264225202533.$$

課題 4（30 点） 次の式を満たす整数 x を求めよ．

$$431147972333069417^x = 511265886395448114 \mod 817108219307103587$$

$$\textit{Hint} : 431147972333069417^{408554109653551793} = 1 \mod 817108219307103587$$

課題 5（40 点） 次の式を満たす整数 x を求めよ．

$$6571397331495670129 77^x = 903837092395032338397 \mod 1077984309859658267861$$

$$\textit{Hint} : 6571397331495670129 77^{8693421853706921515} = 1 \mod 1077984309859658267861$$

4.2 Blockchain

分散型の通貨システムは、以前から研究対象とされてきました。2009 年には、最初のブロックチェーンであるビットコインが提案されました。2014 年には、スマートコントラクトと呼ばれる追加機能を提供するためにイーサリアムが提案されました。スマートコントラクトは、ブロックチェーン上で実行され、お金を送受信することができるプログラムです。この講義では、ビットコインを例に使用します。以下に重要な特性を挙げます：

- **No trusted third party:** システムは分散化されています。1 つの団体がシステムを制御することはありません。
- **Publicly Verifiable:** ブロックチェーンに格納されたすべてのデータは公開されています。
- **Immutable:** 各ブロックのハッシュ値は次のブロックに含まれています。
- **No double spending:** 例えば、アリスがボブに 100 ドルを送った場合、アリスは同じ 100 ドルをチャーリーに送ることはできません。

4.2.1 Keys and Accounts

- **Secret key:** ビットコインは P256k1 曲線を使用しています。アカウントはステップバイステップで作成することができます。

$$x = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD$$

- **Public key:** P256k1 で定義されたベースポイント G を使用します。公開鍵は次のように計算することができます。

$$\begin{aligned} Y &= x \cdot G \\ &= 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD \cdot G \\ &= (F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A, \\ &\quad 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB) \end{aligned}$$

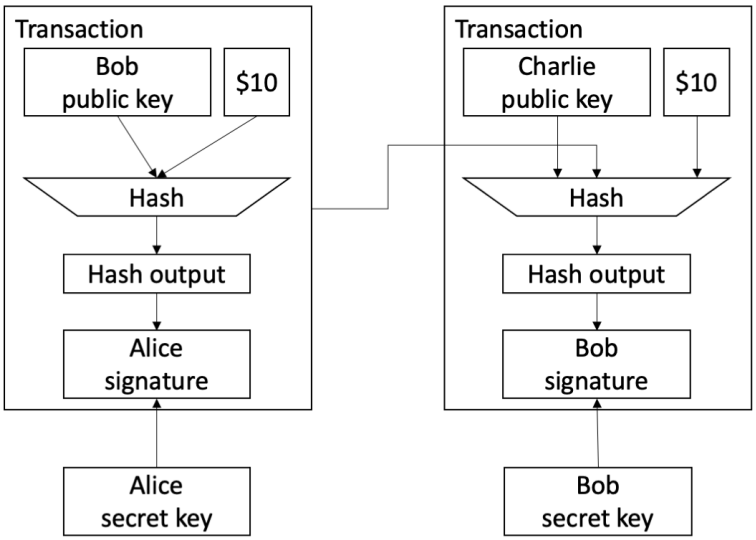


図 2: Transactions

- **Address:** アドレスはハッシュ化された圧縮公開鍵の最後の 20 バイトです。

$$address = 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy$$

4.2.2 Transactions

アリスがボブに 10 ドルを送り、ボブがチャーリーに 10 ドルを送ると仮定します。ボブがチャーリーにお金を送りたい場合、ボブはトランザクションを作成し、秘密鍵で署名する必要があります。公開鍵によって、誰でもボブの署名を検証することができます。各トランザクションは順番に連鎖しています。図 6 にビットコインのトランザクションを示します。

演習 4.1 以下は簡略化されたトランザクションの例です。ユーザーは *10* ビットコインを表 *2* の公開鍵 y_4 に送信します。

(1) 署名を検証してください。署名が有効であれば、y_4 はビットコインを正しく受け取ったことを表します。

```
{
    "transaction": {
        "sender": 10787370389507249913234092798381729525041937566920316815553191028684298987260472593326
        "receiver": 42063703388261034067155901193984173769497588090965563568973262694193054862536363577
        "value": 10,
        "previous tx hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
    },
    "signature": [1360251701595785788244923107714815394180602492790, 4664261513161443094696941551271377
}
```

演習 4.2 署名の生成方法のコードを添付します。表 *2* のパラメータを使用し、サンプルコードを参照してこの問題に答えてください。

(1) トランザクションを作成し、*10* ドルを送信元に返します。

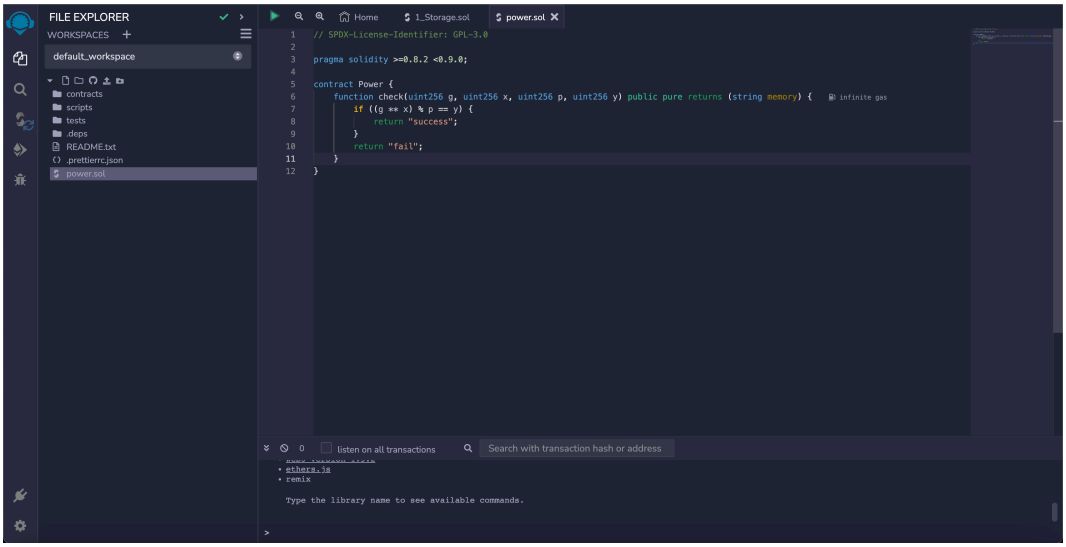


図 3: Remix

(2) トランザクションに署名します。

(3) トランザクションを検証します。

```
tx = '{"sender": 10787370389507249913234092798381729525041937566920316815553191028684298987260472593326310
"receiver": 42063703388261034067155901193984173769497588090965563568973262694193054862536363577662132973
```

```
tx = json.loads(tx)
sha256_hash = hashlib.sha256()
sha256_hash.update(json.dumps(tx).encode())
m = int.from_bytes(sha256_hash.digest(), byteorder='big')

r = randrange(1, L4)
u = pow(G4, r, P4) % L4
v = (pow(r, -1, L4) * (m + X4 * u)) % L4

print(f"u {u}")
print(f"v {v}")
```

4.2.3 Smart Contract

スマートコントラクトはブロックチェーンに格納されたプログラムです。スマートコントラクトを書くためのプログラミング言語は Solidity です。スマートコントラクト内で変数や関数を定義し、計算を行い、ETH を転送することができます。オンラインのスマートコントラクトエディタである https://remix.ethereum.org/ をご利用ください。

演習 4.3 パワー (*pow* 関数) チェックの例としてスマートコントラクトを提供します。正しい入力を与えると、*"success"* と出力されます。失敗した場合は、ガスリミットを増やして再試行することができます。ガスはスマートコントラクト内の計算時間の単位です。より多くの計算にはより多くのガスが必要です。

(1) *Try to get a success output!*

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract Power {
    function check(uint256 g, uint256 x, uint256 p, uint256 y) public pure returns (string memory) {
        if ((g ** x) % p == y % p) {
            return "success";
        }
        return "fail";
    }
}
```

5 講義 5(教科書 7 章)

5.1 DSA 署名

【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. A は，有限体 $\mathbb{F}_p$ とベースポイント $g \in \mathbb{F}_p$ を生成する．g の位数を素数 ℓ とする．
2. 乱数 $x \in \mathbb{Z}_\ell$ を生成し，これを秘密鍵とし，

$$y = g^x \bmod p$$

を計算する．
⟨ 公開鍵 ⟩ p, g, y
⟨ 秘密鍵 ⟩ x
さらに一方向性ハッシュ関数

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

を全ユーザに通知する．

【署名生成】ユーザ A は平文 $m \in \{0,1\}^*$ 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，

$$\begin{aligned} u_1 &= g^r \bmod p \\ u &= u_1 \bmod \ell \end{aligned} \tag{6}$$

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，

$$v = r^{-1}(m' + xu) \bmod \ell \tag{7}$$

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，署名を拒否する．
2. 公開鍵 y を用いて，

$$\begin{aligned} m' &= H(m) \\ u' &= (g^{m'/v} y^{u/v} \bmod p) \bmod \ell \end{aligned}$$

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

問 5.1 素数 $p = 23$ による有限体 $\mathbb{F}_p$，ベースポイント $g = 2$，秘密鍵 2 を用いて *DSA* 署名を考えた場合，以下の問に答えよ．実装せずに手で解いてみよう．
(1) $r = 3$ を用いて，$m = 4$ に署名した結果を求めよ．また検証できることを確かめよ．ここでは，ハッシュ関数は利用せずに，$H(m) = 4$ としてアルゴリズムを適用する．

問 5.2 *DSA* 署名を実装する前に，署名生成および署名検証のアルゴリズムを記載しよう．その後，必要となる乗算回数，べき乗関数の呼び出し回数，逆元の回数を求めよ．それぞれの演算回数がなるべく小さくなるようにアルゴリズムを記載しよう．

演習 5.1 演習 *3.1* の有限体を用いて，*DSA* 署名を実現せよ．なお，ハッシュ関数は *Shake 128* を利用する．
(1) 秘密鍵 d_b = 370 75081 86654 51459 12345 17144 99640 83306 22345 44321 に対する公開鍵 p_b を求めよ．
(2) 公開鍵を用いて，m = 123 75081 11111 51459 12345 17144 99640 33333 55555 44444
r = 123 45678 91234 56789 12345 67891 23456 78912 34567 89123 に署名した結果を求めよ．また検証も確かめよ．
(3) 各自，自分の公開鍵を用いて，目標コースに向けて，実施する課題を記載した文章に署名をして提出する．文章は「アドバンスコースに向けて，○○の演習を実施します.」のフォーマットで報告することとする．*TA* は文章と署名を入手して，署名が正しければ *OK* を掲載しよう．なお，署名は *16* 進数で提出すること．

解析 8 グループ内のメンバと *TA* の間で，今回の *PBL* の感想を合同で作成し，合議文章として発行する．講義の *DSA* 署名のアイデアを用いて，合議署名を設計してみよう．設計書は以下の方針で作成する．
1. ユーザは n 人．$U_1, \cdots, U_n$ (n は各チームの人数)
2. 各自の公開鍵設計
3. DSA 署名による合議の作成は *dsaSignGen* を用いて，どのように秘密鍵をいれるのか，関係を明確にする．
4. セキュリティポリシー（署名の順序を検証できるかできないか，どの部分が秘密として第三者からみえないと考えるか，署名者はルールに従うかなど）
5. スケーラビリティ（署名者をいつの時点で決定するか．署名者の更新は可能か）
6. 性能 (設計した合議方法を実現するのに必要な各人の計算量をべき乗の回数で評価するとともに合議文章の署名サイズ)

6　ハイブリッド暗号

6.1　ハイブリッド暗号の構成

演習 6.1 *ElGamal* 暗号と *DSA* 署名を学習した．ここでは，暗号化と署名を両方実行するプログラムを作成しよう．どちらも離散対数問題を用いる方式なので，各個人は同じ秘密鍵と公開鍵を用いることができる．さらに，利用する乱数 r は併用する．プログラムを用いて以下を実現しよう．演習 *3.1* の秘密鍵，公開鍵を受信者の公開鍵，秘密鍵とする．
(1) m = 123 75081 11111 51459 12345 17144 99640 33333 55555 44444
r = 123 45678 91234 56789 12345 67891 23456 78912 34567 89123 に署名及び暗号化した結果を求めよ．また検証も確かめよ．

演習 6.2 自分の公開鍵と *TA* の公開鍵を利用して以下を実行しよう．なお，演習 *3.1* の有限体を用いて，ハッシュ関数は *Shake 128* を利用する．
(1) 今回，自分が選択したコース，*advanced* コース，*standard* コース，*beginner* コースを *TA* の公開鍵で暗号化し，署名して *moodle* に提示しよう．
(2) TA は復号し，署名を確認後，受講者にコメントを暗号化＋署名して送付する．
(3) 受講者は *TA* のコメントを復号し，内容を確認できたら，*moodle* に *OK* を記載しよう．

演習 6.3 実際の大きさに近づけて実験してみよう．現在では *2024* ビットが推奨されている．表 *4* の例 $\mathbb{F}_{p_5}, g_5 \in \mathbb{F}_{p_5}$, g_5 の位数 ℓ_5 を用いて，署名と暗号化を実現する．なお，ハッシュ関数は *Shake 256* を利用する．ここでは，署名の確認を実施する．
(1) 秘密鍵 d_b = 3707508186654514591234517144996408330622345443217508186654514591234517144996408330622345443 21に対する公開鍵 p_b を求めよ．
(2) 秘密鍵と公開鍵を用いて，
m = 123750811111151459123451714499640333335555544444750811111151459123451714499640333335555544444
r = 123 45678 91234 56789 12345 67891 23456 78912 34567 89123 45678 91234 56789 12345 67891
に署名及び暗号化した結果を求めよ．また検証も確かめよ．
(3) 各自，自分の秘密鍵を生成し，*2048* ビットの公開鍵を *moodle* に掲示する．但し，秘密鍵は *256* ビットの乱数 x($1 < x < \ell_4$) とする．

演習 6.4 自分の秘密鍵と *TA* の公開鍵を利用して以下を実行しよう．なお，演習**??**の有限体を用いて，ハッシュ関数は *Shake 256* を利用する．
(1) 本講義の感想を暗号化できるサイズまでの文章で作成し，*TA* の公開鍵で暗号化し，署名して *moodle* に提示しよう．
(2) TA は復号し，署名を確認後，受講者にコメントを暗号化＋署名して送付する．
(3) 受講者は *TA* のコメントを復号し，内容を確認できたら，*moodle* に *OK* を記載しよう．

6.2 暗号と署名の評価

暗号化と署名が DLP に基づき構築できることを学習した．では，これらの性能の違いにより，どのような場面での利用に有効なのだろうか? オンライン・オフラインは暗号でよく利用される．暗号化と署名はオンライン・オフラインで有効性は同じだろうか？また，鍵サイズの違いを学習した．鍵サイズが 2 倍になると性能（速度やメモリ）にどれくらい影響を与えるのだろうか? 理論値の評価と実験値の評価でそれらの違いを学習しよう．

解析 9 ハイブリッド暗号では暗号化と署名を実施する．暗号化と署名の速度をそれぞれ測ることで，演算の主要時間を明確にすることで，それぞれの違いを明らかにしよう．なお，ここではそれぞれの違いを見ることが目的なので，どちらも，g^r の計算量を入れる．また，計算時間ではオンラインとオフラインという考え方がある．オフラインとは平文 m などに依存しない計算で，事前にできる計算である．一方，オンラインとは，平文 m などに依存する計算で，事前にできず，リアルタイムに要求される計算である．
(1) 暗号化と署名にかかる計算量の理論値を求めよ．鍵は有限体が *1024* ビットとし，暗号化と署名のべき演算はそれぞれ独立に実施する．なお，計算量は計算量は乗算 M，*2* 乗算 S と逆元 I の計算回数で評価する．
(2) 暗号化と署名をそれぞれ独立に実施する場合，実行時間を求めて，比率を計算する．次に，理論値（*1*）と比較せよ．なお，実行時間は 10^4 回の平均を求める．
(3) 暗号化と署名の演算をそれぞれオンラインとオフラインに分ける．オンラインにかかる実行時間を比較しよう．
(4) （2）と（3）を比較することで，暗号，署名生成のなかでどの演算が主要演算となるか議論せよ．*(5)* 復号と検証にかかる計算量の理論値を求めよ．鍵は有限体が *1024* ビットとする．なお，計算量は計算量は乗算 M，*2* 乗算 S と逆元 I の計算回数で評価する．
(6) 復号と検証にかかる実行時間を求める．次に，理論値（*5*）と比較せよ．なお，実行時間は 10^4 回の平均を求める．

解析 10 *2* つの鍵サイズ *1024*，*2048* ビットで暗号と署名を構成した．鍵サイズの違いが暗号化と署名の性能に及ぼす影響を演算の主要時間を明確にすることで，明らかにしよう．なお，ここではそれぞれの違いを見ることが目的なので，どちらも，g^r の計算量を入れる．
(1) 暗号化と署名にかかる計算量の理論値を求めよ．鍵は有限体が *2048* ビットとし，暗号化と署名のべき演算はそれぞれ独立に実施する．なお，計算量は計算量は乗算 M，*2* 乗算 S と逆元 I の計算回数で評価する．
(2) 暗号化と署名にかかる計算量の理論値で，鍵は有限体が *1024* ビットに比べて，何倍になるか求めよ．

(3) 暗号化と署名が *2048* ビットの場合，実行時間を求めて，*1024* ビットの場合と比較せよ．次に，理論値（2）と比較せよ．なお，実行時間は 10^4 回の平均を求める．
(4) 復号と検証にかかる計算量の理論値を求めよ．鍵は有限体が *2048* ビットとし，暗号化と署名のべき演算はそれぞれ独立に実施する．なお，計算量は計算量は乗算 M，2 乗算 S と逆元 I の計算回数で評価する．
(5) 復号と検証にかかる計算量の理論値が鍵が有限体の *1024* ビットに比べて，何倍になるか求めよ．
(6) 復号と検証が鍵が *2048* ビットの場合の実行時間を求めて，*1024* ビットの場合と比較せよ．次に，理論値（5）と比較せよ．なお，実行時間は 10^4 回の平均を求める．

参考文献

[1] H. Cohen, A. Miyaji and T. Ono, “Efficient elliptic curve exponentiation using mixed coordinates”, *Advances in Cryptology-Proceedings of ASIACRYPT'98*, Lecture Notes in Computer Science, **1514**(1998), Springer-Verlag, 51-65.

[2] Raveen R. Goundar, Marc Joye, Atsuko Miyaji, Matthieu Rivain, and Alexandre Venelli, “Scalar Multiplication on Weierstraß Elliptic Curves from Co-Z Arithmetic”, *Journal of Cryptographic Engineering*, to appear.

[3] Patrick Longa and Catherine Gebotys, ”Analysis of Efficient Techniques for Fast Elliptic Curve Cryptography on x86-64 based Processors”,
URL: http://www.freewebs.com/patricklonga/Analysis_of_Efficient_Techniques_for_Fast_Elliptic_Curve_Cryptography_on_x86-64_based_Processors.pdf

[4] D. Stebila and N. Thëriault, “Unified point addtion formulae and side-channel attacks”, CHES'06, Lecture Notes in Computer Science, **4249**(2006), Springer-Verlag, 354–368.

[5] Y. Sakai and K. Sakurai, ”Efficient scalar multiplications on elliptic curves with direct computations of several doublings”, *IEICE Trans*., Fundamentals. vol. E84-A, No.1(2001), 120-129.

[6] Standard for efficient cryptography group, specification of standards for efficient cryptography. Available from: http://www.secg.org

[7] Bernstein, D.J., Lange, T. “Explicit-formulas database (2007)”, Accessible through, http://hyperelliptic.org/EFD

[8] Hisil, H., Carter, G., Dawson, E. “New formulae for efficient elliptic curve arithmetic”, *Progress in Cryptology-INDOCRYPT 2007*, Lecture Notes in Computer Science, **4859**(2007), Springer-Verlag, 138–151.

[9] Marc Joye and Jean-Jacques Quisquater, “Hessian Elliptic Curves and Side-Channel Attacks”, CHES 2001, Lecture Notes in Computer Science, **2162**(2001), Springer-Verlag, 402–410.

[10] “Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC”, NIST Special Publication 800-38D (2007). Available from: `http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf`.

A 数値例

演習問題利用する数値例を列挙する．

B Python の 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．また本書では，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 4.4」(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，対話的にに Python を実行する「IPython」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．

スクリプト言語としての Python: 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．

IPython: Python を対話的 (Interactive) に実行するシェルである．[スタート] - [Anaconda3] - [IPython] で実行し，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．つまり，このファイルは Jupyter Notebook 以外で実行できない．

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の二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．

B.4 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 対話モードが起動する．
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．

B.5 基本的な文法

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

for 文

Python の for 文は C 言語と少し異なり，書式は次のようになる．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．

```
for 変数 in オブジェクト:
   実行する処理 1
   実行する処理 2
   ⋮
```

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．range 関数は連番でリストを作成する関数であり，range(5) は 0≤x<5 となる整数 x のリストを作成する．

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

結果：10

while 文

書式は次のようになる．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．

```
while 条件式:
   条件式が真の時に実行する処理 1
   条件式が真の時に実行する処理 2
   ⋮
```

while 文のサンプルコードを以下に記す．

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

結果：10

リスト

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
len(b) # リストの要素数 3

array = [1, 2, 3, 4]
# 任意の要素を取り出す
first = array.pop(0) # first == 1, array == [2, 3, 4]
# 任意の要素を追加する
array.insert(0, 5) # array == [5, 2, 3]
# 末尾を取り出す
last = array.pop() # last == 3, array = [5, 2]
# 末尾に追加
array.append(9) # array == [5, 2, 9]
# 末尾にリストを追加
array.extend([0, 1]) # array == [5, 2, 9, 0, 1]
```

変数の初期化

Python では変数は必ず初期化して使うものと想定されている．つまり，暗黙の初期値が存在しない．初期化すべき値がない場合には「null」に相当する「None」を代入する

多倍長演算

Python では，Perl や Ruby と同様に，整数演算で値の範囲が固定長の範囲を越えるものを，自動的に多倍長整数に変換する．

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))
```

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．

```
”””
この中は
コメント
”””
```

数値

B.6 パッケージについて

パッケージから利用する関数は下表の通りである．プロセッサ時刻を求める関数 perf_counter は time パッケージに含まれる．そのため，perf_counter を使用する際には，time パッケージをあらかじめ import しておく必要がある．下表は，本稿で利用するパッケージ及びその組み込み関数名を記載している．ただし，chi2 は単純に import することはできず，from を利用する import 方法をとる必要がある．つまり，import の際には「from scipy.stats import chi2」と書く．この場合，chi2 を使用する際は from と import の間に書いた部分「scipy.stats」をプログラム内で省略できる．

パッケージ名	組み込み関数名
sympy	gcd, invert, factorint, randprime
Base64	b64encode, b64decode
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
chi2 (in scipy.stats)	ppf
time	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)
```

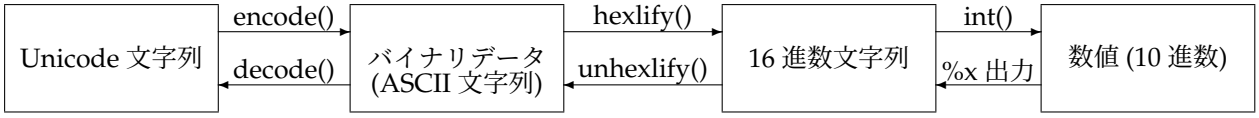
結果：3

B.7 演算における注意点

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 os.urandom(n) の出力はバイナリデータで得られる．そのため，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 進数の文字列に変換する（バイナリデータは 16 進数の文字列とバイト単位で対応）．その後，16 進数の文字列を 10 進数の数値に変換する．

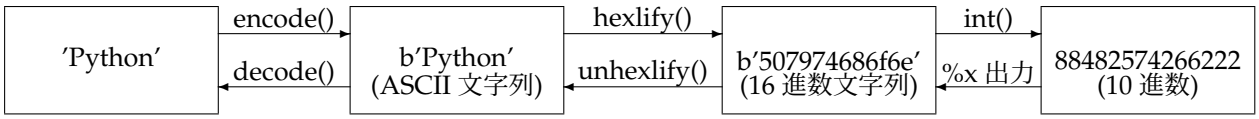
16 進数文字列は単なる中継の役割ではなく，主なメリットが 2 つある．1 つは，バイナリデータをバイト単位で可視化できる点であり，もう 1 つは文字列として保存する際に 10 進数に比べてかなり短くできる点である．もちろんバイナリデータで保存するのが最もデータサイズを短くできるが，テキストデータの方が扱いやすいという場合がある．

なお, Python 3 から 'Python' といったテキスト (Unicode) 文字列はバイト型ではなく str 型で扱われるため, そのような文字列を整数値などに変換するには, encode() を用いて ASCII 文字列 (バイト型) に変換する必要がある. 逆に, ASCII 文字列をテキスト文字列に変換するには decode() を用いる.

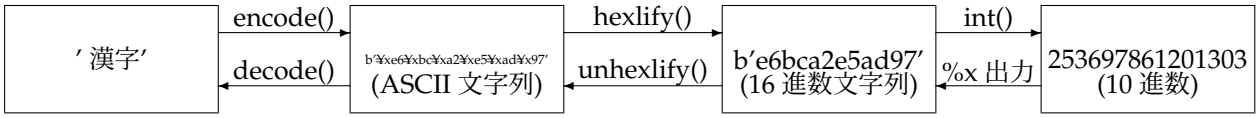


B.8 文字列における注意点

16 進数文字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文字列に過ぎないということに注意する．例えば'Python' という文字列（内部ではバイナリデータとして扱われる）は，16 進数文字列では「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 進数文字列は単なる 16 進数で表現された文字列であり，これをオリジナルデータの文字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文字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．



文字列から変換した整数値は, 文字列がアルファベットの場合, 1 文字につき 1 バイトであるが, 文字列が漢字やひらがな, カタカナの場合は, 1 文字につき 3 バイトとなる.



B.9 組み込み関数

Enthought Python Distribution でデフォルトで組み込まれている最低限必要な関数を例を用いて紹介する．

商の計算

100 を 13 で割った商の計算は次のように入力する．

```
100 // 13
```

結果：7

mod の計算

100 (mod 13) = 9 の計算は次のように入力する．

```
100 % 13
```

結果：9

べき乗法演算

2^{10} (mod 13) = 10 の計算は次のように入力する．

```
pow(2, 10, 13)
```

結果：10

最大公約数 (GCD)

12 と 30 の最大公約数の計算は次のように入力する．ただし，sympy の import が必要．

```
sympy.gcd(12, 30)
```

結果：6

逆元演算

法 13 における 7 の逆元演算は次のように入力する．ただし，sympy の import が必要．

```
sympy.invert(7, 19)
```

結果：11

素因数分解

素因数分解は次のように入力すると，結果が小さい素数でソートされリスト形式で返される．例えば，96525 を素因数分解すると $3^3 \times 5^2 \times 11^1 \times 13^1$ となる．最も大きな素因数を取り出したい場合は，例のように，len() を使ってリストの長さを導出し，リストの最後の素因数を取り出せばよい．ただし，sympy の import が必要である．

```
v = sorted(sympy.factorint(96525).items())
print(v)
n = len(v)
print(v[n-1])
```

結果：[(3, 3), (5, 2), (11, 1), (13, 1)], (13, 1)

文字列 (10/16 進数) から 10 進数の数値への変換

`int()` を使用すると，次のように 10 進数の文字列を 10 進数の数値に変換できる．

```
int('14') #→ 14
int('84') #→ 84
```

さらに，`int()` の 2 番目の引数に基数を指定できる．例えば，14 を 16 進数の文字列として捉えて 10 進数の数値に変換すると 20 である．

```
int('14', 16) #→ 20
int('a', 16) #→ 10
```

10 進数の数値から文字列 (10/16 進数) への変換

10 進数の数値を 10 進数の文字列に変換するサンプルコードを次に記す.

```
val = 255
s = ″%d″ %val
print(s)
```

結果：255

さらに，10 進数の数値を 16 進数の文字列へ変換するサンプルコードを次に記す.

```
val = 255
s = ″%x″ %val
print(s)
```

結果：ff

バイナリデータと 16 進数の文字列の相互の変換

```
binascii.hexlify(data)
binascii.unhexlify(hexstr)
```

`hexlify()` はバイナリデータを 16 進数の文字列に変換し，逆に `unhexlify()` は 16 進数の文字列をバイナリデータに変換する．次にサンプルコードを示す．ただし，下記の'Python' は Unicode 文字列として認識されていることに注意する.

```
import binascii
v1 = binascii.hexlify('Python'.encode())
if (len(v1)%2 == 0):
   v2 = binascii.unhexlify(v1).decode()
else:
   v2 = binascii.unhexlify('0'+v1).decode()
print(v1)
print(v2)
```

結果 1：b'507974686f6e'， 結果 2：'Python'

バイナリデータと 16 進数文字列はバイト単位で対応するため，`unhexlify()` の入力は偶数個の 16 進数文字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 進数文字列としての 0 をパディングする.

16 進数文字列の一部を取り出す

ハッシュ関数 SHA1 の出力 20 バイトに対して，上位の 10 バイトと下位の 10 バイトに分けて取り出すには次のように入力する．ただし，16 進数文字列は 0〜9 及び a〜f の中の 2 文字で 1 バイトを表すことに注意する.

```
import hashlib
v = hashlib.sha1('Python'.encode()).hexdigest()
v1 = v[0:20]
v2 = v[20:40]
print(v1)
print(v2)
```

結果 1：'6e3604888c4b4ec08e28'， 結果 2：'37913d012fe2834ffa83'

16 進数文字列の連結

「+」を利用すると，次のように簡単に文字列の連結ができる.

```
v1 = '6e3604888c4b4ec08e28'
v2 = '37913d012fe2834ffa83'
v = v1 + v2
print(v)
```

結果：'6e3604888c4b4ec08e2837913d012fe2834ffa83'

乱数生成

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

`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 を使う．本関数は，プロセス ID やスレッド ID，システムクロック等のシステム情報を利用することで擬似乱数を生成するものであり，乱数のシードとしても利用できる．`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．urandom() を使用する際には文頭で「os」をインポートし，seed() を使用する際には文頭に「random」をインポートしておく．

以下に乱数を生成するサンプルコードを示す．このサンプルコードは，8 バイト乱数を 16 進数の文字列で出力する．

```
import os
import binascii
myseed = binascii.hexlify(os.urandom(8))
print(myseed)
```

上記サンプルコードでは，16 進数の文字列で表示させるために hexlify() を使用している．

素数生成

```
sympy.randprime(a, b)
```

`randprime(a, b)` は, 整数 a 以上 b 未満の素数の中でランダムなものを 10 進数の数値で返す．これはチェビシェフの定理を実装したものである．使用する際には，文頭で sympy をインポートしておく．

以下に素数乱数を生成するサンプルコードを示す．このサンプルコードは，2 以上 100 未満の素数をランダムに 1 つ出力する．また，乱数のシードとして 1234 を与えている．

```
import sympy
import random
random.seed(1234)
v = sympy.randprime(2, 100)
print(v)
```

ハッシュ関数

```
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.encode()).hexdigest()` は，入力である任意長の文字列 s 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.encode()).hexdigest()` は，入力である任意長の文字列 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.encode()).hexdigest(length)`，`shake_256(s.encode()).hexdigest(length)` が用意されている．hashlib を import しておく必要がある．length の単位はバイトである．正確には，length バイト以下のハッシュ値が計算される．そのため，length を必要なサイズ以上に指定しておき，ハッシュ値の計算後に必要な長さだけ切り出すという方法もある．

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，文字列'Python' のハッシュ値を導出するものである．'Python' は内部では Unicode 文字列 (文字列型) として扱われるため, encode() で ASCII 文字列 (バイト型) に変換する必要がある．

```
import hashlib
v=hashlib.sha1('Python'.encode()).hexdigest()
print(v)
```

結果 1：'6e3604888c4b4ec08e2837913d012fe2834ffa83'

PBL では以下のような，shake128 と shake256 のハッシュ値を計算する関数を利用する．

```
import hashlib, math
# ハッシュ関数 Shake128 によるハッシュ値の計算
# Input: メッセージ m (通常の文字列), ベースポイントの位数 l (整数)
# Output: shake128 による m のハッシュ値 (16 進文字列）
def shake128(m, l):
return hashlib.shake_128(m.encode('utf-8')).hexdigest(math.ceil(l.bit_length()/8)-1)
```

```
import hashlib, math
# ハッシュ関数 Shake256 によるハッシュ値の計算
# Input: メッセージ m (通常の文字列), ベースポイントの位数 l (整数)
# Output: shake256 による m のハッシュ値 (16 進文字列）
def shake256(m, l):
return hashlib.shake_256(m.encode('utf-8')).hexdigest(math.ceil(l.bit_length()/8)-1)
```

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

結果 1：'9be4fd318b9d7d6a6104769ef012618d9f45bda7'

文字列と整数値の相互変換

文字列と整数値の相互変換の方法を紹介する．int.from_bytes() 関数と to_bytes() 関数を利用する．まず，整数値を文字列に変換する方法を述べる．変換する文字列を encode() 関数を用いて ASCII 文字列に変換し，int.from_bytes() 関数の第 1 引数とする．int.from_bytes() 関数の第 2 引数には，byteorder = ″big″ または byteorder = ″little″ を指定する．big は最上位バイトがバイト配列の最初にくるようなバイトオーダー（ビッグエンディアン）を指し，little は最上位バイトがバイト配列の最後にくるようなバイトオーダー（リトルエンディアン）を指す．以上で，文字列を整数値に変換できる．次に，整数値を文字列に変換する方法を述べる．変換する整数値を m としたとき，m を m.to_bytes(length, byteorder = ″big″ (or little)) により length バイトのバイト列に変換する．ここで，int.from_bytes() 関数と to_bytes() 関数とで byteorder を合わせる必要があることに注意する．m を変換したバイト列を decode() 関数で文字列に変換すれば，m に対応する整数値が得られる．以下はサンプルコードである．

```
# 文字列 → 整数値
m = 'ProSec/SecCap/BasicSecCap'
m_byte = m.encode()
m_int = int.from_bytes(m_byte, byteorder = 'big')
print(m_int)
# 504974073587453300352066126476159956376684307532229042725232
と表示
```

```
# 整数値 → 文字列
m_int = 504974073587453300352066126476159956376684307532229042725232
m_byte = m_int.to_bytes(32, byteorder = ‘ big ’ )
m = m_byte.decode()
print(m) # ProSec/SecCap/BasicSecCap と表示
```

以下の方法でも文字列と整数値の相互変換が可能である．結果はどちらも変わらない．

```
import binascii # 文字列 → 整数値 M = “ ProSec/SecCap/BasicSecCap ”
Mb = binascii.hexlify(M.encode())
M_int = int(Mb, 16)
print(M_int) # 504974073587453300352066126476159956376684307532229042725232
```

```
# 整数値 → 文字列
M_int = 504974073587453300352066126476159956376684307532229042725232
Mh = hex(M_int)[2:].encode()
M = binascii.unhexlify(Mh).decode()
print(M) # ProSec/SecCap/BasicSecCap と表示
```

カイ自乗分布表の導出

```
chi2.ppf(1-α, n)
```

`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．

```
from scipy.stats import chi2
v=chi2.ppf(1-0.005, 10)
print(v)
```

結果：'23.209251159'

時間計測

時間計測には `time.perf_counter()` を利用する．この関数は、スリープ中の経過時間も含め、パフォーマンスカウンターの値 (小数点以下がミリ秒) を返し、その値はシステム全体で一意である。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]
p = 184908779667576050572947262326548513689
t0 = time.perf_counter()
for i in range(1000):
   a = int(binascii.hexlify(os.urandom(20)), 16)
   pow(2, a, p)
t1 = time.perf_counter() - t0
print(t1/1000)
```

結果：'0.000131047400794'

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．測定結果を見ても，悪い例では 0.13msec，良い例では 0.12msec と明らかに差があることが分かる．

```
[良い例]
p = 184908779667576050572947262326548513689
a = [ ]
for i in range(1000):
   a.append(int(binascii.hexlify(os.urandom(20)), 16))
t0 = time.perf_counter()
for i in range(1000):
   pow(2, a[i], p)
t1 = time.perf_counter() - t0
print(t1/1000)
```

結果：'0.00012201551483'

B.10 自作関数

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):
   t = a
   a = b
   b = t
   return (a, b)
```

```
print(ex(1, 3))
```
結果：(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という特徴を持つ．異なる型の値を代入すれば，その都度変数の型が変わる．

B.11 ファイル入出力

ファイル入出力では，データがテキストであるかバイナリであるかで扱いが異なる．これに対して，以下の `dump()` と `load()` をペアで使用することによってデータの種類を意識することなくファイル入出力ができる．データ d をファイル data.txt に出力するには以下のように書く．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
f = open('data.txt', 'wb')
   pickle.dump(d, f)
f.close()
```

次に，ファイル data.txt からデータを読み込む場合は以下のように書く．

```
f = open('data.txt', 'rb')
   pickle.load(f)
f.close()
```

B.12 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．基本的な CSV ファイル出力は次のように書く．このサンプルコードは，リストデータをコンマ繋がりの CSV 形式に変換して，指定のファイルに 1 行ずつ書くコードである．ファイルを開く際には，バイナリモード ("wb") を指定した方が無難である．csv を import しておく必要がある．

```
import csv
fh = open("file.csv", "wb")
writer = csv.writer(fh)
writer.writerow([1, 'RSA', 'A'])
writer.writerow([2, 'AES', 'S'])
writer.writerow([3, 'RC4', 'S'])
fh.close()
```

```
ファイル内の結果：
1,RSA,A
2,AES,S
3,RC4,S
```

さらに，上記ファイルをリスト形式で読み込むサンプルコードは以下である．

```
import csv
a = [ ]
fh = open("file.csv", "rb")
reader = csv.reader(fh)
for row in reader:
   a.append(row)
fh.close()
print(a)
```

結果：[['1', 'RSA', 'A'], ['2', 'AES', 'S'], ['3', 'RC4', 'S']]

B.13 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．

```
plt.axis([0.0, 5.0, -1.5, 1.5])
plt.grid(True)
x = np.arange(0, 5, 0.01)
y=sin(x)
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．

```
pylab.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.show()
```

ただし，上記グラフ描画ではパッケージを import する必要がない．

B.14 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．

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”    # ”Hello” と表示される
```

Python3
```
print(”Hello”)    # ”Hello” と表示される
```

a/b の仕様変更

int 型同士の割り算 (/) 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    # 結果は 1
```

Python3

```
3/2    # 結果は 1.5
3//2    # 結果は 1
```

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'
```

B.15 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)
```

C　問題

問題 1

★ ★ ★，★★，★　の順に解きましょう．※ ★ ★ ★ は利用する問題, ★★ は重要な問題, ★ は典型的な問題

★★**(1)** $y = x^2 + 3$ で，$x = 3$ とするときの y の値を，「$y = **$」の書式で出力せよ．
（「$y =$」も出力すること）

★★**(2)** P = [12345678, 9123456] とし，(P の 1 つ目の要素÷ 2)+(P の 2 つ目の要素÷ 2) を出力せよ．

問題 2

※ ★★ は重要な必須問題, ★ は重要な問題

★★**(1)** 1 より大きい偶数を小さいほう方から 10 個，while 文を用いて求めよ．
（解答はリストで出力すること）

★**(2)** 50 より小さい奇数を for 文を用いて求めよ．
（解答はリストで出力すること）

問題 3

x = 1234567890 に対して，偶数なら 2 で割り奇数なら 1 を引いて 2 で割る処理を繰り返した結果，何回でゼロになるかを求めよ．

問題 4

※ ★★ は重要な必須問題, ★ は重要な問題

★★**(2)** 10 進表記の 15643454453 を 2 進表記に変換せよ．

★★**(2)** 2 進表記の 1010101010101010 を 10 進表記に変換せよ．

問題 5

(1) 2 進表記の 11111111 を 8 桁だけ右シフトし，2 進表記で表示せよ．

(2) 2 進表記の 1 を 8 桁だけ左シフトし，2 進表記で表示せよ．

問題 6

以下の値の素因数分解を求めよ．また，素因数分解にかかる時間を調べよ．(計算時間は 100 回実施し, その平均を求める.)

(1) 98765432198765432198765431987654321

(2) 828460094429076562945148777

問題 7

★ for 文を用いて，「2010 年 1 月」から「2010 年 12 月」までの文字列を表示させよ．

問題 8

※ ★★ は重要な必須問題, ★ は重要な問題

★★**(1)** a と b (a>b) を引数とし，a ÷ b の商と余りを戻り値として出力する関数 qr(a, b) を作成せよ．また関数 qr を利用して 123456789 ÷ 12345 の商と余りを求めよ．

★★**(2)** 再帰呼び出し (ある手続き中で再びその手続き自身を呼び出すこと) を用いて，階乗 n!を出力する関数 fact(n) を作成せよ．また関数 fact を利用して 10! を求めよ．

(3) 自然数 n を入力とし, フィボナッチ数列 ($a_1 = 1, a_2 = 1, a_n = a_{n-1} + a_{n-2}$) の第 n 項を出力する関数 fibo(n) を作成せよ．また隣り合う 2 項の比 (a_{n+1}/a_n) が黄金比 $\frac{1+\sqrt{5}}{2} \approx 1.61803$ と小数点以下 3 桁まで一致するときの n を求めよ．

問題 9

★**(1)** n ビット以上の任意のビット長の乱数文字列 r から上位 n ビットを抽出する関数 upper_bits(r, n) を作成し，r = 1001101011 (文字列)，n = 6 のときの出力を求めよ．ただし，ビット操作で実現すること．

(2) n ビット以上の任意のビット長の乱数文字列 r から下位 n ビットを抽出する関数 lower_bits(r, n) を作成し，r = 1001101011 (文字列)，n = 6 のときの出力を求めよ．ただし，ビット操作で実現すること．

問題 10

★★**(1)** 1 から 50 の間の乱数 100 個を 2 セット生成し, それぞれをファイル名 random1.csv, random2.csv の ファイルに出力するプログラムを作成せよ. (1 つのファイルに乱数 1 セット分 (乱数 100 個) を出力する.)

★★**(2)** (1) で作成した csv ファイル 2 つの数値を読み込み, 縦軸を乱数値, 横軸を乱数の個数とする折れ線グラフを図示せよ． ただし, グラフのタイトルは Random data とし, 2 セット分の乱数の折れ線グラフを同じ図に図示すること．

D Python を用いた TCP/IP 通信

本節では Python を用いて TCP/IP 通信を行う簡単な例を示す。

ソースコード 1: Ptyhon による TCP クライアントプログラム

```
import socket, pickle

# 接続先IPアドレス、ポート番号
HOST = 'localhost'
PORT = 10000

# サーバへ接続
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))

# データシリアライズ
arr = ([1,2,3,4,5,6],[1,2,3,4,5,6])
data_string = pickle.dumps(arr)

# データ送信
s.send(data_string)

# データ受信
data = s.recv(4096)
data_arr = pickle.loads(data)
s.close()

print('Received', repr(data_arr))
```

ソースコード 2: Ptyhon による TCP サーバプログラム

```
import socket

# 待受IPアドレス、ポート番号
HOST = '0.0.0.0'
PORT = 10000

# TCP待受開始
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)

# TCP接続受信
conn, addr = s.accept()
print('Connected by', addr)

while 1:
```

```
    # データ受信
    data = conn.recv(4096)
    if not data: break
    conn.send(data)
conn.close()
```

プログラム 1 は TCP クライアントの例であり、サーバへ接続し、データを送信して、受信した後に切断する単純なプログラムである。接続先サーバのアドレス及びポート番号は、4, 5 行目で行っている。この例では、ローカルに立ち上げたサーバへ接続するため、HOST が localhost となっているが、実際は、サーバの IP アドレスを指定する必要がある。8 行目では TCP socket を作成し、9 行目で接続を行う。13 行目では 12 行目で作成したデータを pickle を用いてシリアライズしている。その後、16 行目でデータ送信し、19 行目でデータ受信している。20 行目では、シリアライズされたデータを復元している。21 行目で socket をクローズし、21 行目では受信したデータを標準出力へと出力している。

プログラム 2 は TCP サーバの例であり、クライアントから受け取ったデータを、そのままクライアントへ返答する。4, 5 行目では、待ち受けする IP アドレス及び、ポート番号を指定している。ここでは、不特定の IP アドレスから受信することを想定しているため、0.0.0.0 としている。8 から 10 行目では、socket を作成してクライアントからの TCP 接続の待受を開始している。13 行目で TCP の接続を受信し、TCP コネクションを確立している。16 から 20 行目ではデータを受信して、クライアントにそのまま送信している。最後に、21 行目で TCP コネクションを切断する。

D.1 Translation

The program 1 is an example of a TCP client, a simple program that connects to a server, sends data, receives it, and then disconnects. The address and port number of the connection destination server are given on the 4th and 5th lines. In this example, HOST is localhost because it connects to the locally started server, but in reality, it is necessary to specify the IP address of the server. The 8th line creates a TCP socket, and the 9th line connects. In the 13th line, the data created in the 12th line is serialized using pickle. After that, the data is transmitted on the 16th line and the data is received on the 19th line. Line 20 restores the serialized data. The socket is closed on the 21st line, and the received data is output to the standard output on the 21st line.

The program 2 is an example of a TCP server, and the data received from the client is returned to the client as it is. The 4th and 5th lines specify the listening IP address and port number. Here, 0.0.0.0 is used because it is assumed that the message will be received from an unspecified IP address. Lines 8 to 10 create a socket to start listening for TCP connections from clients. The TCP connection is received on the 13th line, and the TCP connection is established. Lines 16 to 20 receive the data and send it to the client as is. Finally, disconnect the TCP connection on line 21.

E　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．本章は作成する関数の関数名及び関数の入出力変数などの関数仕様書を記述します．

Functions created in the exercises will be reused in some other exercises. In this case, it is recommended that the function names and input/output variables of the functions be unified to facilitate collaboration. This chapter describes the function specifications, including function names and input/output variables of the functions to be created.

Algorithm 1 ユークリッドの互除法 Euclidean algorithm: euclid(a, b)

Input: 整数 Integer: a, b ($b \neq 0$)

Output: 整数 Integer: a, b の最大公約数. The greatest common divisor of a, b.

関連演習 Related exercise: 事前問 4

Algorithm 2 拡張ユークリッドの互除法 Extended Euclidean algorithm: ex_euclid(a, b)

Input: 整数 Integer: a, b ($b \neq 0$)

Output: 整数のリスト List of integers: $[d = ax + by, x, y]$ (d は a, b の最大公約数) (d is the GCD of a, b)

参考関数 Related function： divmod

関連演習 Related exercise: 事前問 5, 解析 1

Algorithm 3 バイナリユークリッドの互除法 Binary Euclidean algorithm: bin_euclid(a, b)

Input: 整数 Integer: a, b ($b \neq 0$)

Output: 整数 Integer: a, b の最大公約数. The GCD of a, b.

関連演習 Related exercise:

Algorithm 4 バイナリ拡張ユークリッドの互除法 Binary extended Euclidean algorithm: bin_ex_euclid(a, b)

Input: 整数 Integer: a, b ($b \neq 0$)

Output: 整数のリスト List of integers: $[d = ax + by, x, y]$ (d は a, b の最大公約数) (d is the GCD of a, b)

関連演習 Related exercise: 演習 1.2

Algorithm 5 逆元 (拡張ユークリッドの互除法を利用） Inverse by extended Euclid algorithm: inv(a, n)

Input: 整数 Integer: a, n (n は素数と合成数のどちらも取りうる) (n can be both a prime and a composite number)

Output: 整数 Integer: $a^{-1} \bmod n$

自作関数 Self-defined function: ex_euclid

関連演習 Related exercise: 事前問 6, 演習 1.1

Algorithm 6 バイナリ拡張ユークリッドの互除法を利用した逆元 Inverse using Binary extended Euclidean algorithm: bin_inv(a, n)

Input: 整数 Integer: a, n (n は素数と合成数のどちらも取りうる) (n can be both a prime and a composite number)

Output: 整数 Integer: $a^{-1} \bmod n$

自作関数 Self-defined function: bin_ex_euclid

関連演習 Related exercise: 演習 1.2, 解析 1

Algorithm 7 逆元 (Fermat の小定理を利用） Inverse by Fermat's little theorem: inv(a, n)

Input: 整数 Integer: a, n (n は素数と合成数のどちらも取りうる) (n can be both a prime and a composite number)

Output: 整数 Integer: $a^{-1} \bmod n$

自作関数 Self-defined function: mod_binary

関連演習 Related exercise: 演習 2.1

Algorithm 8 バイナリ法 Exponentiation by squaring: binary(g, k)

Input: 整数 Integer: g, k

Output: 整数 Integer: a^k

参考関数 Related function: bin, len

関連演習 Related exercise: 事前演習 1

Algorithm 9 法 p 上のバイナリ法 Exponentiation by squaring (mod p): mod_binary(g, k, n, p)

Input: 整数 Integer: g, k, p (p は素数) (p id a prime number), p のビット長: n

Output: $g^k \bmod p$ (ループ数が n になるように実装) (k のビット数が n 以上の場合はエラーとなる）

参考関数 Related function: bin

関連演習 Related exercise: 演習 1.3

Algorithm 10 拡張ユークリッドの互除法のループ回数計測 Counting the loops of extended Euclidean algorithm: exEuclidExp(a, b)

Input: 整数 Integer: a, b ($b \neq 0$)

Output: 整数のリスト List of integers: [$d = ax + by, x, y, counter$], counter はループの実行回数． Counter is the number of loop executions.

関連演習 Related exercise： 解析 1

Algorithm 11 逆元計算のループ回数計測 (拡張ユークリッドを利用) Counting the loops of inverse (using extended Euclidean algorithm): invExp(a, n)

Input: 整数 Integer: a, n ($n \neq 0$ は素数と合成数のどちらも取りうる) ($n \neq 0$ can be both a prime and a composite number)

Output: 整数のリスト List of integers: [$a^{-1} \bmod n, counter$], counter はループの実行回数． Counter is the number of loop executions.

自作関数 Self-defined function：exEuclidExp

関連演習 Related exercise： 解析 1

Algorithm 12 最大値・最小値、平均・分散 Maximum, minimum, mean, and variance: statistics(data)

Input: 数値のリスト List of numbers：data

Output: data の最大値・最小値・平均・分散のリスト The maximum, minimum, mean, and variance of the input: [max, min, mean, var] (max: 最大値, min: 最小値, mean: 平均, var: 分散)

参考関数 Related function： max, min, numpy.mean, numpy.var

関連演習 Related exercise： 解析 1, 2, 4

表 1: 1,024 ビット素体 1

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	=								7447712589	1023456789
g_3	=				2295531362	3907650873	5759914822	7447712589	3907650873	1023456789
g_4	=	7447712589	2295531362	3907650873	5759914822	7447712589	3907650873	1023456789	1023456789
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

表 2: 1,024 ビット素体 3

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
x(秘密鍵)	=					74819354	4676408372	5846279890	4920765777	7532261418
y(公開鍵)	=				42063703	3882610340	6715590119	3984173769	4975880909	6556356897	3262694193
		0548625363	6357766213	2973611138	7447766588	7793895764	7154042547	2897435807	8527271348
		5080760582	1175462811	2594432006	5830421529	7187565565	6066325474	7031747130	4257851754
		6526803538	4441103374	2675004000	6850346436	0481409506	8411389122	4005454945	5335321369

表 3: 1,024 ビット素体 4, 128 ビット素数 ℓ_6 を位数とする g_6

p_6	=		100251188	1435484931	4129174696	5544229225	7574408276	0416911916	9039308639
		5663857221	7744321632	9113417770	5075700315	5089153824	0309704371	9659968254	7463206575
		3566025088	5806797834	6840005733	5825311704	9494100524	0693110169	0805231456	0872602803
		3171408413	3300640894	5969509919	1202300969	7129516056	9189039274	1570026469	7767312889
ℓ_6	=					213058362	1616801677	8548157647	0646580249
g_6	=		20425253	1084207370	9317663392	5511085727	2652145152	4391128993	0000311194
		3123855172	2742411200	0607294361	6421633392	1344625860	4135594225	3993039274	0719510001
		2149910324	2165120894	5840210890	1558103561	5763376457	3126889318	7829782868	1659326993
		4365587642	1518076008	0073790717	3991173457	5931711821	8836964130	4575959144	9246045376

表 4: 2048 ビット素体, 256 ビット素数 ℓ を位数とする g

p_5	=			1834164	2337870524	9338236332	2853594220	0920940262	0119392757
		1459838196	1660313782	9190599736	0221064025	3913064738	2645620265	7192657169	4791447723
		3838062926	6858123708	1513058071	9194253799	0311440320	1489579624	9049762487	7439255641
		7251490522	1528634891	6697895313	7440789189	0651492896	5331481166	4942749220	3322550039
		5352641074	6867281015	8988556133	5915494681	8438790076	2440680784	1660707046	9996045001
		5065581719	3099579695	0602413953	9446644253	5767400953	7407430033	3491198520	1074406769
		9961120921	9791268245	4839213167	8646898409	9293056743	7004010176	2224629825	5200505829
		9567227979	0699062782	6504904587	3537856787	6320604422	8715706968	2571700902	2140607949
ℓ_5	=	6737535	2361764453	2763385633	7618307891	1781662844	8070414868	0052216871	4826655153
g_5	=			442590	3187308506	1597828245	2539974420	0677344484	8487023204
		6020155931	0549373102	1208554911	9683779176	4335096573	1557048234	7216764981	9727327902
		9459367366	2676601474	2753021252	0905574312	6056019582	3104005779	1263613948	3538319553
		5594994747	6577684801	9733469077	0031595912	1629749392	4598681355	1433619557	2695251201
		3534150628	7381625991	6679500326	9601651907	6350431024	8893096051	0155562010	8824595567
		4143933908	2368076776	3689806245	7510508780	4347708156	8799777543	6030359112	5391765456
		3716914432	9096871977	6279674791	9529512749	6050876529	1309568409	6720040377	6451616391
		0326962278	8472118912	2447617877	5556510332	7988784972	7968647523	8934197372	7065980263

Algorithm 13 バイナリ拡張ユークリッドの互除法のループ回数計測 Counting the loops of binary extended Euclidean algorithm: bin_exEuclidExp(a, b)

Input: 整数 Integer: a, b ($b \neq 0$)

Output: 整数のリスト List of integers: [$d = ax + by, x, y, counter$], counter はループの実行回数. Counter is the number of loop executions.

関連演習 Related exercise： 解析 2

Algorithm 14 逆元計算のループ回数計測 (バイナリ拡張ユークリッドを利用) Counting the loops of inverse (using binary extended Euclidean algorithm): bin_invExp(a, n)

Input: 整数 Integer: a, n ($n \neq 0$ は素数と合成数のどちらも取りうる) ($n \neq 0$ can be both a prime and a composite number)

Output: 整数のリスト List of integers: [$a^{-1} \bmod n, counter$], counter はループの実行回数. Counter is the number of loop executions.

自作関数 Self-defined function：bin_exEuclidExp

関連演習 Related exercise： 解析 2

Algorithm 15 バイナリ法の 2 乗算・乗算回数計測 Counting the square and multiplication of exponentiation by squaring: modBinaryExp(k)

Input: 指数 (整数) Exponent (Integer): k

Output: 整数のリスト List of integers: [square_count, mult_count], (整数を k 乗する際の)square_count は 2 乗算の回数, mult_count は乗算回数. square_count is the number of square calculation, and mult_count is the number of multiplication during the exponentiation.

参考関数 Related function： bin, len

関連演習 Related exercise： 演習 2.1, 解析 4

Algorithm 16 逆元計算のループ回数計測 (バイナリ法利用) invExp_modBinary(a, p)

Input: 整数 Integer: a, p (p は素数)

Output: 整数のリスト List of integers: [a^{-1} mod n, *counter*], counter はループの実行回数

自作関数 self-defined function:mod_binary

関連演習 Related exercise：演習 2.1, 解析 4

Algorithm 17 平均・分散 Mean and variance: mv(data)

Input: 数値のリスト List of numbers：data

Output: data の平均・分散のリスト List of mean and variance of input: [mean, var] (mean: 平均, var: 分散)

参考関数 Related function： numpy.mean, numpy.var

自作関数 self-defined function: invExp_exEuclid

関連演習 Related exercise： 解析 4

Algorithm 18 平方根を求める find_square_root(a, p)

Input: 整数 integer：a, 素数 prime number congruent to 3 modulo 4：p ($p \equiv 3 \pmod 4$)

Output: $x^2 \equiv 2 \pmod p$ を満たす 整数 x integer x satisfying $x^2 \equiv 2 \pmod p$

参考関数 Related function： mod_binary

関連演習 Related exercise： 演習 2.2

Algorithm 19 実行時間の計測 Measuring the time: time_check(t_s, t_e)

Input: 計測開始地点 Start of the measurement：t_s, 計測終了地点の時刻 End of the measurement：t_e

Output: $t_e - t_s$

参考関数 Related function：time.perf_counter

関連演習 Related exercise： 解析 4, 9, 10

Algorithm 20 バースデイパラドックス 1(真値) Birthday paradox 1 (true value): birthday_paradox1(m, n)

Input: 整数 Integer：m, n

Output: m 種類の異なるデータから n 個のデータを一様に入手した際に同じデータが少なくとも 1 つ存在する確率 Probability of having at least one identical data when n data are uniformly available from m different data types.

参考関数 Related function: (scipy.special.perm, pow)

関連演習 Related exercise: 問 2.1, 解析 4

Algorithm 21 バースデイパラドックス 2(近似) Birthday paradox 2 (approximation): birthday_paradox2(m, n)

Input: 整数 Integer：m, n

Output: m 種類の異なるデータから n 個のデータを一様に入手した際に同じデータが少なくとも 1 つ存在する確率 (近似) Probability of having at least one identical data when n data are uniformly available from m different data types. (approximation)

参考関数 Related function: math.exp

関連演習 Related exercise: 問 2.1, 解析 5

Algorithm 22 データの図示 Visualize data: draw_fig(num_data, data_x, data_y, x_lavel, y_label, color, fig_label, legend, fig_name)

Input: データ数 (グラフの数) Number of data (number of graphs): num_data, x 軸の数値 (float 型) のリスト list of floats of x-axis： data_x, num_data 個の y 軸の数値のリスト list of values of y-axis (length is num_data): data_y, x 軸のラベル (文字列) label of x-axis (string): x_label, y 軸のラベル (文字列) label of y-axis (string): y_label, num_data 通りのグラフの色 (文字列のリスト) graph color as per num_data (list of strings): color, num_data 個のグラフのラベル (文字列のリスト) num_data labels for graphs (list of strings): fig_label, 凡例の位置 (文字列) location of legend (string): legend, 図の名前 (文字列) name of figure (string): fig_name

Output: data_x の数値を横軸, data_y の数値を縦軸としたグラフ A graph with data_x on x-axis and data_y on y-axis.

参考関数 Related function: matplotlib.pyplot.figure, matplotlib.pyplot.xlabel, matplotlib.pyplot.ylabel, matplotlib.pyplot.scatter, matplotlib.pyplot.legend, matplotlib.pyplot.savefig

関連演習 Related exercise: 問 2.1, 解析 5

Algorithm 23 ElGamal 暗号の暗号化 (一部のみ) Encryption of ElGamal cipher (only partially): elgamal_enc_part(m, p, r, pk)

Input: 平文 (整数) Plaintext (integer)：m, 法 mod：p, 乱数 random number：r, 公開鍵 public key：pk

Output: ElGamal 暗号の暗号文の片方 One side of the ciphertext of the ElGamal cipher：$c = \mathrm{pk}^r m \pmod p$

参考関数 Related function:

自作関数 Self-defined function: mod_binary

関連演習 Related exercise: 解析 5

Algorithm 24 ElGamal 暗号の暗号文のファイル出力 ElGamal encryption to file: elgamal_enc_file_output(m, p, sample_num, c, pk)

Input: 平文 (整数) Plaintext (integer)：m, 法 mod：p, (出力する) 暗号文の数 number of output ciphertext：sample_num, ファイル名 (文字列, csv ファイル) filename (string, extension is csv)：c

Output: sample_num 個の暗号 ww 文 (の片方) sample_num ciphertexts (one of them)：$c = \mathrm{pk}^r m \pmod p$ を c.csv に出力 output in c.csv.

参考関数 Related function: random.randint (secrets.choice), open, csv.writer

自作関数 Self-defined function: elgamal_enc_part

関連演習 Related exercise: 解析 5

Algorithm 25 ElGamal 暗号の暗号文の出現回数計測 Number of occurrences of ElGamal ciphertext: elgamal_enc_collision(m, p, pk, experiments_num, sample_num)

Input: 平文 (整数) Plaintext (integer)：m, 法 mod：p, 公開鍵 public key：pk, 実験回数 number of experiments：experiments_num, 乱数の数 number of random numbers：sample_num

Output: 各実験での同じ暗号文 (の片方) List of the number of encryptions until the same ciphertext (one of) $\mathrm{pk}^r m \pmod p$ が得られるまでの暗号化回数のリスト is obtained for each experiment

参考関数 Related function: random.randint (secrets.choice)

自作関数 Self-defined function: elgamal_enc_part

関連演習 Related exercise: 解析 5

Algorithm 26 リストに含まれる要素数のカウント Count of the number of elements in the list: count_list(n_min, n_max, L)

Input: 整数値のリスト List of integer: L, L の要素の最小値 minimum value of L elements: n_min, L の要素の最大値 maximum value of L elements: n_max

Output: 区間 [n_min, n_max] 内の各整数値が L に何個含まれているかカウントしたリスト Interval[n_min, n_max] List of counts of how many of each integer value are contained in L

参考関数 Related function: collections.Counter (count)

自作関数 Self-defined function:

関連演習 Related exercise: 解析 5

Algorithm 27 ElGamal 暗号化で同じ暗号文が出る確率 elgamal_enc_collision_prob(col_list, experiments_num)

Input: 各暗号化回数における暗号文の衝突数のリスト: col_list, 実験回数: experiments_num

Output: 各暗号化回数における暗号文の衝突確率

参考関数 Related function:

自作関数 Self-defined function: 関連演習 Related exercise: 解析 5

Algorithm 28 ElGamal 暗号の鍵生成 ElGamal key generation: elgamal_key_gen(g, p, sk)

Input: ベースポイント Base point：$g \in \mathbb{F}_p$, 法 mod：p, 秘密鍵 secret key：sk(いずれも整数) (both are integer)

Output: 公開鍵 Public key (integer) $\mathrm{pk} = g^{\mathrm{sk}} \mod p$ なる整数 pk

自作関数 Self-defined function: mod_binary

関連演習 Related exercise: 演習 3.1, 3.2, 6.1

Algorithm 29 ElGamal 暗号の暗号化 ElGamal encryption: elgamal_enc(m, g, p, r, pk)

Input: 平文 (整数) Plaintext (integer)：m, 法 mod：p, 乱数 (整数) random number (integer)：r, ベースポイント base point：g, 公開鍵 public key：pk

Output: 暗号文 (整数の組) Ciphertext (list of integers)：$[u = g^r \pmod p, c = \mathrm{pk}^r m \pmod p]$

自作関数 Self-defined function: mod_binary

関連演習 Related exercise: 演習 3.1, 3.2

Algorithm 30 ElGamal 暗号の復号 ElGamal decryption: elgamal_dec(u, c, p, sk)

Input: 暗号文 (整数の組) Ciphertext (list of integers)：$[u, c]$, 法 mod：p, 秘密鍵 secret key：sk

Output: 平文 (整数) Plaintext (integer)：$m = c(u^{\mathrm{sk}})^{-1}$

自作関数 Self-defined function: mod_binary, inv

関連演習 Related exercise: 演習 3.1, 3.2

Algorithm 31 DH 共有鍵生成関数 DH shared key generation: dh_sk_gen(p, y, sk, key)

Input: 法 mod：p, 鍵を共有する相手の公開鍵 the other person's public key：y, 秘密鍵：sk, 共有鍵を書き込むファイル名 (文字列, 拡張子は csv) Filename to output the shared key (string, extension is csv)：key

Output: 共有鍵 (整数) Shared key (integer)：$K_{a,b}$ (key.csv に出力) (output to key.csv)

参考関数 Related function: open, csv.writer

自作関数 Self-defined function: mod_binary

関連演習 Related exercise: 演習 3.3

Algorithm 32 ID へのパディング Padding ID: id_padding(K, ID, r)

Input: 共有鍵 Shared key：K, 生徒番号 Student ID：ID, 乱数 Random number：r

Output: パディングした ID Padded ID：$ID' = r \parallel 0 \ldots \parallel ID$

参考関数 Related function: bin, int, binascii.hexlify, encode, len

関連演習 Related exercise: 演習 3.3

Algorithm 33 DH 共有鍵と排他的論理和による暗号化 Encrypt DH shared key using XOR: dh_sk_enc(r, key, ID)

Input: 乱数 Random number：r, 平文 (生徒番号) Plaintext (student ID)：ID, 共有鍵ファイル名 (文字列, 拡張子は csv) Filename of the shared key (string, extension is csv)：key

Output: 暗号文 Ciphertext：$C = K \oplus ID'$ ($ID' = r \parallel 0 \ldots \parallel ID$)

参考関数 Related function: open, csv.reader

自作関数 Self-defined function： id_padding

関連演習 Related exercise: 演習 3.3

Algorithm 34 整数から文字列への変換 Convert integer to string: int_to_str(m)

Input: 平文 (整数) Plaintext (integer)：m

Output: message を変換した文字列 Converted string.

参考関数 Related function: binascii.unhexlify, decode

関連演習 Related exercise: 演習 3.2

Algorithm 35 文字列から整数への変換 Conver string to integer: str_to_int(m)

Input: 平文 (Unicode 文字列) Plaintext (Unicode string)：m

Output: message を変換した文字列 Converted integer.

参考関数 Related function: binascii.hexlify, encode, int

関連演習 Related exercise: 演習 3.2

F ライブラリ API

F.1 ライブラリ API 一覧

F.2 ライブラリ API 詳細

Algorithm 36 DH 共有鍵と排他的論理和による復号 Decrypt DH shared key using XOR: dh_sk_dec(key, C)

Input: ID (ID') の暗号文 Ciphertext of ID (ID')：C, 共有鍵ファイル名 (文字列, 拡張子は csv) Filename of the shared key (string, extension is csv)：key

Output: 暗号文 Ciphertext (Cipher：$ID' = K \oplus C$ の下位ビット the lower bits（ID のビット分）

参考関数 Related function: open, csv.reader, bin, len

関連演習 Related exercise: 演習 3.3

Algorithm 37 ランダム関数 random function: random_walk(g, h, p, ℓ, a, b, z)

Input: ベースポイント base point：g, ベースポイントの冪乗 power of base point：$h(= g^x) \pmod p$, 法 mod：p, ベースポイントの位数 base point number position：ℓ, z の g, h の指数 exponent：a, b ($z = g^a h^b$)

Output: a', b', $z' = g^{a'} h^{b'}$

参考関数 Related function： math.ceil, math.sqrt

関連演習 Related exercise: チーム対抗課題 1, 2, 3, 4, 5

Algorithm 38 離散対数の計算 Computing discrete logarithm calculate: calculate_dl(g, h, p, ℓ, a, b)

Input: ベースポイント base point：g, ベースポイントの冪乗 power of base point：$h(= g^x) \pmod p$, 法 mod：p, ベースポイントの位数 base point number position：ℓ, $g^a h^b \equiv 1 \pmod p$ を満たす a, b

Output: 離散対数 discrete log x ($h = g^x \pmod p$)

自作関数 Self-defined function: inv, ex_euclid, mod_binary

関連演習 Related exercise: 問 4.2, チーム対抗課題 1, 2, 3, 4, 5

Algorithm 39 ρ 法 rho method: rho(g, h, p, ℓ, a_0, b_0)

Input: ベースポイント base point：g, ベースポイントの冪乗 power of base point：$h(= g^x) \pmod p$, 法 mod：p, ベースポイントの位数 base point number position：ℓ, 初期値のパラメータ default parameters：a_0, b_0 (初期値 initial value：$z_0 = g^{a_0} h^{b_0}$)

Output: 離散対数 x ($h = g^x \pmod p$), ランダム関数の適用回数, 適用回数の期待値 discrete logarithm x ($h = g^x \pmod p$)), number of times a random function is applied, expected value of the number of times it is applied.

自作関数 Self-defined function: calculate_dl, random_work, mod_binary

参考関数 Related function： math.ceil, math.sqrt

関連演習 Related exercise: チーム対抗課題 1, 2, 3, 4, 5

Algorithm 40 Floyd の循環検出による ρ 法 Floyd circular detection ρ Method: rho_by_floyd(g, h, p, ℓ, a_0, b_0)

Input: ベースポイント base point：g, ベースポイントの冪乗 power of base point：$h(= g^x) \pmod p$, 法 mod：p, ベースポイントの位数 base point number position：ℓ, 初期値のパラメータ default parameters：a_0, b_0 (初期値 initial value：$x_0 = g^{a_0} h^{b_0}$)

Output: 離散対数 x ($h = g^x \pmod p$), ランダム関数の適用回数, 適用回数の期待値 discrete logarithm x ($h = g^x \pmod p$)), number of times a random function is applied, expected value of the number of times it is applied

自作関数 Self-defined function: calculate_dl, random_work, mod_binary

参考関数 Related function： math.ceil, math.sqrt

関連演習 Related exercise: チーム対抗課題 1, 2, 3, 4, 5

Algorithm 41 Distinguished point を用いた ρ 法 Distinguished point using ρ Method: rho_distinguished_point(g, h, p, ℓ, a_0, b_0, param)

Input: ベースポイント base point：g, ベースポイントの冪乗 power of base point：$h(= g^x) \pmod p$, 法 mod：p, ベースポイントの位数 base point number position：ℓ, 初期値のパラメータ default parameters：a_0, b_0 (初期値 initial value：$x_0 = g^{a_0} h^{b_0}$),
distinguished point に関するパラメータのリスト List of parameters relating to distinguished point：param (テキストの例だと Text example parame = [d])

Output: 離散対数 x ($h = g^x \pmod p$), ランダム関数の適用回数, 適用回数の期待値 discrete logarithm x ($h = g^x \pmod p$), number of times a random function is applied, expected value of the number of times it is applied

自作関数 Self-defined function: calculate_dl, random_work, mod_binary

参考関数 Related function： math.ceil, math.sqrt

関連演習 Related exercise: チーム対抗課題 1, 2, 3, 4, 5

Algorithm 42 DSA 署名-公開鍵生成関数 DSA Signature - public key generation function: dsa_sign_gen_key(p, g, ℓ, sk)

Input: 法 mod：p, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 秘密鍵 secret key：sk

Output: 公開鍵 public key: pk = g
sk mod$p$

自作関数 self-defined function:mod_binary

関連演習 Related exercise：演習 5.1, 6.1, 6.2, 6.3, 6.4

Algorithm 43 Shake 128 のハッシュ値 (整数値) の計算 shake256(m, ℓ)

Input: 文字列： m, ベースポイントの位数：ℓ

Output: shake128 による m のハッシュ値 (16 進文字列列)

関連演習: 演習 5.1, 6.1, 6.2

※この関数はこちらから提供します

Algorithm 44 Shake 256 のハッシュ値 (整数値) の計算 shake256(m, ℓ)

Input: 文字列： m, ベースポイントの位数：ℓ

Output: shake256 による m のハッシュ値 (16 進文字列列)

関連演習: 演習 6.3, 6.4

※この関数はこちらから提供します

Algorithm 45 DSA 署名-署名関数 DSA Signatures - Signature functions dsa_sign_gen(m, g, ℓ, r, p, sk)

Input: メッセージ (通常の文字列) Message (string)：m, 秘密鍵 secret key：sk, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 乱数 (整数) random number (integer)：r, 法 mod：p

Output: 署名 (整数の配列) signature (list of integer)：[u, v]

参考関数 Related function：hashlib.shake_128

自作関数 self-defined function:mod_binary, inv, str_to_int

関連演習 Related exercise：演習 5.1, 6.1, 6.2, 6.3, 6.4

Algorithm 46 DSA 署名-署名検証関数 DSA signatures - signature verification functions: dsa_sign_verify(m, sigma, ℓ, g, p, pk)

Input: 署名 signature：sigma=[u, v], メッセージ (通常の文字列) Message (string)：m, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 公開鍵 public key：pk, 法 mod：p

Output: 署名が正しければ True, 誤っていれば False If signatures are correct True, if wrong False.

参考関数 Related function：hashlib.shake_128

自作関数 self-defined function:mod_binary, inv, str_to_int

関連演習 Related exercise：演習 5.1, 6.1, 6.2, 6.3, 6.4

Algorithm 47 多重署名-公開鍵生成関数 multiple signatures - public key generation functions: multi_sig_key_gen(p, g, l, sk)

Input: 法 mod：p, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 秘密鍵 secret key：sk

Output: 多重署名の公開鍵 Public keys for multiple signatures

自作関数 self-defined function:

関連演習 Related exercise： 解析 8

Algorithm 48 多重署名-署名関数 multiple signatures - signature functions: multi_sig_gen_sign(m, sk_list, g, ℓ, p)

Input: メッセージ (通常の文字列) message (string)：m, 秘密鍵のリスト list of secret key：sk_list, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 法 mod：p

Output: 多重署名 multiple signatures

参考関数 Related function：

自作関数 self-defined function:

関連演習 Related exercise： 解析 8

Algorithm 49 多重署名-署名検証関数 multiple signatures - signature verification functions: multi_sig_verify_sign(m, sigma, ℓ, g, pk_list, p)

Input: 署名 signature：sigma, メッセージ (通常の文字列) message (string)：m, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 公開鍵のリスト list of public key：pk_list, 法 mod：p

Output: 署名が正しければ True, 誤っていれば False If signatures are correct True, if wrong False

参考関数 Related function：

自作関数 self-defined function:

関連演習 Related exercise： 解析 8

Algorithm 50 ElGamal-DSA ハイブリッド-公開鍵生成関数 ElGamal-DSA Hybrid - Public key generation function: hybrid_key_gen(p, g, sk)

Input: 法 mod：p, ベースポイント base point：g, 秘密鍵 secret key：sk

Output: 公開鍵 public key: pk = g^{sk} mod p

自作関数 self-defined function:mod_binary

関連演習 Related exercise： 演習 6.1, 6.2, 6.3, 6.4

Algorithm 51 ElGamal-DSA ハイブリッド-暗号化・署名関数 ElGamal-DSA Hybrid - Encryption and signature functions: hybrid_enc_sign_gen(m, g, ℓ, r, p, sk, pk)

Input: メッセージ (通常の文字列) message (string)：m, 秘密鍵 secret key：sk, 公開鍵 public key：pk, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 乱数 (整数) random number (integer)：r, 法 mod：p

Output: m の暗号文 (整数の配列) m cyphertext(integer array)：$[c_1 = g^r, c_2 = \mathrm{pk}^r m]$, 署名 (整数の配列) signature(integer array)：$[u, v]$

参考関数 Related function：hashlib.shake_256

自作関数 self-defined function:dsa_sign_gen, elgamal_enc

関連演習 Related exercise： 演習 6.1, 6.2, 6.3, 6.4, 解析 9, 10

Algorithm 52 ElGamal-DSA ハイブリッド-復号・署名検証関数 ElGamal-DSA Hybrid - decryption and signature verification functions: hybrid_enc_sign_verify(C, sigma, sk, ℓ, g, p, pk)

Input: m の暗号文 ciphertext of m：$C = [c_1, c_2]$, m の署名 signature of m：sigma=$[u, v]$, ベースポイント base point：g, ベースポイントの位数 base point number position：ℓ, 公開鍵 public key：pk, 秘密鍵 secret key：sk, 法 mod：p

Output: C の復号結果 m, 署名が正しければ True, 誤っていれば False C decryption of m, If signatures are correct True, if wrong False.

参考関数 Related function：hashlib.shake_256

自作関数 self-defined function:elgamal_dec, dsa_sign_verify

関連演習 Related exercise： 演習 6.1, 6.2, 6.3, 6.4, 解析 9, 10

表 5: ライブラリ API 一覧

パッケージ	関数	引数	内容
標準	pow(x, y, z)	int 型 x, y, z	累乗計算
標準	divmod(x, y)	int 型 x, y	整数の商と余り $(x//y, x\%y)$ を返す
標準	a.bit_length()	int 型 a	n の 2 進数表現のビット数を返す
標準	int.from_bytes(s, order)	バイト型 s, 文字列型 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)	int 型 n, 文字列型 order	整数を表すバイト列を返す.
標準	s.encode()	文字列型 (Unicode) s	Unicode 文字列をバイト列に変換する
標準	b.decode()	バイト列型 (ASCII) b	バイト列を Unicode 文字列に変換する
標準	zip(L1, ..., Lk)	リスト L1, ..., Lk	複数のリストを一つにまとめる.
標準	bin(k)	int 型 k	整数を先頭に 0b がついた 2 進文字列に変換
標準	L.sort()	リスト L (要素は float 型または int 型)	L の要素を昇順に並べる (破壊的処理).
標準	L.sort(reverse=True)	リスト L (要素は float 型または int 型)	L の要素を昇順に並べる (破壊的処理).
標準	sorted(L)	リスト L (要素は float 型または int 型)	L の要素を昇順に並べる (非破壊的処理).
標準	sorted(L, reverse=True)	リスト L (要素は float 型または int 型)	L の要素を降順に並べる (非破壊的処理).
binascii	binascii.hexlify(s)	バイト列型 (ASCII) 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)	バイト列型 (16 進数) 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_256(m)	バイト列型 (ASCII) m	バイト列 m を入力とする SHAKE256 のオブジェクトを返す
hashlib	hash.shake_256(m).hexdigest(n)	バイト列型 (ASCII) m, int 型 n	バイト列 m のハッシュ値 (2n バイト) を返す.
os	os.urandom(n)	int 型 n	長さ n のランダムなバイト列を生成する (n バイト未満になることもある).
random	random.randrange(a, b)	int 型 a, b	[a, b) に含まれる乱数を返す.
random	random.seed(a, version)	hashable な任意オブジェクト a (int 型など)	乱数関数を初期化する. デフォルトは a=None, version=2
secrets	secrets.choice(L)	リスト L	L からランダムに要素を取り出す.
secrets	secrets.randbelow(n)	int 型 n¿0	0 以上　n 未満の 整数をランダムに選ぶ
secrets	secrets.randbits(k)	int 型 k¿0	k ビットのランダムな整数を返す (k ビット未満になることもある).
sympy	sympy.factorint(n)	int 型 n	n の素因数を返す.
time	time.perf_counter()	なし	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(a)	float 型 a	入力 (浮動小数点) の「天井」 (入力以上の最小の整数) を返す.
math	math.sqrt(n)	float 型 n	$\sqrt{n}$ を返す.
numpy	numpy.mean(L)	リスト L (要素は int 型 または float 型)	入力データの算術平均を返す.
numpy	numpy.var(L)	リスト L (要素は int 型 または float 型)	入力データの分散を返す.
collections	collections.Counter(L)	リスト L	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)	int 型 n, k	順列 $n\mathrm{P}k$ の計算
scipy.special	scipy.special.comb(n, k)	int 型	組み合わせの数 $n\mathrm{C}k$

Algorithm 53 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 54 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

Algorithm 55 ビット長出力 a.bit_length()

Input: 整数 a (int 型)

Output: 整数 a のビット数 (int 型)

Object: int

Algorithm 56 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 57 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 58 Unicode 文字列のエンコード s.encode()

Input: 入力文字列 (Unicode 型) s

Output: バイト列 (bytes 型)

Package: 標準

Algorithm 59 バイト列のデコード b.decode()

Input: 入力バイト列 (byte 型) b

Output: 文字列 (Unicode 型)

Package: 標準

Algorithm 60 複数のリストをまとめる zip()

Input: リスト (1 つ以上)

Output: 入力のリストを 1 つにまとめた zip オブジェクト

Package: 標準

Algorithm 61 整数の 2 進文字列への変換 bin(k)

Input: 整数 k

Output: 入力の 2 進文字列 (先頭に 0b がついている)

Package: 標準

Algorithm 62 リストの要素を昇順に並べる L.sort()

Input: リスト L (要素は float 型または int 型)

Output: L の要素を昇順に並べたリスト

Package: 標準 **Remark:** リスト L は変更される（破壊的処理）. L.sort(reverse=True) にすると降順に要素が並び替えられる.

Algorithm 63 リストの要素を昇順に並べる sorted(L)

Input: リスト L (要素は float 型または int 型)

Output: L の要素を昇順に並べたリスト

Package: 標準 **Remark:** リスト L は変更されない（非破壊的処理）. sorted(L, reverse=True) にすると降順に要素が並び替えられる.

Algorithm 64 ASCII バイト列を 16 進数バイト列へ変換 binascii.hexlify(s)

Input: ASCII バイト列 (bytes 型) s

Output: 16 進数表現バイト列 (bytes 型)

Package: binascii

Algorithm 65 16 進数バイト列を ASCII バイト列へ変換 binascii.unhexlify(s)

Input: 16 進数表現バイト列 (bytes 型) s

Output: ASCII バイト列 (bytes 型)

Package: binascii

Algorithm 66 Shake256 によるハッシュ値 hashlib.shake_256(m).hexdigest(n)

Input: バイト列 (bytes 型) m, 正の整数 n (int 型)

Output: m のハッシュ値 (bytes 型, 16 進数表示) (2n バイト)

Package: hashlib

Algorithm 67 ハッシュアルゴリズム SHAKE256 のオブジェクト生成 hashlib.shake_256(m)

Input: バイト列 (bytes 型) m

Output: SHAKE256 アルゴリズムを扱うオブジェクト (hash 型)

Package: hashlib

Algorithm 68 ランダムなバイト列を生成 os.urandom(n)

Input: バイト数 n (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．n ビット未満になる場合もある

Algorithm 69 k ビットの乱数を生成 random.getrandbits(k)

Input: k: 出力乱数のビット数 (int 型)

Output: 乱数 (int 型)

Package: random

Remark: random.seed で初期化される．k ビット未満になる場合もある

Algorithm 70 [a, b) の範囲内にある乱数を生成 random.randrange(a, b)

Input: 乱数の生成範囲 [a, b) (int 型)

Output: 乱数 (int 型)

Package: random

Remark: random.seed で初期化される.

Algorithm 71 乱数関数の初期化 random.seed(a=None, version=2)

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

Package: random

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

Algorithm 72 リストからランダムに要素を取り出す secrets.choice(L)

Input: リスト L

Output: L のランダムにとられた要素

Package: secrets

Algorithm 73 0 以上 n 未満の 整数をランダムに選ぶ secrets.randbelow(n)

Input: 自然数 n

Output: 0 以上 n 未満の ランダムな整数

Package: secrets

Algorithm 74 k ビットの 整数をランダムに生成 secrets.randbits(k)

Input: 自然数 k

Output: k ビットの ランダムな整数

Package: secrets **Remark:** k ビット未満になる場合もある

Algorithm 75 入力 (整数) の素因数分解を行う．sympy.factorint(n)

Input: 整数 n (int 型)

Output: n の素因数分解

Package: sympy

Algorithm 76 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 77 浮動小数の天井 math.ceil(a)

Input: 浮動小数 a (float 型)

Output: 入力の「天井」 (入力以上の最小の整数). (int 型)

Package: math

Algorithm 78 算術平均 numpy.mean()

Input: データのリスト (リストの要素は float 型)

Output: データの平均 (float 型)

Package: numpy

Algorithm 79 標本分散 numpy.var()

Input: データのリスト (リストの要素は float 型)

Output: データの分散 (float 型)

Package: numpy

Algorithm 80 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 (int 型)

Package: collections

Algorithm 81 順列の計算 scipy.special.perm(n, k)

Input: 整数 n, k ($n \geq k$) (int 型)

Output: 順列 nPk (int 型)

Package: scipy.special

Algorithm 82 組み合わせの数の計算 scipy.special.comb(n, k)

Input: 整数 n, k ($n \geq k$) int(型)

Output: 順列 nCk (int 型)

Package: scipy.special

Algorithm 83 相関係数の計算 numpy.corrcoef(x)

Input: 要素数の等しい数値のリスト組をリストにまとめたもの // 例：x = [[2, 5, 4, 2, 1], [1, 3, 5, 3, 2]]

Output: 入力のリスト同士の相関係数

Package: numpy

D introduction

E English Translation

E.1 Introduction to Number Theory and Necessary Algorithms Underlying Cryptography I (Textbook 2, Chapter 3)

One of the objectives of this lecture is to understand the basic operations of finite numbers, namely multiplication and its inverse operation, division. In particular, the inverse operation is an important component of cryptography. The side-channel attack, a type of cryptanalysis, takes advantage of the fact that the computational complexity changes with each input. Using inverse operations, we will study the case where the computational complexity varies from input to input. In cryptography, n bits means n sequences of 0s and 1s. In other words, $0 = \underbrace{0 \cdots 0}_{160}$ is also 160 bits. A secure cipher means that for any n bits of secret information, the time required to decrypt the secret information is equivalent to an entire n bit search.

E.1.1 Groups, rings, and fields

The set G, with a binary operation, e.g. multiplication, $\cdot$, being a **group** means that the following four conditions are satisfied.

(G0) Any binary product of $a, b \in G$, $a \cdot b \in G$, is defined.

(G1) (Associative law) For all $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

(G2) (Unit element) There exists a unique $1 \in G$, the **unit** or **identity** element, such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$.

(G3) (Inverse element) For any $a \in G$, there exists an **inverse** element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$.

The above four conditions are called **group axioms**. If in a group G,

(G4) (Commutative law) $a \cdot b = b \cdot a$,

is satisfied, G is called a **commutative** group or **Abelian** group. We usually write $a \cdot b = ab$, when convenient.

The number of elements of in a group G, $\#G$, is called the **order of** G. Also, when taking an element $a \in G$, let $\langle a \rangle$ be the entire set of powers of a, i.e.

$$\langle a \rangle = \{a^k \mid k \in \mathbb{Z}\}.$$

$\langle a \rangle$ is clearly a **subgroup** of G (a group where all elements lie inside of G), called the **subgroup of** G **generated by** a.

The order of the subgroup $\langle a \rangle$ is equal to the order of a. The **order of** a is equal to the smallest positive integer k such that $a^k = 1$ (or possibly, infinity).

A set R with two binary operations, e.g. addition, $+$, and multiplication, $\cdot$, is a **ring** if R satisfies the following three conditions.

(R1) It is a commutative group with respect to the addition of R (this is called a **module** and has unit element 0).

(R2) (Associative multiplication law) $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

(R3) (Distributive law)

$$\begin{cases} a(b+c) &= ab + ac \\ (a+b)c &= ac + bc. \end{cases}$$

(R4) (Unit element) There is an element $1 \in R$ which is different from 0 in R and which satisfies $1 \cdot x = x \cdot 1 = x$ for any element x of R.

If in addition to the above properties,

(R5) (Commutative multiplication law) $ab = ba$

is satisfied, R is called a **commutative ring**.

Example 1 *The set of integers, $\mathbb{Z}$, with regular + and ·, is a commutative ring. Note that there is no specification of multiplicative inverses in the definition of a ring.*

In a commutative ring R, when every element other than zero element 0 is a invertible element, R is said to be a **field**.

The set of rational numbers $\mathbb{Q}$, real numbers $\mathbb{R}$, and the set of complex numbers $\mathbb{C}$, are all examples of fields.

E.1.2 Arithmetic of finite field

Fix a positive integer m. When the difference between the integers a and b, $a - b$, is a multiple of m, a and b are said to be **congruent** with respect to the **modulus** m. We write,

$$a \equiv b \pmod{m}$$

to denote this **congruence**. Let r be the remainder when dividing an arbitrary integer a by m , then $0 \leq r \leq m - 1$, each of which definies a **congruence class**. There are m residue classes modulo m.

Example 2 *The following three sets are the congruence classes module* 3.

$$\overline{0} = 3\mathbb{Z} = \{\ldots, -3, 0, 3, \ldots\}$$
$$\overline{1} = 1 + 3\mathbb{Z} = \{\ldots, -2, 1, 4, \ldots\}$$
$$\overline{2} = 2 + 3\mathbb{Z} = \{\ldots, -1, 2, 5, \ldots\}$$

We express the set of representative elements selected from each residue class as

$$\mathbb{Z}/m\mathbb{Z} = \{0, 1, \ldots, m-1\}.$$

At this time, $\mathbb{Z}/m\mathbb{Z}$ becomes a ring by the following operations. For $a, b \in \mathbb{Z}/m\mathbb{Z}$, define

$$a \cdot b \equiv ab \pmod{m}, a + b \equiv a + b \pmod{m}.$$

In this way, the unit element for multiplication is $1 + m\mathbb{Z}$, and the zero element for addition is $m\mathbb{Z}$.

Example 3 *Sums and products in $\mathbb{Z}/3\mathbb{Z}$.*

+	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

Example 4 *Sums and products in $\mathbb{Z}/4\mathbb{Z}$.*

+	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

Problem 1 *Verify the tables from Examples 3 and 4.*

Problem 2 *Find the inverse of* 2 *in* $\mathbb{Z}/3\mathbb{Z}$.

Problem 3 *Create a sum and product table for* $\mathbb{Z}/13\mathbb{Z}$.

E.1.3 Division over a finite field

When the integer c is a divisor of the integer a and is also a divisor of the integer b , c is called a **common divisor** of a and b. Because both a and b are finite, the common divisor c is also finite. Among them, the largest positive common divisor d is called the **greatest common divisor**. In the symbols, $d = \gcd(a, b)$, where gcd stands for the greatest common divisor. In particular, when $\gcd(a, b) = 1$, a and b are said to be **coprime**. For the greatest common divisor $d = gcd(a, b)$ of the two integers a, b, the following holds.

Lemma 1

$$\gcd(a, b) = \gcd(a - b, b) \tag{8}$$

$$\gcd(a, b) = \gcd(r, b) \ (a = bq + r \ for \ b > r \geq 0 \ and \ q \in \mathbb{Z}) \tag{9}$$

The greatest common divisor d of two integers a and b can be obtained using the Euclidean algorithm.
[Euclidean algorithm] We define the **Euclidean algorithm (ECD)** as:

Input: Positive integers a and b, $a \geq b$

Output: $\gcd(a, b) = d$

(1) Let $a_0 = a$, $a_1 = b$ and $i = 1$.

(2) If $a_i = 0$, output $d = a_{i-1}$ and exit.

(3) If $a_i \neq 0$, let $a_{i-1} = a_i q_i + a_{i+1}$, $0 \leq a_{i+1} < |a_i|$, $i = i + 1$, and go to (2).

Problem 4 *Find the the greatest common divisor of the following numbers.*

(1) $\gcd(1234567, 234578)$

(2) $\gcd(11111111, 33332222)$

The following lemma is derived by extending the Euclidean algorithm.

Lemma 2 *Let d be the greatest common divisor of integers a and b, then there exist integers x and y satisfying*

$$ax + by = d.$$

[Extended Euclidean algorithm] We define the **Extended Euclidean algorithm** as:

Input: Positive integers a and b, $a > b$

Output: $\gcd(a, b) = d$ and integers x and y such that $ax + by = d$

(1) Let $a_0 = a$, $a_1 = b$ and $i = 1$.

(2) Let $x_0 = 1$ and $x_1 = 0$.

(3) Let $y_0 = 0$ and $y_1 = 1$.

(4) Assume i=1.

(5) If $a_i = 0$, then $d = a_{i-1}$, $x = x_{i-1}$, and $y = y_{i-1}$. Output and exit.

(6) Using the relation $a_{i-1} = a_i q_i + a_{i+1}$, $0 \le a_{i+1} < a_i$, define a_{i+1} and q_i.

(7) Let $x_{i+1} = x_{i-1} - q_i x_i$.

(8) Let $y_{i+1} = y_{i-1} - q_i y_i$.

(9) Define $i = i + 1$ and go to (5).

Problem 5 *For the following values, find integers x and y such that* $ax + by = 1$.

(1) $a = 23, b = 17$

(2) $a = 13, b = 8$

The inverse of an element in a finite field can be obtained using the extended Euclidean algorithm described above.

Lemma 3 *When* $\gcd(a, m) = 1$*, there exists an integer x that satisfies the following congruence equation.*

$$ax \equiv 1 \pmod{m}.$$

Problem 6 *Implement the Extended Euclidean algorithm. Solve the congruence* $3X \equiv 1 \pmod{20}$*, i.e. find the inverse of* 3 *in* $\mathbb{Z}/20\mathbb{Z}$.

Exercise 1 (Inverse element in a finite field using extended ECD) *Apply the extended Euclidean algorithm and find the inverse element in* $\mathbb{F}_p$ *as asked below.*

(1) Using p_1 *and* g_1 *in Table 1 , find* $g_1^{-1} \pmod{p_1}$.

(2) Using p_1 *and* g_2 *in Table 1 , find* $g_2^{-1} \pmod{p_1}$.

(3) Using p_1 *and* g_2 *in Table 1 , find* $g_2^{-1} \pmod{p_1}$.

(4) Using p_1 *and* g_4 *in Table 1 , find* $g_4^{-1} \pmod{p_1}$.

The greatest common divisor d of two integers a, b is Using the formula (8) in the Lemma 1. The greatest common divisor can be obtained without finding a remainder as follows.

[Euclidean algorithm 2]

Input: Two positive numbers a, b where $a \ge b$.

Output: $\gcd(a, b) = d$.

(1) Let $a_0 = a$, $a_1 = b$, $i = 1$.

(2) If $a_i = 0$, output $d = a_{i-1}$ and finish.

(3) When $a_i \neq 0$, set $a_{i+1} = |a_i - a_{i-1}|$ and $a_i = \min(a_i, a_{i+1})$. Set $i = i + 1$ and go to (2).

Furthermore, when implementing the inverse element in hardware, dividing by 2 can be realized by shifting. Therefore, if the factor part of 2 of the greatest common divisor is treated separately, it can be further optimized, achieving compactness and high speed.

[Binary Euclid's reciprocal division method]

Input: 2 Positive numbers a, $b(a \geq b)$

Output: $\gcd(a,b) = d$

(1) [Initial setup] Assume $a_0 = a$, $a_1 = b$,$k = 0$, $i = 1$.

(2) If $a_i = 0$, output $d = a_{i-1}2^k$ and exit.

(3) If $a_i = 0 \pmod 2$ and $a_{i-1} = 0 \pmod 2$, when $a_i \neq 0 \pmod 2$ or $a_{i-1} \neq 0 \pmod 2$. The following is repeated. $a_i = a_i/2$, $a_{i-1} = a_{i-1}/2$, $k = k+1$. Go to (4)

(4) If $a_i = 0$, output $d = a_{i-1}2^k$ and exit.

(5)

$$\begin{cases} a_i = a_i/2^j (\text{ divide until } a_i \text{ is an odd number}^{9}) & (\text{if } a_i = 0 \pmod 2) \\ a_{i-1} = a_{i-1}/2^j \text{ (divide until } a_{i-1} \text{ is odd)} & (\text{if } a_{i-1} = 0 \pmod 2) \end{cases}$$

Go to (6).

(6)[10] $a_{i+1} = |a_i - a_{i-1}|$, $a_{i+1} = a_{i+1}/2$

$a_{i+1} = \min(a_i, a_{i+1})$[11] $a_i = \max(a_i, a_{i+1})$, Set $i = i+1$ and go to (4).

Exercise 2 *Implement the extended binary Euclidean algorithm. Also, check the value with exercise 1.*

[Inverse element in a finite field using extended ECD] Compare the computational complexity of extended Euclid's reciprocal division and binary extended Euclid's reciprocal division. The computational complexity is compared using number of steps × per loop. The first step is to experimentally predict the characteristics of the input for each maximum number of steps. It is important to visualize the results in a graph. For a small example, save the data in a CSV file, extract the features in a graph, and predict the size of the data to be used in practice.

1. Set $p = 2^{12} - 77$. Find a^{-1} for $\mathbb{F}_p \ni \forall a$. Find the number of steps (the number of times the loop is executed) by counting. Visualize the number of steps for each value. Visualize the number of steps for each value by plotting it on a graph, etc. Discuss which value is the maximum number of steps for each of the binary augmented Euclidean reciprocal division and Discuss. Find the maximum and minimum number of steps, the mean and variance.

2. Do the same as (1) with $p = 2^{11} + 5$.

3. $p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$ is a prime number[12]. Find the expected maximum and average number of steps from (1) and (2), respectively. Also, describe the characteristics of the maximum number of steps and the expected number of steps. (For the average number of steps, state how you obtained it.)

4. Find the average computational complexity of one loop of the extended Euclidean reciprocity and the binary extended Euclidean reciprocity, and find the worst-case and average computational complexity. (Worst computational complexity = expected maximum number of steps × computation time of one loop, Average computational complexity = average number of steps × computation time of one loop)

[10] Since the result of subtracting an odd number is always even, divide by 2. In practice, this is accomplished with a 1-bit right shift.
[11] It should be possible to realize large and small comparisons quickly.
[12] It is a prime of the standardly used elliptic curve specified by NIST

E.1.4 Power operation

The main part of the running time of the ElGamal cipher, which will be introduced later, is the time required to perform the power operations. In this section, we introduce the most basic algorithm for computing power operations, the binary method, which is the most basic algorithm for computing power operations. In cryptography, a 160-bit number includes 0 as 160 sequences of 0s. For this reason, it is important that the binary method works correctly even when the exponent is zero. [Binary Method 1] Here we introduce the **Binary algorithm**.

Input: positive integer $k = \sum_{i=0}^{n-1} k_i 2^i$ for $k_i \in \{0, 1\}$ and $k_{n-1} \neq 0$, and generator, g.

Output: $y = g^k$

`Binary1`(k, g)

(1) Set $y = g$.

(2) `for` $i = n-2, \ldots, 0$, `do`:
`if` $k_i = 1$, `then` $y = y^2 * g$,
`else` $y = y^2$.
Next i.

(3) Output y.

Find a^b for the following numbers.
(1) $a = 23,\ b = 17$
(2) $a = 13,\ b = 8$
(3) $a = 5,\ b = 31$
(4) $a = 7,\ b = 41$ When using the binary method in ElGamal ciphers, etc., the size of g and k are 1024 bits and 160 bits or more, respectively, so the bit length of the variable y becomes too large to work in practical use if the algorithm 5 is used. Algorithm E.1.4 takes the definite $\mathbb{F}_p$ as input and considers the bit length of y to be smaller than the 3-fold length of the definite. In this case, since the computation time of the modulo operation is also required, it is important to minimize the number of modulo operations as much as possible.

[Modulo p on Binary Method 1] `ModBinary1(`g, k,p`)`
`Input:` `positive integer` $k = \sum_{i=0}^{n-1} k_i 2^i$ `for` $k_i \in \{0, 1\}$ `and` $k_{n-1} \neq 0$, `generator` g, `and prime` p.
`Output:` $y = g^k \bmod p$
`ModBinary1(`k, g, p`)`
`1.` $y = 1$.
`2.` `for` $i = n-1, \cdots 0$, `do`
`if` $k_i = 1$, `then` $y =$ `Mod(Mod(`y^2, p`)`$\cdot g$, p`).`
`else` $y =$ `Mod(`y^2, p`)`
`next` i
`3.` `Output` y

Exercise 3 *(1) Calculate* $2^{1234567890} \pmod{97}$.
(2) Calculate $2^0 \pmod{97}$[13].

The accuracy of random number generation has a significant impact on cryptographic security. Python provides several random number generation functions. The accuracy of the random numbers output by each function can

[13]This assignment is to verify that the zero power can be computed correctly.

be used to determine the binary method, and check if the number of multiplications is approximately half of a bit string. Find out how many multiplications and squaring operations are required by the binary method, measure the accuracy of random number generation. Specifically `secrets.randbelow(`n`)`, `random.randint(`a, b`)` etc. Generate 10^3 random numbers k, find the number of multiplications and squares for each k, and compare the average value and the number of squares for each k, and find their mean and variance. The random number generating function to be considered is:
(1) When 64-bit random $k \in \{0,1\}^{64}$ is generated, find the average number of how many multiplications and squaring operations are required on average.
(2) When generating a 128-bit random $k \in \{0,1\}^{128}$, find how many multiplications and squaring operations are required on average.
(3) When generating a 160-bit random $k \in \{0,1\}^{160}$, find how many multiplications and squaring operations are required on average.

Introduction

The predecessor of this PBL exercise, Summer School, held the first in 2000 and the 16th in 2016. After that, from 2017, it changed to PBL exercise that accepted graduate students nationwide, and from 2018, it developed into a highly diverse PBL that accepted working adults, graduate students, and undergraduate students. This year is the $22 = 2 * 11$ time. The total number of participants so far is 349. This exercise introduces new tasks every year, and even if you take it every year, you can learn new things. We hope that every participator this year will also experience the fun of mathematics and security. By introducing flipping this year and solving pre-tasks, we believe that we can carry out more understandable exercises. The summer school is an open course independently conducted by Miyaji Lab for the following purposes.

- Display the importance of security by introducing some of the information security research and technologies to the general public.
- Develop human resource of engineers and researchers involved in research and development in the field of information security.
- Convey that "math is an available science (technology)" by educating how mathematics is applied to security.
- Inform the general public about the research contents of our laboratory and improve the level of laboratory members.

We will continue to hold it as a PBL, so please comment and support us.

The purpose of this course is to learn the theory of mathematics and cryptography through exercises. Therefore, after explaining the theory, we will perform the exercises described in the text. But the text contains a little more exercises, the essential items are listed for each exercise. If you finish the provided exercises, you can solve the others.

- Problem: Deepen your understanding of the lecture. Submit the answer in a notebook.
- Implementation: Understanding the gap between theory and the realization through implementation. Apply Python as well as the functions installed in Python as standard. Follow the API described in the E chapter for the function API to be implemented. The functions used in the standard built-in functions are also described in the F chapter. The created function contains sample data to verify the correctness of the program implemented in the F chapter. After programming, please check the correctness of the program with the sample data. Measure the calculation speed in the specified number of times (10^4, etc.) and calculate the average value.

- Analysis: We will consider from experimental results and theoretical values when performing statistical processing. Experimental data is created electronically using excel etc., and experimental results are analyzed by oneself. An analysis report should include the experimental results. Please save these in one directory, zip them up and submit. The analysis also includes the task of method proposal. Please use word, tex, etc. to create a method proposal.

  To submit the python source, please submit the jupyter notebook file. The file name should be as follows. The analysis submissions are the Python source file and the execution result file requested in the exercise. Save the source file and the execution result file in one directory, compress them with zip, and submit them with the following file names. In addition, please add the acronyms P, I, and A to the alphabet of the exercise number of the file name of the submissions of "Question", "Exercise", and "Analysis" (program result).

  First letter of the name of University_name_exercise number (大学名頭文字：JAIST → J, ProSec → P, 阪大経済学研究科 → OM, 阪大工学部通信コース/工学研究科 → OE, 阪大工学部情シスコース/情報科学研究科 → OI, 岡山大 → Y, 大分大 → IT，長崎県立大 → NK，石川高専 → IK,)
  Example: When Miyaji Lab, the Faculty of Engineering, Osaka University submits the first question: OE_miyaji_P1

In addition, the PBL are divided into three courses, advanced, standard, and beginner, from the level. All lectures will be given, but exercises will be conducted according to each student's skills and preferences.

advanced course exercise 1.1, exercise 1.2, analysis $1^{\dagger}$, analysis E.1.4, exercise3 → exercise 2.1, analysis **??**†, exercise3.3†, analysis 7 → exercise 3.1†, analysis $6^{\dagger}$ → analysis 5, Q2.2† → Q4.1 → Q 5.1, exercise 5.1†, analysis 8 → exercise **??**, exercise **??**, exercise **??**, exercise**??** → team competition

Standard course exercise 1.1, exercise3,analysis E.1.4, → exercise3.3†, analysis 7 → exercise 3.1†, analysis 6 → Q2.2† → Q4.1 → Q 5.1, Q5.2, exercise 5.1†, analysis $8^{\dagger}$ → exercise **??**, exercise **??** → (team competition)

beginner course exercise 1.1, exercise3 → analysis 7 → exercise 3.1†, Q2.2† → Q4.1 → Q 5.1, exercise 5.1† → exercise **??** → (team competition)

Group presentation (analysis 1, analysis E.1.4, analysis 7, analysis 6, analysis 8) from 2 + competition

E.2 Computation Complexity (Textbook Chapter 3, 7.8)

The purpose of this lecture is to understand the theoretical evaluation of computational complexity. Cryptography is performed by operations on the law p. Theoretical evaluation of these computational quantities leads to the evaluation of cryptography. In cryptography 128-bit multiplication, 1024-bit multiplication, squaring, and inverse, these are important to theoretically evaluate the difference in bits and operations. The following are commonly used for evaluation.

- Evaluate the unit of computational complexity of a 160 bit multiplication as M_{160}.
- 160 bits: 1024 bits = 1 : 6
- The computational complexity of n-bit multiplication : The computational complexity of mn-bit multiplication = $1 : m^2$
- Ignore the computational complexity of addition and subtraction
- Multiplication complexity M : squared complexity $S = 1 : 0.8$. Multiplication complexity M : Inverse computational complexity $I = 1 : 11$,

E.2.1 The Application of Fermat's Method

Exercise 1.1 confirms that the execution speed of the calculation of an inverse element using the extended Euclidean algorithm varies depending on the element. In this section, we will consider how to implement the inverse algorithm using Fermart's little theorem and discuss its execution speed.

-Fermat's little theorem-

For some prime number p and a natural number a which is coprime to p, we have $a^{p-1} \equiv 1 \pmod{p}$. This is **Fermat's little theorem**. One of the applications of Fermat's little theorem is the pseudo-prime number determination test. Let n be a given positive integer. For $1 \leq a \leq n-1$ and $\gcd(a, n) = 1$, check if $a^{n-1} \equiv 1 \pmod{n}$ is satisfied for a. The prime number determination is performed probabilistically by repeating the test to determine whether $a^{n-1} \equiv 1 \pmod{n}$ is satisfied. This is called the **Fermat method**. The Fermat method always determines that a prime number is a prime number. Composite numbers may also be probabilistically determined to be prime. This method of probabilistically determining that a number is prime is called called the probabilistic prime number determination method.

From Fermat's little theorem, we get

$$\begin{aligned} a^{p-1} &\equiv 1 \pmod{p} \\ a^{-1} &\equiv a^{p-2} \pmod{p} \end{aligned} \tag{10}$$

Using equation 10, we can construct an algorithm that calculates the inverse element. [Inverse element 1]

`Inverse2(`p`, `g`)`
`Input : prime number `p` and `g
`Output :` $y = g^{p-2} \pmod{p}$
`Inverse(`p`, `g`)`
`1.` $y = g$`.`
`2.` $y =$ `ModBinary1(`$p-2$`, `g`, `p`)`
`3. Output `y

Exercise 4 (Inverse element in a finite field using Fermat's little theorem) *Apply Fermat's little theorem to find the following inverse elements on* $\mathbb{F}_p$.

1. *Find* $g_1^{-1} \pmod{p_1}$ *using* p_1 *and* g_1 *in Table 1.*
2. *Find* $g_2^{-1} \pmod{p_1}$ *using* p_1 *and* g_2 *in Table 1.*
3. *Find* $g_3^{-1} \pmod{p_1}$ *using* p_1 *and* g_3 *in Table 1.*
4. *Find* $g_4^{-1} \pmod{p_1}$ *using* p_1 *and* g_4 *in Table 1.*
5. *Compare the above four computations by the number of 1024-bit multiplications performed.*
6. *Count the number of steps in the above four inversions (the number of loops in ModBinary*1*).* (*Note that estimating by number of steps is a pretty crude estimate.*)

[Inverse element in a finite field using extended ECD] We experiment how the number of multiplications of the inverse element operation that applies the Fermat's small theorem changes depending on the input.

1. Let $p = 2^{12} - 77$. $\forall a \in \mathbb{F}_p$, find a^{-1} while counting the number of 12-bit multiplications. Find the maximum and minimum number of multiplications, the average value, and the variance.
2. Do the same as (1) with $p = 2^{11} + 5$.

3. Consider the prime $p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. Using (1) and (2), predict an a_{max}, a_{min}, for which finding the inverse requires the maximum, respectively minimum, number of multiplications, and actually find the number of multiplications.

4. Compare with Analysis 1.

Exercise 5 *Using the binary method, if there is a square root of $a \in \mathbb{F}_p^*$ for a prime p congruent with 3 in method 4, create an algorithm to find it.*

E.2.2 Accuracy of Random Numbers

-The Birthday Paradox-

The Birthday Paradox is the question, "How many people must get together to have the probability of having at least two people with the same birthday exceed 50%?". The solution to this question is unexpectedly small. Let's check it out.
The probability of no one has the same birthday among n people is $p(365, n) = \frac{364}{365} \cdot \frac{363}{365} \cdot \frac{362}{365} \cdots \frac{365-n+1}{365} = \frac{365!}{365^n(365-n)!}$.
Therefore, the probability of having at least two people with the same birthday among n people $\overline{p(365, n)}$ is

$$\overline{p(365, n)} = 1 - \frac{365!}{365^n(365 - n)!}$$

The probability that there is at least one identical piece of data when n pieces of data are uniformly obtained from M different types of data, which are uniformly distributed, is given by

$$\overline{p_1(M, n)} = 1 - \frac{M!}{M^n(M - n)!}$$

Furthermore, this theoretical value can be approximated using the Taylor expansion as follows

$$\overline{p_2(M, n)} = 1 - exp[-\frac{n(n-1)}{2M}]$$

Problem 7 *Consider two theoretical values $\overline{p_1(m, n)}$ and $\overline{p_2(m, n)}$ in the birthday paradox. Let's see the difference in the speed of convergence. Let us illustrate $\overline{p_1(m, n)}$ and $\overline{p_1(m, n)}$ using the values $m = 365, n = [1, 100]$.*

1. *What is difference between $\overline{p_1(m, n)}$ and $\overline{p_2(m, n)}$?*

2. *Discuss how to use the two theoretical values properly.*

This problem is a problem to learn how to evaluate the difference between theoretical and experimental values. Learn how to evaluate theoretical values using experimental values. Randomly generate a random number r from 0 to 10 using the field and base point of Q 3.1, $m = 4$, and $c = 4 \cdot g^r \pmod{23}$ (r is used to obtain a part of the ciphertext) is calculated n times, and a program that outputs to a file is constructed. However, a random number is generated every time.

1. Perform the encryption operation 100 times until the same ciphertext appears, and calculate the number of encryption times required until the matching ciphertext appears each time. Next, letting the minimum value n_{min} and the maximum value n_{max} of the number of occurrences be each occurrence number within the range $[n_{min}, n_{max}]$. Find the number of times a number has occurred, and plot it in a graph.

2. When the number of types is 10, the probability of each occurrence occurring in the range $[n_{min}, n_{max}]$ is calculated. Let's figure out by using $\overline{p_1(m, n)}$ and $\overline{p_2(m, n)}$ in the Birthday Paradox column.

3. Discuss the difference between the experimental value of (1) and the theoretical value of (2) using a graph.

4. （optional）Since the question (3) is using small numbers, bias may occur in the data. So, for example, for $p = 2^{31} - 1$, let $g = 1090433$, calculate g^k and do the experiment. The order of g is 331. This can also be used to check the accuracy of random number generation. `secret.choise` and `random.randint` can also be used.

Problem 8 *Let's consider an online method for seeking two participants with matching birthdays from this PBL. If several students responded to the online student questions, we can only look at one student's answer per question. Let's estimate how many times the same birthday can be sought for a question. That is, find the expected number of questions to be repeated until a pair with the same birthday is found. (The expected value depends on the method.) Try it out and points will be awarded for those who estimated accurately.*

E.3 ElGamal Cipher (Textbook chapter 8)

There are different levels of privacy. If you encrypt your name, the ciphertext does not reveal who you are. But sometimes that is not enough. What if the same ciphertext represents the same person? This concept is linkability. In this lecture, we will understand public-key cryptography, how to prevent linkability by applying quasi-isomorphism, one of the functions of cryptography, and the problems of linkability. Note that the ElGamal cipher satisfies multiplicative quasi-isomorphism.

E.3.1 ElGamal Cipher

We describe the key generation, encryption, and decryption of the ElGamal cipher.

【Key Generation】 User B generates a key pair consisting of a public key and a secret key as follows:

1. Generate a finite field $\mathbb{F}_p$ (where p is a prime) and a base point $g \in \mathbb{F}_p$ of order l.
2. Generate a random number $x \in \mathbb{Z}_l$ and use it as a secret key,

$$y = g^x \pmod{p}$$

⟨Public Key⟩ $p,\ g,\ y$
⟨Secret key⟩ x

【Encryption】 Suppose that user A wants to encrypt a plaintext $m \in \mathbb{Z}_l$ and send it to B.

1. Generate a random number $r \in \mathbb{Z}_l$. Then,

$$u = g^r \pmod{p} \tag{11}$$

2. Using B's public key y, A encrypts her message m as

$$c = y^r m \pmod{p} \tag{12}$$

and sends the ciphertext $(u, c) \in \mathbb{F}_p \times \mathbb{F}_p$ to B.

【Decryption】 Upon receiving the ciphertext (u, c), B decrypts c as follows.

$$m = c/u^x \pmod{p}. \tag{13}$$

Problem 9 *Answer the following questions when setting finite field $\mathbb{F}_p$, for the prime $p = 23$, and having base point $g = 2$.*

(1) We want to find the order of 2. Although there is a way to calculate $2^\ell = 1 \pmod{23}$ one by one, let's consider a theoretical way to find $\ell > 1$ with at most two power operations.[14]

[14]This question is to understand the relationship between $\mathbb{F}_p$'s original number and order.

(2) *Make a public key of ElGamal encryption using 2 as the secret key.*

(3) *Find the result of encrypting $m = 4$ with $r = 3$ using the key pair from (2). Make sure you can decrypt it.*

(4) *Use 2 to find the primitive root (the element whose order is 22) by one multiplication.*[15]

Exercise 6 *Use the example $\mathbb{F}_{p_4}$, $g_4 \in \mathbb{F}_{p_4}$, the order of g_4, ℓ_4 in Table 2. Implement the ElGamal cipher using finite field $\mathbb{F}_p$ according to the questions. In the implementation, the random numbers used for encryption are given as input.*

(1) *For the secret key $d_b = 172\,35091\,96654\,51459\,12345\,17144\,99640\,83306\,22345\,44321$ find the corresponding public key p_b.*

(2) *Using the public key and the following r, encrypt m.*

$$m = 123\,75081\,11111\,51459\,12345\,17144\,99640\,33333\,55555\,44444$$
$$r = 123\,45678\,91234\,56789\,12345\,67891\,23456\,78912\,34567\,89123$$

Make sure you can also decrypt!

(3) *Let's encrypt today's lunch using TA's public key and submit it to the bulletin board (in hex encoding). Represent the lunch so that it can be encrypted within 1024 bits*[16]*. If TA can decrypt your data correctly, s/he will submit OK to the bulletin board.*

The ElGamal cipher is a multiplicative Homomorphic cipher. Let us consider and design a secure application using the fact that it is a multiplicative quasi-isomorphic cipher. The design is to specify the user who has the private key of the ElGamal cipher and the user who encrypts with the public key. The design should also specify the server on which the public key will be posted. Also, it should be clear what is protected by the introduction of ElGamal encryption compared to before the introduction of ElGamal encryption.

A case study is provided for reference. For example, consider a proposal for a system to manage books in a library by putting book information in RFID tags and managing book check-out. RFID data is easily accessible from the outside. Since borrowed books are carried around, the name of the book can be written directly on the RFID tag. If the name of the book is written directly on the RFID tag, it is easy for people outside the library to obtain information about what kind of book was borrowed. In this case, it is necessary to strengthen privacy from the viewpoint of linkability as well as the confidentiality of book names. On the other hand, when a book is returned, it is necessary to be able to obtain information on which book was checked out from the RFID. From the viewpoint of protecting privacy information, it is necessary to protect privacy from information obtained from the RFID of a book that is being checked out. The book must be returned correctly while protecting privacy from the information obtained from the RFID of the book being checked out. Let's propose a library management system with ElGamal encryption that meets these requirements. If you can't think of anything, please think of the specific situations in which you would set up the cryptographic devices and how you would set up the keys.

E.3.2 Diffie-Hellman Key Sharing Method

Let us now introduce the Diffie-Hellman (DH) key sharing method. The DH key sharing method is a method of sharing keys using a public network.

We define the Diffie-Hellman public key exchange protocol using two phases.

[15] This question is to understand the relationship of the original order.

[16] 1 alphabet is 1 bytes, and one wide Japanese character (UTF-8) is 3 bytes.

【**System Parameters**】 Let $\mathbb{F}_p$ be a finite field and let $g \in \mathbb{F}_p$ be an element (base point) with a large prime number ℓ order. $\mathbb{F}_p$ and g are public parameters.
【**Key Sharing**】

1. **A** generates a random number $x_a \in \mathbb{Z}_\ell^*$, sets $y_a = g^{x_a} \pmod p$, and sends y_a to **B**. y_a is now the public key of **A**.

2. **B** likewise generates a random number $x_b \in \mathbb{Z}_\ell^*$, $y_b = g^{x_b} \pmod p$, and sends y_b to **A**. y_b is now the public key of **B**.

3. **A** uses the received y_b and x_a to calculate
$$K_{a,b} = y_b^{x_a} \bmod p$$
, and $K_{a,b} = g^{x_a x_b} \bmod p$ is the shared key.

4. **B** uses the received y_a and x_b to calculate
$$K_{a,b} = y_a^{x_b} \bmod p$$
, and $K_{a,b} = g^{x_a x_b} \bmod p$ is the shared key.

Exercise 7 *Use the finite field of 3.1.*
(1) Generate a random number r up to $\ell_4 - 1$, generate your private key, and then create your public key $g^r \pmod p$ and submit it to moodle. Also, save your public key on the public key bulletin board.
(2) Use the public key of your team's TA from moodle, enter your private key created in (1) and create a DH shared key K. Create a program that outputs the shared key to a file.
(3) Generate a random number r such that ID' = r || 0000 || ID has the same length of the shared key K. Next, calculate ID' ⊕ K using exclusive OR and submit ID' ⊕ K to the public key bulletin board.
(4) (FYI) When encrypting other data, please use `hexlify` *in the B chapter for character encoding.*
(5) TA decodes (3), confirms the decrypted ID, and posts it on the public key bulletin board. However, the data is converted to the hexadecimal encoding.

Consider the verification of the key shared by the DH key sharing method.
1. In the exercise 3.3, we shared the key with TA and verified that it was shared correctly. However, the verification method is not very safe. Discuss what is wrong with this verification method.
2. Let's think about how to verify that the secret was correctly shared by the DH key sharing method with the public NW (bulletin board available). Also, clarify the security policy (which part is considered secret and invisible to a third party). Furthermore, if possible, use the bulletin board system to verify that the TA and the shared key are correct with the constructed algorithm.

E.4 Discrete logarithm problem decoding

Let G be a finite cyclic group of order n written multiplicatively, and let g be a generator of G. In general, DLP is the problem of finding an integer x such that $h = g^x$ given $h \in G$. The difficulty of the discrete logarithm problem is due to the DH key-sharing methods introduced so far and to the ElGamal cryptography. In this chapter, you will learn how to solve the discrete logarithm problem and actually try to crack the cipher.

In general, DLP is the problem of finding an integer x such that $h = g^x$ given $h \in G$. One way to solve DLP is to find a suitable relation between g and h, e.g., find two integers a, b so that $g^a h^b = 1_G$, in which case $x = -a/b \bmod n$.

One possible way to obtain such a relation is through collisions.

That is, if we have $g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$ for $a_i \neq a_i$ and $b_j \neq b_j$, then we can immediately obtain $g^{a_1 - a_2} h^{b_1 - b_2} = 1_G$. Pollard's rho method is a way to create collisions.

E.4.1 ρ method

Pollard's rho method is a way to create collisions. The idea is to consider a function (not a homomorphism!) f from G to itself and its iterated functions:

$$f^{\circ m} = \underbrace{f \circ f \circ \cdots \circ f}_{m \text{ times}}, \text{ or } f^{\circ m}(x) = \underbrace{f(\cdots(f}_{m \text{ times}}(x))\cdots).$$

Starting from an element $x \in G$, let us consider the orbit of x under $f^{\circ m}$,

$$\left\{x_i = f^{\circ i}(x) : i = 0, 1, 2, \ldots\right\} = \left\{x, f(x), f(f(x)), \ldots\right\}.$$

What would this orbit look like for random f and random x? Since G is finite, it cannot contain any infinite non-repeating orbits. Thus any orbit will necessarily run into repetition at some point, leading to a picture like Figure 4.

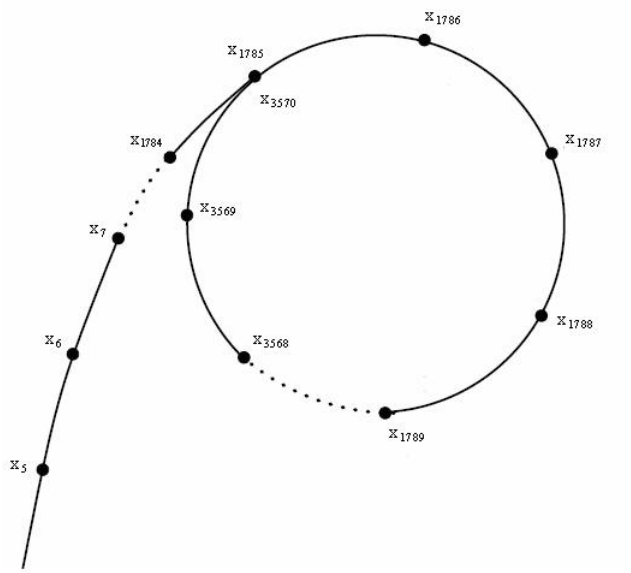


図 4: The column of t_i generated by ρ method

This looks like the Greek letter ρ and hence the name "rho method." On average, both the tail and the circle parts of ρ are of length $O(\sqrt{n})$ due to the usual birthday argument, so the complexity of the algorithm is $O(\sqrt{n})$.

Indeed, Pollard defined the function f as follows. First let us partition G as the disjoint union of three subsets S_1, S_2, and S_3 such that it is easy to figure out to which subset any given element belongs.

$$f(t) = \begin{cases} ht \text{ if } t \in S_1, \\ t^2 \text{ if } t \in S_2, \\ gt \text{ if } t \in S_3 \end{cases}$$

Now, for some integers a_0, b_0, let the initial value $t = g^{a_0}h^{b_0}$. While easily grasping the integers a_i, b_i satisfying $t_i = g^{a_i}h^{b_i}$, we can generate the source of G. The source of G can be generated. Thus when $t_i = t_j$. the collision $g^{a_i}h^{b_i} = g^{a_j}g^{b_j}$ can be found.

If we assume that each t_i generates a uniformly random source of G, then from an argument similar to the birthday paradox, we can expect that there are collisions in $O(\sqrt{n})$. That is, the ρ method can solve the discrete logarithm problem with a computational complexity of $O(\sqrt{n})$.

Problem 10 *(1) Let G be the multiplicative group $(\mathbb{Z}/101\mathbb{Z})^\times$, and let $g = 2$ and $h = 3$.*

$$f(t) = \begin{cases} ht \text{ if } t \in S_1 = \{t \in G \mid x \equiv 1 \mod 3\}, \\ t^2 \text{ if } t \in S_2 = \{t \in G \mid t \equiv 2 \mod 3\}, \\ gt \text{ if } t \in S_3 = \{t \in G \mid t \equiv 3 \mod 3\} \end{cases}$$

We want to find an integer x satisfying $g^x \equiv h \mod 101$. Following the ρ method with initial values $a_0 = 1$ and $b_0 = 0$. Find the collision and x by completing the rest of the following table. First, the steps up to $i = 2$ in this table are explained. The table is updated as follows.

- *when $i = 0$*
  for $a_0 = 1, b_0 = 0$, t

  $$t = g^1 \cdot h^0 = 2^1 \cdot 3^0$$

  Following the ρ method with initial values $a_0 = 1$ and $b_0 = 0$. Next, determine the range of t to transform the updated t as $f(t)$.

  $$t\%3 = 2\%3 \equiv 2$$

  From this t value, $f(t) = t^2$ is performed.

- *when $i = 1$*

  $$t = t^2 = (2^1 \cdot 3^0)^2 = 2^2 \cdot 3^0$$

  and t is updated so that $a_1 = 2, b_1 = 0$ when $i = 1$, as shown in the table. Furthermore, the range of t is determined in order to transform the updated t as $f(t)$ as well as $i = 0$.

  $$t\%3 = (2^2 \cdot 3^0)\%3 = 4\%3 \equiv 1$$

  Perform $f(t) = ht$ from this t value.

- *when $i = 2$*

  $$t = ht = 3 \cdot (2^2 \cdot 3^0) = 2^2 \cdot 3^1$$

  and t are updated so that $a_2 = 2, b_2 = 1$ at $i = 2$, as shown in the table. These are the steps up to $i = 2$. By repeating similar steps, find an integer x satisfying $g^x \equiv h \mod 101$.

Number of steps i	a_i	b_i	$2^{a_i}3^{b_i} \mod 101$
0	1	0	2
1	2	0	4
2	2	1	12
3	3	1	24
		$\vdots$	

(2) Check what the number of steps would be if different initial values were used in (1).
(3) If $h = 7$ in (1), find an integer x satisfying $g^x \equiv h \mod 101$. Note the initial value and the number of iterations.

E.4.2 Collision detection

Composition of function f How to make the function f is not restricted to the question 4.1. For example, using a similar update formula and keeping $S_1 = \{t \in G \mid 0 \leq t \leq 33\}$, $S_2 = \{t \in G \mid 34 \leq t \leq 67\}$, $S_3 = \{t \in G \mid 68 \leq t \leq 100\}$.

However, for S_2 where t^2 is calculated. Note that the unit source 1_G is not included. Because if $t_i = 1_G$, then $x_j = 1_G$ for all $j > i$ thereafter, and No new source is generated.

Reducing memory usage using Distinguished points. Rivest proposed a method to reduce the memory usage of the ρ method by retaining and comparing only points T_i that satisfy certain conditions. The points satisfying this condition are called distinguished points. The condition that is easy to check was used. For example, when only points that are multiples of d out of t_i are retained, the number of points to be retained is approximately $1/d$, which reduces the amount of memory used. On the other hand, the number of t_i generated for collision detection is approximately d more than in the straightforward ρ method. This is because collisions can only be found in the circular part (circle) in Figure 4 of d is generated for the second time.

Parallelisation by the λ method The generation of T_i in the ρ method can be easily parallelised. For example, in parallel with the generation of the original t_i from initial values (a_0, b_0). Consider the case where $t_i^{'}$ is generated from different initial values $(a_0^{'}, b_0^{'})$. If $t_i = t_j^{'}$ for some i, j when both use the same function f, then $t_{i+k} = t_{j+k}^{'}$ for all $k \geq 0$ thereafter. That is, the sequences generated after both have visited the same source once are equal, when one trajectory is considered to merge at the midpoint of the other trajectory. Since it looks like the Greek letter λ, the parallelised computation of the above ρ method is called the λ method. Naturally, the λ method works similarly for $N > 2$ kinds of initial values.

Floyd's Circulation Detection Method. Collisions in the column $\{T_i\}$ can also be found using Floyd's circular detection method. Floyd's circular detection method is also known as the hare and tortoise algorithm. It detects collisions in a column by repeatedly comparing the binary values of t_{2i} (the hare) and t_i (the tortoise). collisions in a column by repeatedly comparing t_{2i} (the hare) and t_i (the tortoise). It does not need to keep all the values of t_i up to collision detection, as in the straightforward ρ method, and has the feature of very low memory usage. It is characterised by very low memory usage.

We next analyse the computational complexity of Floyd's circular detection method. In Figure 4, let o be the length of the circular part (circle) and The other lengths are ℓ. At the ℓth binary comparison, the tortoise t_ℓ is at the start of the circular section and t_ℓ is at the end of the circular section. Rabbit $t_{2\ell}$ exists at its ℓ step ahead. Since the hare continues around the circulatory section, the hare at this time is $o - (\ell \mod o)$, i.e. at most o steps behind the tortoise. Since the hare decreases its distance from the tortoise by 1 for each additional i, it can be seen that The total number of binary comparisons is at most $\ell + o$ times. Collisions can be detected by at most $\ell + o$times binary comparisons in total. From the discussion in E.4.1 chapter, the magnitude of $\ell + o$ is $O(\sqrt{n})$ and The computational complexity of Floyd's circular detection method is $O(\sqrt{n})$. In addition, to obtain t_{i+1} and $t_{2(i+1)}$ based on t_i and t_{2i}, the function f must be calculated three times. Therefore, the computation time of Floyd's circular detection method is generally larger than that of the straightforward ρ method. Note that parallelisation by the λ method cannot be applied to Floyd's circular detection method.

Problem 11 *Let G be $(\mathbb{Z}/47\mathbb{Z})^\times$ and $g = 16, h = 32 \in G$. The function f is defined as in Problem 10. Use $g^{23} \mod 47 = 1$, $(a_i = 1, b_i = 0)$, and $(a_{2i} = 1, b_{2i} = 1)$ to solve by the Floyd method.*

E.4.3 Team Task

Work together with your group members on the following three tasks. Report the secret key found and the number of iterations and initial conditions in a team presentation. Afterwards, submit them to moodle.

Assignment 1 (5) Find an integer x satisfying the following equation.

$$397628281^x = 480992339 \mod 554860571$$

$$Hint : 397628281^{5044187} = 1 \mod 554860571$$

Assignment 2 (5) Find an integer x satisfying the following equation.

$$383929062152^x = 405481391324 \mod 1023417167557.$$

$$Hint : 383929062152^{1740505387} = 1 \mod 1023417167557.$$

Assignment 3 (20) Find an integer x satisfying the following equation.

$$472974174173350^x = 176809774202328 \mod 497264225202533.$$

$$Hint : 472974174173350^{124316056300633} = 1 \mod 497264225202533.$$

Assignment 4 (30) Find an integer x satisfying the following equation.

$$431147972333069417^x = 273418759425577275 \mod 817108219307103587$$

$$Hint : 431147972333069417^{408554109653551793} = 1 \mod 817108219307103587$$

Assignment 5 (40) Find an integer x satisfying the following equation.

$$6571397331495670129 77^x = 772772310495087727220 \mod 1077984309859658267861$$

$$Hint : 6571397331495670129 77^{8693421853706921515} = 1 \mod 1077984309859658267861$$

E.4.4 Public Key Cryptography Applications

Problem 12 *Company A is a large company with clients worldwide and in Japan. The Retail and Corporate Operations Department of this Company A must encrypt the contents of notes taken at meetings using ElGamal ciphers, and send the encrypted contents to the centre for storage. The actual storage method using the ElGamal cipher is as follows.*

1. *The centre generates the key and sends the key generated by the centre to the Retail and Corporate Operations Department.*
2. *Each member of the Retail and Corporate Business Unit encrypts the content of the memo using the ElGamal cipher with the key sent by the centre.*
3. *Each member of the Retail and Corporate Business Department sends the encrypted cipher values to the centre. After receiving the cipher values, the*
4. *centre automatically replies to the individual with the cipher values, stating that it has received them.*

One day, he receives a confirmation e-mail containing his message and a confirmation of receipt from Mr B's centre, which is different from his own. At this time, answer the following questions. The confirmation email to yourself and the confirmation email from Mr B received are as follows. However, use g_4, p_4, ℓ_4 in Table 2.1. The other parameters are to be determined individually.

- *Confirmation message to yourself.*

(m,u,c)=(天津麻婆豆腐,3922912043886732113555706249988353683619621962874435825575429695988721656496540249092316605311
1101878643356572052966228316810431584826316521386496752357562253989344165931277655452120896724553711286015105016857956574173445781631261817710715347576959122803294596702413158479524552845645110431
1797958153260756283877,
7794363403986875742551705965938653641971961358302357710779268847054910782400009859497191787953159232712480053939384339503665228121269303201599335807566047640883105495946847000589941566285266105918820291665411262250409839928672487194907637905295569095992636015067314035706159072017198971348044154796260162859)

- *Mr B*

(u,c) = (3922912043886732113555706249988353683619621962874435825575429695988721656496540249092316605311018786433565720529662283168104315848263165213864967523575622539893441659312776554521208967245537112860151050168579565741734457816312618177107153475769591228032945967024131584795245528456451104317979581532607562838 7,
1970361640236763266087544093328762519026320418528805049200737363611161113542820028301669724577348295765782544969448568411385281090027558697959138373012289289274317943020816337647322034577137332994989617965130792764688135170503949659528246572081216956145319961588074960738293299981275136067785956539724895881)
(1) Create your own free message or random value and create an ElGamal cipher for that message.
(2) Decrypt Mr B's message.

E.4.5 The definitions and Properties of Zero-Knowledge Proof

Zero-knowledge proof is a method by which one party can prove to another party that a given statement is true while the prover avoids conveying any additional information apart from the fact that the statement is indeed true.

The zero-knowledge proof is first formalized in Goldwasser, Micali, and Rackoff's paper in 1985. Since then, zero-knowledge proof became a popular research topic and an important concept in cryptography.

A zero-knowledge proof protocol must provide the following three properties.

1. Completeness: if the statement is true, an honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
2. Soundness: if the statement is false, no cheating prover can convince an honest verifier that it is true, except with some small probability.
3. Zero-knowledge: if the statement is true, no verifier learns anything other than the fact that the statement is true.

E.4.6 Abstract Example: Person Guessing Game

Alice, the verifier, presents a picture to Bob, the prover. The question to Bob is, "Guess Wally in the picture". Bob looks at the picture and finds Wally. However, if we tell Alice the location of the picture, she will also know the location of the picture, so we do not want to specify the location of the picture directly.

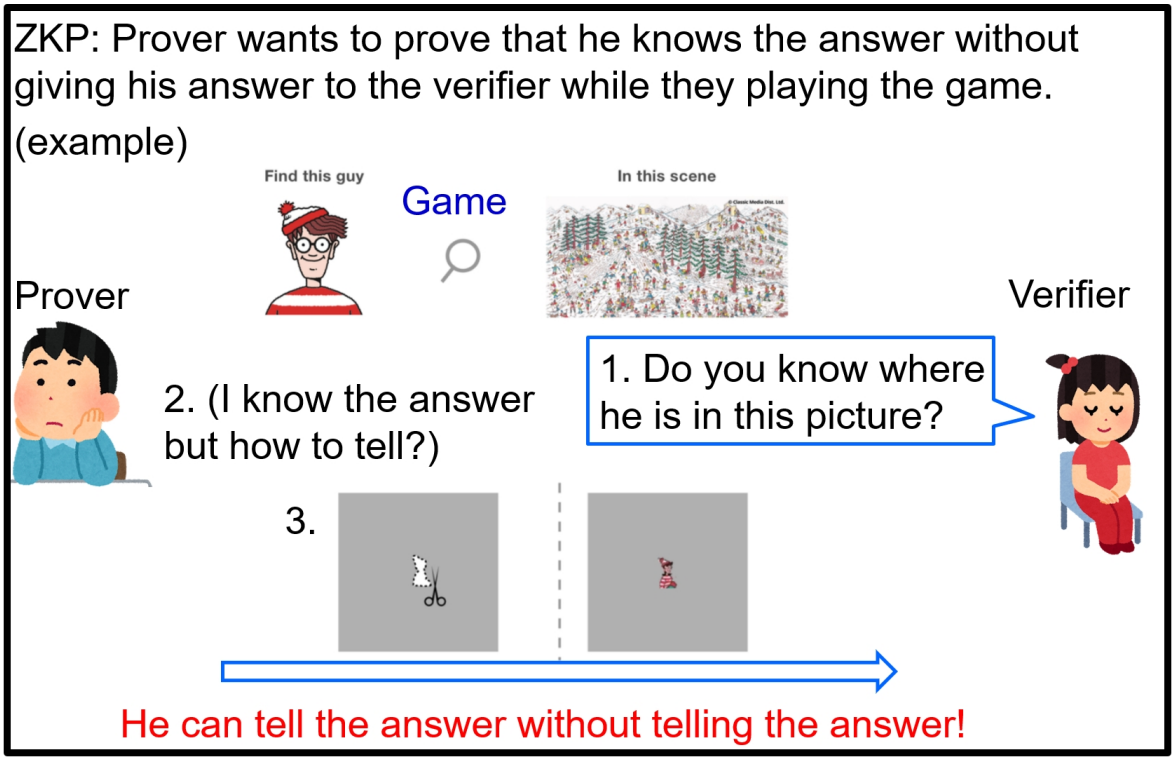


図 5: Wally guessing game

Zero-knowledge interactive proof flow for the wally guessing game:

1. Bob prepares a sheet of colored paper that can cover a picture.
2. Bob cuts the colored paper to fit Wally's size.
3. Bob puts the colored paper on the cut-out area so that the picture of Wally appears as shown in Figure E.4.6. At this point, glue the colored paper and the picture together.
4. Send the picture to Alice.

We now show that this protocol fulfills the three properties.

Completeness: If Bob the prover does not commit any deception, Alice can see from the colored paper that Bob has definitely guessed Wally.

Soundness: If Bob the prover is faking, Alice cannot find Wally no matter how much she searches in the picture.

Zero-knowledge: Since Alice cannot take Bob's colored paper, she cannot obtain information about where Wally is.

E.4.7 Yao's Millionaires' Problem

The applications of zero-knowledge proof include: public-key cryptography, digital signature, user authentication, etc. It is also known that the possession of a proof of any NP-complete problem can be shown by a zero-knowledge proof. Furthermore, there are also various applications of zero-knowledge proof that can be applied to multi-party computation. Here, we describe an example of multi-party computation that uses the ElGamal cryptosystem.

Alice and Bob each has a number (0 - 9), and they want to compare their numbers without seeing each other's number. How can they compare their numbers without knowing the value of the other person's number? (This is the millionaires' problem proposed by Andrew C. Yao in 1982.[17])

The ElGamal cryptosystem can be used to construct a zero-knowledge proof protocol that solves this problem. The procedures are as follows:

1. Alice creates an ElGamal key pair and publishes the public key.
2. Bob randomly chooses an integer x of N-bit with uniform probability, encrypts x to ciphertext (u, c) using Alice's public key, and sends $(u, c - n_B)$ to Alice.
3. Alice computes $(u, c - n_B + i), \forall i \in \{0, 1, \ldots, 9\}$ and decrypts them to get $\{m_0, m_1, \ldots, m_9\}$.
4. Alice uniformly samples a prime number p of length $\frac{N}{2}$-bit and computes $z_i = m_i \mod p$. If z_i and z_j are the same for any $i \neq j$, repeat this procedure.
5. Alice sends $\{z_0, \ldots, z_{n_A}, z_{n_A+1} + 1, \ldots, z_9 + 1\} (\mod p)$ and the prime p to Bob.
6. If z_{n_B} is equal to $x \mod p$, then $n_A \geq n_B$.
7. Bob shares this result to Alice.

E.4.8 Secret Key Zero-Knowledge Proof

The Schnorr non-interactive zero-knowledge proof (Schnorr NIZK proof), a zero knowledge proof protocol that proves the knowledge of the discrete log of a number, is registered as RFC 8235 in 2017[18]. This protocol can be utilized to prove the possession of the private key of an ElGamal cryptosystem. Furthermore, an interactive version of this protocol also exists, and the procedures of an interactive protocol are as follows:

1. Alice randomly generates a number $r \in \mathbb{Z}_l^*$ and sends $a = g^r \mod p$ to Bob.
2. Bob uniformly samples a number $b \in \mathbb{Z}_l$ and sends to Alice.
3. Alice sends $c = r + bx \mod l$ to Bob.
4. If $g^c \mod p$ equals $ay^b \mod p$, then Bob can know that Alice has the discrete log of y and thus has the private key.

We now show that this protocol fulfills the three properties. 1. Completeness: If Alice has the key and follows the protocol, then Bob can be convinced by checking the equality of $g^c \mod p$ and $ay^b \mod p$. 2. Soundness: If Alice does not have the key, the only way to convince Bob is to predict the number b chosen by Bob and calculates (a, c) such that $g^c \equiv ay^b (\mod p)$. However, the probability of a correct prediction is only $\frac{1}{l}$. Therefore, Bob has a high probability, $1 - \frac{1}{l}$, to know that Alice does not have the key. 3. Zero-knowledge: Bob does not know anything other than a and c.

The procedures of the non-interactive version are as follows:

1. Alice samples a random $r \in \mathbb{Z}_l^*$, calculates $a = g^r \mod p$, $b = H(g||a||y||p||l)$, and $c = r + bx \mod l$, and send them to Bob. (H is a hash function such as SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, or SHA3-512.)
2. Bob calculates $a = \frac{g^c}{y^b} \mod p$. If b equals $H(g||a||y||p||l)$, then Alice has the private key.

[17] Yao, Andrew C. "Protocols for secure computations." 23rd annual symposium on foundations of computer science (sfcs 1982). IEEE, 1982.
[18] https://datatracker.ietf.org/doc/html/rfc8235 "Schnorr Non-interactive Zero-Knowledge Proof"

Problem 13 *Please extend the protocol for Yao's Millionaires' Problem such that it can support the comparison of any number, not limited to numbers between 0 to 9.*

Exercise 8 *Use the ElGamal cipher implemented in Exercise 6. Implement the protocol for comparing numbers by Zero Knowledge. Then send* $\{z_1, \cdots, z_{12} + 1\}$ *using the ciphertext received from TA, and compare your birthday month with that of TA by zero-knowledge. (In Python, Sympy.isprime(x) can be used to determine prime numbers.)*

E.5 Lecture 4

E.5.1 ρ method

Introduction to the Discrete Logarithm Problem

Let G be a finite cyclic group with respect to multiplication. The discrete logarithm problem is the problem of finding an integer x that is a discrete logarithm from the generator g and the element h in G, where $h = g^x$.

$$(g, h = g^x) \rightarrow x$$

The difficulty of the discrete logarithm problem forms the basis for the security of the Diffie-Hellman key exchange method and ElGamal encryption. In this chapter, we will learn decryption techniques for solving the discrete logarithm problem and attempt to decrypt a cipher.

Let n be the order of G. The discrete logarithm problem can be solved using two different representations (often referred to as "collisions") of elements in G. That is, if there exist integers a_i, a_j, b_i, and b_j (where i and j are different indices) such that $a_i \neq a_j$ and $b_i \neq b_j$ and the collision equation

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j}$$

is satisfied, then we can transform it as $g^{a_i - a_j} h^{b_i - b_j} = 1_G$, and solve for x as $x = (a_j - a_i)/(b_i - b_j) \bmod n$. In the following sections, we will explain Pollard's ρ method, which is a technique for finding collisions.

Pollard Rho Method

The Pollard Rho method is a prime factorization algorithm used to determine the prime factors of a given integer. It operates on the principle of finding cycles in a sequence generated by a specific function. Here's the mathematical structure of the algorithm:

Algorithm

1. Choose a random starting value x and a function $f(x)$.
2. Initialize two variables a and b to the starting value x.
3. Repeat the following steps until a factor is found:
   (a) Update a by applying the function to it: $a = f(a)$.
   (b) Update b by applying the function twice: $b = f(f(b))$.
   (c) Calculate the greatest common divisor (GCD) of the absolute difference between a and b and the original number.
   (d) If the GCD is not 1, it is a non-trivial factor of the original number, and the algorithm terminates.

(e) If the GCD is 1, continue to the next iteration.

4. If a and b are equal at any point, a cycle has been detected. Restart the process with a new random starting value and function until a non-trivial factor is found or a predefined number of iterations is reached.

E.6 Blockchain

Decentralized monetary system is a research topic since a long time ago. Bitcoin was proposed in 2009 to be the first Blockchain. Ethereum was proposed in 2014, to provide an extra feature called Smart Contracts. Smart Contract is a program that runs by the Blockchain and can send/receive money. In this lecture, we use Bitcoin as an example. Important properties are listed below:

- **No trusted third party:** The system is decentralized. No single party can control the system.
- **Publicly Verifiable:** All data stored on the blockchain are public.
- **Immutable:** Each block's hash value is included in the next block.
- **No double spending:** Assume Alice sends \$100 to Bob, Alice cannot send the same \$100 to Charlie.

E.6.1 Keys and Accounts

Bitcoin uses the curve P256k1. An account can be created step by step.

- **Secret key:** Randomly pick a 256-bit integer that is larger than 0 and small than the order of curve P256k1.

$$x = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD$$

- **Public key:** G is the base point defined by P256k1. The public key can be calculated as

$$\begin{aligned} Y =& x \cdot G \\ =& 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD \cdot G \\ =& (F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A, \\ & 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB) \end{aligned}$$

- **Address:** The address is the last 20 bytes of the hashed and compressed public key.

$$address = 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy$$

E.6.2 Transactions

Assume Alice sent \$10 to Bob and Bob send \$10 to Charlie. When Bob wants to send money to Charlie, Bob needs to create a transaction and sign it with his secret key. Everyone can verify Bob's signature by his public key. Each transaction is chained one after another. In Figure 6, we demonstrate Bitcoin transactions.

演習 E.1 *Assume a simplified transaction as follows. A user sends 10 Bitcoin to the public key y_4 in Table 2.*

(1) Please verify the signature. If the signature is valid, y_4 received the Bitcoin correctly.

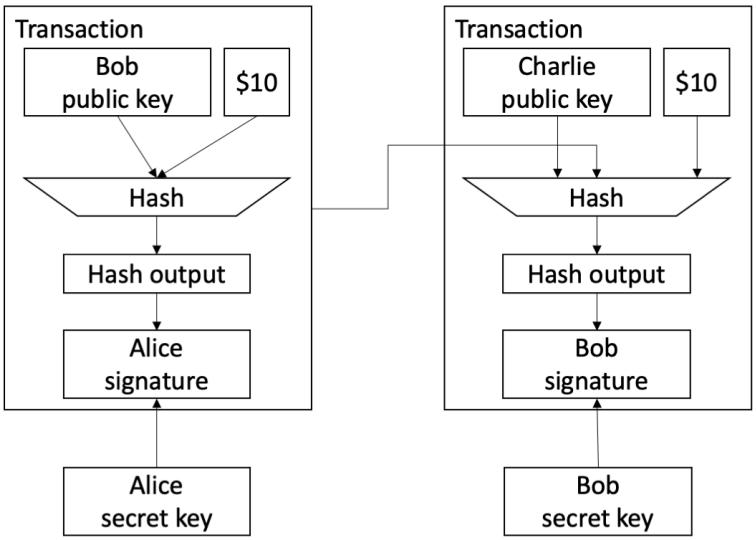


図 6: Transactions


```
{
    "transaction": {
        "sender": 10787370389507249913234092798381729525041937566920316815553191028684298987260472593326
        "receiver": 42063703388261034067155901193984173769497588090965563568973262694193054862536363577
        "value": 10,
        "previous tx hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
    },
    "signature": [136025170159578578824492310771481539418060249279O, 466426151316144309469694155127137790
}
```


演習 E.2 *The code of how the signature is generated is attached. Use the parameters in Table 2 and refer to the sample codes to answer this question.*

(1) Create a transaction, send the $10 *back to the sender.*

(2) Sign the transaction.

(3) Verify your transaction.


```
tx = '{"sender": 10787370389507249913234092798381729525041937566920316815553191028684298987260472593326310
```


```
tx = json.loads(tx)
sha256_hash = hashlib.sha256()
sha256_hash.update(json.dumps(tx).encode())
m = int.from_bytes(sha256_hash.digest(), byteorder='big')

r = randrange(1, L4)
u = pow(G4, r, P4) % L4
v = (pow(r, -1, L4) * (m + X4 * u)) % L4
```

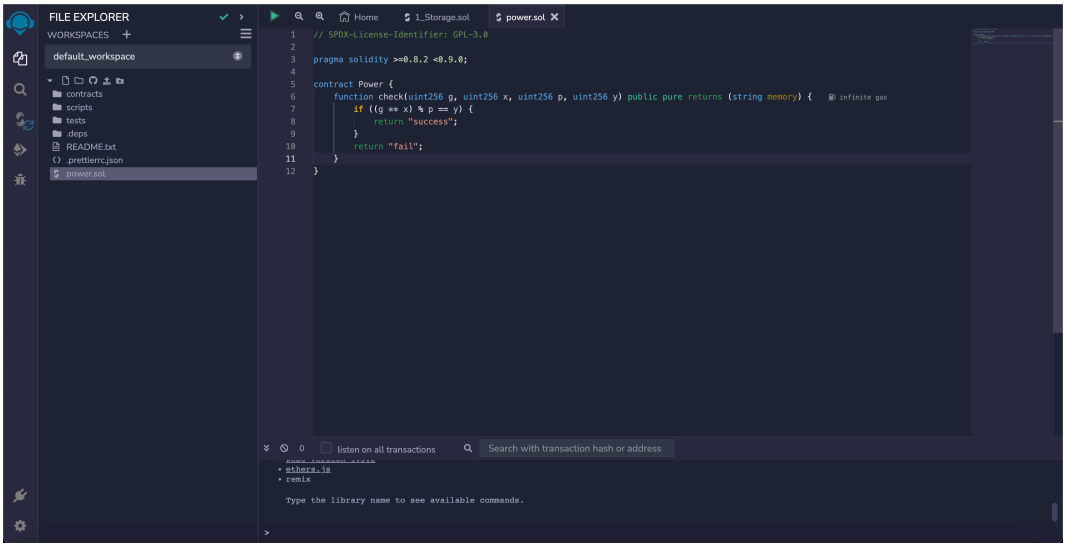


図 7: Remix

```
print(f"u {u}")
print(f"v {v}")
```

E.6.3 Smart Contract

Smart Contract is a program stored on the Blockchain. Solidity is the programming language to write a Smart Contract. We can define variables and functions perform computations and transfer ETH in a Smart Contract. Please visit `https://remix.ethereum.org/`, an online Smart Contract editor.

演習 E.3 *We provide an example Smart Contract of checking power. If you give the correct input, it will output "success". If it failed, you can increase the gas limit and try again. Gas is the unit of computation time in a Smart Contract. More computation needs more gas.*

(1) Try to get a success output!

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2 <0.9.0;

contract Power {
    function check(uint256 g, uint256 x, uint256 p, uint256 y) public pure returns (string memory) {
        if ((g ** x) % p == y % p) {
            return "success";
        }
        return "fail";
    }
}
```

F サンプルデータ

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

F.1 演習 1(担当：寺田)

演習 1.1

(1)g_1^{-1} = 898846567431157953854195783968937265989301480243780058532222118420985901080792596844739168979324627707510902827429902518232202740996195500253964385016779083196147765681195382543678799574112874312875037126510387238562947754789688892122212133086673638146496938343546028030251354054214538464660095640972338135040

(2)g_2^{-1} = 821484817262239251969635763777105390046478038017856655297330390099108723753236748333535192349309705091939781096947461344831318302761981726073772168055703404775276662726712453851705527301598144543104002685067262894641438675383700710414010579428196578006577560415835174664328157385713094912561580386308589901853

(3)g_3^{-1} = 1744989488652585946716122939755801864980004067678984986373240865827214303179365549800008755120887637445884359551327020182996125701005783177084195429526934512547305341322837555076031432759146574264525155130585984340031746934189213459920153214298509765168768201613489608738399858256174953338096104610915359715740

(4)g_4^{-1} = 158324804209047446163952569474573383821323037294730709492283949936547004492689998372090963561491531631999825917117341116724730792920802150497912125203187524279263996068059441287272740626580121169167631813325756995480763080354946262513867336445301273025763771499350812372751569492635020992567228945252431129198

F.2 演習 2.1(担当：Kaiming)

Exercise 2.1

```
def inv(a,n):
    return mod_binary(a,n-2,1024,n)

def modBinaryExp(k):
    mult_count=0
    square_count=0
    k=bin(k).replace('0b','')
    l=len(k)
    if l<1024:
        k=(1024-l)*"0"+k
    for i in range(0,1024):
        if int(k[i]) == 1:
            mult_count+=1
            square_count+=1
        else:
            square_count+=1
    return [square_count,mult_count]
```

```
p_1 = 17976931348623159077083915679378745319786029604875601170644442368419718021615851936894783379586492554150218056548598050364644054819923910005079287700335581663922955313623907650873575991482257486257500742530207744771251893907650873102345678910234567890234567891023456789
gs = [
        2,
        74477125891023456789,
        22955313623907650873575991482274477125893907650873102345678910234567890234567891023456789,
        744771258922955313623907650873575991
        48227447712589390765087310234567891023456789
    ]
def invExp_modBinary(a, p):
    k=bin(p-2).replace('0b','')
    l=len(k)
    counter=0
    y = 1
    for i in range(0,1024):
        counter+=1
        if i<1024-l:
            y = Mod(y ** 2,p)
            continue
        if int(k[i])==1:
            y = Mod(Mod(y ** 2,p) * a,p)
        else:
            y = Mod(y ** 2,p)
    return [y,counter]

def exercise_2_1():
    p_1 = 17976931348623159077083915679378745319786029604875601170644442368419718021615851936894783379586492554150218056548598050364644054819923910005079287700335581663922955313623907650873575991482257486257500742530207744771251893907650873102345678910234567890234567891023456789
    gs = [
            2,
            74477125891023456789,
            22955313623907650873575991482274477125893907650873102345678910234567890234567891023456789,
            744771258922955313623907650873575991482274477125893907650873102345678910234567890234567891023456789
        ]
    print("The inversions of following elements in group G_p1:")
    for g in gs:
        print("\nThe inversion of",g,"is:")
        [y,counter]=invExp_modBinary(g,p_1)
        print(y)
        t=modBinaryExp(p_1-2)
```

```
        print("The number of square operations is:",t[0])
        print("The number of multiplication operations is:",t[1])
        print("The number of loops is:",counter)
    # 160:=M_160, 160:1024=1:6, M:S=1:0.8
        rst=(t[0]*0.8+t[1])*6
        print("The amount of 1024-bit calculation is:",rst,"M160")
```

Result:

The inversions of the following elements in group G_p1:
The inversion of 2 is:
898846567431157953854195783968937265989301480243780058532222118420985901080792596844739168979324627707510902
The number of square operations is: 1024
The number of multiplication operations is: 569
The number of loops is: 1024
The amount of 1024-bit calculation is: 8329.2 M160
The inversion of 74477125891023456789 is:
821484817262239251969635763777105390046478038017856655297330390099108723753236748335351923493097050919397810
The number of square operations is: 1024
The number of multiplication operations is: 569
The number of loops is: 1024
The amount of 1024-bit calculation is: 8329.2 M160
The inversion of 22955313623907650873575991482274477125893907650873102345678910234567891 is:
174498948865258594671612293975580186498000406767898498637324086582721430317936554980000875512088763744588435
The number of square operations is: 1024
The number of multiplication operations is: 569
The number of loops is: 1024
The amount of 1024-bit calculation is: 8329.2 M160
The inversion of 744771258922955313623907650873575991482274477125893907650873102345678910234567891 is:
158324804209047446163952569474573383821323037294730709492283949936547004492689998372090963561491531631999825
The number of square operations is: 1024
The number of multiplication operations is: 569
The number of loops is: 1024
The amount of 1024-bit calculation is: 8329.2 M160

Analysis 4

(1)

```
def mv(data):
    return [np.mean(data),np.var(data)]
def analysis_4():
    p=2**12-7
    print(bin(p-2)[2:])
    data=[]
    [s,m]=modBinaryExp(p-2)
#     for i in range(1,p):
```

```
    print(s,m,mv(s),mv(m))

analysis_4()
```

Result: The number of loops is: 12 111111110111 12 11 [12.0, 0.0] [11.0, 0.0]

(2)

```
import matplotlib.pyplot as plt
import matplotlib
import pandas as pd
import seaborn as sns
from scipy.stats import f_oneway
from scipy.stats import mannwhitneyu
from scipy.stats import ttest_ind
from scipy.stats import shapiro
def time_check(ts, te):
    return te-ts
def run_EX(a,b,k):
    for i in range(k):
        ex_euclid(a,b)
def test_EX(ind,p,k):
    time_list=[]
    for i in ind:
        start=time.time()
        run_EX(p,i,k)
        end=time.time()
        time_list.append(time_check(end-start))
    return time_list
def run_Fer(g,p,k,n):
    for i in range(k):
        mod_binary(g,p-2, n,p)
def test_Fer(ind,p,k,n):
    time_list=[]
    for i in ind:
        start=time.time()
        run_Fer(i,p,k,n)
        end=time.time()
        time_list.append(time_check(end-start))
    return time_list

def draw_fig(num_data, data_x, data_y, x_lavel, y_label, color, fig_label, legend, fig_name
    plt.title(fig_name)
    plt.xlabel(x_lavel)
    plt.ylabel(y_label)
    plt.plot(data_x, data_y, label=fun)
    plt.legend()
```

```
    plt.show()
def analysis_Fer(p,k,n):
    order=[2**i for i in range(1,255)]
    ind=[i for i in range(0,254)]
    Fer_list=test_Fer(order,p,k,n)
    draw_fig(254, order, ind, "Element's index", str(k)+" Running Time", "Blue", "Fermat",
    return Fer_list
def analysis_EXG(p,k,n):
    order=[2**i for i in range(1,255)]
    ind=[i for i in range(0,254)]
    EX_list=test_EX(order,p,k)
#     display(ind,"EXGCD",EX_list)
    draw_fig(254, order, ind, "Element's index", str(k)+" Running Time", "Blue", "EXGCD", "
    return EX_list
def analysis_Fer_rnd(lst,p,k,n):
    ind=[i for i in range(0,len(lst))]
    Fer_list=test_Fer(lst,p,k,n)
    draw_fig(len(lst), ind, Fer_list, "Element's index", str(k)+" Running Time", "Blue", "F
    return Fer_list
def analysis_EXG_rnd(lst,p,k):
    ind=[i for i in range(0,len(lst))]
    EX_list=test_EX(lst,p,k)
    draw_fig(len(lst), ind, Fer_list, "Element's index", str(k)+" Running Time", "Blue", "EX
    return EX_list
p_2=115792089210356248762697446949407573530086143415290314195533631308867097853951
n_2=256
p_3=2**12-77
n_3=12
l1=analysis_Fer(p_2,1000,256)
l2=analysis_EXG(p_2,10000,256)
t1=analysis_Fer_rnd(ind,p_2,10,256)
t2=analysis_EXG_rnd(ind,p_2,10)
```

Result:

Average time and variance of inversion by Fermat's little theorem:
[0.0003599060535430908, 2.444862838462336e-07]
Average time and variance of inversion by exGCD:
[8.868014812469483e-05, 8.354549587168945e-08]
Average time and variance of inversion by Fermat's little theorem:
[4.908204078674317e-06, 4.89675043796467e-09]
Average time and variance of inversion by exGCD:
[2.6918411254882812e-06, 2.678014918037661e-09]

(3)

```
import sympy
def test():
```

```
    t=0
    pl=[]
    while t<5:
        p1=secrets.randbits(256)
        if sympy.isprime(p1):
            pl.append(p1)
            t+=1
    for i in pl:
        print(i)
        rnd_below(i)
test()
```

Result:

289540393139907372339430836105384686676962651933628846074727401190086137504 03

Average time and variance of inversion by Fermat's little theorem:

[0.00034961304664611815, 2.3095280727173074e-07]

Average time and variance of inversion by exGCD:

[8.306362628936767e-05, 7.627907941612136e-08]

890407519516404372301737935982436003336949057289513052976595524164794738531 83

Average time and variance of inversion by Fermat's little theorem:

[0.0003334194183349609, 2.2582847150182717e-07]

Average time and variance of inversion by exGCD:

[9.158647060394287e-05, 8.32380150812781e-08]

807205950707037525123653418012165980881668250982693622131304143497625412716 99

Average time and variance of inversion by Fermat's little theorem:

[0.0003308450937271118, 2.2212491194341e-07]

Average time and variance of inversion by exGCD:

[7.885479927062988e-05, 7.270922536230274e-08]

450389273253415486745715174063702872438962288960487254662712493356229571829

Average time and variance of inversion by Fermat's little theorem:

[0.0003285168409347534, 2.2266551525261493e-07]

Average time and variance of inversion by exGCD:

[8.40954303741455e-05, 7.750896983252235e-08]

98741272278552323125536538152056367099338334033476820801782658816370959642809

Average time and variance of inversion by Fermat's little theorem:

[0.00032133071422576906, 2.1885716585497394e-07]

Average time and variance of inversion by exGCD:

[7.359845638275146e-05, 6.828444761865227e-08]

It can be seen that different p will cause significant changes when using the ex-Euclid algorithm, the changes of variance of Fermat's little theorem are small. Thus, Fermat's little theorem can resist the side channel attack.

(4)

```
def test_rnd_Int():
    random.seed(12345)
    t=[random.randint(0,255) for i in range(10)]
```

```
    print(t)
test_rnd_Int()
random.seed(12345)
print(random.randint(0,255))
print(random.randint(0,255))
print(random.randint(0,255))
print(random.randint(0,255))
print(random.randint(0,255))
```

Result:
[213, 5, 152, 188, 99, 138, 223, 82, 191, 63]
213 5 152 188 99
We can see that if the seed is fixed, the result is predictable. Thus, the randint is insecure.

Exercise 2.2

```
def find_square_root(a, p):
    return mod_binary(int((p+1)/4),a,p)

print(find_square_root(2,23))
```

Output: 18 Check: $18^2 = 324 mod 23 = 2$.

F.3 演習 2.2(担当：上杉)

F.4 演習 3.1(担当：和泉)

F.5 演習 3.2(担当：前野)

F.6 演習 4.1(担当：Nas・田川)

F.7 演習 4.2(担当：tony・山下)

F.8 演習 5(担当：Mathieu)

F.9 演習 6-1 (担当：山月)

F.10 演習 7-1 (担当：He)

G 解答

G.1 演習 1(担当：寺田)

演習 1.1

(1)g_1^{-1} = 898846567431157953854195783968937265989301480243780058532222118420985901080792596844739168979324627707510902827429902518232202740996195500253964385016779083196147765681195382543678799574112874312875037126510387238562947754789688892122212133086673638146496938343546028030251354054 21

45384646600956409723381350 4

(2)g_2^{-1} = 82148481726223925196963576377710539004647803801785665529733039009910872375323674833535192349309705091939781096947461344831318302761981726073772168055703404775276662726712453851705527301598144543104002685067262894641438675383700710414010579428196578006577560415835174664328157385713094912561580386308589901853

(3)g_3^{-1} = 17449894886525859467161229397558018649800040676789849863732408658272143031793655498000087551208876374458843595513270201829961257010057831770841954295269345125473053413228375550760314327591465742645251551305859843400317469341892134599201532142985097651687682016134896087383998582561749533380961046109153597157 4

(4)g_4^{-1} = 15832480420904744616395256947457338382132303729473070949228394993654700449268999837209096356149153163199982591711734111672473079292080215049791212520318752427926399606805944128727274062658012116916763181332575699548076308035494626251386733644530127302576377149935081237275156949263502099256722894525243112919 8

演習 1.2

(1) - (4) 同上

解析 1

(1)
最大ステップ数：
最小ステップ数：
平均値：
分散：

G.2　演習 2.1(担当：Kaiming)

G.3　演習 2.2(担当：上杉)

問 2.1

(1)
$p_1(\bar{M}, n), p_2(\bar{M}, n)$ を図示すると以下のようになる.
(2)
$p_1(\bar{M}, n), p_2(\bar{M}, n)$ の誤差を図示した結果, 以下のようになる． 図から, 誤差は $n = 20$ までは滑らかに増加しており, n が 40 付近で誤差がピークとなっており, 以降は滑らかに減少し, $n = 100$ になると再び 0 に近づく. (3)
(2) の誤差のグラフから $n = 20$ 以下の場合は近似値を使用してもよいが, n が 20 よりも大きい値の場合は理論値を使用した方がよい.

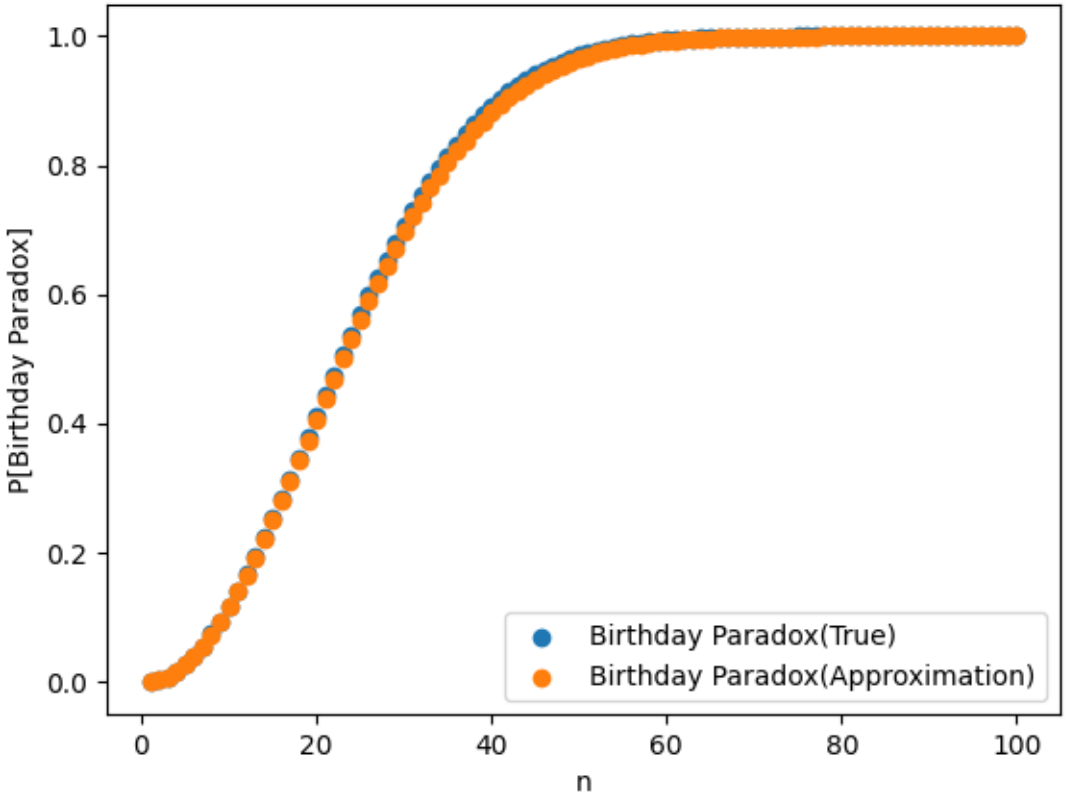


図 8: Birthday Paradox(理論値),Birthday Paradox(近似値)

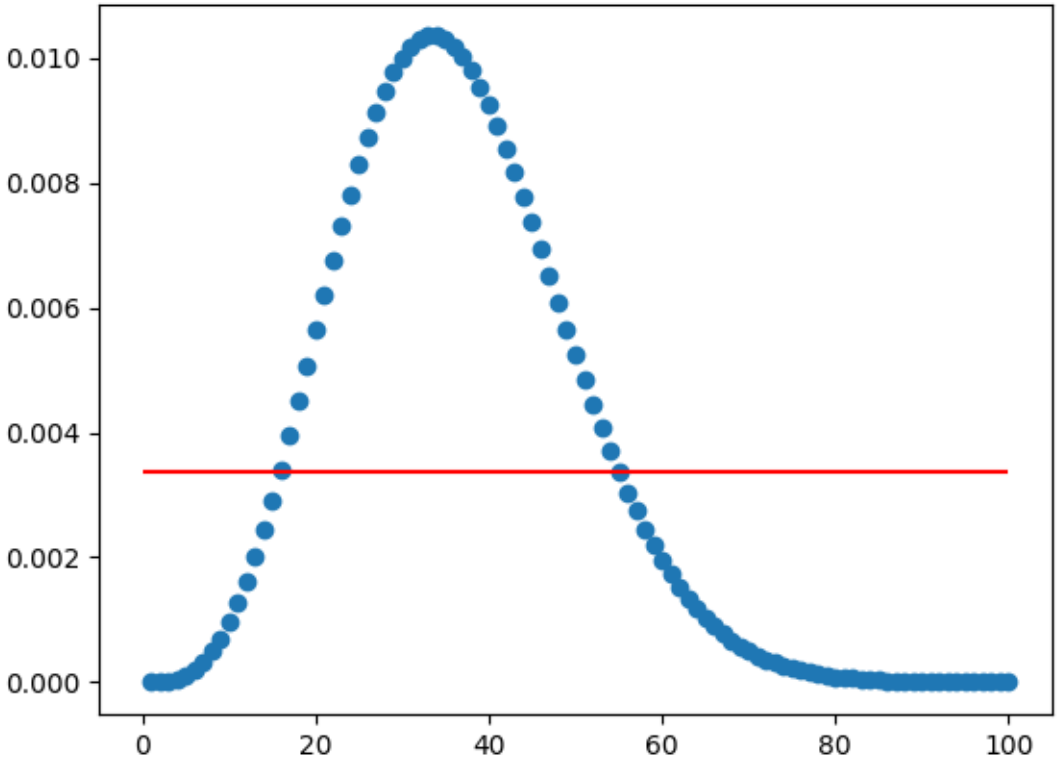


図 9: $p_1(\bar{M}, n), p_2(\bar{M}, n)$ の誤差

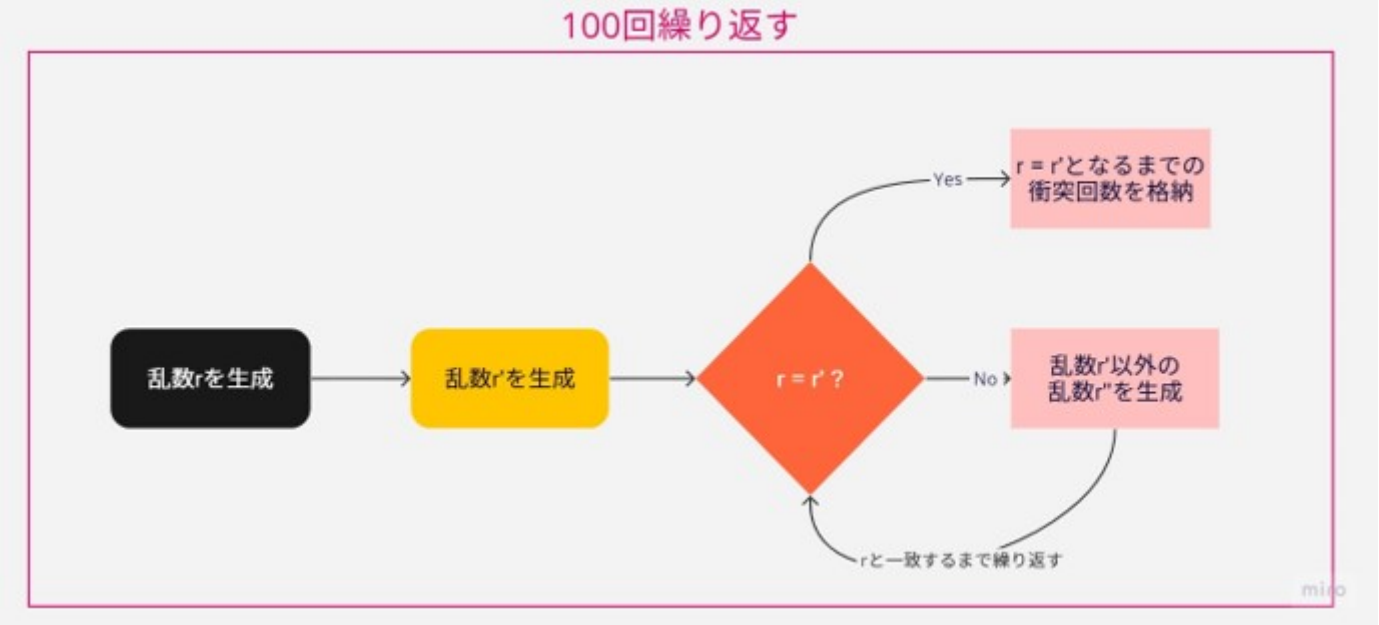


図 10: プロセス

問 2.2

N 人のグループで同じ誕生日の 1 組が存在する期待値は,

$$E[X] = \sum_{n=2}^{N} \frac{365!}{(365^n(365-n)!} \cdot \frac{n}{365} \cdot n$$

今回の PBL の参加者は $N = 43$ より,

$$E[X] = 20.235044318927464 \simeq 21$$

よって, 21 回の質問で同じ誕生日の 1 組が現れることが予想できる.

解析 5

(1-1)
具体的な操作は以下のような手順で行われれる.
これらの操作を行なった結果, 以下のような実験値をグラフに表すことができる.
(1-2)
種類の数が 10 であるとき, $p_1(\bar{M}, n), p_2(\bar{M}, n)$ は $M = 10$ として N を $[n_{min}, n_{max}]$ の範囲で動かせばよい. よって, グラフは以下のようになる
(1-3)
暗号化回数が大きくなると, 理論値と実験値の差が小さくなる傾向にある.
(option) 理論値と実験値を比較すると, 理論値の方が実験値より大きいことがわかり, 差も大きくなることがわかる.

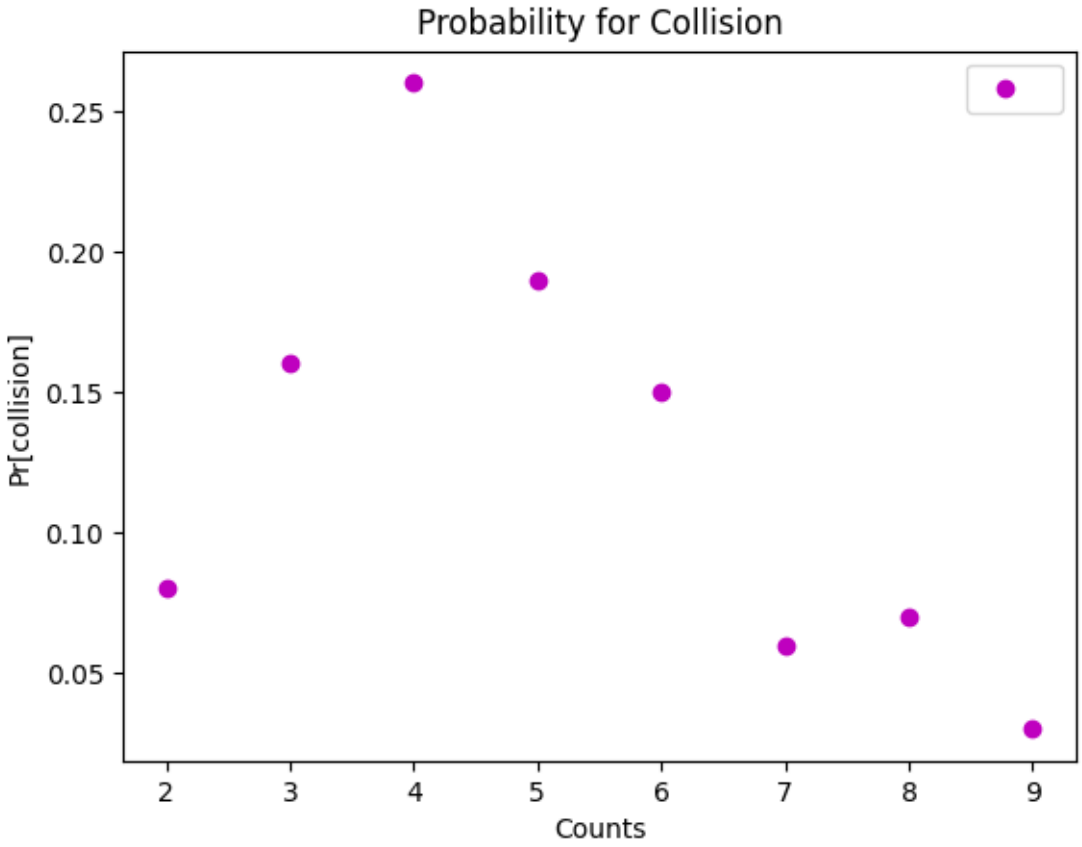


図 11: Birthday Paradox(実験値)

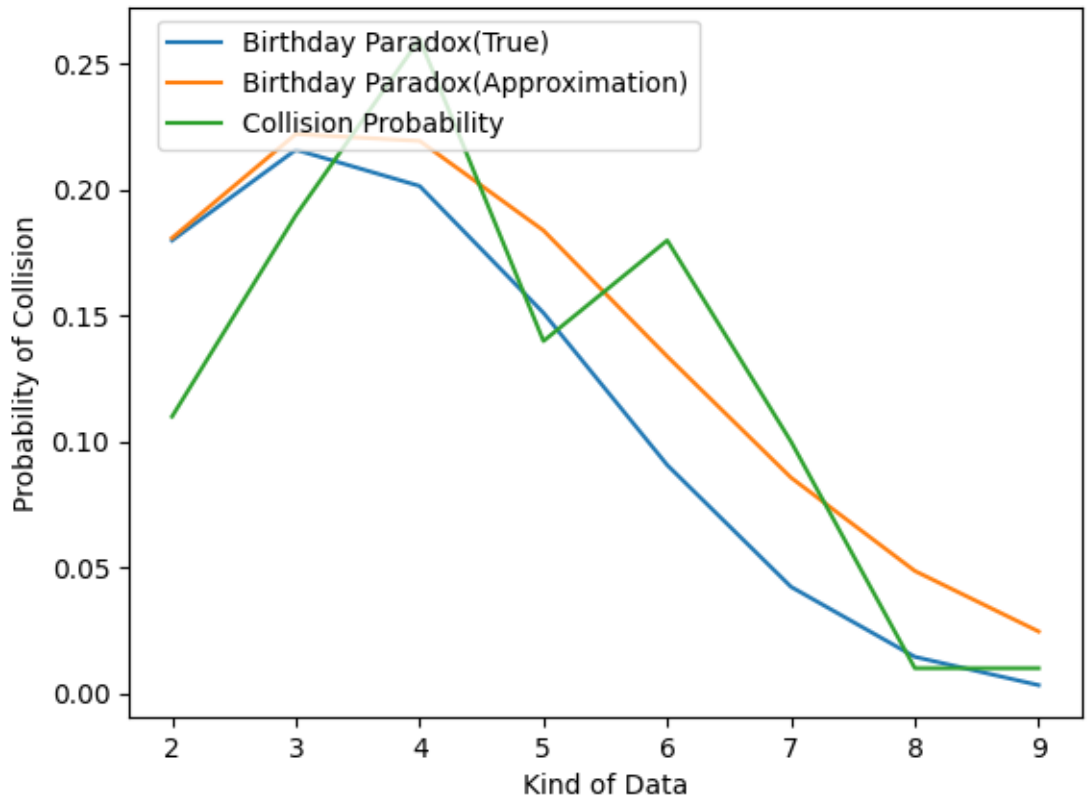


図 12: Birthday Paradox の理論値 (M=10)

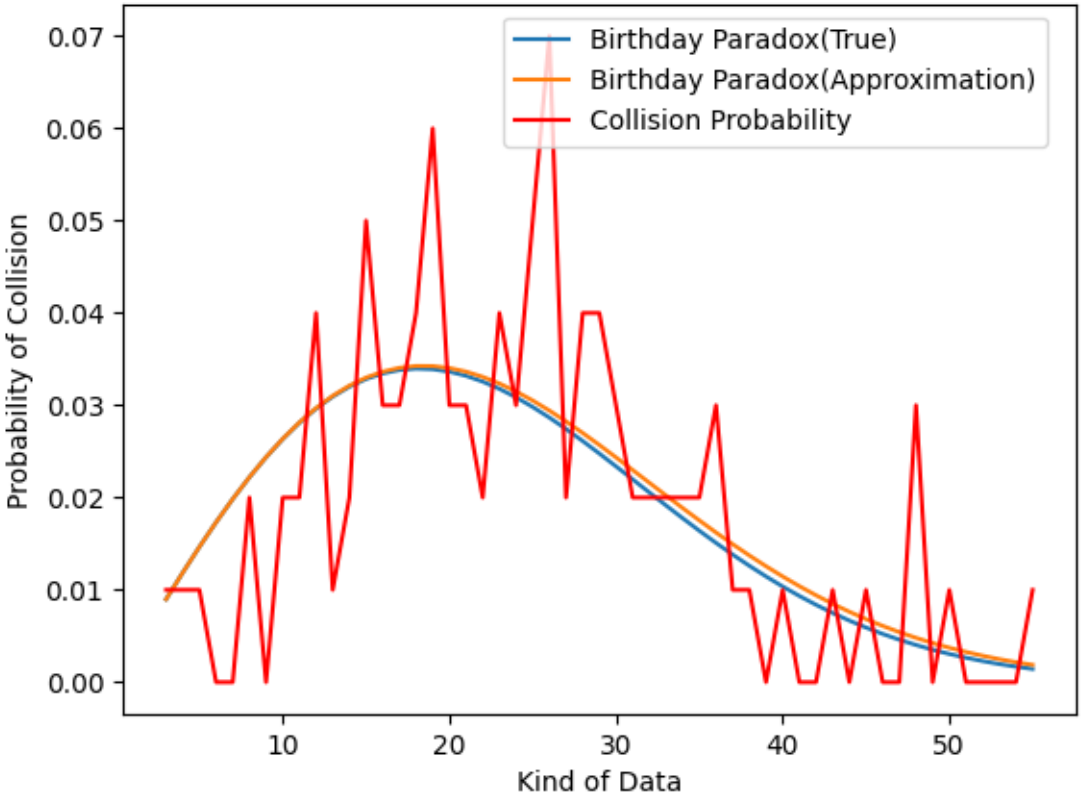


図 13: Birthday Paradox(option)

G.4 演習 3.1(担当：和泉)

G.5 演習 3.2(担当：前野)

G.6 演習 4.1(担当：nas・田川)

問 4.1

(1)

ステップ数 i	a_i	b_i	$x_i = 2^{a_i}3^{b_i} \bmod 101$
0	1	0	2
1	2	0	4
2	2	1	12
3	3	1	24
4	4	1	48
5	5	1	96
6	6	1	91
7	6	2	71
8	12	4	92
9	24	8	81
10	25	8	61
11	25	9	82
12	25	10	44
13	50	20	17
14	100	40	87
15	101	40	73
16	101	41	17

より, $x = -(a_i - a_j)/(b_i - b_j) \bmod n = -51/21 \bmod 100 = 69$.

(2)

例として $a'_0 = 5, b'_0 = 7$ を用いた場合の結果を示す.

ステップ数 i	a_i	b_i	$x_i = 2^{a_i}3^{b_i} \bmod 101$
0	5	7	92
1	10	14	81
2	11	14	61
3	11	15	82
4	11	16	44
5	22	32	17
6	44	64	87
7	45	64	73
8	45	65	17

(3)

$a_0 = 1, b_0 = 0$ を初期値とする.

ステップ数 i	a_i	b_i	$x_i = 2^{a_i}3^{b_i} \bmod 101$
0	1	0	2
1	2	0	4
2	2	1	28
3	2	2	95
4	4	4	36
5	5	4	72
6	6	4	43
7	6	5	99
8	7	5	97
9	7	6	73
10	7	7	6
11	8	7	12
12	9	7	24
13	10	7	48
14	11	7	96
15	12	7	91
16	12	8	31
17	12	9	15
18	13	9	30
19	14	9	60
20	15	9	19
21	15	10	32
22	30	20	14
23	60	40	95

表より, $2^{(60-2)}7^{(40-2)} = 2^{58+38t} = 1 \bmod 101$ となる．$58 + 38t = 0 \bmod 100$ を解くと, $t = 9, 59 \bmod 100$ であり, $2^9 = 7 \bmod 101$ より $t = 9$ が求まる.
また, $a_0 = 4, b_0 = 2$ を初期値とした場合の表を以下に示す.

ステップ数 i	a_i	b_i	$x_i = 2^{a_i}3^{b_i} \bmod 101$
0	4	2	77
1	8	4	71
2	16	8	92
3	32	16	81
4	33	16	61
5	33	17	23
6	66	34	24
7	67	34	48
8	68	34	96
9	69	34	91
10	69	35	31
11	69	36	15
12	70	36	30
13	71	36	60
14	72	36	19
15	72	37	32
16	144	74	14
17	288	148	95
18	576	296	36
19	577	296	72
20	578	296	43
21	578	297	99
22	579	297	97
23	579	298	73
24	579	299	6
25	580	299	12
26	581	299	24

問 4.2

各 i に対する $x_i, a_i, b_i, x_{2i}, a_{2i}, b_{2i}$ を以下の表に示す.

ステップ数 i	x_i	a_i	b_i	x_{2i}	a_{2i}	b_{2i}
1	16	1	0	42	1	1
2	42	1	1	8	4	2
3	14	2	1	7	16	8
4	8	4	2	12	17	9
5	17	8	4	34	18	10
6	7	16	8	36	18	12
7	36	16	9	4	20	12
8	12	17	9	7	20	14
9	4	18	9	12	21	15
10	34	18	10	34	22	16

表より, $x = -(a_{2i} - a_i)/(b_{2i} - b_i) \bmod n = -4/6 \bmod 23 = 7$.

チーム対抗課題

(1) $x = 3842632$
(2) $x = 1146732679$
(3) $x = 89471828204601$
(4) $x = 47884355645935526$
(5) $x = 1012415622044797997$

G.7 演習 4.2(担当：tony・山下)

(1) 表 2 の定数を利用して次のトランザクションを署名，検証してください．

tx = '{
"sender":
1078737038950724991323409279838172952504193756692031681555319102868429898726047259332631218501522710805950161319960604357946370755530212014702989075890345806627524984798222935409882083993759004201296992643556329268870656444575937161661407537253550975178452559499868610373295573552177182325799390537925822897389,
"receiver":
4206370338826103406715590119398417376949758809096556356897326269419305486253636357766213297361113874477665887793895764715404254728974358078527271348508076058211754628112594432006583042152971875655656066325474703174713042578517546526803538444110337426750040006850346436048140950684113891224005454945533532136 9,
"value": 10,
"previous tx hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855" }'

解説

問題文で与えられた署名関数を利用し，u, v を求める．
求めた u, v と $dsa\_sign\_verify()$ を用いて署名の検証を行う．

(2) 先ほどの送信者に $10 を送り返すトランザクションを作成. トランザクションに署名を施す. 署名を検証する.

解説

先ほどのトランザクション (tx) を参考にし，送信者と受信者を入れ替える．また，previous tx hash を tx を hash 化した値に変更する．

(3) Smart Contract of checking power(pow 関数のチェック) を考える．これは正しい入力が行われと，“success “ が出力として返る．もし失敗する場合は，gas の上限を上げて再試行する．

解答のステップ

1. ModBinary.sol ファイルを生成する．

2. スライドにあったコードを添付する.
3. ModBinary.sol ファイルをコンパイルする.
4. その後 deploy & run を行う.

以上で問い（3）は終了です.

G.8 演習 5(担当：Mathieu)

問 5.1

We start by calculating the order l:

$$2^{11} = 2048 = 1 \bmod 23$$

So the order is $l = 11$. We then choose a secret key and calculate the corresponding public key. We choose $x = 3$ We then have

$$y = g^x \bmod p = 8$$

The key pair is $(3, 8)$.

We then calculate the signature:

$$\begin{aligned} u_1 &= g^r \bmod p \\ &= 8 \\ u &= u_1 \bmod l \end{aligned}$$

To calculate v, we first need to calculate the inverse of r (be careful, we calculate the inverse modulo l). We have $3 \times 4 = 1 \bmod 11$ so $r^{-1} = 4$.

we then have

$$\begin{aligned} v &= r^{-1}(m' + xu) \bmod l \\ &= 2 \end{aligned}$$

The signature is $(u, v) = (8, 2)$.

We now verify the signature by calculating $v^{-1} = 6$ (once again modulo l).

$$\begin{aligned} u' &= g^{m' \cdot v^{-1}} \cdot y^{u \cdot v^{-1}} \\ &= 8 \end{aligned}$$

So the signature is correct.

問 5.2

We count the number of multiplications, power and inverses in Signature Generation and Verification. We consider $|p| = n$. Sign: $u = u_1 \bmod l = g^r \bmod p \bmod l \longrightarrow n/2M_p + nS_p$.

$v = r^{-1}(m + xu) \bmod l \longrightarrow I_l + 2M_l$

Verif: $u = (g^{m \cdot v^{-1}} \cdot y^{u \cdot v^{-1}} \bmod p) \bmod l \longrightarrow I_l + 2(n/2M_p + nS_p)$

演習 5.1

```
import binascii
import random
import hashlib
import math

def mod(g,p):
    return g%p

def modpower(g,k,p) :
    k = bin(k)
    y=1

    for i in range(2,len(k)):
        if(k[i] == '1'):
            y = mod(mod(y*y,p)*g,p)
        else :
            y = mod(y*y,p)
    return y


def order(g,p):
    l=1
    while modpower(g,l,p)!=1:
       l=l+1
    return l


def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    else:
        gcd, x, y = extended_gcd(b % a, a)
        return gcd, y - (b // a) * x, x

def modinv(a, p):
    g, x, y = extended_gcd(a, p)
    if g != 1:
        print('inverse does not exist')
        return
    else:
        return x % p

def elgamal_public(g,x,p):
    return modpower(g,x,p)
```

```
def elgamal_encrypt(g,y,m,r,p):
    u = modpower(g, r, p)
    c = mod(modpower(y,r,p)*m,p)

    return u,c


def elgamal_decrypt(g,x,u,c,p):
    return mod(c*modinv(modpower(u,x,p),p),p)

def shake128(m, l):
    return hashlib.shake_128(m.encode('utf-8')).hexdigest(math.ceil(l.bit_length()/8)-1)

def sign(g,p,l,r,m,x) :
    u1 = modpower(g,r,p)
    u = mod(u1,l)
    M = shake128(str(m),l)
    v = mod(modinv(r,l)*(int('0x'+str(M),16)+x*u),l)
    return u,v

def verif(u,v,m,l,p,g,y):
    M = shake128(str(m),l)
    M_int = int('0x'+str(M),16)
    A = modpower(g,M_int*modinv(v,l),p)
    B = modpower(y,u*modinv(v,l),p)
    u2 = mod(mod(A*B,p),l)
    if mod(u2,l)==mod(u,l):
        return True
    else :
        return False
```

[frame=single]

p4 = 14110875533297471160681521826395812338118458821206101844813640482696588943307945378916621823037852222856
g4 = 79207621787760038235763239269746451281552097558625763440050213854787240633084667257397421010854631623596911
l4 = 13621159230992932423699222613052123435618460878834 k4 = 31647783200376541524773537926352571815028898726 7
r4 = 1316477832003765415247735379263525718150288987267

1)

```
db = 37075081866545145912345171449964083306223454432 1
y = elgamal_public(g4,db, p4)
print('Public Key y= ' + str(y))
```

Public Key y= 116857456160049562641994494461989312437338129910496867738099983158759858021180526724378111155869

2)

```
m = 12375081111115145912345171449964033333555554444 4
```

```
r = 123456789123456789123456789123456789123456789123

u,v = sign(g4,p4,l4, r, m, db)
print('Signature: u = ' + str(u) + ', v= ' + str(v))

print(verif(u,v,m,l4,p4,g4, y))
```

Signature: u = 1198642514289996608651292588855800695584441096609, v= 2030302950016775750520487982838315135529003
True

3)

```
x = 748193544676408372584627989049207657777532261418
y = elgamal_public(g4,x, p4)
print('Public Key y= ' + str(y))

m = '今日の体温は 26 度です'
m_ta = int(binascii.hexlify(u'今日の体温は 26 度です'.encode('utf-8')),16)
u,v = sign(g4,p4,l4, r, m_ta, x)



print('Signature: u = ' + str(u) + ', v= ' + str(v))

print(verif(u,v,m_ta,l4,p4,g4, y))
```

G.9 演習 6-1 (担当：山月)

Exercise 6.1

```
p = 141108755332974711606815218263958123381184588212061018448136404826965889433079453789166
l = 1362115923099293242369922261305212343561846087883
g = 792076217877600382357632392697464512815520975586257634400502138547872406330846672573974

def  hybrid_enc_sign_gen(m, g, l, r, p, sk, pk):
    c_1,c_2= elgamal_enc(m, g, p, r, pk)
    u, v = dsa_sign_gen(m, g, l, r, p, sk)
    return [c_1,c_2],[u,v]

def hybrid_enc_sign_verify(C, sigma, sk, l, g, p, pk):
    m = elgamal_dec(C[0], C[1], p, sk)
    varify = dsa_sign_verify(m, sigma, l, g, p, pk)
    return m, varify

def Sender(pk_r):
    print("------sender------")
    m = 1237508111115145912345171449964033333555544444
    r = 123456789123456789123456789123456789123456789123
```

```
    sk_s = r
    pk_s = dsa_sign_gen_key(p, g, l, sk_s)
    C, sigma = hybrid_enc_sign_gen(m, g, l, r, p, sk_s, pk_r)
    print("cypher text=",C)
    print("sign=",sigma)
    return C, sigma, pk_s

def Reciever(C, sigma, pk_s):
    print("------reciever------")
    sk = 748193544676408372584627989049207657777532261418
    print(hybrid_enc_sign_verify(C, sigma, sk, l, g, p, pk_s))

def main():
    r_pk = 42063703388261034067155901193984173769497588090965563568973262694193054862536363
    C, sigma, pk_s = Sender(r_pk)
    Reciever(C, sigma, pk_s)

if __name__ == '__main__':
    main()

if __name__ == '__main__':
    main()
```

result
——sender——
cypher text= [13638083306981492395448350622873403679642902714619415966781263123549879718939997548161261670008607
16534243622657709394312864617360491642072752907887038150722765000032221801792272540320109996012410098985897211
sign= [11986425142899966086512925888558006955844410966 09,
10392291012645648066992631176168963375118920230 20]
——reciever——
(123750811111151459123451714499640333335555544444, True)

G.10 演習 6-2 (担当：He)

Exercise 6.2

```
def shake128(m: str, l: int) -> str:
    return hashlib.shake_128(m.encode()).hexdigest(math.ceil(l.bit_length() / 8) - 1)

def hex_to_int(h: str) -> int:
    return int(h, base=16)

def bytes_to_int(m: bytes) -> int:
    return hex_to_int(m.hex())

def str_to_int(m: str) -> int:
```

```
    return bytes_to_int(m.encode())

def int_to_str(m: int) -> str:
    return bytes.fromhex(f'{m:x}').decode()

if __name__ == '__main__':
    ta_pk = ...     # public key of TA
    pk = ...        # self public key
    sk = ...        # self secret key

    message = 'advanced'
    m = str_to_int(message)

    # ElGamal encryption
    # elgamal_enc(m, g, p, r, pk)
    r = random.randrange(1, l4)
    u = mod_binary(g4, r, p4)
    c = (mod_binary(ta_pk, r, p4) * m) % p4
    print(f'u = {u}, c = {c}')

    # DSA signature
    # dsa_sign_gen(m, g, l, r, p, sk)
    u = mod_binary(g4, r, p4) % l4
    v = inv(r, l4) * (hex_to_int(shake128(message, l4)) + sk * u) % l4
    print(f'u = {u}, v = {v}')

    # ---------

    # ElGamal decryption
    # elgamal_dec(u, c, p, sk)
    ta_u, ta_c = ...
    m = ta_c * inv(mod_binary(ta_u, sk, p4), p4) % p4
    m_str = int_to_str(m)
    print(m_str)

    # DSA verification
    # dsa_sign_verify(m, (u, v), l, g, p, pk)
    ta_u, ta_v = ...
    digest = hex_to_int(shake128(m_str, l4))
    v_inv = inv(ta_v, l4)
    u = mod_binary(g4, digest * v_inv, p4) * mod_binary(ta_y, ta_u * v_inv, p4) % p4 % l4
    assert ta_u == u
```