教養原論 数理の世界
数学とコンピュータ 2011
RSA 暗号解読のためのプログラミング

野呂 正行

神戸大学理学研究科

November 24, 2011

12/1, 12/8 **は** K501 **で実習**

実習の内容

ある Macintosh **上のソフト** (cfep) **を使って**, **公開鍵** $n(= pq)$ **が小さい場合の暗号文を解読する**.

1. **暗号文は全員別々のものを配布する**.
   紙で配布するが, **入力間違いが多いので**, **データを** web **に掲示する予定**.
2. n **を素因数分解する**.
3. p, q **から** $de \bmod (p-1)(q-1) = 1$ **を満たす** d **を作る**
4. **暗号文の各ワード** E_i **に対し** $D_i = E_i^d \bmod n$ **を計算する**.
5. D_i **から平文を復元する**.
6. **質問に答える**.
   暗号文は簡単な質問になっているはず.

cfep とは

- **初心者用数式計算ソフト**
  高山, 野呂 (神戸大学理学研究科) が作成, 授業に使用
- **インストールは** web **から**
  http://www.math.kobe-u.ac.jp/Asir/cfep/intro-ja.html
  1. **授業用ページにリンクがある**.
  2. **「**cfep **のアーカイブ」** (cfep-intel.zip) **をダウンロード**, **展開する**. (**「**Cfep/asir **超入門」も**)
     自動的に (**あるいは** cfep-intel.zip **をダブルクリックすると**), cfep-intel-20100827 **というフォルダが現れる**.
  3. cfep-intel-20100827 **をフォルダごとデスクトップに置く**.
     別フォルダに入れると cfep **が起動できない**.

cfep の起動, 終了

- **起動**
  cfep アイコン (「イノブタ」のつもり) をダブルクリック
- **終了**
  メニューバーの cfep から「cfep を終了」を選択
- **使い方は超入門を参照**
  基本 : 入力ウィンドウに式を入力して 始め を押す
  ⇒ 別ウィンドウがポップアップして, 結果が表示される

cfep でできること

- **数の計算**
  整数, **有理数**, **浮動小数**, **有限体**
- **式の計算**
  多項式の加減乗除, GCD, **因数分解**, **関数の値の計算**
- **プログラミング**
  繰り返し, **条件分岐**, **変数**, **関数 (手続き) 定義**
- **グラフ描画**
  $y = f(x)$, $f(x, y) = 0$ etc.

ごく基本的な使い方

- **式の入力, 計算**
  × は *, 割算 (分数) は /, 余りは %
  カッコは () のみ
  計算式の末尾に ; (セミコロン) を必ず書く.

式の入力方法

$$2 \times (3 + 5^4) \quad \Rightarrow \quad \texttt{2*(3+5\^{}4);}$$

$$\{(2 + \frac{2}{3} \times 4 + \frac{1}{3}\} \times 2 + 5 \quad \Rightarrow \quad \texttt{((2+2/3)*4+1/3)*2+5;}$$

$$10 \bmod 4 \quad \Rightarrow \quad \texttt{10\%4;}$$

- **変数は大文字アルファベット (X, Y など)**
- **= は ← (右辺の値を左辺に上書き)**

代入の例

```
X=2;
X=2*X;
```

繰り返し — for 文

for 文の例 – 10 の階乗の計算

```
F=1;
for ( I = 1; I <= 10; I++ ) {
    F = F*I;
}
```

この例では, F **には** $1, 1\times2, 1\times2\times3, \ldots,$ **が入る**

$\Rightarrow$ for **文を抜けたあとは** $1\times2\times\cdots\times10 = 10!$ **が入る**

条件のいろいろ

`I<10`	10 **より小さければ真**
`I>10`	10 **より大きければ真**
`I<=10`	10 **以下なら真**
`I>=10`	10 **以上なら真**
`I==10`	10 **と等しければ真**
`I!=10`	10 **と等しくなければ真**

条件分岐 — if 文

if 文 : 条件が真のときのみ実行

if 文

```
if ( (X%Y) == 0 ) {
   print(X,0); print("は",0)
   print(Y,0); print("で割り切れる");
}
```

if-else 文 : 条件の真偽で異なる仕事をする

if-else 文

```
if ( (X%2) == 0 ) {
   print(X,0); print("は偶数");
} else {
   print(X,0); print("は奇数");
}
```

プログラム例：素因数分解

N を $\sqrt{N}$ を越えない最大整数までの奇数で割ってみる

P1:素因数分解

```
N=...;
B=isqrt(N);
for ( I=3; I <= B; I = I+2 ) {
  R = N%I;
  if ( R == 0 ) {
     P = I;
     print(P);
  }
}
```

. . . には, N の値をコピーペーストする.

⇒ 実行すると P には最大の因数が入っている.

P1 が 2 つ以上の因子を印字したり, しばらく待っても何も
でないときは, N の入力ミスである.

n' の計算

ここから先はから新規ウィンドウで実行する.

P1 を何度も実行すると時間がかかるので.

P1 を実行して, N の因子 P を求めたら, Q および $N1 = (P-1)(Q-1)$ が計算できる. ($N1$ は n' のつもり)

P2:q, n' の計算

```
N=...;
P=...;
Q=N/P;
N1=(P-1)*(Q-1);
```

... には, N の値および, P1 で得た P の値をコピーペーストする.

$65537d + kn' = 1$ を満たす d の計算

P3: $kn' \bmod 65537 = 1$ を満たす k を探す

```
N1=(P-1)*(Q-1);
for ( I = 1; I <= 65536; I++ ) {
    R = (I*N1) % 65537;
    if ( R == 1 ) {
        K = I;
        print(K);
    }
}
```

P4: $d = (1 - kn')/65537$ から d を計算

```
D=(1-K*N1)/65537;
```

P5: $0 < d < n'$ となるように n' をいくつか足す

```
D=D%N1;
```

実行例

$n = 482783626412797$ **とする.**

n を素因数分解して d を計算する

P1 **により** $p = 19157771$

P2 **により** $q = n/p = 25200407$, $n' = 482783582054620$

P3 **により** $k = 42093$

P4 **により** $d = -310081470305707$

P5 **により** $d = 172702111748913$

検算 : $65537 \times d = 23444 \times n' + 1$

復号：各データを d 乗して mod$n$

$a^d \bmod n$ の計算：繰り返し 2 乗法で実行

1. $d = (b_{m-1} \cdots b_0)_2$ と 2 進表記する.
2. $a_i = a^{2^i} \bmod n \ (i = 0, 1, \ldots, m-1)$ を計算する.
3. $b_i = 1$ となる i についてだけ a_i をかけて mod$n$ したものが $a^d \bmod n$ となる.

正当性:

$d = (b_{m-1}b_{m-1} \cdots b_0)_2 = b_{m-1}2^{m-1} + \cdots + b_0 2^0$ とすると

$$\begin{aligned} a^d &= a^{b_{m-1}2^{m-1}+\cdots+b_0 2^0} \\ &= (a^{2^{m-1}})^{b_{m-1}} \times \cdots \times (a^{2^0})^{b_0} \\ &\equiv (a_{m-1}^{b_{m-1}} \times \cdots \times a_0^{b_0}) \bmod n \end{aligned}$$

$b_i = 0$ または 1 より, 最後の式は $b_i = 1$ となる i についてだけ a_i をかけたものになる.

繰り返し 2 乗法のプログラム

P6: $a^d \bmod n$ の計算

```
A=...;
R=1;
AI = A;
for ( B = D; B > 0; ) {
    BI = B%2;
    if ( BI == 1 ) R = (R*AI)%N;
    AI = (AI*AI) % N;
    B = (B-BI)/2;
}
print(R);
```

i 回繰り返したとき, BI は d の下から i ビット目の値

AI は, a から始めて $a_i = a^{2^i} \bmod n$ となっている

実際の手順 : '...' にプリントの暗号文の数字を一ついれて P6 を実行すると, 数字が一つ印字される $\Rightarrow$ これを P7 に渡す.

数から文字列への変換

課題の平文は, 1 **文字が** 3 **バイトで表現されている**.
(Macintosh **の標準の文字コードが** UTF-8 **のため**)
この平文の 2 **文字** (6 **バイト**) **を一つの数字として暗号化したものが**, **課題の暗号文である**.

例

あいうえおか $\Rightarrow$ (**あい**) (**うえ**) (**おか**) $\Rightarrow e_1\ e_2\ e_3$

よって, **復号は次の手順による**

1. **復号した数字を** 6 **バイトに分割する**
2. **上** 3 **バイト を文字に変換**, **表示**
3. **下** 3 **バイト を文字に変換**, **表示**

数を 6 バイトに分割する

P7: 数を 6 バイトに分割

ここは自分で考えてみよう

```
B1=...; B2=...; ... ; B6 = ...;
```

$B1, \ldots, B6$ **は** $P6$ **で得た数** R **の上から** 1 **バイト目**, . . . 6 **バイト目である. これは**

$$R = B1 \cdot 256^5 + B2 \cdot 256^4 + \cdots + B5 \cdot 256 + B6$$

と 256 **進展開すれば得られる. あとは** $(B1, B2, B3)$, $(B4, B5, B6)$ **を文字に変換すればよい.**

もちろん, 各 $B1, \ldots, B6$ **は** 0 **以上** 255 **以下である.**

問題 : どうやって 256 **進展開するか?**

ヒント : $B6$ **は** R **を** 256 **で割った余りである.**

商は, 余りを引いて, 除数で割れば得られる.

3 バイトを 1 文字に変換

$(B1, B2, B3)$ という 3 バイトを 1 文字に変換するには, 組込み関数 asciitostr を使う

P8 : 3 バイトを 1 文字に変換

```
if (B1 >= 224 && B1 <= 239 )
   asciitostr([B1,B2,B3]);
else
   print("invalid code");
```

if 文の意味は, 1 バイト目が 224 以上 239 以下でない場合, 対応する文字がないことを意味する.
⇒ バイトへの分割が間違っているか, そもそも復号結果が違っているかどちらかなので修正が必要

cfep 使用上の注意

- プログラムの保存, 読み込み
  - 書いたプログラムは保存できる.
    ファイル → 保存
  - 保存してあるプログラムの読み込み
    ファイル → 開く
  - 保存ファイル名に注意
    上書きして大事なファイルを消さないように, ファイル → 別名で保存 も使う.
- 計算中のまま何も表示されない場合
  止 ボタンを押し, 再起動 ボタンを押して復活させる.
  (マニュアル 20, 21 ページ参照)
  再起動するサーバー名は Risa/Asir である.

cfep 使用上の注意

- **素因数分解は一度だけにする.**
  P1 と P2 以降は別ファイルにするとよい.
  P2 の先頭に, N の値, P1 で得た P の値を N=...;,
  P=...; と必ず書くこと.
  P3 以降は P2 に書き足していけばよい.
- **プログラム上のよくある間違い**
  - 式の最後にセミコロン ; がない
  - かっこ (), [] が対応していない
  - セミコロン ; がコロン : になっている
  - 全角文字が入っている
  - 数字の打ち間違い

  プログラムの間違いで実行できない場合, おおよその場所が示される.
  その近辺をよく調べること.

C プログラミングの経験者への注意

Risa/Asir **ドリル参照**

- **ユーザによる関数定義**
  def **関数名** (**引数リスト**){ **文 文** $\cdots$ **文** return **式**; }
  で関数定義を行い, その後でこの関数を呼び出して計算を行うようにできる.
  例: P7, P8 **を関数にしておけば便利**
- **リストの利用**
  $[e_1, e_2, \ldots, e_m]$ (e_i **は任意の式**) **によりリストが生成できる.**
  リスト L **の** I **番目の要素は** $L[I]$ **でアクセスできる**
  例: 入力データをリストにしておき, 1 ワードに対する解読処理を, リストに対する for **文で一度に処理できる.**