暗号化についての理論とプログラミング研究

2年5組　速水　諒介
2年4組　澤近　栄作
指導者　教諭　松浦　哲人

１　課題研究の背景

今日、情報時代となりインターネットを用いた情報伝達が頻繁に行われるようになった。その中には私たちの個人情報も多く含まれており、そういった情報を守るためにも暗号技術がより重要により身近になっている。そこで暗号化技術を学ぶことで、役に立つ知識を身に付けたい。

２　仮説

初心者でも簡単な暗号化プログラムやその解読プログラムを作成することはできるのではないか。実際に自分たちで Ruby 言語を使ってプログラムを作ることで確かめる。

３　暗号の使われ方

(1)　昔は戦争や政治的なことと密接に関わっていた。

(2)　現在は情報化社会となり、一般の人でも暗号に触れることが多くなった。

４　暗号方式の紹介

暗号は英語で"code"または"cipher"という。この 2 つの表現には違いがある。

・code……コードネームという言葉もあるように"code"とはある一定の文字列などを塊として別の文字列に変える方式である。

・cipher……一文字ずつ文字を置き換える方式。我々の研究では主にこちらの方式を取り扱う。

(1)　カエサル暗号

この暗号はとても古くからある暗号である。暗号化の方法は、暗号化する文がアルファベットであれば、一定の数だけアルファベットの順番をずらして表記するというものである。暗号化も簡単であるが､解読する時もアルファベットであれば元の文の候補が 26 しかないため、解読が容易であり、情報を守る技術としては実用的ではない。

(2)　対応表方式

文字の対応表を作成し、その対応表に沿って暗号化する方法である。この方法では、カエサル暗号のように対応がアルファベット順になるということはないため、候補数が増えて解読が難しくなっている。しかし、この暗号では文字の対応が一対一であるため、文字の出現頻度を調べると、ある程度元の文字が推測できるため、あまり実用的ではない。

(3)　公開鍵暗号方式

複数の相手と暗号のやり取りをする際に、普通の暗号では、コード表や暗号化鍵が増えて複雑になってしまう。そこでこの方式が作られた。この方式では公開鍵と秘密鍵の二つの鍵を用意する。公開鍵では暗号化できるが、復号の難しい鍵であり、復号するには秘密鍵が必要である。この暗号を使うことで、それぞれの秘密鍵の管理だけを厳重にすればいいので、利便性と安全性が増した。

(4)　量子暗号

光子には横波のようにふるまう性質や、一つの光子は二つに分割できないという性質がある。量子暗号を使えば暗号化の鍵を安全に共有することができる。初めに二つの位相の波だけを通す枠を用意し、それぞれの波に 0，1 の番号を付ける。それと同じで通す光の異なる枠を用意する。まずたがいに暗号を送りあう二人の片方が、適当な順番で枠を使い、0 または 1 の光を送る。もう片方が適当な順番で枠を使い、光を測定する。すると正しく測定したもの

と間違ったものが出てくる。そこで互いが使った枠を伝え合い正しく伝わったものだけを鍵とする。この方法では、第三者が測定しても正しく観測できたとは限らず、数字を送ることは一度だけであるため、正確な鍵を入手することはできない。

５　暗号についての最近の動向

人々の安全かつ快適で便利な生活を支えるうえで、情報セキュリティおよび、プライバシーを守るニーズでますます高まりつつある。

ハードウェア規模やメモリ等のリソースの限られたデバイスや省電力が求められる機器に適した暗号技術、軽量暗号技術が工夫されてきている。例えば、バッテリー式の体内埋め込み医療機器などで、低消費電力の方がバッテリーが長持ちするため、取り替え回数が少なくて済むことになり、利用者の負担が軽減される。

６　研究に使用したプログラミング言語について

(1)　Ruby 言語の特徴

Ruby はオブジェクト指向スクリプト言語であり、スクリプト言語が用いられてきた領域でのオブジェクト指向プログラミングを実現する。Ruby においては整数や文字列なども含めてデータ型は全てがオブジェクトであり、純粋なオブジェクト指向言語といえる。

```
i = 32
while i < 150 do
 print(i ," ",i.chr + "  ")
  i = i + 1
end
```

図1:プログラム例

(2)　プログラム例

アスキーコード値と文字の対応を表示するプログラム例を図１に示した。

(3)　暗号研究のための Ruby 言語の利点

Ruby は文字列の扱いが得意なプログラミング言語である。したがって今回扱う暗号化プログラムの作成を簡単に行うことができる。

また解読プログラム作成時においても正規表現などを用いれば特定の文字列を見つけることが簡単であり、とても扱いやすい。とくに Ruby は USB メモリ内に動作環境を作成することが可能であり、パソコンがあればどこでも取り組める。これはプログラミング初心者の私たちにとっては、とても大きな利点である。

７　Ruby 言語を用いた暗号処理プログラム開発

(1)　カエサル暗号

ａ．暗号化プログラム(zurasu_kai.rb　22 行)

元の文字 aiueo　　暗号化の鍵 103

① 暗号化する文字のアスキーコード値に変換する。

② 変換したコード値に鍵を足す。

③ もし足した後の値が 122 より大きければ、122 より小さくなるまで 26 引くことを繰り返す。

④ 繰り返した後のコード値を文字に変換する。

⑤ 終わり。

ｂ．解読プログラム(kaidoku1-2.rb　25 行)

暗号化した文字　ifmmp

鍵候補（1、2、……25、26）

```
# カエサル暗号の作成プログラム
moji = "aiueo" ←元の文字
  kay = 103     ←暗号化の鍵
  kazu = moji.bytes.to_a    ←①
  suuji = []
  word = []
  kazu.each do |i|
   i = i + kay          ←②
   while i > 122 do  ←③
    i = i - 26
   end
   while i < 97 do
    i = i + 26
   end
   suuji << i
  end
  suuji.each do |n|
   word << n.chr        ←④
  end
  word.each do |h|
   print(h)
  end                   ←⑤
```

図2:カエサル暗号の作成プログラム

① 解読する文字のアスキーコード値に変換する。
② 変換したコード値に鍵候補を足す。
③ もし足した後のコード値が 122 より大きければ、122 より小さくなるまで 26 ひくことを繰り返す。
④ 繰り返した後のコード値を文字に変換する。
⑤ 変換後の文字を出力する。
⑥ 鍵候補の値を増やしながら 2～5 を繰り返す。
⑦ 出力された文字を見て正しいと思われる文字列を探す。
⑧ 終わり。

```
# カエサル暗号の解読プログラム
word1 = "ifmmp"
key = 0
code2 = []
code3 = []
code1 = word1.bytes.to_a    ←①

while key < 123 do
 code1.each do |i|    ←②
  i = i + key
  if i > 123 then    ←③
   i = i - 90
  end
  code2 << i
 end
 key = key + 1
 code2.each do |m|    ←④
  code3 << m.chr
 end
 code3.each do |n|    ←⑤
  print(n)
 end
 print("¥n" , key)
 code2 = []
 code3 = []
end                  ←⑥
```

図3:カエサル暗号の解読プログラム

(2) 対応表暗号

a 対応表暗号の暗号化プログラム

(taiouhyou_angou2.3.rb 54 行)

あらかじめ対応表と暗号化する文字を入れたファイルを用意しておく。

① 対応表を読み込み文字の対応を作る。
② ファイルを読み込み、文字をアスキーコード値に変換する。
③ 対応に沿ってアスキーコード値を変える。
④ 変えた後のアスキーコード値を文字に変える。
⑤ 別のファイルに書き込む。
⑥ 終わり。

```
# 対応表暗号作成プログラム
file1 = open("mojiretu.txt")
# hash の作成
taiou2 = {}
n = 32
file1.each_line do |i|
 taiou2[n] = i.rstrip        ←①
 n = n + 1
end
file1.close
file2 = open("sample.txt")
moji1
  = file2.read.bytes.to_a    ←②
angoubun = []
moji1.each do |i|
 angoubun << taiou2[i]       ←③
end
angoubun.each do |i|
 print(i)
end
file2.close
file_bun1
  = open("sample2.txt","w")
  file3 = open("sample.txt")
  print("¥n")
  moji2 = []
  while line = file3.gets
   moji2 = line.bytes.to_a ←④
   angoubun2 = []
   moji2.each do |i|
    angoubun2 << taiou2[i]
   end
   angoubun2.each do |i|
    file_bun1.print(i)      ←⑤
    print(i)
   end
   file_bun1.print("¥n")
   print("¥n")
end
file3.close
file_bun1.close
print(moji1.size)
```

図4:対応表暗号作成プログラム

b　対応表暗号の解読プログラム(taiouhyou_angou2.2_kaidoku2.rb　63 行)

① 解読する文字をアスキーコード値に変換する。
② すべての組み合わせの対応表のパターンで元の文候補を作る。
③ ひとつ元の文候補を作るごとによくつかわれる単語の数を数える。
④ もし④の数が一回前の数より多ければこの元の文候補をキープする前の文を廃棄する。
⑤ ②～④を繰り返す。
⑥ キープした元の文候補を出力する。
⑦ 終わり。

８　結論

(1)　仮説の検証

プログラミングを学び始めて一年たたなくても簡単な暗号化プログラムとその解読プログラムを作ることができた。まだ基本的な暗号しか扱っておらず作ったプログラムも改善の余地が多いが研究を続けていけば、より発展的なプログラムを作ることができると考える。

(2)　これまでの達成程度

まだ公開鍵暗号などの新しい暗号までは取り組めていないが、基本的な暗号の暗号化プログラムや解読プログラムを作ることで暗号の基礎となる方法を学ぶことができた。この研究によって、あ分野の数学的な考え方やプログラミング技術など、これから役立つ知識と技能を身に着けることができた。

(3)　今後の課題

公開鍵暗号などにも挑んでいき、より高度な暗号について理解を深めていったり、これまでに作った解読プログラムを見直してより効率的なプログラムを考えたりして、より暗号についての理論を学んでいきたい。

参考文献

・暗号解読(上・下)　サイモン・シン著 青木薫訳　新潮社発行
・Ruby 入門　www.rubylife.jp/ini/