平成22年度 卒業論文

論文題目

現金に替わる電子マネーの実装

神奈川大学 工学部 電子情報フロンティア学科

学籍番号　200702894

大城　翔太

指導担当者　木下宏揚　教授

目 次

第 1 章　序論 **3**

第 2 章　基礎知識 **4**

2.1 暗号技術 4

2.1.1 共通鍵暗号方式 4

2.1.2 公開鍵暗号方式 4

2.2 離散対数問題 9

2.2.1 合同式 9

2.2.2 素数 10

2.2.3 離散対数問題 10

2.3 匿名通信路 10

2.4 RSA 暗号を使用した電子マネー 11

2.5 離散対数問題を使用した電子マネー 15

2.5.1 決済処理の方法 15

第 3 章　提案システム **16**

3.1 提案するシステム 16

3.1.1 Java によるプログラム 16

3.1.2 提案するにあたって 17

3.2 提案したプログラムの動作 18

3.3 提案したプログラムの評価 22

3.3.1 安全性と計算量 22

第 4 章　結論 **23**

謝辞 **24**

参 考 文 献 **25**

質疑応答 **26**

付録 **27**

図 目 次

2.1 MyRSA2.java の実行結果 7
2.2 Money.java の実行結果 14

3.1 取引の流れ 17
3.2 Money3.java の実行結果 21

第1章

序論

近年，日本では電車の乗車券，コンビニエンスストアでの支払いなどの場面で電子的な決済が身近になってきている [1]．それに伴い多くの電子マネーが発行されていて，2010 年現在，JR 東日本が発行する Suica と株式会社パスモの発行する PASMO, ビットワレットが発行する Edy の 3 つだけで発行枚数は約一億万枚で [2][3], その他多種多様な電子マネーが普及している．

しかし，発行会社が違う電子マネー間の相互利用はほとんど出来ない為 [4]，電車の運賃は鉄道会社の発行する交通系電子マネー，買い物はクレジットカード会社の発行する電子マネーで決済するなど，利用するシーンごとに電子マネーを使い分けているのが現状で, さらにその多くがクローズドループ型の電子マネーであり，一度で一回の決済しかできない為, 現金の替わりになるまでには至っていない．

研究室では以前より電子マネーについての研究がおこなわれており，通貨の発行量を調整するために中央銀行を電子マネーの発行機関にする方法や，電子マネーの決済において離散対数問題を用いるなど，現金の特徴を継承しつつ電子マネーの利点を持たせ，安全性を高める方法などが考案されている．

本研究では，現金と代替可能な電子マネーを実装する為に必要なシステムの構成を検討し,Java での記述を前提としたプロトコル部分の実装を行う．

第2章

基礎知識

2.1　暗号技術

2.1.1　共通鍵暗号方式

共通鍵暗号方式とは，メッセージの暗号化と複合化で同じ鍵を使う暗号方式．メッセージの送り手と受け手で秘密に鍵を共有する．扱いが簡単であり処理速度が速い半面，相手毎に固有の鍵を作成しなければならない為, あらかじめ安全な方法で相手に鍵を渡さなければならないことから限られた特定の相手とのやり取りに向いている．

共通鍵暗号方式の代表的なものに,DES 方式がある．

2.1.2　公開鍵暗号方式

公開鍵暗号方式 [5] とは，メッセージを暗号化する鍵（公開鍵）と複合する鍵（複合鍵）の 2 つの鍵を使用する．2 つの鍵には数学的関係があり 2 つの鍵のうちの一方の鍵で暗号化したデータを複合化できるのはもう一方の鍵を使用した場合に限られる．また公開鍵から秘密鍵を解くことが困難であることが数学的の証明されている．公開鍵暗号方式において秘密鍵は第三者に開示しなければ保証される．公開鍵暗号の代表的なものには RSA 暗号 [6] がある．

次項に RSA 暗号を Java で記述したサンプルプログラムと実行結果を示す.

```
// RSA サンプルプログラム MyRSA2.java

import java.math.BigInteger;
import java.util.Random;
import java.io.*;
public class MyRSA2
{
 // 大素数のビット数の指定
   public static final int SIZE = 64;
 // メッセージの暗号化メソッド
   public static BigInteger encodeMessage(BigInteger N,
                                          BigInteger e)
                           throws Exception
  {
    BigInteger message;
    do
    {
      System.out.println("N より小さいメッセージを入力して下さい。");
      InputStreamReader in = new InputStreamReader(System.in);
      BufferedReader br = new BufferedReader(in);
      String input = br.readLine();
      message = new BigInteger(input);
    }
    while ((message.compareTo(N) != -1)
        || (message.gcd(N).compareTo(BigInteger.valueOf(1)) != 0));
    System.out.println();
    BigInteger ctext = message.modPow(e, N);
    System.out.println("暗号文: " + ctext.toString());
    System.out.println();
    return ctext;
  }
```

```
// メインメソッド
public static void main (String[] argv) throws Exception
{
  BigInteger p, q, e, d, N, P, ctext, ptext;
  String input;
  // 異なる大素数 p, q の生成
  p = new BigInteger(SIZE, 10, new Random());
  do
  {
    q = new BigInteger(SIZE, 10, new Random());
  }
  while (q.compareTo(p) == 0);
  N = p.multiply(q);
  // P = (p-1)(q-1) の計算
  P = p.subtract(BigInteger.valueOf(1));
  P = P.multiply(q.subtract(BigInteger.valueOf(1)));
  // e の生成
  do
  {
    e = new BigInteger(2*SIZE, new Random());
  }
  while ((e.compareTo(P) != -1)
      || (e.gcd(P).compareTo(BigInteger.valueOf(1)) != 0));
  // N, e の公開
  System.out.println("N: " + N.toString());
  System.out.println("e: " + e.toString());
  System.out.println();
  // N, e による暗号化
  ctext = encodeMessage(N, e);
  // d = e mod P の逆元の計算
  d = e.modInverse(P);
```

```
    System.out.println("d: " + d.toString());
    // メッセージの復号化
    ptext = ctext.modPow(d, N);
    System.out.println("平文: " + ptext.toString());
 }
}
```

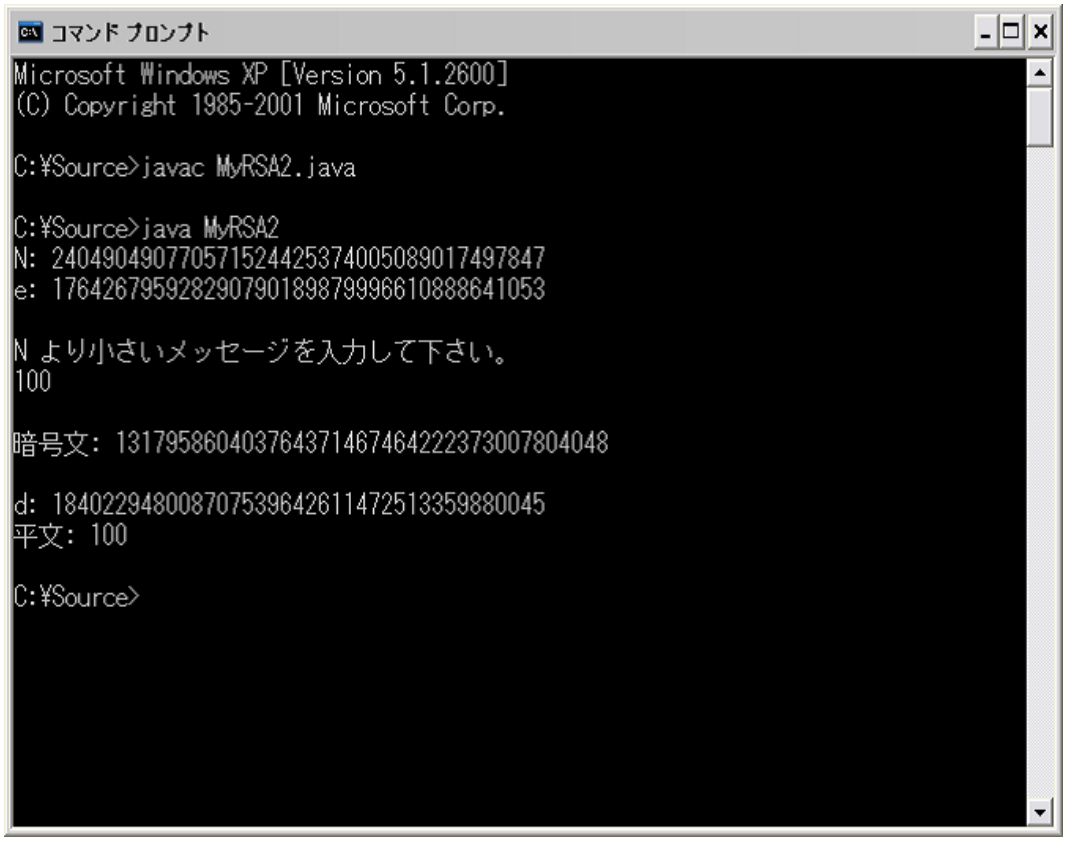

図 2.1 MyRSA2.java の実行結果

RSA 暗号は素因数分解の困難性に基づいた暗号で，上記のプログラム中では，公開鍵である N と e により入力した値 m が

$$c = m^e (mod n)$$

の式により暗号化するものである．
RSA プログラムにおいて，もっとも計算を要するのが，暗号化と復号化における合同式のべき乗計算である．その計算量の為，公開鍵から秘密鍵を見つけるのが事実上不可能となるわけである．

2.2 離散対数問題

2.2.1 合同式

整数 a,b の差 a-b が 0 または整数 N の倍数であるとき

$$a \equiv b(modN)$$

と書き a,b とは N を法として合同であるといいこのような関係式を合同式 [7] と呼ぶ．a を N で割った余りが r のとき $a = qN + r$ と書けるので,(ただし q は商) 明らかに $a \equiv r(modN)$ である．

(modN) の集合として 0,⋯,N-1 の整数の集合を Z_N で表す.

$$Z_N = 0, \cdots, N-1$$

$a \equiv bmodN$ かつ $a \in Z_N$ のとき

$$a \equiv (b(modN))$$

と書く．b が整数のとき a は b を N で割った余りとなる．1; ⋯;N- 1 のうち N との最大公約数が 1（つまり N と互いに素）である整数の集合を Z_N で表す. よって

$$Z_N = x|1 \leq x \leq N-1, gcd(x, N) = 1$$

ここで gcd は（great common divisor）を表す.$-x(modN)$ について $(N-1)-(-1) = N$ などで $N-1 = -1(modN)$ である. 一般に $1 \leq x \leq N-1$ に対し次式が成り立つ．

$$N - x \equiv -xmodN$$

2.2.2 素数

2以上の整数pが1とp自身以外に約数を持たないときpをいう．フェルマーの定理についてpが素数のとき $Z_p = 1, 2, \cdots, p-1$ である．pが素数のとき任意の $a \in Z_P$ に対し次式が成り立つ．

$$a^{p-1} \equiv 1 mod p$$

位数とはpが素数のとき $a \in Z_P$ に対し $a^x \equiv 1 mod p$ となる最小の正整数xをaの位数といい $ord_p(a)$ で表す．フェルマーの定理により $0 \leqq ord_p(a) \leqq p-1$ である．

原子元についてpを素数のとき $ord_p(g) = p-1$ となるpを Z_p の原始根（あるいは原始元）という．2乗3乗を求めていったとき(p-1)乗して初めて1になる数が原始根である．pを素数gを Z_p の原始根とする．このとき $i = 0, \cdots, p-2$ に対し $a_i = g^i mod p$ とおくと

$$a_0, a_1, a_2, \cdots a_{p-2} = 1, 2, \cdots, p-1$$

が成り立つ.

2.2.3 離散対数問題

gが Z_p の原始根のとき任意の $a \in Z_p$ に対し $a = g^x mod p$ となるxが必ず存在するということが分かる．このようなxをaの離散対数という．ここでaの離散対数を求める問題を離散対数問題 [8] という．すなわち離散対数問題とは素数 p, Z_p の原子元g及び $a \in Z_p$ が与えられたとき

$$a = g^x mod p$$

となる $x \in 0, 1, \cdots, p-2$ を求めよという問題である．

xから $a = g^x mod p$ を計算する事は簡単である．しかしpが大きい時aからxを求める事は困難である．離散対数問題を解く効率的なアルゴリズムは見つかっておらず,暗号への応用が出来る．

2.3 匿名通信路

匿名通信路 [9] の利用目的として悪意を持つ人に個人の情報が渡らないようにする自衛の意味と，インターネット上での行為から個人が特定されないようにするという２点が挙げられる．匿名性を実現するための方法として,信頼できる第三者機関を通じて通信する方法が知られている．複数の送信者からのメッセージをセンターで混ぜ合わせることによってメッセージの匿名性を保証するものやノード間で無造作にコピーを繰り返しオリジナルを不明確にすることで匿名性を実現しているものがある．

2.4 RSA 暗号を使用した電子マネー

RSA 暗号を用いて入力した金額を暗号化し，再び復号化するプログラムおよびその結果を下に示す.
これは,暗号化したデータを送受信する事によって安全性を確保している.

```
// RSA を使用した電子マネー  Money.java
import java.math.BigInteger;
import java.util.Random;
import java.io.*;
public class Money
{
 // 大素数のビット数の指定
   public static final int SIZE = 448;
 // メッセージの暗号化メソッド
   public static BigInteger encodeMessage(BigInteger N,
                                          BigInteger e)
                          throws Exception
  {
    BigInteger message;
    do
    {
        System.out.println("金額を入力してください");
        InputStreamReader in = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(in);
        String input = br.readLine();
        message = new BigInteger(input);
    System.out.println("入力した金額は" + message + "です。");
    }
        while ((message.compareTo(N) != -1)
            || (message.gcd(N).compareTo(BigInteger.valueOf(1)) != 0));
        System.out.println();
        BigInteger ctext = message.modPow(e, N);
        System.out.println("暗号文: " + ctext.toString());
        System.out.println();
        return ctext;
  }

  // メインメソッド
  public static void main (String[] argv) throws Exception
  {
        BigInteger p, q, e, d, d1, d2, N, P,ctext, ctext1, ctext2,
        ptext, ptext1, ptext2,user1, user2, t1, t2;
        String input;
```

```
        user1 = new BigInteger("100000");
        user2 = new BigInteger("100000");
        // 異なる大素数 p, q の生成
        p = new BigInteger(SIZE, 10, new Random());
        do
        {
       q = new BigInteger(SIZE, 10, new Random());
        }
        while (q.compareTo(p) == 0);
        N = p.multiply(q);
        // P = (p-1)(q-1) の計算
        P = p.subtract(BigInteger.valueOf(1));
        P = P.multiply(q.subtract(BigInteger.valueOf(1)));
        // e の生成
        do
        {
         e = new BigInteger(2*SIZE, new Random());
        }
        while ((e.compareTo(P) != -1)
            || (e.gcd(P).compareTo(BigInteger.valueOf(1)) != 0));
        // N, e の公開
        System.out.println("N: " + N.toString());
        System.out.println("e: " + e.toString());
        System.out.println();
        // N, e による暗号化
        ctext = encodeMessage(N, e);
        // d = e mod P の逆元の計算
        d = e.modInverse(P);
        System.out.println("d: " + d.toString());
        // メッセージの復号化
        ptext = ctext.modPow(d, N);
        System.out.println("復号化した金額の値: " + ptext.toString());
    // 復号した数値を用いて計算し、再び復号化してユーザーに送信
    t1 = user1.subtract(ptext);  //user1 の所持金から ptext の数値を減算
    t2 = user2.add(ptext);  //user2 の所持金に ptext の数値を加算
    System.out.println("user1 の残高:" + t1.toString());
    System.out.println("user2 の残高:" + t2.toString());

while ((t1.compareTo(N) != -1)
         || (t1.gcd(N).compareTo(BigInteger.valueOf(1)) != 0));
        System.out.println();
        ctext1 = t1.modPow(e, N);
        System.out.println("暗号文: " + ctext1.toString());
```

```
        System.out.println();
        ptext1 = ctext1.modPow(d, N);
        System.out.println("復号したuser1の残高: " + ptext1.toString());

    while ((t2.compareTo(N) != -1)
            || (t2.gcd(N).compareTo(BigInteger.valueOf(1)) != 0));
        System.out.println();
        ctext2 = t2.modPow(e, N);
        System.out.println("暗号文: " + ctext2.toString());
        System.out.println();
        ptext2 = ctext2.modPow(d, N);
        System.out.println("復号したuser2の残高: " + ptext2.toString());

  }
}
```

図 2.2 Money.java の実行結果

2.5 離散対数問題を使用した電子マネー

電子マネーの構造は，S_x を 64bit の金額,R_x を 448bit の乱数とすると

$$M_x = f(S_x, R_x) = 2^{448} S_x + R_x$$

となり，となりデータベースに蓄積される電子マネー x の認証子 D_x は原始根を g とすると

$$D_x = g^{M_x} mod\ n$$

となる．離散対数問題を使用する利点はデータベースが電子マネーの合計金額を管理しているので, 二重支払いなどの問題が起こらず D_x から M_x を求めることが難しく, データベースに対してユーザーの匿名性を保つことができる事である.

2.5.1 決済処理の方法

データベースの情報は，ユーザーの電子マネーを A，受領者の電子マネーを B, 大きな素数を n,n の原始根を g とすると次のようになる．

・取引前

$$D_{A1} = g^{M_{A1}} mod\ n$$

$$D_{B1} = g^{M_{B1}} mod\ n$$

・取引後

$$D_{A2} = g^{M_{A2}} mod\ n$$

$$D_{B2} = g^{M_{B2}} mod\ n$$

となる．
取引の際に誤差が無かったかを確認するには，以下の式を用いる．

$$g^{M_{A1}} g^{M_{B1}} = g^{M_{A2}} g^{M_{B2}} mod\ n$$

第3章

提案システム

3.1　提案するシステム

3.1.1　Java によるプログラム

電子マネーを実装するにあたって、以下の事柄が重要である.

- 利用者の決済情報が第三者に知られないよう, 暗号化した情報から暗号化する前の情報を計算する事を難しくする
- 利用者が利用し易いよう, ウェブ上に実装し易い言語を選択する
- 通常 Java が取り扱える 32bit を超える大きな桁の数字を取り扱えるようにする事

このことから暗号化には離散対数問題を使用する事で安全性を高め, プログラムを書く言語は Java アプレットなどの機能が備わっている Java[10] を選んだ.
そして, 桁の大きな整数が扱えるよう, 大整数計算ライブラリを用いてプログラムを記述する.

3.1.2 提案するにあたって

今回提案するプログラムは，主に商品を購入するユーザーとそれを受け取る受領者の取引に焦点を当てて作成する．

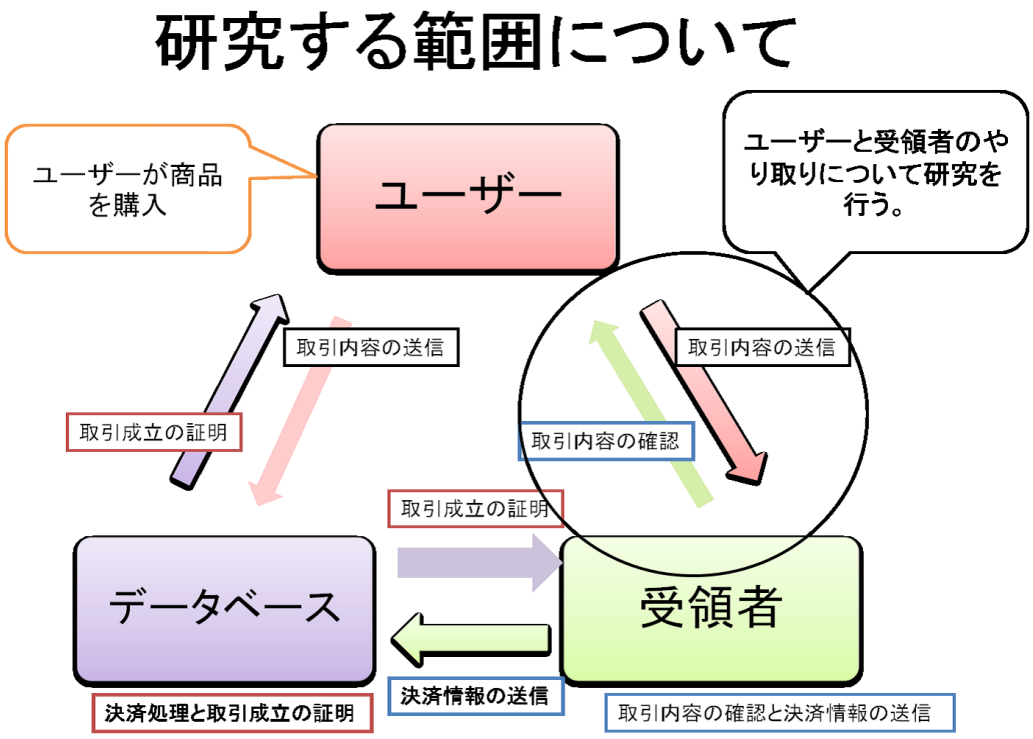

図 3.1 取引の流れ

3.2 提案したプログラムの動作

今回, システム作成に使用したソフトは,

- 開発キット：Sun Microsystems 社の Java Development Kit(JDK)
- テキストエディタ：TeraPad

を使用.
本研究で作成したプログラムは, ユーザーと受領者の最初の所持金を 100000 として, まず 448bit の乱数 p を生成し，それを用いてユーザーと受領者の所持金 Ma1,Mb1, データベースに保存させる Da1,Db1 を計算する。
次に決済処理したい金額を入力させて，Mx を計算し、それに基づいてユーザーと受領者の所持金を Ma1,Mb1 の値から増減させ,Da2,Db2 として暗号化させて，最後に実行結果に誤差が無かったか確認し，決済完了となる.
次項に作成したプログラムと実行結果の図を記述する.

```
// 離散対数プログラム  Money3.java
import java.math.BigInteger;
import java.util.Random;
import java.io.*;
public class Money3
{
 // ビット数の指定
   public static final int SIZE = 448;
 // メッセージの暗号化メソッド
   public static BigInteger encodeMessage(BigInteger t)
                          throws Exception
  {
    BigInteger message;
    do
    {
     System.out.println("金額を入力してください");
     InputStreamReader in = new InputStreamReader(System.in);
     BufferedReader br = new BufferedReader(in);
     String input = br.readLine();
     message = new BigInteger(input);
        System.out.println("入力した金額は" + message + "です。");
    }
     while (message.compareTo(t) != -1);
     System.out.println();
     BigInteger Mx = message.multiply(t);
     System.out.println("Mx: " + Mx.toString());
     System.out.println();
     return Mx;
   }
  // メインメソッド
  public static void main (String[] argv) throws Exception
  {
        BigInteger z, y, g, p, t, n, u1, u2,Sa1,Sb1,Ma1,
        Ma2, Mb1, Mb2, Mx, Da1,Da2, Db1, Db2, Dab1, Dab2,D1,D2;
        int r = 448;
        int x;
        String input;
        n = new BigInteger("976046411"); //大きな素数 n
        u1 = new BigInteger("100000"); //user1 の最初の所持金を決める
        u2 = new BigInteger("100000");   //user2 の最初の所持金を決める
        y = new BigInteger("2"); //2
        g = new BigInteger("2"); //n の原始根 g
        t = y.pow(r); //2^448
        Sa1 = t.multiply(u1); //2^448*user1 の所持金
        Sb1 = t.multiply(u2); //2^448*user2 の所持金
        // 448bit の乱数 p の生成
        p = new BigInteger(SIZE, new Random());
        System.out.println("p:" + p.toString());
        // t による暗号化
        Mx = encodeMessage(t);
//Ma1,Mb1=2^448*(Sa1,Sb1)+p の計算および Da1,Db1=g^Ma1,Mb1 modn の計算
        Ma1 = Sa1.add(p);
        Mb1 = Sb1.add(p);
        Da1 = g.modPow(Ma1,n);
        Db1 = g.modPow(Mb1,n);
        System.out.println("Da1:" + Da1.toString());
```

```
        System.out.println("Db1:" + Db1.toString());
    // Mx から Ma2=Ma1-Mx,Mb2=Mb1+Mx と Da2,Db2=g^Ma2,Mb2 modn を計算
        Ma2 = Ma1.subtract(Mx);
        Mb2 = Mb1.add(Mx);
        Da2 = g.modPow(Ma2,n);
        Db2 = g.modPow(Mb2,n);
        System.out.println("Da2:" + Da2.toString());
        System.out.println("Db2:" + Db2.toString());
        // Da1,Db1,Da2,Db2 から Dab1=Da1*Db1,Dab2=Da2*Db2 を計算
        Dab1 = Da1.multiply(Db1).mod(n);
        Dab2 = Da2.multiply(Db2).mod(n);
        // D1 と D2 から計算に誤りが無いか確認
        x = Dab1.compareTo(Dab2);
        if(x == 0){
            System.out.println("決済が完了しました");
        }
        else{
            System.out.println("決済が正しく行われていません");
        }
    }
}
```

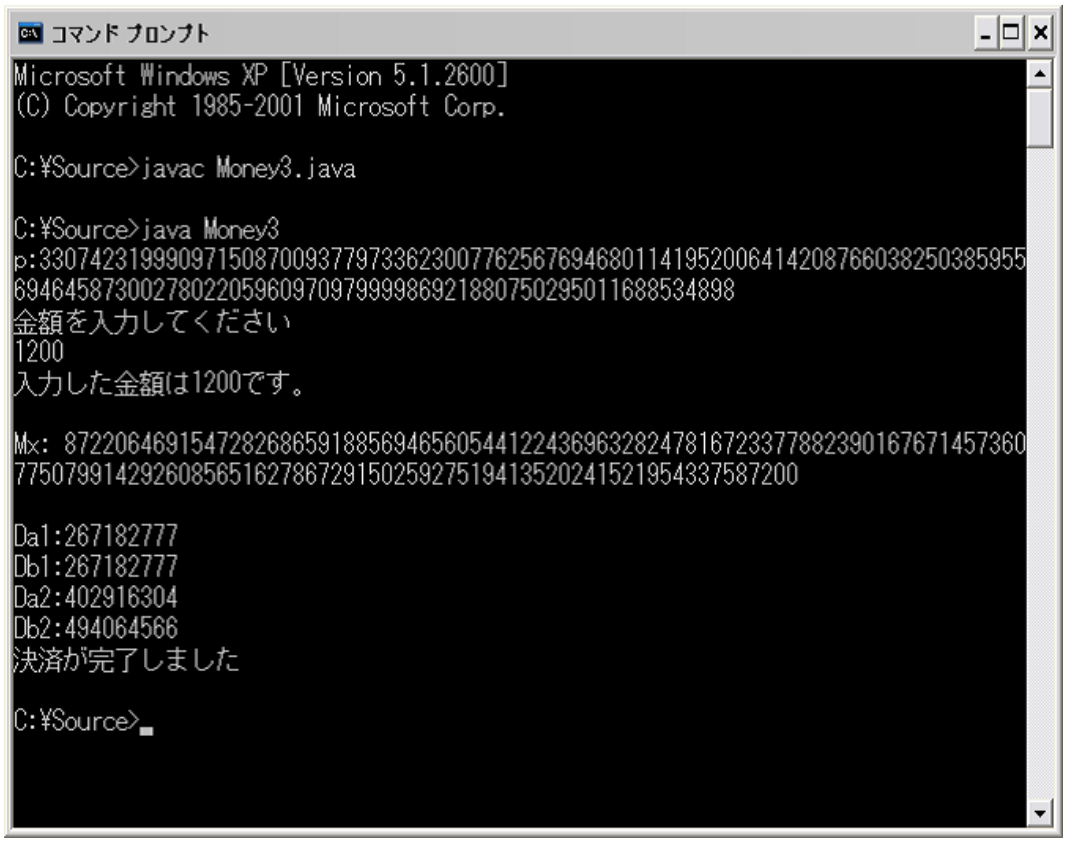

図 3.2 Money3.java の実行結果

3.3 提案したプログラムの評価

3.3.1 安全性と計算量

暗号化した数値の安全性は,D_x から M_x を求める事が非常に困難という離散対数問題の特性により, プライバシーが高められている. 計算量については,S_x のビット数を n とすると,n が 64bit である今回の計算については,O 記法を用いて O(1) で表せる. n が 512bit を大きく超える場合については,O(log n) で表される.

第4章

結論

本研究では離散対数問題を用いて，金額を暗号化するプロトコル実行部分の実装を行った．これにより，もし第三者に暗号化した情報を盗まれても，離散対数問題の複雑さから元の金額を判別する事が非常に困難な為，ユーザーの残高や決済金額が分からず,安全性が高いと言える．
しかし,今回作成したプログラムの他に,ユーザーを管理するシステム,決済情報を処理するデータベース,匿名通信路等を用いたネットワークシステムを構築する必要がある.

さらに，プログラムの言語にJavaを選んだ理由として，インターネット上で利用できるJavaアプレットが使える点があるが，それを利用してユーザにとって使いやすいインタフェースにしていく事が必要である．

謝辞

本研究を行なうにあたり，終始熱心に御指導していただいた木下宏揚教授と鈴木一弘助手に心から感謝致します．また，公私にわたり良き研究生活を送らせていただいた木下研究室の方々に感謝致します．

2011 年 2 月
大城　翔太

参考文献

[1] 岡田仁志:”電子マネーがわかる”，日経文庫 (2008)

[2] ”Wikipedia 〜Edy〜”
http://ja.wikipedia.org/wiki/Edy

[3] 東日本旅客鉄道株式会社 プレスリリース：”2010/11/16”
http://www.jreast.co.jp/press/2010/20101108.pdf

[4] 岩田昭男:”電子マネー最終戦争”, 洋泉社 (2007)

[5] Douglas R.Stinson(櫻井幸一訳):”暗号理論の基礎”，共立出版 (1996)

[6] 赤間世紀：”Java による暗号理論入門”，工学社 (2007)

[7] 草刈圭一朗：”離散数学” http://ocw.nagoya-u.jp/files/16/eng_c02_09.pdf

[8] 高木貞治：”初等関数論講義第 2 版”，共立出版株式会社（1994）

[9] 谷口：”通信ネットワークセキュリティ”，日本実業出版社（1998）

[10] 高橋真奈:”やさしい Java”, ソフトバンクパブリッシング (2000)

質疑応答

能登准教授
Q:M_x はなぜ 512bit なんですか.
離散対数を用いて計算する上で、bit 数が少ないと D_x から M_x が計算されやすくなる恐れがあり,512bit としています。
なお、金額が 64bit, 乱数が 448bit なのは金額に 64bit あれば十分であろうという考えからです. 通貨のインフレーションにより桁数が大きくなる場合はこの部分で対応します.

豊嶋教授
Q:離散対数を用いた暗号化がなぜ安全だと言えるのでしょうか.
離散対数問題を解く効率的なアルゴリズムが見つかっていない事からです. しかし, 桁数によってはスーパーコンピュータなど演算処理能力の高いコンピュータを用いれば暗号を読み取られてしまう可能性があります.

付録

Javaプログラムの実行手順
1. コマンドプロンプトなどを開く（JDKなどをインストールする）
2.javac ○○.javaでコンパイル
3.java ○○　でプログラム実行

Javaプログラムの保管場所

//Sleepy/smb/raid/archive/semi/2010 oshiro/Source Javaソースコード

MyRSA1.java RSA暗号プログラム（１）

MyRSA2.java RSA暗号プログラム（２）

Money.java RSA暗号を用いた電子マネー

Money3.java 離散対数問題を用いた電子マネー