暗号化技術 (2)：公開鍵暗号

1　目的

暗号化プログラムの作成を通じて，公開鍵暗号についての理解を深める．

2　公開鍵暗号方式

2.1　概要

1977 年，マサチューセッツ工科大学の Ronald Rivest，Adi Shamir 及び Len Adleman は，暗号化と復号化に異なる鍵を用いる方式を提案した．この方式は**公開鍵暗号方式（Public Key Crypto System）**と名付けられた．2 種類の鍵は，**公開鍵（Public Key）**及び**秘密鍵（Private Key）**と呼ばれる．ユーザは公開鍵を通信相手に渡しておき，秘密鍵は自分で保持する．また，公開鍵暗号に属する暗号化アルゴリズムは，次の要件を満たす必要がある．

1. ユーザ A の秘密鍵で暗号化したメッセージは，A の公開鍵でのみ復号化が可能である．
2. ユーザ A の公開鍵で暗号化したメッセージは，A の秘密鍵でのみ復号化が可能である．
3. 一方の鍵の値からもう一方の鍵の値を推定することは困難である．

図 1 に公開鍵暗号を用いたメッセージ交換の様子を示す．ここでは、ユーザ A がユーザ B からメッセージを受け取るための仕組を示している．A は予め，他のユーザに対して公開鍵を渡しておく．ユーザ B は，A の公開鍵を使用して原文を暗号化し，A に送信する．暗号文は，A の秘密鍵でしか復号化することができない．従って，第 3 者が不正に公開鍵を取得しても，転送中のメッセージを解読することはできない．これに対し，共通鍵暗号方式では，共通鍵が第 3 者に漏えいすると，メッセージを解読される恐れがある．

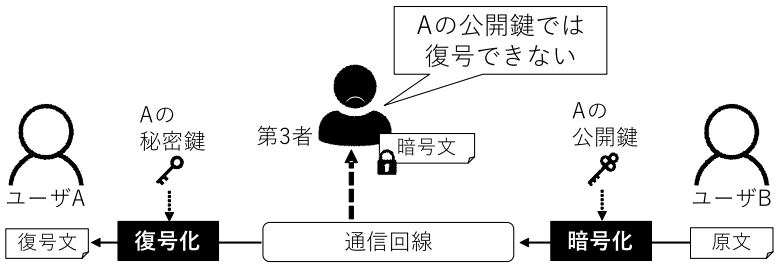

図 1　公開鍵暗号を用いたメッセージ交換

2.2　ディジタル署名との組合せ

図 2 に，公開鍵暗号と**ディジタル署名（Digital Signature）**を組み合せたメッセージ交換の様子を示す．図 1 とは反対向き，即ち，ユーザ A からユーザ B へとメッセージが転送される過程を例にとる．ディジタル署名の実現方法としては，公開鍵暗号とメッセージダイジェスト（高度なハッシュ関数）を組合せるものが主流となっている．最初に，A はメッセージの内容に基づいてメッセージダイジェスト（ハッシュ値）を生成し，この値を A の秘密鍵で暗号化する．暗号化されたメッセージダイジェストがディジタル署名になる．

メッセージを受信した B は，A の公開鍵を使用してディジタル署名を復号化する．そして，受信したメッセージからメッセージダイジェストを算出し，受信したものと比較する．メッセージが途中で改ざんされた場合，2 つのダイジェストは一致しない．また，A の秘密鍵を持たないユーザがメッセージを送信した場合，A の公開鍵で復号化されたダイジェストは，そのユーザが生成したダイジェストの値とは一致しない．従って，受信したダイジェストの値とメッセージから算出したそれが一致しない場合，B は受信したメッセージが不正である（送信者が偽装されているか，通信途中でメッセージが改ざんされている）と判断できる．

ディジタル署名は誰でも作成可能であるが，電子商取引においては、ディジタル署名そのものが不正に作成／取得されたものではないことが保証されなければならない．このため，外部の認証機関が生成したディジタル署名，すなわち**ディジタル証明書（Digital Certificate）**を利用するケースが増えている．ディジタル証明書にはディジタル署名だけでなく，こ

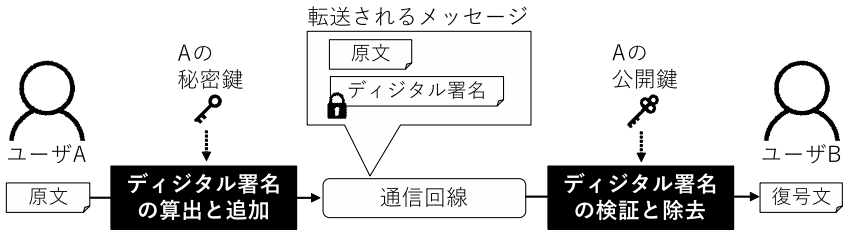

図 2　ディジタル署名

れを発行した認証機関の情報も含まれている．受信者はこの認証機関に問い合わせることによって，ディジタル署名を作成したユーザを確認することができる．

なお，メッセージの本文を共通鍵暗号で暗号化し，これに対して公開鍵暗号を使用したディジタル署名を追加する方式も存在する．ただし，この場合には，ディジタル署名は受信者の公開鍵によって暗号化され，受信者の秘密鍵によって復号化されるのが一般的になっている．

3　RSA 暗号

RSA 暗号（RSA Crypto System）は，公開鍵暗号を提案した 3 名によって発表された，最初の公開鍵暗号アルゴリズムである．3 名の名前の頭文字をとって，RSA 暗号と呼ばれている．RSA 暗号は，「大きな数の素因数分解にはとても長い時間がかかる」という性質を利用している．

まず，RSA 暗号の原理について解説する．暗号化する文字を m としたとき，m の最大値より大きな値 n を選定する．ただし，n は素数 p, q を乗算したものとするため，次式のように表すことができる．

$$n = p \times q \tag{1}$$

このとき，数論におけるフェルマーの定理の変形から，

$$m = m^{k \times \phi(n)+1} \bmod n \tag{2}$$

を導くことができる．式 (2) の中で，$\phi(n)$ は 1 以上 n 未満の数のうち，互いに素であるもの（すなわち n 未満の素数および 1）の個数を示し，k は任意の自然数である．また，“mod” は剰余を求める演算子である．ここで，$k\phi(n)+1$ を互いに異なる素数 e, $d(e \neq d)$ の積と考え，

$$k \times \phi(n) + 1 = e \times d \tag{3}$$

とおき，式 (2) を次のように変形する．

$$m = m^{e \times d} \bmod n \tag{4}$$

式 (4) より，次の 2 式を導くことができる．

$$c = m^e \bmod n \tag{5}$$
$$m = c^d \bmod n \tag{6}$$

式 (5) は暗号化の処理，式 (6) は復号化の処理に相当する．両式の中で，c は暗号化された文字に相当する．そして，e と n のペアが公開鍵、d と n のペアが秘密鍵となる．公開鍵と秘密鍵を決定する手順は付録 A に掲載している．

式 (4) 及び (5) より，RSA 暗号の暗号化と復号化のアルゴリズムはまったく同じである．また，e を使用して暗号化された文字は，d を使用しないと復号化することができない．同様に，d を使用して暗号化された文字は，e でのみ復号化することが可能である．

次に，素因数分解との関連について解説する．第 3 者が不正に公開鍵（e と n）を得たとする．暗号文を解読するためには d の値が必要になるが，その前に $\phi(n)$ の値を見つけなければならない．この $\phi(n)$ は

$$\phi(n) = (p-1)(q-1) \tag{7}$$

として求められるが，十分に大きな数 n を素因数分解して p と q を得るには，膨大な時間が必要である．最近では，n の値として 1,024 ビット（10 進数表現で約 300 桁）あるいは 2,048 ビット（10 進数表現で約 600 桁）の数値が使用されている．また，素数 p, q, e, d は 100〜300 桁程度の数になる．

4 RSA 暗号の実験（その 1）

4.1 公開鍵と秘密鍵の準備

今回の実験では，鍵などには小さな数を使用する．ここでは，$p = 17$，$q = 23$，$n = 391$ とする．これらの値に基づき，$e = 13$，$d = 149$ とする．

4.2 暗号化プログラムの準備

まず，暗号化プログラムを作成する．次のように入力して，暗号化プログラムを実験用ディレクトリにコピーする．

```
$ cd ~/cel-B
$ cp sample-programs/rsa_encrypt.c .
```

コピーしたプログラムには，各自で追加すべき部分（文）がいくつか用意されている．テキストエディタを用いて，必要な部分を追加する．

次に、GCC を使用して次のようにコンパイルを行う．コンパイルが正常に終了すると、rsa_encrypt という実行可能ファイルが生成される．

```
$ gcc -o rsa_encrypt rsa_encrypt.c
```

4.3 原文の準備

テキストエディタを使用して，半角英字と半角空白から成る文章を入力する．入力した原文を rsa_sample.txt という名称で、プログラムと同じディレクトリに保存する．

4.4 暗号化プログラムの実行

下記のように入力して，暗号化プログラムを実行する．実行時に鍵の入力を促されるので，公開鍵の値を入力する．実行後には，rsa_sample.txt の内容が暗号化され，rsa_encrypted.txt というファイルに保存される．

```
$ ./rsa_encrypt rsa_sample.txt rsa_encrypted.txt
```

4.5 暗号文の表示

下記のように入力して、暗号文の内容を画面に表示させる．暗号化されていることを確認すること．

```
$ more rsa_encrypted.txt
```

4.6 復号化プログラムの準備

続いて，復号化プログラムを作成する．次のように入力して，復号化プログラムを実験用ディレクトリにコピーする．RSA 暗号では暗号化と復号化にまったく同じアルゴリズムを使用するが，原文と暗号文のデータ量が異なる．そのため，暗号化と復号化の各プログラムでは，ファイルを読み書きする部分に相違がある．

```
$ cp sample-programs/rsa_decrypt.c .
```

コピーしたプログラムには，各自で追加すべき部分（文）がいくつか用意されている．テキストエディタを用いて，必要な部分を追加する．

次に、GCC を使用して次のようにコンパイルを行う．コンパイルが正常に終了すると、rsa_decrypt という実行可能ファイルが生成される．

```
$ gcc -o rsa_decrypt rsa_decrypt.c
```

4.7 復号化プログラムの実行

公開鍵暗号では暗号化と復号化のアルゴリズムがまったく同じであるため，暗号化と同じプログラムを使用すれば復号化が可能である．実行時に鍵の入力を促されるので，秘密鍵の値を入力する．実行後には rsa_encrypted.txt の内容が復号化され，rsa_decrypted.txt というファイルに保存される．

```
$ ./rsa_decrypt rsa_encrypted.txt rsa_decrypted.txt
```

4.8 復号文の確認

下記のように入力して、復号文の内容を画面に表示させる．原文が復元されているかどうかを確認する．

```
$ more rsa_decrypted.txt
```

4.9 鍵の効力の確認

復号化のプログラムを再度実行し，今度は正しい秘密鍵とは異なる値を入力する．生成された復号文の内容を画面に表示させ，復号文と原文とが異なることを確認する．

4.10 公開鍵暗号の要件の確認

今度は原文を秘密鍵で暗号化し，暗号文を公開鍵で復号化できることを確認すること．

5 RSA 暗号の実験（その 2）

5.1 原文の準備

下記のように入力して，画像ファイルを実験用ディレクトリにコピーする．

```
$ cp sample-programs/images/clover.png .
$ cp sample-programs/images/usagi.png .
```

5.2 原文の確認

適当なソフトウェアを使用して，clover.png の内容を画面に表示させる．

5.3 暗号化プログラムの実行

下記のように入力して，暗号化プログラムを実行する．実行時に鍵の入力を促されるので，公開鍵の値を入力する．実行後には clover.png の内容が暗号化され，clover_encrypted.png というファイルに保存される．

```
$ ./rsa_encrypt clover.png clover_encrypted.png
```

5.4 復号化プログラムの実行

下記のように入力して，画像ファイルの復号化を行う．実行時に鍵の入力を促されるので，秘密鍵の値を入力する．実行後には clover_encrypted.png の内容が復号化され，clover_decrypted.png というファイルに保存される．

```
$ ./rsa_decrypt clover_encrypted.png clover_decrypted.png
```

5.5 復号文の確認

clover_decrypted.png の内容を画面に表示させ、原文と同じ画像であることを確認する．原文と復号文のファイルサイズが同じかどうかも確認すること．以下のように入力すると良い．

```
$ ls -l clover_*.png
```

5.6 鍵の効力の確認

下記のように入力して，画像ファイル clover_encrypted.png に対して復号化のプログラムを実行し，今度は正しい秘密鍵とは異なる値を入力する．

```
$ ./rsa_decript clover_encrypted.png clover_decrypted2.png
```

生成された画像ファイルの内容を画面に表示させ，正しく復号化されていないことを確認する．

時間があれば，usagi.png についても，5.3 節以降の手順に従って暗号化と復号化を行うこと．

6　報告内容

レポートには，以下の内容を記載すること．

6.1　プログラムの実行結果

プログラムの実行結果，および鍵の効力を確認した結果（4.4〜4.10 節までの実行結果，および 5.3〜5.6 節までの実行結果）を掲載すること．なお，5.5 節および 5.6 節で表示させる画像については追加提出物にて確認するため，レポートには掲載しなくて良い．端末に出力された結果のみ掲載すること．

6.2　課題

(1) $n = 15$，$e = 3$，$d = 3$ としたとき，6 以上で 8 以下の整数 m について，いずれの m に対しても式 (5) 及び (6) が成立することを確認せよ．

(2) 今回の実験で用いたサンプルプログラムでは，暗号文のデータ量は原文の 2 倍になる．このように，RSA 暗号では，正しく暗号化と復号化がなされるような公開鍵と復号鍵を作成すると，暗号文のデータ量（ビット数）は原文のデータ量（ビット数）よりも大きくなってしまう．この理由について説明せよ．

7　追加提出物

レポートに加え，完成させたプログラムのソースコード (`rsa_encrypt.c`, `rsa_decrypt.c`)，実験（その 1）での動作検証に用いた原文 (`rsa_sample.txt`)，および実験（その 2）で生成されたファイル (`clover_encrypted.png`, `clover_decrypted.png`, `clover_decrypted2.png`) をレポート提出先ディレクトリに提出すること．`usagi.png` に対する実験を行った場合は，その際に生成されたファイルも提出すること．

参考文献

[1] ウィリアム・スターリングス，「暗号とネットワークセキュリティ—理論と実際」，ピアソン・エデュケーション，2001 年．
[2] 宮地充子，「情報セキュリティ」，オーム社，2003 年．

第 1〜4 回 提出物の確認

今回の実験まで完了した時点で，第 1〜4 回分の成果物として，以下のファイルが提出用ディレクトリに提出されることになる．漏れなく提出されているか各自で確認すること．第 1 レポート提出期限となった時点で提出漏れがあった場合は減点対象となるので注意すること．

- `crc.c`
- `caesar_encrypt.c caesar_decrypt.c caesar_sample.txt`
- `original_encrypt.c original_decrypt.c`
- `rsa_encrypt.c rsa_decrypt.c rsa_sample.txt`
- `clover_encrypted.png clover_decrypted.png clover_decrypted2.png`
- (`usagi_encrypted.png usagi_decrypted.png usagi_decrypted2.png`)：実験を行った場合のみ．

提出漏れの有無は，提出物確認用スクリプトを用いると良い．以下のように実行する．

```
$ cd ~/cel-B
$ ./submit-check.sh

何回目のレポート分について確認しますか？ [0-3]: 1
第 1 レポートとあわせて提出が必要なファイルについてチェックします。

 crc.c ... ok.
 ...（以下，すべてに対して ok. となっていれば良い）
```

付録 A　公開鍵と暗号鍵の決定方法

A.1　原理のそのままの方法

1. 基本となる素数 p と q を決定する．
2. p と q を乗算して n を求める．
3. $(p-1)(q-1)$ に対して互いに素となる自然数 e をひとつ選択する．
4. 次式を満たすような自然数をひとつ選択して d とする（ただし、k は任意の自然数）．

$$e \times d = k(p-1)(q-1) + 1 \tag{8}$$

A.2　よく利用される方法

前節で示した原理のそのままの手法では，3. 及び 4. の演算を実装しにくいので、下記の手法がよく利用されている．

1. 基本となる素数 p と q を決定する．
2. p と q を乗算して n を求める．
3. $\phi(n) = (p-1)(q-1)$ を求める．
4. $\phi(n)$ との最大公約数が 1 となり、かつ $1 < e < n$ となる整数 e をひとつ選択する．
5. 次式を満たし、かつ $1 < d < \phi(n)$ となる整数 d をひとつ選択する．

$$(e \times d) \bmod \phi(n) = 1 \tag{9}$$

この手法は前節のものに比べて容易に鍵の値を求められるが，自然数 k を省略しているため，$e \times d$ の候補が多くない．そのため，素数 p, q が小さいときには d と e が等しくなる．