12. Szyfrowanie z kluczem publicznym. Algorytm RSA

Niektóre zagadnienia dotyczące cyberbezpieczeństwa znasz już z wcześniejszych lekcji informatyki. Być może pamiętasz, że z szyfrowania z użyciem klucza publicznego korzystasz praktycznie codziennie – podczas przeglądania szyfrowanej strony internetowej czy logowania do bankowości elektronicznej. W tym temacie przyjrzymy się algorytmom związanym z kryptografią asymetryczną oraz zaprogramujemy jeden z najważniejszych algorytmów kryptograficznych – algorytm RSA.

Cele lekcji

- Utrwalisz wiadomości dotyczące kryptografii symetrycznej i asymetrycznej.
- Poznasz algorytm RSA i jego przykładową implementację.
- Wygenerujesz klucze: publiczny i prywatny.
- Zaszyfrujesz informacje kluczem publicznym i odszyfrujesz je kluczem prywatnym.

Podstawy kryptografii, podręcznik *Informatyka na czasie 1. Zakres rozszerzony*, s. 214–221

Podstawami kryptografii, czyli zagadnieniami zawiązanymi z szyfrowaniem informacji, zajmowaliśmy się już w części pierwszej podręcznika. Przypomnijmy, że podczas szyfrowania informacja jawna jest przekształcana w **szyfrogram** (**kryptogram**). Wykorzystuje się przy tym **klucz**, czyli dane umożliwiające zaszyfrowanie lub odszyfrowanie informacji.

Szyfrogram (kryptogram) **Klucz**

Jeśli do deszyfrowania służy ten sam klucz, którego użyto do szyfrowania, lub można go łatwo wyznaczyć na podstawie tego klucza, to mamy do czynienia z **kryptografią symetryczną**. Taki klucz musi być utrzymywany w tajemnicy.

Kryptografia symetryczna

Bezpieczniejszym sposobem szyfrowania jest zastosowanie **kryptografii asymetrycznej**, nazywanej też **kryptografią klucza publicznego**, w której występują dwa klucze: klucz publiczny oraz klucz prywatny. Na podstawie jednego klucza nie da się w rozsądnym czasie wyznaczyć drugiego, dlatego **klucz publiczny** może być przekazany wszystkim zainteresowanym, którzy chcą zaszyfrować informację. Odszyfrować może ją jednak tylko właściciel **klucza prywatnego** – nie można tego zrobić za pomocą klucza użytego do szyfrowania.

Kryptografia asymetryczna (kryptografia klucza publicznego) **Klucz publiczny** **Klucz prywatny**

Podpis elektroniczny, podręcznik *Informatyka na czasie 1. Zakres rozszerzony*, s. 219–221

Możliwa jest także czynność odwrotna: szyfrowanie kluczem prywatnym i odszyfrowywanie kluczem publicznym. Wówczas informację może odczytać każdy, kto ma dostęp do klucza publicznego. Wykorzystuje się to w **podpisie elektronicznym**.

12.1. Algorytm RSA

Jedną z najbardziej popularnych metod szyfrowania z kluczem publicznym jest **algorytm RSA**. Wykorzystuje on fakt, że nie potrafimy w rozsądnym czasie wykonać **faktoryzacji** bardzo dużej (kilkusetcyfrowej) liczby, czyli rozłożyć jej na czynniki pierwsze.

Algorytm RSA **Faktoryzacja liczby**

W metodzie RSA można wyróżnić trzy główne etapy: generowanie kluczy, szyfrowanie z kluczem publicznym oraz deszyfrowanie z kluczem prywatnym.

Warto wiedzieć

Nazwa algorytmu RSA pochodzi od pierwszych liter nazwisk jego twórców, profesorów Massachusetts Institute of Technology – Rona Rivesta, Adiego Shamira i Leonarda Adlemana. Opisali oni ten algorytm w 1977 r.

Algorytm RSA – generowanie kluczy

W pierwszym etapie generuje się klucze: publiczny i prywatny. Można to zrobić w trzech krokach.

Krok 1. Obliczenia wstępne

Wybieramy dwie duże liczby pierwsze p i q oraz obliczamy iloczyny:

$$n = p \cdot q$$
$$m = (p - 1) \cdot (q - 1)$$

Liczby pierwsze p i q nie będą już potrzebne. Liczba n będzie elementem zarówno klucza publicznego, jak i prywatnego.

Warto wiedzieć

Liczba m jest równa wartości funkcji Eulera $\varphi(n)$ określającej, ile jest liczb względnie pierwszych z n mniejszych od n.

Krok 2. Generowanie klucza publicznego

Poszukujemy liczby e ($1 < e < m$) względnie pierwszej z liczbą m utworzoną w poprzednim kroku, czyli takiej, która spełnia równość:

$$\text{NWD}(e, m) = 1$$

Można wykorzystać do tego **algorytm Euklidesa**. Sprawdzimy za jego pomocą, dla jakiej liczby e wartość NWD(e, m) będzie równa 1. Liczba e jest liczbą nieparzystą, ponieważ m jest liczbą parzystą (jako iloczyn liczb parzystych), ale nie musi ona być liczbą pierwszą.

Para liczb (e, n) to klucz publiczny.

Algorytm Euklidesa, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 102–108

Krok 3. Generowanie klucza prywatnego

Poszukujemy liczby d odwrotnej do liczby e w **arytmetyce modularnej** modulo m, tzn. szukamy liczby d spełniającej równość:

$$(d \cdot e) \bmod m = 1$$

W poprzednim kroku znaleźliśmy liczbę e, która spełnia równanie NWD(e, m) = 1. Spełnia ona również równanie:

$$\text{NWD}(e, m) \bmod m = 1$$

Arytmetyka modularna, s. 208

Z **rozszerzonego algorytmu Euklidesa** wiemy, że istnieją liczby całkowite x i y, które spełniają równość $\text{NWD}(e, m) = x \cdot e + y \cdot m$. Możemy więc zapisać:

$$(x \cdot e + y \cdot m) \bmod m = 1 \Leftrightarrow (x \cdot e) \bmod m + (y \cdot m) \bmod m = 1$$
$$\Leftrightarrow (x \cdot e) \bmod m = 1$$

Ostatnia równoważność wynika z tego, że $(y \cdot m) \bmod m = 0$, ponieważ liczba $y \cdot m$ dzieli się bez reszty przez m. Zauważ, że poszukiwana liczba d to współczynnik x przy liczbie e w ostatnim zapisie. Para liczb (d, n) to klucz prywatny.

Rozszerzony algorytm Euklidesa, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 250–252

Algorytm RSA – szyfrowanie informacji

Warto wiedzieć

Liczby 0 i 1 nie są szyfrowane, ponieważ po podniesieniu ich do potęgi otrzymamy niezmienione wartości.

W algorytmie RSA szyfrowane są liczby całkowite z przedziału $(1, n)$. Jeśli szyfrowany jest tekst, to jego kolejne znaki są zamieniane na liczby z tego przedziału. Kody znaków można przekształcić w jedną dużą liczbę. Można wykorzystać kody ASCII znaków i łączyć je np. po dwa znaki. Wtedy kod pierwszego znaku wystarczy pomnożyć przez 256 (liczbę wszystkich znaków ASCII) i dodać kod drugiego znaku.

Klucz publiczny, s. 214

Liczbę $x \in (1, n)$ szyfrujemy z wykorzystaniem **klucza publicznego** (e, n) w następujący sposób:

$$y = x^e \bmod n$$

Liczba y jest szyfrogramem liczby x. Potęgowanie wykonujemy w arytmetyce modulo n, więc wynik będzie liczbą mniejszą od n. Do szyfrowania można wykorzystać **algorytm szybkiego podnoszenia do potęgi** w arytmetyce modulo n.

Algorytm szybkiego podnoszenia do potęgi, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 57–58

Algorytm RSA – deszyfrowanie informacji

Klucz prywatny, s. 214

Żeby odszyfrować kryptogram y, należy wykonać potęgowanie z wykorzystaniem danych z **klucza prywatnego** (d, n):

$$x = y^d \bmod n$$

Wartość wykładnika d zna tylko właściciel klucza prywatnego – nie da się jej wyznaczyć w rozsądnym czasie na podstawie znajomości klucza publicznego (e, n).

Zapamiętaj

Algorytm RSA jest metodą szyfrowania z wykorzystaniem kluczy publicznego i prywatnego. Do wyznaczenia klucza publicznego można zastosować algorytm Euklidesa, a do wyznaczenia klucza prywatnego – rozszerzony algorytm Euklidesa. Szyfrowanie i odszyfrowywanie informacji zapisanej w postaci liczby sprowadza się do wykonania potęgowania w arytmetyce modulo n, np. za pomocą algorytmu szybkiego podnoszenia do potęgi.

A to ciekawe

Rewolucja w kryptografii

Przez tysiące lat systemy kryptograficzne działały na zasadzie klucza symetrycznego. Jego postać i sposób użycia stawały się coraz bardziej skomplikowane, a zarazem nierozwiązany pozostawał problem, jak ten klucz bezpiecznie przekazywać i przechowywać. W 1976 r. Whitfield Diffie oraz Martin Hellman zaprezentowali koncepcję systemu z kluczem publicznym, która rozpoczęła nową erę w kryptografii. W konsekwencji powstały znacznie bezpieczniejsze systemy takie jak RSA, DSA czy ElGamal.

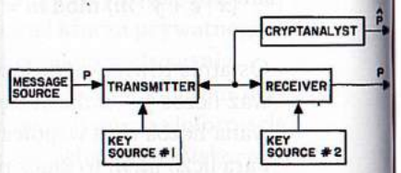


Fig. 2. Flow of information in public key system.

12.2. Program generujący klucz publiczny i klucz prywatny

Napiszemy teraz program generujący klucze zgodnie z algorytmem RSA. Nie będziemy się posługiwać wielkimi liczbami (kilkusetcyfrowymi), lecz wykorzystamy dwie małe liczby pierwsze $p = 13$ i $q = 23$, które pozwolą używać typu `int`. Prześledźmy kolejne kroki obliczeń.

▸ **Krok 1. Obliczenia wstępne**

$$n = 13 \cdot 23 = 299$$
$$m = 12 \cdot 22 = 264$$

▸ **Krok 2. Generowanie klucza publicznego**

Poszukujemy liczby względnie pierwszej z liczbą 264. Ponieważ musi to być liczba nieparzysta, rozpoczynamy poszukiwanie od liczby 3. Korzystając z algorytmu Euklidesa, możemy obliczyć, że NWD(3, 264) = 3, zatem będziemy przeglądać kolejne liczby nieparzyste (znajdziemy najmniejszą taką liczbę). Następną liczbą jest 5, dla której NWD(5, 264) = 1.

Znaleźliśmy więc parę (5, 299), która jest kluczem publicznym.

Warto wiedzieć

Liczb e większych od 1 i mniejszych od m spełniających warunek NWD(e, n) = 1 może być wiele, a więc na podstawie liczb pierwszych p i q można wyznaczyć więcej niż jeden klucz publiczny.

▸ **Krok 3. Generowanie klucza prywatnego**

Za pomocą **rozszerzonego algorytmu Euklidesa** znajdujemy rozkład:

$$\text{NWD}(5, 264) = 53 \cdot 5 + (-1) \cdot 264$$

Para (53, 299) jest kluczem prywatnym dla klucza publicznego (5, 299).

Rozszerzony algorytm Euklidesa, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 250–252

Ćwiczenie 1

a. Poszukaj innej pary kluczy dla liczb pierwszych 13 i 23.
b. Znajdź klucze publiczny i prywatny dla liczb pierwszych 17 i 19.

Specyfikację problemu generowania kluczy w algorytmie RSA możemy sformułować następująco:

Specyfikacja

Dane: p, q – liczby pierwsze, $p > 2$, $q > 2$.

Wynik: d, e, n – liczby całkowite dodatnie tworzące odpowiednio klucz publiczny (e, n) i klucz prywatny (d, n) szyfru RSA.

Zapiszmy w pseudokodzie algorytm generowania kluczy, zgodnie z powyższą specyfikacją.

```
n ← p*q
m ← (p-1)*(q-1)
e ← 3
dopóki NWD(e,m) ≠ 1 wykonuj e ← e + 2
(x,y) ← NWDRoz(e,m)
d ← x
jeśli d < 0 to d ← d + m
```

W pseudokodzie wykorzystaliśmy funkcje NWD oraz NWDRoz, realizujące odpowiednio algorytm Euklidesa oraz rozszerzony algorytm Euklidesa. Oto kod źródłowy programu generującego klucze w algorytmie RSA:

Fragment kodu źródłowego programu generującego klucz publiczny i klucz prywatny w algorytmie RSA

```
1.  int NWD(int a, int b)
2.  {
3.      int pom;
4.      while (b!=0)
5.      {
6.          pom=b;
7.          b=a%b;
8.          a=pom;
9.      }
10.     return a;
11. }
12.
13. pair<int,int> NWDRoz(int a, int b)
14. {
15.     if (b==0) return make_pair(1,0);
16.     pair<int,int> pom=NWDRoz(b,a%b);
17.     return make_pair(pom.second,pom.first-(a/b)*pom.second);
18. }
19.
20. int main()
21. {
22.     int d, e, n, m, p, q;
23.     cout<<"Podaj dwie liczby pierwsze p i q:"<<endl;
24.     cout<<"p = "; cin>>p;
25.     cout<<"q = "; cin>>q;
26.     n=p*q;
27.     m=(p-1)*(q-1);
28.     e=3;
29.     while (NWD(e,m)!=1) e=e+2;
30.     d=NWDRoz(e,m).first;
31.     if (d<0) d=d+m;
32.     cout<<"Klucz publiczny: "<<e<<" "<<n<<endl;
33.     cout<<"Klucz prywatny: "<<d<<" "<<n;
34.     return 0;
35. }
```

Dobra rada

Omówienie kodu źródłowego funkcji NWD oraz NWDRoz znajdziesz w podręczniku *Informatyka na czasie 2. Zakres rozszerzony* na s. 106–108 oraz 250–252.

W liniach 1–11 zdefiniowana jest funkcja NWD, realizująca algorytm Euklidesa, a w liniach 13–18 – funkcja NWDRoz, realizująca rozszerzony algorytm Euklidesa.

W funkcji main w liniach 24–25 wczytywane są dwie liczby pierwsze. W linii 26 obliczana jest wartość zmiennej n, w liniach 28–29 – zmiennej e, a w liniach 30–31 – zmiennej d. Jeśli funkcja NWDRoz zwróci ujemną wartość d, to dodawana jest do niej wartość m (linia 31). W liniach 32–33 wypisywane są wartości obliczonych kluczy.

Ćwiczenie 2

Napisz program, który wczyta dwie liczby pierwsze p i q oraz wygeneruje klucz publiczny i klucz prywatny według algorytmu RSA. Wykorzystaj funkcje NWD oraz NWDRoz.

12.3. Program szyfrujący informacje kluczem publicznym

Przypomnijmy, że w algorytmie RSA szyfrowane są liczby mniejsze od n. W naszym przykładzie n jest równe 299, więc możemy szyfrować kody ASCII pojedynczych znaków (liczby mniejsze od 256).

Na przykład szyfrowanie wielkiej litery A (kod ASCII 65) kluczem publicznym (5, 299) można zapisać następująco:

$$y = 65^5 \bmod 299 = (65 \cdot (65^4 \bmod 299)) \bmod 299 =$$
$$= (65 \cdot ((65^2 \bmod 299) \cdot (65^2 \bmod 299)) \bmod 299) \bmod 299 =$$
$$= (65 \cdot (39 \cdot 39) \bmod 299) \bmod 299 = (65 \cdot 26) \bmod 299 = 195$$

Ćwiczenie 3

Wyznacz szyfrogram litery Z (kod ASCII 90) algorytmem RSA z kluczem (5, 299).

Napiszemy teraz program, który będzie szyfrował plik tekstowy szyfrem RSA. Będziemy szyfrować wszystkie znaki tekstu łącznie ze spacjami, znakami przestankowymi i sterującymi, np. znacznikiem końca wiersza.

Ponieważ będziemy używać klucza bazującego na małych liczbach, będziemy szyfrować pojedyncze znaki (ich kody ASCII). Wynikiem działania programu będzie plik tekstowy złożony z liczb całkowitych oddzielonych spacjami, czyli szyfrogramów poszczególnych znaków.

Oto specyfikacja problemu:

Specyfikacja

Dane: e, n – liczby całkowite dodatnie tworzące klucz publiczny RSA, `plik wejściowy` – plik tekstowy do zaszyfrowania.

Wynik: `plik wynikowy` – plik tekstowy składający się z liczb oddzielonych spacjami, który jest szyfrogramem tekstu z pliku `plik wejściowy` utworzonym zgodnie z algorytmem RSA z kluczem publicznym (e, n).

Zapiszemy w pseudokodzie algorytm szyfrowania informacji. Szyfrowanie pojedynczego znaku realizuje funkcja RSA. Zapiszemy również w języku C++ kod źródłowy funkcji RSA oraz funkcji main.

Warto wiedzieć

Innym sposobem na szyfrowanie znaków tekstu jawnego mogłoby być łączenie kodów kilku znaków w jedną liczbę. Na przykład dla dwóch znaków byłby to wzór $a \cdot 256 + b$, gdzie a jest kodem ASCII pierwszego znaku, a b – kodem ASCII drugiego znaku. Liczba n powinna być wówczas większa od 65 535.

```
dopóki nie koniec pliku wejściowego wykonuj
    ch ← znak z pliku wejściowego
    x ← kod_ASCII(ch)
    y ← RSA(x,e,n)
    zapisz y do pliku wynikowego
```

Fragment kodu źródłowego programu szyfrującego tekst z wykorzystaniem klucza publicznego w algorytmie RSA

```
1.  int RSA(int podst, int wykl, int n)
2.  {
3.      int w=1;
4.      while (wykl>0)
5.      {
6.          if (wykl%2==1) w=(w*podst)%n;
7.          wykl=wykl/2;
8.          if (wykl>0) podst=(podst*podst)%n;
9.      }
10.     return w;
11. }
12.
13. int main()
14. {
15.     ifstream we("t_jawny_RSA.txt");
16.     ofstream wy("szyfrogram_RSA.txt");
17.     unsigned char ch;
18.     int e,n;
19.     cout<<"Podaj klucz publiczny:"<<endl;
20.     cout<<"e = "; cin>>e;
21.     cout<<"n = "; cin>>n;
22.     we>>noskipws;
23.     while (we>>ch) wy<<RSA(ch,e,n)<<" ";
24.     we.close(); wy.close();
25.     cout<<"Tekst zostal zaszyfrowany";
26.     return 0;
27. }
```

Dobra rada

Pamiętaj, że obsługa plików w programie wymaga dołączenia do kodu źródłowego informacji o korzystaniu z bliblioteki `fstream`.

Funkcja RSA (linie 1–11) realizuje algorytm szybkiego podnoszenia do potęgi w arytmetyce modulo *n*, analogicznie jak funkcja Potega w **algorytmie Karpa–Rabina**. W linii 15 otwierany jest do odczytu plik wejściowy o nazwie *t_jawny_RSA.txt*, a w linii 16 otwierany jest do zapisu plik wynikowy *szyfrogram_RSA.txt*.

Algorytm Karpa–Rabina, s. 207

Typ unsigned char

W linii 17 deklarujemy zmienną ch **typu unsigned char**, który reprezentuje wartości od 0 do 255. Ponieważ chcemy szyfrować wszystkie znaki zawarte w tablicy ASCII, a typ **char** reprezentuje liczby z zakresu od –128 do 127. Znaki, których kody ASCII przekraczają 127, byłyby reprezentowane przez liczby ujemne.

Biały znak

W liniach 19–21 wczytywany jest klucz publiczny służący do szyfrowania. Wyjaśnijmy linie 22 i 23. Domyślnie tzw. **białe znaki** (ang. *whitespaces*), np. spacje lub znaczniki końca wiersza, są pomijane podczas odczytywania danych wejściowych za pomocą instrukcji we>>ch.

Ponieważ chcemy odczytywać również białe znaki ze strumienia wejścia, należy to wymusić z wykorzystaniem **manipulatora** noskipws (linia 22). W warunku pętli w linii 23 znajduje się instrukcja odczytu danych we>>ch. Jeśli zakończy się ona niepowodzeniem, to warunek przyjmie wartość liczbową 0, oznaczającą fałsz, w przeciwnym przypadku – wartość różną od zera, interpretowaną jako prawda. Do pliku zapisywane są: wynik funkcji RSA (czyli liczba będąca zakodowanym znakiem) oraz spacja oddzielająca kolejne liczby.

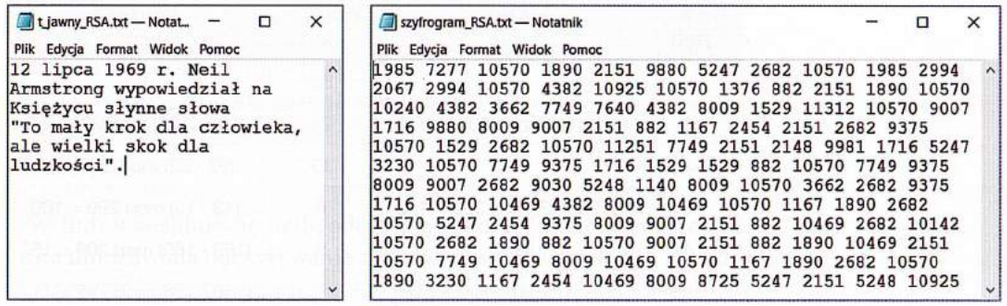
Manipulator noskipws

Warto wiedzieć

Nazwa manipulatora noskipws pochodzi od angielskiego wyrażenia *don't skip whitespaces*.

Pierwszym parametrem funkcji RSA jest znak reprezentowany przez kod ASCII w zakresie do 255. Parametr ten jest wykorzystywany jako zmienna lokalna `podst` typu **`int`**, więc jej wartość może przekroczyć 255. Dwa następne parametry to liczby określające klucz publiczny.

Rysunek 12.1 przedstawia tekst jawny i utworzony szyfrogram.



Rys. 12.1. Tekst jawny i jego szyfrogram dla klucza publicznego (11, 11561)

Ćwiczenie 4

Napisz program wczytujący klucz publiczny i szyfrujący dane z pliku wejściowego o nazwie *t_jawny_RSA.txt* do pliku *szyfrogram_RSA.txt*. Wykorzystaj funkcję RSA.

A to ciekawe

Czy szyfr RSA można złamać?

Obecnie powszechnie stosuje się klucze RSA o długości 2048 bitów. Najdłuższy klucz, który dotychczas złamano, składał się z 768 bitów. Udało się to w 2009 r. z wykorzystaniem klastera, czyli wielu połączonych jednostek komputerowych. Zastosowano wówczas metodę brute-force, a łączny czas prac nad projektem wyniósł 2 lata. Szacuje się, że komputery kwantowe dzięki swojej szybkości będą mogły złamać klucz o długości 2048 bitów w ok. 8 godzin. Należy się więc spodziewać, że w niedalekiej przyszłości do użytku wejdą klucze składające się z 4096 bitów.

12.4. Program deszyfrujący kryptogram kluczem prywatnym

Żeby odszyfrować informację, należy wykonać takie same czynności jak podczas szyfrowania, ale z wykorzystaniem liczb z szyfrogramu i klucza prywatnego.

Odszyfrujmy szyfrogram 195 kluczem (53, 299).

$$x = 195^{53} \bmod 299 = 65$$

Otrzymujemy liczbę 65, która interpretowana jako kod ASCII reprezentuje literę A.

Kod źródłowy funkcji RSA, s. 220

Prześledźmy obliczenia, które wykonuje **funkcja** RSA dla powyższych argumentów. Pośrednie kroki wykonywane w celu wyznaczenia wartości funkcji RSA(195,53,299) ilustruje tabela 12.1.

Iteracja pętli	Wartość w	Wartość wykl	Wartość podst
–	1	53	195
1	(1 · 195) mod 299 = 195	26	(195 · 195) mod 299 = 52
2	195	13	(52 · 52) mod 299 = 13
3	(195 · 13) mod 299 = 143	6	(13 · 13) mod 299 = 169
4	143	3	(169 · 169) mod 299 = 156
5	(143 · 156) mod 299 = 182	1	(156 · 156) mod 299 = 117
6	(182 · 117) mod 299 = 65	0	–

Tabela 12.1. Kolejne kroki obliczania wartości 195^{53} mod 299

Ćwiczenie 5

Odszyfruj szyfrogram 129 kluczem prywatnym (53, 299).

Napiszemy teraz program, który będzie deszyfrował kryptogram utworzony szyfrem RSA. Danymi będą plik tekstowy złożony z liczb oddzielonych spacjami oraz klucz prywatny. Wynikiem będzie plik tekstowy z odszyfrowanym tekstem.

Oto specyfikacja problemu:

Specyfikacja

Dane: *d*, *n* – liczby całkowite dodatnie tworzące klucz prywatny RSA, `plik wejściowy` – plik tekstowy złożony z liczb całkowitych dodatnich oddzielonych spacjami – zaszyfrowany tekst.

Wynik: `plik wynikowy` – plik tekstowy zawierający kryptogram z pliku `plik wejściowy` odszyfrowany kluczem prywatnym (*d*, *n*).

Zapiszmy w pseudokodzie algorytm deszyfrowania informacji.

```
dopóki nie koniec pliku wejściowego wykonuj
    y ← liczba z pliku wejściowego
    x ← RSA(y,d,n)
    zapisz znak_ASCII(x) do pliku wynikowego
```

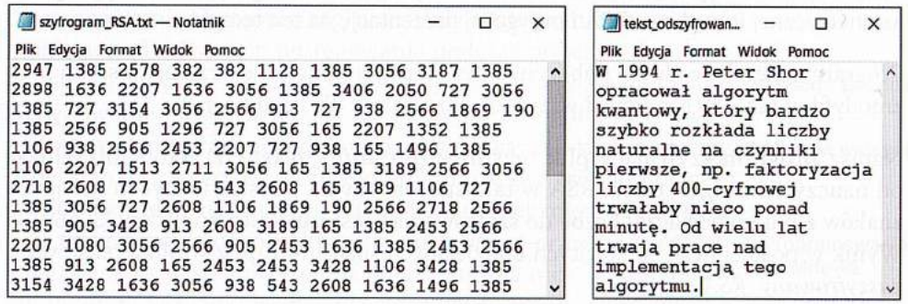
Oto kod źródłowy funkcji `main` w programie deszyfrującym:

```
1.  int main()
2.  {
3.      ifstream we("szyfrogram_RSA.txt");
4.      ofstream wy("tekst_odszyfrowany.txt");
5.      int d,n,x;
6.      cout<<"Podaj klucz prywatny:"<<endl;
7.      cout<<"d = "; cin>>d;
8.      cout<<"n = "; cin>>n;
9.      while (we>>x) wy<<char(RSA(x,d,n));
10.     we.close(); wy.close();
11.     cout<<"Tekst zostal odszyfrowany";
12.     return 0;
13. }
```

● **Kod źródłowy funkcji `main` programu deszyfrującego szyfrogram z wykorzystaniem klucza prywatnego w algorytmie RSA**

W linii 9 znajduje się pętla odczytująca liczby, analogiczna do tej, która umożliwiała odczytywanie znaków tekstu jawnego. Jeśli nie uda się odczytać liczby (np. gdy program dojdzie do końca pliku), instrukcja `we>>x` zwróci wartość liczbową 0, czyli fałsz. Do pliku zapisujemy znak, więc dokonujemy konwersji kodu ASCII na typ `char`. **Kod źródłowy funkcji** RSA jest taki sam jak w programie szyfrującym. Rysunek 12.2 przedstawia szyfrogram i tekst jawny.

Kod źródłowy funkcji RSA, s. 220



Rys. 12.2. Fragment szyfrogramu i tekst odszyfrowany z kluczem prywatnym (2107, 3487)

Ćwiczenie 6

Napisz program wczytujący klucz prywatny i deszyfrujący szyfrogram z pliku *szyfrogram_RSA.txt*. Odszyfrowany tekst zapisz do pliku *tekst_odszyfrowany.txt*. Wykorzystaj funkcję RSA.

Podsumowanie

- W kryptografii asymetrycznej wykorzystuje się dwa klucze: klucz publiczny (ogólnie dostępny) i prywatny (dostępny tylko dla właściciela). Informację zaszyfrowaną kluczem publicznym można odszyfrować tylko kluczem prywatnym.
- W podpisie elektronicznym stosuje się szyfrowanie kluczem prywatnym i odszyfrowywanie kluczem publicznym. Wówczas informację może odszyfrować każdy, kto ma dostęp do klucza publicznego.
- Jednym z algorytmów szyfrowania z kluczem publicznym jest algorytm RSA. Jego działanie opiera się na fakcie, że nie znamy szybkiego algorytmu faktoryzacji (rozkładu na czynniki pierwsze) bardzo dużych liczb.
- W algorytmie RSA klucz publiczny to para liczb (e, n), gdzie n jest iloczynem dwóch dużych liczb pierwszych p i q, natomiast e jest liczbą względnie pierwszą z liczbą $m = (p-1) \cdot (q-1)$, $1 < e < m$. Klucz prywatny to para (d, n), gdzie d jest odwrotnością liczby e w arytmetyce modulo m.
- W algorytmie RSA szyfrowane są liczby całkowite większe od 1 i mniejsze od n. Kryptogramem liczby x jest liczba $y = x^e \bmod n$. Żeby odszyfrować informację, należy wykonać działanie $y^d \bmod n$.

Zadania

* **1** Wygeneruj klucze publiczny i prywatny z użyciem liczb pierwszych $p = 251$ i $q = 263$.

* **2** Przygotuj prezentację na temat szyfrowania z kluczem publicznym.

** **3** Poszukaj w dostępnych źródłach informacji na temat algorytmów kryptografii asymetrycznej innych niż RSA i przygotuj prezentację na ten temat.

** **4** Program generujący klucze publiczny i prywatny na podstawie liczb pierwszych p i q zmodyfikuj tak, aby generował wszystkie pary kluczy dla liczb p i q.

*** **5** Napisz program szyfrujący plik tekstowy (np. *tajny_tekst.txt*), który otrzymasz od nauczyciela, algorytmem RSA w taki sposób, aby na podstawie dwóch kolejnych znaków z pliku wyznaczał liczbę do szyfrowania. Zastosuj klucz publiczny (3, 66013). Wynik w postaci liczb całkowitych oddzielanych spacjami zapisz w pliku tekstowym *zaszyfrowany_RSA.txt*.
Wskazówka: Parametr `podst` funkcji szyfrującej powinien być typu **`long long`**.

*** **6** Napisz program deszyfrujący kryptogram zaszyfrowany algorytmem RSA zawarty w pliku tekstowym (np. *tajny_szyfrogram.txt*), który otrzymasz od nauczyciela. Kryptogram składa się z liczb całkowitych oddzielonych spacjami. Jedna liczba stanowi szyfrogram dwóch znaków tekstu jawnego. Do odszyfrowania użyj klucza prywatnego (43667, 66013). Wynik zapisz w pliku tekstowym *tajny_odszyfrowany.txt*.
Wskazówka: Parametr `podst` funkcji deszyfrującej powinien być typu **`long long`**.

13. Programowanie obiektowe

Wiele systemów informatycznych i programów ma budowę okienkową. Dzięki niej aplikacje stały się łatwiejsze w obsłudze i dostępne dla szerszej grupy użytkowników. Szybki rozwój tego typu oprogramowania umożliwiły m.in. zasady programowania obiektowego, którego podstawy poznasz w tym temacie.

Cele lekcji

- Poznasz pojęcia klasy i obiektu oraz atrybutu i metody.
- Dowiesz się, na czym polega hermetyzacja danych.
- Zbudujesz hierarchię klas.
- Poznasz pojęcie polimorfizmu i zastosujesz metody wirtualne.

Do tej pory tworzyliśmy programy zgodnie z zasadami **programowania strukturalnego**. Instrukcje wykonywały się sekwencyjnie, do sterowania wykorzystywaliśmy instrukcje warunkowe i pętle, a rozwiązania mniejszych problemów zapisywaliśmy w postaci funkcji. Dane i operacje na nich traktowane były jednak niezależnie. Komunikacja między funkcjami odbywała się za pośrednictwem parametrów oraz zwracanych wartości.

Programowanie strukturalne

Postawiony problem staraliśmy się podzielić na mniejsze problemy, określić sposób komunikacji między funkcjami i zapisać algorytm rozwiązania problemu głównego, bez zajmowania się na początku szczegółami. Następnie zapisywaliśmy rozwiązania mniejszych problemów. Taki sposób postępowania podczas projektowania programów nazywa się **metodą zstępującą** (ang. *top-down approach*). Postępowanie odwrotne, czyli od szczegółu do ogółu, nazywa się **metodą wstępującą** (ang. *bottom-up approach*). Jeśli ją zastosujemy, rozwiązania problemów szczegółowych może być trudno połączyć w całość bez odpowiednio uzgodnionej komunikacji między funkcjami.

Metoda zstępująca

Metoda wstępująca

W **programowaniu obiektowym** (ang. *object-oriented programming*) dane są ściśle połączone z operacjami na nich wykonywanymi. Program jest traktowany jako zbiór obiektów współpracujących ze sobą w celu rozwiązania problemu. Programowanie strukturalne i obiektowe często są łączone. Wiele języków programowania, w tym C++, umożliwia korzystanie z obu tych podejść.

Programowanie obiektowe

My także w tym podręczniku stosowaliśmy programowanie obiektowe, wykorzystując chociażby typ **`string`**. Nauczymy się teraz definiować własne struktury obiektowe.