Ochrana dat pomocí šifrovacích algoritmů

Bc. Tomáš Studený

Diplomová práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Tomáš STUDENÝ**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Ochrana dat pomocí šifrovacích algoritmů**

Zásady pro vypracování:

1) Analyzujte teoretické informační zdroje z oblasti modulární aritmetiky.
2) Zhodnoťte dostupné symetrické a asymetrické šifrovací algoritmy.
3) Analyzujte vybrané algoritmy, zhodnoťte jejich bezpečnost a možnosti použití.
4) Vytvořte aplikaci, demonstrující šifrování textu dle vybraných algoritmů.

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1.Grošek, O. – Podrubský, Š.: Šifrovanie – algoritmy, metódy, prax. Bratislava : GRADA, 1992. ISBN 80-85424-62-2
2.Doseděl, T.:Počítačová bezpečnost a ochrana dat. Praha: Computer Press, a. s., 2004. ISBN: 80-251-0106-1
3.Klíma, V.: Security, Cryptology.[online]. Dostupný z URL: http://cryptography.hyperlink.cz
4.Pinkava, J.: Základy kryptografie.[online]. Dostupný z URL: http://crypto-world.info/pinkava/uvod/bulletin1.pdf

Vedoucí diplomové práce: **Mgr. Roman Jašek, Ph.D.**
Ústav informatiky a statistiky

Datum zadání diplomové práce: **14. února 2006**

Termín odevzdání diplomové práce: **26. května 2006**

Ve Zlíně dne 14. února 2006

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Diplomová práce pojednává o dnes v praxi používaných symetrických a asymetrických kryptosytémech. V úvodu jsou popsány matematické základy a operace, které jsou nutné k pochopení navazujících kapitol. Stěžejní místem v diplomové práci je popis jednotlivých šifrovacích kryptosystémů. U jednotlivých kryptosystémů je uvedena jejich bezpečnost a jsou srovnány s ostatními kryptosystémy, a to z hlediska bezpečnosti a možnosti použití. Součástí diplomové práce je i demonstrační aplikace implementující vybrané algoritmy podle jednotlivých kapitol.

Klíčová slova: šifra, kryptografie, kryptosystém, kryptoanalýza, symetrická šifra, asymetrická šifra, útok hrubou sílou

ABSTRACT

In this thesis, studies have been concentrated on symmetric and asymmetric cryptosystems used in practice today. In the introductory part, the mathematical principles and operations necessary for understanding consequential chapters, are described. The most important part of the thesis is the description of component cryptosystems. For each of the cryptosystems, the safety is stated, and they are compared with other cryptosystems in terms of safety and possible use. The demonstrational application implementing selected algorithms in accordance with particular chapters is also part of the thesis.

Keywords: cypher: cryptography, cryptosystem, cryptoanalysis, symmetric cypher, asymmetric cypher, brute force attack

Děkuji tímto vedoucímu mé diplomové práce Mgr. Romanu Jaškovi Ph.D., za odborné vedení, rady a připomínky, které mi poskytl během řešení mé práce.

OBSAH

ÚVOD

Moderní šifrování, neboli kryptografie[1] je jedna z vyšších forem ochrany dat před nežádoucím přístupem k nim. Algoritmická ochrana dat spočívá v transformaci chráněných údajů do jiného nečitelného tvaru. Dnes je to disciplína, bez které si neumíme představit moderní počítačové a komunikační systémy.

Moderní kryptografie má kořeny ve druhé světové válce, která přinesla velký zájem o kryptografii i kryptoanalýzu[2]. Kryptoanalýza se stala tichou, neviditelnou a účinnou zbraní. Po válce se kryptografie stala vojenským zbožím, kdy všechny velmoci začali budovali kryptografická a luštící centra. Vyvrcholení nastalo v sedmdesátých letech dvacátého stolení, kterému pomohla počítačová revoluce. V této době vznikly moderní blokové šifry a byl objeven princip kryptografie s veřejným klíčem.

Šifrovací systémy se rozdělují na symetrické, neboli se soukromým klíčem, které se dále rozdělují na proudové a blokové, a asymetrické, neboli s veřejným klíčem. Symetrické šifry jsou převážně založeny na principech binárních operací s jednotlivými bity. Asymetrické šifry jsou založeny na matematických principech modulární aritmetiky. K dešifrování zašifrované zprávy je třeba dešifrovací klíč. Pokud se někdo pokouší o prolomení zašifrované zprávy jedná se o kryptoanalýzu.

Bezpečnost kryptosystému je funkce dvou proměnných – síly algoritmu a velikosti klíče. Algoritmus se silný tehdy, jestliže neexistuje lepší způsob kryptoanalýzy, než zkoušet všechny možné klíče. Takový útok se nazývá hrubou silou a je nejméně efektivní. Klíč je silný tehdy, jestliže počet všech možných kombinací je tak vysoký, že útok hrubou silou je prakticky nepoužitelný.

V současné době jsou požadavky na bezpečný kryptosystém takové, že v případě kryptoanalýzy je znám šifrovací algoritmus a není znám pouze šifrovací klíč. Pokud v

[1] Moderní kryptografie podle [5] můžeme v širším významu definovat jako studium matematických metod pro zajištění informační bezpečnosti.

[2] Moderní kryptoanalýzu podle [5] můžeme v širším významu definovat jako vědu o hledání slabin nebo prolamování matematických metod informační bezpečnosti.

takovém případě šifra odolá kryptoanalýze, můžeme tento kryptosystém považovat za bezpečný.

I. TEORETICKÁ ČÁST

1 MATEMATICKÝ ZÁKLAD

Tato kapitola se zabývá matematickým základem na kterém jsou založeny dále popisované symetrické a asymetrické kryptosystémy. Bylo čerpáno převážně z [1] [2] a [3].

1.1 Teorie čísel

Teorie čísel je základní část k pochopení modulární aritmetiky, která je nutná k pochopení algoritmů popisovaných v dalších kapitolách, především však algoritmu RSA. Pracujeme v oboru přirozených čísel, případně čísel celých.

Největší společný dělitel

Největší společný dělitel (GCD) dvou přirozených čísel *a,b* je největší přirozené číslo, které dělí obě dvě čísla zároveň. Označuje se jako *GCD(a,b).* Jeli *GCD(a,b) = 1*, potom se čísla *a,b* nazývají nesoudělitelná.

Věta: Pro každé celé číslo *n* platí*: GCD(n,n+1) = 1*.

Euklidův algoritmus

Euklidův algoritmus je postup pro nalezení GCD dvou čísel. Opírá se o následující vlastnosti libovolných přirozených čísel *a,b*.

- *GCD(a,a) = a*
- *GCD(a,b) = GCD(b,a)*
- *GCD(a,b) = GCD(a – b,b)* pro *a > b*

Euklidův algoritmus pochází přibližně z doby 300 let př. n. l. a je nejstarší netriviální algoritmus, který se používá dodnes. Algoritmus funguje následovně:

```
// vstup: dvě přirozená čísla
// výstup: největší společný dělitel

public static int GCD(int a, int b)
{
    int g = b;
    while (a > 0)
    {
```

```
        g = a;
        a = b % a;
        b = g;
    }
    return g;
}
```

Eulerova funkce

Eulerova funkce $\Phi(n)$ udává počet přirozených čísel menších než n, která je nesoudělitelná s n (to znamená taková $a < n$, pro která platí $GCD(n,a) = 1$. Jeli n prvočíslo, potom $\Phi(n) = n - 1$, protože každé přirozené číslo menší než n je s n nesoudělné. Na základě této definice funkce $\Phi(n)$ platí: $\Phi(1) = 1$.

Prvočísla

Přirozené číslo $p \geq 2$ se nazývá prvočíslo, jeli dělitelná pouze sebou sama a jedničkou. V opačném případě se nazývá číslo složené.

Věta: Každé přirozené číslo větší než 1 je buď prvočíslo nebo se dá zapsat jednoznačně jako součin mocnin různých prvočísel $p_1^{e1} . p_2^{e2} \ldots\ldots p_n^{en}$

Věta: Prvočísel existuje nekonečně mnoho.

Věta: Pro každé přirozené číslo n, existuje jedno prvočíslo p takové, že *platí* $n \leq p \leq 2n$.

1.2 Modulární aritmetika

Modulární aritmetika je upravená aritmetika nad celými čísly, které na přelomu 19. století formuloval K. F. Gauss (1801). Výsledkem operací v modulární aritmetice jsou prvky konečné množiny celých čísel.

Funkce modulo (mod) vrací zbytek po celočíselném dělení. Je-li $a = t \cdot n + z$, a,n jsou přirozená čísla, t,z celá nezáporná čísla, $0 \leq z < n$, pak píšeme $a \bmod n = z$ nebo $a \equiv z(\bmod n)$.

Modulární aritmetika je komutativní, asociativní a distributivní. Mimo jiné platí:

- $(a \pm b) \bmod n = ((a \bmod n) \pm (b \bmod n)) \bmod n$

- $(a \cdot b) \bmod n = ((a \bmod n) \cdot (b \bmod n)) \bmod n$

Kongruence

Nechť *m* je cele kladné číslo. Čísla *x,y* se nazývají kongruentní modulo *m*, když $x - y$ je dělitelné *m*. Top znamená, že zbytek po dělení m je u obou čísel stejný. Tento vztah zapisujeme:

$$x \equiv y \pmod m$$

Eulerova věta

Nechť *a* je kladné celé číslo nesoudělné s *n*, potom platí

$$a^{\Phi(n)} \equiv n = 1$$

což se dá také zapsat jako

$$a^{\Phi(n)} \equiv 1 (n = 1)$$

Inverzní modulo

Inverze pro operaci násobení označované * znamená najít pro libovolné *x* takovou hodnotu *x',* že platí $x * x' = 1$. V modulární aritmetice znamená inverze *m* nalezení k *x* takového čísla *x',* že platí $(x' \cdot x) \bmod m = 1$.

Věta: Nechť *x* a *m* jsou nesoudělitelná přirozená čísla. Potom existuje právě jedno přirozené číslo *a*, tak že platí

$$(a \cdot x) \bmod m = 1$$

Tento vztah lze zapsat jako kongruenční rovnici

$$a \cdot x \equiv 1 (\bmod m)$$

Řešení této rovnice je ekvivalentní hledání takových přirozených čísel *a* a *k*, která by splňovala podmínku

$$a \cdot x \equiv k \cdot m + 1$$

Na základě zmíněného vztahu se nabízí jednoduchý algoritmus pro nalezení inverzního modula, který však není příliš efektivní. Funkce algoritmu je znázorněna na konkrétním příkladě hledání inverze 3 modulo 10 (viz. Tab. 1.). Jelikož jsou čísla 3 a 10 nesoudělitelná, existuje právě jedno řešení ve tvaru $a \cdot 3 = k \cdot 10 + 1$.

Tab. 1. Výpočet inverzního modula

a	**a · 3**	**k**	**k · 10 + 1**
1	3	1	11
2	6	2	21
3	9	3	31
4	12	4	41
5	15	5	51
6	18	6	61
7	21	7	71
8	24	8	81
9	27	9	91

Z tabulky vyplývá, že jsou a = 7 a k = 2. Pro kontrolu ověříme, že platí $7 \cdot 3 = 2 \cdot 10 + 1$. Sofistikovanější a v praxi nejpoužívanější algoritmus pro výpočet inverzního modula se nazývá rozšířený Euklidův algoritmus.

Rozšířený Euklidův algoritmus

Tento algoritmus vychází z algoritmu pro určení největšího společného dělitele. Pro zadání čísel *x*, *y* nalezne čísla *a,b* tak, že platí $ax + by = v$, kde $v = GCD(x,y)$. Následuje programový kód algoritmu.

```
// jako vstup dvě přirozená čísla x,y
// výstup čísla a, (b) taková, že platí: ax + by = GCD(x,y)
// funkce Even() je true pokud je cislo sude

public ulong InverseModulo(ulong x, ulong y)
{
    ulong a, b, c, d, u, v, g;

    g = 1;

    while (Even(x) && Even(y))
    {
        x /= 2; y /= 2; g *= 2;
    }

    u = x; v = y; a = 1; b = 0; c = 0; d = 1;

    do
```

```
    {
        while (Even(u))
        {
            u /= 2;
            if (Even(a) && Even(b))
            {
                a /= 2; b /= 2;
            }
            else
            {
                a = (a + y) / 2; b = (b - x) / 2;
            }
        }

        while (Even(v))
        {
            v /= 2;
            if (Even(c) && Even(d))
            {
                c /= 2; d /= 2;
            }
            else
            {
                c = (c + y) / 2; d = (d - x) / 2;
            }
        }

        if (u >= v)
        {
            u -= v; a -= c; b -= d;
        }
        else
        {
            v -= u; c -= a; d -= b;
        }

    } while (u != 0);

    a = c; b = d;

    return a;
}
```

1.3 Bitové operace

Kapitola shrnuje bitové operace nezbytně nutné k pochopení algoritmů popisovaných v dalších kapitolách, především však u symetrických kryptosystémů.

Sčítání

Předpokládejme sčítání dvou čísel ve dvojkové soustavě s délkou k bitů (v případě, že jedno z čísel má méně bitů, je doplněno zleva nulami).

Sečtení dvou k-bitových čísel vyžaduje k-bitových operací:

1. Přečteme horní a dolní bit a také je přenos u horního bitu.
2. Pokud jsou oba bity nula a není žádný přenos, napíšeme 0 a posuneme se doleva.
3. Pokud jsou (a) oba bity 0 a je přenos nebo (b) jeden z bitů je 0 druhý je 1 a není přenos, pak napíšeme 1 a posuneme se.
4. Pokud je (a) jeden z bitů 0 a druhý je 1 a přenos nebo (b) oba bity jsou 1 a není přenos, pak napíšeme 0, nastavíme přenos do dalšího sloupce a posuneme se.
5. Pokud oba bity jsou 1 a je přenos, pak napíšeme 1, nastavíme přenos do dalšího sloupce a posuneme.

Příklad sečtení dvou čísel.

```
172 + 234

 10101100
+11101010
---------
110010110
```

Násobení

Při násobení k-místného dvojkového čísla l-místným dvojkovým číslem, kde k > 1 dostaneme v nejhorším případě *l* pod sebou zapsaných posunutých kopii většího činitele.

Po doplnění nulami je délka maximálně k + *l* – 1. Při sčítaní pak provedeme maximálně *l* – 1 součtů k + *l* – 1 bitových operací.

Příklad násobení dvou čísel.

```
45 * 13

   101101
     1101
----------
   101101
  101101
 101101
----------
1001001001
```

Dělení

Při dělení k-místného dvojkového čísla l-místným dvojkovým číslem kde *l* ≥ k musíme vykonat v nejhorším případě *(k – l + 1)* odčítání (*l + 1)* místných čísel.

Dohromady $(k - l + 1)(l + 1)$

Příklad dělení dvou čísel.

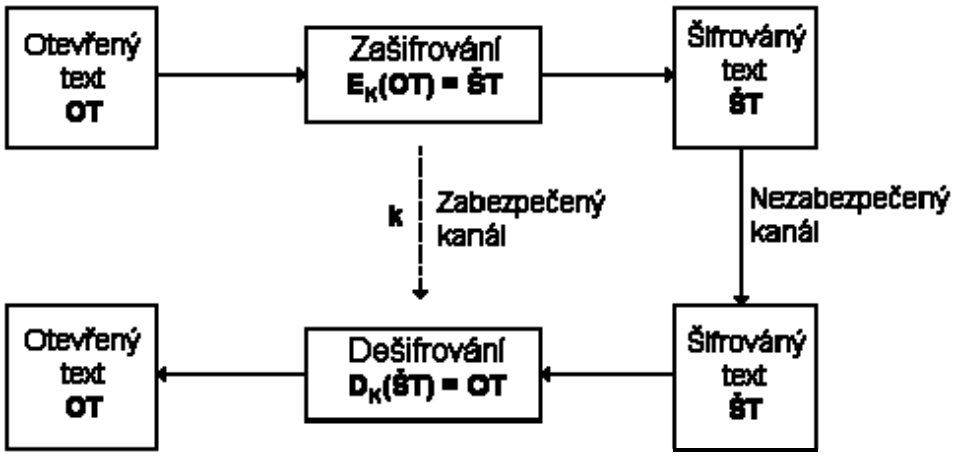
```
585 / 13

1001001001:1101 = 101101
-1101
----------
  10101
  -1101
----------
   10000
   -1101
----------
      1101
     -1101
----------
         0
```

2 SYMETRICKÉ ŠIFROVÁNÍ

Kapitola pojednává o principech symetrické kryptografie a některých oblíbených moderních symetrických algoritmech, které jsou popisovány důkladněji. Bylo čerpáno převážně z [1],[5] a [6].

Definice: Symetrická šifra je taková šifra, kde pro každé $k \in K$ lze z transformace zašifrování E_k určit transformaci dešifrování D_k a naopak (viz. Obr. 1.).



Obr. 1. Schéma symetrického šifrování

Symetrické šifry se dělí podle způsobu šifrování na:

- Proudové
- Blokové

Proudová šifra šifruje zvlášť jednotlivé znaky abecedy, zatímco bloková šifra zpracovává najednou bloky (řetězce) délky *t* znaků.

Symetrická kryptografie je vhodná převážně pro:

- komunikační protokoly
- šifrování velkých objemů dat

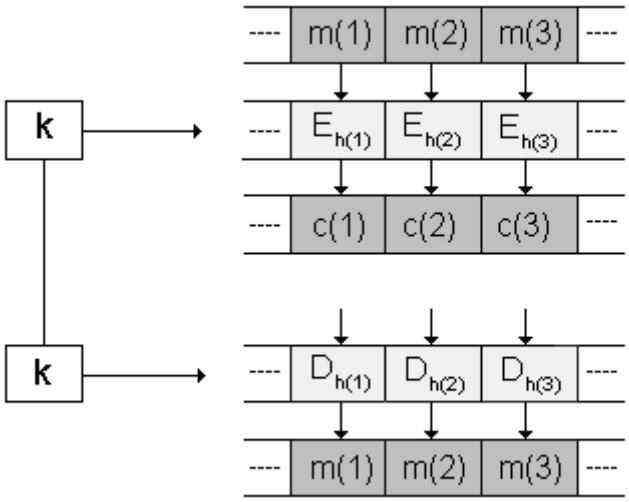
2.1 Proudové šifry

Definice: Nechť A je abeceda q symbolů, nechť $M = C$ je množina všech konečných řetězců nad A a nechť K je množina klíčů. Proudová šifra se skládá z transformace (generátoru) G, zobrazení E a zobrazení D. Pro každý klíč $k \in K$ generátor G vytváří posloupnost hesla $h(1), h(2),\ldots$, přičemž prvky $h(i)$ reprezentují libovolné substituce $E_{h(1)}$, $E_{h(2)}$, .. nad abecedou A. Zobrazení E a D každému klíči $k \in K$ přiřazují transformace zašifrování E_K a odšifrování D_K. Zašifrování otevřeného textu $m = m(1), m(2),..$ probíhá podle vztahu

$$c(1) = E_{h(1)}(m(1)),\ c(2) = E_{h(2)}(m(2))$$

a dešifrovaného textu $c = c(1), c(2),$ .. probíhá podle vztahu

$$m(1) = D_{h(1)}(c(1));\ m(2) = D_{h(2)}(c(2)),\ kde\ D_{h(i)} = E_{h(1)}^{-1}$$ (viz. Obr. 2.).



Obr. 2. Schéma proudové šifry

Proudové šifry se používají u tzv. šifrátorů, kdy do komunikačního kanálu přichází jednotlivé znaky v pravidelných nebo nepravidelných časových intervalech, přičemž v daném okamžiku je nutné tento znak okamžitě přenést, takže není vhodné nebo možné čekat na zbývající znaky bloku. Proudové šifry se také používají v případech, kde šifrovací zařízení má omezenou paměť na průchozí data.

Další výhodou oproti blokovým šifrám je rekonstrukce textu v případě chyby jednoho znaku v komunikačním kanálu, projeví se tato chyba u proudových šifer pouze v jednom odpovídajícím znaku otevřeného textu. U blokové šifry má vliv na celý blok znaků.

2.1.1 XOR

XOR je logická funkce *exkluzivní nebo*, která se používá jako jedna z nejjednodušších šifrovacích algoritmů. Funkce XOR je znázorněna v tabulce 2.

Tab. 2. XOR

x	y	x xor y
0	0	0
0	1	1
1	0	1
1	1	0

Šifrování probíhá tak, že na proud zprávy kterou chceme zašifrovat, aplikujeme periodicky klíč (heslo) metodou XOR a tím vznikne zašifrovaná zpráva. Dešifrování se provádí stejným způsobem:

Chceme zašifrovat zprávu *10101000111001101010111100l001* pomocí klíče *1100111010.* Postup šifrovaní a dešifrování ukazuje tabulka 3.

Tab. 3. Šifrování a dešifrování zprávy pomocí XOR

Zpráva	1	0	1	0	0	0	0	1	1	1	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	0	0	1	1
Heslo	1	1	0	0	1	1	1	0	1	0	1	1	1	0	0	1	1	1	0	1	0	1	1	1	0	0	1	1	1	0
Šifr. z.	0	1	1	0	1	1	1	1	0	1	1	1	0	1	0	0	1	0	0	0	1	0	0	1	0	1	1	1	0	1
Heslo	1	1	0	0	1	1	1	0	1	0	1	1	1	0	0	1	1	1	0	1	0	1	1	1	0	0	1	1	1	0
Dešifr. z.	1	0	1	0	0	0	0	1	1	1	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	0	0	1	1

Nejdůležitějším bezpečnostním aspektem je, aby šifrovací klíč nebyl příliš krátký a tím i délka periody, může hrozit, že jednotlivé časti textu budou zašifrovány stejným způsobem, což muže usnadnit případnou kryptoanalýzu.

Následující kód ukazuje programový zápis funkce XOR.

```
// vstup: dva binární řetězce - šifrovaný text a heslo
// výstup: binární řetězec zašifrovaného textu
```

```
public string Xor(string binaryString, string binaryPasswd)
{
   string resultString = "";
   int j = 0;

   while(j < binaryString.Length-1)
   {
      for(int i = 0; i < binaryPasswd.Length; i++)
      {
          if(binaryString[j] == binaryPasswd[i])
                {
                    resultString += "0";
                }
          else
                {
              resultString += "1";
                }

          if(j < binaryString.Length-1)
                {
                   j++;
                }
             else
                {
                   break;
                }
        }
   }
   return resultString;
}
```

2.1.2 Vermanova šifra

Vermanova šifra používá náhodné heslo stejně dlouhé jako otevřený text a po použití se ničí, takže nikdy není použito k šifrování dvou různých otevřených textů. Tento způsob šifrování navrhl major americké armády Joseph Mauborgn krátce po první světové válce, ale nazývá se Vermanova šifra po Gilbertu Vermanovi, který ji nechal patentovat.

Předpokládáme ze máme k dispozici klíče stejně dlouhé jako přenášená zpráva a klíč je použit jen jednou. Celá tato bezpečnost činí tento kryptosystém absolutně bezpečný. Šifrovací algoritmus může být přitom velice jednoduchý, nejčastěji se používá logická funkce XOR. Pokud mám dvě zprávy stejné délky a klíč také stejné délky tak spočítám

$K_A = A \; xor \; K$ a $K_B = B \; xor \; K$. Dále vypočítám $X = A \; xor \; B$ a nakonec $Y = X \; xor \; K$ =>

- $A = Y \; xor \; K_B$
- $B = Y \; xor \; K_A$

Klíče *KA* a *KB* dešifrují zprávu *Y* na dvě různé zprávy. Toho se dá využit např. ke zmatení nepřítele tím, že mu podstrčíme jeden z dešifrovacích klíčů.

Důkaz absolutní bezpečnosti:

Nechť *h(i), o(i)* a *c(i)* jsou po řadě bit hesla, otevřeného a šifrovaného textu.

Máme $P\{o(i) = 0\} = P\{c(i) - h(i) = 0\} = P\{h(i) = c(i)\}$.

Tento výraz je roven

$P\{h(i) = 0\}$, v případě, že $c(i) = 0$

$P\{h(i) = 1\}$, v případě, že $c(i) = 1$.

Protože $P\{h(i) = 0\} = P\{h(i) = 1\} = ½$, je v obou případech výraz roven ½, tedy celkově $P\{o(i) = 0\} = ½$. Obdobně ukážeme, že $P\{o(i) = 1\} = ½$ nezávisle na hodnotě šifrovaného textu. Právě jsme dokázali, že šifrovaný text nenese žádnou informaci o otevřeném textu, což je definice absolutně bezpečné šifry.

Nevýhoda tohoto algoritmu je příliš velký klíč, který musí být někde uložen a z toho vyplývají další bezpečnostní rizika.

2.2 Blokové šifry

Definice: Nechť A je abeceda q symbolů, $t \in N$ a $M = C$ je množina všech řetězců délky t nad A. Nechť K je množina klíčů. Bloková šifra je šifrovací systém *(M,C, K, E, D),* kde E a D jsou zobrazením, definující pro každé $k \in K$ transformaci zašifrování E_k a dešifrování D_k tak, že zašifrování bloků otevřeného *textu m(1), m(2), m(3),…, (*kde $m(i) \in M$ pro každé N*)* probíhá podle vztahu

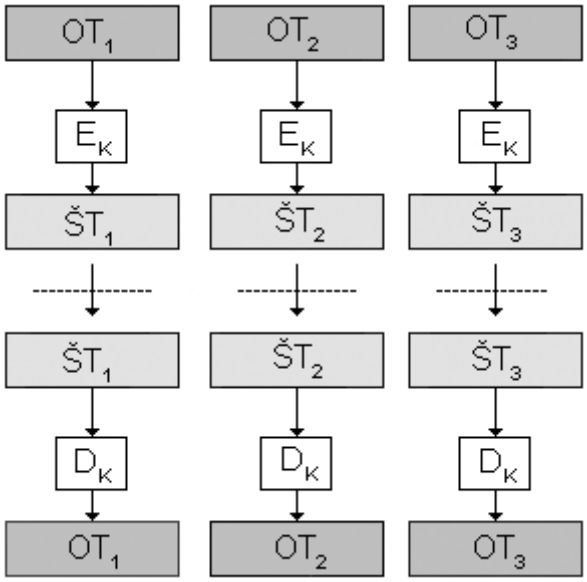
$$c(i) = E_k(m(i)) \text{ pro každé } N$$

a dešifrování podle vztahu

$$m(i) = D_k(c(i)) \text{ pro každé } i \in N.$$

Pro definici blokové šifry je podstatné, že všechny bloky otevřeného textu jsou šifrovány toutéž transformací a všechny bloky šifrovaného textu jsou dešifrovány toutéž transformací.

Schéma šifrování u blokové šifry viz. Obr. 3.



Obr. 3. Schéma blokové šifry

Blokové šifry zašifrovávají současně celý blok. Velikost vstupního bloku blokové šifry má základní význam pro bezpečnost celého algoritmu. Pokud by velikost tohoto bloku byla malá, pak by bylo možné vytvořit "slovník", tj. sestavit kompletní seznam (při určitém klíči) vstupních a jim odpovídající výstupních hodnot algoritmu. To by samozřejmě mělo velmi nepříznivý dopad na bezpečnost celého algoritmu. Proto je nezbytné volit velikost

vstupního bloku "dostatečně velikou", tj. takovou, aby vytvoření takovéhoto slovníku bylo nereálné. Pokud bych použil blok o velikosti 64 bitů - odpovídající slovník by měl velikost 2^{64} slov.

2.2.1 DES

Algoritmus DES (Data Encryption Standard) byl šifrovacím standardem po více než 20 let. Je to pravděpodobně nejznámější bloková šifra a o DES bylo napsáno více prací než na jakékoliv jiné téma v kryptografii. I když je tento algoritmus v současnosti už zastaralý, je stále ještě používán. Jelikož byl navržen v 70. letech, není divu že nedokáže odolávat moderním kryptoanalytickým útokům.

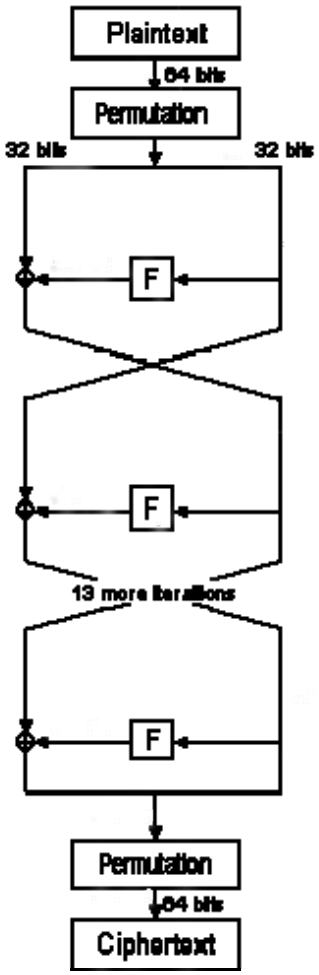
V současné době má algoritmus DES spíše historický význam, a proto zde nebude popisován podrobněji. Jelikož se stal inspirací pro řadu v současnosti používaných algoritmů, je vhodné se s ním alespoň seznámit.

DES je symetrický algoritmus. Pracuje s bloky o velikosti 64 bitů. Velikost klíče je 56 bitů, v některých případech se udává délka klíče 64 bitů. V takovém případě se vždy nejnižší bit v bajtu považuje za lichou paritu od horních sedmi bitů. Jelikož několik klíčů je považováno za tzn. slabé klíče, doporučuje se je během výběru vyloučit, jelikož veškerá bezpečnost šifry je založena na síle klíče.

Vlastní šifrovací algoritmus se skládá z opakování dvou operací, které se provádí v každém cyklu (viz Obr. 4.):

- Substituce
- Permutace

Algoritmus DES má 16 cyklů, každý cyklus sestává z jednoduchých aritmetických operací. Vlastní operace jsou však prováděny s 32-bitovými bloky, které vzniknou rozdělením 64-bitového bloku na dvě části. Tyto subbloky se znovu spojí až po ukončení 16-tého cyklu. Potom je celý 64-bitový blok podroben konečné transformaci.

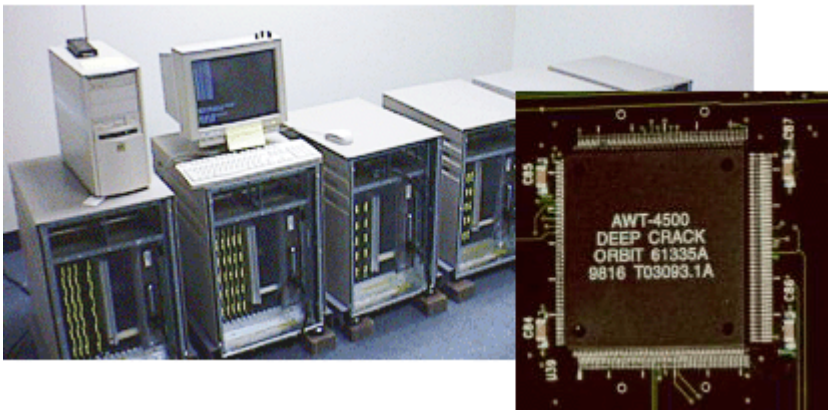


Obr. 4. Schéma DES

Kryptoanalýza algoritmu DES

Největší problém algoritmu DES je na dnešní poměry příliš krátký klíč. K nalezení klíče hrubou silou je třeba vyzkoušet v průměrném případě 2^{55} možností.Úspěšně na něj byla použita lineární i diferenciální kryptoanalýza.

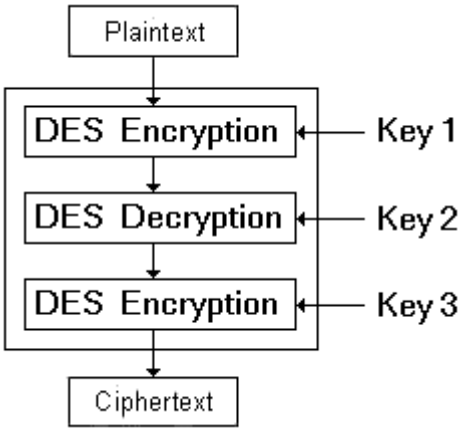
Aby se ilustrovala možnost sestrojení luštícího stroje, byl v roce 1998 sestrojen DES-Cracker[3] (viz. Obr. 5.). Principiálně jednoduchý stroj v ceně asi 250 milionů dolarů je schopen vyzkoušet všechny možné klíče kterých je 2^{56} na daném šifrovaném textu. Obsahuje 29 desek, každá z nich má 64 čipů, které zkouší klíče z různých částí klíčového prostoru. Celkem se dosahuje rychlosti 90 miliard zkoušek klíčů za sekundu, což umožňuje prohledat celý klíčový prostor za 9 dní. Naposledy byl DES-Cracker použit v rámci soutěže DES-Challenge II, kdy nalezl klíč na 22 hodin.



Obr. 5. DES-Cracker

TripleDES

Jedná se o trojnásobné použití DES s třemi obecně různými klíči ($3 \cdot 56 = 168$bitový klíč). I když DES byla zlomena hrubou silou, TripleDES (3DES) se považuje za spolehlivou, protože klíč je dostatečně dlouhý a teoretickým slabinám (slabé klíče) se dá předcházet. Schéma 3DES viz. Obr. 6.



Obr. 6. Schéma 3DES

[3] http://www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/

2.2.2 Blowfish

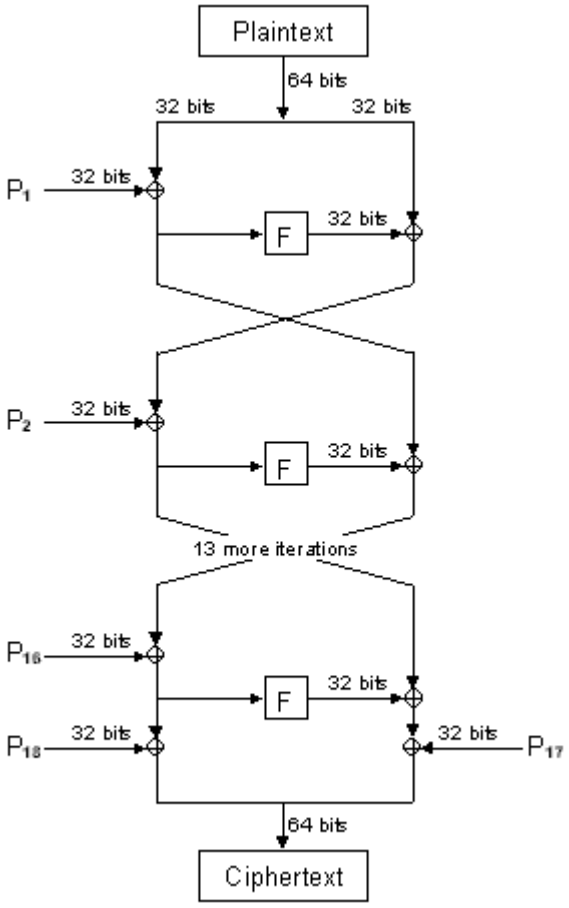
Autorem algoritmu Blowfish je B. Schneier, který jej publikoval v roce 1993. Tento algoritmus není patentován a je volně šiřitelný. Jde o velmi rychlý a jednoduchý algoritmus.

Šifra pracuje s bloky o velikosti 64 bitů a se subbloky o velikosti 32 bitů. Maximální délka klíče je 448 bitů. Schéma Blowfish je na obrázku 7.

Při inicializaci algoritmu je vytvořeno 1042 32-bitových polí. V každém kroku nahra-zujeme vždy 64 bitů tohoto pole, tak že pro náhradu celého pole je třeba 1042 / 2 = 521 kroků, které jsou modifikovány zadaným klíčem a šifrovaný samotným algoritmem Blowfish. Pomocí těchto polí následně probíhá šifrování textu po 64 bitech. Na každých 64 bitů vstupního textu je 18-krát aplikován algoritmus Blowfish. Výstupem je zašifrovaný text.

Při šifrování se požívají tyto matematické operace:

- XOR -> ⊕
- Sčítání modulo 2^{32} -> [+]



Obr. 7. Schéma Blowfish

Šifrování a dešifrování:

Vstupem (označujme ho *x*) je 64-bitové slovo, které se rozdělí na dvě 32-bitová slova *xL* a *xR* (*xL* zahrnuje bity 32..63, *xR* pak bity 0..31 vstupu *x*). Šifrování probíhá v 16-ti rundách, což naznačuje následující funkce.

```
// vstup: 64-bitové slovo, které se rozdělí na dvě 32-bitová slova xL a
// xR
```

```
// výstup spojení xL a xR do výstupního 64-bitového bloku zašifrovaného
// textu

public string BlowFish(ref string xL,ref string xR)
{
    for(int i = 0; i < 16; i++)
    {
        xL = Xor(xL,PBoxy[i]);
        xR = Xor(F(xL),xR);
        Swap(ref xL,ref xR);
    }

    Swap(ref xL,ref xR);
    xR = Xor(xR,PBoxy[16]);
    xL = Xor(xL,PBoxy[17]);

    return xL + xR;
}
```

Dešifrování probíhá přesně opačným způsobem jak je vidět v následující funkci.

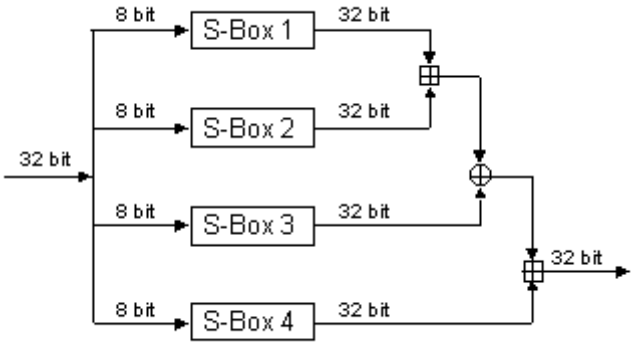
```
// vstup: 64-bitové zašifrované slovo, které se rozdělí na dvě 32-bitová
// slova xL a xR
// výstup: spojení xL a xR do výstupního 64-bitového bloku dešifrovaného
// textu

public string DeBlowFish(ref string xL, ref string xR)
{
    Swap(ref xL,ref xR);
    xR = Xor(xR,PBoxy[17]);
    xL = Xor(xL,PBoxy[16]);

    for(int i = 15; i >= 0; i--)
    {
        Swap(ref xL,ref xR);
        xR = Xor(F(xL),xR);
        xL = Xor(xL,PBoxy[i]);
    }

    return xL + xR;
}
```

Funkce F rozdělí 32-bitové vstupní slovo na čtvrtiny *a, b, c, d*. Tyto části po řadě představují jednotlivé byty (tj. 8 bitů) vstupního slova zleva doprava (tj. *a* představuje bity 24..31, *b* bity 16..23, atd.) a používají se jako indexy do S-boxů. Výstupní hodnota funkce je pak definována následovně (grafické vyjádření viz Obr. 8.).



Obr. 8. Schéma funkce F

Blowfish používá velký počet podklíčů, které musí být vypočteny ze zadaného klíče ještě před samotným šifrováním, resp. dešifrováním dat. Podklíče jsou uloženy celkem v pěti polích. První pole, označované jako P-pole nebo P-box, má celkem 18 32-bitových položek, dále označovaných P1, P2,…, P18. Zbývající pole jsou označována jako S-pole nebo S-boxy. Každý S-box má 256 32-bitových položek. Pokud budeme pracovat s S-boxem *i* (*i* = 1, 2, 3, 4), pak jednotlivé položky budeme označovat S*i*,0, S*i*,1,…, S*i*,255.

Generování podklíčů:

1. Inicializace P-boxu a S-boxů pomocí pevně definovaného řetězce, který tvoří desetinná část čísla π, nebo generátorem náhodných čísel. Inicializace probíhá v tomto pořadí: P1, P2,…, P18, S1,0, S1,1,…, S1,255, S2,0, S2,1,…,S2,255, S3,0, S3,1,…, S3,255, S4,0, S4,1,…, S4,255.
2. XOR P1 s prvními 32 bity klíče, XOR P2 s dalšími 32 bity klíče, a tak pokračujeme dále pro všechny bity klíče (což nám při nejdelším možném klíči vyjde až na P14). Pak opakujeme stejný postup pro zbývající položky P-boxu s cyklickým procházením bitů klíče.
3. Nyní vezmeme podklíče vytvořené v P-boxu pomocí předchozích kroků a aplikujeme Blowfish-algoritmus šifrování na nulový řetězec (tj. 64-bitové vstupní slovo pro algoritmus má všechny bity nulové).
4. Výsledek předchozího kroku (tj. 64-bitové výstupní slovo z algoritmu) použijeme k náhradě 64 bitů tvořených P1 a P2.

5. Výstup kroku (3) zašifrujeme pomocí Blowfish algoritmu (nyní již s pozměněnými podklíči v P1 a P2).
6. Nahradíme P3 a P4 výstupem z kroku (5).
7. Tímto způsobem (tj. šifrový výstup Blowfish algoritmu použijeme k náhradě dalších podklíčů a znovu zašifrujeme) pokračujeme v nahrazování dalších podklíčů v P-boxu a pak v jednotlivých S-boxech.

Míru dosažené bezpečnosti lze regulovat délkou použitého klíče. Rovněž lze omezit počet kol šifrovacího procesu. Snížení počtu kol vede k jistému snížení odolnosti vůči kryptoanalýze, výhodou je ovšem vyšší rychlost šifrování. Naopak se nezdá, že by další zvyšování počtu kol mělo zásadní vliv na zvýšení bezpečnosti algoritmu. Výhodou je , že algoritmus není licencován a je zdarma použitelný.

Kryptoanalýza algoritmu Blowfish

Dešifrování textu zašifrované zprávy bez známosti klíče se nazývané kryptoanalýza algoritmu. Tento hrubý silový útok se sestává ze zkoušení všech možných hodnot z klíčů dokud není nalezen ten správný. Bezpečnost tohoto algoritmu je taky ovlivněna proměnlivou délkou klíče. Klíč má délku až 448 bitů. 448- bitový šifrovací klíč je 2,1 x 1096krát silnější než 128- bitový klíč.

V současnosti není znám lepší způsob kryptoanalýzy, než hrubou silou. V takovém případě by bylo třeba vyzkoušet při maximální délce klíče 2^{448} (2^{447} v průměrném případě) možných kombinací.

2.2.3 IDEA

IDEA (International Data Encryption Algorithm) je bloková šifra založená na moderní myšlence kombinování matematických operací z různých algebraických skupin. Jedná se o vylepšenou verzi šifry PES (Proposed Encryption Standard), která byla upravena tak, aby byla odolnější proti moderním kryptoanalytickým útokům. Při návrhu algoritmu byl také brán ohled na efektivní implementaci v HW i SW. IDEA je iterační šifra sestávající se z 8 identických cyklů následovaných konečnou transformací, pracuje s bloky o velikosti 64 bitů a se subbloky o velikosti 16 bitů. Velikost klíče je 128 bitů. Stejný algoritmus je použit pro šifrování i dešifrování. Jelikož mají všechny operace prováděné s bloky velikost 16 bitů, je algoritmus IDEA efektivní i na 16-bitových procesorech. Základem šifry jsou následující tři matematické operace:

- Logická funkce XOR, označovaná symbolem $\oplus$
- Sčítání modulo 2^{16}, označovaná symbolem [+]
- Násobení modulo $2^{16} + 1$, označovaná symbolem $\odot$

Algoritmus je uspořádaný tak, že výstup získaný z jedné matematické operace není nikdy použít jako vstup do operace stejného typu. Všechny při operace jsou nekompatibilního typu. To znamená, že jsou:

Nedistributivní

$$a[+](b \odot c) \neq (a[+]c) \odot (a\ [+]\ c)$$

Neasociativni

$$a[+](b \oplus c) \neq (a[+]b) \oplus c$$

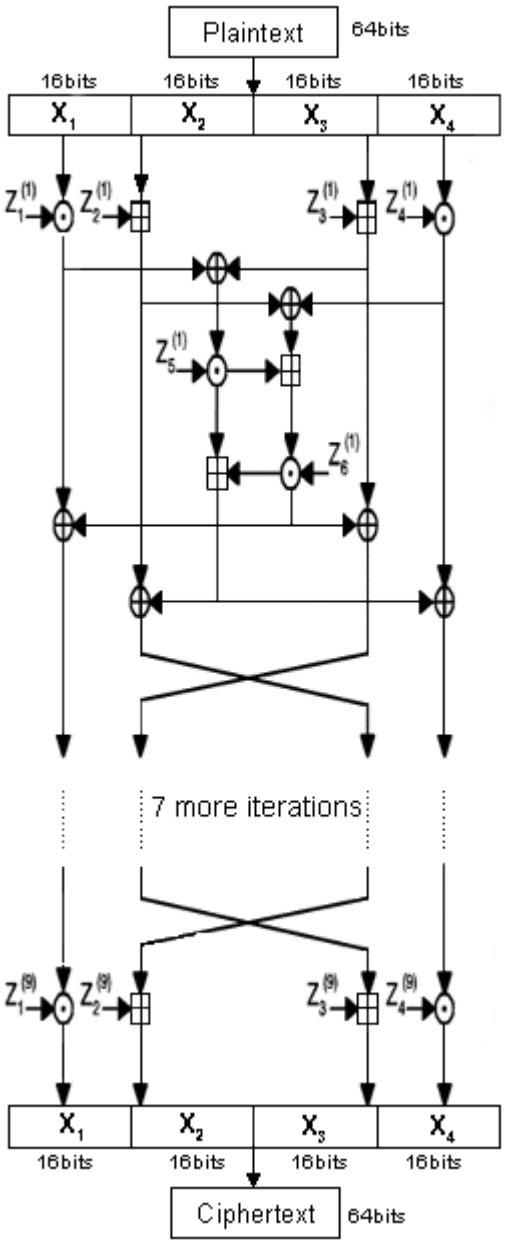
Šifrování probíhá následovně:

64-bitový blok dat X je rozdělen na čtyři 16-bitové subbloky X_1, X_2, X_3, X_4. Tyto bloky tvoří vstup prvního cyklu. V každém cyklu jsou subbloky xorovány, sčítány a násobeny mezi sebou navzájem a s 16-ti bitovými subklíči. Vždy mezi cykly jsou zaměněny druhý a třetí subblok.

Během celého algoritmu je použito 52 subklíčů (6 pro každý cyklus a 4 pro konečnou transformaci). Na začátku je 128-bitový klíč K rozdělen na 8 16-bitových subklíčů Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7, Z_8, přičemž v každém cyklu jsou použity jen subklíče Z_1 až Z_6. Po ukončení každého cyklu je klíč K pootočen o 25 míst na klíč Z($z_j = z_{(i+25) \bmod 128}$, kde i je index bitu) a Z je pak znovu rozdělen na 8 subklíčů. V každém cyklu se provádí následující operace:

1. vynásob X_1 a Z_1
2. sečti X_2 a Z_2
3. sečti X_3 a Z_3
4. vynásob X_4 a Z_4
5. xoruj výsledek z kroků (1) a (3)
6. xoruj výsledek z kroků (2) a (4)
7. vynásob výsledek z kroku (5) a Z_5
8. sečti výsledek z kroku (6) a (7)
9. vynásob výsledek kroku (8) a Z_6
10. sečti výsledek kroků (1) a (9)
11. xoruj výsledek kroků (1) a (9)
12. xoruj výsledek kroků (3) a (9)
13. xoruj výsledek kroků (2) a (10)
14. xoruj výsledek kroků (4) a (10)

Výstupem cyklu jsou čtyři subbloky, které jsou výsledkem kroků (11) a (12), (13) a (14). Subbloky (12) a (13) se mezi sebou zamění a výsledné 4 subbloky tvoří vstup do dalšího cyklu. Po posledním osmém cyklu se provede konečná transformace:



Obr. 9. Schéma IDEA

1. vynásob X_1 a Z_1
2. sečti X_2 a Z_2
3. sečti X_3 a Z_3
4. vynásob X_4 a Z_4

Nakonec jsou 4 výsledné subbloky spojeny a tvoří 64-bitový zašifrovaný text. Schéma šifrování viz. Obr. 9.

Pro dešifrování se používá stejný algoritmus jako pro šifrování. Vstupem je zašifrovaná zpráva a klíč Z, použití klíče je však odlišné. Nejprve se spočítají všechny šifrovací subklíče $Z_i^{(c)}$ kde i je číslo subklíče a c je pořadí cyklu. Z těchto klíčů se vypočtou dešifrovací klíče $Z_i^{(c)}$ podle následující tabulky číslo 4.

Tab. 4. IDEA - Výpočet dešifrovacích klíčů

cyklus c	$\mathbf{Z_1^{(c)}}$	$\mathbf{Z_2^{(c)}}$	$\mathbf{Z_3^{(c)}}$	$\mathbf{Z_4^{(c)}}$	$\mathbf{Z_5^{(c)}}$	$\mathbf{Z_6^{(c)}}$
c = 1	$Z_1^{(8)}$	$-Z_2^{(9)}$	$-Z_3^{(9)}$	$Z_4^{(8)}$	$Z_5^{(8)}$	$Z_6^{(8)}$
2 <= c <= 8	$Z_1^{(9-c)}$	$-Z_3^{(10-c)}$	$-Z_2^{(10-c)}$	$Z_4^{(9-c)}$	$Z_5^{(9-c)}$	$Z_6^{(9-c)}$
c = 9	$Z_1^{(0)}$	$-Z_2^{(1)}$	$-Z_3^{(1)}$	$Z_4^{(0)}$	-	-

Kryptoanalýza algoritmu IDEA

Klíč má délku 128 bitů. V současnosti není znám lepší způsob kryptoanalýzy, než hrubou silou. V takovém případě by bylo třeba vyzkoušet 2^{128} (2^{127} v průměrném případě) možných kombinací. Kdybychom použily počítač, který by dokázal vyzkoušet 10^9 klíčů za sekundu a takových počítačů kdybychom měli 10^9, stále by výpočet trval 10^{13} roků.

Potenciálním slabým místem algoritmu by mohla být malá velikost bloku (64 bitů). Další komplikací je to, že algoritmus je licencován a je zdarma použitelný pouze pro nekomerční aplikace. Přesto se těší velké oblibě. Za svou popularitu vděčí především také tomu, že je použit v populárním programu PGP[4].

[4] http://www.pgp.com/

2.2.4 AES

AES[5] (Advanced Encryption Standard) je bloková šifra, na kterou bylo vypsáno v roce 1997 výběrové řízení americkým standardizačním úřadem a měla nahradit zastaralou šifru DES. Toto výběrové řízení vyhrál algoritmus Rijndael. Jako AES byl schválen americkým Národním úřadem pro standardizaci (NIST) s účinností od května 2002. Autory jsou belgičtí kryptologové Joan Daemen a Vincent Rijmen. AES je 128-bitová šifra, která podporuje tři délky klíče: 128, 192 a 256 bitů. Očekává se že bude mít životnost 20 až 30 let a bude masově používán po celém světě.

AES je interaktivní šifra, přičemž počet iteračních kol se provádí podle délky zvoleného klíče Nr = Nk + 6, kde Nr je počet iterací a Nk je počet 32-bitových slov klíče (128, 192, 256). Algoritmus pracuje s prvky Galoisova tělesa GF(2^8) a s polynomy $b_7x^7 + ... + b_1x^1 + b_0$ a operace násobení bajtů odpovídá násobení těchto polynomů modulo $m(x) = x^8 + x^4 + x^3 + x^1 + 1$.

Šifra je považována za bezpečnou, jelikož její výběr byl velice důkladný a její bezpečnost nebyla do dnešního dne zpochybněna. K této bezpečnosti přispívá hlavně velká délka klíče, která odolá útoku hrubou silou po dlouhá léta. Čeká se proto masové rozšíření této šifry po celém světě, jelikož nebude důvod několik desetiletí měnit systémy pracující s utajenými informacemi.

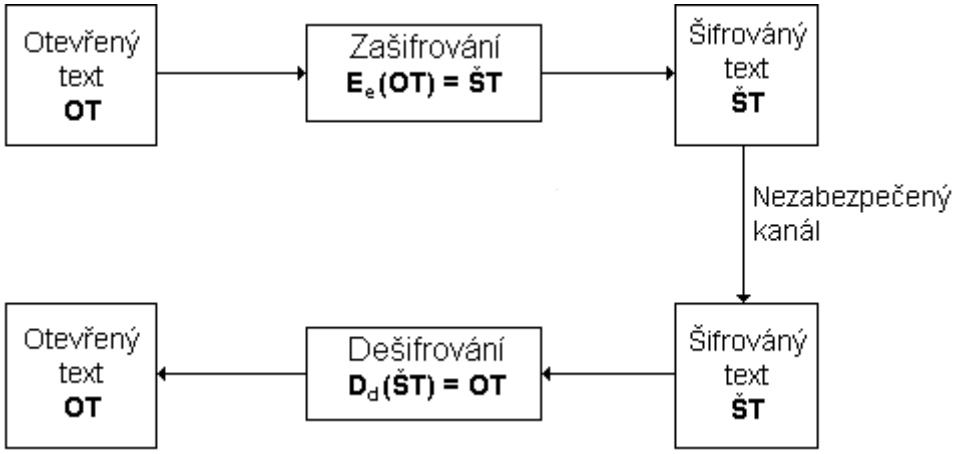
[5] http://csrc.nist.gov/CryptoToolkit/aes/

3 ASYMETRICKÉ ŠIFROVÁNÍ

Tato kapitola pojednává o principech asymetrických kryptosystémů a blíže popisuje asymetrický algoritmus RSA. Jako alternativa je uváděn kryptosytém na bázi eliptických křivek. V kapitole bylo čerpáno z [1] a [5].

Definice: Asymetrická šifra je taková šifra, kde pro skoro všechna $k \in K$ nelze z transformace pro zašifrovaní E_k určit transformaci pro dešifrování D_k.

V praxi je u asymetrických šifer klíč k tajným nastavením, z kterého se vhodnou transformací G vygeneruje dvojice parametrů (e,d), které se nazývají po řadě veřejný (e) a privátní (d) klíč. Ty potom parametrizují transformace zašifrování a dešifrování, takže pro jednoduchost nenapíšeme E_k a D_k, ale přímo E_e a D_d. Schéma šifry viz. Obr. 10.



Obr. 10. Schéma asymetrického šifrování

Veřejný klíč je všeobecně komukoliv dostupný. Tímto klíčem lze pouze zašifrovat zprávu pro určitého uživatele. Tajný klíč má každý u sebe schovaný a určitým způsobem chráněný proti ukradení (heslem, na čipové kartě, na magnetické kartě). Tímto tajným klíčem lze provádět dešifrování přijatých zpráv.

V současné době se asymetrické kryptosystémy používají pro:

- výměnu tajných klíčů symetrické kryptografie
- digitální podpisy
- pro šifrování

3.1 RSA

RSA[6] byl objeven roku 1977 a jeho autoři jsou Ron Rivest, Adi Shamir a Joe Adleman – odtud RSA. Systém je založen na teoreticky jednoduché úvaze: Je snadné vynásobit dvě dlouhá (100-místná a vícemístná) prvočísla, ale bez jejich znalosti je prakticky nemožné zpětně provést rozklad výsledku na původní prvočísla. Součin těchto čísel je tedy veřejný klíč. Přitom obě prvočísla potřebujeme pro dešifrování.

Vzhledem k tomu, že není znám rychlý algoritmus na faktorizaci velkého čísla, je algoritmus RSA bezpečný. Velkým problémem není jen faktorizace[7] čísla *N*, ale najít prvočísla *p* a *q*, potřebné k tvorbě klíčů. Najít dostatečně velké prvočíslo je dosti těžké (resp. pomalé), proto se hledají čísla, která jsou prvočísly s velmi velkou pravděpodobností.

Algoritmus lze rozdělit na dvě části:

- Vygenerování páru veřejný-soukromý klíč
- Šifrování a dešifrování

3.1.1 Postup šifrování

Vygenerování páru veřejný-soukromý klíč

Pro vygenerování klíčů potřebujeme dvě různá, dostatečně velká prvočísla. Označíme si je *p, q*. V praxi se používají prvočísla *512* až *4096* bitů velká, někdy i delší. Vypočteme

$$N = p \cdot q$$

Prvočísla *p, q* by měla být volena tak, aby byla přibližně stejné délky, ale zároveň aby byla dostatečně odlišná. V případě, že by jsme zvolili čísla *p, q* tak, že $p \approx q$, potom by bylo pro útočníka snadné faktorizovat n, neboť $p \approx q \approx \sqrt{n}$. Dále si zvolíme číslo e takové, že

$$GCD\ (e, (p - 1) \times (q - 1)) = 1$$

[6] http://www.rsasecurity.com/

[7] Faktorizace je rozklad čísla na dvě prvočísla, jejichž součin je rozkládané číslo.

Jelikož bude číslo *e* použito ve veřejném klíči, není důvod toto číslo tajit. Dokonce existují doporučení, že číslo *e* by mělo být voleno některé z čísel *3, 17, 65537*. Tato čísla mají tu vlastnost, že jsou to prvočísla a jejich binární zápis obsahuje pouze dvě jedničky, což značně zjednodušuje výpočet. Nakonec vypočteme pomocí Euklidova algoritmu dešifrovací klíč *d*

$$d = e^{-1} \bmod ((p - 1) \times (q - 1))$$

podmínka nesoudělitelnosti *e* a $(p - 1) \cdot (q - 1)$ je nutná proto, aby existovalo právě jedno řešení. Tento zápis je ekvivalentní nalezení takového *d*, že platí

$$(e \cdot d) \bmod ((p - 1) \cdot (q - 1)) = 1$$

Nyní prvočísla *p, q* nebudeme nikdy potřebovat a proto je bezpečně zničíme. Čísla *n* a *d* tvoří soukromý klíč, *n* a *e* je veřejný klíč.

Šifrování a dešifrování

Nejprve zprávu *M*, kterou chceme šifrovat rozdělíme po řadě na $m_1, m_2, \ldots, m_k$ tak, že

$M = m_1, m_2, \ldots, m_k$ a pro $1 \leq i \leq k$ platí vztah

- Zašifrovaný text $C = c_1 c_2 \ldots c_k$, získáme $c_i = m_i^e \bmod n$
- Dešifrovaný text m získáme $m_i = c_i^d \bmod n$

3.1.2 Demonstrační příklad

Předpokládejme, že chceme zašifrovat zprávu *M = 123456789*. Zvolíme si dvě prvočísla

$$p = 29$$

$$q = 19$$

Vypočteme modulus

$$n = 551$$

Zvolíme číslo *e* tak, aby platilo $GCD(e, 28 \cdot 18) = 1$. Této podmínce vyhovuje třeba číslo 17.

$$e = 17$$

Dále potřebujeme vypočítat číslo *d*

$$d = 17^{-1} \bmod (28 \cdot 18)$$

použitím rozšířeného euklidova algoritmu získáme

$$d = 89$$

Nyní rozdělíme zprávu *M* na části m_i tak $m_i < n$

$$m_1 = 123;\ m_2 = 456;\ m_3 = 78;\ m_4 = 9$$

a požitím šifrovacího algoritmu vyjádříme

$$c_1 = 123^{17} \bmod 551$$

$$c_2 = 456^{17} \bmod 551$$

$$c_3 = 78^{17} \bmod 551$$

$$c_4 = 9^{17} \bmod 551$$

po výpočtu získáme

$$c_1 = 169;\ c2 = 19;\ c_3 = 257;\ c_4 = 207$$

pro dešifrování provedeme

$$m_1 = 169^{89} \bmod 551$$

$$m_2 = 19^{89} \bmod 551$$

$$m_3 = 257^{89} \bmod 551$$

$$m_4 = 207^{89} \bmod 551$$

a pro vypočtení skutečně dostaneme původní zprávu.

$$m_1 = 123;\ m_2 = 456;\ m_3 = 78;\ m_4 = 9$$

I přesto, že jsme zvolili velice malá prvočísla *p,q* i exponent *e* výpočet byl značně náročný. Dále je zřejmé, že volba malého exponentu může usnadnit výpočet při šifrování, ale jelikož se při dešifrování exponent *e* nepoužívá, nemá na výpočetní náročnost při dešifrování vliv.

3.1.3 Implementace RSA

Největší problém při implementaci algoritmu RSA zůstává jeho výpočetní náročnost. I přes rostoucí výkon výpočetní techniky je tento algoritmus neúnosně pomalý, což značně omezuje jeho použití v praxi. Důvodem této náročnosti je aritmetika s extrémně vysokými čísly. Výpočet je možno usnadnit vhodným zvoleným exponentu *e*, jak již bylo naznačeno dříve. Takové zrychlení je ale malé.

3.1.4 Generování prvočísel

Generování prvočísel je podstatnou částí algoritmu RSA a jejich volba značně ovlivňuje bezpečnost celého kryptosystému. Obě prvočísla by měla být přibližně stejné délky ale zároveň by měla být dostatečně odlišná. Dále by měla splňovat určitá doporučení, která znesnadní použití existujících faktorizačních algoritmů. Doporučuje se používat tzn. silná prvočísla, což jsou prvočísla, které mají následující vlastnosti :

- prvočíselný rozklad *čísel* $p - 1$, $q - 1$, $p + 1$ a $q + 1$ by neměl obsahovat malá čísla
- totéž platí pro $p - 2$ a $q - 2$
- $p - 1 / 2$ a $q - 1 / 2$ by měla být prvočísla

Rozhodnout, zde je nějaké číslo prvočíslo nebo číslo složené je nesrovnatelně jednoduší, než faktorizovat složené číslo. Na tomto rozdílu je založena bezpečnost mnoha kryptosystémů. Existují efektivní algoritmy na generování náhodných k-bitových prvočísel. Obecný algoritmus na generování k-bitových prvočísel pracuje na následujícím principu:

1. vygeneruj náhodné k-bitové prvočíslo p
2. nastav první a poslední bit na hodnotu 1 (1 na prvním bitu zaručí, že číslo bude právě k-bitové s 1 na konci zaručí, že číslo bude liché)
3. pro všechna prvočísla $i < 2000$ zkontroluj jestli platí $GCD(p,i) = 1$. V případě splnění podmínky jdi na bod 4, v případě nesplnění podmínky jdi na bod 1.
4. použij některý ze speciálních pravděpodobnostních algoritmů

Existuje celá řada pravděpodobnostních algoritmů na testování prvočíselnosti. Určují zda je dané číslo prvočíslo s určitou pravděpodobností. Pravděpodobnost, že složené číslo bude označeno za prvočíslo lze snížit na libovolně nízkou hodnotu tím, že algoritmus libovolně-krát opakujeme. Jeli úspěšnost algoritmu 50%, tak už po 10 opakováních se pravděpodobnost omylu sníží na 0,01%.

V současné době je největší známé prvočíslo $2^{30402457} - 1$ nalezené vědci Curtis Cooper a Steven Boone z univerzity ve Warrenburgu v americkém státě Missouri s pomocí sítě 700 propojených osobních počítačů matematických nadšenců. Na jednom běžném osobním počítači by výpočet takové cifry trval 4500 let.

3.1.5 Bezpečnost RSA

Existuje mnoho důvodů věřit, že algoritmus RSA je bezpečný. Je dokázáno, že získat soukromý klíč z veřejného je stejně složité jako faktorizovat n. Bezpečnost celého kryptosystému spočívá v tom že k získání dešifrovacího exponentu d potřebujeme znát $(p - 1) \cdot (q - 1)$ a to nezjistíme bez faktorizace n (předpokládáme, že potencionální útočník má číslo n, neboť je součástí veřejného klíče). Náhlý pokrok v teorii čísel, který by umožnil faktorizaci velkých čísel by vážně ohrozil bezpečnost algoritmů, které jsou založeny na tomto problému. Na druhou stranu je třeba říci, že algoritmus RSA má za sebou mnoho let kryptoanalýzy je opodstatnělé se domnívat, že i do budoucnosti zůstane bezpečný. Je třeba ovšem zvolit bezpečná veliká prvočísla p a q.

Další možností narušení bezpečnosti RSA je náhlí pokrok ve výpočetní technice, který by umožnil faktorizovat velká čísla.

Otázku bezpečnosti algoritmu RSA však nelze redukovat pouze na diskusi o délce klíče. Bezpečnost závisí i na správné implementaci a mnoha dalších detailech. Potenciální kryptoanalytický útok bude vždy cílen na nejslabší místo celého kryptosystému a tím je zpravidla nevhodná implementace.

3.2 Eliptické křivky

Kryptografické systémy na bázi eliptických křivek navrhli nezávisle na sobě Victor Miller a Neal Koblitz v polovině osmdesátých let dvacátého století (1985). Jedná se o analogii kryptosystému s veřejným klíčem, ve kterých je modulární aritmetika nahrazena operacemi nad eliptickou křivkou (ECC).

Následující popis byl čerpán z [13] a [16]. Pro bližší seznámení s tímto kryptosystémem odkazuji přímo na uvedené zdroje.

3.2.1 Teorie eliptických křivek

Eliptická křivka (E) je dána obecnou rovnicí

$$y^2 = x^3 + a \cdot x + b.$$

Operace sčítání

Kryptosystém eliptických křivek je založen na matematických operacích s body ležícími na křivce. Takto definujeme operaci součtu dvou bodů $P + Q$. Součtem vznikne opět bod který leží na křivce E takto:

Spojíme body $P = (x_P, y_P)$ a $Q = (x_Q, y_Q)$ přímkou; ta protne křivku v bodě, který označíme $-R$, a výsledek je bod R, symetrický k $-R$ podle osy x.

Směrnice přímky, která body spojuje body P a Q má rovnici

$$s = (y_Q - y_P) / (x_Q, x_P)$$

a souřadnice bodu $R = (x_R, y_R)$ se vypočítá z rovnice přímky jako

$$x_R = s^2 - x_P - x_Q \quad a \quad y_R = s(x_P - x_R) - x_P$$

V případě že $P = Q$ se spojnice mění v tečnu ke křivce E a její směrnice je rovna

$$s = (3x^2 + a) / (2y_P)$$

Dále byl definován bod v nekonečnu (O) pro případ, že sčítáme body opačně tj. $P = -Q$. Pro tento bod je též definována operace sčítání

$$P + O = P \; ; \; O + O = O \; ; \; -O = O$$

Operace dělení

Operaci dělení definujeme jako násobení inverzním číslem např. x / y je $x \cdot (y^{-1})$, kde $y^{-1} \cdot y = 1$

$$y^{-1} \cdot y \equiv 1 (mod\ p)$$

např. pro $GF(23)$ (viz. další podkapitola) máme $5^{-1} = 14$, jelikož $14 \cdot 5 = 70 = 23 \cdot 3 + 1$, a tudíž $14 \cdot 5 \equiv 1\ (mod\ p)$.

Prvočíselné těleso GF(p)

Abychom mohli šifrovat text potřebujeme se pohybovat v oblasti diskrétních hodnot a nikoliv reálných čísel. Jelikož si nemůžeme dovolit zaokrouhlování, nahradíme těleso reálných čísel jiným tělesem F, v našem případe se budeme zabývat pouze prvočíselným tělesem $GF(p)$, kde p je prvočíslo. Toto těleso obsahuje čísla $\{0,1,..p\text{-}1\}$, kde p je obvykle velmi velké prvočíslo, a operace vněm se provádějí *modulo p*.

Eliptická křivka je definována jako bod v nekonečnu O společně s množinou bodů $P = (x,y)$, kde x a y jsou z tělesa $GF(p)$ a splňují rovnici $y^2 \equiv x^3 + a \cdot x + b$. Koeficienty a, b v rovnici jsou také prvky tělesa $GF(p)$ a musí splňovat podmínku $4a^3 + 27b^2 (mod\ p) \neq 0$.

Definujeme nenulové body $P = (x_P, y_P) \in E$ a také $-P = (x_P, y_P\ mod\ p)$, dále pro všechny body $P \in E$ definujeme $P + -P = O$ a $P + O = P$. Sčítaní dvou stejných nenulových bodů $R = P + P = (x_P, y_P)$, kde směrnice s je rovna

$$s = (3x_P^2 + a) / (2y_P)$$

a souřadnice

$$x_R = s^2 - 2x_P \; ; \; y_R = s(x_P - x_R) - y_P.$$

Sčítání různých nenulových a vzájemně neinverzních bodů $P = (x_P, y_P)$ a $Q = (x_P, y_P)$ definujeme jako $R = P + Q = (x_R, y_R)$, kde směrnice s je rovna

$$s = (y_Q - y_P) / (x_Q - x_P)$$

a souřadnice

$$x_R = s^2 - x_P - x_Q \; ; \; y_R = s(x_P - x_R) - y_P$$

Nakonec se musí doplnit, že všechny operace jsou prováděny modulo *p,* a to i operace sčítání a dělení uvedené výše, pokud je provádíme na křivce nad tělesem.

3.2.2 Příklad výpočtu bodů elipsy nad tělesem

Zvolme $p = 23, a = 1, b = 1$. Protože $4a^3 + 27b^2 \pmod{23} \neq 0$, máme tak eliptickou křivku $E: y^2 = x^3 + x + 1$ nad tělesem $GF(23)$. Její body jsou v tabulce č. 5. Nyní podle definice sečteme body $P + P$, kde $P = (13,16)$. Máme $R = P + P = (x_R, y_R)$, kde $s = (3 \cdot 13^2 + 1) / (2 \cdot 16) = 508/32 = 508 \cdot 18 = 9144 \bmod 23 = 13$. Dále vypočítáme $xR = 13^2 - 13 - 13 = 143 = 5; yR = 13 \cdot (13 - 5) - 16 = 88 \bmod 23 = 19 \Rightarrow R = (5, 19)$

Tab. 5. Body křivky $E: y^2 = x^3 + x + 1$ nad tělesem $GF(23)$			
(0, 1)	(6, 4)	(12, 19)	(0, 22)
(6, 19)	(13, 7)	(1, 7)	(7,11)
(13, 16)	(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)	(3, 13)
(9, 16)	(18, 3)	(4, 0)	(11, 3)
(18, 20)	(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)	*O*

Vypočteme ještě $R = P + P + P = (P + P) + P = 2P + P$. Sčítáme tedy body $(5, 19)$ a $(13, 16)$: $s = (16 - 19) / (13 - 5) = -3 / 8 = -3 \cdot 3 = 14$, $x_R = 14^2 - 5 - 13 = 178 \bmod 23 = 17$, $y_R = 14 \cdot (5 - 17) - 19 \bmod 23 = 20$, takže $3P = (17, 20)$. Stejným způsobem by se počítal bod $4P, 5P$ atd..

Problém diskrétního logaritmu

Pro šifrování a digitální podpisy se využívá tzv. problém diskrétního logaritmu. Pokud z bodu $P = (x_P, y_P) \in E,$ vypočteme postupně $2P, 3P, 4P$ atd., získáme konečný počet bodů (označený $\#E$) na křivce, který se časem zacyklí. Definujeme přirozené číslo r takové, že $r \cdot P = O$. Je jasné, že v posloupnosti $P, 2P, 3P, 4P..$, se vždy nakonec dostaneme do bodu

O. Poté cyklus začíná znovu od bodu *P*. Nejmenší takové *r*, pro něž je $r \cdot P = O$, nazýváme řád bodu *P*, v našem případě bod *(13, 16)* měl řád $r = 7$. V kryptografické praxi volíme takový bod, jehož řád je roven největšímu prvočíselnému rozkladu čísla *#E*.

Při šifrování elektronickém podpisování využíváme velké posloupnosti *r* (jelikož dojde k zacyklení až po r-tém kroku, v praxi je posloupnost dlouhá např. 2^{256}), a to v souvislosti s problémem diskrétního logaritmu.

Zvolíme tajné číslo *k* (privátní klíč) tak aby $r - 1 \geq k \geq 1$ a vypočteme $Q = k \cdot P$. Součástí veřejného klíče bude čtveřice *(E, P, r, Q)*. Problém diskrétního logaritmu je právě úloha, jak z bodů *P* a *Q* určit tajné číslo *k* tak, že $Q = k \cdot P$.

3.2.3 Digitální podpis podle schématu ECDSA

ECDSA (Eliptic Curve Discrete Digital Signature Algorithm) je schéma digitálního podpisu používajících právě eliptické křivky.

Generování dvojice klíčů pro eliptické kryptosystémy

Asymetrické kryptosystémy používají páry klíčů soukromý / veřejný. Soukromý klíč *d* je celé číslo náhodně vygenerované v intervalu $1 < d < r - 1$. Veřejný klíč je bod *Q* na eliptické křivce vypočtený jako $Q = d \cdot P$. Součástí veřejného klíče, který můžeme zveřejnit je čtveřice *(E, P, r, Q)*.

Vytvoření podpisu podle schématu ECDSA

Mějme zprávu *M*.

1. Vybereme jedinečné číslo *k* tak aby $r - 1 \geq k \geq 1$,
2. vypočteme bod $k \cdot P = (x_1, y_1)$ a *číslo* $n = x_1 \bmod r$,
3. jeli $n = 0$, pak postup opakujeme od generování čísla *k* (to je nutné proto, aby v hodnotě *s* byl obsažen privátní klíč),

4. vypočteme $k^{-1} \bmod r$,
5. vypočteme $s = k^{-1}(h(M) + d \cdot n) \bmod r$, kde h je hashovací funkce SHA-1[8],
6. je-li $s = 0$, pak opět jdeme na první bod generování nového k (neexistoval by $s^{-1} \bmod r$, viz. dále proces ověření),
7. podpisem zprávy M je dvojice čísel (n,s).

Ověření podpisu ECDSA

Mějme zprávu M a její podpis (n,s).

1. Získáme veřejný klíč (E, P, r, Q),
2. ověříme, že $w = s^{-1} \bmod r$ a $h(M)$,
3. vypočteme $u_1 = h(M)w \bmod r$ a $u_2 = nw \bmod r$,
4. vypočteme $u_1P + u_2Q = (x_0, y_0)$ a $v = x_0 \bmod r$,
5. podpis je platný právě tehdy, když $v = n$.

V současnosti se staly eliptické kryptosystémy alternativou ke klasickým asymetrickým kryptosytemům. Mají své výhody zejména v rychlosti a menší náročnosti na hardware a software. Nasazení eliptických kryptosytémů se zdá být pomalé. Příčinou může být to, že klasické asymetrické kryptosystémy jsou používány, studovány a známy déle. Avšak výhodou kryptosystémů na bázi eliptických křivek je jejich velká kryptoanalytická bezpečnost vzhledem k danému klíči. Význačně kratší délka klíčů oproti klasickým kryptosystémům vede k menší parametrům systému, a tedy i k větší výpočetní efektivitě algoritmů.

[8] Hašovací funkce je předpis pro výpočet kontrolního součtu (haše) ze zprávy či většího množství dat. Může sloužit ke kontrole integrity dat, k rychlému porovnání dvojice zpráv, indexování, vyhledávání apod. Více o SHA-1 viz. [17]

3.2.4 Bezpečnost eliptických křivek

Stejně jako v jiných systémech s veřejným klíčem, tak i systém eliptických křivek spoléhají na výpočetně těžký úkol. Pokud by byl řád $r = 2^{256}$, pak by útok hrubou silou podle závislosti $(\pi \cdot r / 2)^{1/2}$ trval cca 2^{128} roků což je zhruba na úrovni symetrické šifry s 128bitovým klíčem. Z tohoto důvodu lze pokládat kryptosystém eliptických křivek za bezpečný.

4 ANALÝZA SYMETRICKÝCH A ASYMETRICKÝCH ŠIFER

V této kapitole srovnáme vybrané algoritmy z několika pohledů, a to nejen mezi symetrickými a asymetrickými algoritmy, ale také mezi sebou.

4.1 Komunikace mezi více účastníky

V této oblasti vítězí asymetrické šifry. Mezi N účastníky, kdy chce komunikovat každý účastník s každým utajeně pomocí symetrického algoritmu, je počet tajných klíčů roven $(N \times (N - 1)) / 2$ a tyto klíče je třeba distribuovat pouze příslušným dvěma účastníkům. Pokud však účastníci používají asymetrický algoritmus, je celá situace jednoduší. Každý účastník vlastní dvojici klíčů – veřejný/soukromý. Veřejné klíče jsou vhodným způsobem zveřejněny. Každý účastník zašle tajnou zprávu při použití pouze veřejně dostupné informace. Zaslanou zprávu může dešifrovat pouze příjemce, který vlastní soukromí klíč.

Asymetrické šifry řeší i problém, kdy si nemůžeme vyměnit nezabezpečeným komunikačním kanálem sdílení klíč, který si je potřeba vyměnit pro komunikaci pomocí symetrické kryptografie.

4.2 Bezpečnost

Bezpečnost obou kryptosystémů je závislá především na délce klíče, u většiny dnes používaného symetrického kryptosystému není možné prolomení jiné, než útok hrubou silou na heslo (klíč) a u asymetrického kryptosystému nemožnost faktorizace velkého čísla. U symetrických kryptosystémů nám stačí menší délka klíče než u asymetrické kryptografie. Následující tabulka číslo 6 ukazuje srovnání bezpečnosti různých kryptosytémů podle [7] při různých délkách klíčů:

Tab. 6. Porovnání bezpečnosti

Blokové šifry	RSA	Eliptické křivky
56	417	105
64	682	120
80	1464	149
86	1881	161
109	4047	206

4.3 Rychlost

Asymetrické kryptosystémy jsou několika násobně pomalejší (až 1000krát) než symetrické kryptosystémy. Z tohoto hlediska jasně vyznívá rychlost použití pro symetrické šifry.

Podle [8] byl proveden test rychlosti symetrických algoritmů (viz. Tab. 7.). U každé šifry je uveden počet zašifrovaných bytů dat za sekundu. Test byl proveden na počítači s procesorem Intel Celeron 450MHz. V dnešní době dosáhneme hodnot výrazně vyšších.

Tab. 7. Porovnání rychlosti

Šifra	B/s
Blowfish	9 013 043
DES	7 372 170
IDEA	6 388 278
3DES	2 457 390
AES	11 751322

4.4 Použití

Asymetrické kryptosystémy používají pro:

- výměnu tajných klíčů symetrické kryptografie
- digitální podpisy
- pro šifrování

Symetrická kryptografie je vhodná převážně pro:

- komunikační protokoly
- šifrování velkého objemu dat

Často se používá kombinace symetrických a asymetrických šifer. Např. při výměně tajného klíče k symetrickému kryptosystému po nezabezpečeném komunikačním kanálu použijeme asymetrický kryptosystém. Kombinaci obou kryptosystémů používá také velice populární program pro bezpečnou komunikaci PGP.

PGP

PGP[9] (Pretty Good Privacy) je kombinovaný šifrovací systém. Navenek se jeví jako program s veřejným šifrovacím klíčem, plně využívající asymetrického šifrování. To se však ve skutečnosti používá pouze pro zakódování klíče symetrické šifry, kterou je pak zašifrována samotná zpráva. Digitálním podpisem je kontrolní součet zprávy (hash), který je zašifrován asymetrickou šifrou.

V PGP jsou použity tyto algoritmy: RSA jako asymetrická šifra, IDEA (128 efektivních bitů klíče), TripleDES (168 bitů).

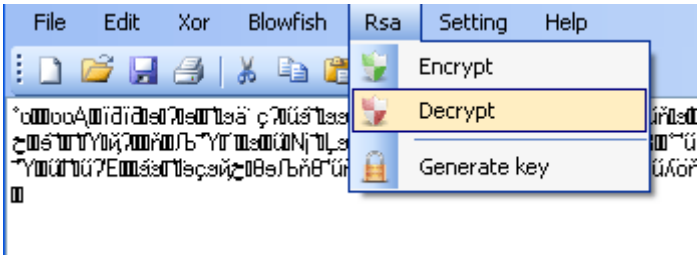
[9] http://www.pgp.com/

II. PRAKTICKÁ ČÁST

5 DEMONSTRAČNÍ APLIKACE – EASYCRYPT

EasyCrypt je demonstrační aplikace, kterou vytvořil autor a je určen k demonstraci šifrování textu. Implementovány jsou následující tři algoritmy:

- Xor, jako zástupce symetrických proudových šifer viz. kapitola 2.1.1
- Blowfish, jako zástupce symetrických blokových šifer viz kapitola 2.2.2.
- RSA, jako zástupce asymetrických šifer viz kapitola 3.1.



Obr. 11. Aplikace Easycrypt

EasyCrypt (viz. Obr. 11.) je naprogramován v jazyku c# pro platformu dot.NET. Algoritmy které vytvořil autor, jsou pouze demonstrační a nejsou optimalizované pro komerční použití.

5.1 Popis aplikace

EasyCrypt má všechny funkce klasického textového editoru pro jednoduchou práci s textem jako je kopírování, vkládání, mazání atd. Dále je možno text načítat a ukládat v textových souborech.

Dále se budeme zabývat nadstandardními šifrovacími funkcemi a některými dalšími funkcemi které s programem souvisí.

V záhlaví v menu se nacházejí následující položky:

- **Xor** – pokud chceme transformovat text pomocí tohoto algoritmu jsme vyzváni k zadání hesla pro transformaci. Stejným heslem se šifruje i dešifruje. Maximální

délka hesla musí být kratší než délka textu. Transformace textu probíhá pro přehlednost na binární úrovni, proto můžeme v položce **Setting** volit binární vstup/výstup podle potřeby. Zdrojový kód viz. Příloha I.

- **Blowfish** - pokud chceme transformovat text pomocí tohoto algoritmu jsme vyzváni k zadání hesla pro transformaci. Jelikož se jedná o symetrickou šifru, stejným heslem se šifrujeme i dešifrujeme. Maximální délka hesla je 448 bitů, bez závislosti na délce šifrovaného textu. Transformace textu probíhá pro přehlednost na binární úrovni, proto můžeme v položce **Setting** volit binární vstup/výstup podle potřeby. Zdrojový kód viz. Příloha II.

- **RSA** - pokud chceme transformovat text pomocí tohoto algoritmu musíme nejprve místo hesla vygenerovat klíčový pár (privátní /soukromý klíč). Veřejný klíč použijeme pro šifrování a soukromý pro dešifrování. Tento klíč vygeneruje tzv. generátor náhodných prvočísel, který je sice v programu naprogramován, ale jen v omezené demonstrační míře. Algoritmus pracuje pouze v 32-bitovými čísly. Zdrojový kód viz. Příloha III.

- **Setting** – algoritmy Xor a Blowfish transformují text na binární úrovni, proto je možné volit binární vstup/výstup podle potřeby.

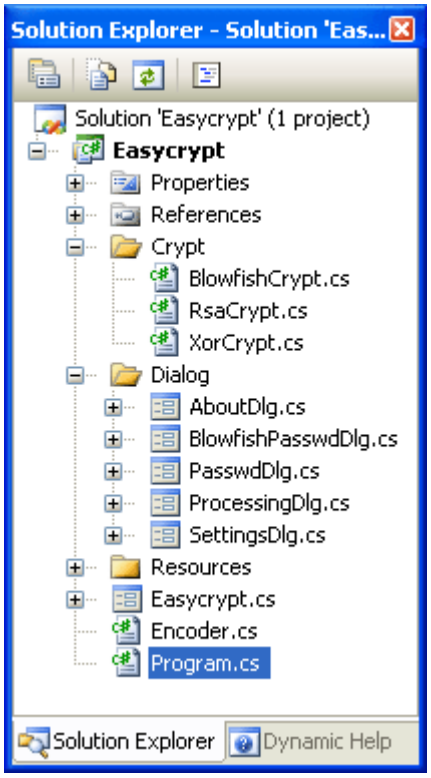
- **About** – obsahuje dialogové okno O programu (About Easycrypt)

Pro lepší pochopení této aplikace odkazuji na předešle teoretické kapitoly, popř. na přílohy jednotlivých šifrovacích algoritmů nebo přímo na celý zdrojový kód aplikace na přiloženém CD jehož strukturu si ukážeme v následující podkapitole.

5.2 Vývojová struktura programu

Aplikace je programovaná ve vývojovým prostředí Microsoft Visual C# 2005 Express Edition[10] pro platformu .NET Framework 2.0[11], bez kterého nelze aplikaci spustit.

Program se skládá z několik tříd tříděné do složek podle příslušnosti Crypt, Dialog, Resources, které můžeme vidět na obrázku 12.



Obr. 12. Solution Explorer

Složka Crypt

Složka Crypt obsahuje tři třídy implementovaných šifrovacích algoritmů

- BlowfishCrypt.cs – implementuje algoritmus Blowfish, jeho přepis viz. Příloha I.
- RsaCrypt.cs – implementuje algoritmus RSA, jeho přepis viz, Příloha II.
- XorCrypt.cs – implementuje algoritmus XOR, jeho přepis viz, Příloha III.

[10] http://msdn.microsoft.com/vstudio/express/visualcsharp/default.aspx

[11] http://msdn.microsoft.com/netframework/

Složka Dialog

Složka Dialog obsahuje všechny autorem vytvořené dialogy

- AboutDlg.cs – dialogové okno O programu (About Easycrypt)
- PasswdDlg.cs – dialogové okno zadaní hesla k algoritmu Blowfish
- BlowfishDlg.cs – dialogové okno zadaní hesla k algoritmu XOR
- SettingsDlg.cs – dialogové okno nastavené programu (Settings)

Složka Resources

Složka Resources obsahuje zdroje aplikace jako jsou obrázky, ikony atd.

Root

V rootu se nachází základní třídy aplikace

- Easycrypt.cs – jedná se o hlavní okno aplikace obsahující ovládací prvky a komponenty. Mimo to se stará o zpracování textu do přijatelné podoby pro jednotlivé šifrovací algoritmy (třídy) v takovém stavu aby jen mohli bez problémů šifrovat nebo dešifrovat. Tzn. např. posílaní algoritmu Blowfish textové bloky o 64 bitech atd..
- Encoder.cs – třída pro převod textu na binární řetězec, tuto třídu využívá algoritmus XOR a Blowfish.

Všechny zdrojové soubory a kompletní řešení aplikace jsou na přiloženém CD.

ZÁVĚR

V současné době je po mobilních telefonech nejčastějším komunikačním prostředkem internet, konkrétně elektronická pošta. Málokdo z uživatelů těchto komunikačních prostředků si uvědomuje jak často používá kryptografii, například když komunikujeme se svojí bankou přes webové rozhraní i mobilní telefon, při vybírání peněz z bankomatu pomocí čipové karty. Pro pohodlí uživatelů je u mnoha těchto systémů šifrovací klíč uložen v chráněném hardware, například v čipové kartě, SIM kartě nebo obecně tzv. tokenech. Tokeny jsou zařízení, která jsou realizovaná malými předměty (do ruky) a mají různý tvar i podobu. Mohou to být přívěsky na klíče, miniaturní infračervené ovladače, čipy v prstenu, tzv. dotykové paměti, čipové karty a pod. Mají tu výhodu, že uživatel si klíč nemusí vůbec pamatovat, a v některých případech ho ani nemusí znát. Budoucnost šifrování a jeho rozmach v České republice patří snadné komunikaci s úřady veřejné správy pomocí elektronických podpisů[12].

V diplomové práci jsou popisovány a diskutovány některé používané moderní symetrické a asymetrické kryptosystémy. Cílem jejich srovnání není určit vítěze mezi nimi, ale určit oblasti, kde je možné a vhodné jednotlivé kryptosystémy použít v praxi. Symetrické šifry se používají u komunikačních protokolů a šifrování velkého objemu dat. Asymetrické šifry mají své použití převážně u bezpečné elektronické korespondence a elektronických podpisů. Jak bylo zmíněno, používají se často tzv. hybridní kryptosystémy, které využívají výhod symetrických i asymetrických šifer dohromady.

Popsané kryptosystémy bezpečně odolávají kryptoanalýze útokem na algoritmus a jeho prolomení. Jediná slabina je tedy správné zvolení bezpečné délky klíče, která je u asymetrických šifer několikrát delší než u symetrických tak, aby odolal útoku hrubou silou. Další možný směr útoku je na nesprávnou implementaci algoritmu. Většina kryptosystémů je postavena na nedostatečných možnostech výpočetní techniky pro řešení kryptoanalýzy hrubou silou (zkoušení všech možností najít správný klíč) u symetrických kryptosystémů a nemožnosti faktorizovat veliké číslo (stovky řádů dlouhé) jako je to u asymetrických kryptosystémů. Proto jsou stále populární šifry ze sedmdesátých let dvacátého století. I

[12] Zákon č. 227/2000 Sb., o elektronickém podpisu a o změně některých dalších zákonů (zákon o elektronickém podpisu)

když od té doby dosáhla výpočetní technika významného pokroku a stala se dosažitelná i pro běžné uživatele, stále nemá dostatečný výkon pro provedení rychlé a úspěšné kryptoanalýzy. Velké bezpečnostní riziko by však mohl přinést vývojový převrat ve výpočetní technice např. v kvantových počítačích, nebo významný pokrok v teorii čísel. Takový pokrok se však v nejbližších 20 až 30 letech nepředpokládá.

SEZNAM POUŽITÉ LITERATURY

[1] GROŠEK, O., PORUBSKÝ, Š. :*Šifrovanie – algoritmy, metody, prax*, Grada, Praha 1993, ISBN 80-85424-62-2 *Algebra 2 – Teorie čísel*

[2] BULANT, M. : *Algebra 2 – Teorie čísel* [online]. Masarykova Univerzita. Brno. [cit. 2006-02-08]. Dostupné na WWW: <http://www.math.muni.cz/~bulik/vyuka/Algebra-2/alg2-print.pdf>

[3] KUČERA, R. : *Algoritmy Teorie čísel* [online]. Masarykova Univerzita. Brno. 2006. [cit. 2006-02-10]. Dostupné na WWW: <http://www.math.muni.cz/~kucera/texty/ATC06.pdf>

[4] SCHNEIER, B. : *Description of a New Variable-Length Key, 64-Bit Block Cipher* [online]. [cit. 2005-12-12]. Dostupné na WWW: <http://www.schneier.com/paper-blowfish-fse.html>

[5] KLÍMA, V. : *Základy moderní kryptologie – Symetrická kryptografie I* [online]. [cit. 2006-03-22]. Dostupné na WWW: <http://www.karlin.mff.cuni.cz/~tuma/nciphers.html>

[6] KLÍMA, V. : *Základy moderní kryptologie – Symetrická kryptografie II* [online]. [cit. 2006-03-22]. Dostupné na WWW: <http://www.karlin.mff.cuni.cz/~tuma/nciphers.html>

[7] LENSTRA A., VERHEUL E.: Selecting Cryptographic Key. *Journal of Cryptology*, 2001

[8] NOVICKÝ, P.: Šifrované filesystémy [online]. *Abíčko*. 2003. [cit. 2006-04-01]. Dostupné na WWW: <http://www.abclinuxu.cz/download/abicko/2003/abicko-2003-08.pdf>

[9] DRMOLA, R. : *Popis šifry Blowfish* [online]. [cit. 2005-12-12]. Dostupné na WWW: <http://bobhy.wz.cz/clanky/blowfish.html>

[10] KLÍMA, V., ROSA, T. : Kryptologie pro praxi – Nejpoužívanější šifry [online]. *Sdělovací technika.* 2003. [cit. 2006-03-18]. Dostupné na WWW: <http://cryptography.hyperlink.cz/2003/ST_2003_11_17_18.pdf >

[11] KLÍMA, V., ROSA, T. : Kryptologie pro praxi – Volba klíče [online]. *Sdělovací technika*. 2004. [cit. 2006-03-18]. Dostupné na WWW: <http://cryptography.hyperlink.cz/2004/ST_2004_07_14_15.pdf >

[12] KLÍMA, V., ROSA, T. : Kryptologie pro praxi – RSA [online]. *Sdělovací technika. 2004*. [cit. 2006-03-18]. Dostupné na WWW: <http://cryptography.hyperlink.cz/2003/ST_2003_11_17_18.pdf>

[13] KLÍMA, V. : Eliptické křivky a šifrování [online]. *Chip*. 2002. [cit. 2006-04-22]. Dostupné na WWW: <http://cryptography.hyperlink.cz/2002/chip-2002-09-134-136.pdf>

[14] KLÍMA, V. : Nová šifra nastupuje [online]. *Chip*. 2002. [cit. 2006-03-11]. Dostupné na WWW: <cryptography.hyperlink.cz/2002/chip-2002-05-142-144.pdf >

[15] KLÍMA, V. : Faktorizace [online]. *Chip*. 2001. [cit. 2006-03-11]. Dostupné na WWW: <http://cryptography.hyperlink.cz/2001/chip-2001-09-176-180.pdf >

[16] OCHODKOVÁ, E. : *Přínos teorie eliptických křivek k řešení moderních kryptografických systémů* [online]. VŠB Ostrava. [cit. 2006-04-22]. Dostupné na WWW: <http://www.cs.vsb.cz/arg/workshop/files/ecc_eli.pdf >

[17] SECURE HASH STANDARD [online]. *FIPS*. 2002. [cit. 2006-04-22]. Dostupné na WWW: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[18] IDEA: Technical Description [online]. [cit. 2006-03-03]. Dostupné na WWW: <http://www.mediacrypt.com/_pdf/IDEA_Technical_Description_0105.pdf>

[19] PRETTY GOOD PRIVACY [online]. [cit. 2006-02-03]. Dostupné na WWW: <http://www.pgp.cz/>

[20] WIKIPEDIA [online]. [cit. 2006-02-03]. Dostupné na WWW: <http://cs.wikipedia.org/>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AES	Advanced Encryption Standard
DES	Data Encryption Standard
ECDSA	Eliptic Curve Discrete Digital Signature Algorithm
FIPS	Federal Information Processing Standards
GCD	Greatest Common Divisor (Největší společný dělitel)
IDEA	International Data Encryption Algorithm
NIST	National Institute of Standards and Technology
PGP	Pretty Good Privacy
XOR	Logická funkce *exkluzivní nebo*

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA PI - ALGORITMUS XOR

```
//výpis ze souboru XorCrypt.cs

using System;

namespace Easycrypt
{
   public class XorCrypt
   {
      // Funkce Xor
      // xoruje dva binarni retezce, a vraci vysledny binarni
      //retezec
      public string Xor(string binaryString, string binaryPasswd)
      {
         string resultString = "";
         int j = 0;
         while(j < binaryString.Length-1)
         {
            for(int i = 0; i < binaryPasswd.Length; i++)
            {
               if(   binaryString[j] == binaryPasswd[i])
                  resultString += "0";
               else
                  resultString += "1";
               if(j < binaryString.Length-1)
               {
                  j++;
               }
               else break;
            }
         }
         return resultString;
      }
   }
}
```

PŘÍLOHA PII - ALGORITMUS BLOWFISH

```
//výpis ze souboru BlowfishCrypt.cs
using System;

namespace Easycrypt
{

   public class BlowfishCrypt
   {
      private static string []PBoxy = new string[18];
      private static string []SBoxy1 = new string[256];
      private static string []SBoxy2 = new string[256];
      private static string []SBoxy3 = new string[256];
      private static string []SBoxy4 = new string[256];
      private string xL = "00000000000000000000000000000000";
      private string xR = "00000000000000000000000000000000";

      public string GetXL
      {
         set
         {
            xL = value;
         }

         get
         {
            return xL;
         }
      }

      public string GetXR
      {
         set
         {
            xR = value;
         }
         get
         {
            return xR;
         }
      }

      // Funkce InitBoxy
      // inicializacni funkce, kdy se PBoxy a SBoxy inicializuji
      // podle zadaneho hesla algoritmem Blowfish
      public void InitBoxy(string passwdBlowFish)
      {
         // pocatecni nastaveni, generovane generatorem nahodnych
                  //  cisel
         PBoxy[0] = "11000100011111111101000011010011";
         PBoxy[1] = "00110010011101001101011111000110";
         PBoxy[2] = "10011000100101000101101000011001";
         PBoxy[3] = "01011111001111100010111001110000";
         PBoxy[4] = "11010000100001001011110011111011";
         PBoxy[5] = "10111100000001001100110011011101";
         PBoxy[6] = "11111111011000011100011000110100";
         PBoxy[7] = "10010010100101011001010111101000";
         PBoxy[8] = "00100100111010110101100011001111";
         PBoxy[9] = "10010100001010001101011000010101";
         PBoxy[10] = "11010110000010011100010101111110";
         PBoxy[11] = "10000001000010110000111000000101";
         PBoxy[12] = "11010111110010001101110110001000";
         PBoxy[13] = "10001101101100010000010110000010";
```

```
PBoxy[14] = "11100000110101100111001100101001";
PBoxy[15] = "10010000010001000011011000100111";
PBoxy[16] = "10110110100010011010110101110011";
PBoxy[17] = "11101101010111101000101001101010";


int z = 0;
for(int i = 0; i < 256; i++)
{
   if(z > 17) z = 0;
   SBoxy1[i] = PBoxy[z];
   SBoxy2[i] = PBoxy[z];
   SBoxy3[i] = PBoxy[z];
   SBoxy4[i] = PBoxy[z];
   i++;
   z++;
   SBoxy1[i] = PBoxy[z];
   SBoxy2[i] = PBoxy[z];
   SBoxy3[i] = PBoxy[z];
   SBoxy4[i] = PBoxy[z];
   z++;
}

int size = passwdBlowFish.Length/32;
string [] passwdArray = new string[18];

int t = 0;

// cyklicke helo delky 576
while(passwdBlowFish.Length < 576)
{
   passwdBlowFish += passwdBlowFish[t];
   t++;
}

int inc = 0;

// rozdeleni hesla po 32 bitech
for(int i = 1; i <= passwdBlowFish.Length; i++)
{
   passwdArray[inc] += passwdBlowFish[i-1];
   if(i % 32 == 0)
   {
      inc++;
   }
}

for(int r = 0; r < 18; r++)
{
   PBoxy[r] = Xor(PBoxy[r],passwdArray[r]);
}

// BlowFish na zbytek podklicu
for(int f = 0; f < 18 ; f += 2)
{
   BlowFish(ref xL, ref xR);
   PBoxy[f] = xL;
   PBoxy[f+1] = xR;
}

for(int d = 0; d < 256 ; d += 2)
{
   BlowFish(ref xL, ref xR);
   SBoxy1[d] = xL;
```

```
      SBoxy1[d+1] = xR;
   }

   for(int d = 0; d < 256 ; d += 2)
   {
      BlowFish(ref xL, ref xR);
      SBoxy2[d] = xL;
      SBoxy2[d+1] = xR;
   }

   for(int d = 0; d < 256 ; d += 2)
   {
      BlowFish(ref xL, ref xR);
      SBoxy3[d] = xL;
      SBoxy3[d+1] = xR;
   }

   for(int d = 0; d < 256 ; d += 2)
   {
      BlowFish(ref xL, ref xR);
      SBoxy4[d] = xL;
      SBoxy4[d+1] = xR;
   }
}

// Funkce sifrovani Blowfish,
// ocekavany vstup: levy a pravy blok bitu
public string BlowFish(ref string xL,ref string xR)
{
    // nuly doplni pozadovanou delku 32 bitu na blok
    // pokud je blok mensi nez 32
   if(xL.Length < 32)
      for(int i = xL.Length; i <= 32; i++)
      {
         xL += "0";
      }
   if(xR.Length < 32)
      for(int i = xR.Length; i <= 32; i++)
      {
         xR += "0";
      }

   for(int i = 0; i < 16; i++)
   {
      xL = Xor(xL,PBoxy[i]);
      xR = Xor(F(xL),xR);

      Swap(ref xL,ref xR);
   }

   Swap(ref xL,ref xR);
   xR = Xor(xR,PBoxy[16]);
   xL = Xor(xL,PBoxy[17]);

   return xL + xR;
}

// Funkce desifrovani
// ocekavany vstup: levy a pravy blok bitu
public string DeBlowFish(ref string xL, ref string xR)
{
   Swap(ref xL,ref xR);
   xR = Xor(xR,PBoxy[17]);
   xL = Xor(xL,PBoxy[16]);
```

```
         for(int i = 15; i >= 0; i--)
         {
            Swap(ref xL,ref xR);

            xR = Xor(F(xL),xR);
            xL = Xor(xL,PBoxy[i]);
         }
         return xL + xR;
      }

      // Funkce F
      // provadi permutaci bloku dat
      private string F(string xL)
      {
         uint XL = ToDecimal(xL);
         uint xLong = ((ToDecimal(SBoxy1[Split(xL,'a')]) +
ToDecimal(SBoxy2[Split(xL,'b')]) % 4294967295) ^
ToDecimal(SBoxy3[Split(xL,'c')])) + ToDecimal(SBoxy4[Split(xL,'d')]) %
4294967295;

         return ToBinary(xLong);
      }

      // Funkce Split
      // urci jake Sboxy se pouziji ve funkci F
      private uint Split(string xL, char size)
      {
         int i = 0;
         string split = "";
         switch(size)
         {
            case 'a': i = 24;
               break;
            case 'b': i = 16;
               break;
            case 'c': i = 8;
               break;
            case 'd': i = 0;
               break;
         }

         for(int j = i ; j < i+8; j ++)
            split += xL[j].ToString();

         return ToDecimal(split);
      }

      // Funkce Swap
      // prehodi levy a pravy blok dat
      private void Swap(ref string xL,ref string xR)
      {
         string temp = xL;
         xL = xR;
         xR = temp;
      }

      // Funkce Xor
      // provede xorovani
      private static string Xor(string xL,string xR)
      {
         string resultString = "";
         int j = 0;
         while(j < xL.Length-1)
```

```
   {

      for(int i = 0; i < xR.Length; i++)
      {
         if(   xL[j] == xR[i])
            resultString += "0";
         else
            resultString += "1";
         if(j < xL.Length-1)
         {
            j++;
         }
         else break;
      }

   }
   return resultString;
}

// Funkce ToDecimal
// prevadi binarni retezec na dekadicke cislo
private static uint ToDecimal(string binaryString)
{
   uint intNumber = 0;
   uint inc = 1;

   for(int z = 1; z <= binaryString.Length; z++)
   {
      if(binaryString[binaryString.Length-z] == '1')
      {
         intNumber += inc;
      }

      inc *= 2;

   }
   return intNumber;
}


// Funkce ToBinary
// prevadi dekadicke cislo na binarni retezec
private string ToBinary(uint decimalNumber)
{
   string str = "";

   while(true)
   {
      if(decimalNumber % 2 == 1)
      {
         str += 1;
      }
      if(decimalNumber % 2 == 0)
      {
         str +=  0;
      }

      decimalNumber /= 2;
      if(decimalNumber == 1)
      {
         str += 1;
         break;
      }
   }
```

```
            int i = str.Length;
            string rstr = "";
            while(i < 32)
            {
               rstr += "0";
               i++;
            }
            rstr += str;
            return rstr;
         }
      }
   }
```

PŘÍLOHA PIII – ALGORITMUS RSA

```
//výpis ze souboru RsaCrypt.cs

using System;
using System.Collections.Generic;
using System.Text;

namespace Easycrypt
{
    class RsaCrypt
    {

        // Funkce PrimeNumeber
        // je nahradou generatoru prvocisel, ktery by mel gererovat
        // prvocisla delky kolem 100 cislic
        private int PrimeNumber()
        {
            int[] primeNumber = new int[] { 11, 13, 17, 19, 23, 29, 31,
37, 41, 43, 47 };
            Random random = new Random();
            int number = random.Next(0, 10);
            return primeNumber[number];
        }

        // Funkce Generate
        // generovani soukromeho a verejneho klice
        public ulong[] Generate()
        {
            ulong n, d, p, q, e;

            p = Convert.ToUInt64(PrimeNumber());
            q = Convert.ToUInt64(PrimeNumber());

            while (p == q)
            {
                q = Convert.ToUInt64(PrimeNumber());
            }

            // hodnota e se nastavuje napevno
            e = 17;

            // vypocet verejneho klice n
            n = p * q;

            // vypocet soukromeho klice d
            d = InverseModulo(e, (p - 1) * (q - 1));

            ulong []arrayKey = new ulong[] {n, d, e};
            return arrayKey;
        }

        // Funkce sifrovani Encrypt
        public string Encrypt(string instr, ulong n, ulong e)
        {
            string outstr = String.Empty;

            for (int j = 0; j < instr.Length; j++)
            {
                ulong t = (ulong)instr[j];
                t = Power(t, e, n);
                outstr += (char)t;
            }
```

```
    return outstr;
}

// Funkce desifrovani Decrypt
public string Decrypt(string outstr, ulong n, ulong d)
{
    //zadejte soukromy klic d:

    string instr = String.Empty;

    for (int j = 0; j < outstr.Length; j++)
    {
        ulong t = (ulong)outstr[j];
        t = Power(t, d, n);
        instr += (char)t;
    }
    return instr;
}

// Funkce Power
// vraci cislo y u ktereho plati y = a^x mod n,
// jinak se rekurzivne vola funkce Power
public static ulong Power(ulong a, ulong x, ulong n)
{
    if (x == 1)
    {
        return (a % n);
    }
    else
    {
        return (((a % n) * (Power(a, x - 1, n))) % n);
    }
}

// Funkce Even
// kontroluje sudost cisla
public static bool Even(ulong number)
{
    if ((number % 2) == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

// Funkce InverseModulo
// k cislum x a y
public static ulong InverseModulo(ulong x, ulong y)
{
    ulong a, b, c, d, u, v, g;

    g = 1;

    while (Even(x) && Even(y))
    {
        x /= 2;
        y /= 2;
        g *= 2;
    }

    u = x;
```

```
    v = y;
    a = 1;
    b = 0;
    c = 0;
    d = 1;

    do
    {
        while (Even(u))
        {
            u /= 2;
            if (Even(a) && Even(b))
            {
                a /= 2;
                b /= 2;
            }
            else
            {
                a = (a + y) / 2;
                b = (b - x) / 2;
            }
        }

        while (Even(v))
        {
            v /= 2;
            if (Even(c) && Even(d))
            {
                c /= 2;
                d /= 2;
            }
            else
            {
                c = (c + y) / 2;
                d = (d - x) / 2;
            }
        }

        if (u >= v)
        {
            u -= v;
            a -= c;
            b -= d;
        }
        else
        {
            v -= u;
            c -= a;
            d -= b;
        }

    } while (u != 0);

    a = c;
    b = d;
    return a;
}

// Funkce GCD
// vrati nejvetsi spolecny delitel dvou cisel
public static ulong GCD(ulong x, ulong y)
{
    ulong g;
```

```
            g = y;

            while (x > 0)
            {
                g = x;
                x = y % x;
                y = g;
            }
            return g;
        }
    }
}
```