ASYMMETRIC-KEY ALGORITHM IN NETWORK WITH LIMITED SOURCES

Radek Fujdiak

Master Degree Programme (2), FEEC BUT

E-mail: xfujdi00@stud.feec.vutbr.cz

Supervised by: Petr Mlýnek

E-mail: mlynek@feec.vutbr.cz

Abstract: This article describes asymmetric-key algorithm for key arrangement in the symmetric cryptography. AES method is used for symmetric cryptography. The implementation is accomplished into limited sources devices (e.g. low-power microcontroler). In next part of this article the asymetric cryptography, Diffie-Hellman (D-H) and Elliptic curve Diffie-Hellman (ECDH) is described, which offers a solution for key arrangement in the symmetric cryptography. Firstly, D-H algorithm simply code is described. Next, the work with big numbers is showed and finally ECDH with small numbers is described.

Keywords: key management, AES, Diffie-Hellman, elliptic curve.

1. ÚVOD

Kryptografie se dělí na dvě části. Na symetrickou kryptografii a na asymetrickou kryptografii. Příkladem asymetrické šifry je například Advenced Encryption Standard (AES), využívající stejné klíče (K_E=K_D) [1]. U symetrických šifer je problémem domluva klíče pro samotné šifrování, většinou jsou tyto klíče uloženy v paměti zařízení. Řešením tohoto problému je nasazení některé z asymetrických šifer. S přihlédnutím k omezenému výkonu mikrokontrolerů je vhodné volit algoritmy využívající eliptické křivky a to z důvodu jejich menší výkonové náročnosti (pro příklad 3072-bitový RSA odpovídá 256-bitové variantě nad eliptickou křivkou) [2]. Dále pak také vzhledem k mezinárodním doporučením v SUITE B [3], je nejvhodnější zvolit algoritmus Diffie-Hellman (D-H), tedy konkrétně D-H nad eliptickou křivkou (ECDH). Výkonově jsou algoritmy využívající eliptické křivky výhodnější, jejich implementace je však obecně složitějším problémem. Nejvýhodnější variantou je poté zvolit symetrickou šifru pro samotné šifrování a asymetrickou pro domluvu klíčů.

2. IMPLEMENTACE ECDH

Implementace ECDH probíhala do mikrokontroleru MSP430, který je jedním ze zástupců nízkovýkonových mikrokontrolerů. Implementace byla rozdělena do následujících částí:

1. *Reprezentace velkých čísel v mikrokontroleru s omezeným výkonem*

   Jelikož dostupné klasické datové typy `int, long` aj. jsou nedostatečné svou velikostí, je nutné vytvořit datovou strukturu, která bude velká čísla reprezentovat. Díky této datové struktuře jsme poté schopni provádět operace modulární aritmetiky potřebné pro ECDH.

   Pro obecný model reprezentace čísel využijeme poziční číselnou soustavu [4]:

$$N = \sum_{i=0}^{k-1} n_i r^i = n_{k-1} r^{k-1} + n_{k-2} r^{k-2} + \ldots + n_0 r^0, \qquad (1)$$

kde N je hodnota čísla pro danou číselnou soustavu, n_i představuje číslici (i představuje pozici této číslice), která je vynásobena mocninou se základem r.

Dle tohoto logického vzoru byla vytvořena datová struktura `struct bignum_st` [5]:

```
struct bignum_st
 {
        BN_ULONG *d;
        int top;
        int dmax;
        int neg;
        int flags;
        }
```

Kód 1: Struktura pro reprezentaci velkých čísel

Kde první prvek je ukazatel pole `(*d)`, další řádek určuje pozici nejvíce významného bitu (top), dále je pak velikost pole `d` (dmax), příznak zda se jedná o záporné číslo (neg, 1 pro záporné) a poté pomocná proměnná pro další příznaky. Jedná se tedy o pole, které je uložené na pozici ukazatele v paměti.

2. *Matematické operace s velkými čísly*

Vzhledem k nutnosti využít vlastní datovou strukturu, bylo třeba vytvořit algoritmy pro všechny potřebné matematické operace (sčítání, odčítání, násobení, dělení a modulo). Uvažujeme-li dvě celá čísla o velikosti k (pokud se jedná o různě velká čísla, vždy je třeba menší z nich dorovnat nulami) a základu r.

Algoritmy pro velká čísla vychází ze základních algoritmů pro malá čísla. Násobení, sčítání, odčítání vychází z algoritmů, které jsou používány pro operace tzv. na papíře pod sebou. Dělení pak můžeme spojit s operací modulo. Vycházíme totiž z algoritmů pro dělení celých čísel se zbytkem a tedy výstup funkce je výsledek a zbytek.

Ukázku kódu pro násobení dvou velkých čísel můžeme vidět v kódu 3. Jedná se o algoritmus simulující právě papírové násobení.

```
for (i=0;i<size;i++) {
   for (j=0;j<size;j++) {
        if (x[j]*y[i]+prebytek >= 10) then {
                for (k=1;k<10;k++{
                         if ((x[j]*y[i]+prebytek-k*10) > 10) then {
                         prebytek = k;
                         v[i][j+i] = x[j]*y[i];
                         }
                        else {
                         v[i][j+i] = x[j]*y[i]+prebytek;
                         prebytek = 0;
                         }
                if {prebytek != 0} then {v[i][j+1] = prebytek;}}}
       prebytek = 0
for (i=0;i<size;i++) {
   for (j=0;j<size;j++) {
        vysledek[i] += v[j][i]}
   if (vysledek[i] >= 10) then {
        for (k=1;k<10;k++{
                if ((vysledek[i]+prebytek-k*10) > 10) then {
                         prebytek = k;
                         vysledek[i] -= k*10;
                         }
                        else {
                         vysledek[i] += prebytek;
                         prebytek = 0;}}
```

Kód 2: Ukázka algoritmu pro násobení velkých čísel

3. *Základní principy ECDH a rozdíly oproti D-H*

Jako první jsme ověřili funkčnost základního algoritmu D-H s malými čísly (Kód 3).

```
void encrypt() {
        int i;
        c = 1;
        for(i=0;i< e;i++)
        c=c*M%n;
        c = c%n;
        }
```

```
void decrypt() {
         int i;
         M = 1;
         for(i=0;i< d;i++)
                M=M*c%n;
                M = M%n;
        }
```

Kód 3: Šifrování a dešifrování D-H s malými čísly

Oproti D-H je nutno dohodnout nejprve společné parametry (*p, a, b, G, n, h*) a při volbě těchto parametrů je nutno dodržet singularitu eliptické křivky:

$$4a^3 + 27b^2 \neq 0. \qquad (2)$$

Poté volíme bod na eliptické křivce d_A (či analogicky d_B). Volba bodu na křivce je náhodná a je tedy nutné vytvořit generátor náhodných čísel. V MSP430 je vytvořen náhodný generátor pomocí vnitřních časovačů (`SMCLK a ACLK`), které jsou spolu schopné vygenerovat až 16 bitové náhodné číslo. Náhodnost jevu je zaručena nemožností určit následující pulz i se znalostí předchozího stavu časovačů [6].

Jeden z nejdůležitějších výpočtů v ECDH je výpočet bodů Q_A a Q_B, po kterém se následně ověřují body $Z=Z'$. Jeho ukázku můžeme vidět v kódu 4.

```
s = ((3x^2+a)/2y)%p
xa = (s^2-2x)%p
ya = (s*(x-xa)-y)%p
```

Kód 4: Ukázka výpočtu bodu Q_A pro ECDH s malými čísly

3. ZÁVĚR

Příspěvek řeší problematiku implementace asymetrické kryptografie, konkrétně eliptických křivek do nízko-výkonových mikrokontrolerů. V rámci příspěvku je popsáno řešení implementace ECDH do MSP430 od firmy Texas Instruments. Byl vypracován jednoduchý kód pro D-H, který byl posléze rozšířen o složitější operace s velkými čísly, dále byla vyřešena algoritmizace ECDH a vytvořen kód ECDH pro malá čísla.

LITERATURA

[1] Burda, K.: Bezpečnost informačních systémů, Brno, 2005, s. 6-30

[2] U.S. NSA: Fact Sheet NSA Suite B Cryptography, 2009
Dostupné z URL: http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml

[3] Rybak, Š.: Skupina algoritmů NSA Suite-B Cryptography, Brno, 2011, s. 33-35

[4] Němec, P.: Číselné soustavy, Praha, 2009, s. 1-4
Dostupné z URL: http://dum.rvp.cz/materialy/stahnout.html?s=qwzccdrf

[5] OpenSSL: Cryptography and SSL/TSL Toolkit
Dostupné z URL: http://www.openssl.org/docs/crypto/bn_internal.html

[6] Texas Instruments: Random Number Generator Using MSP430, 2006, s. 1-4