# Aritmetika s velkými čísly na čipové kartě

Smart Cards & Devices Forum 2013

**Ivo Rosol**
**ředitel divize vývoje**
**OKsystem s.r.o.**

Praha, 23.5.2013

OKsystem

# Čísla v kryptografii

- V kryptografii se zásadně pracuje s celými čísly, jejichž velikost, v závislosti na typu kryptografie, může být ve stovkách, nebo tisících bitů. S čísly této velikosti se provádí základní aritmetické operace – sčítání, odčítání, násobení a umocňování a to běžné, nebo častěji modulární.
- V počítači jsou čísla reprezentována v binární podobě a procesory počítačů umí pracovat pouze s omezenou množinou typů celých čísel, například s 8, 16,  32 nebo 64 bitovými celými čísly. Programovací jazyky zpravidla nabízejí stejné základní celočíselné datové typy, delší čísla je nutné reprezentovat ve formě pole a aritmetiku realizovat nad polem čísel.
- Pokud takové operace příslušný programovací jazyk, nebo knihovna nepodporuje, je nutné tyto výpočty realizovat programem.

# Obsah přednášky

V přednášce budou ukázány některé algoritmy, vhodné pro aritmetiku s velkými čísly na smart kartách.

Budou ukázány výsledky porovnání platforem čipových karet Java card, .NET a Multos při provádění výpočtů kryptografických primitiv – odečítání, násobení, modulární násobení a mocnění s velkými čísly.

Práce byly realizovány v rámci realizace projektu výzkumu a vývoje TAČR, program ALFA - Systém pro kryptografickou ochranu elektronické identity, který zpracovává OKsystem společně s VUT Brno.

# R&D projekt ANAUT

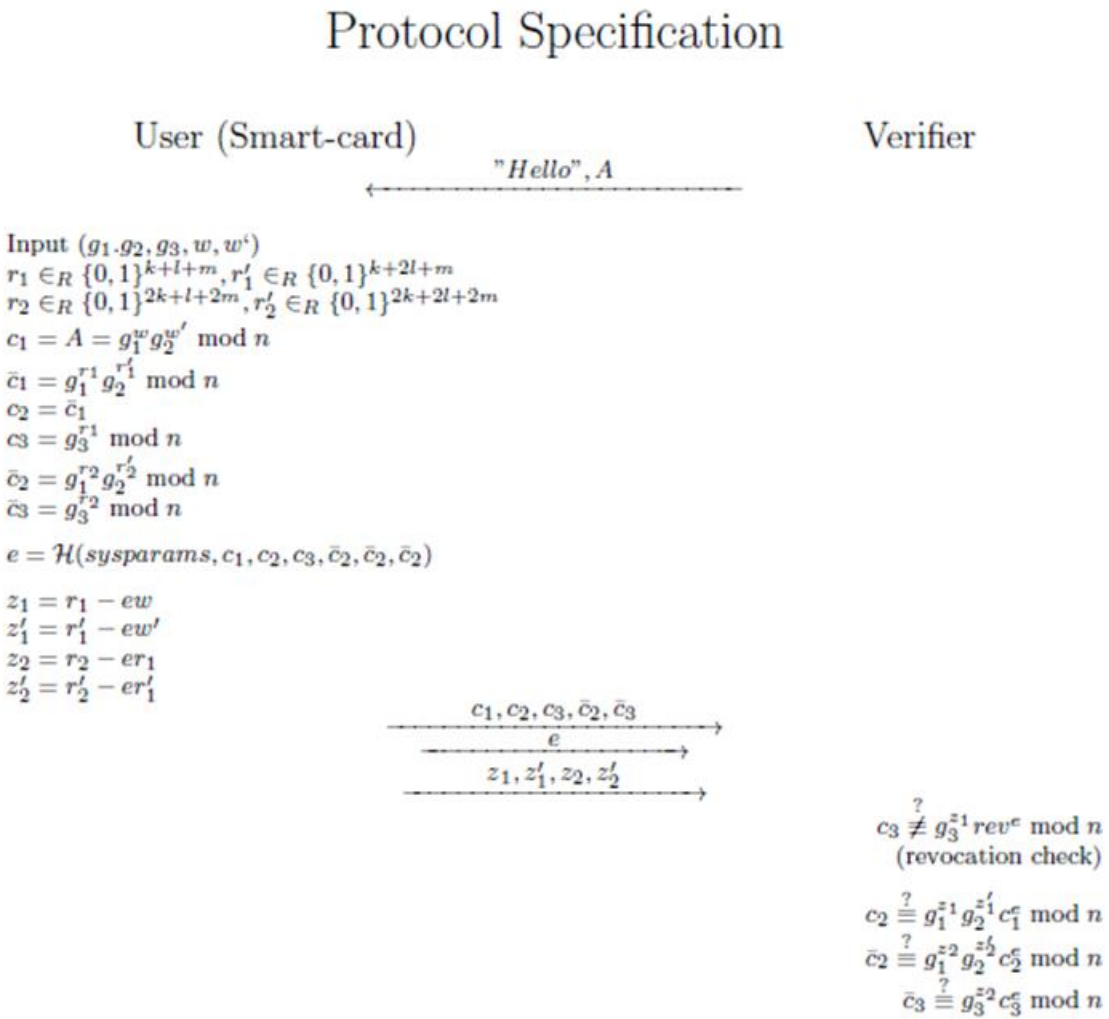



Figure 1: Verification Protocol (Optimized)

OKsystem

# Sčítání

Nejprve si ukážeme základní stavební kámen všech výpočtů – binární sčítačku, a její využití pro součet vícebitových čísel.

Použití jednobitové sčítačky vede na problém postupného šíření přenosu do vyšších řádů, procesory proto využívají vícebitové sčítačky, nebo sofistikované metody urychlení přenosu.

# Binární jednobitová poloscítačka

Jednobitová „poloscítačka“ s přenosem do vyššího řádu
Pravdivostní tabulka

| $A_0$ | $B_0$ | $S_0$ | $C_1$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

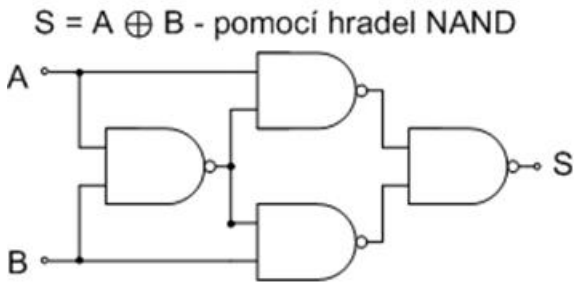
$$S_0 = \overline{A_0} * B_0 + A_0 * \overline{B_0} = A \oplus B$$
$$C_1 = A_0 * B_0$$

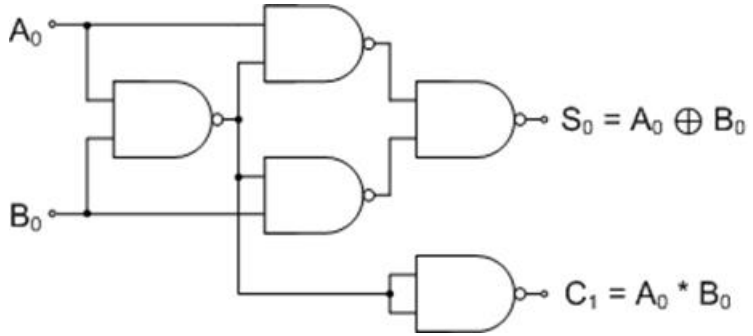
OKsystem

# Optimalizovaná poloscčítačka

Operace $\oplus$ se dá realizovat pomocí pouze 4 hradel NAND (místo 5 hradel), pokud se výraz upraví tak, aby byl mezivýsledek negace (A * B) použit dvakrát.

$$
\begin{aligned}
S &= \overline{A} * B + A * \overline{B} = \\
&= \left(\overline{A} * B + 0\right) + \left(0 + A * \overline{B}\right) = \\
&= \left(\overline{A} * B + B * \overline{B}\right) + \left(A * \overline{A} + A * \overline{B}\right) = \\
&= \left(\overline{A} + \overline{B}\right) * B + \left(\overline{A} + \overline{B}\right) * A = \\
&= \overline{(A * B)} * B + \overline{(B * A)} * A = \\
&= \overline{\left(\overline{(A * B)} * B\right)} * \overline{\left(\overline{(B * A)} * A\right)}
\end{aligned}
$$

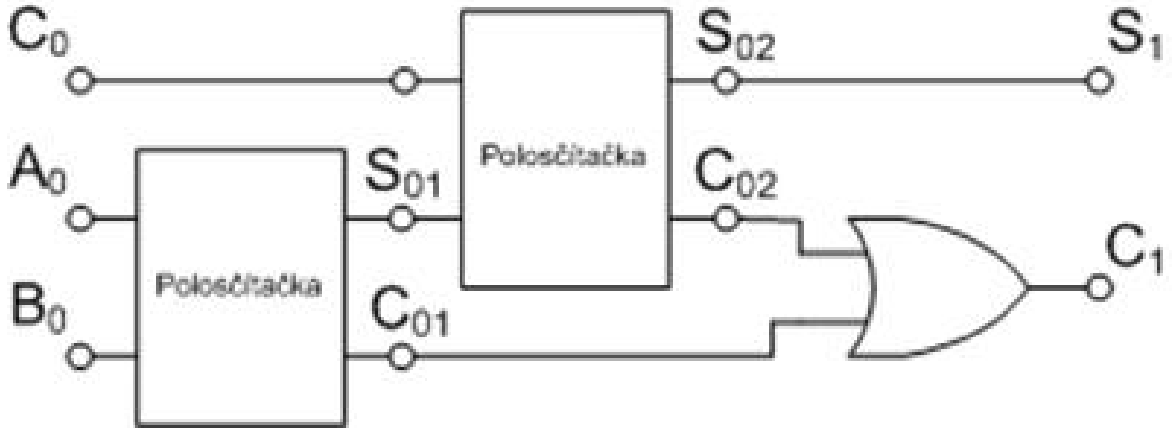



OKsystem

# Úplná jednobitová sčítačka

- **Úplná jednobitová sčítačka ze 2 polosčítaček**
- Úplnou jednobitovou sčítačku lze sestavit (neoptimálně) ze 2 polosčítaček.
- Součet $S_1$ získám kaskádovým sečtením $S_{01}=A_0+B_0$ v první polosčítačce a následným přičtením přenosu $C_0$ v druhé polosčítačce.
- Přenos $C_1$ se generuje, pokud je přenos $C_{01}$ ze součtu bitů, nebo je přenos $C_{02}$ ze součtu přenosu a součtu bitů:



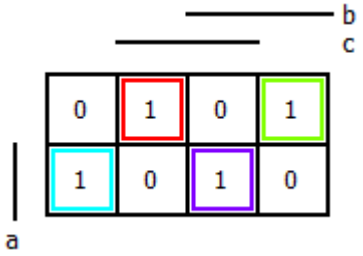

OKsystem

# Úplná jednobitová sčítačka (optimální)

**Pravdivostní tabulka úplné sčítačky**

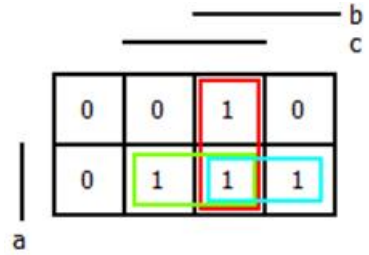
| $A_0$ | $B_0$ | $C_0$ | $S_1$ | $C_1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Karnaughova mapa pro součet $S_1$**




$$S_1 = \overline{A_0}*\overline{B_0}*C_0 + \overline{A_0}*B_0*\overline{C_0} + A_0*\overline{B_0}*\overline{C_0} + A_0*B_0*C_0$$

**Karnaughova mapa pro přenos $C_1$**




$$C_1 = B_0*C_0 + A_0*C_0 + A_0*B_0$$

OKsystem

# Bez Karnaughovy mapy je minimalizace pracnější

$C_1 = \overline{A_0}*B_0*C_0 + A_0*\overline{B_0}*C_0 + A_0*B_0*\overline{C_0} + A_0*B_0*C_0 =$

$= \overline{A_0}*B_0*C_0 + A_0*(\overline{B_0}*C_0 + B_0*\overline{C_0} + B_0*C_0) =$

$= \overline{A_0}*B_0*C_0 + A_0*(\overline{B_0}*C_0 + B_0*\overline{C_0} + B_0*C_0 + B_0*C_0)$ = /přičteno $B_0*C_0$, protože $A + A = A$

$= \overline{A_0}*B_0*C_0 + A_0*((\overline{B_0} + B_0)*C_0 + B_0*(\overline{C_0} + C_0))$ = / vytknuto

$= \overline{A_0}*B_0*C_0 + A_0*(C_0 + B_0)$ = / protože $\overline{B} + B = 1$

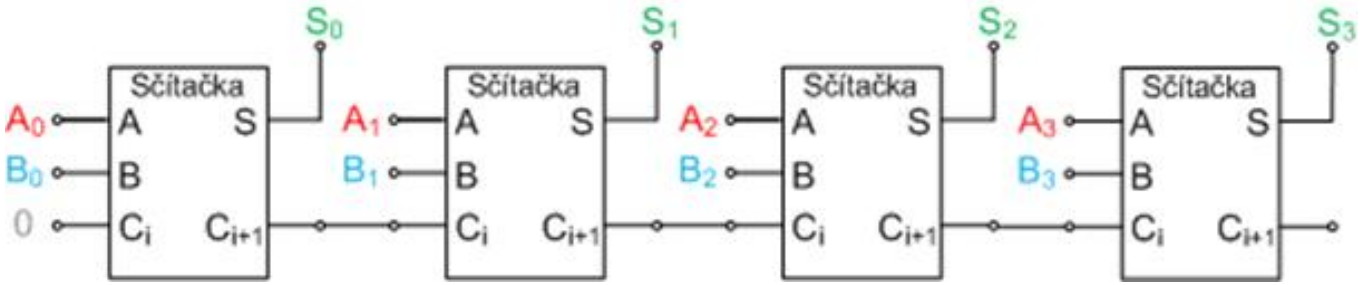
$= \overline{A_0}*B_0*C_0 + A_0*C_0 + A_0*B_0 =$

$= (\overline{A_0}*B_0 + A_0)*C_0 + A_0*B_0$ = / protože $((\overline{A}*B + A) = (B + A)$

$= B_0*C_0 + A_0*C_0 + A_0*B_0$

# Vícebitová sčítačka s šířením přenosu

- Dochází k šíření přenosu z nižších řádů k vyšším. Správný součet je k dispozici až po ustálení přenosů.



# Vícebitová sčítačka s predikcí přenosu

Predikce (urychlení) přenosu spočívá v převodu na dvouúrovňový kombinační obvod (součet součinů).

Princip spočívá ve vyjádření výstupního přenosu v i-tém řádu pomocí i-tých a nižších bitů sčítanců a vstupního přenosu v 0-tém řádu. Složitost obvodu rychle roste s počtem bitů sčítanců.

**$C_{i+1} = A_i*B_i + C_i*(A_i+B_i)$**

$C_1 = A_0*B_0 + C_0*(A_0+B_0)$

$C_2 = A_1*B_1 + C_1*(A_1+B_1)$
$C_2 = A_1*B_1 + (A_0*B_0 + C_0*(A_0+B_0))*(A_1+B_1)$
$C_2 = A_1*B_1 + A_0*B_0*(A_1+B_1)+ C_0*(A_0+B_0)*(A_1+B_1)$

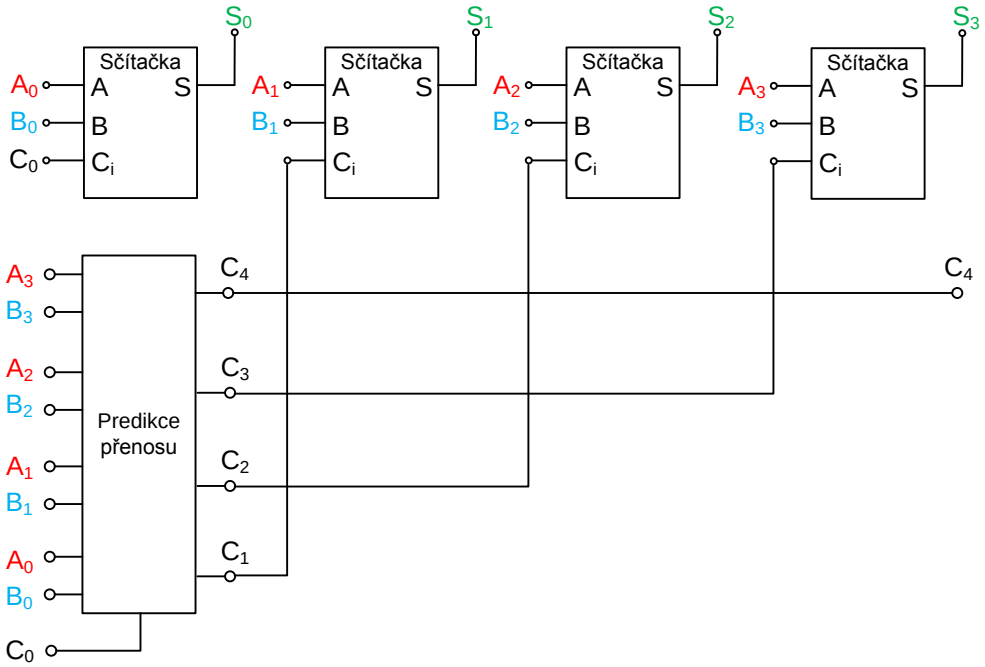
$C_3 = A_2*B_2 + C_2*(A_2+B_2)$
$C_3 = A_2*B_2 + (A_1*B_1 + A_0*B_0*(A_1+B_1)+ C_0*(A_0+B_0)*(A_1+B_1))*(A_2+B_2)$ / dosazeno za $C_2$
$C_3 = A_2*B_2 + A_1*B_1*(A_2+B_2) + A_0*B_0*(A_1+B_1)*(A_2+B_2) + C_0*(A_0+B_0)*(A_1+B_1)*(A_2+B_2)$ / roznásobeno

$C_4 = A_3*B_3 + C_3*(A_3+B_3)$
$C_4 = A_3*B_3 + (A_2*B_2 + A_1*B_1*(A_2+B_2) + A_0*B_0*(A_1+B_1)*(A_2+B_2) +$
$C_0*(A_0+B_0)*(A_1+B_1)*(A_2+B_2))*(A_3+B_3)$
$C_4 = A_3*B_3 + A_2*B_2*(A_3+B_3) + A_1*B_1*(A_2+B_2)*(A_3+B_3) + A_0*B_0*(A_1+B_1)*(A_2+B_2)*(A_3+B_3) +$
$C_0*(A_0+B_0)*(A_1+B_1)*(A_2+B_2)*(A_3+B_3)$

OKsystem

# Vícebitová sčítačka s predikcí přenosu

- Výrazy pro $C_1$, $C_2$, $C_3$ a $C_4$ realizujeme jako dvouúrovňový kombinační obvod, neboť je závislý pouze na okamžité hodnotě $\mathbf{A_i}$, $\mathbf{B_i}$ a $\mathbf{C_0}$




OKsystem

# Kuličková dřevěná binární sčítačka

- Web: http://woodgears.ca/marbleadd/
- video: http://www.youtube.com/watch?feature=player_embedded&v=GcDshWmhF4A

# Násobení

- Zatímco binární jednobitová násobička je jednodušší než sčítačka (aritmetický součin dvou bitů má stejnou hodnotu jako jejich logický součin, přenos při jednobitovém násobení nevzniká), násobení vícebitových čísel je časově náročnější.
- Program pro násobení může teoreticky využít takzvaný školský algoritmus pro násobení, jeho efektivita je však úměrná $n^2$. Naštěstí existují účinnější metody, ukázán bude Karatsubův algoritmus pro násobení velkých čísel. V praxi se také často využívá Combův algoritmus, který optimalizuje přenosy do vyšších řádů.

# Školský algoritmus

- Násobení dvou n-ciferných čísel vyžaduje n x n násobení jednociferných čísel a součty mezivýsledků.
- Ve dvojkové soustavě vede školský algoritmus na jednoduché operace – pro každý bit s hodnotou 1 v násobiteli se bity násobence posunou na odpovídající pozici a přičtou k mezivýsledku.
- **Školský algoritmus – příklad:**

```
        8  9
    *   2  3
-------------
        2  6  7
  1  7  8
-------------
  2  0  4  7
```

V tomto případě jsou 4 přenosy při násobení a 2 přenosy při sčítání mezivýsledků, celkem 6 přenosů

Při implementaci není nutné si pamatovat jednotlivé dílčí násobky a sčítat je na závěr, místo toho je možné jednotlivé sloupce (řády) sčítat průběžně a pamatovat si přenosy.

# Combův algoritmus

Combův algoritmus spočívá v optimalizaci implementačních detailů školského násobení. Ruší přenosy při násobení (každý přenos je přičítání hodnoty přenosu s výsledkem násobení ve vyšším řádu) a tyto přenosy realizuje až při závěrečném sčítání mezivýsledků vzniklých při násobení jednotlivými řády násobitele. Aby nebylo nutné používat přenosy při násobení jednotlivými ciframi násobitele, zapamatují se jednotlivé mezivýsledky v plné hodnotě (nikoli jen cifra u řádu jednotek a přenos).

## Combův algoritmus – příklad:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | 8 | 9 | |
| | | | | * | 2 | 3 | |
| | | | 2 | 4 | 2 | 7 | |
| | | 1 | 6 | 1 | 8 | | |
| 2 | | _0 | | _4 | | 7 | |

V tomto případě je 0 přenosů při násobení a **pouze 2 přenosy při sčítání** mezivýsledků (ale větší počet sčítanců)

# Combův algoritmus – větší příklad

```
   35782
  *78339
-----------
 2803126098
```

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | **3** | | **5** | | **7** | | **8** | | **2** | **Násobenec** |
| * | | | | | | | | **7** | | **8** | | **3** | | **9** | | | **Násobitel** |
| + | | | | | | | 2 | 7 | 4 | 5 | 6 | 3 | 7 | 2 | 1 | 8 | Mezivýsledek |
| + | | | | | | 0 | 9 | 1 | 5 | 2 | 1 | 2 | 4 | 0 | 6 | | Mezivýsledek |
| + | | | | | 0 | 9 | 1 | 5 | 2 | 1 | 2 | 4 | 0 | 6 | | | Mezivýsledek |
| + | | | 2 | 4 | 4 | 0 | 5 | 6 | 6 | 4 | 1 | 6 | | | | | Mezivýsledek |
| + | 2 | 1 | 3 | 5 | 4 | 9 | 5 | 6 | 1 | 4 | | | | | | | Mezivýsledek |
| + | 0 | | 2 | | 3 | | 4 | | 3 | | 2 | | 2 | | 0 | | 0 | | Přenos |
| | 1 | | 3 | | 5 | | 7 | | 9 | | 9 | | 7 | | 5 | | 2 | | **48 sčítání** |
| **=** | **2** | | **8** | | **0** | | **3** | | **1** | | **2** | | **6** | | **0** | | **9** | **8** | **Výsledek** |

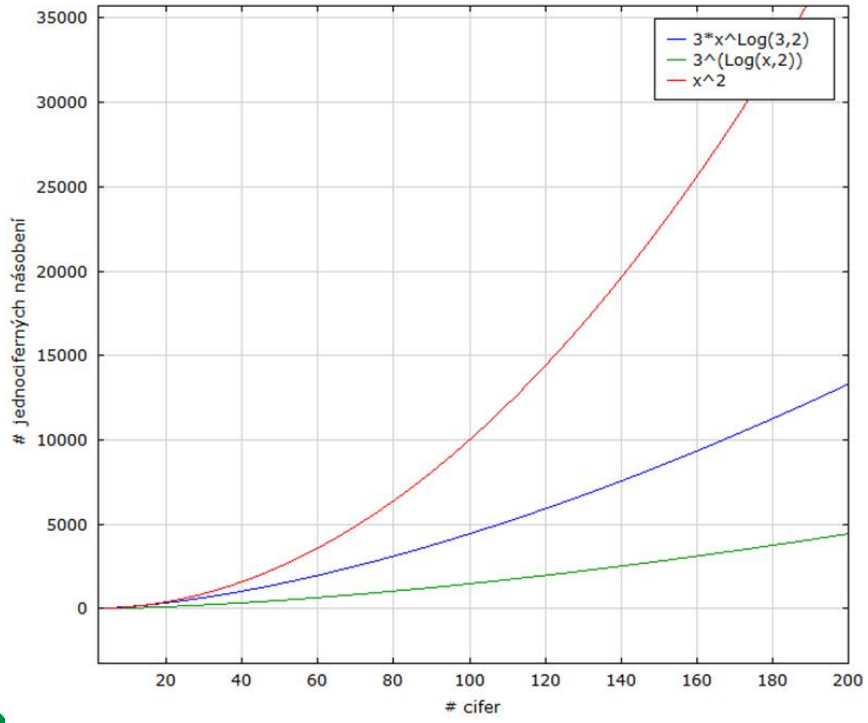
Při správné implementaci je Combův algoritmus o cca 30% výkonnější než školský algoritmus

OKsystem

# Karatsubův algoritmus

- Vhodný pro velmi velká čísla, efektivní pro čísla větší než 500 bitů (kvůli sčítání a režii rekurze)
- Anatolij Alexejevič Karatsuba, žák Andreje Kolmogorova, který na přednášce tvrdil, že násobení dvou n-ciferných čísel vyžaduje $n^2$ násobení. O týden později Karatsuba předložil algoritmus, který asymptoticky vyžaduje pouze N násobení:

$$N \leq 3^{\log_2 n} \leq 3 * n^{\log_2 3}$$




OKsystem

# Karatsubův algoritmus 1/3

A je n-ciferné číslo (např. A = 15321)

B je n-ciferné číslo (např. B = 5762)

A i B vyjádříme jako součet dvou částí – vyššího řádu a nižšího řádu, s polovičním počtem cifer. Z je základ číselné soustavy, m = n / 2 (celočíselně)

$A = A_1 * Z^m + A_0$ ($A = 15321 = 153 * 10^2 + 21$)

$B = B_1 * Z^m + B_0$ ($B = 5762 = 57 * 10^2 + 62$)

$A*B = (A_1*Z^m + A_0)*(B_1*Z^m + B_0) = A_1*B_1*Z^{2m} + (A_1*B_0 + A_0*B_1)* Z^m + A_0*B_0$

Označíme:

- $k_0 = A_0*B_0$
- $k_1 = A_1*B_0 + A_0*B_1$
- $k_2 = A_1*B_1$
- **$A*B = k_2* Z^{2m} + k_1* Z^m + k_0$**

Pro výpočet $k_0$, $k_1$, $k_2$ jsou potřeba 4 násobení s čísly s polovičním počtem cifer, tedy zůstává stejný počet jednociferných násobení $4* (n/2)^2$ jako ve školském algoritmu. $Z^m$ a $Z^{2m}$ pouze označují řád mezivýsledků a ve skutečnosti se nepočítají, pouze se dělá řádový posun při sčítání mezivýsledků.

# Karatsubův algoritmus 2/3

Karatsubův trik, kdy již dříve spočtené násobky použiju pro výpočet $k_1$, kde jsou 2 násobení:

$$\mathbf{k_1 = A_1*B_0 + A_0*B_1 =}$$

$$\mathbf{= (A_1 + A_0)*(B_1 + B_0) - A_1*B_1 - A_0*B_0 =}$$

$$\mathbf{= (A_1 + A_0)*(B_1 + B_0) - k_2 - k_0}$$

- Zde je jenom jedno násobení, protože výraz $k_2 = A_1*B_1$ a výraz $k_0 = A_0*B_0$ již byly spočítány.
- Celkem jsou **pouze 3 násobení** s (přibližně) polovičním počtem cifer. Přibývají ovšem 2 sčítání.
- Algoritmus se použije rekurzivně ve výrazech pro výpočet koeficientů $k_0, k_1, k_2$

# Karatsubův algoritmus 3/3 - příklad

A: 15321

B: 5762

A*B = **88 279 602**

Z = 10, n = 5, m = n/2 = 2

A: 15321 = 153 * $10^2$ + 21

B: 5762 = 57 * $10^2$ + 62

$k_0 = A_0*B_0$ = 21 * 62 = (Kartasuba rekurze) → 1302

$k_2 = A_1*B_1$ = 153 * 57 = (Kartasuba rekurze) → 8721

$k_1 = (A_1 + A_0)*(B_1 + B_0) - k_2 - k_0$ =

= (153 + 21)*(57+62) - 8721 - 1302 =

= (174 * 119) (Kartasuba rekurze) → 20706 - 10023 = 10683

$A*B = k_2* Z^{2m} + k_1* Z^m + k_0$ = 8721*$10^{2*2}$+ 10683*$10^2$ + 1302 = **88 279 602**

**Poznámky:**

- V binární soustavě se 32 bitovými celými čísly se výhodně volí $Z = 2^{31}$ a rekurze pokračuje, dokud délka činitelů neklesne na 1 bit.
- Rekurze není k dispozici na čipových kartách a převod na iteraci je implementačně náročný.

OKsystem

# Modulární násobení - RSA tunel

RSA tunel se používá pro modulární násobení v případě, že je k dispozici RSA koprocesor, který rychle spočítá modulární mocninu. Ze známého vztahu:

$$(A + B)^2 = A^2 + 2AB + B^2$$

Vyplývá (všechny operace jsou modulární):

$$A * B = \frac{(A + B)^2 - A^2 - B^2}{2}$$

Pro výpočet součinu je tedy potřeba spočítat 3 druhé mocniny (a součet modulárně dělit 2).

**Poznámky:**

- pokud je modul n větší než A*B, tak lze RSA tunel použít i pro násobení (nemodulární)
- modul n nemusí být ve tvaru n=p*q, kde p a q jsou prvočísla, neboť nevyužíváme algoritmus RSA, pouze koprocesor. Modul se volí ve tvaru 1000000000000000……..000000000001 (liché n s potřebným počtem bitů a s minimálním počtem jedniček).
- V závislosti na implementaci RSA na čipové kartě nemusí být možné použít mocninu s veřejným klíčem, protože velikost veřejného exponentu bývá implementačně omezená (často se používá hodnota 65 537). V tomto případě je nutné použít mocninu s „privátním“ klíčem.
- Některé implementace počítají mocninu s privátním klíčem výhradně s využitím CRT (čínský teorém o zbytku). Pak nelze zvolit libovolný modul a RSA tunel není použitelný pro násobení.

# RSA tunel – modulární dělení 2

Modulární dělení 2:

Pro x<n, n liché, platí:

$$\frac{x}{2} \bmod n = \begin{cases} \frac{x}{2} & x \text{ sudé} (x \text{ má } nejnižší\ bit = 0) \\ \frac{x+n}{2} & x \text{ liché } (x+n \text{ má } nejnižší\ bit = 0) \end{cases}$$

Příklad: $\frac{9}{2}(mod\ 11) = \frac{9+11}{2} = 10$

Zkouška: $(10+10)\ mod\ 11 = 20\ mod\ 11 = 9$

OKsystem

# RSA tunel 2

Dokonce lze součin spočítat pouze pomocí 2 druhých mocnin:

$$(A+B)^2-(A-B)^2= A^2+2AB+B^2-A^2+2AB-A^2=4AB$$

$$A*B=\frac{(A+B)^2-(A-B)^2}{4}$$

- Modulární dělení 4 se provádí pomocí dvojnásobného dělení 2, dle předchozího příkladu.
- Reálná výkonnost RSA a RSA 2 tunelu je na čipových kartách srovnatelná. Úspora výpočtu jedné mocniny je eliminována opakovaným modulárním dělením 2.

# Prostá mocnina čísla

Dvojkové umocňování zprava. Základní myšlenkou je rozepsání exponentu jako součtu jeho jednotlivých řádů a následné převedení na součin jednotlivých mocnin, podle známých vztahů:

$Z^{(A+B)} = Z^A * Z^B$ a $Z^{(A*B)} = (Z^A)^B$

$$e = \sum_{i=0}^{i=n-1} a_i 2^i$$

Kde $a_i$ je 0 nebo 1

$$Z^e = Z^{\sum_{i=0}^{i=n-1} a_i 2^i} = \prod_{i=0}^{i=n-1} \left(Z^{2^i}\right)^{a_i} = \prod_{i=0}^{i=n-1} (M(i))^{a_i}$$

Zde jsme označili mezivýsledek $M(i) = Z^{2^i}$

Ve výše uvedeném vztahu je podstatné průběžné počítání a uchování mezivýsledku $Z^{2^i}$ a jeho opětovné použití v dalším výpočtu.

$$M(i+1) = Z^{2^{(i+1)}} = Z^{2^i * 2} = \left(Z^{2^i}\right)^2 = M(i)^2$$

Tedy v kroku (i+1) předchozí mezivýsledek $M(i)$ umocníme na druhou a opět uložíme $M(i+1)$.

- Pokud $a_{i+1} = 1$, tak mezivýsledek a vynásobíme s řetězovým součinem. Pokud $a_i = 0$, neděláme nic – násobení jedničkou. Zvětšíme i a přejdeme do vyššího řádu.
- Počet operací (druhá mocnina mezivýsledku, tedy násobení mezivýsledku sama se sebou) je pouze $\log_2 n$.

# Modulární mocnina čísla

V modulární aritmetice platí (mimo jiné):

$$(a*b) \bmod n = [(a \bmod n) * (b \bmod n)] \bmod n$$

Proto lze modulární mocninu počítat stejně jako prostou mocninu iterací umocňování zprava, každý mezivýsledek redukujeme (mod n), stejně jako každé násobení.

$$Z^e mod\ n = \prod_{i=0}^{i=n-1} \left(Z^{2^i}\right)^{a_i} mod\ n$$

Montgomeryho algoritmus se používá pro redukci (mod n), pokud je nutné redukovat velmi velké číslo.

# Grafy z implementace algoritmů na různých platformách

Byly testovány následující platformy programovatelných čipových karet:

- Java Card (Gemalto TOP IM, Oberthur Id One v7)
- .NET card (Gemalto)
- Multos (ML2-80k, ML3-36k)

Všechny karty měly kontaktní rozhraní.

Měření byla zatížena malou systematickou chybou spočívající v započtení doby komunikace 1 APDU s kartou k době výpočtu.

OKsystem

# Násobení

Na jednotlivých platformách byla porovnána:

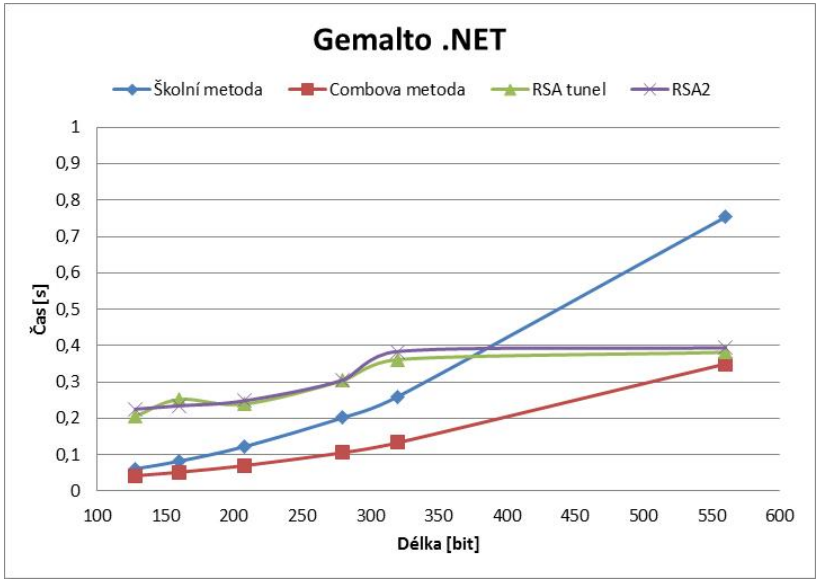
- Školní metoda
- Combova metoda
- RSA tunel
- RSA 2 tunel

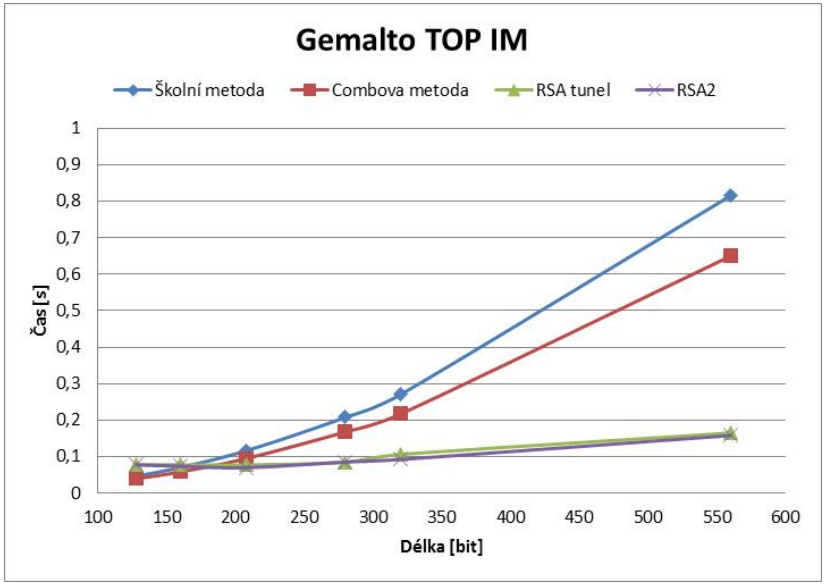
Porovnání bylo provedeno pro délky činitelů použitých v algoritmu ProveAtt (1024 a 2048 bitů)

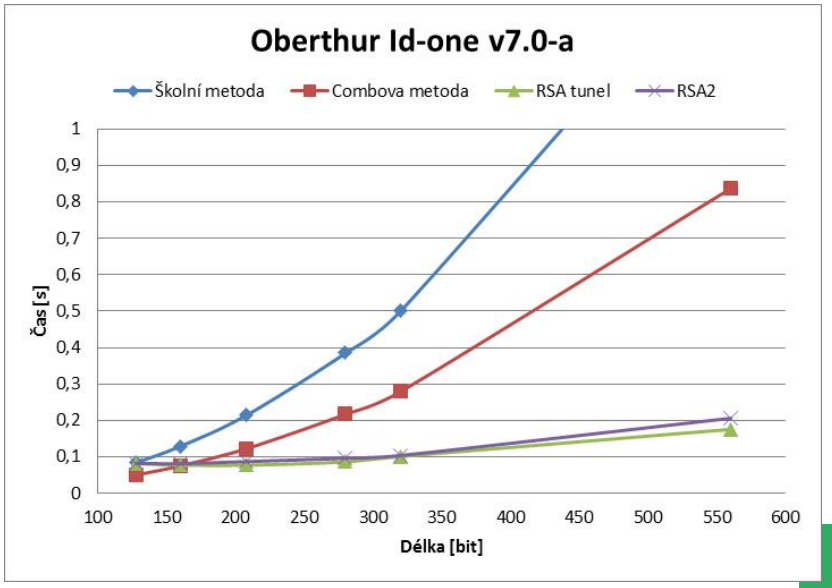
Karatsubova metoda nebyla použita (rekurze).

Pro každou platformu byl vybrán nejrychlejší algoritmus a platformy byly porovnány.

# Násobení – porovnání algoritmů

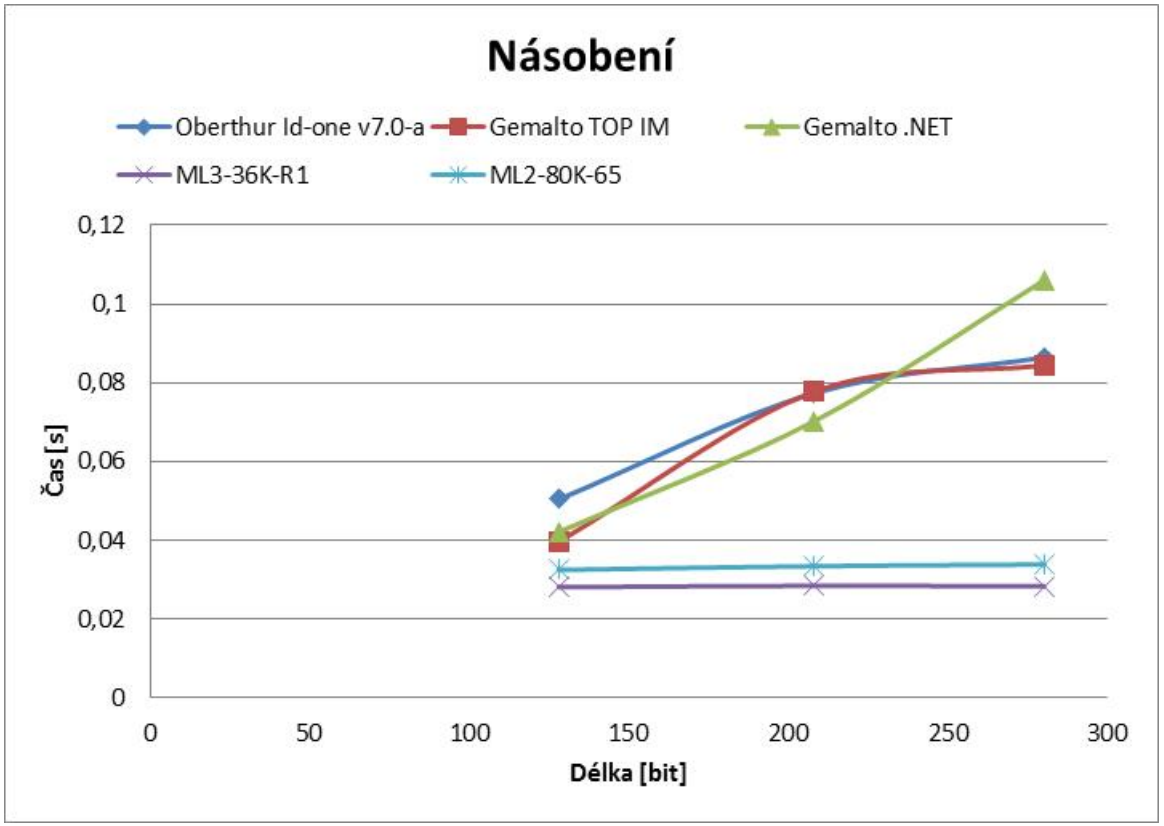


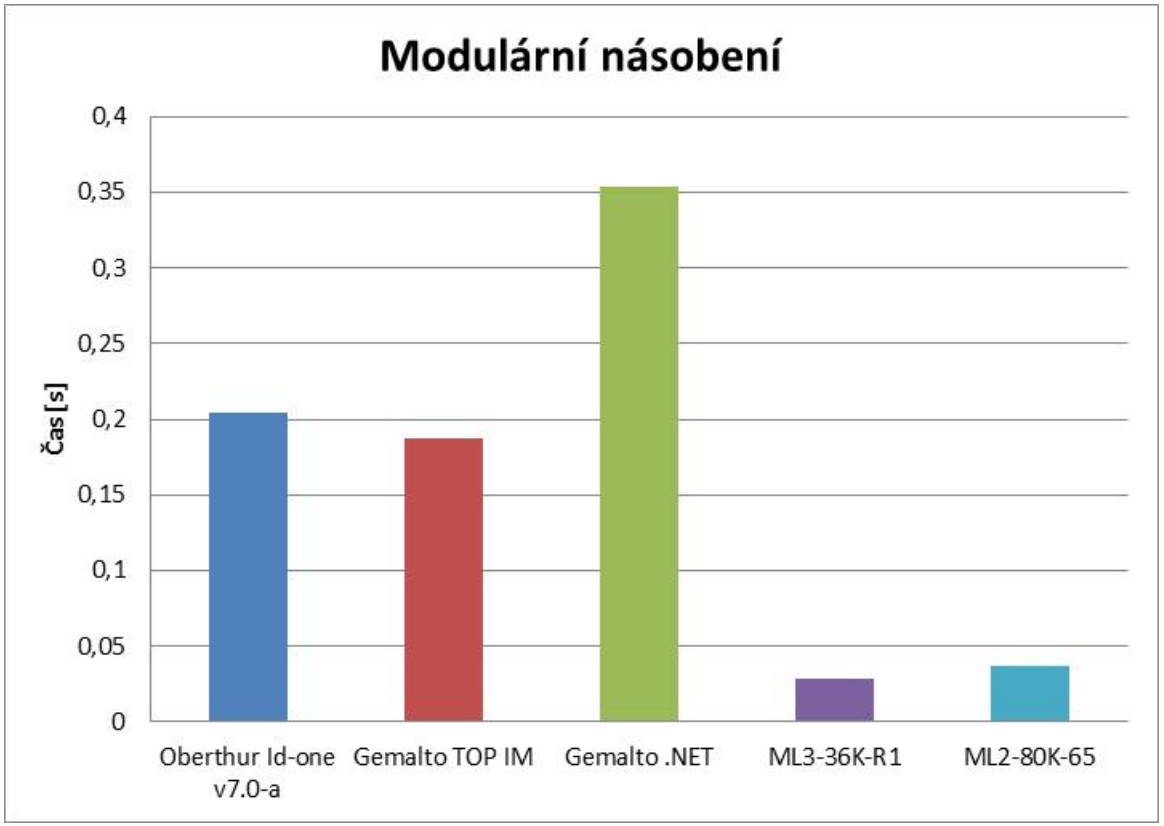

OKsystem

# Násobení – porovnání platforem (nejlepší algoritmy)



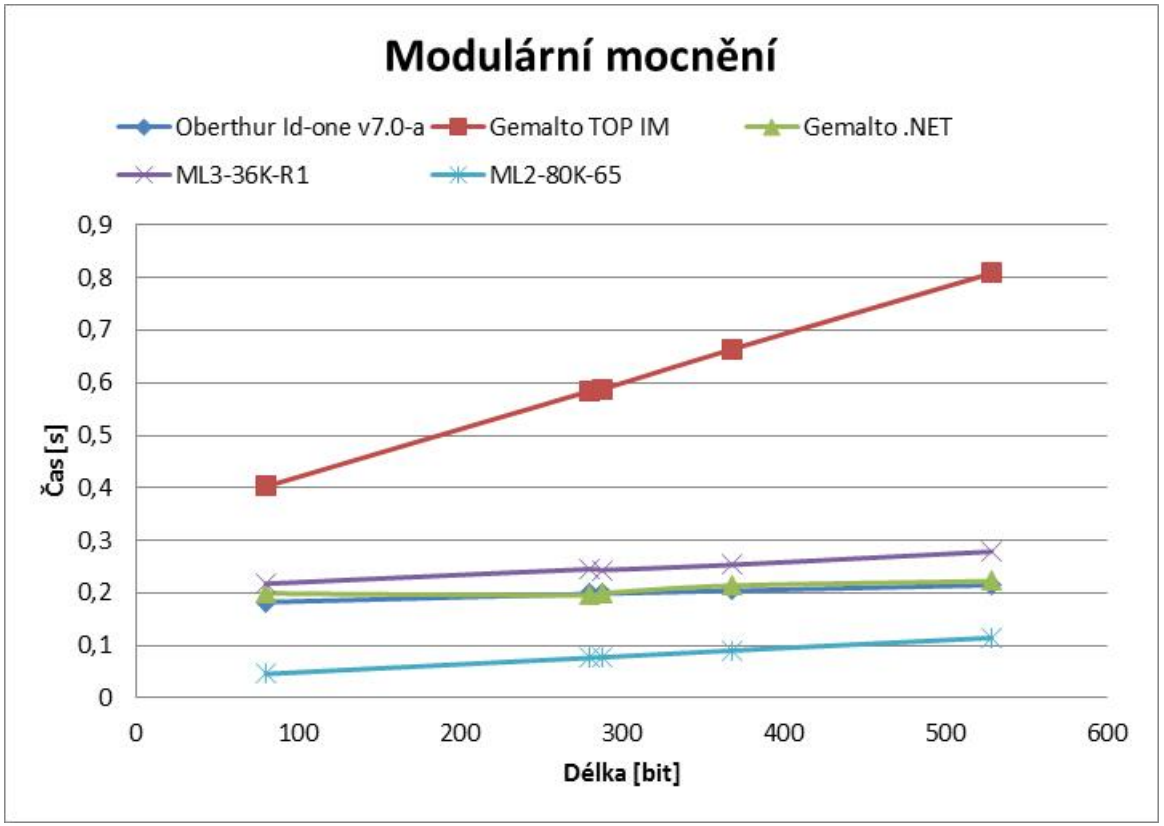

OKsystem

# Modulární násobení – porovnání platforem



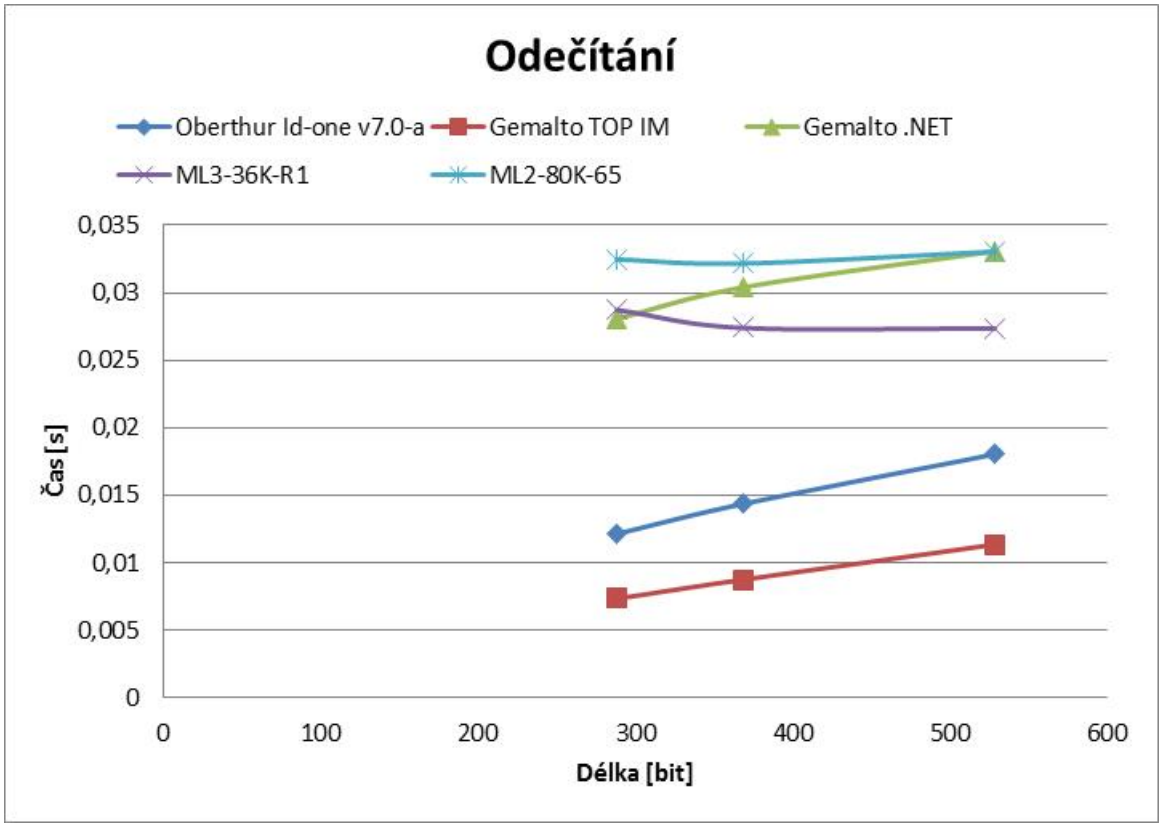

OKsystem

# Modulární mocnění



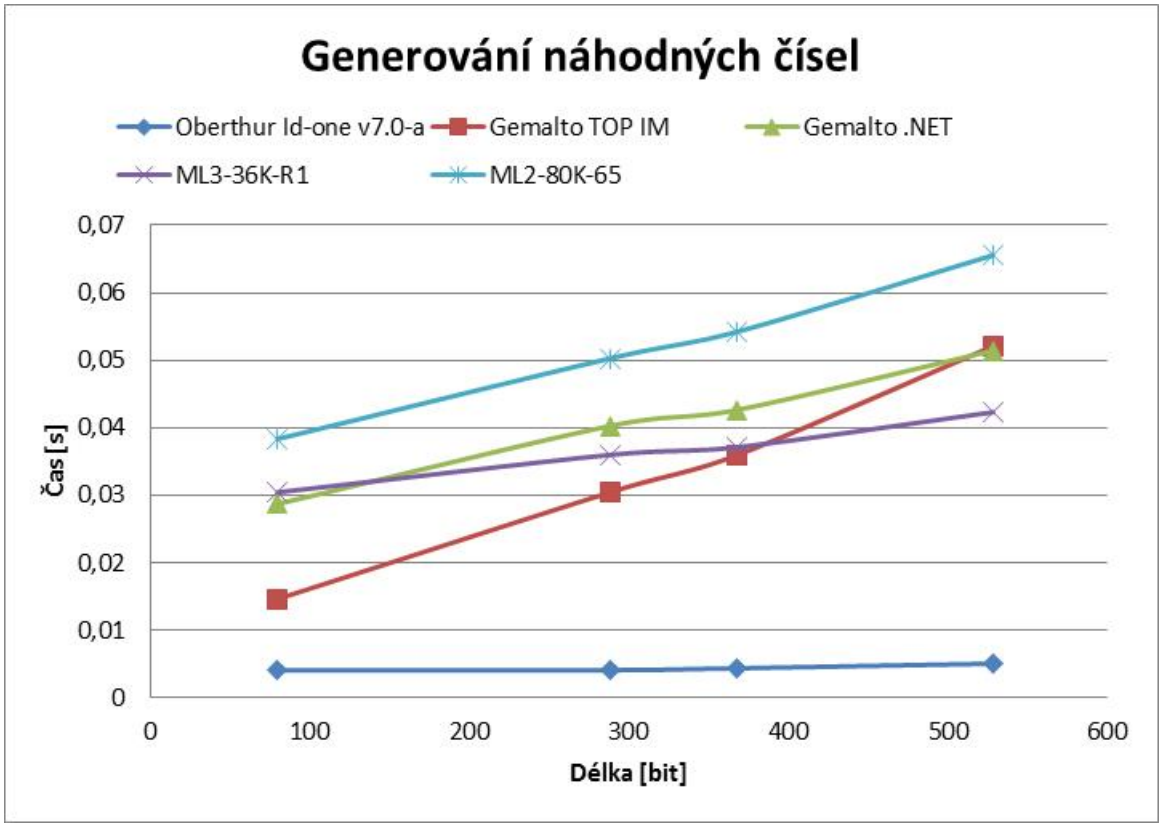

OKsystem

# Odečítání



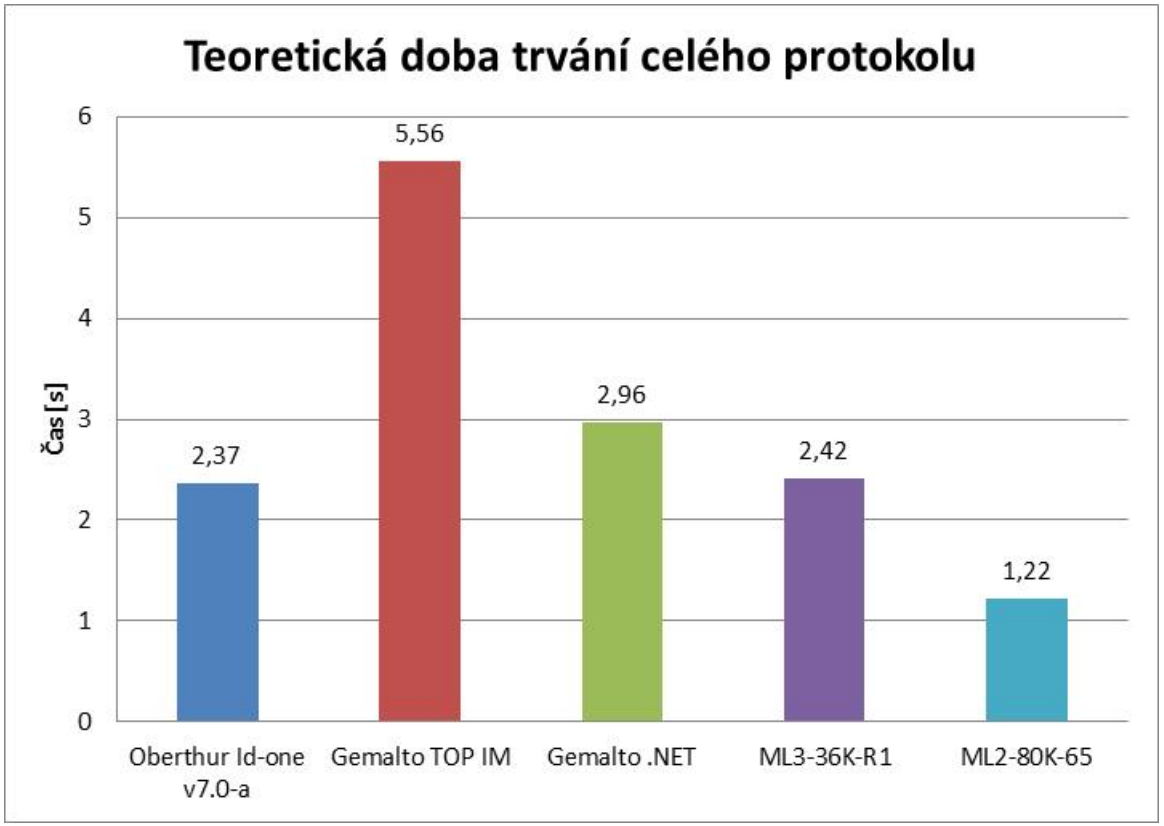

OKsystem

# Generování náhodných čísel




OKsystem

# Teoretická doba výpočtu celého algoritmu




OKsystem

# Zhodnocení

Celkovým vítězem pro implementaci protokolů ANAUT je platforma Multos, především díky knihovní podpoře násobení polí čísel. Gemalto .NET má výkonný 32 bitový procesor ale relativně málo výkonný RSA koprocesor. Gemalto TOP IM ztrácí především v modulárním mocnění – málo výkonný RSA koprocesor.

## Zhodnocení

Smart karty mají podporu pro algoritmus RSA ve formě RSA koprocesoru (samozřejmě i ECC, AES, DES). Tato podpora nestačí pro komplikovanější kryptografické algoritmy, i když ji lze „zneužít“ pro některé výpočty. Univerzálně použitelná kryptografická smart karta by měla mít knihovní podporu pro běžnou a modulární aritmetiku s poli čísel. Tomu se blíží (u nás málo známý a málo používaný) systém Multos, který jsme vybrali pro implementaci ANAUT.

OKsystem

**OKsystem s.r.o.**
Na Pankráci 125
140 21 Praha 4
tel: +420 236 072 111
www.oksystem.cz
www.oksystem.com
www.getbabel.com
www.okbase.cz
www.oksmart.cz

# Otázky?

Děkuji za pozornost

Ivo Rosol
rosol@oksystem.cz