New Algorithm for Classical Modular Inverse

Róbert Lórencz

CTU in Prague

CR

Introduction - Modular Inverse

- Inseparable part of cryptographic algorithms.
- Always needed classical modular inverse (CMI).
- Computation CMI over $GF(p)$ is based mainly on algorithms derived from Euclidean algorithm.
- Efficiency of computing CMI for large integers depends on adaptability of the algorithm to the architecture.

Algorithms solving CMI suitable for HW implementation

- Penk's binary algorithm (right-shift)
- Algorithm based on the Montgomery algorithm (right-shift)
- Proposed left-shift algorithm

All algorithms are based on solving

gcd with extended Euclidean algorithm.

Algorithm computing CMI

Euclidean Algorithm

p and a positive integer, gcd$(p,a)=1$, $p>a>0$

$$
\begin{aligned}
r_0 &= p \\
r_1 &= a \\
q_i &= \lfloor r_{i-2} / r_{i-1} \rfloor \\
r_i &= r_{i-2} - q_i r_{i-1} \\
0 &< r_i < r_{i-1} \\
f_i &= f_{i-2} - q_i f_{i-1} \\
g_i &= g_{i-2} - q_i g_{i-1} \\
a^{-1}\text{mod}(p) &= g_n \text{mod}(p)
\end{aligned}
$$

Starting conditions, guarding conditions, and recurrent equations for computing CMI.

Penk’s Algorithm for CMI

Description

i	q_i	*operations*	*va*		
0		r_0	1		
1		r_1	12	0	1
				/2=-6	(1+17)/2=9
				2)/2=-7	(3+17)/2=10
		$r_1/4$	3	-9	13

Conversion of odd integers g_i.

Conversion of negative integers g_i.

Guarding conditions

$r_i = r_{i-2} - qr_{i-1}$

$q_i = 2^x,\ q_i \leq 1$

x = (# LS zeros of r_{i-2}) – (# LS zeros of r_{i-1})

$0 < r_i < r_{i-1}$

if ($r_i < 0$) then

```
        $r_i = r_{i-1} - qr_{i-2},\ x := -x$
        $0 < r_i < r_{i-2}$
```

$g_i > 0$

)/4 – (17.12)/4

4

$$12^{-1} \bmod (17) = 10 \bmod (17) = 10$$

Montgomery Algorithm for CMI

Description

$r_2 = r_0 - q_2 r_1$

$r_2 = 17 - 1/4[12] = 14$

$(q_2^{-1})r_2 = r_0\,(q_2^{-1}) - r_1$

$(4)14 = 17(4) - 12(1)$

I. phase of the Montgomery Algorithm computes $2^k a^{-1} \bmod (p)$, where k is the number of deferred halvings.

Guarding conditions

$r_i = r_{i-2} - q_i\, r_{i-1}$

$q_i = 2^x,\ \ q_i \leq 1$

x = (# LS zeros of r_{i-2}) – (# LS zeros of r_{i-1}

$0 < r_i < r_{i-1}$

if ($r_i < 0$) then

$r_i = r_{i-1} - q_i\, r_{i-2}, \quad x := -x$

$0 < r_i < r_{i-2}$

~~$q_i > 0$~~

This condition is eliminated by multiplying equation $r_i = r_{i-2} - q_i r_{i-1}$ with q_i in each iteration. Then we obtain Diophantine equations $q_1^{-1}q_2^{-1}\ldots q_i^{-1}r_i = pf_i + ag_i$, where $q_1^{-1}q_2^{-1}\ldots q_i^{-1}$ induce deferred halvings.

$128 = -17(8) + 12(5 + 17)$

$2^7 12^{-1} \bmod (17) = 22 \bmod (17) = 5$

Drawbacks of previous algorithms

Summary

Both algorithms convert odd integers, and test conditions for performing operations +/- ($r_i > 0$).

Penk's Algorithm:

- conversions of odd and negative values (includes testing) ⇒ more +/- operations,
- conversions are carried out simultaneously with computing remainders ⇒ less shifts.

Montgomery Algorithm for CMI:

- computation without negative numbers ⇒ no conversions and testing ⇒ less +/- operations,
- computing $a^{-1} \bmod p$ in 2nd phase ⇒ conversion of odd integers (deferred halvings) in k iterations ⇒ more shifts steps.

New Left-shift (LS) Algorithm for CMI

Description

- It computes efficiently CMI without redundancies of arithmetical operations in extended Euclidean Algorithm.
- Left-shifting approach needs no conversions of odd or negative values.
- 2’s complementary code allows to work with negative integers and choose easily operations +/- in computing CMI.

New LS Algorithm for CMI

Description

$r_2 = r_0 - q_2 r_1$

$r_2 = 17 - 2[12] = -7$

$-7 = 17(1) - 12(2)$

$r_2 = pf_2 + ag_2$

$r_3 = r_1 + q_3 r_2$

$r_3 = 12 + 2[17(1) - 12$

$-2 = 17(2) - 12(3)$

$r_4 = r_2 - q_4 r_3$

$r_4 = 17(1) - 12(2) - 2[$

$-3 = -17(3) + 12(4)$

$r_5 = r_4 - q_4 r_3$

$r_5 = -17(3) + 12(4) - 2[17(2) - 12(3)] = -1$

$-1 = -17(5) + 12(7)$

Guarding conditions

$r_i = r_{i-2} \pm q_i r_{i-1}$

$q_i = 2^x,\ q_i \geq 1$

x = (# needed bits of r_{i-2}) – (# needed bits of r_{i-1})

$0 < |r_i| < |r_{i-1}|$ ⇒ negative integers r_i

if ($q_i < 0$) then ⇒ simple bit test

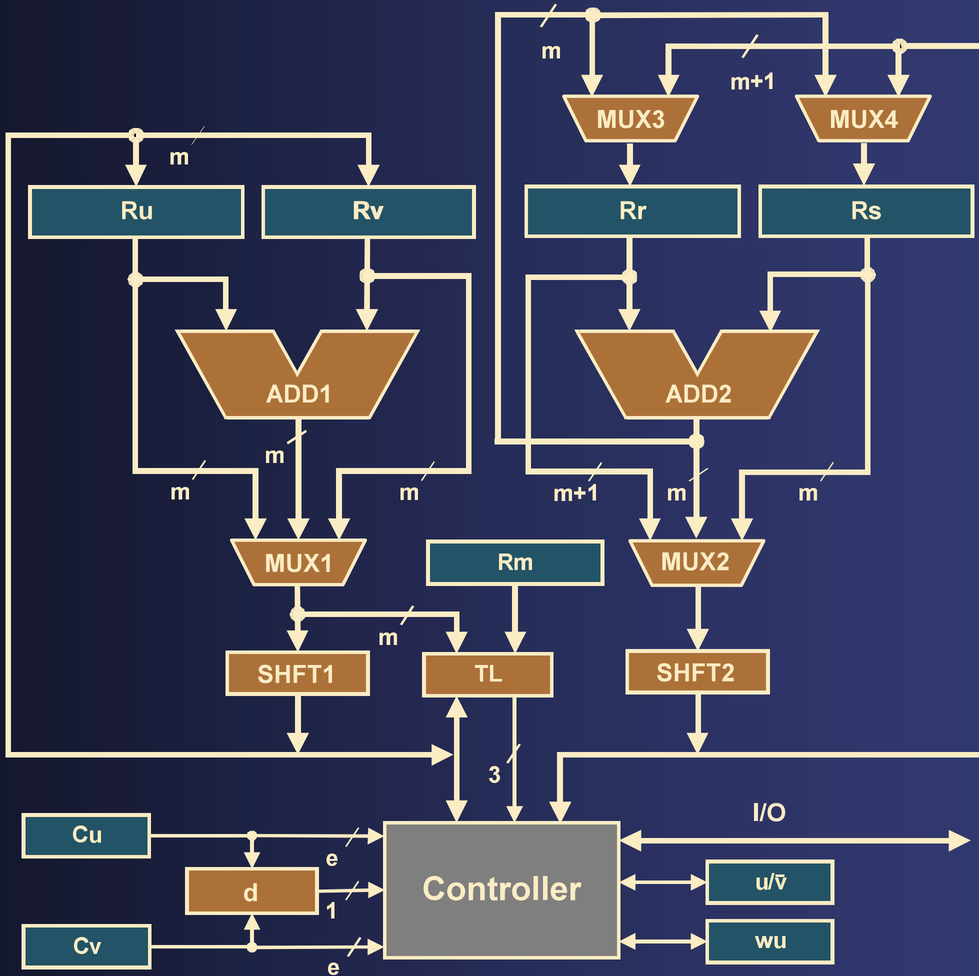
$r_i = r_{i-1} \pm qr_{i-2},\ x := -x$

$0 < |r_i| < |r_{i-2}|$

Operation +/- is chosen according to sign bits of operands.

$$a^{-1}\text{mod}(p) = (-g_5)\text{mod}(p) = (-7)\text{mod}(17) = 10$$

A circuit implementation of LS Algorithm



Registers, counters, and flip-flops, combination circuits.

Performance analysis and comparison

Simulation for $p < 2^{14}$

Valid only if special HW is employed.

Simulation of computation of CMI w 2^{14}.
More than 14.10^6 inverses was comp y each algorithm.

Algorithm	+/-		s s		+/- & tests	
	min, max	av.	mi max	av.	min, max	av.
LS	2-21	9.9	2-26	23.3	2-21	9.9
Montgomery	4-40	21.1	6-54	38.2	5-45	26.2
Penk's	6-53	27.1	2-26	18.1	9-80	40.4

- LS Algorithm is optimized for reducing the # of +/- operations.
- The +/- operations are critical in integer arithmetic due to carry propagation in long words.
- The table does not include tests $v > 0$ (this is essentially $v \neq 0$).

Performance analysis and comparison

LS Algorithm for 3 cryptographic primes

Primes	n	+/-		shifts		inverses
		min, max	av.	min, max	av.	
$2^{192} – 2^{64} – 1$	192	64-182	133	343-382	380	3,929,880
$2^{224} – 2^{96} + 1$	224	81-213	155	408-446	441	4,782,054
$2^{521} – 1$	521	18-472	388	999-1040	1029	4,311,179

- The average # of +/- operations approximately grows linearly with n. The multiplicative coefficient is ≈ 0.7 for all 3 primes.
- The average # of shifts is nearly $2n$.
- Similar results hold for primes $p < 2^{14}$.

Performance analysis and comparison
Summary

- Time complexity of a +/- operations increases approximately with $\log_2$(# of bits of a word), shift complexity remains constant.
- In case of >160 bit words the coefficient is >7 ⇒ LS Algorithm is:
  - 2x faster than Mongomery Algorithm and
  - 2.7x faster than Penk’s Algorithm.

Conclusion

- The new algorithm is always faster and in case of larger word lengths, it is at least 2x faster.
- ⇒ it is suitable for cryptographic systems.
- It was designed with the aim to allow easy and efficient HW implementation.
- The future work will concentrate on embedding into FPGA or ASIC circuitry used in cryptographic coprocessors, accelerators,etc.