Modular Multiplication Without Trial Division (Montgomery multiplication)

Wenbo Liu

Rial division is slow and hard to calculate for computer. The best way to deal this nuisance is to avoid modulo operation altogether.

Montgomery multiplication is not efficient for performing just one modular reduction and only becomes worthwhile when there is a chain of modular operations.

Montgomery Space

The space is defined by the modulo *n* and a positive integer $r \geq n$ coprime to n. The algorithm involves modulo and division by *r*, so in practice, *r* is chosen to be 2^{32} or 2^{64}, so that these operations can be done with a right-shift and a bitwise AND respectively.
For example:
The representative $\bar{x}$ of a number x in the Montgomery space is defined as

$$\bar{x} = x \cdot r \bmod n$$

Inside the Montgomery space, addition, subtraction, and checking for equality is performed as usual:

$$x \cdot r + y \cdot r \equiv (x + y) \cdot r \bmod n$$
$$\bar{x} * \bar{y} = \overline{x \cdot y} = (x \cdot y) \cdot r \bmod n$$
$$\bar{x} \cdot \bar{y} = (x \cdot y) \cdot r \cdot r \bmod n$$

Multiplication in Montgomery space is defined as:

$$\bar{x} * \bar{y} = \bar{x} \cdot \bar{y} \cdot r^{-1} \bmod n$$

Montgomery Reduction

The multiplication of two numbers in the Montgomery space requires an efficient computation of $x \cdot r^{-1} \bmod n$. This operation is called the Montgomery reduction.

```
function reduce(x):
    q = (x mod r) * n' mod r
    a = (x - q * n) / r
    if a < 0:
        a += n
    return a
```

Where $r \cdot r^{-1} + n \cdot n' = 1$ (Extended Euclidean Algorithm).

Process

1. Choose a base *R* which is coprime to *N*. *R* usually is 2^{32} or 2^{64}
2. Transforming numbers to the Montgomery space. $\bar{x} = x \cdot r \bmod n$
3. Modular multiplication.
4. Reduction.

Hardware Implementation

A hardware implementation can use a variation of these ideas to overlap the multiplication and reduction phases. Suppose $R = 2^n$ and N is odd. Let $x = (x_{n-1}x_{n-2} \dots x_0)_2$, where each x_i is 0 or 1. Let $0 \leq y < N$. To compute $xyR^{-1} \bmod N$, set $S_0 = 0$ and S_{i+1} to $(S_i + x_i y)/2$ or $(S_i + x_i y + N)/2$, whichever is an integer, for $i = 0, 1, 2, \dots, n-1$. By induction, $2^i S_i \equiv (x_{i-1} \dots x_0) y \bmod N$ and $0 \leq S_i < N + y < 2N$. Therefore, $xyR^{-1} \bmod N$ is either S_n or $S_n - N$.

Why good for hardware?

The algorithm doesn't need to calculate division and it can be implemented based on the iteration, which can be implemented efficiently on the hardware.