RSA Cryptography

Nikola Bežanić
Jelena Popović-Božović
Veljko Milutinović
Ivan Popović

University of Belgrade
School of Electrical Engineering

Introduction

- Public key cryptography
- Block algorithm for data encryption/decryption
- Prepare data for encryption
  - Split original message to blocks (integers)
  - Integer values between 0 and n-1 for some chosen n
  - E.g. n is encoded with 1024 bits
    - $0 \leq n < 2^{1024}$
    - 309 decimal digits
    - 256 hexadecimal digits
- Perform encryption through exponentiation

Basic expression

- RSA is asymmetric cryptosystem
  - Exponentiation expression

$$C = M^e \bmod n$$

    - M – plaintext
    - C – ciphertext
    - $\{e, n\}$ is a public key
  - Decryption

$$M = C^d \bmod n$$

    - $\{d, n\}$ is a private key
- Difficult to compute in a straightforward manner

Montgomery's method

- Uses modular multiplication, but replaces n with $r = 2^k$
- Arithmetic based on r is more suitable for CPU
- Number a in Montgomery's domain

$$a_m = ar \bmod n$$

- Montgomery's product

$$R_m = a_m b_m r^{-1} \bmod n$$

where R_m is Montgomery's image of R

$$R = ab \bmod n$$

Montgomery's method

- Expression r^{-1} satisfies next relation

$$rr^{-1} = 1 \bmod n$$

- Another intermediate variable is introduced (n')

$$rr^{-1} - nn' = 1$$

- This enables modulo *r* arithmetic
- Method is efficient for large number of products
- Otherwise there is an overhead
  of moving to Montgomery's domain
- Starting point for RSA acceleration on Maxeler

Montgomery's product: algorithm

- Numbers *a* and *b* (length: s*w bits)
- (s – num of digits; w-digit width);
- e.g. w=32 bits
- <u>a</u>– a in Montgomery's domain; r = 2^(s*w)

---

function *MonPro*($\underline{a}$, $\underline{b}$) ($r * r^{-1} - n * n' = 1$)

Step 1. $t := \underline{a} * \underline{b}$

Step 2. $m := t * n' \pmod r$

Step 3. $u := (t + m * n)/r$

Step 4. ***if*** $u \geq n$ ***then return*** $u - n$

else return u

Montgomery's method: algorithm

function *ModExp(M, e, n) { n is an odd number }*

Step 1. *Compute n' using the extended Euclidean algorithm.*

Step 2. $\mu := M*r \pmod n$

Step 3. $\rho := 1*r \pmod n$

Step 4. ***for*** *i = k - 1* ***down to*** *0* ***do***

Step 5. $\rho := MonPro(\rho, \rho)$

Step 6. ***if*** $e_i = 1$ ***then*** $\rho := MonPro(\mu, \rho)$

Step 7. $\rho := MonPro(\rho, 1)$

Step 8. ***return*** ρ

$$e = \sum_{i=0}^{k-1} e_i 2^i$$