Lab 1

Multiplicative inverse, Modular exponentiation and the **RSA** cryptosystem

Greatest Common Divisor (GCD) and the Euclidean Algorithm

Compute gcd(57, 93) using the Euclidean algorithm, and find integers **s** and **t** such that $57s + 93t = \gcd(57, 93)$.

Multiplicative inverse and the EXTENDED EUCLIDEAN ALGORITHM

Compute the following multiplicative inverses:

(a) $17^{-1} \bmod 101$

(b) $357^{-1} \bmod 1234$

(c) $3125^{-1} \bmod 9987$.

The RSA Cryptosystem: an example

Suppose Bob chooses $\boldsymbol{p}$ **= 101** and $\boldsymbol{q}$ **= 113**. Then $\boldsymbol{n}$ **= 11413** and $\varphi(n) = 100 \times 112 = 11200$. Since $11200 = 2^6 5^2 7$, an integer $\boldsymbol{b}$ can be used as an **encryption exponent** if and only if $\boldsymbol{b}$ is not divisible by 2, 5, or 7.

(In practice, however, Bob will not factor $\varphi(n)$. He will verify that $\gcd(\varphi(n), b) = 1$ using the Euclidean Algorithm, <u>slide #7</u>)

The RSA Cryptosystem: an example

Suppose Bob chooses $\boldsymbol{b}$ **= 3533**.

- Please verify that $\gcd(\varphi(n), b) = 1$ using the Euclidean Algo.
- Now compute Bob's secret decryption exponent, a, using the Multiplicative Inverse Algorithm (Algorithm 6.3, at slide #9).

Bob publishes $n = 11413$ and $b = 3533$ in a directory.

The RSA Cryptosystem: an example

Now, suppose Alice wants to encrypt the plaintext 9726 to send to Bob. She will compute $9726^{3533} \bmod 11413$ and send the ciphertext **c** over the channel.

- Please determine **c**'s value using the square and multiply algorithm (Algorithm 6.5 at slide #10)

When Bob receives the ciphertext **c**, he uses his secret decryption exponent **a** to compute the plaintext sent by Alice.

Algorithm 6.1: EUCLIDEAN ALGORITHM(a, b) **Computation of gcd(a,b)**

$r_0 \leftarrow a$
$r_1 \leftarrow b$
$m \leftarrow 1$
while $r_m \neq 0$

$$\textbf{do} \begin{cases} q_m \leftarrow \left\lfloor \frac{r_{m-1}}{r_m} \right\rfloor \\ r_{m+1} \leftarrow r_{m-1} - q_m r_m \\ m \leftarrow m + 1 \end{cases}$$

$m \leftarrow m - 1$
return $(q_1, \ldots, q_m; r_m)$
comment: $r_m = \gcd(a, b)$

Algorithm 6.2: EXTENDED EUCLIDEAN ALGORITHM(a, b)

$a_0 \leftarrow a$
$b_0 \leftarrow b$
$t_0 \leftarrow 0$
$t \leftarrow 1$
$s_0 \leftarrow 1$
$s \leftarrow 0$
$q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
$r \leftarrow a_0 - qb_0$
while $r > 0$

$$\textbf{do} \begin{cases} temp \leftarrow t_0 - qt \\ t_0 \leftarrow t \\ t \leftarrow temp \\ temp \leftarrow s_0 - qs \\ s_0 \leftarrow s \\ s \leftarrow temp \\ a_0 \leftarrow b_0 \\ b_0 \leftarrow r \\ q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor \\ r \leftarrow a_0 - qb_0 \end{cases}$$

$r \leftarrow b_0$
return (r, s, t)
comment: $r = \gcd(a, b)$ and $sa + tb = r$

Algorithm 6.3: MULTIPLICATIVE INVERSE(a, b)

Computation of b^{-1} mod a

$a_0 \leftarrow a$
$b_0 \leftarrow b$
$t_0 \leftarrow 0$
$t \leftarrow 1$
$q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
$r \leftarrow a_0 - qb_0$
while $r > 0$

$$\textbf{do} \begin{cases} temp \leftarrow (t_0 - qt) \bmod a \\ t_0 \leftarrow t \\ t \leftarrow temp \\ a_0 \leftarrow b_0 \\ b_0 \leftarrow r \\ q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor \\ r \leftarrow a_0 - qb_0 \end{cases}$$

if $b_0 \neq 1$
then b has no inverse modulo a
else return (t)

Computation of x^c *mod n*

Algorithm 6.5: SQUARE-AND-MULTIPLY(x, c, n)

$z \leftarrow 1$
for $i \leftarrow \ell - 1$ **downto** 0

$$\textbf{do} \begin{cases} z \leftarrow z^2 \bmod n \\ \textbf{if } c_i = 1 \\ \quad \textbf{then } z \leftarrow (z \times x) \bmod n \end{cases}$$

return (z)

References

1. William Stallings, Lawrie Brown, **Computer Security Principles and Practice**,
2. William Stallings, **Cryptography and Network Security: Principles and Practice**,
3. Douglas R. Stinson, Maura B. Paterson, **Cryptography Theory and Practice**,
4. Dan Boneh, Victor Shoup, **A Graduate Course in Applied Cryptography**.