

HW # 9

Solution

1. Compute the following exponentiations $x^e \bmod m$ applying the square-and-multiply algorithm (also called left-to-right binary exponentiation). After every iteration step, show the exponent of the intermediate result in binary notation.

(a) $x = 5, e = 83, m = 101, e = 1010011_2$

y	Operation	exponent
$y = 5 \bmod 101$	NOP	1
$y = 5^2 = 25 \bmod 101$	SQR	10
$y = 25^2 = 19 \bmod 101$	SQR	100
$y = 19 \cdot 5 = 95 \bmod 101$	MUL	101
$y = 95^2 = 36 \bmod 101$	SQR	1010
$y = 36^2 = 84 \bmod 101$	SQR	10100
$y = 84^2 = 87 \bmod 101$	SQR	101000
$y = 87 \cdot 5 = 31 \bmod 101$	MUL	101001
$y = 31^2 = 52 \bmod 101$	SQR	1010010
$y = 52 \cdot 5 = 58 \bmod 101$	MUL	1010011

(b) $x = 7, e = 199, m = 101, e = 11000111_2$

y	Operation	exponent
$y = 7 \bmod 101$	NOP	1
$y = 7^2 = 49 \bmod 101$	SQR	10
$y = 49 \cdot 7 = 40 \bmod 101$	MUL	11
$y = 40^2 = 85 \bmod 101$	SQR	110
$y = 85^2 = 54 \bmod 101$	SQR	1100
$y = 54^2 = 88 \bmod 101$	SQR	11000
$y = 88^2 = 68 \bmod 101$	SQR	110000
$y = 68 \cdot 7 = 72 \bmod 101$	MUL	110001
$y = 72^2 = 33 \bmod 101$	SQR	1100010
$y = 33 \cdot 7 = 29 \bmod 101$	MUL	1100011
$y = 29^2 = 33 \bmod 101$	SQR	11000110
$y = 33 \cdot 7 = 29 \bmod 101$	MUL	11000111

2. Write a C program which computes $x^e \bmod n$ using the square-and-multiply algorithm (also called left-to-right binary exponentiation). Two tips:
- Use `long` type variables for all operations.
 - Make sure you apply the modulo reduction in *every* step of the algorithm. Otherwise you run very rapidly out of the 32 bit range covered by long type variables.
- (a) Provide a print out of the C program as well as a sample run.
- (b) Use your program to verify your results from Problem 1.

No solution given.

3. As we saw in the lecture, the modulus of RSA has been enlarged over the years in order to thwart the dramatically improved attacks. As one would assume, public-key algorithms are becoming slower as the word length increases. We will study the relation between word length and performance in this problem. The performance of RSA, and of almost any other public-key algorithm, is dependent on how fast modulo exponentiation with large numbers can be performed.

Assume that one modulo multiplication or squaring with k bit numbers takes $c \cdot k^2$ clock cycles, where c is a constant. How much slower is RSA encryption/decryption with 1024 bits compared to RSA with 512 on average. Only consider the encryption/decryption itself with an exponent of full length and the square-and-multiply algorithm.

A 1024-bit RSA encryption/decryption requires exponentiation of a 1024-bit number to the power of a 1024-bit number. By using square-and-multiply algorithm 1024-bit exponentiation requires on average 1024 squarings and 512 multiplications, which takes $1536 \cdot c \cdot 1024^2$ clock cycles.

A 512-bit RSA encryption/decryption on the other hand requires exponentiation of a 512-bit number to the power of a 512-bit number. By using square-and-multiply algorithm this requires on average 512 squarings and 256 multiplications, which takes $768 \cdot c \cdot 512^2$ clock cycles.

The time ratio is

$$\frac{1536 \cdot c \cdot 1024^2}{768 \cdot c \cdot 512^2} = \frac{768 \cdot c \cdot 2 \cdot 2^2 \cdot 512^2}{768 \cdot c \cdot 512^2} = 8$$

Hence, 1024-bit RSA is 8 times slower than 512-bit RSA.

4. There are ways to improve the square-and-multiply algorithm, that is, to reduce the number of operations required. Although the number of squaring is fixed, the number of multiplications can be reduced. Your task is to come up with a modified version of the square-and-multiply algorithm (also called left-to-right binary exponentiation) which requires fewer multiplications. Give a detailed description of how the new algorithm works and what the complexity is (number of operations).

Hint: Try to develop a generalization of the square-and-multiply algorithm which processes more than one bit at a time. The basic idea is to handle k (e.g., $k = 3$) exponent bits per iteration rather than one bit in the original square-and-multiply algorithm.

Assume we use a window of $k = 3$ exponent bits to iterate through the binary representation of the exponent. We have 7 possible bit patterns that might appear in the window: 000, 001, 010, 011, 100, 101, 110, 111.

The algorithm first requires that these powers of the base are precomputed (e.g., $x^0, x^1, x^2, x^3, x^4, x^5, x^6, x^7$).

Assuming the exponent is divided into l consecutive k bit windows as $e = (e_{l-1}, e_{l-2}, \dots, e_1, e_0)$ the algorithm builds the modular exponentiation from the precomputed values as follows:

$$\begin{aligned}
 y &\leftarrow 1 \\
 y &\leftarrow y \cdot x^{e_{l-1}} \\
 y &\leftarrow y^{2^k} \\
 y &\leftarrow y \cdot x^{e_{l-2}} \\
 y &\leftarrow y^{2^k} \\
 &\dots \\
 y &\leftarrow y \cdot x^{e_1} \\
 y &\leftarrow y^{2^k} \\
 y &\leftarrow y \cdot x^{e_0}
 \end{aligned}$$

Note that after the final iteration the exponent e is fully assembled. The construction did not make any assumptions about the size of k . Hence it will work for any k .

5. In this problem we are examining the effect of the Chinese Remainder Theorem on RSA decryption. Assume we have RSA with the following parameters: $p = 11$, $q = 13$, and $d = 113$.

- (a) Decrypt the message $y = 40$ as $x = y^d \pmod n$ on paper using square-and-multiply algorithm. Show every single step. For multiplication and squaring you can use any method you like and it is sufficient to show just the result of the multiplication and squaring operations.

$$y = 40, d = 113, m = p \cdot q = 11 \cdot 13 = 143, d = 1110001_2$$

x	Operation	exponent
$x = 40 \pmod{143}$	NOP	1
$x = 40^2 = 27 \pmod{143}$	SQR	10
$x = 27 \cdot 40 = 79 \pmod{143}$	MUL	11
$x = 79^2 = 92 \pmod{143}$	SQR	110
$x = 92 \cdot 40 = 105 \pmod{143}$	MUL	111
$x = 105^2 = 14 \pmod{143}$	SQR	1110
$x = 14^2 = 53 \pmod{143}$	SQR	11100
$x = 53^2 = 92 \pmod{143}$	SQR	111000
$x = 92^2 = 27 \pmod{143}$	SQR	1110000
$x = 27 \cdot 40 = 79 \pmod{143}$	MUL	1110001

- (b) How many multiplications and how many squaring operations did you have to perform, how many total?
 6 squarings and 3 multiplications leading to a total of 9 operations.
- (c) Recompute y using the Chinese Remainder Theorem.

$$y = 40, d = 113, p = 11, q = 13, n = p \cdot q = 143$$

$$\text{For } p : y_p = y \bmod p = 40 \bmod 11 = 7, d_p = d \bmod (p - 1) = 113 \bmod 10 = 3 = 11_2$$

x_p	Operation	exponent
$x_p = 7 \bmod 11$	NOP	1
$x_p = 7^2 = 5 \bmod 11$	SQR	10
$x_p = 5 \cdot 7 = 2 \bmod 11$	MUL	11

$$\text{For } q : y_q = y \bmod q = 40 \bmod 13 = 1, d_q = d \bmod (q - 1) = 113 \bmod 12 = 5 = 101_2$$

x_q	Operation	exponent
$x_q = 1 \bmod 13$	NOP	1
$x_q = 1^2 = 1 \bmod 13$	SQR	10
$x_q = 1^2 = 1 \bmod 13$	SQR	100
$x_q = 1 \cdot 1 = 1 \bmod 13$	MUL	101

$$x = x_p \cdot q \cdot \underbrace{[q^{-1} \bmod p]}_{r_q} + x_q \cdot p \cdot \underbrace{[p^{-1} \bmod q]}_{r_p} \bmod n$$

$$x = 2 \cdot 13 \cdot [13^{-1} \bmod 11] + 1 \cdot 11 \cdot [11^{-1} \bmod 13] \bmod 143$$

$$x = 2 \cdot 13 \cdot 6 + 1 \cdot 11 \cdot 6 \bmod 143$$

$$x = 2 \cdot 78 + 1 \cdot 66 = 79$$

- (d) How many multiplications, how many squarings and how many operations in total did you have to perform now?
 1 squaring and 1 multiplication for x_p
 3 squarings and 1 multiplication for x_q
 2 multiplications to compute the final x , assuming r_p and r_q are precomputed
 Total number of squarings is 4 and multiplications is 4, total of 8 operations.

6. Use a spreadsheet (such as Excel), or a calculator, and test all odd numbers in the range from 219 to 233 for primality using the Miller-Rabin test with base $a = 2$. Document results of all intermediate modular multiplications. Determine the number of modular multiplications.

Let b be the number of bits necessary to represent r in binary. Then the number of modular multiplications m required to compute $(a^r)^{2^s}$ computes as:

$$m = b - 1 + \text{number of 1s}(b) - 1 + s$$

on average (i.e. r in binary representation contains as many “1”s as “0”s)

$$m = b - 1 + \frac{1}{2}(b - 1) + s = 1.5 \cdot (b - 1) + s$$

n	$n - 1$ Binary	s	r	$(a^r)^{2^0}$	$(a^r)^{2^1}$	$(a^r)^{2^2}$	$(a^r)^{2^3}$	$(a^r)^{2^4}$	$(a^r)^{2^5}$	Result	Reason	Num Mult.
219	11011010	1	109	2	4					composite	no '1'	12
221	11011100	2	55	128	30	16				composite	no '1'	12
223	11011110	1	111	1	1					prime	$\sqrt{1} = 1$	13
225	11100000	5	7	128	184	106	211	196	166	composite	no '1'	10
227	11100010	1	113	-1	1					prime	$\sqrt{1} = -1$	11
229	11100100	2	57	122	-1	1				prime	$\sqrt{1} = -1$	11
231	11100110	1	115	65	67					composite	no '1'	12
233	11101000	3	29	1	1	1	1			prime	$\sqrt{1} = 1$	11