Why RSA Works

Peter Franušić *

Arthur C. Clarke once quipped that "any sufficiently advanced technology is indistinguishable from magic." Cryptography is the magic that transmogrifies a meaningful message into gibberish and then back again. For thousands of years, military-grade cryptography was the exclusive domain of diplomats and generals, partly due to the high cost of keeping secret keys secret. But around 1975 something happened to change all that: *public-key* cryptography was invented. Public-key cryptography dramatically reduces the cost of secret key management.

The Rivest-Shamir-Adleman algorithm (RSA) is a well-established computational method for public-key cryptography.[1] We offer the reader an understanding of why RSA works. A simple proof of the RSA identity is developed using an illustrative approach. Table 5 is particularly revealing.

The scope of the article is limited to understanding the RSA identity. The discussion therefore omits related topics such as multi-prime RSA, key generation, conversion of text to integers and integers to text, padding of cleartext messages, various speed-ups such as Montgomery reduction and the Chinese remainder theorem, and the latest factoring algorithms. RSA authentication is not covered because the math is identical to that of RSA encryption.

The presentation differs in several ways from conventional treatments of the RSA algorithm. The algebraic equations utilize ring notation and equal signs rather than modular notation and equivalence signs. Instead of Euler's totient function and Fermat's little theorem, a proof of the RSA identity employs the Carmichael function and a corollary from the literature.[2]

 Email: pete@sargo.com

1 Huge integers

Figure 1 shows an RSA cryptosystem. It consists of a transmitter, a receiver, and a sniffer in between. Tradition has it that *Bob* is the transmitter, *Alice* is the receiver, and *Eve* is the sniffer. (*Eavesdropper*, get it?) The twin engines of the system are the modular exponentiation (modex) functions.

RSA uses *huge* integers. By huge we mean integers with over 300 decimal digits. It would take over four lines to print a 300 digit integer on this page. Happily, we can represent huge integers with single letters. Each of the letters in the figure represents a huge integer. These are: message m, modulus n, encryptor e, ciphertext c, decryptor d, and output y.

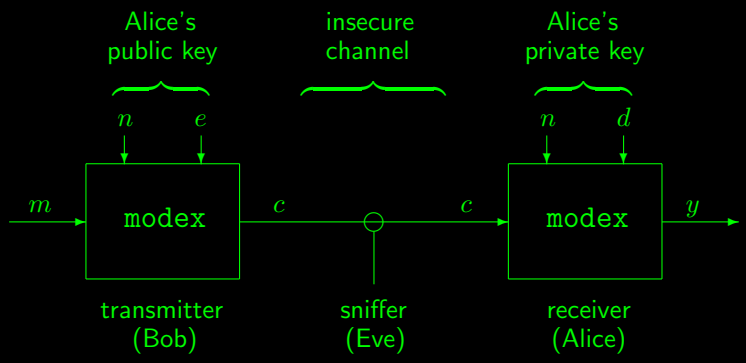

Figure 1: An RSA cryptosystem

Bob generates encrypted messages and transmits them to Alice. He originates message m, computes the modex function using modulus n and encryptor e, then writes ciphertext c into the insecure channel. The public key (n, e) was generated and published by Alice prior to any transmission by Bob.

Alice receives encrypted messages from Bob and decrypts them. She reads ciphertext c from the insecure channel, computes the modex function using modulus n and decryptor d, then writes output y. The magic of RSA is that output y is identical to message m. I.e., $y = m$. Alice generated and secured her private key (n, d) prior to receiving any ciphertext from Bob.

Eve is the threat that exists on every insecure channel. She attempts to read messages that are meant to be read only by Alice. But Eve is defeated by RSA encryption. She can intercept ciphertext c but she won't be able to compute y because she doesn't have access to decryptor d. Only Alice has access to decryptor d.

2 Simulation

We can simulate RSA messaging using a Lisp interpreter. Our version of Lisp uses a question mark for the prompt. Lisp commands may be variable names or compound expressions. The interpreter reads each command, evaluates it, and prints the result.

Alice must precompute three integers before Bob can send her RSA encrypted messages. These are modulus n, encryptor e, and decryptor d. Some examples have been precomputed and are displayed below in decimal format. Note that modulus n has 56 decimal digits. This turns out to be 186 bits. A typical RSA modulus has at least 1024 bits, but we use 186 bits here for the sake of brevity.

```
? n
97397795163266888271167242107545263613895906874319587249
? e
10306926753200670273346978999454444249925952109333797079
? d
46445936998769783647957439537275126296124161350172130481
```

A typical RSA message is an AES-128 session key. This is a random 128-bit integer used by the AES algorithm to encrypt a high bandwidth session. Here m has 39 decimal digits, or 128 bits. Normally m would be padded with extra random bits to make it about the same size as n, but we've omitted padding steps in this demonstration for the sake of clarity.

```
? m
325004947599823818213341565111207349415
```

Bob commands his Lisp interpreter to compute ciphertext c. Lisp computes the modex function with inputs m, e, n, assigns this value to the variable c, then prints the result. Note that, whereas m has only 39 digis, ciphertext c has 56 digits, the same as n.

```
? (setf c (modex m e n))
65406940630722215589598713946252700262213080283568050086
```

Alice commands her Lisp interpreter to compute output y. Lisp computes the modex function with inputs c, d, n, assigns this value to the variable y, then prints the result. Note that y is identical to m. *Magic!*

```
? (setf y (modex c d n))
325004947599823818213341565111207349415
```

3 Rings

RSA uses mathematical structures called rings. A *ring* is a set equipped with two binary operators.[3] The ring displays several well-defined algebraic properties, including both additive closure and multiplicative closure.

Recall that a set is simply a collection of elements. These elements can be anything, but in the case of RSA, the elements are integers. RSA uses sets with a finite number of elements. The number of elements in a set is called the *modulus*. The modulus is represented by the symbol n.

A binary operator is something that takes two elements and computes a third. Rings use two binary operators, which we denote here as $\oplus$ (pronounced OH plus) and $\otimes$ (pronounced OH times). The $\oplus$ operator is similar to addition. The $\otimes$ operator is similar to multiplication.

In general, we say that the ring $\mathcal{R}_n$ consists of the set Z_n, the $\oplus$ operator, and the $\otimes$ operator.

$$\mathcal{R}_n = (Z_n, \oplus, \otimes)$$

4 The set Z_n

A finite set Z_n can be specified in several different ways. When a set has just a few elements, they can be explicitly enumerated, listed within curly brackets. For example, the set Z_{15} consists of the 15 integers starting with 0 and ending with 14.

$$Z_{15} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$$

When a set has a huge number of elements, they cannot be enumerated. But if a set consists entirely of sequential elements, it can be specified by listing the first few elements, an ellipsis, and the last few elements. For example, the set Z_n consists of a sequence of n integers, starting with 0 and ending with $(n-1)$.

$$Z_n = \{0, 1, 2, 3, \ldots, (n-2), (n-1)\}$$

When RSA generates a pair of keys, it selects some modulus n that is the product of two distinct primes p and q. The term *product* means that we multiply p times q. Instead of writing $p \times q$ we use the abbreviation pq.

$$n = pq$$

The term *distinct* means that p and q are different from each other. That is, $p \neq q$. Recall that a *prime* is any integer greater than 1 that cannot be divided evenly by any other integer except 1 and itself. The first five primes are 2, 3, 5, 7, and 11. In the example of Z_{15} above, the modulus 15 is the product of the two distinct primes 3 and 5.

5 The $\oplus$ operator

When n is small, the $\oplus$ operator can be specified using a table. Table 1 specifies the $\oplus$ operator for the ring $\mathcal{R}_{15}$. The table is a 15 by 15 block of integers. There are 15 rows and 15 columns. Recall that rows run left and right, columns run up and down. Row numbers are specified by the extra column along the left side of the block. Column numbers are specified by the extra row along the top of the block. Note the diagonal stripe pattern that is visible in the table.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	0
2	2	3	4	5	6	7	8	9	10	11	12	13	14	0	1
3	3	4	5	6	7	8	9	10	11	12	13	14	0	1	2
4	4	5	6	7	8	9	10	11	12	13	14	0	1	2	3
5	5	6	7	8	9	10	11	12	13	14	0	1	2	3	4
6	6	7	8	9	10	11	12	13	14	0	1	2	3	4	5
7	7	8	9	10	11	12	13	14	0	1	2	3	4	5	6
8	8	9	10	11	12	13	14	0	1	2	3	4	5	6	7
9	9	10	11	12	13	14	0	1	2	3	4	5	6	7	8
10	10	11	12	13	14	0	1	2	3	4	5	6	7	8	9
11	11	12	13	14	0	1	2	3	4	5	6	7	8	9	10
12	12	13	14	0	1	2	3	4	5	6	7	8	9	10	11
13	13	14	0	1	2	3	4	5	6	7	8	9	10	11	12
14	14	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Table 1: $a \oplus b \quad (\mathcal{R}_{15})$

Table 1 specifies the value of $a \oplus b$ for every possible pair of a and b. For example, let $a = 10$ and $b = 12$. The value of $10 \oplus 12$ is specified at the intersection of row 10 and column 12. This value is 7. Therefore $10 \oplus 12 = 7$.

Notice that every element in the table is in the set Z_{15}. This demonstrates the *additive closure* property of rings. The additive closure property states that for every pair of elements a and b in Z_n, the sum $a \oplus b$ is also an element in Z_n.

$$a \oplus b \in Z_n$$

The value of $a \oplus b$ can also be specified using a rule. To compute $10 \oplus 12$ we first calculate $10 + 12$ to get 22. Since 22 is not an element in Z_{15} we subtract the modulus 15, i.e. $22 - 15 = 7$. Since $7 \in Z_{15}$ we stop and 7 is our final result.

In general, the $\oplus$ operator takes two integers a and b, adds them together using normal addition, then subtracts some multiple of n such that the final value is in Z_n. The term kn signifies some multiple of n. That is, $kn = 0n, 1n, 2n, 3n, \ldots$ We simply use whichever kn works in order to get closure, where $a \oplus b \in Z_n$.

$$a \oplus b = a + b - kn$$

6 The $\otimes$ operator

When n is small, the $\otimes$ operator can be specified using a table. Table 2 specifies the $\otimes$ operator for the ring $\mathcal{R}_{15}$. The format is the same as Table 1. The values, of course, are different. Note the rose-like pattern visible in the table. The table is symmetrical about the diagonals. If we ignore column 0, then row 1 is the reverse of row 14, row 2 is the reverse of row 13, etc.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	0	2	4	6	8	10	12	14	1	3	5	7	9	11	13
3	0	3	6	9	12	0	3	6	9	12	0	3	6	9	12
4	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11
5	0	5	10	0	5	10	0	5	10	0	5	10	0	5	10
6	0	6	12	3	9	0	6	12	3	9	0	6	12	3	9
7	0	7	14	6	13	5	12	4	11	3	10	2	9	1	8
8	0	8	1	9	2	10	3	11	4	12	5	13	6	14	7
9	0	9	3	12	6	0	9	3	12	6	0	9	3	12	6
10	0	10	5	0	10	5	0	10	5	0	10	5	0	10	5
11	0	11	7	3	14	10	6	2	13	9	5	1	12	8	4
12	0	12	9	6	3	0	12	9	6	3	0	12	9	6	3
13	0	13	11	9	7	5	3	1	14	12	10	8	6	4	2
14	0	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Table 2: $a \otimes b \quad (\mathcal{R}_{15})$

Table 2 specifies the value of $a \otimes b$ for every possible pair of a and b. For example, let $a = 11$ and $b = 8$. The value of $11 \otimes 8$ is specified at the intersection of row 11 and column 8. This value is 13. Therefore $11 \otimes 8 = 13$.

Notice that every element in the table is in the set Z_{15}. This demonstrates the *multiplicative closure* property of rings. The multiplicative closure property states that for every pair of elements a and b in Z_n, the product $a \otimes b$ is also an element in Z_n.

$$a \otimes b \in Z_n$$

The value of $a \otimes b$ can also be specified using a rule. To compute $11 \otimes 8$ we first calculate 11×8 to get 88. Since 88 is not an element in Z_{15} we subtract a multiple of the modulus, kn. In this case, $kn = 5 \times 15 = 75$. Therefore $88 - 75 = 13$. And $13 \in Z_{15}$ so 13 is our final result.

In general, the $\otimes$ operator takes two integers a and b, multiplies them together using normal multiplication, then subtracts some multiple of n such that the final value is in Z_n. In other words, we subtract whichever kn works in order to get closure, where $a \otimes b \in Z_n$.

$$a \otimes b = ab - kn$$

7 Exponential notation

Let's say we're given three elements a, b, c which are members of the set Z_n. We're also given the expression $a \otimes b \otimes c$. The question is this: How do we compute this expression? Do we first multiply a and b and then multiply c? Or do we multiply b and c and then multiply a? The answer is that either way is correct. It doesn't matter what order we multiply the elements. This is because the ring $\mathcal{R}_n$ has the property of *multiplicative association*. The multiplicative association property says that when we have a series of $\otimes$ operations, we can do the operations in whatever order we want. The answer will be the same.

$$\begin{aligned} a \otimes b \otimes c &= (a \otimes b) \otimes c \\ &= a \otimes (b \otimes c) \end{aligned}$$

The modex function is represented mathematically using *exponential notation*. Exponential notation is an efficient way to describe a series of multiplications of the same value. For example, the value m can be multiplied by itself any number of times. We use exponential notation to describe this. Remember that it doesn't matter in what order the m's are multiplied together.

$$\begin{aligned} \overbrace{m}^{1} &= m^1 \\ \overbrace{m \otimes m}^{2} &= m^2 \\ \overbrace{m \otimes m \otimes m}^{3} &= m^3 \\ \overbrace{m \otimes m \otimes m \otimes m}^{4} &= m^4 \\ &\vdots \end{aligned}$$

RSA uses the exponential notation m^e. The value m is the *message* integer. The value e is the *encryptor* exponent. The exponential notation m^e means that e copies of m are multiplied together using the $\otimes$ operator in the ring $\mathcal{R}_n$.

$$m^e = \overbrace{m \otimes m \otimes m \cdots \otimes m \otimes m}^{e}$$

RSA also uses the exponential notation c^d. The value c is the *ciphertext* integer. The value d is the *decryptor* exponent. The exponential notation c^d means that d copies of c are multiplied together using the $\otimes$ operator in the ring $\mathcal{R}_n$.

$$c^d = \overbrace{c \otimes c \otimes c \cdots \otimes c \otimes c}^{d}$$

8 The modex function

The term *modex* is a contraction of modular exponentiation. The modex function performs exponentiation in the ring $\mathcal{R}_n$. It performs the equivalent of a series of $\otimes$ operations.

The RSA cryptosystem in Figure 1 uses two modex functions: one in the transmitter and the other in the receiver. Both modex functions have three inputs and one output. We specify the output equation for each.

Receiver output equation: The receiver's modex function takes the inputs c, n, d and computes the output y. The modex output y is the equivalent of d copies of c multiplied together using the $\otimes$ operator in the ring $\mathcal{R}_n$.

$$y = c^d \tag{1}$$

Transmitter output equation: The transmitter takes the inputs m, n, e and computes the output c. The modex output c is the equivalent of e copies of m multiplied together using the $\otimes$ operator in the ring $\mathcal{R}_n$.

$$c = m^e \tag{2}$$

The modex function doesn't actually multiply e copies of m in order to compute m^e. This would take eons for huge values of e. Instead, modex actually uses a method called *square-and-multiply*. A register r is first initialized to m. Then it's repeatedly squared $(r \otimes r)$ and multiplied $(r \otimes m)$ depending on the number of bits in e and the value of each bit. For example, if e is 1024 bits long, there'll be 1023 squares and about 512 multiplies. A lot less than 2^{1024} multiplies.

The modex function uses the $\otimes$ operator in the ring $\mathcal{R}_n$. Recall that the $\otimes$ operator takes two integers, multiplies them, then subtracts some multiple of n so that the result is in Z_n. That is, $a \otimes b = ab - kn$. The subtraction step is called *reduction* and may be implemented by taking the remainder of a division. The product ab is divided by n, the quotient is k, and the remainder is $ab - kn$. But division is time consuming, and most modex implementations do not use division for the reduction step. Instead, they use a faster method called *Montgomery reduction*, which replaces slow divisions with fast truncations.

The modex function can be very time-consuming to compute. Square-and-multiply and Montgomery reduction are two *speed-ups* that are used to shorten the compute time. There are others. The enumeration and details of these speed-ups are outside the scope of this paper, but they are well-documented in the literature. [4][5][6]

9 Exponent arithmetic

RSA uses exponential notation in the ring $\mathcal{R}_n$. Exponential notation is simply a mathematical shorthand for writing a series of multiplications using the $\otimes$ operator. The multiplicative association property allows us to derive two rules for doing arithmetic with exponents.

Consider the set of three equations below. The left side of the first equation is the expression $m^2 \otimes m^3$. We can replace the m^2 with $(m \otimes m)$. We can also replace the m^3 with $(m \otimes m \otimes m)$. The right side of the first equation shows this. The multiplicative association property says that we can ignore the parentheses and simply count the number of m's that are multiplied. There are 5 and we show this in the second equation. Note that 5 is the sum of 2 plus 3. So instead of expanding the expression $m^2 \otimes m^3$ we can simply add 2 and 3, as shown in the third equation.

$$\begin{aligned} m^2 \otimes m^3 &= (m \otimes m) \otimes (m \otimes m \otimes m) \\ &= m^5 \\ &= m^{2+3} \end{aligned}$$

Exponent addition rule: In general, when we have an expression of the form $m^e \otimes m^d$ in the ring $\mathcal{R}_n$ we can simply add the exponents.

$$m^e \otimes m^d = m^{e+d}$$

Consider the set of four equations below. The left side of the first equation is the expression $(m^2)^3$. This means three copies of m^2 are multiplied using the $\otimes$ operator. The right side of the first equation shows this. In the second equation, we replace each m^2 with $(m \otimes m)$. The multiplicative association property says that we can ignore the parentheses and simply count the number of m's that are multiplied. There are 6 and we show this in the third equation. Note that 6 is the product of 2 times 3. Instead of expanding the expression $(m^2)^3$ we can simply multiply 2 and 3, as shown in the fourth equation.

$$\begin{aligned} (m^2)^3 &= m^2 \otimes m^2 \otimes m^2 \\ &= (m \otimes m) \otimes (m \otimes m) \otimes (m \otimes m) \\ &= m^6 \\ &= m^{2 \times 3} \end{aligned}$$

Exponent multiplication rule: In general, when we have an expression of the form $(m^e)^d$ in the ring $\mathcal{R}_n$ we can simply multiply the exponents.

$$(m^e)^d = m^{ed} \tag{3}$$

10 Multiple-plus-one

RSA uses two integers as exponents. One is the encryptor e and the other is the decryptor d. In order for RSA to work, the product ed must satisfy a strict condition. The condition is that the product ed must have a *multiple-plus-one* form. The product must be able to be written in the form $k\lambda + 1$. The reason for this condition will become apparent later. For now, however, we need to understand what the expression $k\lambda + 1$ means.

Table 3 contains some examples of multiple-plus-one products. Each product ends in 001. Each product is a multiple of 1000, plus one. In the first example, the product 174001 is equal to $174 \cdot 1000 + 1$.

$911 \cdot 191 = 174001$	$931 \cdot 971 = 904001$	$951 \cdot 551 = 524001$	$971 \cdot 931 = 904001$
$913 \cdot 977 = 892001$	$933 \cdot 597 = 557001$	$953 \cdot 617 = 588001$	$973 \cdot 37 = 36001$
$917 \cdot 253 = 232001$	$937 \cdot 873 = 818001$	$957 \cdot 93 = 89001$	$977 \cdot 913 = 892001$
$919 \cdot 679 = 624001$	$939 \cdot 459 = 431001$	$959 \cdot 439 = 421001$	$979 \cdot 619 = 606001$
$921 \cdot 481 = 443001$	$941 \cdot 661 = 622001$	$961 \cdot 641 = 616001$	$981 \cdot 421 = 413001$
$923 \cdot 987 = 911001$	$943 \cdot 807 = 761001$	$963 \cdot 27 = 26001$	$983 \cdot 647 = 636001$
$927 \cdot 863 = 800001$	$947 \cdot 283 = 268001$	$967 \cdot 303 = 293001$	$987 \cdot 923 = 911001$
$929 \cdot 169 = 157001$	$949 \cdot 549 = 521001$	$969 \cdot 129 = 125001$	$989 \cdot 909 = 899001$

Table 3: Multiples of 1000, plus one

The Greek letter λ (pronounced LAM duh) is specified in the RSA literature.[7] We use λ here as an integer constant. It typically has a huge value, almost as large as modulus n. In the context of Table 3 it has a small value, $\lambda = 1000$. The products can therefore be written like this:

$$\begin{aligned} 911 \cdot 191 &= 174\lambda + 1 \\ 913 \cdot 977 &= 892\lambda + 1 \\ 917 \cdot 253 &= 232\lambda + 1 \\ &\vdots \end{aligned}$$

The products in the table can be written in the form $k\lambda + 1$. The symbol k signifies some positive integer. Its value is not important. The term $k\lambda$ simply means *some integer multiple of* λ. This meaning of k allows the multiple-plus-one condition to be stated succinctly.

Multiple-plus-one condition: Given positive integers e, d, and λ, the product ed shall be an integer multiple of λ, plus one.

$$ed = k\lambda + 1 \tag{4}$$

11 Exponential tables

We now take a closer look at exponential products m^a in the ring $\mathcal{R}_n$. When n is very small we can compute exponential products by hand. As an example we compute 7^3 in the ring $\mathcal{R}_{15}$ using Table 2.

$$7^3 \quad = \quad 7 \otimes 7 \otimes 7 \quad = \quad (7 \otimes 7) \otimes 7 \quad = \quad 4 \otimes 7 \quad = \quad 13$$

We can go on to calculate the exponential product of each pair of elements in Z_{15} and put them all in a table. Table 4 specifies the exponential products m^a in the ring $\mathcal{R}_{15}$. The product of 7^3 is at the intersection of row 7 and column 3.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	1	2	4	8	1	2	4	8	1	2	4
3	1	3	9	12	6	3	9	12	6	3	9	12	6	3	9
4	1	4	1	4	1	4	1	4	1	4	1	4	1	4	1
5	1	5	10	5	10	5	10	5	10	5	10	5	10	5	10
6	1	6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	4	13	1	7	4	13	1	7	4	13	1	7	4
8	1	8	4	2	1	8	4	2	1	8	4	2	1	8	4
9	1	9	6	9	6	9	6	9	6	9	6	9	6	9	6
10	1	10	10	10	10	10	10	10	10	10	10	10	10	10	10
11	1	11	1	11	1	11	1	11	1	11	1	11	1	11	1
12	1	12	9	3	6	12	9	3	6	12	9	3	6	12	9
13	1	13	4	7	1	13	4	7	1	13	4	7	1	13	4
14	1	14	1	14	1	14	1	14	1	14	1	14	1	14	1

Table 4: $m^a \quad (\mathcal{R}_{15})$

Now consider the product sequence in row 3 (shown below). Notice how the sequence starts at 1 and then repeats itself. The shortest repetitive part of a sequence is called a *cycle*. The cycle in row 3 is (3, 9, 12, 6). The *period* of this cycle is 4.

$$1 \quad \overbrace{3 \quad 9 \quad 12 \quad 6} \quad \overbrace{3 \quad 9 \quad 12 \quad 6} \quad \cdots$$

Each row in Table 4 is a sequence that starts with 1 followed by a series of cycles. Each cycle in the various rows has a period of either 1 or 2 or 4. Remarkably, all of the cycles line up vertically in such a way as to provide what may be called *identity columns*. Consider columns 1, 5, 9, and 13. These are the identity columns in the table. Each is identical to the row number column on the left side of the table. So for any $m \in Z_{15}$ we have

$$m^1 = m^5 = m^9 = m^{13}$$

12 Wallpaper

We now consider a larger exponentiation table. Table 5 specifies exponential products m^a in the ring $\mathcal{R}_{33}$. The table is small enough to fit on a page yet big enough for us to visually perceive a *wallpaper* pattern. There appear to be three identical strips of wallpaper side by side. Each strip is 10 columns wide and runs from top to bottom.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	16	32	31	29	25	17	1	2	4	8	16	32	31	29	25	17	1	2	4	8	16	32	31	29	25	17	1	2	4
3	1	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9	27	15	12	3	9
4	1	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16	31	25	1	4	16
5	1	5	25	26	31	23	16	14	4	20	1	5	25	26	31	23	16	14	4	20	1	5	25	26	31	23	16	14	4	20	1	5	25
6	1	6	3	18	9	21	27	30	15	24	12	6	3	18	9	21	27	30	15	24	12	6	3	18	9	21	27	30	15	24	12	6	3
7	1	7	16	13	25	10	4	28	31	19	1	7	16	13	25	10	4	28	31	19	1	7	16	13	25	10	4	28	31	19	1	7	16
8	1	8	31	17	4	32	25	2	16	29	1	8	31	17	4	32	25	2	16	29	1	8	31	17	4	32	25	2	16	29	1	8	31
9	1	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15	3	27	12	9	15
10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1	10	1
11	1	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22	11	22
12	1	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
13	1	13	4	19	16	10	31	7	25	28	1	13	4	19	16	10	31	7	25	28	1	13	4	19	16	10	31	7	25	28	1	13	4
14	1	14	31	5	4	23	25	20	16	26	1	14	31	5	4	23	25	20	16	26	1	14	31	5	4	23	25	20	16	26	1	14	31
15	1	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27	9	3	12	15	27
16	1	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25	4	31	1	16	25
17	1	17	25	29	31	32	16	8	4	2	1	17	25	29	31	32	16	8	4	2	1	17	25	29	31	32	16	8	4	2	1	17	25
18	1	18	27	24	3	21	15	6	9	30	12	18	27	24	3	21	15	6	9	30	12	18	27	24	3	21	15	6	9	30	12	18	27
19	1	19	31	28	4	10	25	13	16	7	1	19	31	28	4	10	25	13	16	7	1	19	31	28	4	10	25	13	16	7	1	19	31
20	1	20	4	14	16	23	31	26	25	5	1	20	4	14	16	23	31	26	25	5	1	20	4	14	16	23	31	26	25	5	1	20	4
21	1	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12	21	12
22	1	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22
23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1	23	1
24	1	24	15	30	27	21	9	18	3	6	12	24	15	30	27	21	9	18	3	6	12	24	15	30	27	21	9	18	3	6	12	24	15
25	1	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31	16	4	1	25	31
26	1	26	16	20	25	23	4	5	31	14	1	26	16	20	25	23	4	5	31	14	1	26	16	20	25	23	4	5	31	14	1	26	16
27	1	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3	15	9	12	27	3
28	1	28	25	7	31	10	16	19	4	13	1	28	25	7	31	10	16	19	4	13	1	28	25	7	31	10	16	19	4	13	1	28	25
29	1	29	16	2	25	32	4	17	31	8	1	29	16	2	25	32	4	17	31	8	1	29	16	2	25	32	4	17	31	8	1	29	16
30	1	30	9	6	15	21	3	24	27	18	12	30	9	6	15	21	3	24	27	18	12	30	9	6	15	21	3	24	27	18	12	30	9
31	1	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4	25	16	1	31	4
32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1	32	1

Table 5: $m^a \quad (\mathcal{R}_{33})$

Notice that this table also contains identity columns. They are columns 1, 11, 21, and 31. Also notice that column 2 is the same as columns 12 and 22, column 3 is the same as columns 13 and 23, and so on. In fact, the entire block of columns 1 through 10 is repeated in columns 11 through 20, and this block pattern continues to repeat for columns beyond 20.

We can easily represent this wallpaper pattern with an equation. Any column a is the same as column $10+a$ and column $20+a$ and so on. We use the notation $k \cdot 10$ to denote some multiple of 10. So for any $m \in Z_{33}$ and any integer $a > 0$ we have

$$m^a = m^{k \cdot 10 + a}$$

Each row in Table 5 is a sequence of exponential products. Each sequence is a 1 followed by a series of cycles. These cycles have various periods. For this table the periods are 1, 2, 5, and 10. The period of the longest cycle is symbolized by λ. This is also known as the *Carmichael function value*. For this particular exponential table we have $\lambda = 10$. However, for any two distinct primes p and q, it turns out that the Carmichael function value λ is the *least common multiple* of $p-1$ and $q-1$.

$$\lambda = \text{lcm}(p-1, q-1)$$

When p and q are small we can compute the Carmichael function value by hand. For example, let $p = 11$ and $q = 13$ so that $\lambda = \text{lcm}(10, 12)$. The multiples of 10 are 10, 20, 30, etc. The multiples of 12 are 12, 24, 36, etc. The multiples that are common to both are 60, 120, 180, etc. The least of these is 60.

Multiples of 10	=	10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, ...
Multiples of 12	=	12, 24, 36, 48, 60, 72, 84, 96, 108, 120, 132, ...
Common to both	=	60, 120, 180, ...
lcm(10, 12)	=	60

We described the wallpaper pattern of Table 5 using the equation $m^a = m^{k \cdot 10 + a}$. We can replace the 10 with λ. This gives us $m^a = m^{k\lambda + a}$. This equation holds for primes $p = 3$ and $q = 11$. But does it hold for *any* pair of primes? We assert that it does and we offer the following theorem without proof.

Wallpaper theorem: Given two distinct primes p and q, the ring $\mathcal{R}_{pq}$, the Carmichael function value $\lambda = \text{lcm}(p-1, q-1)$, any $m \in Z_{pq}$, any integer $a > 0$, and any integer $k \geq 0$, then

$$m^a = m^{k\lambda + a}$$

RSA uses a special case of the Wallpaper theorem where $a = 1$. We call this special case the *Carmichael identity*. The m^a on the left side is replaced by m, since $m^1 = m$. The $m^{k\lambda + a}$ on the right side is replaced by $m^{k\lambda + 1}$.

Carmichael identity: Given two distinct primes p and q, the ring $\mathcal{R}_{pq}$, the Carmichael function value $\lambda = \text{lcm}(p-1, q-1)$, any $m \in Z_{pq}$, and any integer $k \geq 0$, then

$$m = m^{k\lambda + 1} \tag{5}$$

13 Mappings

RSA encryption maps an integer m to an integer m^e. Likewise, RSA decryption maps an integer c to an integer c^d. This is demonstrated in Table 6, where $e = 3$ and $d = 7$. In the encryption procedure, every element in the m column maps to a unique element in the m^3 column. And in the decryption procedure, every element in the c column maps to a unique element in the c^7 column.

m	m^3	c	c^7	y
0	0	0	0	0
1	1	1	1	1
2	8	2	29	2
3	27	3	9	3
4	31	4	16	4
5	26	5	14	5
6	18	6	30	6
7	13	7	28	7
8	17	8	2	8
9	3	9	15	9
10	10	10	10	10
11	11	11	11	11
12	12	12	12	12
13	19	13	7	13
14	5	14	20	14
15	9	15	27	15
16	4	16	25	16
17	29	17	8	17
18	24	18	6	18
19	28	19	13	19
20	14	20	26	20
21	21	21	21	21
22	22	22	22	22
23	23	23	23	23
24	30	24	18	24
25	16	25	31	25
26	20	26	5	26
27	15	27	3	27
28	7	28	19	28
29	2	29	17	29
30	6	30	24	30
31	25	31	4	31
32	32	32	32	32

Table 6: $m \to c \to y \quad (\mathcal{R}_{33})$

The table also illustrates an example. First $m = 13$ is mapped to $m^3 = 19$ which then becomes c. Subsequently $c = 19$ is mapped to $c^7 = 13$ which then becomes y. The final result, $y = 13$, is identical to the original, $m = 13$. This is how RSA works, no matter which m one starts with, no matter how large n is. The final y will always be identical to the original m.

Some of the values in column m are the same as their corresponding value in m^3. For example, $10 \rightarrow 10$. There are 9 of these, out of a total of 33. That means that 28 percent of the ciphertexts are identical to their plaintext messages. This would be unacceptable, of course, except that the percentage is infinitesimal for huge values of n.

Columns m^3 and c^7 each contain every element in Z_{33}. This is a requirement in order to make the two mappings work, so that $y = m$, Every element in m needs to map to a unique element in m^3, and every element in c must map to a unique element in c^7. If this were not the case, if some element in Z_{33} was not in column m^3, or if some element in Z_{33} was in column m^3 more than once, then RSA would not work for every m in Z_{33}.

Refer again to Table 5 in the Wallpaper section. Many of the columns do not contain every element in Z_{33}. These are columns 0, 2, 4, 5, 6, 8, 10, 12, 14, 15, 16, 18, 20, 22, 24, 25, 26, 28, 30, and 32. That's 20 out of 33 columns, or 61 percent. In each of these columns, some elements of Z_{33} are missing and some appear more than once. For example, the element 1 appears more that once in each of these columns.

It turns out that for each of these columns (except column 0), the column number shares a prime factor with $\lambda = 10$. The prime factors of 10 are 2 and 5. So if a column number is a multiple of 2 or a multiple of 5, the column itself won't contain every element in Z_{33}. Notice that neither 3 nor 7 shares a prime factor with λ.

We can use the *greatest common divisor* function (gcd) to determine if two integers share a prime factor. When the two integers are small we can compute the greatest common divisor by hand. As an example we compute the gcd of 6468 and 7560 by hand. First we list the prime factors of 6468 and the prime factors of 7560. Then we list the prime factors that are common to both. Finally, we compute the product of these common factors. This gives us the gcd.

$$\begin{array}{lcl} \text{Factors of } 6468 & = & 2 \cdot 2 \cdot 3 \cdot 7 \cdot 7 \cdot 11 \\ \text{Factors of } 7560 & = & 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 3 \cdot 5 \cdot 7 \\ \text{Common to both} & = & 2 \cdot 2 \cdot 3 \cdot 7 \\ \gcd(6468, 7560) & = & 84 \end{array}$$

Most implementations of the gcd function return 1 if no prime factors are shared. Therefore, if the gcd function returns anything greater than 1, we are assured that the two numbers share one or more prime factors. RSA uses the gcd function during key generation in order to select an encryptor e that shares no prime factors with λ.

14 A simple proof

We need to convince ourselves that RSA works under a broad set of conditions. That is, we need to demonstrate that we can start with any $m \in Z_n$, perform two modex operations on it, and get m back. Here's the set of conditions:

- two prime integers p and q such that $p \neq q$
- the ring $\mathcal{R}_n = (Z_n, \oplus, \otimes)$ where $n = pq$
- exponential notation in $\mathcal{R}_n$ (e.g. $m^3 = m \otimes m \otimes m$)
- the Carmichael function value $\lambda = \text{lcm}(p-1, q-1)$
- two integers e and d such that $ed = k\lambda + 1$
- an integer m such that $m \in Z_n$

Refer to the RSA cryptosystem of Figure 1. The message m is presented at the input of Bob's transmitter. The message y is produced at the output of Alice's receiver. We will demonstrate that $y = m$.

The receiver output equation (1) states that $y = c^d$. This is what we begin our proof with. In the following steps, we will modify the right side of this equation.

$$y = c^d$$

The transmitter output equation (2) states that $c = m^e$. We replace c in the equation above with (m^e).

$$y = (m^e)^d$$

The exponent multiplication rule (3) states that $(m^e)^d = m^{ed}$. We replace $(m^e)^d$ in the equation above with m^{ed}.

$$y = m^{ed}$$

The multiple-plus-one condition (4) states that $ed = k\lambda + 1$. We replace ed in the equation above with $k\lambda + 1$.

$$y = m^{k\lambda+1}$$

The Carmichael identity (5) states that $m = m^{k\lambda+1}$. We replace $m^{k\lambda+1}$ in the equation above with m.

$$y = m$$

QED

15 Hard problems

The security of RSA is based on several hard problems. The most prominent of these is *integer factorization*. The problem is to write an algorithm that computes the prime factors of some huge integer n and does it using a small number of computing operations. An algorithm is "fast" if it requires only a few operations to complete the solution. It is a hard problem to write an algorithm that is fast enough to factor a 1024-bit RSA modulus within any reasonable amount of time.

Four factoring algorithms are graphed in Figure 2. They are, from slowest to fastest: Trial Division (TD), the Quadratic Sieve (QS), the Number Field Sieve (NFS), and Peter Shor's algorithm for quantum computers.[8] The graph plots the number of operations required to factor some modulus n. For example, it will take roughly 10^{12} operations to factor a 768-bit modulus using the NFS algorithm. This is about 1500 years on a single core 2.2 GHz AMD Opteron processor with 2 GB RAM.[9]

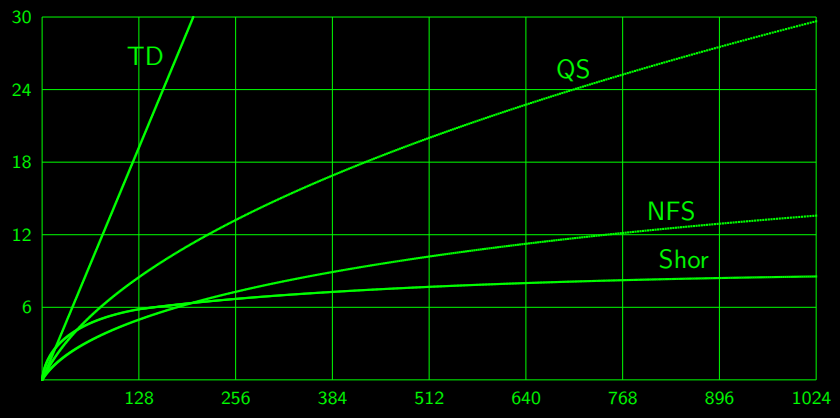

Figure 2: $\log_{10}$ operations per $\log_2 n$

An RSA cryptosystem can be broken if the modulus can be factored. That is, if Eve can factor n into p and q, she can easily compute d. She first computes the Carmichael function value $\lambda = \text{lcm}(p-1, q-1)$. Then she computes d such that $ed = k\lambda + 1$. Trouble is, factoring a huge integer takes a *very* long time.

Eve can try to solve the RSA problem[10] and compute the e^{th} root of m^e, i.e., compute $m = \sqrt[e]{m^e}$. But computing roots takes just as long as factoring. There are other algorithms that can theoretically break RSA but they're all just as slow as integer factorization. The point is that Eve will not be able break an RSA cryptosystem with a huge modulus in any reasonable amount of time.

16 Conclusions

So why does RSA work? Why is it that we can take some message m, run it through two modex operations, and come out with the same m? There are several reasons. First of all, RSA computations are done in a commutative ring and the multiplicative association property holds in commutative rings. This property tells us that the two exponentiations $(m^e)^d$ are the same as the one exponentiation m^{ed}.

A second reason is that exponents e and d are chosen such that they satisfy the multiples-plus-one condition $ed = k\lambda + 1$. This insures that ed is one of the identity columns in the exponential table of ring $\mathcal{R}_n$.

A third reason is that the exponential table contains repeating blocks of columns where $m^a = m^{k\lambda+a}$. This is the wallpaper pattern that we saw in Table 5. This pattern is the reason for the multiples-plus-one condition.

Finally, RSA works because it relies on the intractability of the factoring problem. A huge RSA modulus n cannot be factored expeditiously. Given that n is the product of two distinct huge random primes, it is virtually impossible to factor n in any reasonable amount of time, even if the factoring effort is distributed across thousands of computers.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120-126.

[2] Andreas de Vries. The ray attack, an inefficient trial to break RSA cryptosystems. FH Südwestfalen University of Applied Sciences, Haldener StraBe 182, D-58095 Hagen, 2003.

[3] Ring (mathematics). *Wikipedia*, `www.wikipedia.com`.

[4] Çetin Kaya Koç. High-speed RSA implementation. RSA Labs, 1994.

[5] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1994.

[6] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[7] PKCS #1 v2.1: RSA cryptography standard. RSA Labs, 2002.

[8] Peter W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," 1996.

[9] Thorsten Kleinjung et al. Factorization of a 768-bit RSA modulus. Version 1.4, February 18, 2010.

[10] Ronald L. Rivest and Burt Kaliski. RSA problem. RSA Labs, 2003.