https://doi.org/10.55843/ivisum23810161

RSA & EXTENDED EUCLIDEAN ALGORITHM WITH EXAMPLES OF EXPONENTIAL RSA CIPHERS, RSA EXAMPLE SOLUTION WITH EXTENDED EUCLIDEAN ALGORITHM

Ergin Diko, Mushab Ibraimi, page 161-175

ABSTRACT

In inter-system connections or between any two points in communication, it is necessary to make sure that the data goes securely. This is achieved by encrypting the sent data. As the disciplines of cryptography and network security matured, more practical applications were developed that were not readily available to ensure network security. Today, encryption has become a necessity in the digital environment. In this study, basic cryptography terms are mentioned. The RSA algorithm (Rivest-Shamir-Adleman) is the basis of a cryptographic system, a suite of cryptographic algorithms used for private security services or purposes, and this allows public key encryption, widely used to secure particularly sensitive data sent over an insecure network such as the internet. Commonly used methods were examined and RSA encryption method was chosen in accordance with the purpose of the study. RSA, a public key encryption technique, is built on the difficulty of generating and processing very large integers. In this study, operations with large numbers that take a long time are completed in a short time using various methods. It has been reflected in the study by creating a more secure structure by using large prime numbers for the key generation process.

Keywords: Cryptography, RSA, Extended Euclidean Theorem, Modular Exponentiation

Mr. Ergin Diko

Computer Engineering, International Vision University

e-mail: ergin.diko@vizyon.edu.mk

Mr. Mushab Ibraimi

Computer Engineering, International Vision University

e-mail: mushab@vizyon.edu.mk

UDK: 004.7.056

Date of received: 15.01.2023

Date of acceptance: 25.02.2023

Declaration of interest: The authors reported no conflict of interest related to this article.

INTRODUCTION

Today, with the development of technology, computers and internet started to take a big place in our lives. As more and more people are online every day, it has become inevitable to transact over the internet. In addition, security should be given priority so that important functions such as banking transactions, commercial relations, government affairs, military affairs, private meetings and similar can be carried out smoothly. It is necessary to make sure that the data goes safely in inter-system connections or communication between any two points. Encryption and identity are the way to ensure security (Saracevic , Selimi, & Selimovic, Generation of cryptographic keys with algorithm of polygon triangulation and Catalan numbers, 2018).

This is achieved by encrypting the sent data. Thus, it is ensured that data is transmitted securely by using open communication channels (Saračević, Selimi, & Plojović, 2019). If an open communication channel is used in communication, the thought that the information that is intended to be kept confidential can be listened to by an unauthorized person or that it may enter the communication channel (interference) and corrupt or change the data (sending the wrong data), always creates an important problem. Cryptology is mainly divided into two parts, cryptography (encryption) and cryptanalysis (decryption). The original message to be sent is called plain text and the encrypted version of this message is called the cipher text-cryptograph. Encryption has been used for millennia to ensure security in military and diplomatic communications (Saracevic, Selimi, & Pepić, 2022). However, today it is also needed in the private sector. Communication between computers on subjects such as health services and financial affairs is carried out using open channels. During the use of these open channels, encryption is required in order to perform the above-mentioned works in a secure and confidential manner. Encryption has become a necessity in the digital environment today (Saračević, Selimi, & Mujevic, 2018). In this study, basic cryptography terms were mentioned, commonly used methods were examined and RSA encryption method was chosen in accordance with the purpose of the study. RSA is an encryption method created by Rivest, Shamir, and Adleman for the secure storage and transfer of data in digital media.

RSA ENCRYPTION ALGORITHM

In 1978, Ron Rivest, Adi Shamir, and Leonard Adleman introduced a cryptographic algorithm that would essentially replace the less secure National Bureau of Standards (NBS) algorithm. Most importantly, RSA implements a public key encryption system alongside digital signatures. RSA was motivated by the work of Diffie and Hellman published several years ago, which described the idea of such an algorithm but never really developed it.

Introduced at a time when the age of electronic e-mail was expected to emerge soon, RSA implemented two important ideas:

1. Public key encryption. This idea eliminates the need for a "courier" to deliver the keys to the recipients via another secure channel before delivering the originally intended message. In RSA, encryption keys are public, while decryption keys are not, so only the person with the correct decryption key can decrypt an encrypted message. Each has its own encryption and decryption keys. The keys must be made in such a way that the decryption key cannot be easily extracted from the public encryption key.

2. Digital signatures. The receiver may need to verify that the transmitted message actually originated from the sender (signature) and did not just come from there (authentication). This is done using the sender's decryption key, and the signature can later be verified by anyone using the corresponding public encryption key. Therefore, signatures cannot be forged. Also, no signer can later deny that he signed the message.

This is not only useful for email, but also for other electronic transactions and transmissions, such as fund transfers. The security of the RSA algorithm has so far been proven, as no attempt to break it has yet been successful, mostly due to the difficulty of factoring large numbers $n = pq$, where p and q are large prime numbers. RSA is a public-key encryption technique that uses very large integers. It has been thought about the difficulty of creating and processing these numbers. A more secure structure has been created by using prime numbers for key generation. The key generation algorithm is as follows:

- Two very large prime numbers such as P and Q are chosen.
- The product of these two prime numbers $N = P \cdot Q$ and their one missing $\varphi(N) = (P-)(Q-1)$ is calculated.
- A prime integer E with $\varphi(N)$ greater than 1 and less than $\varphi(N)$ is selected.
- The selected integer E is inverted in mod $\varphi(N)$, the result is is an integer something like D.
- Integers E and N make up the public key, and integers D and N make up the private key.

After creating the public and private keys, the information to be sent is encrypted with the public key. Encryption is done as follows:

The digital equivalent of the information to be encrypted is taken to the E power, and its equivalent in mode N forms the encrypted text. A text encrypted with a public key can only be opened with a private key. Therefore, the cipher text, in the same way, is taken to the D power of the cipher text's numerical equivalent, and its mod N equivalent forms the original text.

The reason for generating a key using the product of two prime numbers in this algorithm is because it is more difficult to factor the product of two prime numbers into prime factors than to separate non-prime numbers.

The most time-consuming processes when formula is processed are superposition and mode finding. A small or easy-to-calculate value of E can be chosen to speed up the process. As we mentioned above, the smallness and repetitive use of the value reduces the security.

But there are also methods to prevent this situation. In this study

Encryption is done by using extended Euclidean algorithm and modular exponentiation methods, by processing large numbers in a short time.

2. Public-key cryptosystems

Each user has their own encryption and decryption procedures, E and D, the first is in public file and the second is kept private. These procedures deal specifically with keys, which are two sets of special numbers in RSA. Of course, we start with the message itself, symbolized

by the "to be encrypted" M. There are four procedures that are specific and necessary to a public-key cryptosystem:

a) Decrypting an encrypted message gives you the original message, especially

$$D(E(M) = M. \quad (1)$$

b) Reversing the procedures still returns M:

$$E(D(M)) = M . \quad (2)$$

c) E and D are easy to calculate.

d) The introduction of E does not jeopardize D's privacy, so you cannot easily tell D from E.

With a given E, we are still not given an efficient way of computing D. If $C = E(M)$ is the ciphertext, then trying to figure out D by trying to satisfy an M in $E(M) = C$ is unreasonably difficult: the number

of messages to test would be impractically large.

An E that satisfies (a), (c), and (d) is called a “trap-door one-way function” and is also a “trap-door one-way permutation”. It is a trap door because since it’s inverse D is easy to compute if certain “trapdoor” information is available, but otherwise hard. It is one-way because it is easy to compute in one direction, but hard in the other. It is a permutation because it satisfies (b), meaning every cipher text is a potential message, and every message is a cipher text of some other message. Statement (b) is in fact just needed to provide “signatures”.

Now we turn to specific keys, and imagine users A and B (Alice and Bob) on a two-user public-key cryptosystem, with their keys: EA, EB, DA, DB.

With public key cryptography, all parties interested in secure communications publish their public keys (Merkle, p. 2000). As to how that is done depends on the protocol. In the SSH protocol, each server makes available through its port 22 the public key it has stored for your login id on the server. When a client, such as your laptop, wants to make a connection with an SSHD server, it sends a connection request to port 22 of the server machine and the server makes its host key available automatically. On the other hand, in the SSL/TLS protocol, an HTTPS web server makes its public key available through a certificate of the sort you'll see in the next lecture. As we will see, this solves one of the most vexing problems associated with symmetric-key cryptography — the problem of key distribution.

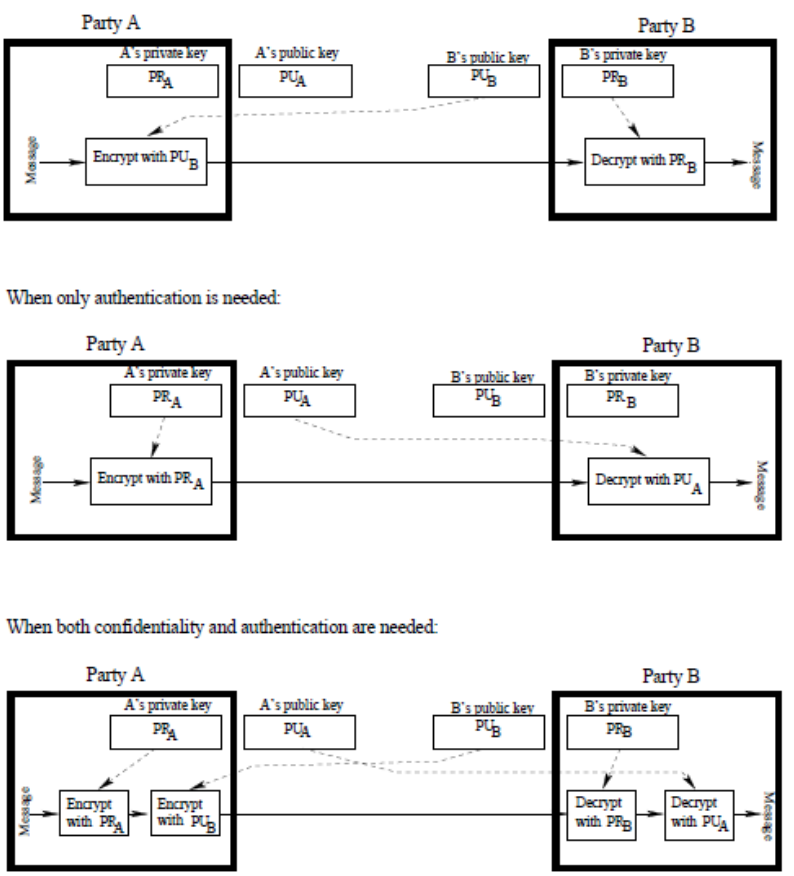

Figure 1: This figure shows how public-key cryptography can be used for confidentiality, for digital signatures, and

for both. (This figure is from Lecture 12 of "Computer and Network Security" by Avi Kak.)

Party *A*, if wanting to communicate confidentially with party *B*, can encrypt a message using *B*'s publicly available key. Such a communication would only be decipherable by *B* as only *B* would have access to the corresponding private key. This is illustrated by the top communication link in Figure 1. Party *A*, if wanting to send an authenticated message to party *B*, would encrypt the message with *A*'s own private key. Since this message would only be decipherable with *A*'s public key, that would establish the authenticity of the message — meaning that *A* was indeed the source of the message. This is illustrated by the middle communication link in Figure 1.

The communication link at the bottom of Figure 1 shows how public-key encryption can be used to provide both confidentiality and authentication at the same time. Note again that confidentiality means that we want to protect a message from eavesdroppers and authentication means that the recipient needs a guarantee as to the identity of the sender.

In Figure 1, *A*'s public and private keys are designated PU_{A} and PR_{A}. *B*'s public and private keys are designated PU_{B} and PR_{B}. As shown at the bottom of Figure 1, let's say that *A* wants to send a message *M* to *B* with both authentication and confidentiality.

The processing steps undertaken by A to convert M into its encrypted form C that can be placed on the wire are:

$$C = E\,(PU_{\mathrm{B}}, E\,(PR_{\mathrm{A}}, M)) \tag{3}$$

where E() stands for encryption. The processing steps under- taken by B to recover M from C are

$$M = D(PU_{\mathrm{A}}, D(PR_{\mathrm{B}}, C)) \tag{4}$$

where D() stands for decryption.

The sender *A* encrypting his/her message with its own private key PR_{A} provides authentication. This step constitutes *A* putting his/her digital signature on the message.

Instead of applying the private key to the entire message, a sender may also "sign" a message by applying his/her private key to just a small block of data that is derived from the message to be sent.

The sender *A* further encrypting his/her message with the receiver's public key PU_B provides confidentiality.

Of course, the price paid for achieving confidentiality and authentication at the same time is that now the message must be processed four times in all for encryption/decryption. The message goes through two encryptions at the sender's place and two decryptions at the receiver's place. Each of these four steps involves separately the computationally complex public-key algorithm.

The math of the method

So far, we expect to make *E* and *D* easy to compute through simple arithmetic. We must now represent the message numerically, so that we can perform these arithmetic algorithms on it. Now let's represent *M* by an integer between 0 and $n - 1$. If the message is too long, sparse it up and encrypt separately. Let *e, d, n* be positive integers, with *(e, n)* as the encryption key, *(d, n)* the decryption key, $n = pq$ (Niven, New York, 1972.).

Now, we encrypt the message by raising it to the *e*th power modulo *n* to obtain *C*, the cipher text. We then decrypt *C* by raising it to the *d*th power modulo n to obtain M again. Formally, we obtain these encryption and decryption algorithms for *E* and *D*:

$$C \equiv E(M) \equiv M^{e} (mod\ n) \tag{5}$$

$$M \equiv D(C) \equiv C^{d} (mod\ n)\,.$$

Note that we are preserving the same information size, since M and C are integers between 0 and n − 1, and because of the modular

congruence. Also note the simplicity of the fact that the encryption/decryption keys are both just pairs of integers, (e, n) and (d, n). These are different for every user, and should generally be subscripted, but we'll consider just the general case here.

Now comes the question of creating the encryption key itself. First, choosing two "random" large primes p and q, we multiply and produce $n = pq$. Although n is public, it will not reveal p and q since it

is essentially impossible to factor them form n, and therefore will assure that d is practically impossible to derive from e.

Now we want to obtain the appropriate e and d. We pick d to be a random large integer, which must be coprime to $(p - 1) \cdot (q - 1)$, meaning the following equation has to be satisfied:

$$gcd(d, (p - 1) \cdot (q - 1)) = 1 . \quad (6)$$

"gcd" means greatest common divisor.

The reason we want d to be coprime to $(p - 1) \cdot (q - 1)$ is peculiar. I will not show the "direct motivation" behind it; rather, it will become clear why that statement is important when l show towards the end of this section that it guarantees (1) and (2).

We will want to compute e from $d, p, and\ q$, where e is the multiplicative inverse of d. That meanswe need to satisfy

$$e \cdot d = 1\ (mod\ \varphi(n)) \quad (7)$$

Here, we introduce the Euler totient function $\varphi(n)$, whose output is the number of positive integersless than n which are coprime to n. For primes p, this clearly becomes

$$\varphi(p) = p - 1 .$$

For n, we obtain, by elementary properties of the totient function, that

$$\varphi(n) = \varphi(p) \cdot \varphi(q)$$

$$= (p - 1) \cdot (q - 1) \tag{8}$$

$= n - (p + q) + 1$.

From this equation, we can substitute $\varphi(n)$ into equation (7) and obtain

$$e \cdot d \equiv 1 \, (mod \, \varphi(n))$$

which is equivalent to

$$e \cdot d = k \cdot \varphi(n) + 1$$

for some integer k.

y the laws of modular arithmetic, the multiplicative inverse of a modulo m exists if and only if a and m are coprime. Indeed, since d and φ(n) are coprime, d has a multiplicative inverse e in the ring of integers modulo φ(n).

So far, we can safely assured the following:

$$D(E(M)) \equiv (E(M))^{d} \equiv (M^{e})^{d} \, (mod \, n) = M^{e \cdot d} (mod \, n)$$

$$E(D(M)) \equiv (D(M))^{e} \equiv (M^{d})^{e} \, (mod \, n) = M^{e \cdot d} (mod \, n)$$

Also, since $e \cdot d = k \cdot \varphi(n) + 1$, we can substitute into the above equations and obtain

$$M^{e \cdot d} \equiv M^{k \cdot \varphi(n)+1} \quad (mod \, n) .$$

Clearly, we want that to equal M. To prove this, will need an important identity due to Euler and Fermat: for any integer M coprime to n, we have

$$M^{\varphi(n)} \equiv 1 \, (mod \, n) . \tag{9}$$

Since we previously specified that $0 \leq M < n$, we know that M would not be coprime to n if and only if M was either $p \; or \; q$, of the integers in

that interval. Therefore, the chances of M happening to be $p\ or\ q$ are on the same order of magnitude as $2/n$. This means that M is almost definitely relatively prime to n, therefore equation (9) holds and, using it, we evaluate:

$$Me \cdot d \equiv M^{k\cdot\varphi(n)+1} \equiv (M^{\varphi(n)})^{k} M \equiv 1^{k} M\ (mod\ n) = M.$$

It turns out this works for all M, and in fact we see that (1) and (2) hold for all

$$M, 0 \leq M < n.$$

Therefore E and D are inverse permutations.

4. RSA Examples Solved with Extended Euclidean Algorithm

Solution Example 1

RSA Algorithm:

1. Choose 2 prime numbers, e.g., **p = 11** and **q = 17**

2. Compute **n** such that n = p * q = 11 * 17 **= 187**

3. Compute **z** such that z = (p-1)(q-1) = 10 * 16 **= 160**

4. Choose **e** relatively prime to **160.**

Since 160= 5 * 5 * 5, it means that e cannot be 5. Pick any other prime.

Let e = 19

5. Compute **d** such that

d * e = 1 (mod 160), i.e., **$d = e^{-1}$ (mod 160)**

We read this as: d*e is congruent to 1, in modulo 160 math. That means that reminder of d*e/160 is the same as the reminder of 1/160, i.e. 1.

In other words, d*e mod 160 = 1 mod 160.

This also can be read as "d is the multiplicative inverse of e = 19 mod 160." This also be written as d = e-1 mod 160.

(FYI: Modulo 160 math, roughly speaking, means that we take numbers in chunks of 160 numbers, i.e. numbers "repeat" after 160th.

For example: In modulo 10 math,

1 is "the same" as 11, 21, 31, 41, ... i.e. 1 and 11, etc. are congruent;

also, 2 is congruent to 12,22, 32, .

3 is congruent to 13, 23, 33, ... etc.).

So, we need to find d such that d * e = 1 (mod 160).

Brute force method: You could guess what d should be - if you are super good with large numbers, try to find d such that reminder of d*e/160 is 1. i.e. d*e mod 160=1

So try all multiples of 19 until you find one that that works.

d=1: 19 mod 160=19 is not 1, so d=1 doesn't work

d=2: 38 mod 160 = 38

d=3: 57 mod 160 = 57

so it's not worth checking d's less than 8.

d=9: 178 mod 160 = 18 so d=9 won't work

d=10 190 mod 160 = 30

d=11

.....

keep on going until you find d=59. Good luck! d can have a large value.

There is another, shorter, brute force methods:

d = (z+1)/e but only if the result is whole. So, try to find multiples of z that give a whole number when added 1 and divided by e. It is shorter than the above brute force method.

d= (1*160+1)/19 is not whole

(2*160 +1)/19 is not whole

(3*160 + 1)/19

finally 7*z gives result d=59.

ExcelExampleBruteForce

e	19	
z	160	
d=(z+1)/e	1	8.47
	2	16.89
	3	25.32
	4	33.74
	5	42.16
	6	50.58
	7	59

It might be easier to use the Extended Euclidean Algorithm to compute value for d. See solution details below. It turns out that:

d = 59

Check: 19*59mod 160 = 1121 mod 160 = 1

1 mod 160 = 1

So yes, d*e = 1 (mod 160).

Therefore:

Public key = (e, n) = (19, 187)

Private key = (d,n) = (59, 187)

Encrypt the message

After we calculated the keys, the only thing left to do is to encrypt the message. Convert each character of the message into ASCII equivalent, and split the message into blocks of digits. Take the numerical value of each block, let's call its value PB, and encrypt each plain block PB using formula:

encrypted(PB) = PB^e mod n.

Decrypt cyphered block CB using formula:

decrypted(CB) = CB^d mod n.

Let's assume that we are trying to encrypt message $\alpha\beta\gamma\delta$ which has ASCII representation of 001002. Let's say we break the message into blocks of 3.

So the blocks are 001, 002.

So, the cyphertext is 001 127:

00119 mod 187 = reminder of 119 / 187 = 1

00219 mod 187 = 127

Now let's decrypt 001 127.

Again, we take the blocks of 3:

001^{59} mod 187 = 001

127^{59} mod 187 = 002

EUCLIDIAN ALGORITHM IN DETAIL:

Extended Euclidian Algorithm fits numbers into this Pattern:

Extended Euclidean Algorithm Example for e = 19, z = 160

Start from the top, from z and e:

160/19 = 8 R=8 8 = 160 * 1 - 19 * 8 (step 1)

160 is the square, 19 is the triangle, 8 is circle, 8 is star

Now fill in the same "starry" pattern, but this time plug in different values:

put the current value of the triangle(i.e. 19) into the square; and put the current value of the star (i.e. 8) into the place of triangle.

19/8 = 2 R=3 3 = 19 - 8 * 2 (step 2)

19 is the square, 8 is the triangle,2 is circle, 3 is star

Keep on doing this same thing until you reach reminder=1:

8/3 = 2 R=2 2 = 8 - 3 * 2 (step 3)

3/2 = 1 R=1 1 = 3 - 2 * 1 (step 4)

Now try to make all remainders look like a combo of 160 and 19, i.e. try to make them look like:

8 = 160*__ + 19* __

3 = 160*__ + 19* __

2 = 160*__ + 19* __

1 = 160*__ + 19* __

You could guess what values to put in! If you are good with large numbers, go ahead. However, it might be a little too time consuming.

It is easiest to get this done if you substitute expressions which have 160 and 19 in them already. So - start from 8 and keep on substituting.

Applying the Extended Euclidean Algorithm to make appropriate substitutions, we get:

8 = 160 - 8*19 (step 1)

3 = 19 - 2*8 (step 2)

= 19 - 2(160 - 8*19)
(substitution for 8)

= 19 - 2*160 + 16*19
(algebraic simplification)

= -2*160 + 17*19
(simplest form)

2 = 8 - 2*3 (step 3)

= (160 - 8*19) - 2*(-2*160 +17*19)
(substitution for 8 and 3)

= 160 - 8*19 + 4*160 - 34*19
(simplify)

= 5*160 - 42*19
(simplest form)

1 = 3 - 1*2 (step 4)

= (-2*160 + 17*19) - (5*160 - 42*19)
(replace 3 and 2)

= -2*160 + 17*19 - 5*160 + 42*19
(simplify)

= -7*160 + 59*19
(simplest form)

This means that d = 59 is the multiplicative inverse of e = 19 mod 160, which can also be written as $59 = 19^{-1} \bmod 160$

The public key is (e, 187) = (19, 187)

The private key is (d, 187) = (59, 187)

Extended Euclidean Algorithm Solutions Example 2

Choose 2 prime numbers, e.g., p = 47 and q = 71

Compute, n = p * q = 47 * 71 = 3337

Compute, z = (p-1)(q-1) = 46 * 70 = 3220

Note: p-1 = 46 = 2 * 23, q-1 = 70 = 2 * 5 * 7

Choose e relatively prime to 3220

e = 79 which does not contain any common factors with p-1, q-1

Compute d such that

d * e = 1 mod 3220, which can be written as, $d = e^{-1} \bmod 3220$

By Extended Euclidean Algorithm, determine that d = 1019
Public key = (e, n) = (79, 3337)
Private key = (d,n) = (1019, 3337)

Extended Euclidean Algorithm Example for e = 69, z = 3220

3220/79	= 40 R 60	=>	60 = 3220 - 79 * 40 (step 1)
79/60	= 1 R 19	=>	19 = 79 - 60 * 1 (step 2)
60/19	= 3 R 3	=>	3 = 60 - 19 * 3 (step 3)
19/3	= 6 R 1	=>	1 = 19 - 3 * 6 (step 4)

Applying the Extended Euclidean Algorithm, we get:

60 = 3220 - 40 * 79 (step 1)
19 = 79 - 1 * 60 (step 2)
= 79 - 1*(3220 – 40*79)
= 79 - 3220 + 40*79
= -3220 + 41*79
3 = 60 - 3*19 (step 3)
= (3220 - 40*79) - 3(-3220 + 41*79)
= 3220 – 40*79 +3*3220 -123*79
= 4*3220 – 163*79
1 = 19 – 6*3 (step 4)
= (-3220 + 41*79) – 6(4*3220 – 163*79)
= -3220 + 41*79 -24 *3220 + 978*79
= -25*3220 + 1019*79

The final equation means that d = 1019 is the multiplicative inverse of e = 79 mod 3220, which can also be written as 1019 = 79-1 mod 3220

The public key is (e, 3337) = (79, 3337)
The private key is (d,3337) = (1019, 3337)

Exponential and RSA Ciphers

In the section on Fermat's Little theorem, we proved the following, which we shall put to use:

Fermat's theorem. If p is a prime number and a is any number not divisible by p, then

$a^{p-1} \equiv 1 \bmod p$

Fermat's theorem (Two Prime version). If p and q are different primes and a is any number not divisible by p or by q, then

$a^{(p-1)(q-1)} \equiv 1 \bmod pq$

Some examples:

$522 \equiv 1 \bmod 23$, $788 \equiv 1 \bmod 89$, $23316336 \equiv 6499$

The first two depend on your knowledge that 23 and 89 are primes. The second depends on the factorization $6499 = 67 \cdot 97 = pq$. In this case $(p - 1)(q - 1) = 66 \cdot 98 = 6336$. Here 2332 is not divisible by 67 or by 97. Less obvious are

$513580908 \equiv 1 \bmod 81493$ and $31891548238 \equiv 1 \bmod 548239$

The first congruence depends on the factorization $81493 = 227 \cdot 359$ into primes. The exponent 80908 is the product of 226 and 358 (each one less than the primes in the factorization of the modulus.) The second congruence depends on the fact that 548238 is a prime (gotten from the PrimeWizard program in the lab.)

Large numbers like the last examples are clearly difficult to handle manually, though computers can deal with them by factoring into primes and by determining if a number is prime. Such congruences are used in cryptography. However the prime numbers will have 100 to 200 digits! In this case, the above numbers will seem like chicken feed. And for such huge primes, computers today can determine a factorization into primes only with great difficulty. A fast computer might take 20 or so years working full time to determine if a number is prime. And factoring

tremendous numbers is even more hopeless. So if a cipher scheme can be found which involves huge numbers and factoring them into primes, the code would be practically unbreakable even with the best computers and smartest minds.

We first show how Fermat's two result are used. We illustrate with $522 \equiv 1 \bmod 23$. If we wish to find $5223 \bmod 23$, we write $223 = 22 \cdot 10 + 3$, so

$5^{223} = {}^{522 \cdot 10+3} = (5^{22})^{10}5^3 \equiv 1^{10}5^3 = 5^3 = 125 \equiv 10 \bmod 23$

This amounts to replacing the exponent 223 by its value mod 22. (We divided by 22 to get the remainder 3.) In general, to compute as mod p, where p and a are as before, we can replace s by its value mod $(p - 1)$. Note: The congruence is mod p but the exponents of a can be taken mod $p - 1$.

Important consequence of Fermat's little theorem. If p is a prime and a is a number not divisible by p, then $a^r \equiv a^8 \bmod p$ when $r \equiv s \bmod (p - 1)$.

Example. Find $17^{1388} \bmod 67$.

Answer: Working the exponent mod 66, we find $1388 \equiv 2 \bmod 66$. So

$17^{1388} \equiv 17^2 = 189 \equiv 25 \bmod 67$.

In exactly the same way, we can reduce a power mod n when n is the product of different primes p and q.

Important consequence of Fermat's little theorem (Two Prime Version). If p and q are different primes and a is a number not divisible by p or by q, then $ar \equiv as \bmod p$ when $r \equiv s \bmod (p - 1)(q - 1)$.

Example. Find $7^{2763} \bmod 143$.

Answer: Note that $143 = 11 \cdot 13$. (If you didn't see this immediately, check $143 \bmod 11$ by the alternating sum test.) So here $p = 11$ and $q = 13$, so (p

$-1)(q-1) = 10 \cdot 12 = 120$. We can reduce the exponent 2763 mod 120. We have $2763 \equiv 3 \bmod 120$ so

$7^{2763} \equiv 7^3 = 343 \equiv 57 \bmod 143$.

Exponential Codes – One Prime. A code attempts to send a message which can be read only by someone who has the key to the code. The message will be a stream of letters, but this is usually coded and transmitted as a stream of numbers. The receiver then decodes these numbers to get the original stream and hence the letters and the message. For example, suppose we wish to sent the message

DONT COME BEFORE FIVE CLOCK

Ignoring spaces, we convert each of these letters into their number equivalent (alway using two digits and ignoring spaces) and we try to code the number stream

04151420031513050205061518050609220515031215031 1

We break this into chunks depending on our coding system – say chunks of 3 – and we then proceed to send coded version of the three digit numbers

041 514 200 315 130 502 050 615 180 506 092 205 150 312 150 311

A (one prime) exponential code is given by the formula $C \equiv P^e \bmod p$ where p is a fixed prime, and e is relatively prime to $(p-1)$. Here, P is the plaintext (as number) and C is its cipher. For example, let's take $p = 947$. Then $p - 1 = 946 = 2 \cdot 11 \cdot 43$. Let's take $e = 53$ which is prime to 946 since it is not divisible by 2 or 3 or 503.1 So in this case, the exponential cipher we are using will be

$C \equiv P^{53} \bmod 947$

Our "message" is 041 514 200 315 . . . . We code this one three digit number at a time. For P = 041, we get

$C \equiv 041^{53} \equiv 544 \bmod 947$

Similarly, P = 514 is coded as $C \equiv 514^{53} \bmod 947$ giving C = 390, and the third triple digit number 200 is coded as $20053 \equiv 677 \bmod 947$. Proceeding in this way, we generate the cipher text for each part of our message and we send the message

544 390 677 ...

confident that no one will understand. The receiver knows all about the prime 947 but she takes exponent 357. (This will be shortly explained.) So she decodes 544 to get $P = 544^{357} \bmod 947$. This works out to 41 or 041 since three digit number are understood. Similarly, $390^{357} \equiv 514 \bmod 947$ and $677^{357} \equiv 200 \bmod 947$. So the decoded message starts as 041 514 200 . . . . Making a stream of digits, we get 041514200. . . or 04 15 14 20 . . . . In letters, the decoded message is DONT. . . .

Summarizing this technique, we chose p = 947 and e = 53 for coding three digit numbers, and we chose d = 357 for decoding:

$C \equiv P^{53} \bmod 947$; $P \equiv C^{357} \bmod 947$

The coder knows exponent 53, the decoder knows exponent 357, both know the prime 947.

What is the magic use of the exponents 357 and 53? How does that work? The answer is the 357 and 53 are inverses mod 946. (Note: 946 not 947!). It works because the coded message is $C \equiv P^{53} \bmod 947$. The decoder then take $C^{357} = P^{53 \cdot 357}$. But this exponent is congruent to 1 mod 946 since that's how 53 and 357 were chosen. Thus $C^{357} \equiv P^1 = P \bmod 947$. Note that we always get and code three digit numbers since that's the size of the remainders mod 947.

Summary of exponential coding mod p. For given prime p and number e relatively prime to p−1, choose d so that $ed \equiv 1 \bmod (p-1)$. Then the coding is given by $C \equiv P^e \bmod p$ and the decoding is given by $P \equiv C^d \bmod p$. In all cases P and C are chosen between 0 and p − 1 inclusive, namely the possible remainders when dividing by p.

Exponential Codes – Two primes. The situation here is almost the same, but it is the basis for the RSA coding system, considered the most powerful, so far unbreakable coding system ever. We start with two different primes p and q. Take n = pq, and take e relatively prime to (p − 1)(q − 1), and take d as an inverse of e mod (p − 1)(q − 1). Then ed ≡ 1 mod (p − 1)(q − 1). The coder is given the exponent e and the decoder the exponent d. Both know the number n = pq. The coder computes the code using the formula $C \equiv P^e \bmod n$ and the decoder decodes using the formula $P \equiv C^d \bmod n$. Everything on the surface is the same. For example, let's choose small numbers so we can transmit two digits (or one letter) at a time. Take p = 11 and q = 17. Then n = pq = 187. Here (p − 1)(q − 1) = 160 with prime divisors 2 and 5. Choose e = 57, say. Then we must find its inverse d mod 160. This calculates to (trust me) d = 73. The coder is given the number n = 187 and exponent e = 57. The decoder is given the number n = 187 and exponent d = 73. Say the coder wants to sent the message NO, or 14 15. He computes 14^{57} mod 187 which works out to be 20, and $15^{57} \equiv 49 \bmod 187$. the message is simply

20 49

The receiver gets the message 20 49. He decodes $2073 \equiv 14 \bmod 187$ and $4973 \equiv 15$. The decoded message is, naturally, 14 15 or NO!

Summary of exponential coding mod pq. For given primes p and q take n = pq and number e relatively prime to (p − 1)(q − 1), choose d so that ed ≡ 1 mod (p − 1)(q − 1). Then the coding is given by C ≡ Pe mod n and the decoding is given by P ≡ Cd mod n. In all cases P and C are chosen between 0 and n − 1 inclusive, namely the possible remainders when dividing by n.

CONCLUSION

RSA is a strong encryption algorithm that has stood a partial test of time. RSA implements a public-key cryptosystem that allows secure communications and "digital signatures", and its security rests in part on the difficulty of factoring large numbers (Diffie, (June 1977)) and (Diffie, (Nov. 1976)). The authors urged anyone to attempt to break their code,

whether by factorization techniques or otherwise, and nobody to date seems to have succeeded. This has in effect certified RSA, and will continue to assure its security for as long as it stands the test of time against such break-ins.

At the time, the RSA encryption function seemed to be the only known candidate for a trap-door one-way permutation, but now, others certainly exist, such as those described in (Glasby, 72(3)(1999), 228–230.) and (Pollard, p. 1973).

The average size of n must increase with time as more efficient factoring algorithms are made and as computers are getting faster. In 1978, the authors of RSA suggested 200-digit long values for n. "As of 2008, the largest (known) number factored by a general-purpose factoring algorithm was [200 digits (663 bits)] long" . Currently, RSA keys are typically between 1024 and 2048 bits long, which experts predict may be breakable in the near future. So far, no one sees 4096-bit keys to be broken anytime soon.

Today, an n no longer than 300 bits can be factored on a PC in several hours, thus keys are typically 4-7 times longer today.

RSA is slower than certain other symmetric cryptosystems. RSA is, in fact, commonly used to securely transmit the keys for another less secure, but faster algorithm. Several issues in fact exist that could potentially damage RSA's security, such as timing attacks and problems with key distribution. I will not go into detail about these issues here. They are described succinctly in (Pohlig, 1975). In fact, these issues have solutions; the only downside is that any device implementing RSA would have to have much more hardware and software to counter certain types of attacks or attempts at eavesdropping.

A very major threat to RSA would be a solution to the Riemann hypothesis. Thus a solution has neither been proven to exist nor to not exist. Development on the Riemann hypothesis is currently relatively stagnant (Miller). However, if a solution were found, prime numbers would be too easy to find, and RSA would fall apart. Undoubtedly, much more sophisticated algorithms than RSA will continue to be developed as mathematicians discover more in the fields of number theory and cryptanalysis.

REFERENCES

Diffie, W. a. ((June 1977)). M. Exhaustive cryptanalysis of the NBS data encryption. In W. a. Diffie. Computer 10.

Diffie, W. a. ((Nov. 1976)). M. New directions in cryptography. IEEE Trans. Inform.

Glasby, S. P. (72(3)(1999), 228–230.). Extended Euclid's algorithm via backward recurrence relations. In *Mathematics Magazine.*

Merkle, R. (n.d.). Secure communications over an insecure channel. In *Submitted to Comm.*

Miller, G. (n.d.). Riemann's hypothesis and tests for primality. Proc. Seventh Annual.

Niven, I. a. (New York, 1972.). H.S. An Introduction to the Theory of Numbers. Wiley.

Pohlig, S. a. (1975). M.E. An improved algorithm for computing logarithms To appear in IEEE Trans. Inform.

Pollard, J. (n.d.). Theorems on factorization and primality testing. Proc. Camb. Phil.

Saracevic , M., Selimi, A., & Selimovic, F. (2018). Generation of cryptographic keys with algorithm of polygon triangulation and Catalan numbers. *Computer Science, 19*, 243-256.

Saračević, M., Selimi, A., & Mujevic, M. (2018). Implementation Example of the Expert system for Decision Support on Android platform based on a specific Dataset. *Periodicals of Engineering and Natural Sciences, 6*(1), 76-83.

Saracevic, M., Selimi, A., & Pepić, S. (2022). Implementation of encryption and data hiding in E-health application. In *In Research Anthology on Securing Medical Systems and Records* (pp. 644-661). IGI Global.

Saračević, M., Selimi, A., & Plojović, Š. (2019). Some specific examples of attacks on information systems and smart cities applications. In *Cybersecurity and Secure Information Systems: Challenges and Solutions in Smart Environments.* 205-226.