CS 467 - Cyber Security: Course Project

Brendan Teasdale (201704913)

April 2021

Contents

1 RSA Cipher

1.1 Background

RSA, or Rivest–Shamir–Adleman, is a public-key cryptosystem that is widely used for secure data transmission. The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the ”factoring problem”. RSA is a relatively slow algorithm. Because of this, it is not commonly used to directly encrypt user data. More often, RSA is used to transmit shared keys for symmetric key cryptography, which are then used for bulk encryption-decryption.[1]

1.2 Problem

The following example of RSA cipher text is presented. Your task is to decrypt it. The public parameters of the systems are $n = 31313$ and $e = 4913$. In order to translate the plaintext back into ordinary English text, you need to know how alphabetic characters are encoded as elements in Zn. Each element of Zn represents three alphabetic characters as in the following examples:

DOG: 3x26^2 + 14x26 + 6 = 2398
CAT: 2x26^2 + 0x26 + 19 = 1371
ZZZ: 25x26^2 + 25x26 + 25 = 17575

6340 8309 14010 8936 27358 25023 16481 25809 23614 7135 24996 30590 27570 26486 30388 9395 27584 14999 4517 12146 29421 26439 1606 17881 25774 7647 23901 7372 25774 18436 12056 13547 7908 8635 2149 1908 22076 7372 8686 1304 4082 11803 5314 107 7359 22470 7372 22827 15698 30317 4685 14696 30388 8671 29956 15705 1417 26905 25809 28347 26277 7897 20240 21519 12437 1108 27106 18743 24144 10685 25234 30155 23005 8267 9917 7994 9694 2149 10042 27705 15930 29748 8635 23645 11738 24591 20240 27212 27486 9741 2149 29329 2149 5501 14015 30155 18154 22319 27705 20321 23254 13624 3249 5443 2149 16975 16087 14600 27705 19386 7325 26277 19554 23614 7553 4734 8091 23973 14015 107 3183 17347 25234 4595 21498 6360 19837 8463 6000 31280 29413 2066 369 23204 8425 7792 25973 4477 30989

You will have to invert this process as the final step in your program.

1.3 Solution

The solution to the problem was done using the Python programming language. First, I needed to initialize my global variables that stored both my public parameters given in the problem and all letters to the alphabet to later be used for conversion.

[1] RSA (Cryptosystem): https://en.wikipedia.org/wiki/RSA_(cryptosystem)

```
PUBLIC_KEY = {
    "n": 31313,
    "e": 4913
}

ALPHABET = [
'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'
]
```

The three helper functions in my program were *gcd()*, *phi()*, *modInverse()*. They are defined below:

```
def gcd(x:int, y:int) -> int:
    """The largest positive integer that divides each of the params."""
    if x == 0:
        return y
    return gcd(y % x, x)


def phi(n: int) -> int:
    """Euler's Totient Function, returns number of totatives of n"""
    result = 1
    for i in range(2, n):
        if gcd(i, n) == 1:
            result += 1
    return result

def modInverse(base: int, mod: int) -> int:
    """Modular multiplicative inverse"""
    return pow(base=base, exp=-1, mod=mod)
```

The function that had the task of decrypting the users input was the function *decryptMessage()* defined below:

```
def decryptMessage():
    """Translates the plaintext back into ordinary English text"""
    exponents = [2, 1, 0]
    encryptedMessage = input("Please enter the RSA encrypted message: \n")
    messageSplit = encryptedMessage.split(" ")
    print("")
    for c in messageSplit:
        d = modInverse(PUBLIC_KEY["e"], phi(PUBLIC_KEY["n"]))
        p = (int(c) ** d) % PUBLIC_KEY["n"]
        for e in exponents:
            letter = math.trunc((p/pow(26, e)) % 26)
            print(ALPHABET[letter], end="")
        print(" ", end="")
```

In this *decryptMessage* function, the first step was to initialize the exponent values. Because we know that each integer value in the encrypted message represents three characters, the exponents that should be available will be $[2, 1, 0]$ (from the problem $a * 26^{\mathbf{2}} + b * 26^{\mathbf{1}} + c * 26^{\mathbf{0}} = p$). Next, was to retrieve the encrypted message from the user and split the message on the spaces. Now that we have an array of all the integer values, we can now get our private key by following the formula: $d = e^{-1} mod(\phi(n))$ using any helper function we already have in the code. Using the private key, we now can get our decrypted plaintext using the formula: $p = c^d mod(n)$. Lastly, all we need to do is get our a, b, and c values by taking p and dividing it by 26 raised to the first exponent in the exponent list modulus 26. The truncated value will give us our first index to use to get our first letter from our 'ALPHABET' global variable. This process is then repeated to give us the next two letters, and so on.

1.4 Output

LAK EWO BEG ONI SMO STL YPO ORS AND YSO ILA NDE VER YSP RIN GTH EEA RTH HEA VES UPA NEW CRO POF ROC KSP ILE SOF ROC KST ENF EET HIG HIN THE COR NER SOF FIE LDS PIC KED BYG ENE RAT ION SOF USM ONU MEN TST OOU RIN DUS TRY OUR ANC EST ORS CHO SET HEP LAC ETI RED FRO MTH EIR LON GJO URN EYS ADF ORH AVI NGL EFT THE MOT HER LAN DBE HIN DAN DTH ISP LAC ERE MIN DED THE MOF THE RES OTH EYS ETT LED HER EFO RGE TTI NGT HAT THE YHA DLE FTT HER EBE CAU SET HEL AND WAS NTS OGO ODS OTH ENE WLI FET URN EDO UTT OBE ALO TLI KET HEO LDE XCE PTT HEW INT ERS ARE WOR SEZ

2 ElGamal Cipher

2.1 Background

ElGamal encryption system is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie–Hellman key exchange. ElGamal encryption can be defined over any cyclic group G, like multiplicative group of integers modulo n. Its security depends upon the difficulty of a certain problem in G related to computing discrete logarithms.[2]

2.2 Problem

Decrypt the ElGamal ciphertext presented in the following table. The parameters of the system are the prime number $p = 31847$, primitive root $e1 = 5$, $e2 = 18074$. Each element of Zn represents three alphabetic characters as in the above problem.

[2]ElGamal encryption: https://en.wikipedia.org/wiki/ElGamal_encryption

ElGamal Ciphertext
(3781, 14409) (31552, 3930) (27214, 15442) (5809, 30274) (5400, 31486) (19936, 721) (27765, 29284) (29820, 7710) (31590, 26470) (3781, 14409) (15898, 30844) (19048, 12914) (16160, 3129) (301, 17252) (24689, 7776) (28856, 15720) (30555, 24611) (20501, 2922) (13659, 5015) (5740, 31233) (1616, 14170) (4294, 2307) (2320, 29174) (3036, 20132) (14130, 22010) (25910, 19663) (19557, 10145) (18899, 27609) (26004, 25056) (5400, 31486) (9526, 3019) (12962, 15189) (29538, 5408) (3149, 7400) (9396, 3058) (27149, 20535) (1777, 8737) (26117, 14251) (7129, 18195) (25302, 10248) (23258, 3468) (26052, 20545) (21958, 5713) (346, 31194) (8836, 25898) (8794, 17358) (1777, 8737) (25038, 12483) (10422, 5552) (1777, 8737) (3780, 16360) (11685, 133) (25115, 10840) (14130, 22010) (16081, 16414) (28580, 20845) (23418, 22058) (24139, 9580) (173, 17075) (2016, 18131) (19886, 22344) (21600, 25505) (27119, 19921) (23312.16906) (21563, 7891) (28250, 21321) (28327, 19237) (15313, 28649) (24271, 8480) (26592, 25457) (9660, 7939) (10267, 20623) (30499, 14423) (5839, 24179) (12846, 6598) (9284, 27858) (24875, 17641) (1777, 8737) (18825, 19671) (31306, 11929) (3576, 4630) (26664, 27572) (27011, 29164) (22763, 8992) (3149, 7400) (8951, 29435) (2059, 3977) (16258, 30341) (21541, 19004) (5865, 29526) (10536, 6941) (1777, 8737) (17561, 11884) (2209, 6107) (10422, 5552) (19371, 21005) (26521, 5803) (14884, 14280) (4328, 8635) (28250, 21321) (28327, 19237) (15313, 28649)

2.3 Solution

The solution to the ElGamal problem, like the RSA implementation, was done using the Python programming language. First, I had initialized the global variables such as the public parameters and encrypted message given in the problem. These were implemented as follows:

```
PUBLIC_PARAMS = {
    "p": 31847,
    "e1": 5,
    "e2": 8074
}

ENCRYPTED_DATA = [(3781, 14409),(31552, 3930),(27214, 15442),
(5809, 30274),(5400, 31486),(19936, 721),(27765, 29284),
(29820, 7710),(31590, 26470),(3781, 14409),(15898, 30844),
(19048, 12914),(16160, 3129),(301, 17252),
(24689, 7776),(28856, 15720),(30555, 24611),
(20501, 2922),(13659, 5015),(5740, 31233),
(1616, 14170),(4294, 2307),(2320, 29174),(3036, 20132),
(14130, 22010),(25910, 19663),(19557, 10145),
(18899, 27609),(26004, 25056),(5400, 31486),(9526, 3019),
(12962, 15189),(29538, 5408),(3149, 7400),(9396, 3058),
(27149, 20535),(1777, 8737),(26117, 14251),(7129, 18195),
(25302, 10248),(23258, 3468),(26052, 20545),
```

```
(21958, 5713),(346, 31194),(8836, 25898),(8794, 17358),
(1777, 8737),(25038, 12483),(10422, 5552),
(1777, 8737),(3780, 16360),(11685, 133),(25115, 10840),
(14130, 22010),(16081, 16414),(28580, 20845),
(23418, 22058),(24139, 9580),(173, 17075),(2016, 18131),
(19886, 22344),(21600, 25505),(27119, 19921),
(23312, 16906),(21563, 7891),(28250, 21321),(28327, 19237),
(15313, 28649),(24271, 8480),(26592, 25457),
(9660, 7939),(10267, 20623),(30499, 14423),(5839, 24179),
(12846, 6598),(9284, 27858),(24875, 17641),
(1777, 8737),(18825, 19671),(31306, 11929),(3576, 4630),
(26664, 27572),(27011, 29164),(22763, 8992),
(3149, 7400),(8951, 29435),(2059, 3977),(16258, 30341),
(21541, 19004),(5865, 29526),(10536, 6941),
(1777, 8737),(17561, 11884),(2209, 6107),(10422, 5552),
(19371, 21005),(26521, 5803),(14884, 14280),
(4328, 8635),(28250, 21321),(28327, 19237),(15313, 28649)]
```

To be able to perform the decryption, I had created two functions, *decrypt()* and *decode()*. The function *decrypt()* retrieves the ciphertext for each tuple from the encrypted message. It uses the private key, which was found to be 7899, to get the decrypted plaintext using the formula: Plaintext $= [C_2 \times (C_1^d)^{-1}] mod(p)$. The function *decode(msg)* takes in the decrypted plaintext that is returned from *decrypt()* and translates the plaintext back to ordinary english. The function definitions are presented below:

```
def decrypt() -> List[int]:
    """Retrieves the ciphertext for each tuple from ENCRYPED_DATA"""
    ciphertext = []
    d = 7899
    for pair in ENCRYPTED_DATA:
        c1, c2 = pair
        decrypted_text = (c2 * pow(c1**d, -1, PUBLIC_PARAMS["p"]) +
                          PUBLIC_PARAMS["p"]) % PUBLIC_PARAMS["p"]
        ciphertext.append(decrypted_text)
    return ciphertext

def decode(msg: List[int]) -> None:
    """Translates the plaintext back into ordinary English text"""
    for c in msg:
        word = ""
        for j in range(3):
            char = int(c % 26)
            word = chr(char + 65) + word
            c = (c - char) / 26
        print(word + " ", end="")
    print("")
```

2.4 Output

SHE STA NDS UPI NTH EGA RDE NWH ERE SHE HAS BEE NWO RKI NGA NDL OOK SIN TOT HED IST ANC ESH EHA SSE NSE DAC HAN GEI NTH EWE ATH ERT HER EIS ANO THE RGU STO FWI NDA BUC KLE OFN OIS EIN THE AIR AND THE TAL LCY PRE SSE SSW AYS HET URN SAN DMO VES UPH ILL TOW ARD STH EHO USE CLI MBI NGO VER ALO WWA LLF EEL ING THE FIR STD ROP SOF RAI NON HER BAR EAR MSS HEC ROS SES THE LOG GIA AND QUI CKL YEN TER STH EHO USE