

Homework 1 (CSCI-B 504)

Kushal Pokharel

September 2025

1 Implement the DES system in C++, Java, perl, etc. and then decrypt the ciphertext.

(NOTE: copy DES parameters and ciphertext from ciphers-parameter-matrix.zip Download ciphers-parameter-matrix.zip.) using key=8B2A7FF25E98C35D (in hexadecimal). The decrypted plaintext should be in normal english text, not in hexadecimal. (Hints: 1. the ciphertext is in hexadecimal, therefore you need to translate every two bytes in ciphertext into one byte in binary format. for example: B85D, when you read from the file, you get four chars: B,8,5,D. B and 8 should change to one byte:10111000, and 5 and D should be 01011101. 2. you need to skip the new line character. 3. using the following result to verify your algorithm first: plaintext (in hexadecimal): 0123456789ABCDEF key (in hexadecimal): 133457799BBCDFF1 ciphertext(in hexadecimal): 85E813540F0AB405)

1.1 Approach

Since the key was given, the problem was just to implement the DES circuit. With a lot of bit manipulation and representation of the given text in a Vector of u8's (buffer), I was able to get the encryption of the given test case. And the decryption can be obtained with the same circuit when applied in reverse, actually.

One problem I had with this was: Suppose 8f was the first byte, then if I were to access the 5th bit, I would look for the first byte and shift it to the left by 5 bits, and then AND with 01. But this was actually the third bit, going from left to right. I got stuck for a while in this convention.

Block of 64 bytes.

1.2 Decrypted plaintext

Computer
y resear

y Intern
are bec
re compl
could ca
more ha
previous
s: Inter
rity
Sy
-Force r
engineer
hta says
ster wor
example,
rily had
ect thro
rt usual
ed, thus
ing its
to sprea
t as pos
imda, ho
epresent
complex
that it
d intern
rks and
f local
s that w
er to br
and liv
ne Labs'
lman
ag
t worms
more to
te syste
ploiting
e
vulne
es or fi
her ways
agate on
e the sy
artner's
Stienno

against
slow” w
cks that
oticed b
strators
urity sy
til they
a cata
attack,
eha say
tion adv
uld allo
to
carr
otent ex
s withou
identifi
ing so.
the cu
curity p
are the
esponsib
users a
are
ven
o create
ed produ
are eas
but als
ulnerabl
ternet a
nterpris
o more t
t themse
adopting
d
firew
t inspec
s and in
detectio
s that b
n data
tocols i
f just m
patterns
mpanies

lso asse
r securi
and wri
for whe
w applic
an conne
he Inter
ennon ad
organiza
ould not
a monol
archit
but vary
omponent
er to be
tain pos
eak-ins.

2 Decrypt the ciphertext encrypted by RSA method.

The encoding for the RSA is as following: 1) divide the plaintext into blocks. 2) each block forms a big number as following: (because all characters are from a 10X10 matrix) find the row and column number from the matrix for the first character and the row number and column number are connected together to form a partial number; find the row and column number from the matrix for the second character, then append the row number and column number to the above number,... for example: This homework.... will be encoded to 527273830072798275... (because T is in row 5 and column 2. Note: the row and column numbers begin from Zero.) 3) encrypt each block using RSA method. 4) Therefore, after your decryption, you need to decode by changing every pair of two numbers (as row number and column number) into a character by looking up the matrix. 5) you can get the ciphertext and matrix from ciphers-parameter-matrix.zip Download ciphers-parameter-matrix.zip 6) Every row in the ciphertext is an encrypted RSA number. So you need to decrypt the ciphertext row by row. 7) Here is the public key for this RSA system: $n = 68102916241556953901301068745501609390192169871097881297$ (modulo) $b = 3663908873840754089455092320224101809992059348223191165$ (public exponent) 8) This problem needs big integer. There are several possibilities: a). use Java which contains BigInteger class. b). use GMP C library which contains operations on big integers Here are the manual link: <http://gmplib.org/manual/> and the integer function link: <http://gmplib.org/manual/Integer-Functions.html> Integer-Functions c). download Crypto++ Library from <http://www.eskimo.com/~weidai/cryptlib.html>. following instructions in readme. You may need to play for a while. Crypto++ Library can be run on both Windows and Unix/Linux. d). download LiDIA library from <http://www.informatik.tu-darmstadt.de/TI/LiDIA/>. LiDIA is only for Unix/Linux. 9) There is a need to add one ZERO before the numbers having odd number of digits. 10) Since there is no factorization function in the above Libraries you need to write your own Pollard p-1 factoring program to find the factor of n in this problem. Note: set B=1500 or larger.

2.1 Approach

Use pollard's p-1 factoring algorithm to factor N, but since 1500 ($= 2*2*3*5*5*5$) was given as a hint, I tried manually guessing B (with small primes to the power big number) which would be a multiple of (p-1). But it didn't work so I worked with seive table to find the primes up to 500 and raise each prime to the power until it becomes greater than N. This finally gave the non-trivial gcd between $x^{B \% n} - 1$ and N.

After getting the factor p, I got the other factor with N/p . Solved for the $\phi(n) = (p-1)(q-1)$. Found the inverse of the public exponent in $\phi(n)$ with Extended Euclidean algorithm, which is the private exponent used for decryption.

Each line from the ciphertext was read as a number and raised to the power of this private exponent mod n. This gives the decrypted number corresponding

to the ciphertext and this number was traversed two character at a time to get the equivalent entry with row and column in the matrix finally leading to a sensible plaintext.

2.2 p, Private exponent and, decrypted text

2.2.1 p and q as factors of N

p=761059198034099969 q=89484387571261623539483274324628239313

2.2.2 Private exponent

743634723523581782187325327276236523726254293

2.2.3 Decrypted text

As the attack was in progress, the bombs began to fall in earnest, the officers began shouting orders for everyone to head to the nearby rifle range to be issued firearms and ammunition.

About this time, the men at Schofield could look down towards the harbor and view the terrible sight unfolding as the attacking planes began to wreak their havoc among the anchored ships in the harbor. They could see what appeared to be a "mist" or "fog" rising from the harbor area. Jacques did not elaborate on this (quite possibly results of the bombing).

After the men were in the process of being armed, the men who were anti-aircraft trained, such as Jacques, were ordered to head to the mouth of the harbor to man the battery of anti-aircraft guns (3-inch) located there. The guns were situated in a "firing pit" of sorts that allowed for the weapon to rotate to follow attacking aircraft.

Upon reaching this assignment, the men began firing on the attacking aircraft (it is assumed that at this time, the attack had entered into the second wave of aircraft).

The weapons were fired and targets were plentiful, indeed. The firing was to the extent that the barrels became red hot and the guns began jamming.

Some of the jamming guns would actually "buck like a bucking bronco" and literally fall back onto the gunners in the firing pits! The officers and noncoms on hand began issuing orders to exit the firing pits for fear of the weapons exploding and injuring or killing the men in the pits. Jacques did as he was told, and got out of the firing pit, and began to run, falling into a large hole. He recalls being dazed, stunned and appeared to have fallen into a large "black hole" in which he had to climb out. (bomb crater?)

3 Suppose Bill has carelessly revealed his decryption exponent to be $a=14039$ in his RSA system with public key $n=36581$ and $b=4679$. Implement a randomized algorithm to factor n given this information. Test your algorithm with the "random" choices $w=9983$ and $w=13461$. Show all computations.

We can use randomized algorithm which is based on the fact that if we are able to find the non-trivial square root of 1, then we can easily factorize n . Even though, the choice of w is random, it still guarantees that n can be factored with $1/2$ probability.

The code is given below. I reused the exponentiation and gcd function that I had written for breaking RSA in the above question. Exponentiation is based on square and multiply.

```
use curv::BigInt;

// Randomized algorithm to factor n based on square root of 1(non-trivial)
// Used when private exponent is exposed - implication : if you expose private exponent, N can be factored
// using a different private key isn't the enough then, you need to change N.
use crate::RSA::{exponentiation, gcd};

pub fn rsa_factor(n:BigInt, a:BigInt, b:BigInt)-> BigInt{
    let mut phi = a*b-1;
    println!("here {:?}", &phi>>1);
    //divide by 2 until we can to get an odd (r)
    while (&phi & BigInt::from(1)) ==BigInt::from(0){
        phi = &phi>>1;
    }
    //choose w at random, for now set some value to w.
    let w = BigInt::from(9983);
    println!("Random choice of w is {w}");
    println!("Value of s (odd) {phi}");

    let x = gcd(w.clone(), n.clone());
    if x<n && x>BigInt::from(1){
        println!("Unlucky. use different w");
        x
    }
    else{
        let mut v = exponentiation(w.clone(), phi.clone(), n.clone());
```

```

let mut v0 = v.clone();
if v==BigInt::from(1){
    println!("Unlucky. use different w");
    return BigInt::from(-1);
}
while v!=BigInt::from(1){
    v0 = v.clone();
    println!("Value of v {v0}");

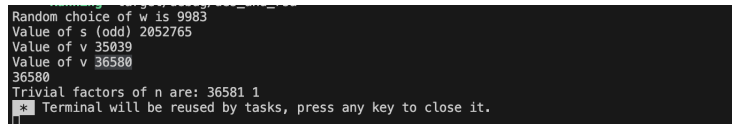
    // repeatedly power w by 2 until we reach s in which case v will be 2.
    v = exponentiation(v.clone(), BigInt::from(2), n.clone());

}
if &v0%&n==BigInt::from(-1){
    println!("Unable to factorize n as we didn't get the non-trivial factor but -1");
    return BigInt::from(-1)
}
else{
    return gcd(v0+1, n);
}
}

fn rsa_decryption_exponent(){
    let n = BigInt::from(36581);
    let result = RSA_exponent::rsa_factor(n.clone(), BigInt::from(14039), BigInt::from(4091));
    if result == BigInt::from(-1) || result == BigInt::from(1) || result==BigInt::from(n){
        println!("Trivial factors of n are: {result} {:?}" , {n/&result});
    }
    else{
        println!("Two factors of n are: {result} {:?}" , {n/&result});
    }
}
}
}

```

3.1 Output with w=9983 and 13461



```

Random choice of w is 9983
Value of s (odd) 2852765
Value of v 35039
Value of v 36580
36580
Trivial factors of n are: 36581 1
Terminal will be reused by tasks, press any key to close it.

```

Figure 1: $W = 9983$ (Doesn't factor n)


```
Running target/debug/oes_and_rsa
Random choice of w is 13461
Value of s (odd) 2052765
Value of v 11747
Value of v 8477
Value of v 14445
We are able to factorize n successfully using random w = 13461
Two factors of n are: 233 157
[*] Terminal will be reused by tasks, press any key to close it.
```

Figure 2: $W = 13461$ (Factors n)

4 Use the Extended Euclidean Algorithm to compute the multiplicative inverse (show all your computations step by step): $6283^{-1} \bmod 9347$.

To compute gcd of a number (b) in a modulus (a), we can use the Euclidean algorithm as:

$a = q_1 * b + r_1$ (r_1 is the remainder, q is the quotient when a divided b)

$b = q_2 * c + r_2$

\vdots
 $k = q_i * \text{gcd} + r_i$

If we substitute the lower equations to upper equations, we get a linear combination of $a * X + b * Y = \text{gcd}(a, b) = 1$ (for co-prime a and b).

If we apply %a to this equation, we get $bY \% a = 1$, which means Y is the inverse of b in this modulus.

```

----- rsa::tests::test_extended_euclidian stdout -----
Setup with a = 9347 b = 6283 s1 = 1 s2=0 t1=0 t2 = 1
Step 0 with quotient = 1 a = 6283 b = 3064 s1 = 0 s2=1 t1=-1 t2 = -1
Step 1 with quotient = 2 a = 3064 b = 155 s1 = 1 s2=-2 t1=-1 t2 = 3
Step 2 with quotient = 19 a = 155 b = 119 s1 = -2 s2=39 t1=3 t2 = -58
Step 3 with quotient = 1 a = 119 b = 36 s1 = 39 s2=-41 t1=-58 t2 = 61
Step 4 with quotient = 3 a = 36 b = 11 s1 = -41 s2=162 t1=61 t2 = -241
Step 5 with quotient = 3 a = 11 b = 3 s1 = 162 s2=-527 t1=-241 t2 = 784
Step 6 with quotient = 3 a = 3 b = 2 s1 = -527 s2=1743 t1=784 t2 = -2593
Step 7 with quotient = 1 a = 2 b = 1 s1 = 1743 s2=-2270 t1=-2593 t2 = 3377
Step 8 with quotient = 2 a = 1 b = 0 s1 = -2270 s2=6283 t1=3377 t2 = -9347
3377
Inverse of 6283 in 9347 is: 3377

```

Figure 3: Step by step calculation of inverse using EEA

Initialized with a = 9347 (a will always be the bigger number), b=6283, s1=1, s2=0, t1=0, t2 = 1

Repeatedly apply

```

let q = &a/&b;
b = &a % &b;
a = c;
let temp_s = &s1 - &q*&s2;
(s1,s2) = (s2,temp_s);
let temp_t = &t1 - &q*&t2;
(t1,t2) = (t2,temp_t);

```

Return t1 which is the inverse of b in a.

- 5 Even though RSA in principle can be secure (due to the difficulty of factorization). But a concrete implementation of a RSA system can be insecure. As We discussed in class that RSA can be broken if prime p or q is selected inappropriately. Apart from that, RSA can fail if the protocol is not used appropriately, different RSA protocol failures do exist in real world. One of them is the following and please solve the problem to understand the reason. Suppose that three users in a network, say Bob, Bart, and Bert, all have public encryption exponent $b=3$. Let their moduli be denoted by n_1, n_2, n_3 and assume that n_1, n_2, n_3 are pairwise relatively prime. Now, suppose that Alice encrypts the same plaintext x to send to Bob, Bart, and Bert. That is, Alice computes $y_i = x^3 \bmod n_i$, for $1 \leq i \leq 3$. Describe how Oscar can compute x , given y_1, y_2, y_3 , without factorizing any of the moduli. Please generate an example to illustrate such an attack.

5.1 Approach

The pairwise co-prime line gave me a hint that this could be an attack based on CRT. But even if I find a y that is in modulo $n_1 * n_2 * n_3$, I wasn't able to reason how finding a cube root in this modulo would be easy. But I knew that in real numbers, finding a cube root is easy as we can sort of binary search. Hence, I came to realise that whatever y I get in $n_1 * n_2 * n_3$, that would be the real cube of that number as:

x has to be less than $\min(n_1, n_2, n_3)$. And $n = n_1 * n_2 * n_3$ is greater than $\min(n_1, n_2, n_3)^3$. Hence whatever y I get is the real cube of x and was able to solve for x .

For the sake of an example, I use $x=14$, $n_1=5*7$, $n_2=11*13$ and $n_3=17*19$ (resembles RSA and are pairwise co-prime). I got the value of y_i in each modulus

n_i and used CRT to find y in modulus n . Finally, getting the cube root of n gave me the value of x (plaintext).

5.2 Screenshot of the attack

```

--- rsa::tests::crt stdout ---
Choice of x : 14
Choice of three n's - n1, n2, n3 : 35 143 323
Bigger modulo n = : 1616615
x raised to 3 in modulus n1,n2,n3 : 14 27 160
Setup with a = 35 b = 24 s1 = 1 s2=0 t1=0 t2 = 1
Setup with a = 143 b = 8 s1 = 1 s2=0 t1=0 t2 = 1
Setup with a = 323 b = 160 s1 = 1 s2=0 t1=0 t2 = 1
Inverse of n2*n3 in n1 : -16
Inverse of n1*n3 in n2 : 18
Inverse of n1*n2 in n3 : 107
Finding y with the formula &y1*&n2*&n3*&inv1+&y2*&n1*&n3*&inv2+&y3*&n2*&n1*&inv3; y=80833494
y in modulus n where n=n1*n2*n3 = 2744
Finding cube root in real field is easy: cube root of y 13.999999999999998

successes:
  rsa::tests::crt

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 4 filtered out; finished in 0.00s

* Terminal will be reused by tasks, press any key to close it.

```

Figure 4: Step by step calculation of inverse using EEA

5.3 Code

```

#[test]
fn crt(){
    // find the inverse of each number in their moduli to get such x that satisfy all the
    let n1 = BigInt::from(5*7);
    let n2 = BigInt::from(11*13);
    let n3 = BigInt::from(17*19);
    let x: BigInt = BigInt::from(14);
    println!("Choice of x : {x}");
    println!("Choice of three n's - n1, n2, n3 : {n1} {n2} {n3}");
    println!("Bigger modulo n = : {:?}",&n1*&n2*&n3);

    let y1 = x.pow(3)%&n1;
    let y2 = x.pow(3)%&n2;
    let y3 = x.pow(3)%&n3;
    println!("x raised to 3 in modulus n1,n2,n3 : {y1} {y2} {y3}");

    let n = &n1*&n2*&n3;

    let (_, mut inv1) = get_inverse_of_b_in_phi( (&n2*&n3)%&n1, n1.clone());
    let (_, mut inv2) = get_inverse_of_b_in_phi( (&n1*&n3)%&n2, n2.clone());
    let (_, mut inv3) = get_inverse_of_b_in_phi( (&n1*&n2)%&n3, n3.clone());

    println!("Inverse of n2*n3 in n1 : {inv1} ");

```

```

println!("Inverse of  $n_1 n_3$  in  $n_2$  : {inv2} ");
println!("Inverse of  $n_1 n_2$  in  $n_3$  : {inv3} ");

// Y is constructed in such a way that when %n1 it gives y1, when %n2 it gives y2 and
let mut y = &y1*&n2*&n3*&inv1+&y2*&n1*&n3*&inv2+&y3*&n2*&n1*&inv3;
let ymodn1 = &y%&n1;

println!("Finding y with the formula  $y_1 * n_2 * n_3 * \text{inv}_1 + y_2 * n_1 * n_3 * \text{inv}_2 + y_3 * n_2 * n_1 * \text{inv}_3$ ");

// With chinese remainder theorem y%n1 should give y1.
assert_eq!((ymodn1+&n1)%&n1, y1);

// After getting the y in higher modulo, get the cube root of that number in real number
// should be our answer.
while &y< &BigInt::from(0){
    y+=&n;
}
y = y%&n;

println!("y in modulus n where  $n=n_1*n_2*n_3 = \{y\}$ ");

let y_string = y.to_str_radix(10);

let y_64 = u64::from_str_radix(&y_string, 10).unwrap();
let cube_root_of_y = f64::powf(y_64 as f64, 1.0/3.0);

println!("Finding cube root in real field is easy: cube root of y {cube_root_of_y}");
assert_eq!(BigInt::from(cube_root_of_y.round() as u64), x);
}

```