CSE13S Assignment Writeup

Zion Kang

February 2023

1 Discoveries

```
n = 5b6cce8d160c2f5e5ee943287385bd02b72b4322a279667dd862808c998d1fd9d
pq = 3429a98159d88833eb05a592a8260d94cb726e46705601
d = 3f67d811e24755737bb91fb6daa19254c567848cdb565
size for encrypt = 16
*** stack smashing detected ***: terminated
==7953==
==7953== Process terminating with default action of signal 6 (SIGABRT)
==7953==    at 0x497C26B: __pthread_kill_implementation (pthread_kill.c:44)
==7953==    by 0x497C26B: __pthread_kill_internal (pthread_kill.c:78)
==7953==    by 0x497C26B: pthread_kill@@GLIBC_2.34 (pthread_kill.c:89)
==7953==    by 0x4925C45: raise (raise.c:26)
==7953==    by 0x490C7FB: abort (abort.c:79)
==7953==    by 0x496F0BD: __libc_message (libc_fatal.c:155)
==7953==    by 0x4A19669: __fortify_fail (fortify_fail.c:26)
==7953==    by 0x4A19635: __stack_chk_fail (stack_chk_fail.c:24)
==7953==    by 0x10AEE1: main (in /home/zkang5/cse13s/asgn5/ss)
==7953==
==7953== HEAP SUMMARY:
==7953==     in use at exit: 1,744 bytes in 19 blocks
==7953==   total heap usage: 244 allocs, 225 frees, 54,751 bytes allocated
==7953==
==7953== LEAK SUMMARY:
==7953==    definitely lost: 720 bytes in 18 blocks
==7953==    indirectly lost: 0 bytes in 0 blocks
==7953==      possibly lost: 0 bytes in 0 blocks
==7953==    still reachable: 1,024 bytes in 1 blocks
==7953==         suppressed: 0 bytes in 0 blocks
==7953== Rerun with --leak-check=full to see details of leaked memory
==7953==
==7953== For lists of detected and suppressed errors, rerun with: -s
==7953== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Aborted (core dumped)
```

Figure 1: Error encountered when trying to encrypt file

While completing this assignment, I was able to debug by using valgrind and creating print statements within my functions to pinpoint where the bug was. However, when trying to fix my encrypt file function, I ran into an error I had never seen before or knew where to begin to look.

After consulting other peers/TA's on discord and coming to no solution, I decided on rewriting the way I iterated through a file. As a result of this, I came across the ftell function, which would get the exact byte size of a file. In order to use this function though, I first had to use fseek to set the file pointer to the end of the file as can be seen in the code bit below.

```
// while there are still unprocessed bytes in infile
uint64_t bytestoread = 0; // var for total bytes to read in file
fseek(infile, 0, SEEK_END); // set pointer in file to end of file
bytestoread = ftell(infile); // ftell gives total size of file
fseek(infile, 0, SEEK_SET); // set pointer in file back to top
while (bytestoread > 0) {
    // read at most k - 1 bytes and let j be the number of bytes actually read
    uint64_t j = fread(block + 1, sizeof(uint8_t), size - 1, infile);
    if (j < 1) { // when we reach EOF
        j = fread(
            block + 1, sizeof(uint8_t), bytestoread, infile); // read in bytes left in file
    }
    bytestoread -= j; // update bytes left to read by subtracting bytes read
```

Figure 2: Code bit of encrypt file

By rewriting my code to track the total bytes in the file and updating it by subtracting the number of bytes read in from fread, I was able to get the exact read on the last few bytes left before it reached the end of the file and not run into any errors.

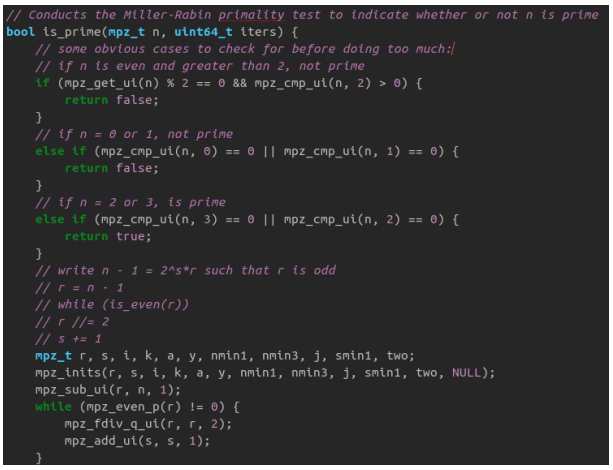
Another troubling function was the isprime function. When approaching this problem, a part that I struggled to understand was writing $n-1 = 2^s * r$ such that r is odd. A huge help in interpreting this equation into code was the python code bit below.

```
def get_d_r(n):
    """
    Factors n into the form d * 2 ** r.
    """
    d = n
    r = 0
    while is_even(d):
        d //= 2
        r += 1

    return (d, r)
```

Figure 3: Code bit of isprime in python

After writing this bit and testing the function, I ran into a couple of issues when trying to test the primality of some simple cases like for the numbers 1, 2, and 3. As a result of Ben Grant's suggestions on discord, I decided to write some obvious cases which returned the respective primalities before running the rest of the code which would take much longer.



```
// Conducts the Miller-Rabin primality test to indicate whether or not n is prime
bool is_prime(mpz_t n, uint64_t iters) {
    // some obvious cases to check for before doing too much:
    // if n is even and greater than 2, not prime
    if (mpz_get_ui(n) % 2 == 0 && mpz_cmp_ui(n, 2) > 0) {
        return false;
    }
    // if n = 0 or 1, not prime
    else if (mpz_cmp_ui(n, 0) == 0 || mpz_cmp_ui(n, 1) == 0) {
        return false;
    }
    // if n = 2 or 3, is prime
    else if (mpz_cmp_ui(n, 3) == 0 || mpz_cmp_ui(n, 2) == 0) {
        return true;
    }
    // write n - 1 = 2^s*r such that r is odd
    // r = n - 1
    // while (is_even(r))
    // r //= 2
    // s += 1
    mpz_t r, s, i, k, a, y, nmin1, nmin3, j, smin1, two;
    mpz_inits(r, s, i, k, a, y, nmin1, nmin3, j, smin1, two, NULL);
    mpz_sub_ui(r, n, 1);
    while (mpz_even_p(r) != 0) {
        mpz_fdiv_q_ui(r, r, 2);
        mpz_add_ui(s, s, 1);
    }
```

Figure 4: Code bit of isprime in python

2 Application of Public-Private Cryptography

Public-Private cryptography comes in handy when solving problems regarding the confidentiality/integrity of data. Because of this, its applications in modern society consist of but are not limited to: secure message transmission, certification, and authentication.

More specifically, the usage of public keys allows for the signing of messages on behalf of an original sender. "Integration of proxy signature and signcryption public key paradigms provides secure transmission. It is efficient in terms of computation and communication costs. It is used for low computers in which a given device may transmit and receive messages from an arbitrarily large number of other computers" ("A Survey on the Applications of Cryptography"). Based on the information from this article, it could be seen that such cryptographic algorithms are used in low power computers for secure message transmission.

Regarding certificates and authentication, "a certificate is an electronic is an electronic document which identifies an individual, a server, a company, or some other entity and to associate that identity with a public key. Certificate authorities issued certificate which binds a particular public key to the name of the entity that the certificate identifies" ("A Survey on the Applications of Cryptography").

One day-to-day usage of the application mentioned above is email. Due to the way only the certified public key works with its corresponding private key file allows for the encryption of emails. As emails are something we use everyday, it is quite amusing to think that cryptographic algorithms are in use behind some of the services I used online on a day-to-day basis.