CS161 Project 0

Due Monday, September 22 at 12:00pm

1 The Bear Begins

After a particularly invigorating 161 lecture about how notorious minds attempt to subvert cryptography to dupe innocent civilians, you realize that the police force just isn't enough. Following the infamous billionaire Bruce Wayne's example, you don a bat costume and prepare to dish out vigilante justice against cybercrime. Realizing that that the batsuit may be construed as a violation of copyright, you switch to a golden bear costume instead.

Not having access to Mr. Wayne's vast network to discover crimes, you immediately head to your favorite news source: The Onion. You're shocked to see that the front page is reporting that a money-hungry Certificate Authority (CA) has assigned a certificate to the notorious Joker! You suit up and prepare to bring down this villain.

2 Your Task

Moved by your dedication to protect the innocent denizens of Interwebz, your TAs lend you their aid. They track down a few signed certificates, out of which one is definitely the Joker's. After much sleuthing, they identify "whysoserious.com" as the domain name field for his certificate. They leave it to you to decrypt the certificates and discover the which one belongs to the Joker.

2.1 Cracking the Ridiculously Sinister Algorithm

The corrupt CA uses the Ridiculously Sinister Algorithm (RSA) to sign their certificates. Fortunately, this means that you can use the CA's public key to validate all of the certificates and check the domain names. Find the one that belongs to the Joker, so you can put an end to his evil schemes.

In **proj0.c**, fill in the `perform_rsa()` method. The modulus (N) and exponent (d) needed for RSA are provided in the files **ca_n.key** and **ca_d.key** respectively. In this case, the CA's public key is **ca_d.key**, and is using e as the signing key. The certificates are stored in **cert0.crt**, **cert1.crt**, and **cert2.crt**.

We've provided you with quite large numbers, so the program will be extremely slow (read: not terminate) if you use a naive algorithm. Instead, think about some optimization methods on RSA that were discussed in lecture and discussion.

The usage of **proj0.c** is provided if you just try running `./proj0` with no arguments. You must provide it with all arguments (**m, n, d**) for it to work. Example usage: To check your RSA solution against **cert0.crt**, you would use the command `./proj0 -m cert0.crt -n ca_n.key -d ca_d.key`. It will use your answer to attempt to decrypt the certificate and display its contents.

Compile with `make`.

2.2 Answering to the Media (Write-up)

Alas, the life of a vigilante is not an easy one. Even after your daring defeat of the Joker, you are not regarded as a hero by all. The media demands that you tell the public about your actions, or face the consequences of being a social outcast like Batman. Deciding that Batman's lonely life is not for you, you hang up the bear suit and prepare to answer the demanding questions of reporters.

In **proj0-writeup.txt**, answer the following questions:

1. Which certificate belongs to the Joker? Simply writing the file name will suffice.
2. Explain the steps you used to decrypt and check certificates in Task 1 (RSA).
3. What is the runtime of RSA if you don't use repeated squaring? Assume that you calculate m^e via repeated multiplication. Explain your reasoning. (Assume that multiplication is $O(1)$.)
4. What is the runtime of RSA using repeated squaring? Explain your reasoning. (Assume that multiplication is $O(1)$.)
5. What is the role of a Certificate Authority? What are some dangers if a certificate authority signs certificates without properly background checking the requestor?

Do not spend more than a paragraph on each response.

3 Certificate Format

Each certificate will have the fields **DomainName**, **PublicKey**, **ExpirationDate**, as discussed in class. These fields will be encased between BEGIN_CERTIFICATE and END_CERTIFICATE flags. Below is an example plaintext certificate:

```
BEGIN_CERTIFICATE
DomainName: securityisawesome.com
PublicKey: 0x123456789
ExpirationDate: 12-31-2020
END_CERTIFICATE
```

4 Submission

We will be using glookup for submission. Copy your submission files onto an instructional machine into a directory labelled `proj0` and run the command `submit proj0`.

Submission files:

- **proj0.c**: contains the solutions to the project.
- **proj0-writeup.txt**: contains the solutions to the write-up.

5 Using GMP Arithmetic Library

You will be using GMP arithmetic library for this homework. We'll be dealing with numbers that are far too large to fit in the traditional `long` data type. GMP provides `mpz_t`, which can store large numbers, and arithmetic functions on `mpz_t` to resolve this issue.

5.1 Installing GMP

Below are the installation instructions for GMP on a Linux machine where you have **root access**. This will not work on instructional machines; see Piazza for details on that.

1. Download the zipped version of GMP from `https://gmplib.org/#DOWNLOAD`.
2. Unzip the downloaded file.
3. `cd` into the unzipped directory and run `./configure`.
4. Use `sudo make install` to install it for all users.

5.2 Allowed functions

You may only use the following functions, or their `*_ui` forms. If you feel that you need to use any functions not on this list, you may ask the GSIs for permission. **DO NOT** use unapproved methods. You will lose a lot of points, or possibly get a **zero** for breaking this rule. We are aware of existing RSA implementations in GMP. Do not use them.

- `void mpz_init(mpz_t input)`: allocates memory and initializes an `mpz_t` variable to 0. Similar to `calloc()`.
- `void mpz_clear(mpz_t input)`: frees the memory allocated to `input`. Similar to `free()`.
- `void mpz_mul(mpz_t result, mpz_t a, mpz_t b)`: stores the result of `a*b` into `result`.
- `void mpz_div(mpz_t result, mpz_t a, mpz_t b)`: stores the result of `a/b` into `result`.

- `void mpz_add(mpz_t result, mpz_t a, mpz_t b)`: stores the result of `a+b` into `result`.
- `void mpz_sub(mpz_t result, mpz_t a, mpz_t b)`: stores the result of `a-b` into `result`.
- `void mpz_mod(mpz_t result, mpz_t a, mpz_t b)`: stores the result of `a%b` into `result`.
- `int mpz_cmp(mpz_t a, mpz_t b)`: compares the values of `a` and `b`. Returns a positive value if `a>b`, a negative value if `b>a` and 0 if `a==b`.
- `void mpz_set_str(mpz_t result, char* str, int base)`: parses `str` as a number, using `base` (ex. 16 means hexadecimal), and stores the result into `result`.
- `void gmp_printf(char* fmt, argv...)`: exactly like `printf`, but allows you to print `mpz_t` variables using the `%Zd` format. (This should only be used for debugging.)