# CSCI 400 Cryptography Lab
# Topic: Cryptography, Collisions, MITM, and High Factors

Prof. Dietrich

September 5, 2019

Group size: 4 — Setup needed: Windows/MacOS X, Unix hosts, DETER (or closed network setup), Amazon EC2.

## 1 General description

This lab explores hash collisions, man-in-the-middle attacks (MITM, only cryptographic), and website certificates.

1. For hash collisions: contrary to dictionary attacks, where we are trying to find M : H(M) = C, here given M, we are looking for M' : H(M) = H(M').
2. For website certificates: we want to challenge the security of a certificate. With a weak key we can attack the connection and decrypt network traffic.

Except maybe for the first item, these exercises require proper planning, computational power, and time management.

## 2 File with samples and results

The files are in lab-crypto1.zip. It contains most files to get you started. In some cases, you still have to develop some scripts or write your own code to improve the results.

## 3 Attacking systems

We attack systems at multiple levels: from challenging the trust in integrity primitives such as MD5, we move to question the network itself and its ability to provide a secure channel since there may be an attacker in the middle. Further the trust instilled in us by a certificate needs to be proper constructed: picking a low-strength key could allow for an attacker to extract enough information to eavesdrop on the connection secured by SSL/TLS.

### 3.1 Requirement

- You should use the tools provided in the file, but are not limited to them. If for some reason, you choose not to, please justify your choice. If you wrote scripts or additional code, please include it.

- Your output should demonstrate that you have done the experiments, such as screenshots, Unix typescripts, packet or screen recordings.

## 4 Tasks

### 4.1 Finding MD5 hash collisions

Using the tools provided, evilize, selfextract, and web_version, find MD5 hash collisions of each type:

- Executable files
- Self-extracting archives
- Strings

See the lab file for examples. Can you extend this to other hash functions? If so, how? If not, why not?

### 4.2 Finding SHA-1 hash collisions

Using the tools provided, find SHA-1 hash collisions for:

- Two PDF documents

Both online and offline tools are provided. Examples of SHA-1 collisions are shown in the lab file. Start with the online tools to produce a collision between two images. Then extend to the offline tool for greater flexibility of the characteristics of the input documents and use two PDFs of your choice.

Prepare a report of your findings and for each tool include the 2 original images or PDFs, the hash values for each original file, the newly created PDFs, and the new common hash values.

### 4.3 Man-in-the-middle attacks on SSL/TLS

For this task, you will need to perform some man-in-the-middle attacks (MITM) on SSL/TLS-protected IMAP traffic, e.g. checking on email. This can be done with a mobile device and a broadband router that you control, or with a set of two virtual machine guests implementing the email client and router respectively. The router should provide access to the Internet (facing Hotmail, Gmail, etc.) for your email client. All traffic for the email client must go through the router for this to work.

- Find the MITM (man-in-the-middle) script at `https://github.com/jrmdev/mitm_relay`. Using this script, implement an attack against Hotmail.com, GMail.com, or similar. You may have to filter out advanced authentication mechanisms, effectively forcing a downgrade to plaintext passwords, using the companion script for mitm shown here for hotmail.com, filtering out the SASL-IR mechanism:

```
def handle_response(server_response):
        """
        This function will be called when a response is received from the server.
        It must return the response to be forwarded to the client (or proxy if specified).
        """

        modified_response = server_response.replace('SASL-IR', '')

        return modified_response
```

- Setup a (fake) email account at Hotmail.com (preferred) or Gmail.com, e.g. csci400-0X-lab2-groupY-2019@hotmail.com. Add this to your IMAP email client (mobile, desktop, etc.).
- You will need to fake a DNS entry for mail.gmail.com, or imap.google.com, or the Hotmail equivalent (e.g. imap-mail.outlook.com), to point to your server hosting the mitm python script. Suppose that your fake server hosting the mitm script is at 192.168.1.1, then the DNS entry for imap.google.com should point to 192.168.1.1. You can do this in the /etc/hosts file, or in the local DNS server you use for your email client. A `dnsmasq` entry would look something like this:

  ```
  address=/imap.gmail.com/192.168.1.1
  address=/imap-mail.outlook.com/192.168.1.1
  ```

- You will need to create fake SSL/TLS site certificates to match the fully-qualified domain names (FQDNs) of the mail server, such as imap.google.com, imap.hotmail.com, or imap-mail.outlook.com. You may need to create a Certification Authority (CA) as well, in order to sign your SSL/TLS site certificates. The generated keys would look something like this (the ones shown here are not functional, so you must generate your own):

  ```
  % openssl x509 -in imap-hotmail.pem -noout -text
  Certificate:
      Data:
          Version: 1 (0x0)
          Serial Number: 1425481389557476122
      Signature Algorithm: sha256WithRSAEncryption
          Issuer: CN=CSCI 400 CA
          Validity
              Not Before: Mar 20 15:28:45 2019 GMT
              Not After : Mar 19 15:28:45 2020 GMT
          Subject: CN=imap-mail.outlook.com
          Subject Public Key Info:
              Public Key Algorithm: rsaEncryption
                  Public-Key: (2048 bit)
                  Modulus:
                      00:b8:1e:3b:c6:c0:f5:5b:2b:4c:db:8c:4a:96:cf:
                      70:b9:77:6f:40:aa:94:8f:be:2c:2c:11:36:63:26:
                      b2:e8:4e:0a:b5:46:c4:f6:8c:07:46:85:25:72:7b:
                      7d:c9:0d:0c:45:ea:9c:64:aa:82:09:04:6c:cd:b4:
                      8f:b5:cc:95:02:94:a5:0f:55:21:29:ee:49:56:5e:
                      65:32:68:6a:c7:87:c8:d4:86:e9:59:ed:5d:cc:f0:
                      46:d4:2f:8a:6e:55:85:22:1f:70:68:c0:84:38:e7:
                      38:7c:02:a5:c8:92:d2:f4:b2:cc:7c:34:79:1a:49:
                      3b:18:88:57:64:6c:3a:5c:d9:c4:3a:c4:6e:ba:0e:
  ```

```
                    a2:ce:69:04:bc:f4:e4:dd:fe:98:1f:65:bf:7c:b3:
                    bf:c5:1a:ac:46:cb:1a:ea:9d:89:af:68:d9:ec:e3:
                    bd:3f:8d:87:c5:0c:d7:7d:d8:42:e3:34:86:72:76:
                    22:17:7e:34:8c:6e:37:82:e8:b2:3b:52:51:c5:1a:
                    4e:69:d2:d5:09:1a:7d:67:dc:6d:5e:57:9b:5a:09:
                    29:c2:7c:cb:f8:19:0c:3b:05:6f:ca:54:30:be:c2:
                    66:98:eb:31:81:69:74:23:9a:05:a9:3e:ff:e8:34:
                    fe:33:17:44:34:20:78:37:c8:d1:de:4c:56:6c:50:
                    20:23
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         57:94:35:76:89:54:51:08:e0:b2:b9:49:f1:38:53:92:f9:ab:
         40:af:4c:b0:0f:ee:25:7b:97:81:f5:62:31:d0:6c:ef:65:60:
         d8:64:71:62:ea:3c:23:f9:57:cf:7d:29:2f:62:27:81:4a:a4:
         07:de:ab:5c:1e:a0:07:24:7f:ac:1e:80:50:2c:cd:47:ee:5a:
         9f:a5:54:9c:05:be:56:0c:5f:ab:0b:d8:64:a9:fa:63:72:1f:
         64:cf:4a:e3:db:03:33:69:e0:5f:ec:05:30:15:21:c8:aa:62:
         fa:d7:a9:30:c5:55:5b:20:7c:a0:e5:32:91:56:e3:5f:b1:90:
         62:40

% openssl x509 -in ca.crt -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1513867478203591475572
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=CSCI 400 CA
        Validity
            Not Before: Nov  4 20:33:53 2016 GMT
            Not After : Nov  2 20:33:53 2026 GMT
        Subject: CN=CSCI 400 CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:ba:96:b4:52:82:9b:06:73:01:69:0c:ca:9a:b4:
                    a9:ce:27:8f:fe:e5:2a:f9:d6:5b:85:2f:05:7d:ad:
                    f9:89:c5:48:b9:0c:a5:9d:d8:fd:a8:19:53:58:8f:
                    f4:41:6a:f8:92:12:59:a2:ab:f3:d8:57:22:ae:10:
                    c1:34:2c:cf:3a:e7:7a:d1:66:57:62:6c:51:80:24:
                    c9:88:19:f0:9b:28:18:d8:a8:4a:8f:4d:d5:54:32:
                    43:b5:7b:68:78:9d:7f:88:4a:4b:ff:be:55:60:4d:
                    3a:4d:d6:02:47:47:68:c6:84:7b:f2:e3:f4:68:72:
                    1f:a2:90:bc:7e:dc:43:b1:23
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                64:C9:3E:39:8E:FF:08:2A:41:23:00:FF:90:C0:40:9C:36:82:D4:04
            X509v3 Authority Key Identifier:
                keyid:64:C9:3E:30:8E:FB:08:2A:41:23:00:FF:90:C0:40:9C:36:82:D4:04
                DirName:/CN=CSCI 400 CA
                serial:D1:F3:CC:C5:A5:FF:41:74

            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Key Usage:
                Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         70:07:05:4e:9b:8a:19:0b:ac:64:1d:e4:b8:f5:53:6d:5e:1c:
         38:db:89:cd:54:a7:d0:66:f9:8c:8c:d6:bd:a7:36:c8:05:f7:
         1e:2a:7a:3f:b9:c0:cc:aa:f4:09:4b:65:49:19:41:4b:ca:43:
         80:64:68:d5:ae:97:02:c2:db:ef:81:d7:7c:fe:e5:4e:f2:df:
         d6:9b:7f:69:1e:6f:b4:33:8c:9c:96:71:7a:54:c8:ba:b0:da:
         ef:53:a4:ea:35:04:00:DD:74:96:f4:e0:80:91:16:64:11:95:
         a7:40:a8:cb:b1:3b:ce:1b:e4:f7:12:77:9d:d7:25:23:53:a5:
         f3:ee
```

- You will need to import those site certificates and CA certificates to the device(s) being deceived, such as your iPhone, iPad, Android phone, laptop, or similar.
- Intercept the traffic between your IMAP email client and the purported email/IMAP server, and show that you managed to capture and read the network traffic between the client and the IMAP server, and in particular reveal the plaintext password for the email account created above. The interception is done with the script itself, which can dump it to stdout or to a log file.
- After your experiment, please make sure to remove these certificates from any production devices, if you chose to add them there.
- Note: You may use the Burp interception technique (described in connection with mitm script) if you wish, but it is not necessary or required.

## 4.4 Web site SSL/TLS certificates

Create a small RSA-based x509 certificate (384 bit modulus, about 116 decimal digits), extract the modulus n, factor n into primes p and q (use openssl to create the certificate).

Example of RSA key generation for 100 bits:

```
% openssl genrsa 100
Generating RSA private key, 100 bit long modulus
.++++++++++++++++++++++++++++
.++++++++++++++++++++++++++++
e is 65537 (0x10001)
-----BEGIN RSA PRIVATE KEY-----
MFMCAQACDQq2QURzwNyohBfTkh0CAwEAAQINAXEDGWYGNgmGaDb+NQIHA2/qhQir
UwIHAx3SrfOOzwIHA1OsTOlwKwIHAswsqPYvxQIHAdEuAUuWxA==
-----END RSA PRIVATE KEY-----
% cat >test.pem
-----BEGIN RSA PRIVATE KEY-----
```

```
MFMCAQACDQq2QURzwNyohBfTkhOCAwEAAQINAXEDGWYGNgmGaDb+NQIHA2/qhQir
UwIHAx3SrfOOzwIHA1OsTOlwKwIHAswsqPYvxQIHAdEuAUuWxA==
-----END RSA PRIVATE KEY-----
^D
% openssl rsa -in test.pem -pubout -out pubkey.pem -text
writing RSA key
% more pubkey.pem
Private-Key: (100 bit)
modulus:
    0a:b6:41:44:73:c0:dc:a8:84:17:d3:92:1d
publicExponent: 65537 (0x10001)
privateExponent:
    01:71:03:19:66:06:36:09:86:68:36:fe:35
prime1: 967477975100243 (0x36fea8508ab53)
prime2: 877215628889807 (0x31dd2adf38ecf)
exponent1: 946869780443179 (0x35d2c4ce9702b)
exponent2: 787442138755013 (0x2cc2ca8f62fc5)
coefficient: 511470497142468 (0x1d12e014b96c4)
-----BEGIN PUBLIC KEY-----
MCgwDQYJKoZIhvcNAQEBBQADFwAwFAINCrZBRHPA3KiEF9OSHQIDAQAB
-----END PUBLIC KEY-----
```

To create the certificate (in some cases, the private key must be ≥ 384 bits for this to work), pick an RSA 384-bit key for all practical purposes.

Example of RSA key generation for 384 bits:

```
% openssl genrsa 384 > test1.pem
% openssl rsa -in test1.pem -pubout -out pubkey.pem -text
writing RSA key
% more pubkey.pem
Private-Key: (384 bit)
modulus:
    00:b1:fa:91:98:f9:48:fb:88:f9:37:41:51:fc:9b:
    f7:26:da:d5:a6:a6:ce:b6:3a:b5:1a:8e:e3:ab:ea:
    a8:6a:63:53:f5:e5:14:d8:35:c8:eb:30:e7:f9:e3:
    4f:ea:96:6f
publicExponent: 65537 (0x10001)
privateExponent:
    70:8c:a8:2e:38:d6:b2:5a:78:5f:3c:eb:7f:f7:91:
    5f:fc:db:47:3c:0d:54:a7:e2:79:01:b9:35:67:81:
    ff:6a:61:81:98:a4:8f:0f:8b:2d:51:44:97:01:4a:
    8f:b8:01
prime1:
    00:dc:59:dd:00:2d:37:e9:3a:a5:7c:f1:68:83:eb:
    77:2f:2c:9d:96:15:35:e7:69:01
prime2:
    00:ce:c5:cd:2e:3e:51:ff:2a:e6:49:bf:08:1f:17:
    e0:58:0b:97:a6:74:ac:6d:0f:6f
exponent1:
    00:c0:64:01:ce:f6:ac:3b:8a:06:15:ca:1d:ac:18:
    fa:0e:09:51:6a:4a:08:af:8d:01
exponent2:
    06:7b:00:91:40:76:c5:6e:8c:c5:26:ed:94:3b:e5:
    56:1d:16:e5:aa:a6:74:cc:95
coefficient:
    67:ab:a6:33:bd:0c:43:85:a0:79:40:22:73:df:9e:
    63:9c:d0:c8:e8:42:82:31:08
-----BEGIN PUBLIC KEY-----
MEwwDQYJKoZIhvcNAQEBBQADOwAwOAIxALH6kZj5SPuI+TdBUfyb9yba1aamzrY6
tRqO46vqqGpjU/XlFNg1yOsw5/njT+qWbwIDAQAB
-----END PUBLIC KEY-----
```

Certificate generation using openssl:

```
% openssl req -new -x509 -nodes -md5 -days 100 -key test1.pem > host.cert
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:New York
Organization Name (eg, company) [Internet Widgits Pty Ltd]:John Jay College of Criminal Justice
Organizational Unit Name (eg, section) []:CSCI400
Common Name (eg, YOUR name) []:Group 1
Email Address []:none@jjay.cuny.edu
```

You will need to provide proper information instead of the generic one above. Give the certificate to other group, then have them extract public key (and its modulus, e.g. using openssl) and factor the modulus:

```
% openssl x509 -modulus -in host.cert
Modulus=B1FA9198F948FB88F9374151FC9BF726DAD5A6A6CEB63AB51A8EE3ABEAA
86A6353F5E514D835C8EB30E7F9E34FEA966F
-----BEGIN CERTIFICATE-----
MIIDEDCCAsqgAwIBAgIJANRHSvpgJ/2OMA0GCSqGSIb3DQEBBAUAMIGZMQswCQYD
VQQGEwJVUzELMAkGA1UECBMCTkoxEDAOBgNVBAcTB0hvYm9rZW4xKDAmBgNVBAoT
H1N0ZXZlbnMgSW5zdGl0dXRlIG9mIFRlY2hub2xvZ3kxDjAMBgNVBAsTBUNTNTc3
MRAwDgYDVQQDEwdHcm91cCAxMR8wHQYJKoZIhvcNAQkBFhBub25lQHN0ZXZlbnMu
ZWR1MB4XDTEwMDkwMzIyNTA1NloXDTEwMTIxMjIyNTA1NlowgZkxCzAJBgNVBAYT
AlVTMQswCQYDVQQIEwJOSjEQMA4GA1UEBxMHSG9ib2tlbjEoMCYGA1UEChMfU3Rl
dmVucyBJbnN0aXR1dGUgb2YgVGVjaG5vbG9neTEOMAwGA1UECxMFQ1M1NzcxEDAO
BgNVBAMTB0dyb3VwIDExHzAdBgkqhkiG9w0BCQEWEG5vbmVAc3RldmVucy5lZHUw
TDANBgkqhkiG9w0BAQEFAAM7ADA4AjEAsfqRmPlI+4j5N0FR/Jv3JtrVpqbOtjq1
Go7jq+qoamNT9eUU2DXI6zDn+eNP6pZvAgMBAAGjggEBMIH+MB0GA1UdDgQWBBTb
KO6DHUt9ffzqn2aVdHFth4LfjDCBzgYDVR0jBIHGMIHDgBTbKO6DHUt9ffzqn2aV
```

```
dHFth4LfjKGBn6SBnDCBmTELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk5KMRAwDgYD
VQQHEwdIb2Jva2VuMSgwJgYDVQQKEx9TdGV2ZW5zIEluc3RpdHV0ZSBvZiBUZWNo
bm9sb2d5MQ4wDAYDVQQLEwVDUzU3NzEQMA4GA1UEAxMHR3JvdXAgMTEfMB0GCSqG
SIb3DQEJARYQbm9uZUBzdGV2ZW5zLmVkdYIJANRHSvpgJ/2OMAwGA1UdEwQFMAMB
Af8wDQYJKoZIhvcNAQEEBQADMQBYlHNXBakDY/aI+7ZDhZPGUB8m0FXMq58I+8vA
zomd4MMs4AbcOQPn/y1BB/pVCS8=
-----END CERTIFICATE-----
```

To factor RSA moduli you can use the tools provided in the lab file. Note that some of the tools have GPU variants that can speed up the search by one order of magnitude. You need to budget for at least 3-4 days (!!!) of running time for this factoring task, depending on your computational resources, to get it right. For example, on a 2.5 GHz Intel Core 2 Quad-core CPU (that's a 2009 vintage CPU) with 8GB RAM, Linux 64-bit, gcc 4.3.4, CUDA 4.x, and an nVidia GTX 470 GPU with 448 cores, expect one attempt (all stages) to take up to 12 hours, as shown below. More recent systems, such as Haswell-based Intel i7 CPUs and later (e.g. Sky Lake, Coffee Lake), can finish this task in 12-36 hours without GPU support using CADO-NFS, depending on CPU speed and the number of CPU cores present and dedicated to the factoring task. Using a more powerful nVidia GPU with the Maxwell, Pascal, Volta, or Turing microarchitectures or later, this can take less than 6 hours. Using a cloud computing setup such as Amazon EC2 (Elastic Computing Cloud), this may take about an hour. You mileage will vary, so plan ahead.

```
Number: test1
N=2739344201407841589396402602284965513153231411558204532769022006705339054
7370424645978920352567060347331661787731567 (116 digits)
Divisors found:
r1=5070050921231669393107366907812846365472704107674059214703 (pp58)
r2=5402991496468780364151462837452123176727335299122731641089 (pp58)
Total time: 11.36 hours.
Factorization parameters were as follows:
name: test1
n: 273934420140784158939640260228496551315323141155820453276902200670533905
47370424645978920352567060347331661787731567
skew: 67593.27
# norm 1.08e+16
c5: 47880
c4: -1112605826
c3: -556562348466331
c2: 5094969536946614890
c1: 1212490460901956687931432
c0: -3623493151674501618805852320
# alpha -6.35
Y1: 2676600482779
Y0: -14174210360748146256331
# Murphy_E 4.99e-10
# M 1859182284716557687382211058188627505365577598492526333509899557
306929266546578831318557911026365887432689150431841
type: gnfs
rlim: 3400000
alim: 3400000
lpbr: 27
```

```
lpba: 27
mfbr: 53
mfba: 53
rlambda: 2.5
alambda: 2.5
qintsize: 100000
Factor base limits: 3400000/3400000
Large primes per side: 3
Large prime bits: 27/27
Sieved algebraic special-q in [0, 0)
Total raw relations: 9782049
Relations: 1042218 relations
Pruned matrix : 581911 x 582136
Polynomial selection time: 2.99 hours.
Total sieving time: 7.53 hours.
Total relation processing time: 0.17 hours.
Matrix solve time: 0.57 hours.
time per square root: 0.10 hours.
Prototype def-par.txt line would be: gnfs,115,5,63,2000,2.6e-05,0.28,250,20,5000
0,3600,3400000,3400000,27,27,53,53,2.5,2.5,100000
total time: 11.36 hours.

OS: Linux-2.6.x-x86_64
processors: 4, speed: 2.50GHz
```

You can verify that the modulus matches the one from the 384-bit host key above, as well as divisors r1 and r2 match prime2 and prime1 respectively above.

Bonus to try:

- Factor the modulus of a 512-bit RSA key. The key in question is:

  ```
  Public-Key: (512 bit)
  modulus:
      00:a3:44:26:e2:c0:2b:79:71:28:f1:25:e7:c9:7e:
      8c:5a:bd:7f:66:f3:24:c7:1d:40:fd:ea:ae:a1:36:
      a4:f8:c2:90:10:15:41:9c:dd:5f:03:bd:32:db:d9:
      d6:d5:2f:19:d0:00:e8:38:b9:bb:8c:21:3a:d4:75:
      50:46:c9:fd:71
  publicExponent: 65537 (0x10001)
  ```

You will most likely need Amazon EC2 credentials for this task. You can get $100 of Amazon AWS credit for free per person. Please inquire with the instructor on how to do so, but please realize that it can take up to 2-3 business working days to setup. By comparison, on a dual Xeon Gold 6132 2.6 GHz with 128GB RAM, this takes about 33 hours with CADO-NFS 3.0b.

## 4.5 Kicking it up a notch: 1024-bit RSA keys

In this section we push the limits of computation and technique. 1024-bit RSA keys are still in common use, despite warnings to gradually upgrade to 2048-bit keys in the long run. Let's see what can be done, given some special conditions.

### 4.5.1 Attacking the RSA modulus from a different angle

Factor the moduli for the following RSA public keys:

```
Public-Key: (1024 bit)
modulus:
    00:d9:57:af:3a:15:5e:15:a8:1f:9f:fc:ef:85:de:
    f8:b9:dc:2d:f8:d0:d4:03:5d:63:fc:6c:ed:a6:38:
    e1:50:07:ca:c3:dd:8d:3f:16:f4:3a:33:a8:1a:18:
    92:86:25:ea:1f:9a:62:9c:1e:6c:49:81:74:8d:68:
    38:15:5e:e4:7a:5f:21:9e:a4:5c:d0:48:0f:20:61:
    58:69:60:cf:aa:08:b4:ef:68:ea:ce:f6:dd:27:f9:
    23:39:51:df:af:73:bc:3b:77:f8:48:3d:52:0a:01:
    61:2f:49:a0:de:94:b3:1d:d0:f4:a5:ae:fb:65:ba:
    04:dd:f3:f4:56:d8:64:5d:d7
publicExponent: 65537 (0x10001)

Public-Key: (1024 bit)
modulus:
    00:a6:8e:a1:94:b9:fd:c8:62:ad:e8:d3:96:f1:b1:
    ed:8d:5b:78:32:a8:5e:00:bb:de:75:4a:53:aa:03:
    30:5a:24:75:f7:82:f7:4f:0a:ef:47:3d:41:99:ae:
    4f:52:04:1e:8f:8d:98:94:b5:c9:dd:be:9d:32:2f:
    60:96:6d:39:73:79:05:4f:3f:76:fc:20:7a:58:61:
    af:95:2e:0a:de:5a:ed:f3:20:d6:f2:0a:8a:3f:22:
    ad:5d:dc:00:d3:31:39:df:a7:59:2d:c0:d7:92:f6:
    d6:79:8e:54:f6:2a:ff:4c:0e:fa:8f:31:60:52:fe:
    0b:ae:35:0b:75:b9:46:7d:71
publicExponent: 65537 (0x10001)
```

So you are sitting there, staring at this assignment to factor a 1024-bit RSA modulus. You have been struggling with or contemplating factoring a 512-bit integer with the help of heavy-duty computing power, and now this? A friend of yours who has taken number theory swings by, looks at your computer screen, and you mumble something in frustration. She tells you something that sounds like "mind your own p's and q's," but you haven't said anything wrong. Or was it that she said "mine your p's and q's?" Confused, yet intrigued, you start searching for an answer...

### 4.5.2 Decrypting SSL/TLS traffic

Having successfully factored the RSA moduli, apply this newfound knowledge and move on to the next task, namely decrypting SSL/TLS traffic based on one of these two 1024-bit RSA private keys. Look inside the traffic to determine which private key is need, i.e. look for the public key matching one of the keys in the previous section. You may have to (re)construct the RSA private key from the two prime factors of the modulus and the given exponent by writing some code or manipulating the correct tools. Using WireShark and the guidelines from the lab file, decrypt the SSL/TLS traffic found in the file `ssldump.pcap`. The SSL/TLS traffic in question is on `tcp port 44330`, which you can narrow down using a filter in WireShark: `tcp port 44330`. There is no other relevant traffic in the pcap file, so the filter should not be necessary.

Supply the prime factors of the RSA moduli and decrypted SSL/TLS traffic (the conversation should be 2671 bytes long), both in text form, as part of the report.

## 5 Word Problems

1. Summarize the attack techniques used by the tools.
2. How would you use the gained knowledge, namely the factored modulus, to attack an encrypted connection in general? Is it localized to a single session where you intercepted the public key in transit? Or does it apply to all encrypted sessions with the server hosting that specific public key, in the past, present, and future?
3. How would you thwart the other groups' efforts, i.e. from attacking your systems using the techniques above?
4. Estimate the largest RSA key modulus you could factor with your available resources (list them) in a week.
5. How would you approach breaking a Diffie-Hellman public key?

## 6 Deliverables

1. A report describing all your findings above.
2. A zip file containing:
    - The source code of any tools you developed and data files, including the particular technique and keys used, if any.
    - Answers to all word problems in Part 5.
3. Submit by the beginning of class, September 11, 2019.

## 7 Grading

Points will be subtracted if any of the pieces of the deliverables are missing or incomplete.