CSE 5473 — Network Security Autumn 2020

Assignment #2

Asymmetric Cryptography *Due Oct 4th, 11:59PM*

1 Diffie-Hellman [15 pts]

Consider a Diffie-Hellman scheme with a common prime q = 11 and a primitive root g = 2.

1. If user A has public key Y_A = 9, what is A's private key XA?
2. If user B has public key Y_B = 3, what is the secret key K shared with A?

Answer

1. 6, because 2^6 mod 11 = 9
2. K= 3^6 mod 11 = 3

2 RSA [20 pts]

1. Construct a table showing an example of the RSA cryptosystem with parameters p = 17, q = 19, and e = 5. The table should have two rows, one for the plaintext M and the other for the ciphertext C. The columns should correspond to integer values in the range [10; 15] for M. Hint: Write a small program or use a spreadsheet.
2. In a public-key system using RSA, you intercept the ciphertex C = 10, sent to a user whose public key is e = 5, n = 35. What is the plaintext M?
3. In a public-key system using RSA, the public key of a certain user is e = 31, n = 3599. What is the plaintext M? Hint: you may use the Unix program `factor`[1].
4. In a public-key system using RSA, the public key of a certain user with public key e; n leaks his private key d. Being lazy, he re-computes a new e and d using the same n. Is this safe? Why or why not?

[1] http://www.gnu.org/software/coreutils/factor

Answer

1. Using RSA, $p = 17$, $q = 19$, and $e = 5$ to encrypt $[10, 15]$, we get the following table by computing $X^5 \mod (17 \times 19)$ for each $X \in [10, 15]$:

Plaintext	ciphertext
10	193
11	197
12	122
13	166
14	29
15	2

2. 35 can only be factored as $p = 5$ and $q = 7$. Because $\varphi(n) = (p-1)(q-1) = 24$, $e = 5$, and $de = 1 \mod \varphi(n)$, we get $d = 5$. Then, the plaintext is $(10^5 \mod 35) = 5$

3. $n = 3599$, it can be factored to 59×61. Then, $p = 59$, $q = 61$, and $\varphi(n) = (p-1) \times (q-1) = 3480$. Because $e = 31$ and $de = 1 \mod 3480$, we get $d = 3031$. Therefore $M = C^{3031} \mod 3599$.

4. This is not secure because the attacker can figure out $\varphi(n)$ from the leaked d and e, then use $\varphi(n)$ to compute the new private key. To compute $\varphi(n)$ from d, e, and n:

   $de = 1 \mod \varphi(n)$, assume $k = \frac{de-1}{\varphi(n)}$; Because $\varphi(n) = (p-1)(q-1)$ and $n = pq$, $\varphi(n)$ is very close to n. Therefore, $\frac{de-1}{n}$ provides a good estimation of k (usually, $k = \frac{de-1}{n} + 1$). Use question 3 as an example, $\frac{de-1}{n} = \frac{31 \times 3031 - 1}{3599} = 26$, while $\frac{de-1}{\varphi(n)} = \frac{31 \times 3031 - 1}{3480} = 27$.

   Moreover, it is also trivial to compute p and q by solving the following two equations, given both n and $\varphi(n)$:

   $$p \times q = n$$
   $$(p-1) \times (q-1) = \varphi(n)$$

3 Key Exchange [20 pts]

Tatebayashi, Matsuzaki, and Newman (TMN) proposed the following protocol, which enables Alice and Bob to establish a shared symmetric key K with the help of a trusted server S. Both Alice and Bob know the server's public key K_S. Alice randomly generates a temporary secret K_A, while Bob randomly generates the new key K to be shared with Alice. The protocol then proceeds as follows:

Alice $\Rightarrow$ Server: $K_S\{K_A\}$
Bob $\Rightarrow$ Server: $K_S\{K\}$
Server $\Rightarrow$ Alice: $K \oplus K_A$
Alice recovers key K as $K_A \oplus (K \oplus K_A)$

To summarize, Alice sends her secret to the server encrypted with the server's public key, while Bob sends the newly generated key, also encrypted with the server's public key. The server XORs the two values together and sends the result to Alice. As a result, both Alice and Bob know K. Suppose that evil Charlie eavesdropped on Bob's message to the server. How can he with the help of his equally evil buddy Don, extract the key K that Alice and Bob are using to protect their communications? Assume that Charlie and Don can engage in the TMN protocol with the server, but they do not know the server's private key.

Answer In this attack, Charlie first eavesdrops the key exchange between Alice and bob:

1. Alice sends $K_S\{K_A\}$ to the server, requesting to establish a secret key with Bob
2. Bob sends $K_S\{K\}$ to the server. This message is eavesdropped by Charlie and saved for later use.
3. The server sends $K_A \oplus K$ to Alice. Alice then recovers the key K by $\oplus\ (K_A \oplus K)$with its own secret key K_A.

Then, Charlie and Don collaborates to reveal the secret key used by Alice and Bob:

4. Don chooses a secret key K_D and sends $K_S\{K_D\}$ to the server, requesting to communicate with Charlie.
5. Charlie sends previously saved $K_S\{K\}$ to the server
6. The server sends $K_D \oplus K$ to Don. By now, Don can recover the key shared by Alice and Bob by $\oplus\ (K_D \oplus K)$ with his own secret key K_D.

4 Performance Comparison: RSA vs. AES [10 Points]

Asymetric cryptography is typically much slower than symetric cryptography. Please prepare a file (`message.txt`) that contains a 16-byte message. Please also generate an 1024-bit RSA public/private key pair. Then, do the following:

1. Encrypt `message.txt` using the public key; save the the output in message `enc.txt`. (Hint: using command `openssl genrsa -des3 -out rsa.key 1024` will generate a public/private key pair stored in file rsa.key, then using `cat message.txt | openssl rsautl -encrypt -inkey rsa.key > message.enc`. If `openssl` is not istalled, please use `sudo apt-get install openssl`)
2. Decrypt message `enc.txt` using the private key. (Hint: `cat message.enc | openssl rsautl -decrypt -inkey rsa.key > message.dec`)
3. Encrypt `message.txt` using a 128-bit AES key. (Hint: command such as `openssl enc -aes-128-cbc -e -in msg.txt -out mes.enc -K 00112233445566778899aabbccddeeff -iv 0102030405060708` will perform this job).
4. Compare the time spent on each of the above operations, and describe your observations. If an operation is too fast, you may want to repeat it for many times, and then take an average.

After you finish the above assignment, you can now use OpenSSL's speed command to do such a bench- marking. Please describe whether your observations are similar to those from the outputs of the speed command. The following command shows examples of using speed to benchmark rsa and aes:

```
% openssl speed rsa
% openssl speed aes
```

Answer

Encryption time for 1024-bit averaged over 1000 runs is 0.00550090456008911 seconds
Decryption time for 1024-bit averaged over 1000 runs is 0.00597498846054077 seconds
Encryption time for 128-bit averaged over 1000 runs is 0.0045092978477478 seconds

openssl speed rsa1024
Doing 1024 bit private rsa's for 10s: 64021 1024 bit private RSA's in 9.99s
Doing 1024 bit public rsa's for 10s: 782170 1024 bit public RSA's in 9.98s
sign verify sign/s verify/s
rsa 1024 bits 0.000156s 0.000013s 6408.5 78373.7

openssl speed aes128
Doing aes-128 cbc for 3s on 16 size blocks: 20047618 aes-128 cbc's in 2.96s
Doing aes-128 cbc for 3s on 64 size blocks: 5951462 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 256 size blocks: 1539531 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 772632 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 8192 size blocks: 109220 aes-128 cbc's in 3.00s
The 'numbers' are in 1000s of bytes per second processed.
type 16 bytes 64 bytes 256 bytes 1024 bytes 8192 bytes
aes-128 cbc 108365.50k 127389.15k 131373.31k 263725.06k 298243.41k

From our trial runs, encrypting RSA-1024 seems to be faster than decrypting by about 0.5 milliseconds. We can also see that AES-128 is about 1.0 milliseconds faster than RSA-1024 when it comes to encrypting.

According to the speed command, it takes RSA-1024 about 0.0000125 seconds to encrypt and about 0.000156 seconds to decrypt. Additionally AES-128 with 16-byte blocks takes about 0.000000147 seconds to encrypt.

The results from the speed command generally reflect our observations; namely that RSA-1024 encrypting is faster than decrypting, and AES-128 encrypting is faster than RSA-1024 encrypting. However, the speed command results show a much greater than disparity than our results, likely due to differences in timing code implementation.

5 Testing Digital Signatures [15 Points]

Let's use OpenSSL to generate digital signatures. Please prepare a file (example.txt) of any size. Please also prepare an RSA public/private key pair, then do the following:

1. Sign the SHA256 hash of example.txt; save the output in example.sha256.

2. Verify the digital signature in example.sha256.

3. Slightly modify example.txt, and verify the digital signature again

Please describe how you did the above operations (e.g., what commands do you use, etc.). Explain your observations. Please also explain why digital signatures are useful.

Answer

```
echo 'Example' > example.txt
openssl genrsa -des3 -out rsa2.key 1024
openssl rsa -in rsa2.key -pubout > rsa2.pub
openssl dgst -sha256 -sign rsa2.key -out example.sha256 example.txt
openssl dgst -verify rsa2.pub -signature example.sha256 example.txt
Verified OK

echo 'example' > example.txt
openssl dgst -verify rsa2.pub -signature example.sha256 example.txt
Verification Failure
```

We saw that signing is performed with the private key, while verification of the signature is performed with the public key. Hashes of messages are signed instead of messages. This is because signing is just encryption with the private key, so encrypting a large message would be undesirable for common use, due to RSA's speed restrictions, and this would necessitate sending another message as large as the original, essentially doubling the amount of data sent. Even a small change of a few bits in the original message causes the verification to fail.

Digital signatures are useful as a means of fulfilling parts of traditional security concepts; namely integrity, authentication, and non-repudiation. Because the message can be sent with a signature, the message can be verified as having originated from the sender. Additionally, because only the sender supposedly has access to the private key, they cannot deny sending the message afterwards. Finally, because the signature is an encrypted hash of the message, the receiver can verify the message was not altered while in transit, as changing the message causes verification to fail with high probability. An attacker cannot intercept the message and recompute the hash because they would still need to encrypt it with the appropriate private key. Of course, this really only verifies someone who has possession of the private key sent the message.

6 Sending emails with public key cryptography [20 Points]

Public Keys is a concept where two keys are involved. One key is a Public Key that can be spread through all sorts of media and may be obtained by anyone. The other key is the Private Key.

This key is secret and cannot be spread. This key is only available to the owner. When the system is well implemented the secret key cannot be derived from the public key. Now the sender will crypt the message with the public key belonging to the receiver. Then decryption will be done with the secret key of the receiver.

In this assignment, you will follow detailed instructions and gain some hands on experience of how to use the public key cryptography. To begin with, please follow the manual of GPG on how to create your own keys. Please keep in mind, using public key cryptography to send message will be useful in your whole life. Understanding how to use it is not of wasting of your time.

Also, there are many example tutorials, such as those:

- `http://www.dewinter.com/gnupg_howto/english/GPGMiniHowto.html` or
- `https://help.ubuntu.com/community/GnuPrivacyGuardHowto`. We recommend this link as it contains detailed instructions on how to create your keys, etc.

To install gnupg, you can use:

```
%sudo apt-get install gnupg
```

Assignment Details. This task has to be solved by pairs. First, please try with your peers, and then test with the TA. Specifically:

- Assume you are Bob, please first find another student (say Alice) in the class and ask her to sign her public key with her private key, with the following message (using the instruction manuals for GPG). "`[Alice.num] has the following public key [PK]`", where [Alice.num] is her OSU ID, and PK is her public key.

  Then, Alice sends the signed message (along with the public key) to Bob, and then Bob verifies this is really from Alice by using the public key in the message. So Bob knows this is really from Alice.

- Next, for every student, please find the public key of the TA in the following link: `https://piazza.com/class/ke7xiqs6nk34l6?cid=73`. Use this public key to encrypt the email message sent to TA (his email address is `ma.1189@buckeyemail.osu.edu`). The email message should have the following Subject line: `[CSE 5473] HW2 <Last Name> <First Name>`, with the content of your public key, and signature of the public key (signed with your private key).

  Then the TA will send you an encrypted message with your public key (with some secret content up to the TA), and you need to write it down what you have observed in all these steps, as well as the decrypted message in your report.

Answer

To show what you need to do for the second part of this question, I created a public/private key pair for a fake person Bill Gates. And you can imagine that is you. Notice that you can use '−−sign' (the message being signed is compressed) or '−−clearsign' (causes the message to be wrapped in an ASCII-armored signature). The first part is trivial when you understand the second part.

For signing and encrypting

```
gpg --armor --export bill.1@osu.edu | gpg --armor --sign -u DBAFA43A699DA9FA532B71EE67724905B0A3EBD5 |
    gpg --armor -e --recipient ma.1189@osu.edu > bill_message.txt
```

Explanation of each step split by pipe '|':

- step1: export your public key.
- step2: sign with your private key. '-u DBAF....' is used to specify the local user (using which private key to sign). You can find the user id with 'gpg –list-keys', but this may not be necessary for you as you may just have your own private key.
- step3: encrypt with TA's public key (specifying with '–recipient ma.1189@osu.edu')
- step4: finally, we output the encrypted signature to a file. Send this file and your public key (gpg –armor –export [your email]) to your TA.

For verifying(decrypting) and decrypting

```
cat bill_message.txt | gpg -d | gpg -d > bill_message.txt
```

Explanation: when your TA received your public key and encrypted signed file, he will import your public key and then check your encrypted and signed file. The result looks like:

```
Sixiangs-MBP% cat bill_message.txt | gpg -d | gpg -d
gpg: encrypted with 3072-bit RSA key, ID 92556084D3F339EC, created 2020-09-25
      "sixiang ma <ma.1189@osu.edu>"
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBF93Y6kBDACr95eHGKfXqvxCzgu9cA7a0BvnsLuppuYALbVhy/oYy5G3+lt0
1AOsWZ4UCX+/t60EtIYf157NbYwJub6JVYx4K03iS7mdkUKrQs71F5U6/ncXtuoH
9z0xxgmae/BzDTSyBSHwfkENn6xWx954r2jPB4TXtmW7FgjcBgmzEbbkKglgDgxH
3+1QkxP50MKVt7nb2AcJV499Bz3wmMqJ5EIH+V0fAKK9vKDhf+CZ1H0TdI4fteXN
N0Ffa2m/GDihlC03NFU9sduqjjF8msH4Mlrk5jyuC3BpRuJ0c9ESYcG2oHiSLleV
YZRbmubDjmYWl4zqoLa+dGVXszvRpQb9O0FmTG83xqu148TPJX6PBbgiiVrvhCKD
b9w36j/DQUbMNTxA9/hv8eMQV9+CboDYGmQR2qymehLY/7I3TH9rlbMJrDtX4xgd
KOlhN+AZgpJEKgyRUstIzb9ZdpzwlQUWKE9rV1la8/M7KdARxBQrGrc3hCbgIKpA
5SOBQjNk3erUNHkAEQEAAbQbQmlsbCBHYXRlcyA8YmlsbC4xQG9zdS5lZHU+iQHU
BBMBCAA+FiEE26+kOmmdqfpTK3HuZ3JJBbCj69UFAl93Y6kCGwMFCQPCZwAFCwkI
BwIGFQoJCAsCBBYCAwECHgECF4AACgkQZ3JJBbCj69UP5Av/UDRCIGaI5yYnlJVo
bKsCrY876a4vpWCk2qIhKoI/jwzRWFjZs1MA9lJxqlK9MW5NJ4W99xcqCZZrdQBP
QLTW6eWXzjIQygb+ea82gs5pfX0dxqzo9uQ/sNgFTZNZPlg5PBU4LjKBN1sFqYaD
GL2h3AVSO21IFXLhmwLwdZZYDNognL1oebCwBVPO4uzX/G31pi08HQSQiaeqEUCu
wt1R+eIwXCChGGOzi4PYlndgFei9niUb3j21nyZhf7PTWnCDhp3hAZjjj8rY/K1W
VIzx0k5diq9RHS87ScBhAkX2hwCJeA8TFCdi0qpolHbI5y5tyX0fouxFzw63RSYq
Lggo/Db/INbUXmfff+LxYm4oq0af6iJNaDcKainFP2XEi9IAYnIkG1zrLkLfNdkt
302Y1BzdnA9teoCMdYxfPhm9QibQwe7rf3lulurbJbiXIvE2bzC0CI5xALfFSVQE
gu4jI4c+IpZnkRPKU+n03wTKfE6eyd2NMK2mGERRsaVdxG+XuQGNBF93Y6kBDADD
+J7bie0RNuRkQcrh8jAJDryEPL8CmJ7oCM09KGLS3IdPg4Az+uH6Fkyd8yhu7t6L
paA/Mmd4irBLNbh+PDC2ZDhoHbtrujLa3EZi0FeHQNz69vthrT9wyMozd2zBAi+F
WWIhPtgJaRHAcyRtuULNPDakF5xfRzPYgUeXS0PrwkVfaeYKbCmJ5JXRryKKHUB4
ezIuSKZQ1NoViZkW9YU1PLIAUQi6IbLDsuzGFAiVcuLub3SrQjDtkpALGm55ngRm
0cvQI/JJopoUnhYeFSUlVZKqputKbZuNL99+WO8qVn36TjoiSdPza1bqtpIGMXon
AFMokjvq68s9PirzZHc+Lbr7Wpna8Q15mBsWNOCX/ThVwDJO5uOFb/3Apkaocf3n
5UVu2p7kEf9+CAbNjFxcRNkKvdmeb2SLUmUL6iArflySZP5BSJjIuS1uAcbwKOuQ
ybptG1SdKpDxb3vXY00BSzVGgKug3mgO8Th59Zb0ZQLFVSEBVbPGGoDRj0dpYQcA
EQEAAYkBvAQYAQgAJhYhBNuvpDppnan6Uytx7mdySQWwo+vVBQJfd2OpAhsMBQkD
wmcAAAoJEGdySQWwo+vVDC8L/3nKhEsbrbi/aRzy7d3D48HsF3P9igT8PnFXQ5HX
blJU+Sa5Aqp7QYTAngGOhHCFr2TT4i/nuzjHZ77ag1DtcS0M0T94KizNiC2Bk1Da
+8iYmTzhL56SdrWYEGWmgIw/J5S2p7Vo4rN0JtvCgr2mqi33D8MKCdVU1+6uc3Ow
a6QuRmmpN+zEDGyYRL1eMP8GdiWkTKf0PLZFJDVbBKKnf1Zw7VuoNip8uQJy7ktG
03waFFmubvJvP1IcJvrMcK1hfr4uv/6Lf6OJsjOiIQLj9x2VfgC8/yzzAxVaUQLT
AmBnGoWO1mjQukBuyhKhzqtcndto5SQUrAXOcV0GPgsEif9qR+g+NJG5HtOWe7x9
QrwF3VUYgPiVRMrL9J9cSgdpuuRABG5OTJ6lUROsNeRse2RujclN8FfJzd2rc/1Y
zkVGO0B0GZJUgrtYSgE2/4LzOY3IBn3Bm2VBh0HTkJ/txdy+/CRlhjDLEswENMtp
UH8NbR4HeGMbHbgQeDo1PxKxMg==
=5Cx1
-----END PGP PUBLIC KEY BLOCK-----
gpg: Signature made Fri Oct  2 14:08:20 2020 EDT
gpg:                using RSA key DBAFA43A699DA9FA532B71EE67724905B0A3EBD5
gpg: Good signature from "Bill Gates <bill.1@osu.edu>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: DBAF A43A 699D A9FA 532B  71EE 6772 4905 B0A3 EBD5
```

7 Submiting your report

Please write a report describing how you solve each of the problem above, and submit at CARMEN.