Problem Set 5–Public key Cryptography

CSCI3381-Cryptography

Due Monday, March 27

This assignment consists exclusively of programming problems. In the first part, you are to implement a variety of attacks on 'textbook' RSA. The second part is devoted to Diffie-Hellman key exchange and ElGamal encryption. There are eight problems; each one is worth twenty points. A 'perfect' score is 100. You will receive extra credit if you do more than five problems.

1 RSA

Each of these problems contains an RSA ciphertext encrypted with a 1024-bit modulus, but in each case it is possible to recover the plaintext by attacking some weakness in the implementation. As we discussed in class, RSA encryption is usually used to encrypt keys for symmetric ciphers, but I do try to make the solutions a little more amusing to read than a random key: RSA decryption gives the plaintext as an integer, but you should convert your solution to an ASCII string; when you do this, Problems 1-4 give readable (even singable!) answers, and Problem 5 is a very short message appropriate for this class.

As you work these problems, think about which of these are vulnerable because of the absence of padding, and which because of poor practice in generating keys. Which of the attacks recover the decryption key, and which recover only the plaintext?

1. *Short message, small exponent* The ciphertext for Problem 1 was obtained by encrypting ASCII plaintext with exponent = 3 and the given modulus. Viewed as an integer, however, the plaintext message m satisfies $m^3 < N$, where N is the modulus. (You have to take an exact cube root, which you should have solved on the previous assignment.)

2. *Small exponent.* A single message was encrypted using three different RSA moduli, all with exponent $e = 3$. The three ciphertexts and moduli are given in the accompanying web page. Find the plaintext. (You have to apply the Chinese Remainder Theorem.)

3. *Common Modulus* Everyone in Alice's office is given an RSA public-private key pair with the same modulus. Alice's encryption exponent is 5, and she sees that Alicia's encryption exponent is 3. Alice intercepts an encrypted message from Roberto to Alicia and is able to decrypt it, and so are you. (Alice can use her own private key to factor the modulus.)

4. *Common Factor* Alex, on the other hand, uses different moduli to generate the RSA keys used by her staff, but admits that she has a 'lucky prime' that she uses to generate all these keys. Eva uses this information, and the moduli of two of the staff members, and decrypts messages sent to each of them. Find the plaintexts. (Eva can easily factor both moduli.)

5. *Small Message.* The plaintext is a very brief ASCII text. You are given the modulus, encryption exponent, and ciphertext. If we view the plaintext as an integer m, we have $m > 10^9$,, so it would take quite a long time to try out every possibility in a slowpoke language like Python. But this integer factors into two integers each less than four million. Find the plaintext. (Use a meet-in-the-middle attack: If c is the ciphertext, and $m = m_1 m_2$ the message, then $cm_1^{-e} \equiv m_2^e \pmod{N}$.)

2 Diffie-Hellman and El Gamal

Let's recall briefly what the fundamental algorithms are. In the Diffie-Hellman key agreement protocol, Alice and Bob agree publicly on a large prime p and a primitive element g mod p. [1] Alice generates a secret value $1 \leq x < p$, and Bob similarly generates $1 \leq y < p$.

Alice sends $g^x \bmod p$ to Bob, and Bob sends $g^y \bmod p$ to Alice. Each of them couples this information they receive with their secret information to compute the shared secret $g^{xy} \bmod p$.

All our problems concern the variant of Diffie-Hellman called the *El Gamal public key cryptosystem.* Here, Alice keeps x as her secret key and publishes $(p, g, g^x \bmod p)$ as her public encryption key. Messages are integers m in the range $1 \leq m < p$. To encrypt this message, Bob generates a random $1 \leq y < p$ and sends

$$(\alpha = g^y \bmod p, \beta = (g^{xy} \cdot m) \bmod p)$$

to Alice. To decrypt, Alice takes the first component α and her secret value x to compute $g^{xy} \bmod p$, then computes its inverse $g^{-xy} \bmod p$, and multiplies this by the second component β of Bob's message to recover m.

In the problems below, the messages are brief ASCII texts that have been converted, as usual, into sequences of bytes and then into long integers. You should convert the decrypted value back into text. All the parameters and messages are given on the accompanying web page.

You need not hand this in, but you should verify that p is prime and that $g = 5$ is a primitive element mod p. To show that g is a primitive element, you should verify that $p = 2q + 1$ for a prime q and verify that $g^q \bmod p \neq 1$ and $g^2 \bmod p \neq 1$ (and be able to explain why this shows that g is primitive.)

6. Alice's public and private parameters and Bob's message pair (α_1, β_2) are given on the Web page. Decrypt Bob's message.

[1] In practice g is often not actually a primitive element mod p, but rather an element such that the cycle of powers of g contains a large prime number of elements. In our present example, $25 = 5^2$ would be such a value for g.

7. Bob sends Alice another pair (α_2, β_2). It is later revealed that the message sent was

```
Now my charms are all o'erthrown and what strength I have's mine own.
```

Subsequently, Eve intercepts yet another message (α_2, β_3), and notices that it has the same first component as the previous message. This is because Bob made the fatal error of reusing his random value y. Decrypt this new message.

8. I did not give you just any old pair (p, g). This is the pair used by what was, until recently, the Discrete Log record holder. The website also contains the value $\alpha_4 = g^y \bmod p$ whose discrete log was successfully computed, and the discrete log y itself. Eve intercepts Bob's communciation (α_4, β_4) to Alice, and uses this priceless discrete log information to decrypt the message.

You are assuming here that the value whose discrete log was known is sent as the first component of Bob's message to Alice. What would be the result if this value were posted as the first component of Alice's public key?

The moral of problem 7, if you didn't catch it, is that Bob cannot reuse his secret value y. The moral of problem 8 is that p must be large enough to beat the best technology for computing discrete logs. In this example, p has 180 decimal digits, roughly 600 bits. For this reason, 1024 bits is considered an appropriate size.

3 What to Hand In

The Python functions you submit should be labeled `problem1, problem2,...problem8` and be contained in a single file. I have also dealt with the fact that the originally posted version of the assignment had no problem 5 and two problem 6's, and corrected this. A short text document giving the plaintext solutions should also be supplied. The Python functions should go into a single file called FirstInitialLastNameHW5.py, and put into a folder FirstInitialLastName along with the text document, then submitted.

Here are the required formats of the functions:

- `problem1` should take two arguments: the base-64 encoded ciphertext and the modulus N. If the attack works, the function should return the plaintext as an ASCII string. If the attack fails (because the message m is too large), it should return some sort of error message. You can include the kthroot function from the previous assignment in your file, and call it in your function.

- `problem2` should take 6 arguments: the three ciphertexts encoded in base-64, and the 3 corresponding moduli, in that order. It should return the ASCII plaintext.

- `problem3` should take three arguments: The base-64 encoded ciphertext, the decryption exponent corresponding to the encryption exponent $e = 5$, and the common modulus. It should return the ASCII plaintext.

- `problem4` takes four arguments: The two base-64 encoded ciphertexts, and the two moduli. You can hard-code the encryption exponent 65537 into your

function. It should return the ASCII plaintext in the event that the attack succeeds (because a common prime factor was found), and some kind of error message in the event of a failure.

- `problem5` takes three arguments: the base-64 encoded ciphertext, the modulus, and the encryption exponent. It should return the ASCII plaintext in the event that the attack succeed, and an error message for a failure.

- `problem6` takes the following 6 arguments:

$$p, g, g^x \bmod p, x, \alpha, \beta,$$

  where the first three components are Alice's public parameters, x is Alice's secret information, and α, β are the components of Bob's message. It should return the ASCII plaintext. This is the function Alice uses to decrypt normal traffic.

  In the next three functions the secret x is absent from the parameter list, because these are the functions the attacker uses to break the encryption on intercepted messages.

- `problem7` takes parameters

$$p, g, g^x \bmod p, \alpha, \beta, m, \beta'.$$

  Here m is an ASCII string, the decryption of the ciphertext (α, β). The function should return the decryption of (α, β').

- `problem8` takes the parameters

$$(p, g, g^x \bmod p, \alpha, \beta, y).$$

  Here y is the discrete log of α. The function returns the ASCII plaintext.