650.445/600.454: Practical Cryptographic Systems October 24, 2018

Assignment 3, Part 1 & 2

Instructor: Matthew Green *Due: 11:59 pm November 12, 2018*

Name: _____

The assignment must be completed individually. You are permitted to use the Internet and any printed references, but your code must be your own!

Please submit the completed assignment via Blackboard.

Problem 1: Implementing the Diffie-Hellman cryptosystem (40 points)

For this problem you will produce a Python or Go program that implements the Diffie-Hellman key exchange protocol at a 1024-bit key strength. This will consist of parameter generation, share exchange, and key derivation. Normally the D-H scheme is an interactive protocol between two parties. However, for this implementation we will use files as our communication channel. As in the previous problems, your implementation should be "from scratch", although you are welcome to re-use any of the code you wrote in the previous parts of this assignment.

Your implementation will consist of three separate programs. The first models Alice's initial message to Bob, and outputs a secret key to be stored for later. The second models Bob's receipt of the message from Alice, and outputs a response message back to Alice. The final program models Alice's receipt of Bob's response. For grading purposes you will hand in the following programs.

`dh-alice1 <filename for message to Bob> <filename to store secret key>`.
Outputs decimal-formatted (p, g, g^a) to Bob, writes (p, g, a) to a second file.

`dh-bob <filename of message from Alice> <filename of message back to Alice>`.
Reads in Alice's message, outputs (g^b) to Alice, prints the shared secret g^{ab}.

`dh-alice2 <filename of message from Bob> <filename to read secret key>`.
Reads in Bob's message and Alice's stored secret, prints the shared secret g^{ab}.

The output of your programs should consist of *decimal formatted* integers separated by commas and parentheses, as in the previous problems. Your shared secret (integers) should be printed to `stdout`. All of the relevant parameters should be generated randomly each time you run the program.

Problem 2: Implementing Elgamal (20 points)

Now that you have implemented Diffie-Hellman, modify your implementation to implement a variant of the Elgamal public-key encryption scheme. This variant will use a hash function (SHA256) to compute a key for the AES-GCM encryption scheme. You may use library

calls to implement both algorithms. Your assignment should consist of the following three programs:

`elg-keygen <filename to store public key> <filename to store secret key>`.
This program should be identical to the `dh-alice1` program.

`elg-encrypt <message text as a string with quotes> <filename of public key> <filename of ciphertext>`.

Reads in the public key (p, g, g^a) produced by `elg-keygen`. Generates b and computes $k = SHA256(g^a \| g^b \| g^{ab})$. Outputs ($g^b, \mathsf{AESGCM}_k(M)$) to a ciphertext file, where the latter value is encoded as a *hexadecimal* string.

`elg-decrypt <filename of ciphertext> <filename to read secret key>`.
Reads in the ciphertext produced by the previous program and a stored secret, prints the recovered message or error.

Note: For compatibility, please encode your input to SHA256 using *decimal* formatted integers separated by a single space character. You should encode your IV for the AESGCM scheme in the first 16 bytes of the output ciphertext.

Problem 3: Breaking Diffie-Hellman (40 points)

For this problem you will implement two discrete logarithm programs that can break Diffie-Hellman at various key strengths. In both cases, the input to the program should be (p, g, h) structured in the same format as the output of your `dh-alice1` program above, although likely using a much smaller prime. Each of these **two** programs should find an integer x such that $g^x \equiv h \mod p$, and should then print x in decimal formatting.

1. A brute-force algorithm that simply tries every possibly x.
2. An implementation of the Pollard-rho or baby-step-giant-step algorithm. Both algorithms can be found in the HAC.

For grading purposes you will hand in the following two programs.

`dl-brute <filename for inputs>`.
On input a file containing decimal-formatted (p, g, h), prints x to standard output.

`dl-efficient <filename for inputs>`.
On input a file containing decimal-formatted (p, g, h), prints x to standard output.

Note: For grading and testing purposes you will *not* run the above programs on large primes! You should alter your code from Problem 1 to generate DH parameters with $|p|$ ranging from 20 to 40 bits. Your final grade will be based on a sequence of tests with parameters we generate, and the efficiency of your code will matter!