CSc 466/566 Computer Security

Assignment 5

Due : See below
Worth 25% (ugrads), 25% (grads)

Christian Collberg
Department of Computer Science, University of Arizona

1. Introduction

In this assignment we'll examine symmetric and public key cryptography. You should work in a team of two students.

- Implement your programs in Java. You should use the packages
  - `java.util.BitSet` and
  - `java.math.BigInteger`
  - `java.security.SecureRandom`
  - `java.util.Base64.Encoder`

  to perform bitwise operations (useful for DES), arithmetic on large integers (useful for RSA), generation of random numbers (useful for key generation), and encoding of binary data.
- You cannot use the rest of the `java.security` package, nor any other code not written by yourselves to implement this program.
- You will be evaluated on the correctness and quality of your code. I.e., the code must be readable and well documented.

NOTE: In this assignment we rely on the honor code: You cannot look at any DES/RSA implementations. I'm sure there must be billions of implementations on the web (in C, perl, Visual Basic, etc.) and in textbooks, but you cannot look at any of them. You may, of course, read about the algorithms themselves — you just can't look at any code.

NOTE: Due dates:

- **Part A: Oct 30**
- **Part B and C, Nov 9**

NOTE: You will be given Java scaffolds for your programs. You can download them from `d2l`.

2. Part A: Symmetric Key Cryptography

Implement the DES algorithm to encrypt/decrypt a file. /40

1. `java DES -h`
   - This should list out all the command line options supported by your program.
2. `java DES -k`:
   - This should generate a DES key, encoded in hex, printed on the command line.
   - Each time this mode is executed, a different key must be generated, i.e., you must extract some entropy from the environment.
   - You should *not* generate a weak key (see `https://en.wikipedia.org/wiki/Weak_key#Weak_keys_in_DES`).
3. `java DES -e <64_bit_key_in_hex> -i <input_file> -o <output_file>`:
   - This should encrypt the file `<input_file>` using `<64_bit_key_in_hex>` and store the encrypted file in `<output_file>`.
   - Each encrypted block should be printed as 16 ascii hex characters, separated by newlines. The last block should be padded appropriately.
   - There is no restriction on the size of the input file.
   - Use CBC mode. Use a cryprographically secure random number generator to create the initialization vector (IV). Prepend this IV to the output (in cleartext).
4. `java DES -d <64_bit_key_in_hex> -i <input_file> -o <output_file>`:
   - This should decrypt the file `<input_file>` using `<64_bit_key_in_hex>` and store the plain text file in `<output_file>`.

NOTE: You can use `java.util.BitSet` or `java.math.BigInteger`, or, for better performance, perform all operations on Java's `long` (using the built-in operations). In the latter case, beware of problems stemming from Java not having `unsigned` integers.

NOTE: We will only test with an ASCII text file as input. You can choose to encode the input file in any way you want.

3. Part B: Public-Key Cryptography

Implement the RSA algorithm to generate key pairs, encrypt a number, and decrypt a number. /40

Your program should support the following operations:

1. `java RSA -h`
   - This should list out all the command line options supported by your program.
2. `java RSA -k -b <bit_size>`:
   - This should generate a public/private key pair, and print these, on standard output, one per line, encoded in hex.
   - The size of the key should be `<bit_size>` bits. You should support at least 1024 bit keys.

- In case `-b` option is not specified, the default bit size should be 1024.
- Each time this mode is executed, different key pairs must be generated, i.e., you must extract some entropy from the environment.

3. `java RSA -e <public_key> -n <modulus> -i <plaintext_value>`:
   - This should encrypt the integer `<plaintext_value>` (encoded in hex) using `<public_key>` and print the result (in hex) to standard output.
   - The size of `<plaintext_value>` is no larger than the modulus.
4. `java RSA -d <private_key> -n <modulus> -i <ciphertext_value>`:
   - This should decrypt the `<ciphertext_value>` using `<private_key>` and print the result to standard output (encoded in hex).

4. Part C: A Hybrid Protocol

Use the code from part A and part B to implement a simple chat program using a hybrid protocol to set up encrypted communication. You will be given the majority of code for this part of the assignment: your job is to integrate your own code into this scaffold. /20

1. `java CHAT -h`
   - This should list out all the command line options supported by your program.
2. `java CHAT --alice -a <private_key_alice> -m <alice_modulus> -b <public_key_bob> -n <bob_modulus> -p port -i ip_address`:
   - This is what Alice would run. She knows Bob's public key and the port and IP address on which to talk to him.
   - She starts up first, and lays down to wait for Bob to come online.
3. `java CHAT --bob -b <private_key_bob> -n <bob_modulus> -a <public_key_alice> -m <alice_modulus> -p port -i ip_address`:
   - This is what Bob would run. He knows Alice's public key and the port and IP address on which to talk to her.
   - When Bob starts up, he initializes the hybrid protocol, by generating a DES key, encrypting it with Alice's public key, and sending her the encrypted package.
   - Alice returns the string "OK" (encrypted) to Bob.
   - Once they are done with this simple handshake, they exchange messages read from the command line, encrypted with DES in CBC mode.