650.445/600.454: Practical Cryptographic Systems September 12, 2016

Assignment 1

Instructor: Matthew Green *Due: 11:59pm, September 19*

Name: ____

The assignment should be completed individually. You are permitted to use the Internet and any printed references.

Please submit the completed assignment via Blackboard.

Problem 1: Warm up: implementing a simple cryptographic specification (30 points)

Implement the following specification for encrypting and decrypting messages. You are free to use any reasonable programming language, although I recommend C, Python or Java. You can obtain an implementation of the AES cipher ("ECB mode") and the SHA1 hash function from a cryptographic library, *e.g.,* the Java crypto libraries, OpenSSL, PyCrypto or BouncyCastle. **You must implement both CBC mode and HMAC yourself.**[1]

Submission and grading: A major purpose of this assignment is to make sure you can submit code that works. Your code must compile on one of the standard graduate lab machines (MSSI, UGrad, or Masters lab). If you are using a language that is not present on these machines, and you feel this is absolutely necessary, you may submit a VM image that contains all necessary tools. You may include a single script entitled **build** (*e.g.,* **build.sh**) that performs all actions necessary to compile your software. *If your code does not meet these requirements you will get a* 0 *on this assignment.*

Interface: Your code must use the following command-line interface:

```
assignment1 <mode> -k <16-byte key in hexadecimal> -i <input file> -o <output
file>
```

Where `<mode>` is one of either `encrypt` or `decrypt`, and the input/output files contain raw binary data.

Notation. We denote concatenation by $||$. By $|M|$ we denote the length of an octet-string M in *bytes.* Recall that AES has a fixed block size of 16 bytes.

Encrypt(k_{enc}, k_{mac}, M). Given a 16-byte secret key k_{enc}, a 16-byte secret key k_{mac}, and a variable-length octet string M, encrypt M as follows:

1. First, apply the HMAC-SHA1 algorithm on input (k_{mac}, M) to obtain a 20-byte MAC tag T.
2. Compute $M' = M||T$.

[1]For a description of the HMAC construction, see FIPS 198 or even Wikipedia.

3. Compute $M'' = M'||PS$ where PS is a padding string computed using the method of PKCS #5 as follows: first let $n = |M'| \ mod \ 16$. Now:
   (a) If $n \neq 0$, then set PS to be a string of $16 - n$ bytes, with each byte set to the value $(16 - n)$.[2]
   (b) If $n = 0$ then set PS to be a string consisting of 16 bytes, where each byte is set to 16 (0x10).
4. Finally, select a random 16-byte Initialization Vector IV and encrypt the padded message M'' using AES-128 in CBC mode under key k_{enc}:

$$C' = \text{AES-CBC-ENC}(k_{enc}, IV, M'')$$

Note: you must implement the CBC mode of operation in your own code, although you are free to use a library implementation of the AES cipher.

The output of the encryption algorithm is the ciphertext $C = (IV||C')$.

Decrypt(k_{enc}, k_{mac}, C). Given a 16-byte key k_{enc}, a 16-byte key k_{mac} and a ciphertext C, decryption is conducted as follows:

1. First, parse $C = (IV||C')$ and decrypt using AES-128 in CBC mode to obtain M'':

$$M'' = \text{AES-CBC-DEC}(k_{enc}, IV, C')$$

Note: you must implement the CBC mode of operation in your own code, although you are free to use a library implementation of the AES cipher.

2. Next, validate that the message padding is correctly structured. Let n be the value of the last byte in M''. Ensure that each of the final n bytes in M'' is equal to the value n.

   If this check fails, output the distinguished error message "INVALID PADDING" and stop. Otherwise, strip the last n bytes from M'' to obtain M'.
3. Parse M' as $M||T$ where T is a 20-byte HMAC-SHA1 tag.
4. Apply the HMAC-SHA1 algorithm on input (k_{mac}, M) to obtain T'. If $T \neq T'$ output the distinguished error message "INVALID MAC" and stop. Otherwise, output the decrypted message M.

[2] For $n = 9$ this would produce the padding string: 0x 07 07 07 07 07 07 07