F14 18-487 Introduction to Computer Security, Network Security, and Applied Cryptography
Homework #2

David Brumley, Peter Chapman, Zachary Weinberg

October 27, 2014

Due: **2:30 pm on November 5, 2014**

Introduction

A few notes on this assignment:

- Submissions will be submitted through Blackboard. You can submit this assignment multiple times before the deadline without penalty. The most recent submission is the only one that will be graded. Late submissions will not be accepted.
- You may look up relevant concepts online, but you may not search for specific solutions or proofs for these problems. Please list any resources you used (except for class materials) in your submission.

Problem 1 (15 pts)

We mentioned in class that if the key for a One-Time Pad is all zeros then the message will be sent completely in the clear. To avoid this, we can imagine modifying the One-Time Pad such that we skip any key that is all zeros and use the next sequence of bits off the pad instead. With this modification, does the One-Time Pad still have perfect secrecy? Prove or Disprove.

Problem 2 (10 pts)

Let $\overline{x}$ denote the bitwise-complement of x (i.e. $\overline{1001} = 0110$). An interesting property of DES is that if $DES_k(m) = c$, then $DES_{\overline{k}}(\overline{m}) = \overline{c}$. How does this property of DES help an attacker who wants to use a brute-force chosen plaintext attack to determine the key? Give a detailed explanation. You can assume that DES is a relatively expensive operation.

Problem 3 (15 pts)

You have a message m, which you intend to encrypt using a stream cipher. Recall that a stream cipher $\mathcal{S}(k, IV)$ produces a *keystream* of unlimited length, which is XORed with the message to produce the ciphertext. To make sure the message is not modified in transit, you compute the CRC of m, and append the checksum to the message before encryption. The message you transmit will be

$$C = IV \parallel (\mathcal{S}(k, IV) \oplus (m \parallel \text{crc}(m)))$$

You will choose a new IV uniformly at random for each message.

i) Is this scheme semantically secure? Explain.

ii) Suppose the message m is composed of two strings, $a \parallel b$, and the adversary knows a. Can they modify the ciphertext so that, when decrypted, it reads $\alpha \parallel b$? If so, how? If not, why is it impossible?

(Hint: $\text{crc}(x) \oplus \text{crc}(y) = \text{crc}(x \oplus y)$.)

Problem 4 (20 pts)

In class, we briefly discussed the formal definition of a secure PRNG. One version of this definition says that a PRNG is secure if and only if there is *no polynomial-time algorithm* that can distinguish a string output by the PRNG from a randomly chosen member of the set of *all possible strings* of the same length. (This is known as Yao's Test.)

i) When we prove something secure, we often make use of *reductions*. A reduction is a proof in which we *assume* that algorithm A is secure, and then we prove that *if algorithm B were* not *secure*, that would imply that algorithm A wasn't secure either, contradicting the initial assumption.

Suppose $f(x)$ is a secure PRNG according to the definitions above. Prove that $g(x) = \text{reverse}(f(x))$ is also a secure PRNG. ($\text{reverse}(x)$ emits the bit reversal of x, e.g. $\text{reverse}(1010) = 0101$.)

ii) Suppose $f(x)$ and $g(x)$ are both secure PRNGs. Is $f(x) \oplus g(x)$ also a secure PRNG? If so, prove it, if not, show why.

Problem 5 (20 pts)

In your latest mission as a secret agent for the `18487` hacking group, you have been assigned the task of breaking the crypto scheme used by the world's most villainous organization: Evil Corp. After some initial investigation, you discover that Evil Corp has an interesting Python script running at `debian.ece.cmu.edu:13706`. The script takes in a new username as a command line argument, and then sends an encrypted message to another Evil Corp server (Server Z) containing the new username as well as a **S**uper **S**ecret **N**umber (which has the format XXX-XX-XXXX where Xs are numbers from 1 to 9).

You find that you can intercept the messages between the Python script and Server Z by sniffing traffic on the network. You can invoke the script using Netcat with `nc debian.ece.cmu.edu 13706`. The script uses a one-time pad for encryption and never reuses or runs out of key material. Additionally it is worth noting that SSNs do not have more than two repeated consecutive digits. Here is the Python source code:

```
while True:
    flag = "XXX-XX-XXXX" # Redacted
    name = raw_input()
    message = ''Authorize %s, My SSN is %s. Repeat %s'' % (name, flag, flag)
    comp_msg = zlib.compress(message)
    ciphertext = one_time_pad(comp_msg)
    send_to_server_z(ciphertext)
    print ciphertext
```

When possible, try to send multiple requests over the same TCP connection, rather than opening a new connection for each request. You can assume that there are no weird race conditions involving the one-time pad and that all messages always successfully make it through the network. Given this setup, is it possible to determine the value of the Super Secret Number (assuming that you have polynomial time bounded computation resources)? If yes, provide the Super Secret Number and the source code for programs you wrote. If not, prove why this is the case.

Problem 6 (20 pts)

Another Python script is running at `debian.ece.cmu.edu:13707`. This script encrypts a secret using AES with a null initialization vector, random key, and 16-byte block. The script sends the encrypted text to the user in a hex format, and attempts to decrypt user input in the same hex format using the same key. Here is the Python source code:

```
key = "".join([struct.pack("B",random.getrandbits(8)) for i in range(16) ])
iv = "\x00"*16
flag = 'flag-XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX-18-487' # Actual Flag Redacted
cipher = AES.new(key, AES.MODE_CBC, iv)
ctext = cipher.encrypt(flag)
print "Hex Cipher Text"
print hexlify(ctext)
while True:
    input = raw_input()
    if len(input) == 0:
        print "No Input"
    if pkcs(unhexilfy(input)):
        if data_integrity(unhexilfy(input)):
            print "Message Received"
        else:
            print "Message Received"
    else:
        print "Padding Failure"
```

Note that the key is newly generated for each session. Given this setup, is it possible to determine the value of the flag (assuming that you have polynomial time bounded computation resources)? If yes, provide the flag and the source code for programs you wrote. If not, prove why this is the case.

Problem 7 (10 pts Extra Credit)

A final Python script is running at `debian.ece.cmu.edu:13708`. This script encrypts a secret using AES with a null initialization vector, random key, and 16-byte block. The script sends the encrypted text to the user in a hex format, and attempts to decrypt user input in the same hex format using the same key. Here is the Python source code:

```
key = "".join([struct.pack("B",random.getrandbits(8)) for i in range(16) ])
iv = "\x00"*16
flag = 'flag_XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX_18-487' # Actual Flag Redacted
cipher = AES.new(key, AES.MODE_CBC, iv)
ctext = cipher.encrypt(flag)
print "Hex Cipher Text"
print hexlify(ctext)
while True:
    input = raw_input()
    if len(input) == 0:
        print "No Input"
    if pkcs(unhexilfy(input)):
        if data_integrity(unhexilfy(input)): # Expensive Operation
            print "Message Received"
        else:
            print "Message Received"
    else:
        print "Message Received"
```

Note that the key is newly generated for each session. Given this setup, is it possible to determine the value of the flag (assuming that you have polynomial time bounded computation resources)? If yes, provide the flag and the source code for programs you wrote. If not, prove why this is the case. You may want to perform your experiments via an on-campus wired network connection. You may want to perform your experiments via an on-campus wired network connection.