F13 18-487 Introduction to Computer Security, Network Security, and Applied Cryptography
Homework #2

David Brumley, Ed Schwartz, Greg Nazario, Jonathan Burket

Revised on October 17, 2013

Due: **2:00 pm on October 21, 2013**

Introduction

A few notes on this assignment:

- Submissions should be uploaded on 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.
- Please submit your homework as a PDF. LaTeX is ideal.
- 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 (15 pts)

Suppose Alice and Bob are sending messages with DES using a shared key k. You are somehow able to recover a single ciphertext/plaintext pair (m, c) such that $DES_k(m) = c$. In an attempt to learn the value of the key, you brute force possible values of k by checking if $DES_{k'}(m) = c$ for all $k' \in K$. What is the approximate probability that you will find only one k' such that $DES_{k'}(m) = c$ and therefore be able to identify k as k'? For simplicity, you may assume that for any key k, DES behaves like a completely random permutation from $M \rightarrow M$.

This question shows a) brute force attack has high probability of finding a unique key, and b) there is a chance it's not right given a single PT/CT pair.

Problem 3 (15 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 4 (15 pts)

You are given a message m with $CRC(m)$. Let the plaintext be $m||CRC(m)$. You decide to encrypt with a $PRNG(IV||k)$, where k is chosen uniformly at random and IV is a 24-bit initialization vector that is incremented once per message (note that k is kept constant).

Encryption is: $E(m, k) = IV||(PRNG(IV||k) \oplus m||crc(m))$

How can this scheme be broken?

Problem 5 (20 pts)

In class, we briefly discussed the formal definition of a secure PRNG. The *next-bit test* says that a PRNG is secure if and only if given the first i bits of PRNG output, no polynomial-time algorithm can predict bit $i + 1$ with probability greater than $0.5 + \epsilon$.

This test is equivalent to a slightly more general formulation that is similar to the semantic security test. *Yao's Test* says that a PRNG that produces strings of length n is secure if and only if the ensemble of uniformly distributed of length n strings U_n is *computationally indistinguishable* from the ensemble of strings X_n produced by the PRNG. Two ensembles X_n and Y_n are considered *computationally indistinguishable* if for an arbitrarily selected x from X_n and y from Y_n: $P|[A(x) = 1] - P[A(y) = 1]| \leq \epsilon(n)$ for any polynomial-time algorithm A. Basically, this is saying that no polynomial time statistical test (number of 1s vs. 0s, repeated patterns, etc.) can differentiate between the results of the PRNG and the set of all possible strings of the same length.

One of the cooler things about secure Pseudorandom Number Generators is that they can be composed (i.e. if $f(x)$ and $g(x)$ are secure PRNGs, then so is $f(g(x))$). Consider these other PRNG transformations and prove whether or not they produce secure PRNGs.

(a) $reverse(f(x))$, for arbitrary secure PRNG $f(x)$, where $reverse$ returns the input sequence backwards

(b) $f(x) \oplus g(x)$, for arbitrary secure PRNGs $f(x)$ and $g(x)$

Problem 6 (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 `plaid.cylab.cmu.edu:18487`. The script takes in a new username as a command line argument, 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 also find that you can intercept the messages between the Python script and Server Z by sniffing traffic on the network. You can also invoke the script by using `nc plaid.cylab.cmu.edu 18487`. The script uses a one-time pad for encryption and never reuses or runs out of key material. Here is the Python source code:

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

Do not send more than 10 requests per second to the script. 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 any tools you used to get it (i.e. source code for programs you wrote). If not, prove why this is the case.