CS 161 Computer Security
Spring 2010 Paxson/Wagner
HW 3

Due Thursday March 11, 5:00pm

Instructions: Submit your solution by Thursday, March 11, 5:00pm *electronically*. Write up your answers in either PDF format or plain text (7-bit ASCII), in a separate file for each question. Your files should be named `q1.txt`, `q2.txt`, `q3.txt`, ..., `q6.txt` (for plain text), or `q1.pdf`, ..., `q6.pdf` (for PDF). You may mix-and-match text and PDF if you like. Also, include a file called `collaborators.txt`, containing a list of everyone you collaborated with (see below). Then run `submit hw3`.

You may collaborate with at most 3 other CS161 students on this homework. Collaboration may involve discussing the problems with other students, but must *not* involve sharing solutions or having access to anyone else's solutions *at any time*. You must develop and write up your solutions *entirely on your own*. You must never read or copy the solutions of other students, and you must not share your own solutions with other students.

At the beginning of each file, include your name, class account (e.g., cs161-xy), and the *number* of the discussion section where you want to pick up your graded homework (e.g., 102; not your TA's name or the time). We will deduct points if this is missing for any of the files for your answers to Q1–Q5 or if you list the wrong class account. (You don't need to include this for `collaborators.txt`.)

1. (20 pts.) One-Time Pads

To communicate using a one-time pad, Alice and Bob first need to agree on a secret, random n-bit key $K \in \{0,1\}^n$ (e.g., established at an in-person meeting). Then given the shared secret K, Alice can later encrypt an n-bit message $M \in \{0,1\}^n$ as follows: $C = M \oplus K$. Bob can recover the message from this ciphertext via the equation $M = C \oplus K$.

Having just learned about the one-time pad, Alice is excited about the security it offers. However, she is unhappy with the inconvenience of establishing a shared key with Bob. The following protocol occurs to her.

> When Alice is ready to send her message M, she randomly selects $R \in \{0,1\}^n$ and sends Bob $C = M \oplus R$. Bob then randomly selects $S \in \{0,1\}^n$ and sends Alice $D = C \oplus S$. Next, Alice computes $E = D \oplus R$ and sends it to Bob. Bob may then compute the message as $E \oplus S = M$.

This idea seems similar to the one-time pad, but does not require prior distribution of a shared key. Note that new random values R, S are chosen for each invocation of this protocol (i.e., for each new message Alice wants to send)—they are not reused.

(a) Is Alice's protocol secure?

(b) If your answer is yes, briefly specify your attack model and state why you think the scheme is secure under that model. If your answer is no, give an attack that breaks the protocol.

2. (20 pts.) A Lousy Mode of Operation

Prof. Puce proposes the following mode of operation:

> Given a message M, break it into 128-bit blocks $M[1], M[2], \ldots, M[n]$. Let $M[0]$ be a random 128-bit string (the initialization vector). Now compute $C[i] = \text{AES}_K(M[i-1]) \oplus M[i]$ for $i = 1, 2, \ldots, n$. Also set $C[0] = M[0]$. The ciphertext is $C = C[0] \cdot C[1] \cdot C[2] \cdots C[n]$.

Unfortunately, this mode turns out to be insecure against chosen-plaintext attack.

Suppose Eve has observed a ciphertext $C = C[0] \cdot C[1] \cdot C[2] \cdots C[n]$, which is the encryption of some unknown message M which Eve would like to recover. Assume Eve has the ability to request the encryption of plaintexts of her choice and observe the result. Explain how Eve can learn the entire message M.

3. (20 pts.) The Role of Four Primitives

Four primary types of cryptographic primitives are symmetric encryption, asymmetric encryption, message authentication codes (MACs), and digital signatures. In this question we will compare their suitability in an example scenario. For this question, assume all four have the same computational costs.

Suppose you are a malware author writing bot software to power a new botnet. You will spread the malware by exploiting various vulnerabilities, either manually or automatically (e.g., as a "worm"). After the botnet is in place, you will want to send commands to the bots and also retrieve information from the infected machines. To retrieve information anonymously, you have programmed the bots to post it to a public message board where you can later download it.

(a) Suppose you wish to ensure that the bots will only respond to commands that you have sent and will ignore any other commands. Which two of the four primitives would be helpful in accomplishing this?

(b) Of those two, which would you use? Describe the primary reason why that choice would be better than the alternative.

(c) Now suppose that you are trying to ensure that only you can read information the bots have posted on the message board and that anyone else will find it unintelligible. Which two of the four primitives would be helpful in accomplishing this? Of those two, which would you use? Describe the primary reason why that choice would be better than the alternative.

4. (20 pts.) Let's Make Some Money

(a) BankOBits, the hapless bank from the midterm, uses a one-time pad to secure communication between its ATMs and the bank's central server. Each message from the central server is 24 bits long. The first 8 bits are a code indicating the action the ATM should take:

- `0x45` (binary 01000101, ASCII '`E`') means that the ATM should eject the user's ATM card and ignore the next 16 bits.
- `0x4B` (binary 01001011, ASCII '`K`') means that the ATM should keep the user's ATM card, ask the user to call phone support, and ignore the next 16 bits.
- `0x44` (01000100, '`D`') means that the ATM should dispense a number of dollars specified in the next 16 bits.

The ATM and server are loaded with a huge supply of one-time pad key material, and they never reuse any key material: each 24-bit message is encrypted by xor-ing it with the next 24 bits of key material that has not been used yet. In other words, if M is the 24-bit message from the server and Z is the next 24 bits of unused key material, the server computes $C = M \oplus Z$ and sends C to the ATM.

You can assume that under normal conditions the message that the central server will send to the ATM are predictable. For instance, if the user withdraws $200 and there is no problem with the transaction, the messages sent by the server will be `0x44 0x00 0xC8` (the instruction to dispense $200) followed by `0x45 0x00 0x00` (the instruction to eject the user's ATM card). Whenever the last 16 bits are going to be ignored by the ATM, the server fills them with all-zeros. Of course, all of these messages are encrypted with the one-time pad before being sent over the communication link from the server to the ATM, as described above.

An enterprising criminal manages to splice into the communication line between the ATM and the bank's central server. The splice allows them to not only to observe the traffic but also act as a man-in-the-middle. Explain how the criminal could make lots of money by tampering with the encrypted messages from the server to the ATM.

(b) In class, David claimed one can prove that the one-time pad is secure as long as the key material is never reused. Yet in this example the criminal was able to defraud the bank even though the key material was never reused. What gives? Explain the resolution to this paradox.

(c) BankOBits is considering replacing their one-time pad with a stream cipher, such as AES in Counter mode[1]. How will your attack from (a) have to change to defeat this alternative scheme?

(d) The bank learns of this flaw in its communication protocol, and hires you to fix its protocol. Of all of the cryptographic primitives you saw this week, which one do you think would be most appropriate for protecting messages from the server to the ATM against this kind of attack? Why?

5. (20 pts.) Break Majordomo

(a) In this problem, you're going to break the cookie generation algorithm used by Majordomo 1.94.4, a widely used open source package for mailing list management. To sign up for a mailing list, the user provides their email address to the Majordomo software. Majordomo then sends a short confirmation email to that email address, and includes a 32-bit pseudorandom "cookie" in the confirmation email. If the user replies and includes the 32-bit pseudorandom cookie in their response, Majordomo infers that someone who can read email at that address consents to receive email from that mailing list, so Majordomo signs them up to the mailing list at that time.

This mechanism is intended to prevent Attila from signing Vicky up to mailing lists without her consent (which would be a pretty nasty denial-of-service attack on Vicky, if Attila signed her up for a few hundred mailing lists).

The security of this scheme relies upon the unpredictability of Majordomo's cookies. Unfortunately, the algorithm that Majordomo used for cookie generation was flawed. The algorithm worked as follows: there is a secret key that is specific to each Majordomo installation. This key is a string K. Majordomo forms the string

$$S = K \text{ / } \texttt{subscribe} \text{ / } L \text{ / } M$$

where the string L is the email address of the mailing list and the string M is the email address of the user to be added to the mailing list. In other words, Majordomo concatenates K, "subscribe", L, and M, separated by slashes. Suppose S is n characters long, so $S = S_1 \cdots S_n$. The cookie is computed using the following algorithm:

[1] Each time the server has a 24-bit message to send to the ATM, it takes the next 24 bits from the Counter mode stream $Z = \text{AES}_K(0) \cdot \text{AES}_K(1) \cdot \text{AES}_K(2) \cdots$ and xors it against the message, to obtain the ciphertext that is sent to the ATM. The server never re-uses any part of the Counter mode stream.

1. Set $c := 0$.
2. For $i := 1, \ldots, n$:
3. Set $c = c \oplus S_i$ (treating S_i as a 8-bit byte, in ASCII).
4. Set $c = c \lll 4$ (rotate the 32-bit value c left by 4 bit positions).
5. Output c, in hex.

Here $x \lll 4$ denotes the result of rotating the 32-bit value x leftwards by 4 bit positions, i.e., if the binary representation of x is $x_{31} \ldots x_1 x_0$, then the binary representation of $x \lll 4$ is $x_{27} \ldots x_1 x_0 x_{31} x_{30} x_{29} x_{28}$. In C, if x is an unsigned 32-bit int, then $x \lll 4$ can be computed as follows:

```
(x<<4) | ((x>>28)&0xF)
```

Xavier, a fellow CS161 student, wants to subscribe David to the `cryptography@metzdowd.com` mailing list without his consent, because he figures David might enjoy it. Xavier first signs up to the list himself, to probe Majordomo: Xavier sends Majordomo a request to add his own email address, `cs161-xy@imail.eecs.berkeley.edu`, to the `cryptography@metzdowd.com` list. Xavier receives a confirmation email from Majordomo with the cookie `a07a0fbf`, and Xavier responds to confirm his subscription to the mailing list.

Now Xavier wants to somehow fool Majordomo into adding `daw@cs.berkeley.edu` to the mailing list. Xavier is going to send Majordomo a request to add `daw@cs.berkeley.edu` to the mailing list. When Xavier does that, Majordomo will send David a confirmation email containing some cookie. Xavier wants to forge a response to that confirmation email so that David will be added to the mailing list, but he's not sure what cookie to include in his forged response. As he has not yet figured out how to hack David's email account, he's going to need to predict what cookie will be sent to David.

Help Xavier out. Use your knowledge of the Majordomo algorithm and the information above to predict the cookie that will be sent to David. Include the cookie value (in hex) in your answer, and describe how you got it.

HINT: You might want to write a small program to help you out.

HINT: Here is a test case that you can use to test your small program, if you want.

Seed (K): `test`
List (L): `cryptography@metzdowd.com`
User (M): `cs161-xy@imail.eecs.berkeley.edu`
Cookie: `a174139d`

Seed (K): `test`
List (L): `cryptography@metzdowd.com`
User (M): `daw@cs.berkeley.edu`
Cookie: `a703ca03`

The code I used to generate these test cases is in `~cs161/hw3-files/` on instructional machines.

(b) How could you fix the Majordomo cookie generation algorithm, using the cryptographic techniques you learned about this week? Which of the primitives you learned about would be most appropriate for securing Majordomo's cookie generation algorithm?

NOTE: We are looking for a drop-in replacement for Majordomo's cookie generation algorithm, not for changes to the overall protocol for signing up to mailing lists. You can assume that Majordomo can generate a cryptographic key when it is installed and save that cryptographic key somewhere for its future use. Your fix should not require any changes to the experience that ordinary users see (apart from the fact that users might receive a different cookie value).

(c) How could you fix the flaw in part (a), this time without using any cryptography? What advantage does cryptography provide over your crypto-less solution?

6. (0 pts.) Optional: any feedback?

Optionally, feel free to include feedback in a file called `q6.txt`. What's the single thing we could to make the class better? Or, what did you find most difficult or confusing from lectures or the rest of class, and what would you like to see explained better?

Your answers will not affect your grade. Feel free to be frank: we appreciate all feedback, even (especially) critical feedback.