CS 161 Computer Security
Spring 2010 Paxson/Wagner
HW 3 Soln

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.

Solution:

(a) No, Alice's protocol is not secure.

(b) If a passive eavesdropper witnesses C, D, and E, s/he can then compute $C \oplus D \oplus E$ to find M. Note:

$$C \oplus D \oplus E = (M \oplus R) \oplus (M \oplus R \oplus S) \oplus (M \oplus R \oplus S \oplus R) = M.$$

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.

Solution: Let's start with the main trick. If Eve has any 128-bit value x (of her choice), the main trick lets her learn $\text{AES}_K(x)$. To do this, Eve should request the encryption of the two-block message $M' = x \cdot z$, where z is a block of zeros. In other words, $M'[1] = x$ and $M'[2] =$ all zeros. The encryption function will return the three-block ciphertext $C' = C'[0] \cdot C'[1]' \cdot C'[2]$. Because of the way this mode of operation is defined, we find:

$$\begin{aligned} C'[0] &= IV' && \text{(a random, new IV)} \\ C'[1] &= \text{AES}_K(IV') \oplus x \\ C'[2] &= \text{AES}_K(x) \oplus z = \text{AES}_K(x) \end{aligned}$$

So just by looking at $C'[2]$ (the last block of the encryption of $x \cdot z$), Eve learns the value $\text{AES}_K(x)$. This trick allows Eve to compute $\text{AES}_K(x)$ for any value x of her choosing, at the price of a single chosen-plaintext query.

Now let's see how Eve can use this trick to decrypt the ciphertext $C = C[0] \cdot C[1] \cdot C[2] \cdots C[n]$. Eve knows $M[0]$, as $M[0] = C[0]$. Now for $i = 1, 2, \ldots, n$, we have:

$$M[i] = \text{AES}_K(M[i-1]) \oplus C[i].$$

First, Eve can apply the trick above with $x = M[0]$ to learn $\text{AES}_K(M[0])$, and this lets her recover $M[1]$ via the equation $M[1] = \text{AES}_K(M[0]) \oplus C[1]$. Once Eve knows $M[1]$, she can apply the same idea (with $x = M[1]$) to learn $M[2]$. And so on.

In general, Eve does the following, for $i = 1, 2, \ldots, n$: she applies the trick with $x = M[i-1]$ to learn $\text{AES}_K(M[i-1])$, which then lets her recover $M[i]$ via the equation $M[i] = \text{AES}_K(M[i-1]) \oplus C[i]$, and now she is ready to move on to the next iteration of the loop. After n chosen plaintext queries, Eve learns the entire message $M = M[1] \cdot M[2] \cdot M[3] \cdots M[n]$.

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?

Solution: Digital signatures and MACs.

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

Solution: Digital signatures. MACs would not be a good idea because anyone who inspected or reverse-engineered the bot's code could get the MAC key and use it send commands to the bots. With a digital signature, the commands will be signed with a private key that only the attacker has. Having the public key (from the bot code) would not help a third party trying to send their own commands.

(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.

Solution: Symmetric and asymmetric encryption. Asymmetric encryption would be better to use. Again, if symmetric key encryption were used, the bot's code would have the shared key, and anyone who disassembled the code could get the secret key and read the information. With asymmetric encryption, the bots only have the public key. Only the attacker has the private key, so only the attacker can read the message.

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.

Solution: First, the criminal should make the server send a message to the ATM. Probably the simplest way would be to put in an ATM card, then press "cancel" so the server will send the "eject card" message. The criminal should intercept this message and XOR it with `0x01ffff` before forwarding it on to the ATM. This will change the "E" to "D" and set the next 16 bits to all 1's causing the ATM to dispense the maximum amount: $65,535.

Of course, changing any other message to a "D" message (or increasing the amount of an existing "D" message) will work as well. Provided the criminal knows what message to expect from the server, they can XOR it with an appropriate value to change it to any desired message.

(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.

Solution: Encryption only ensures confidentiality; it does not protect the integrity of messages. The one-time pad does indeed perfectly hide the content of the messages from the server (although someone

using the ATM can predict what they will be anyway), but it does not prevent the messages from being modified.

(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?

Solution: The attack does not need to change at all. The exact same procedure will yield the same results.

(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?

Solution: The server should authenticate the messages using either a MAC (message authentication code) or digital signature, and the ATM should ignore any messages that do not have a valid MAC or signature. While either would work, digital signatures have the advantage that someone who disassembles the ATM and gets the verification key will not be able to use it to generate signed messages. Of course, once they have disassembled the ATM, they could just go straight for the money rather than the key—but they will only get the money in that one ATM, whereas someone who get access to the key may be able to defraud many ATMs.

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. 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.

[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.

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.

Solution 1: The cookie that will be sent to David is `46c1ea13`. Here's how Xavier can determine this. When he receives the cookie `a07a0fbf`, he can run the algorithm in reverse to determine its state before his email address was incorporated into the cookie value, then continue the algorithm (forward) from that point using David's email address.

Specifically, let R be Xavier's email address (`cs161-xy@imail.eecs.berkeley.edu`), T be David's email address (`daw@cs.berkeley.edu`), and n_1 and n_2 be their respective lengths. Then do the following:

1. Set $c :=$ `a07a0fbf`.
2. For $i := n_1, \ldots, 1$:
3. Set $c = c \ggg 4$.
4. Set $c = c \oplus R_i$.
5. For $i := 1, \ldots, n_2$:
6. Set $c = c \oplus T_i$.
7. Set $c = c \lll 4$.
8. Output c, in hex.

Solution 2: Another way to predict the cookie value is to run a brute force attack by trying all 2^{32} values that c could assume after the algorithm has processed the part of the string that holds the key K. For each trial, run the algorithm forward over the rest of the test string (with Xavier's email address) and check to see whether you get the cookie that Xavier observed. Once you have found the correct initial value of c, you can run the algorithm again with David's email address to predict the new cookie. This a slightly less elegant solution, but just as effective.

Solution 3: Let C denote the value of c immediately after processing the part of S that holds the key K. Each bit of the final cookie value can be written as a linear function of the bits of L, M, and C. So, from the 32-bit cookie that Xavier observed, we obtain 32 linear equations. Each linear equation is taken modulo 2. There are 32 unknowns, namely, the bits of C. Given these 32 equations in 32 unknowns, we can solve for C. Then, once C is known, we can run the algorithm forward to learn the value of the cookie that will be sent to David.

(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).

Solution: Replace the cookie generation algorithm with a MAC, that is, set

$$S = \texttt{subscribe} \,/\, L \,/\, M$$

and compute $c = \mathrm{MAC}_K(S)$.

(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?

Solution: Here is one way to do it. Each time a cookie is needed, the server can pick a new random 32-bit value for that cookie, and store it in a table associated with the request. When the server receives a confirmation, it checks that the accompanying cookie matches the value stored for that request. The server will need to store this table somewhere—for instance, in a file on the filesystem.

An advantage of the solution employing cryptography is that it allows the server to be stateless. This might help prevent denial-of-service attacks. (In the solution without cryptography, someone might be able to exhaust the storage of the server by making many requests.)