CSci 5271: Introduction to Computer Security

Exercise Set 3 **due: November 24th, 2021**

Ground Rules. You may complete these exercises in a group of up to three students. Each group should turn in **one** set of answers, designating all group members using the group submission feature of Gradescope. You may use any source you can find to help with this assignment but you **must** reference any source you use besides the lecture notes or assigned readings. Submit an electronic copy of your solution online by 11:59pm on Wednesday, November 24th.

For this assignment, we will use Gradescope's template-based submission style. We will supply a template document with spaces to insert your answers in, in your choice of LaTeX or Google Docs formats. Prepare your answers by filling in this document and converting it to a PDF in a way that preserves the template formatting, and submit this to Gradescope. Note that we still ask that you type your written answers, rather than hand-writing and scanning them, since it is best for both you and us if your answers are easy to read.

1. Caesar's block cipher. (30 pts) The Caesar cipher is a historical encryption method based on advancing letters circularly through the alphabet. To discuss it in a modern context on ASCII, we can consider it to be a block cipher with an 8-bit block and a 5-bit key k. The encryption function E_k is defined as:

$$E_k(b) = \begin{cases} \texttt{0x41} + ((b - \texttt{0x41} + k) \bmod 26) & \text{if } \texttt{0x41} \le b \le \texttt{0x5A} \\ \texttt{0x61} + ((b - \texttt{0x61} + k) \bmod 26) & \text{if } \texttt{0x61} \le b \le \texttt{0x7A} \\ b, & \text{otherwise} \end{cases}$$

Recall that `0x41` through `0x5A` are the ASCII codes for `A` through `Z`, and similarly `0x61` through `0x7A` are `a` through `z`. The inverse operation is just shifting by the same amount in the opposite direction, so $D_k = E_{26-k}$ (we use the convention that the result of mod is always positive when the modulus is). ROT-13 corresponds to the special case $E_{13} = D_{13}$.

(a) CaesarCrypt S.p.A. is an Italian computer security company which got started building on their national heritage to market modern block ciphers that also have an 8-bit block size, but they have taken the lesson that the original Caesar cipher had too small a key size. Their first flagship product CCEA1 was a 8-bit block cipher with a 2048-bit key size. Their new successor cryptosystem, named CCEA2, increases CCEA1's key size to 4096 bits. CaesarCrypt's marketing materials suggest that this yields an astronomical increase in security by a factor of 2^{2048}. What do you think of this security claim: can CCEA2 really be more secure than CCEA1?

(b) In fact there are some general problems that affect any block cipher with a small block size. Describe a chosen-plaintext attack that would easily break any block cipher with an 8-bit block size.

After learning about the problems you found above, CaesarCrypt's next design is a block cipher named CCEA3 with a 16-byte (128-bit) block size, made by applying the function E_k separately to each of the 16 bytes. Each byte position has a different rotation amount. In other words the key is 80 bits divided as 16 5-bit fields $K = (k_0, \ldots, k_{15})$. If a 128-bit plaintext block is split into 16 1-byte values as $P = (p_0, \ldots, p_{15})$, the corresponding ciphertext block C is computed as $(E_{k_0}(p_0), \ldots, E_{k_{15}}(p_{15}))$.

In the next few parts, we'll explore using this block cipher with some standard modes of operation. We want to encrypt the message "GALLIA EST OMNIS DIVISA IN PARTES TRES.". After splitting the ASCII encoding of the text into 16-byte blocks and adding padding to the last block, this corresponds to the following 3 plaintext blocks in hexadecimal:

```
47 41 4c 4c  49 41 20 45   53 54 20 4f  4d 4e 49 53
20 44 49 56  49 53 41 20   49 4e 20 50  41 52 54 45
53 20 54 52  45 53 2e 09   09 09 09 09  09 09 09 09
```

The key we'll use, consisting of 16 rotation amounts in decimal, is:

```
6,19,1,6, 17,19,25,3,  17,25,25,18, 19,14,24,5
```

For modes of operation that use an IV, we'll use:

```
78 44 4f a1  76 57 70 e5   47 54 72 52  4d 6e 49 71
```

This message is of the length where it would take comparable amounts of effort to do to the encryption by hand, or to write computer code to do it. To support either approach, linked next to the assignment on the course web page are some framework C code you can build on, or a PDF you can print with tables to more easily compute XORs and alphabet rotations in hex.

In each of the next four parts, give the message ciphertext in hexadecimal bytes under each of the following modes:

(c) ECB mode

(d) CTR mode, where the initial counter value is the IV, and a block is incremented as if it were a 128-bit big-endian number

(e) CBC mode

(f) OFB mode

(g) Unfortunately, despite the more reasonable block size, CCEA3 is still not a secure encryption algorithm. Explain why, by demonstrating either a specific theoretical property that a secure block cipher would have, which CCEA3 does not, or by explaining a practical attack.

2. (Mis-)using message authentication codes. (26 pts) Armed with a copy of Schneier's *Applied Cryptography* from a used bookstore, Sly can't wait to design his own encrypted thinga-madoodad protocol. He starts off with a super-secure key exchange protocol that ends with Alice and Bob sharing secret keys for encryption (K_e) and authentication (K_a). Now he wants to design a secure symmetric channel using these keys.

(a) Sly decides at first that he wants to use a CBC-MAC based on AES with 128 bit blocks for integrity. He looks carefully at his key exchange protocol and realizes that an adversary can interfere to make Alice and Bob end up deciding on different keys. So the first message sent over by Alice will be $\tau_0 = \mathsf{cbcMAC}_{K_a}(0^{128}) = \mathsf{aesEncrypt}_{K_a}(0^{128})$. (The notation 0^n means n zero bits.) If Bob's local value doesn't check out, he aborts, otherwise the channel is usable. Afterwards, whenever Alice wants to send the message M over the secure channel, she'll compute $\tau_M \leftarrow \mathsf{cbcMAC}_{K_a}(M)$ and send the pair (M, τ_M) over the channel; Bob will check whether $\tau_M = \mathsf{cbcMAC}_{K_a}(M)$ and if so will conclude that Alice said M.

This is a pretty bad idea. Show how to use the values τ_0, M and τ_M to compose a message to Bob that will convince him Alice meant to say the two-block message (M, τ_M) instead of just M. Explain why your message will convince Bob that Alice meant to say (M, τ_M) rather than just M. Hint: try writing a recursive definition of CBC-MAC, and use the facts that for any string A, $A \oplus A = 0^{|A|}$ and $A \oplus 0^{|A|} = A$.

Since τ_M is just 128 random-looking bits, why is this a big deal?

(b) Sly's friend Sally notices the same attack on his scheme. She proposes a different method of authenticating (and encrypting) messages: ignore the key K_a. Instead, to authenticate and encrypt the message M, first compute $H(M)$ using SHA-256; then encrypt $(M, H(M))$ together, using AES-CTR encryption. So the message sent on the insecure channel would be $\mathsf{CTR\text{-}Encrypt}_{K_e}(M, H(M))$; Bob would decrypt the message using K_e, check that the last 256 bits of the plaintext are the hash of all of the previous bits, and accept the message if they are.

Show that this is also a bad idea: if Alice ever sends a ciphertext corresponding to the message M, where Eve knows M, Eve can generate a ciphertext corresponding to any message M', (of the same length as M) that Bob will accept. (For example, if Alice sends the message "ATTACK AT TEN AM" Eve can drop it and make Bob accept the message "GO BACK HOME BOB" instead.)

3. Protocol (an)droids. (24 pts) Two robots Artoo and C3-2-0 often fly on different starships and need to alert each other to their presence when their ships come in contact—otherwise they might accidentally blow each other up! They agree on a shared key K and a MAC algorithm that outputs 256-bit tags to use in the following protocol.

1. $A \longrightarrow C$: a random 256-bit string N_A and $\mathsf{MAC}_K(N_A)$.
2. C: on message n, t check that $\mathsf{MAC}_K(n) = t$, and if so, accept A, otherwise blow up the other party.
3. $C \longrightarrow A$: $\mathsf{MAC}_K(t)$.
4. A: on message t' check that $t' = \mathsf{MAC}_K(\mathsf{MAC}_K(N_A))$. If so, accept C, otherwise blow C up.

The idea here is that A proves it is A by correctly MACing N_A (which, if the key is secret, only A or C could do) and C proves it is C by MACing the MAC. But...

(a) A and C use this protocol for a while and then discover, to their dismay, that sometimes the evil galactic robo-emperor, E, has been successfully fooling C into believing it is A. Even supposing that robot-in-the-middle attacks are prevented by speed-of-light limitations or some other plot contrivance, what is a simple way for E to do this?

(b) A and C decide that one way to prevent the attack is for C to remember every value of N_A used in a previous challenge and reject if one is ever reused. Suppose E sees one authentication between A and C. How can it fool C into believing it is A as many times afterwards as it wants?

4. Hashing and Signing. (20 pts) Nearly every digital signature scheme works by first hashing a message to be signed (with a cryptographic hash function) and then performing some operation on the hash—so in essence, we are "signing the hash" and not the message. In particular, if Eve sees Alice's signature on the message M and can find a message $M' \neq M$ so that $H(M) = H(M')$, she can convince people that Alice signed M'. This is OK, since a good crypto hash function H will resist finding targeted collisions (second pre-images) like this.

Suppose our signature scheme uses a hash function H with an output length ℓ that is sufficient to resist second pre-images but NOT resistant to free collisions (e.g. the hash length is around 100-120 bits). Then it is possible that if Eve can get Alice to sign one of a pair of colliding messages, she can later claim that Alice signed the other.

The classic birthday attack works by hashing random messages until two have the same hash. This could already be a problem in some applications, but you might object that Alice is unlikely to agree to sign a random message. So let's think about how to create a collision with more specific messages.

Suppose that a message is "favorable" if it is something that Alice would sign, for example "I will pay \$5 to McDonald's for my lunch." Suppose that a message is "undesirable" if it is something that Alice would not sign, like "I will pay \$500,000 to Eve for her lunch." Notice that we can generate 256 different "favorable" messages from the example above, for instance by varying the number of space characters between words between 1 and 2. Extend this idea to show how to generate a pair of messages, one favorable and one undesirable, with the same hash. Your attack should compute about as many hashes as the birthday attack.

Then, describe how Eve completes the attack using the pair she generates to her advantage.