Paxson
Spring 2013

CS 161
Computer Security

Homework 3

Due: Wednesday April 10, at 11:59pm

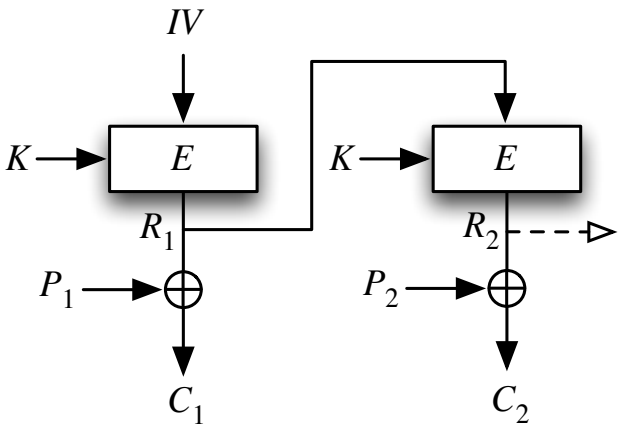
Instructions. Submit your solution electronically *via your class account* by Wednesday April 10, at 11:59pm. You should upload a single file, `HW3.pdf`. Your writeup should include your name, your class account name (e.g., `cs161-xy`), your TA's name, your discussion section, members of your study group (if any; **see below**), and "HW3" prominently on the first page. Use a legible font and clearly label each solution with the problem/subproblem to which it belongs. You *must* submit a PDF file; we will not accept other formats.

You can work on this homework in study groups of up to four people; however, you **must** write up the solutions *on your own*. You must never read or copy the solutions of other students (or from online materials), or co-develop writeups, and you must not share your own solutions with other students. You must explicitly acknowledge everyone who you worked with or who has given you any significant ideas about the homework.

NOTE: In this problem set, you can assume that the underlying authentication and encryption functions are secure and have no implementation flaws. Furthermore, ignore attacks on availability and restrict your analysis to attacks on confidentiality, integrity, and authenticity.

Problem 1 ***Integrity and OFB*** **(16 points)**

The lecture notes discuss a block cipher mode called *Output Feedback Mode*, or **OFB**. Its operation looks like this:



Suppose Alice needs to transmit to Bob the urgent 24-byte message M = "`The traitor is: Mallory!`". She and Bob have agreed upon the use of a block-cipher E with a block size of 64 bits and a key size of 128 bits. They also agree to use OFB to encrypt messages longer than a single block. They've previously exchanged a secret shared key K.

Alice generates a random IV and computes C, the ciphertext corresponding to M, using OFB with K and the IV. She attempts to transmit $\{IV, C\}$ to Bob ...

... but, alas, Mallory intercepts the message! Bob never receives it.

(a) Describe how Mallory could construct a ciphertext C' that when decrypted by Bob (using the same IV as Alice chose) will implicate the innocent Charlie rather than Mallory. (Mallory does not possess K.)

(b) Describe an approach that Alice and Bob could agree to use that would make it infeasible for Mallory to successfully fool Bob with the attack you presented in the previous part.

Problem 2 ***Outwitting Eve and Mallory*** **(20 points)**

Alice and Bob would like to use what they learned in CS 161 to communicate by email without letting anyone else read their letters.

(a) Alice and Bob decide to use public-key cryptography. Alice emails Bob her public key, and vice versa. Now they can communicate by encrypting messages using the respective public keys. Is Alice and Bob's communication secure against Eve, the passive eavesdropper? How about Mallory, the MITM attacker?

(b) Alice and Bob decide instead to use the Diffie-Hellman Key Exchange. Bob emails Alice a value g^x and Alice emails Bob a value g^y. They then use these to establish a secret key g^{xy} that they use to encrypt all of their subsequent emails. Is Alice and Bob's communication secure against Eve? How about Mallory?

(c) Alice and Bob decide not to use email for sending each other public keys. Instead they meet at Prude Awakening, the local English tea house, and just tell each other the public keys. Now they can communicate by encrypting messages using the respective public keys. Unfortunately, Prof. Evil overhears their conversation, and, being evil, will communicate everything he heard to all evil-doers in the world, including Eve and Mallory.

Will Alice and Bob's subsequent communication be secure against Eve? How about Mallory?

(d) Freaked out by the appearance of Prof. Evil, Alice insists on a further security measure. She insists that for every communication that Alice and Bob do in the future, they begin with a Diffie-Hellman Key Exchange. They will use their public-key encryption keys for sending the DH key exchange messages (like g^x), but after establishing a fresh key g^{xy}, the actual secret messages will be sent using that key and not the respective public keys. Is there any advantage of this additional requirement?

Problem 3 ***Order of Authentication and Encryption*** **(20 points)**

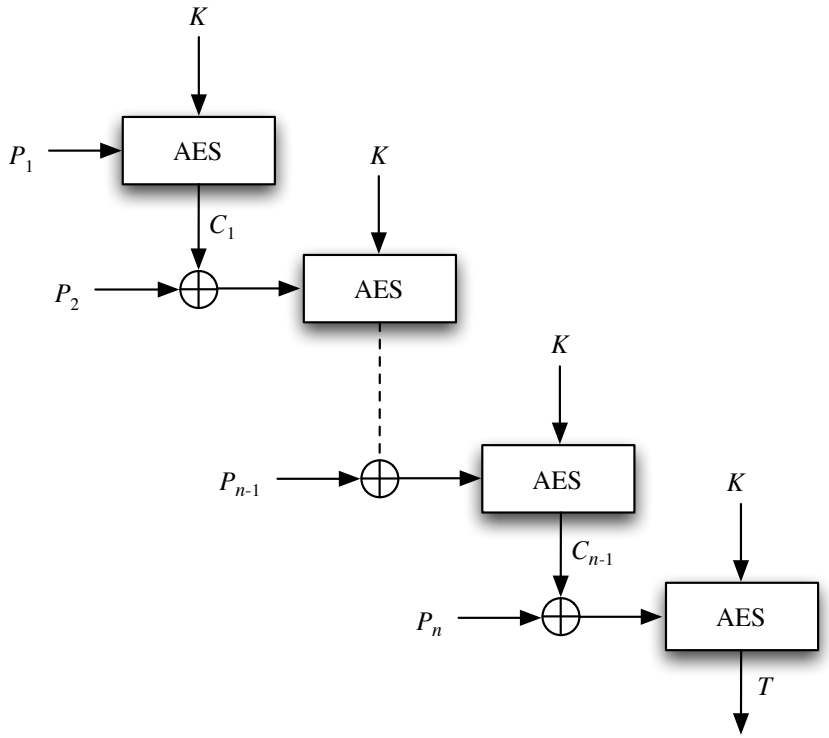
Given a message M, an authentication function A_{k_1} (MAC or digital signature), and an encryption function E_{k_2} (symmetric or asymmetric), there are several choices of how to authenticate and encrypt messages. Three ways you can do this are:

$$A_{k_1}(M), E_{k_2}(M) \tag{1}$$
$$A_{k_1}(E_{k_2}(M)), E_{k_2}(M) \tag{2}$$
$$E_{k_2}(M \mid\mid A_{k_1}(M)) \tag{3}$$

(1) says send the authentication of the message and the encryption of the message separately. (2) says to encrypt the message, then send the authentication of the encryption along with the encryption. Finally, (3) says first authenticate the message, then encrypt the concatenation of the message and its authentication.

For each of these, discuss to what degree it is secure, and what trade-offs a given approach provides, such as regarding performance.

Problem 4 ***MAC Attack*** **(20 points)**



Consider the MAC algorithm shown in the diagram above. Each P_i is the ith block of a given message. At each stage, we encrypt the XOR of the previous stage and the next message block using the key K.

This algorithm is quite similar to AES-EMAC (shown in lecture and in the lecture notes), but differs in final stage, by using the same key as in the earlier stages, not including a second invocation of AES.

(a) Suppose Mallory observes two single-block messages, M_1 and M_2, and the corresponding tags for these, T_1 and T_2. Show that Mallory can construct a message M_3 for which Mallory knows the associated tag T_3, even though Mallory does not know K.

(b) Generalize this attack for the case where M_1 is b_1 blocks in size, and M_2 is b_2 blocks in size.

Problem 5 ***Photo Authentication Flaw*** **(24 points)**

CalPix, a new social photo-sharing site, lets users upload photos to its servers and share them with their friends. For privacy, the service has implemented a form of access control: upon uploading a photo, it lets a user decide which other users can view it.

Photos are retrieved via calls to the `http://calpix.com/showphoto.php` page, which checks whether the given user should be able to view the photo, and if so, sends the photo file's contents.

The script `showphoto.php` accepts two URL parameters: `filename`, which is the filename of the photo to show, and `hash`, which is an authenticator. Photos are grouped into subdirectories by username, so the `filename` parameter is actually the path to the file, relative to the base photo directory (for example, `someuser/somephoto.jpg`).

Before showing any photo, `showphoto.php` checks the `hash` value sent. This value is the first five bytes (10 hex digits) of a SHA-256 digest of a secret `$encryption_key` concatenated with the filename. Only the server knows the secret key.[1]

The page `http://calpix.com/checkfile.php` looks up the current logged-in user's identity and checks whether the user is allowed to access a particular photo file. If so, `checkfile.php` generates the correct hash value and redirects to `showphoto.php`. `checkfile.php` is able to generate the correct hash value since it runs on the server side and thus has access to the secret `$encryption_key`.

Suppose access is allowed to the target photo `caladmin/me.jpg`. The script `checkfile.php` first generates a 256-bit SHA-256 hash, say:

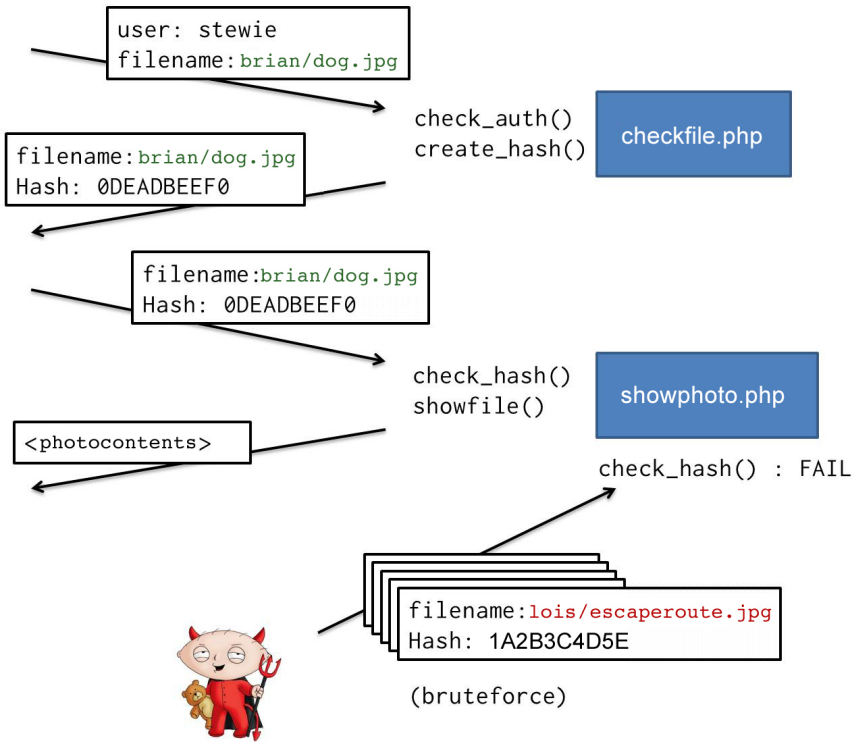
`BA5EBA1100A8E75E441F9D28EB4A9FDB7B40C2E3CA5FDB606E7E81229B4A4406`

It then truncates the hash and redirects the user to:

`http://calpix.com/showphoto.php?filename=caladmin/me.jpg&hash=BA5EBA1100`

The diagram below sketches this protocol. User `stewie` has access rights for the file `brian/dog.jpg`, but wants access to the `lois/escaperoute.jpg` file, which he tries to access via a brute-force attack.

[1] You may find this scheme peculiar, but in fact this example was inspired by a real life example, with the only significant change being that we're specifying the hash function as SHA-256, whereas in reality the function was MD5. We didn't want students distracted by possible weaknesses in MD5, which don't play a role here.



The hash value is checked by the following PHP code in `showphoto.php`:

```
// $filename: The filename parameter in the URL (e.g. username/photo.jpg)
// $given: The string value from the hash parameter in the URL.

// The code below generates a hash value and uses its first 5 bytes
// (10 hex characters) as a string.
$hash = substr(sha256($filename . $encryption_key), 10);
if ($hash == $given)
  // Access granted, code to dump photo contents...
else
  // Access denied.
```

(a) Operating systems provide *access control* to files based on permissions associated with each user of a given system. The kernel enforces these permissions on a per-user basis. Given that, why does `checkfile.php` need to do any checks? Won't the kernel ensure that only appropriate people get access?

(b) Of the primitives developed in class (signatures, encryption, MAC, hash), what functionality does the above hashing scheme aim to provide?

(c) Why do we need a secret key as an input to the hash function?

(d) A simple way to attack this scheme is to brute-force the URL. For example, the attacker can make a request to:

`http://calpix.com/showphoto.php?filename=lois/escaperoute.jpg&hash=1A2B3C4D5E`

and hope that the hash is correct. With high probability, it won't be. The attacker

can try more hash values and see if any happen to work. What is the probability of correctly guessing the hash value the first time?

NOTE: The `substr` function takes the 128-bit hash value and keeps only the first 40 bits (the first 10 hex characters, or first 5 bytes).

(e) There is another trick up the attacker's sleeve. Instead of guessing the hash value, the attacker could keep the hash value the same across guesses and change the filename parameter instead. In other words, the attacker could repeatedly try different URLs of the form `http://calpix.com/showphoto.php?filename=<`*changing_filename*`>&hash=`*h* for some value of h. The idea is to change the filename without changing the actual file referenced.

Sketch a way that the attacker could do this, assuming that the server runs on a Unix platform/file system.

(f) A subtle bug is introduced by the use of the equality operator (`==`). In PHP, this operator performs implicit type conversion; when numerical strings are compared, PHP converts the strings to numbers, and performs the comparison numerically. (The string-to-number conversion is explained in detail here: http://www.php.net/manual/en/language.types.string.php#language.types.string.conversion)

Due to this type conversion, for example the following expressions are true in PHP:

```
"0" == "00";     // true
"1" == "01";     // true
"10" == "1e1";   // true
"0100" == "1e2"; // true
```

though for example the following are false:

```
"0" == "a";      // false
"0" == "0a";     // false
"1" == "010";    // false
"100" == "1e2e"; // false
```

By leveraging this behavior, how high can the attacker make the probability of correctly guessing the hash value the first time? (HINT: Use the idea from part (e) above.)

You may find it helpful to use an online PHP interpreter (e.g., http://www.ideone.com) or an interactive shell for PHP (http://www.phpsh.org) to experiment and check your answers for this question.