Paxson Spring 2011 — CS 161 Computer Security — Homework 3

Updated 04Apr11: We changed the figure in Problem #3 illustrating the MAC algorithm to use different labeling for the message blocks, to avoid confusion with the names for different, distinct messages in the problem.

Due: Wednesday April 6, at 11:59pm

Instructions. Submit your solution electronically *via your class account* by Wednesday April 6, 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, the discussion section time where you want to pick up your graded homework, 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 must work on your own on this 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 ***Order of Authentication and Encryption*** **(20 points)**

Given a message M, an authentication function T_{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:

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

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

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

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 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, the passive eavesdropper? How about Mallory, the MITM attacker?

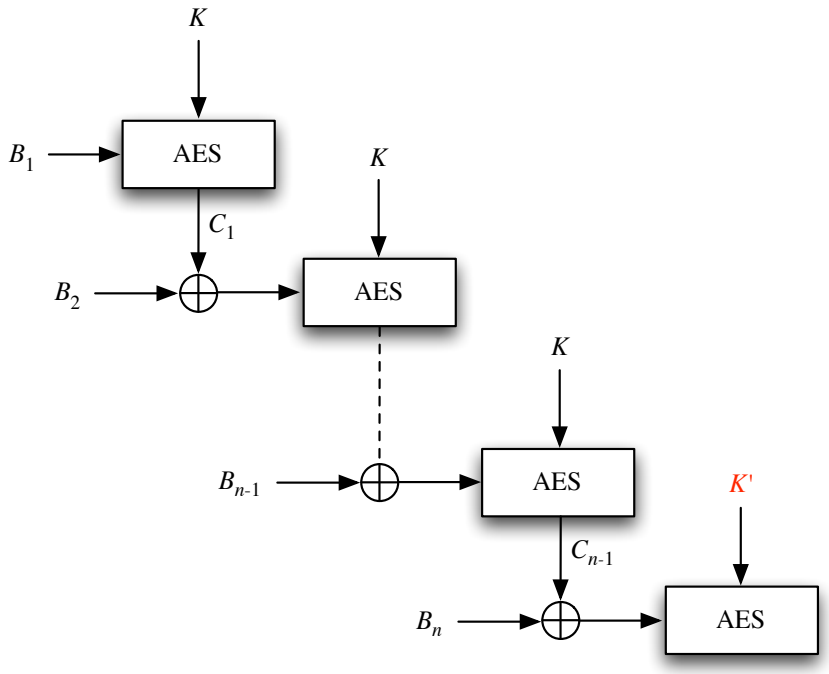
(b) Alice and Bob decide to instead use public-key cryptography. Alice emails Bob her public key, and vice versa. Now they can communicate by encrypting messages using the respective public keys. Is Alice and Bob's communication secure against Eve? How about Mallory?

(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 ***MAC Attack*** **(20 points)**



The diagram shows the MAC algorithm (based on AES) presented in lecture. Each B_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, *except* for the final stage, where we use a separate key, K'.

Consider a version of this algorithm that uses K throughout, i.e., $K' = K$.

(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 4 ***Properties of Block Cipher Modes*** **(16 points)**

The lecture notes describe four block-cipher modes of operation: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Output Feedback (OFB), and Counter (CTR). Let $E_K(\cdot)$ and $D_K(\cdot)$ denote encryption and decryption of a single block under key K, respectively. Then given a message $M = M_1, \ldots, M_\ell$, each of those four modes defines a different way of computing the ciphertext blocks $C_1, \ldots, C_\ell$. Some modes also require a random initial vector IV to be selected before encrypting a message.

(a) Suppose $M_i = M_j$ for some $i, j \in \{1, \ldots, \ell\}$ where $i \neq j$. Which of the four modes ensure that $C_i = C_j$ in this case? Is this good or bad?

(b) Encryption of messages with many blocks can be made more efficient if the individual blocks can be encrypted in parallel. This is possible whenever each ciphertext block C_i can be computed from only a fixed number of message blocks (that is, independent of ℓ) along with the IV (if there is one). Which mode or modes allow parallel encryption in this sense?

(c) Similarly, decryption can be parallelized if each message block M_i can be computed from only a fixed number of ciphertext blocks along with the IV (if there is one). Which mode or modes allow parallel decryption in this sense?

(d) Suppose a ciphertext block C_i is changed in transit. Then after decryption, some of the resulting message blocks will be corrupted. For each of the four modes, what are the indices of the message blocks that will be corrupted? Moreover, how could you detect the fact that C_i has been changed in transit?

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

Imagine a web server running a content management system (CMS) that gives read access to files via the `http://cms/showfile.php` page. Obviously, not everyone should be allowed access to files: access control is necessary.

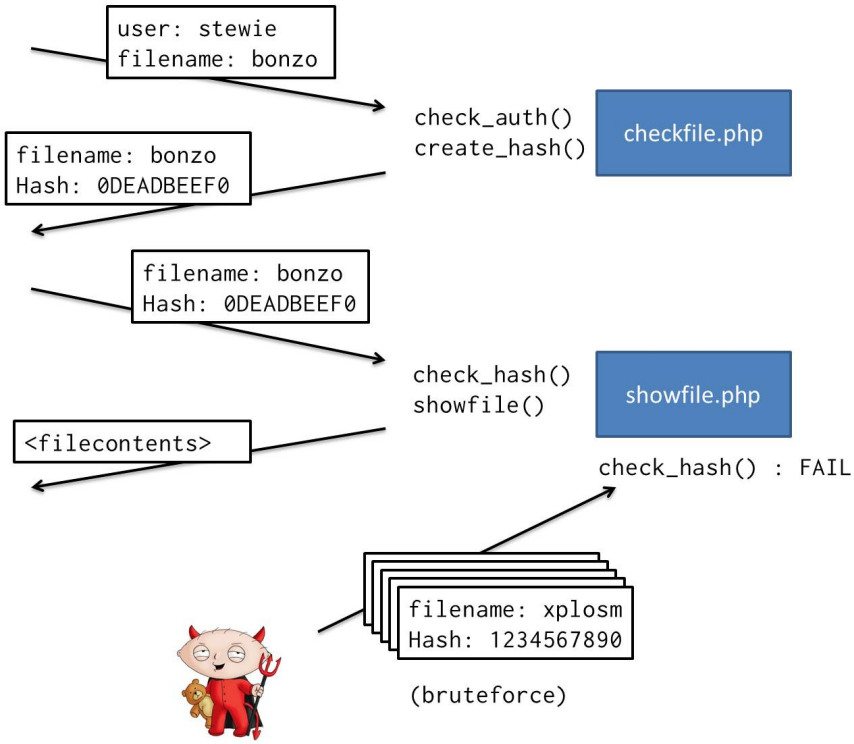
Therefore, `showfile.php` accepts two URL parameters: `filename`, which is the name of file to show, and `hash`, which is an authenticator. Before showing any file, `showfile.php` checks the `hash` value sent. This value is the first five bytes (10 hex digits) of a SHA-256 digest of a secret `$encryption_key` concatenated with the filename. Only the server knows the secret key.[1]

For example, the page `http://cms/checkfile.php` can look up the current logged-in user's identity and check whether the user is allowed to access a particular file. If so, `checkfile.php` generates the correct hash value and redirects to `showfile.php`.

[1] You may find this scheme peculiar, but in fact this example is taken from real life, 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.

`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`.

The diagram below sketches this protocol. User `stewie` has access rights for the file `bonzo`, but wants access to the `xplosm` file, which he tries via a brute-force attack.



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

```
// $filename: The filename parameter in the URL.
// $given: The value from the hash parameter in the URL.

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

(a) We discussed in class how files have an access control list enforced by the kernel. Why does `checkfile.php` need to do any checks then? 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. The attacker can make a request to `http://cms/showfile?filename=/etc/passwd&hash=1234567890` 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 at 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://cms/showfile?filename=<`*changing_filename*`>&hash=0`. The idea is to change the filename without changing the actual file referenced. How could the attacker do this?

(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, or a string is compared to a number, 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, the following expressions are true in PHP,

```
0 == "a";      // true
"1" == "01";   // true
"10" == "1e1"; // true
100 == "1e2";  // true
```

How can the attacker leverage this behavior to improve the chances at brute-forcing? (HINT: Use the idea from part (e) above.)

Using this additional knowledge, what is the probability of correctly guessing the hash value at the first time?