IK2206 – Internet Security and Privacy
Chapter 5 – Hashes and Message Digests

5.1 Introduction (READ CAREFULLY)

- What properties does the book associate with the term "randomness"?
  - If 1000 inputs are selected at random, any particular bit in the 1000 resulting outputs should be on about half the time
  - Each output should have, with high probability, about half the bits on
  - Any two outputs should be completely uncorrelated

- What does this imply for message digests?

  It is true that someone who knows the message and the digest function can calculate the output. It should not be possible, other than by computing the message digest, to predict any portion of the output.

- What does it mean to "fingerprint" a program?

  It prevent that a program or document can be modified undetected. It stores the message digest securely (for example with encrypted) and compute the message digest of the program before running it, and check to make sure that it matches the stored value.

- What message digest size is suggested in the book? (Later you can compare this with the sizes of SHA family of message digests.)

  A message digest functions have outputs of at least 128 bits, because it is not considered feasible to search 2^{64} messages given the current state of the art.

  If a digest has m bits, then it would take only about $2^{m/2}$ messages, chosen at random, before one would find two with the same value.

- What is the "birthday problem"? How does it apply when evaluating the strength of a message digest?

  If there are 23 or more people in a room, the odds are better than 50% that two of them will have the same birthday.

  Let's assume n inputs and k possible outputs, and an unpredictable mapping from input to output. With n inputs, there are n(n-1)/2 pairs of inputs, For each pair there is a probability of 1/k of both inputs producing the same output values. Therefore we will need about k/2 pairs in order for the probability to be about 50 %

5.2 Nifty Things to Do with a Hash (READ)

5.2.1 Authentication (SKIP)

5.2.2 Computing a MAC with a Hash (READ)

- What is a "keyed hash"?

  Any hash combining the secret key and the data.

- Why can't a hash be used for integrity protecting a message "as is", i.e., why do you also need a secret key?

  A hash is comparable with a checksum. As soon as data changes, the hash does also change. Since everyone can calculate hashes from data, hashing does not preserve the integrity property. An attacker could change transmitted data and simply reattach a new hash code. How could the receiver find out that 1) the message is unchanged and 2) the identity of the sender wasn't faked during the transmission?

  HMAC is a hashing algorithm that takes a secret key as additional parameter. A receiver knows that hmaced messages can only come from someone that knows the secret key.

⇨ **IS THIS ALL CORRECT???**

- To compute a MAC, why isn't it a good idea to use MD(K|m)?

  Let's assume Carol would like to send a different message to Bob, and have it look like it came from Alice. Carol does only care what is in the end of the message. Therefore, she manipulates the end of the message. Alice has transmitted some message m, and MD(K|m). Carol can see both of those quantities. She concatenates the padding and then whatever she likes to the end of m, and initializes the message digest computation with MD(K|m). She does not need to know the shared secret in order to compute the MAC

-
  - Why would it be better to use MD(m|K)?

  The design of the HMAC specification was motivated by the existence of attacks on more trivial mechanisms for combining a key with a hash function. For example, one might assume the same security that HMAC provides could be achieved with MAC = H(key | message). However, this method suffers from a serious flaw: with most hash functions, it is easy to append data to the message without knowing the key and obtain another valid MAC. The alternative, appending the key using MAC = H(message | key), suffers from the problem that an attacker who can find a collision in the (unkeyed) hash function has a collision in the MAC. Using MAC = H(key | message | key) is better, however various security papers have suggested vulnerabilities with this approach, even when two different keys are used.
  http://en.wikipedia.org/wiki/HMAC

  - What other methods does the book suggest?

    - Use only half the bits of the message digest as the MAC
    - Concatenate the secret to both the front and the back of the message
    - H(key1 | H(key2 | message)) because the outer application of the hash function masks the intermediate result of the internal hash.

5.2.3 Encryption with a Message Digest (READ)

5.2.3.1 Generating a One-Time Pad (READ)

- How can one generate a one-time pad using a hash function?

  Compute MD(Kab) -> gives the first block of the bit stream b1. Then compute MD(Kab|b1) and uses that as b2, and in general b_i is MD($K_{AB}|b_{i-1}$)

- How do you encrypt a message longer than the message digest length of your hash function?

  Take the intermediate result of one step as the input of the next step. **(???)**

- What is the purpose of the Initialization Vector IV?

  It is not secure to use the same bit stream twice. Therefore, Alice starts with an IV. -> MD(Kab|IV). Alice must transmit the IV to Bob.

5.2.3.2 Mixing In the Plaintext (READ BRIEFLY)

5.2.4 Using Secret Key for a Hash (READ BRIEFLY)

5.2.4.1 UNIX Password Hash (READ BRIEFLY)

5.2.4.2 Hashing Large Messages (SKIP)

5.3 -- 5.5 MD2-MD5 (SKIP)

Read the fact box on p. 122 instead to learn about why there are so many message digest algorithms.

5.6 SHA-1 (READ BRIEFLY)

5.7 HMAC (READ)

- What is HMAC? Is it a hash algorithm?

  Hash-based Message Authentication Code. It is a specific construction for calculating a message authentication code involving a cryptographic hash function in combination with a secret key.

- How can HMAC promise to safely produce a MAC from a secret key and a hash function (a keyed hash), without specifying the hash function?

  HMAC prepends the key to the data, digest it, and then prepends the key to the result and digest that. Nested digest with secret inputs prevents from attacks.

- Although relatively computationally intensive, HMAC has become popular to use in many standardized protocols when creating MACs. Could you think of a reason why?

The strongest attack known against HMAC is based on the frequency of collisions for the hash function H ("birthday attack") and is totally impractical for minimally reasonable hash functions.

As an example, if we consider a hash function like MD5 where the output length equals L=16 bytes (128 bits) the attacker needs to acquire the correct message authentication tags computed (with the same secret key K) on about 2^{64} known plaintexts. This would require the processing of at least 2^{64} blocks under H, an impossible task in any realistic scenario (for a block length of 64 bytes this would take 250,000 years in a continuous 1Gbps link, and without changing the secret key K during all this time).

This attack could become realistic only if serious flaws in the collision behavior of the function H are discovered (e.g. collisions found after 2^{30} messages).

Source: http://www.faqs.org/rfcs/rfc2104.html

http://technet.microsoft.com/en-us/library/cc781476(WS.10).aspx