U.C. Berkeley — CS276: Cryptography | Handout N10 (Draft)
Luca Trevisan | February 19, 2009

# Notes for Lecture 10 (Draft)

## Summary

Cryptographic hash functions, as defined last time, have various application. Their use for message authentication is common, but, unless it is carefully designed, it can lead to insecure systems.

We conclude the lecture with an overview of how the current standard pseudorandom permutation (AES) works. Next time we begin a study of how to construct pseudorandom permutations (and hence secure MACs and CCA-secure encryption) starting from well-understood assumptions such as the hardness of factoring or discrete log.

## 1 Hash Functions and Authentication

Suppose $H : \{0,1\}^k \times \{0,1\}^{2\ell} \to \{0,1\}^\ell$ is a collision-resistant hash function and $H_{MD} : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\ell$ is the hash function derived from the Merkle-Damgård transform.

Two popular schemes for message authentication involve using a key $K \in \{0,1\}^\ell$, and authenticating a message $M$ as either

1. $H^s_{MD}(K, M)$ or
2. $H^s_{MD}(M, K)$.

The first scheme has a serious problem: given the tag of a message $M_1, \ldots, M_B$ of length $\ell$, we can compute the tag of any message that starts with $M_1, \ldots, M_B, \ell$.

The NMAC scheme works roughly as a scheme of the second type, but with two keys, the first used in place of IV in the Merkle-Damgård transform, the second put at the end of the message.

This is secure assuming that $H^s(K, M)$ (where $H$ is the fixed-length hash function) is a secure MAC for fixed length messages.

An alternative scheme (which is easier to implement given current cryptographic libraries) is HMAC, which uses $H^s(IV, K \oplus ipad)$ as a first key and $H^s(IV, K \oplus opad)$

as a second key, where *ipad* and *opad* are two fixed strings. This is secure if, in addition to the assumptions that make NMAC secure, we have that the mapping

$$s, K \to s, H^s(IV, K \oplus ipad), H^s(IV, K \oplus opad)$$

is a pseudorandom generator.

## 2 The AES Pseudorandom Permutation

AES is a pseudorandom permutation with $m = 128$ and $k = 128$, 192 or 256. It was the winner of a competition run by NIST between 1997 and 2000 to create a new encryption standard to replace DES (which had $k = 56$ and was nearly broken by that time).

The conjectured security of practical pseudorandom permutations such as AES does not rely on the hardness of a well-defined computational problem, but rather on a combination of design principles and of an understanding of current attack strategies and of methods to defy them.

AES keeps a state, which is initially equal to the input, which is a $4 \times 4$ matrix of bytes. The states is processed in 4 stages. This processing is done 10, 12, or 14 times (depending on key length), and the final state is the output.

1. In the first stage, a 128-bit string derived from the key (and dependent on the current round) is added to the state. This is the only stage that depends on the key;
2. A fixed bijection (which is part of the specification of AES) $p : \{0,1\}^8 \to \{0,1\}^8$ is applied to each byte of the state
3. The rows of the matrix are shifted (row $i$ is shifted $i-1$ places)
4. An invertible linear transformation (over the field $GF(256)$) is applied to the matrix

The general structure is common to other conjecturally secure pseudorandom permutations:

- There is one (or more) small "random-like" permutations that are hard-wired in the construction, such as $p : \{0,1\}^8 \to \{0,1\}^8$ in AES. Traditionally, those hard-wired functions are called "S-boxes."

- A “key-scheduler” produces several “pseudorandom” strings from the key. (Usually, the scheduler is not a true pseudorandom generator, but does something very simple.)
- The construction proceeds in several rounds. At each round there is some combination of:
  - “Confuse:” apply the hard-wired S-boxes locally to the input (Stage 1 in AES)
  - “Diffuse:” rearrange bits so as to obscure the local nature of the application of the S-boxes (Stages 3 and 4 in AES)
  - “Mix:” use a string produced by the key-scheduler to add key-dependent randomness to the input (Stage 2 in AES)