Lecture 15: Hashing for Message Authentication

Lecture Notes on "Computer and Network Security"

by Avi Kak (kak@purdue.edu)

March 29, 2017
1:30am

Goals:

- What is a hash function?
- Different ways to use hashing for message authentication
- The birthday paradox and the birthday attack
- Structure of cryptographically secure hash functions
- SHA Series of Hash Functions
- **Compact Python and Perl implementations for SHA-1 using BitVector** [Although SHA-1 is now considered to be fully broken (see Section 15.7.1), programming it is still a good exercise if you are learning how to code Merkle type hash functions.]
- Message Authentication Codes

CONTENTS

15.1: WHAT IS A HASH FUNCTION?

- In the context of message authentication, a hash function takes a **variable sized input message** and produces a **fixed-sized output**. The output is usually referred to as the **hashcode** or the **hash value** or the **message digest**. [Hash functions are also extremely important for creating efficient storage structures for associative arrays in the memory of a computer. (As to what is meant by an "associative array", think of a telephone directory that consists of <*name,number*> pairs.) Those types of hash functions also play a central role in many modern big-data processing algorithms. For example, in the MapReduce framework used in Hadoop, a hash function is applied to the "keys' related to the Map tasks in order to determine their bucket addresses, with each bucket constituting a Reduce task. In this lecture, the notion of a hash function for efficient storage is briefly reviewed in Section 15.9.]

- For example, the SHA-512 hash function takes for input messages of length up to 2^{128} bits and produces as output a 512-bit **message digest (MD)**. **SHA** stands for **Secure Hash Algorithm**. [A series of **SHA** algorithms has been developed by the National Institute of Standards and Technology and published as Federal Information Processing Standards (FIPS).]

- We can think of the hashcode (or the message digest) as a **fixed-**

sized fingerprint of a variable-sized message.

- Message digests produced by the most commonly used hash functions range in length from 160 to 512 bits depending on the algorithm used.

- Since a message digest depends on all the bits in the input message, any alteration of the input message during transmission would cause its message digest to not match with its original message digest. This can be used to check for forgeries, unauthorized alterations, etc. To see the change in the hashcode produced by an innocuous (practically invisible) change in a message, here is an example:

```
Message:            "The quick brown fox jumps over the lazy dog"
SHA1 hashcode:      2fd4e1c67a2d28fced849ee1bb76e7391b93eb12

Message:            "The quick brown  fox jumps over the lazy dog"
SHA1 hashcode:      8de49570b9d941fb26045fa1f5595005eb5f3cf2
```

  The only difference between the two messages shown above is the extra space between the words “brown” and “fox” in the second message. Notice how completely different the hashcodes look. SHA-1 produces a 160 bit hashcode. It takes 40 hex characters to show the code in hex.

- The two hashcodes (or, message digests, if you would rather call them that) shown above were produced by the following Perl script:

```
#!/usr/bin/perl -w
use Digest::SHA1;
my $hasher =  Digest::SHA1->new();
$hasher->add( "The quick brown fox jumps over the lazy dog" );
print $hasher->hexdigest;
print "\n";
$hasher->add( "The quick brown  fox jumps over the lazy dog" );
print $hasher->hexdigest;
print "\n";
```

As the script shows, this uses the SHA-1 algorithm for creating the message digest. [I downloaded the module `Digest-SHA1` directly from `http://search.cpan.org/`. When I tried to do the same by downloading the libraries `libdigest-perl` and `libdigest-sha-perl` through the Synaptic Package Manager on my Ubuntu laptop, it did not work for me.]

- Perl's **Digest** module, used in the script shown above, can be used to invoke any of over fifteen different hash algorithms. The module can output the hashcode in either **binary** format, or in **hex** format, or a binary string output as in the form of a **Base64**-encoded string. A similar functionality in Python is provided by the **hashlib** library. Both the **Digest** module for Perl and the **hashlib** library for Python come with the standard distribution of the two languages.