

1-02-328: Communication and E-Commerce Security

Recitation 6: Hashing

Michael J. May

March 30, 2011

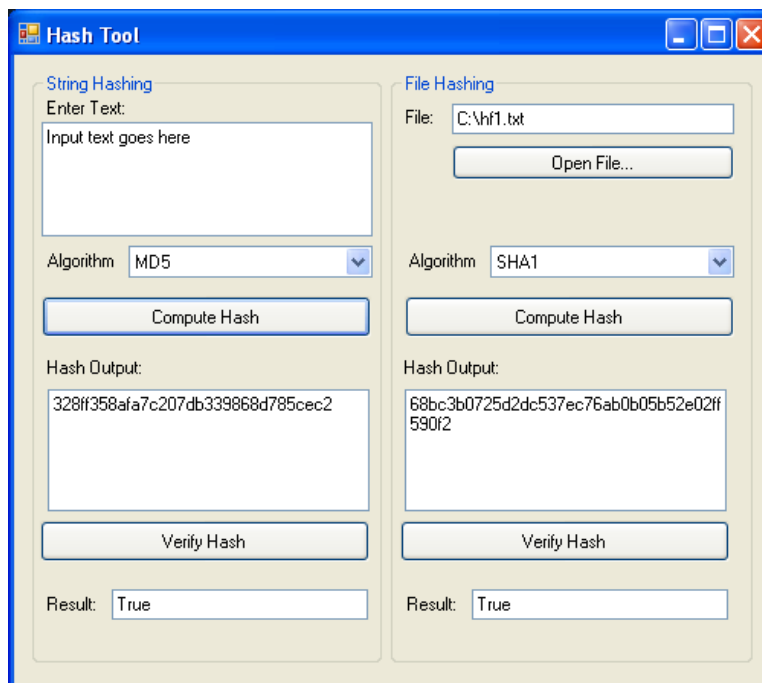
In this recitation we will write some code which uses the .NET cryptographic libraries for hashing and hash verification.

1 Introduction

In class we talked about two hash functions, SHA1 and MD5. We didn't go into the details of how they are computed, but both functions take a variable length input and produce a fixed length output. Microsoft .NET offers managed implementations of both MD5 and SHA1 for you to use to compute hashes. Hash verification is fairly simple - just examine whether the computed hash value is equal to the value it is supposed to have.

2 GUI

The following (empty) GUI is provided for you to start with:



Just like DES (and AES), the output of the hash functions is a byte array (`byte[]`) which can be displayed as is. It is more common to display digests as hexadecimal strings, so I have written a simple function

```
string ShowHex(byte[] ar)
```

which takes a byte array and returns an equivalent hexadecimal string. For those interested, the output C# byte array stores the the highest order byte in array entry 0. You can use the function I provided to format the `byte[]` output in a hexadecimal string.

3 What to do

Your job is to use the MS .NET implementation to perform the following tasks:

1. Compute a hash using MD5 or SHA1 on a text string
2. Compute a hash using MD5 or SHA1 on a file
3. Verify the hash using MD5 or SHA1 on a text string
4. Verify the hash using MD5 or SHA1 on a file

Since .NET already provides implementations of SHA1 and MD5, your task is quite simple. The .NET implementation of MD5 is called `MD5CryptoServiceProvider` and the implementation of SHA1 is called `SHA1CryptoServiceProvider`. You will need to create instances of both classes to do the work for you.

3.1 Using Hashes

The interface for the hash functions is straightforward, you just need to use the function:

```
byte[] ComputeHash()
```

For a text string you must first convert it to a byte array (use the `Encoding.ASCII.GetBytes()` function for this).

```
byte[] ComputeHash(encodedInput)
```

For a file you must first build a `FileStream` object based on the provided file name and then provide it to the function:

```
byte[] ComputeHash(fileStream)
```

Note: Don't forget to close the `FileStream` when you are done using it.

3.2 Verifying Hashes

The verification problem is as follows:

1. You receive a hashed value y and a value x . The person who gave it to you claims that $Hash(x) == y$.
2. You must verify that claim.
3. The only way to do that is to calculate $Hash(x) \rightarrow y'$ yourself. If $y' == y$, you know that the claim was correct.

This is the only way to do hash verification using modern hashes - so that is what you must do in your code.

4 Experiments

Once you have completed your hashing tool, use it to perform the following experiments:

1. Input the text “Octogenarian tadpoles” into the hash **text** box of the window and compute its digest using SHA1 and MD5. Make sure it verifies with SHA1 and MD5 (respectively).
2. Delete the last character (“s”) in the hash **text** box. Compute the digest using SHA1 and MD5. Compare your results from the previous experiment. How much of the string changed with SHA1? How much changed with MD5? What does that tell you about the sensitivity of the two algorithms to small changes in the input?
3. Put back the last “s” you erased in the previous experiment. Create a new file “hf1.txt” with the text “Octogenarian tadpoles” in it. Use the file select dialog box to select it and open it using the tool. Compute the hash using SHA1 and MD5. Is it identical to the result you calculated in the first experiment?
4. Go back into the portion of your code which converts the hash text input into a byte array (it’s probably in function `bCHashText_Click`). Change the encoding from ASCII to Unicode:

```
Encoding.Unicode.GetBytes()
```

Recompile and run your program again. Try the previous experiment (computing the hash on “Octogenarian tadpoles” as a string and file `hf1.txt`). Is the output identical? What does that tell you about the sensitivity of the algorithms to the file system and its default encodings?