

<http://www.cs.arizona.edu/classes/cs345/spring08/>

Program #4: “They did the MASH ...”

Due Date: April 10th, 2008, at the beginning of class

Overview: One practical application of hashing is found in the creation of *message digests*. A message digest is a bit-pattern formed from a given ‘message’ by a *digest function*, which can be thought of as a complex hash function. The idea is that the digest function produces a message digest from which the original message cannot easily be found. That is the basic principle behind the original UNIX password system: The encrypted passwords were available for anyone to look at, because people assumed that it was too hard to use the encrypted password to determine the plaintext password. In addition to being difficult to reverse, the digest function should be constructed to make it unlikely that two messages m and n have the same digest. This is analogous to choosing a hash function that minimizes collisions.

Message digests are used for two common tasks in cryptography: To provide a means for a user to detect if data has been altered, and to provide a way to ‘sign’ data so that its origin can be verified.

There are quite a few digest functions in existence; see the end of this handout for a reference. For this assignment, we’ll be implementing the MASH-2 (Modular Arithmetic Secure Hash) digest function; the MASH functions are defined in ISO/IEC Standard 10118.

The Algorithm: Don’t anyone pass out at the sight of this; yes, it’s rather involved, as are most things in Computer Science, but still straight-forward.

Given: A message x of length (in bits) b .
Two prime numbers, p and q .

1. Compute $M = pq$, and determine m , the number of bits in M .
2. Determine n , the largest multiple of 16 that is $\leq m$. n is the size of the message digest, in bits.
3. Create two n -bit binary numbers H_0 and A . Set $H_0 = 0$, and set A to a value consisting of four 1-bits and $n - 4$ 0-bits. Thus, $A = 11110000\dots0000$.
4. Append 0-bits to the right of x to make its length (in bits) equal to the nearest multiple of $\frac{n}{2}$.
5. Divide x into t blocks of $\frac{n}{2}$ bits each. Call these blocks x_1, x_2, \dots, x_t .
6. Create an $\frac{n}{2}$ -bit block x_{t+1} that is the binary representation of b (the original length of x in bits).
7. For each block x_1 through x_t :
 - (a) Partition x_i into nybbles. (A nybble is half a byte; a 4-bit chunk)
 - (b) Insert 1111 ahead of (that is, to the left of) each nybble to form the n -bit block y_i
8. Partition x_{t+1} into nybbles. Insert 1010 before each to form the n -bit block y_{t+1} .
9. For each block y_1 through y_{t+1} :
 - (a) $F_i = ((H_{i-1} \oplus y_i) \vee A)^{257} \% M$, where \oplus is bitwise eXclusive-OR, \vee is bitwise inclusive-OR, and 257 is 257_{10} .
 - (b) $G_i =$ the rightmost n bits of F_i
 - (c) $H_i = G_i \oplus H_{i-1}$

Result: H_{t+1} is the message digest.

An example would seem to be in order.

Given: $x = 101101_2$; thus, $b = 6_{10} = 110_2$.

The two prime numbers are $p = 6911_{10}$ and $q = 6947_{10}$.

1. $M = pq = 48010717_{10} = 10110111001001010111011101_2$, and thus $m = 26$ bits.
2. $n = 16$ (16 is the largest multiple of $16 \leq m$)
3. $H_0 = 000000000000000_2$ and $A = 111100000000000_2$
4. $b = 6$ and the nearest multiple of $\frac{n}{2}$ is 8, so x becomes 10110100.
5. $x_1 = 10110100$; $t = 1$.
6. $x_2 = 00000110$
7. $10110100 \Rightarrow 1011$ and $0100 \Rightarrow 1111$ 1011 1111 0100. Thus, $y_1 = 1111101111110100$.
8. $00000110 \Rightarrow 0000$ and $0110 \Rightarrow 1010$ 0000 1010 0110. Thus, $y_2 = 1010000010100110$.
9. For y_1 :
 - (a) $F_1 = 101011011010010100101011$ (no, I didn't much feel like showing any powers of 257 here)
 - (b) $G_1 = 1010010100101011$
 - (c) $H_1 = 1010010100101011$

For y_2 :

- (a) $F_2 = 10001000101100100000010111$
- (b) $G_2 = 1100100000010111$
- (c) $H_2 = 0110110100111100$

Result: $H_2 = 0110110100111100$ is the message digest.

Assignment: Write a complete, well-documented program that implements the MASH-2 algorithm. Your implementation is to accept two prime numbers (`ints`) and a message (of no less than two characters but no more than 6 characters in length) from the command line in that order, and display to the screen the digest (in binary and decimal) produced by the MASH-2 algorithm.

Data: The 2-to-6 character messages may consist of letters, digits, or punctuation symbols. For example, `133t!` is an acceptable message.

Convert the messages to binary representations as follows: For each character, determine the character's 7-bit ASCII value, and concatenate them to form a $(7n)$ -bit message, where n is the number of characters in the message. For example, consider the message "abc". a's ASCII value is 97, or 1100001_2 . b's is 1100010_2 , and c's is 1100011_2 . The binary representation of the message is 110000111000101100011_2 .

When you submit your program source code printout, also include a printout of the output it produces on the following groups of input values:

Primes		Message
7	11	dunno
6911	6947	cat
6911	6947	parrot
55511	77797	1pig

As mentioned above, supply these values, in the order shown, to your program on the command line. For example, this is what you'd type into a terminal window for the first group: `java Prog3 7 11 dunno`

As usual, test your program with other data values, because we will, too.

Output: If the user supplies valid input, then the program will produce the message's digest in binary and decimal representations. Here is the format that you are to use:

```
Binary: <binary representation using the characters 0 and 1>  
Decimal: <decimal representation>
```

And an example:

```
Binary: 0110110100111100  
Decimal: 27964
```

If any of the command-line input is inappropriate, display the message "Invalid Command Line Parameter" (without the quotes) and have the program terminate. To facilitate grading, please don't generate any other output.

Hand In: On the due date, turn in a printed listing of your well-documented program statements and the printed output of the program when run on the supplied test data.

As usual, you are required to submit your completed program file(s) to the class submission directory, using the `turnin` command. The turnin location for this assignment is `cs345p4`. Name your main program source file `Prog4.java` so that we don't have to guess which file to compile/translate.

Want to Learn More?

- A list of message digest systems can be found here:
<http://www.users.zetnet.co.uk/hopwood/crypto/scan/md.html>
- The algorithm description in this handout is a slight variation of the one presented in the book *Introduction to Cryptography with Java Applets* by David Bishop. This book does contain an implementation of the algorithm, but it is somewhat optimized and rather distinctive. Resist the temptation to find the book and use any portion of Bishop's code (or that of any other implementation you may find) as your own; that's academic dishonesty, and would earn you a zero with no opportunity to resubmit.

Other Requirements, Hints, and Miscellaneous Babbling:

- You will find Java's `java.math.BigInteger` class to be very useful. It's as if it were designed just for this sort of thing...
- Be aware that several punctuation symbols have special meaning to UNIX command shells. Thus, using them as input data on a command line can be difficult. You may need to escape some of them, or put double-quotes around the entire message.
- Origin of the assignment title: "Monster Mash" is a 1962 song written by Bobby Pickett and Lenny Capizzi, and was first performed by Bobby 'Boris' Pickett and the Crypt-Kickers. The second chorus starts with the line "They did the mash".