A MATLAB Implementation of the WHIRLPOOL Hashing Function

Karl Petre <kp2272@columbia.edu>
Xavier Ortiz <xo2106@columbia.edu>

December 8, 2008

We present a polished MATLAB implementation of the WHIRLPOOL hashing algorithm. Our software package is composed of 10 functions in total.

Our decision to produce this code was motivated in part by the limited number of example implementations provided by the authors. While their documentation is thorough, they only provide implementations in the lower-level languages of C and Java; we felt it would be beneficial to the academic community to implement the algorithm in a language accessible to a wider variety of individuals.

Execution of the algorithm requires frequent conversion between binary, hexadecimal and decimal number representations. MATLAB has a built-in function library for converting between these data types (including functions such as `dec2hex()`, `bitxor()`, etc.); however, these only accommodate numbers smaller than 10000000000000_x in length. Unfortunately, the operations within the WHIRLPOOL algorithm deal primarily with hexadecimal vectors of length 64. As such, our package is built on a custom (albeit straightforward) data type, which allows MATLAB to handle said representations up to an arbitrary size. We introduce a data type called the *binary vector*, or *bv*, which is a 1-by-n array of ones and zeros, along with a library for working with this type. The provided library functions are the following:

- The function `bv2dec()` computes the decimal equivalent of the binary vector number a, which is expressed in a 1-by-length(a) array of ones and zeros. Usage is

  ```
  [x] = bv2dec(a);
  ```

  where a is the input number in binary vector form.
- The function `bv2hex()` computes the hexadecimal equivalent of the binary vector number a, which is expressed in a 1-by-length(a) array of ones and zeros. Usage is

  ```
  [x] = bv2hex(a);
  ```

  where a is the input number in binary vector form.
- The function `bvshift()` computes the bitwise shift of the binary vector number a, which is expressed in a 1-by-length(a) array of ones and zeros. This is equivalent to the binary operation `a << s`. The output is a 1-by-p array of ones and zeros. Usage is

  ```
  [x] = bvshift(a,s,p);
  ```

  Note that calling `a = bvshift(a,0,p)` is a means for shortening or lengthening the binary vector a to length p.
- The function `dec2bv()` computes the binary vector equivalent of the decimal number a; it returns the value as a 1-by-p array of ones and zeros. Usage is

  ```
  [x] = dec2bv(a,p);
  ```

- The function `hex2bv()` computes the binary vector equivalent of the hexadecimal number a; it returns the value as a 1-by-p array of ones and zeros. Usage is

  ```
  [x] = hex2bv(h,p);
  ```

We include one simple utility function to expedite the printing of strings to the terminal:

- The function `pp()` groups a string of hexadecimal digits into groups of p characters, inserting a space between each p characters and returning the result as a string. Usage is

```
[x] = pp(a,p);
```

There are three main functions which together compute the WHIRLPOOL hash for an input string:

- The function `buildconstants()` builds the WHIRLPOOL constants from the substitution box. This is the most computationally intensive function in the whole software package; in fact, the authors' C implementation simply loads the precomputed values instead of calculating them as is done here. Usage is

```
[C,rc] = buildconstants(R,vb);
```

  where R is the number of rounds, and setting the vb flag to 1 enables a verbose (debugging) output. Calling the function with no inputs sets R to the default of 10 and vb to a default of 0.

- The function `processbuffer()` executes the core WHIRLPOOL transform. Usage is

```
[hash,K] = processbuffer(C,rc,R,buffer,hash,vb);
```

  where setting the vb flag to 1 enables a verbose output.

- The function `whirlpool()` computes the WHIRLPOOL hash of the input string s, presented in ASCII characters (e.g., "abc"). The hash is returned in a binary vector representation – as an 8-by-64 array of ones and zeros. (The output hash is easily converted to its hexadecimal equivalent using `bv2hex()`.) This is the function called under normal use of the software package. The padded input strings and output hashes of each stage are printed to the terminal automatically in hexadecimal form. Usage is

```
[hash] = whirlpool(s,C,rc,vb);
```

  where setting the vb flag to 1 enables a verbose output. The input arguments C and rc are optional; if these are not specified, the function will calculate them first before proceeding. If more than one hash is to be generated, the user may save time by calling `buildconstants()`, saving the computed values of C and rc, and then repeatedly calling `whirlpool()` with C and rc as input arguments. (Consult the code in `whirlpooliso()` for an example of this.)

The final component of the software package is a pre-written function for computing the ISO/IEC 10118-3 test vector set for WHIRLPOOL:

- The function `whirlpooliso()` generates the ISO/IEC 10118-3 test vector set for WHIRLPOOL. It stores these hashes in the file `isotestvectors.m`. Usage is

```
whirlpooliso(vb);
```

  where setting the vb flag to 1 enables a verbose output; the value of vb defaults to 0 without an input argument. This script also serves as example code for the batch-processing of WHIRLPOOL hashes. Note that the function comes with the last test vector commented out; computation of the vector is quite time consuming (and not necessary under normal circumstances).

In order to assure the accuracy of our software, we have made thorough comparisons to the sample hashes provided by the authors. We found perfect agreement between hashes for all attempted input strings; furthermore, all constants computed by `buildconstants()` have been verified.