```
// hide.cpp - HIDE A FILE IN THE UNUSED STORAGE ON A HARD DISK
//
// MAINTENANCE HISTORY
// DATE             PROGRAMMER AND DETAILS
// 18-09-14   JS     Original
// 23-09-14   MPF    Correctly select a non-overlapping page
// 13-10-14   MPF    Add random data to cipher page
//
//-----------------------------------------------------------------------------

#include <cstring>          // C-style string manipulation functions
#include <cstdlib>          // C-style standard library
#include <set>                      // C++ set declarations
using namespace std;        // Expand the standard namespace
#include <stdint.h>         // Standard integer type declarations
#include "hideAndSeek.h"    // Hide and seek declarations

//-----------------------------------------------------------------------------

// MAIN LINE

int
main (
       int           argc,         // Argument count
       char          *argv[])      // Argument value pointers
{
       const char    *key;         // Encryption and signature key
       FILE          *tmpFp;             // Temporary file pointer
       long long     spaceAvail;   // Space available on the disk drive
       long long     pagesAvail;   // Pages available on the disk drive
       int           ch;           // Input character
       PlainPage_t   plainPage;    // Plaintext page
       CipherPage_t  cipherPage;   // Encrypted page
       size_t        i, j;         // General purpose index
       set<unsigned short> copyNoSet;     // Copy number set
       set<long long>       pageNoSet;    // Page number set
       set<long long>::iterator pageNoIter; // Page number iterator
       long long     pageNo;             // Page number
       size_t        writeCnt;     // Number of bytes written

       // Decode arguments

       if (argc != 2) {
              fprintf (stderr, "Error: usage: hide password < input\n");
              exit (1);
       }
       key = argv[1];

       // Open a temporary file and fill it with random data until the disk
       // is full as indicated by an error on the stream
       // For testing purposes, assume that this happens after 128Mb

       spaceAvail = 0;
#      if 0
              tmpFp = tmpfile ();
#      else
              tmpFp = fopen ("wholedisk", "w+b");
#      endif
```

```
while ( ! ferror (tmpFp)) {
	fputc (GetRandomByte (), tmpFp);
	spaceAvail ++ ;
	if (spaceAvail >= 128*1024*1024) break;
}

// Rewind the temporary file to reset the error flags

rewind (tmpFp);

// Determine the number of ciphertext pages that will fit
// in the temporary file.  Do not use the last 16 pages because
// they might not physically exist.

pagesAvail = (spaceAvail - 16*1024) / sizeof(CipherPage_t);
if (pagesAvail <= 0) {
	fprintf (stderr, "Error: no free pages\n");
	exit (1);
}

// Process each page of input

ch = getchar ();
plainPage.pageNumber = 0;
do {
	// Load the plaintext into the buffer

	i = 0;
	while (ch != EOF && i < PLAIN_DATA_LEN) {
		plainPage.plainData[i] = ch;
		i ++ ;
		ch = getchar ();
	}
	plainPage.plainSize = i;

	// If necessary, pad the plaintext buffer

	while (i < PLAIN_DATA_LEN) {
		plainPage.plainData[i] = GetRandomByte();
		i ++ ;
	}

	// Create the copies of each page

	copyNoSet.clear ();
	for (j = 0; j < PAGE_COPIES; j++) {

		// Allocate a copy number

		do {
			plainPage.copyNumber = GetRandomByte ();
			plainPage.copyNumber <<= 8;
			plainPage.copyNumber |= GetRandomByte ();
		} while (copyNoSet.count (plainPage.copyNumber) != 0);
		copyNoSet.insert (plainPage.copyNumber);

		// Encrypt the page
```

```
                        Encrypt (key, &plainPage, cipherPage.cipherText);

                        // Fill in random data

                        for (i = 0; i < sizeof(cipherPage.randomData); i++)
                                cipherPage.randomData[i] = GetRandomByte();

                        // Sign the page

                        GetSignature (key, cipherPage.cipherText,
                                cipherPage.signature);

                        // Randomly select a free page number

                        if (pageNoSet.size() >= pagesAvail) {
                                fprintf (stderr, "Error: no free pages\n");
                                exit (1);
                        }
                        pageNo = 0;
                        for (i = 0; i < sizeof(pageNo); i++) {
                                pageNo <<= 8;
                                pageNo |= GetRandomByte ();
                        }
                        if (pageNo < 0) pageNo = ~pageNo;
                        pageNo %= pagesAvail - pageNoSet.size();
                        pageNoIter = pageNoSet.begin();
                        while (pageNoIter != pageNoSet.end()) {
                                if (*pageNoIter == pageNo)
                                        pageNo ++ ;
                                pageNoIter ++ ;
                        }
                        pageNoSet.insert (pageNo);

                        // Write the page at the selected position in the
                        // output file

                        if (fseek (tmpFp, pageNo*sizeof(cipherPage), 0) == -1) {
                                fprintf (stderr, "Error: cannot seek page\n");
                                exit (1);
                        }
                        writeCnt = fwrite (&cipherPage, sizeof(cipherPage),
                                1, tmpFp);
                        if (writeCnt != 1) {
                                fprintf (stderr, "Error: cannot write page\n");
                                exit (1);
                        }
                }

                // Move to the next page number

                plainPage.pageNumber ++ ;
        } while (ch != EOF);

        // Close the temporary file (which should delete the file)

        fclose (tmpFp);
        return 0;
}
```