Big iron lessons, Part 5: Introduction to cryptography, from Egypt through Enigma

Why embedding encryption is not easy, or "TKU YHLNGZSMP HKQFDWPCHD DL TUV FTCV"

Sam Siewert (Sam.Siewert@Colorado.edu), Adjunct Professor, University of Colorado

Summary: When is an enigma not an enigma? When it is deciphered. Learn how to make your own paper enigma encoder, then peruse the source code that duplicates its action to understand its simple yet ingenious inner workings -- as well as the basics of good cryptography. Finally, accept the challenge to crack a simple transposition cipher, because all great encryption architects must first master the art of cryptanalysis.

View more content in this series

Date: 26 Jul 2005
Level: Introductory
Activity: 473 views
Comments: 0 (Add comments)


Average rating (based on 52 votes)

If you design systems that require end-to-end encryption or encrypted segments with stored or transported data, you should never underestimate the difficulty of fielding a truly secure, cost- effective, high-performance system. Embedding encryption in devices like automated teller machines, or credit card and point-of-sale systems can be tricky given the significant resource limitations and the vulnerability of these systems, which are often placed in public environments. The National Institute of Standards and Technology (NIST) has defined the FIPS (Federal Information Processing Standards) 140, 140-1, and, most recently, 140-2 standards, to ensure that public encryption for commerce is reasonably secure -- tamper proof, not vulnerable to cryptanalysis attacks over short periods of time, but still amenable to practical implementations.

This article is the first part of a two-part series that continues the Big iron lessons series. This first part provides a historical background on the evolution of ciphers and encryption through World War II, when automated encryption systems began to emerge and usher in the era of digital encryption. The earliest digital encryption was pioneered with an IBM mainframe application known as *Lucifer*. Be sure to read the second part to learn more about the continuing evolution. Working with the simple encryption schemes and practicing cryptanalysis in this first part will help the serious architect understand and appreciate the challenges of encryption and cryptanalysis.

A brief history of cryptography

Natural language ciphers have existed for over 4,000 years, with the Egyptians being the first that we know of to engage in the practice. Not long after the first text was encrypted, the art of decrypting the cipher -- *cryptanalysis* -- emerged. Early ciphers used substitution to replace letters and eventually evolved to also include transpositions of characters as well as simple substitution. Many experts in the field of cryptography have argued that all good cryptographers are also good at cryptanalysis. The idea is that if you have experience with methods of attack and cracking codes, you probably have better ideas

on how to construct codes that can't be cracked.

Radio communications that could be intercepted and the widespread battlefields of World War II brought about electronic eavesdropping and cryptography as a defensive measure to protect critical battlefield communications. The Enigma cryptography machine is the most famous part of this story, and the episode led to fame for Alan Turing, one of the key cryptanalysts who helped crack the Enigma code. Enigma used a series of rotors providing automated substitution and transposition of a series of input characters. (While Turing's work in Britain deciphering Enigma helped bring him fame, many historians also note that much of the work was done by early cryptanalysis in Poland. The Polish were able to capture an Enigma machine, which was also very critical to the code-breaking task. See Resources for Tony Sale's account and this part of the story.)

Cryptography as a strategic weapon

The emergence of Enigma, along with the success of codebreaking and use of ciphers in World War II by the Allies, caused the U.S. government to consider encryption mechanisms and secure signaling to be vital to security. This lead to the 1947 Central Intelligence Act and the secret creation of the NSA (National Security Agency) by President Truman in 1952. Whitfield Diffie and Susan Landau describe the NSA: "From its creation the new organization sought to capture control of all cryptographic and cryptanalytic work within the United States. Overall, this effort was remarkably successful." (Diffie and Landau give an excellent history of government policy related to encryption in their book, *Privacy on the Line;* see Resources for a link.)

Is strong encryption a right?

Philip Zimmermann, inventor of PGP (Pretty Good Privacy) believes strong encryption should be available to all. Zimmermann was under investigation by the U.S. government for over three years after PGP was posted to the Internet. PGP is now distributed by MIT and requires acceptance of a license agreement and U.S. or Canadian citizenship. The U.S. government felt that the posting violated the laws preventing export of munitions. The PGP encryption algorithm is a public key encryption scheme providing both privacy and message authentication.

The stance that encryption methods are weapons, along with the accompanying control on export of computers capable of such encryption, remained in effect until only recently. Given that Enigma gave the Germans a huge advantage of secure surprise attack until its encoding was laboriously cracked, this attitude makes sense. Phil Zimmerman, creator of PGP (Pretty Good Privacy), challenged the U.S. policies limiting use of strong encryption by providing PGP as freeware. Mr. Zimmerman felt that a citizen's right to privacy warranted availability of strong encryption and further felt that much of the encryption available prior to PGP was "snake oil" -- encryption not strong enough to be trusted. (Find more detail on PGP, its status today, and how it works in the continuation of this series.)

One example seems to prove that Zimmerman's scorn for pre-PGP encryption was justified. Early versions of UNIX® used a password encryption scheme based on Enigma -- yes, the same Enigma whose coding was broken in World War II! This remained the case until the release of AT&T UNIX Version 7 in 1978, when `crypt(1)` was upgraded to `crypt(3)`, based on the stronger DES (Date Encryption Standard).

A simple Enigma cipher

Many variations of the basic Enigma machine exist. The simplest involves three encoding rotors, an

input panel, and a reflector panel. The Paper Enigma machine is a fun way to explore encryption concepts and can be downloaded, printed, and cut up to make a hand-operated encoder (see the Resources section below for a link to the Paper Enigma Web site).

The C code in Listing 1 provides the same functionality, based on the same configuration, as the Paper Enigma. I was able to write the code easily after coming to understand Enigma using the paper version. Try the paper machine, and then compare it to the code to gain an appreciation for encryption and to understand how even a simple scheme becomes computationally complex. The example below can encipher or decipher the same example text as the paper machine, but also provides for encryption of files.

Listing 1. Simple three-rotor Enigma encryption

```
#include "stdio.h"
#include "unistd.h"

// Enigma3
//
// Simple C implementation to automate the 3 rotor enigma based upon
the Paper Enigma
// Machine by Michael Koss found at http://mckoss.com/Crypto/Enigma.htm
//
// Will make on Linux, Windows cygwin, or MacOS

#define ALPHA_SIZE 26
#define ALPHA_BASE (int)('A')

char inputFile[80]="Unspecified";
char outFile[80]="Unspecified";
char demoString[]="QMJIDO MZWZJFJR";
char demoEncode[]="ENIGMA REVEALED";
char plainTextLine[256];
char cipherTextLine[256];

typedef struct
{
     char leftChar;
     char rightChar;
} cipherPair_t;

typedef struct
{
     int inPos;
     int outPos;
} refMap_t;

cipherPair_t rotorOne[]={
     {'A', 'E'}, {'B', 'K'}, {'C', 'M'}, {'D', 'F'},
     {'E', 'L'}, {'F', 'G'}, {'G', 'D'}, {'H', 'Q'},
     {'I', 'V'}, {'J', 'Z'}, {'K', 'N'}, {'L', 'T'},
     {'M', 'O'}, {'N', 'W'}, {'O', 'Y'}, {'P', 'H'},
     {'Q', 'X'}, {'R', 'U'}, {'S', 'S'}, {'T', 'P'},
     {'U', 'A'}, {'V', 'I'}, {'W', 'B'}, {'X', 'R'},
     {'Y', 'C'}, {'Z', 'J'}
};
```

```
cipherPair_t rotorTwo[]={
     {'A', 'A'}, {'B', 'J'}, {'C', 'D'}, {'D', 'K'},
     {'E', 'S'}, {'F', 'I'}, {'G', 'R'}, {'H', 'U'},
     {'I', 'X'}, {'J', 'B'}, {'K', 'L'}, {'L', 'H'},
     {'M', 'W'}, {'N', 'T'}, {'O', 'M'}, {'P', 'C'},
     {'Q', 'Q'}, {'R', 'G'}, {'S', 'Z'}, {'T', 'N'},
     {'U', 'P'}, {'V', 'Y'}, {'W', 'F'}, {'X', 'V'},
     {'Y', 'O'}, {'Z', 'E'}
};

cipherPair_t rotorThree[]={
     {'A', 'B'}, {'B', 'D'}, {'C', 'F'}, {'D', 'H'},
     {'E', 'J'}, {'F', 'L'}, {'G', 'C'}, {'H', 'P'},
     {'I', 'R'}, {'J', 'T'}, {'K', 'X'}, {'L', 'V'},
     {'M', 'Z'}, {'N', 'N'}, {'O', 'Y'}, {'P', 'E'},
     {'Q', 'I'}, {'R', 'W'}, {'S', 'G'}, {'T', 'A'},
     {'U', 'K'}, {'V', 'M'}, {'W', 'U'}, {'X', 'S'},
     {'Y', 'Q'}, {'Z', 'O'}
};

refMap_t reflectorMap[]={
     {0,  24}, {1,  17}, {2,  20}, {3,   7},
     {4,  16}, {5,  18}, {6,  11}, {7,   3},
     {8,  15}, {9,  23}, {10, 13}, {11,  6},
     {12, 14}, {13, 10}, {14, 12}, {15,  8},
     {16,  4}, {17,  1}, {18,  5}, {19, 25},
     {20,  2}, {21, 22}, {22, 21}, {23,  9},
     {24,  0}, {25, 19}
};

char refString[]="ABCDEFGDIJKGMKMIEBFTCVVJAT";

int findMatch(cipherPair_t *cmap, char inChar)
{
     int idx;

     for(idx=0;idx %lt; ALPHA_SIZE;idx++)
     {
        if(cmap[idx].rightChar == inChar)
           break;
     }
     return idx;
}

int findDist(char pos1, char pos2)
{
    if(pos2 > pos1)
       return (pos2 - pos1);
    else
       return ((ALPHA_SIZE - pos1) + pos2);
}


int main( int argc, char *argv[] )
{
     int demoRun=0;
     int dbgOn=0, foundEnd=0;
     FILE *fpIn, *fpOut;
     int lineSz=128;
     int bytesRd=0;
```

```
     // Initial conditions for machine
     int r1Idx=12, r2Idx=2, r3Idx=10, inputIdx=0, idx, rIdx, rDiff;
     int notch1Idx=16, notch2Idx=4, notch3Idx=21;

     if(argc == 1 || argc == 2)
     {
        demoRun=1;
        printf("Will run simple demo\n");
        if(argc == 2 && (strncmp(argv[1],"-D", 2) == 0))
        {
           dbgOn=1;
        }
     }
     else if(argc >= 3)
     {
        strcpy(inputFile, argv[1]);
        strcpy(outFile, argv[2]);
        printf("Will run Enigma3 encode on input file=%s and produce
file=%s\n",
               inputFile, outFile);

        if(argc == 4 && (strncmp(argv[3],"-D", 2) == 0))
        {
           dbgOn=1;
        }
     }
     else
     {
        printf("Usage: enigma3 file-to-encode outfile [-D]\n");
        exit(-1);
     }

     if(demoRun)
     {
        strcpy(plainTextLine, demoEncode);
        foundEnd = 1;
     }
     else
     {
        if( (fpIn=fopen(inputFile, "r")) == (FILE *)0)
        {
           printf("ERROR: can't open input file\n");
           exit(-1);
        }
        if( (fpOut=fopen(outFile, "w")) == (FILE *)0)
        {
           printf("ERROR: can't open output file\n");
           exit(-1);
        }

        if((fgets(plainTextLine, lineSz, fpIn)) == (char *)0)
           exit(-1);
     }

     do
     {
        for(idx=0;idx %lt; strlen(plainTextLine);idx++)
        {
```

```
            // Ensure that string is uppercase
            plainTextLine[idx] = (char)toupper((char)plainTextLine
[idx]);

            // just copy over all non-alpha characters in string
            if(plainTextLine[idx] %lt; 'A' || plainTextLine[idx] > 'Z')
            {
               cipherTextLine[idx] = plainTextLine[idx];
               continue;
            }

            inputIdx = ((int)plainTextLine[idx]) - ALPHA_BASE;

            // Rotor adjust before encode/decode
            //
            r3Idx++; // rotate Right rotor
            r3Idx %= ALPHA_SIZE;

            if(dbgOn)
            {
               printf("%c, %c, %c, input=%c\n",
                        (char)(ALPHA_BASE +  r1Idx), (char)(ALPHA_BASE
+ r2Idx),
                        (char)(ALPHA_BASE + r3Idx), (char)(ALPHA_BASE
+  inputIdx));
            }

            // select r3 pair input
            rIdx = (inputIdx + r3Idx) % ALPHA_SIZE;
            if(dbgOn) printf("R3: %c, ", (char)(ALPHA_BASE + rIdx));

            // select r3 pair output
            rIdx = (rotorThree[rIdx].rightChar) - ALPHA_BASE;
            if(dbgOn) printf("%c\n", (char)(ALPHA_BASE + rIdx));

            // select r2 pair input
            rIdx = (r2Idx + findDist(r3Idx, rIdx)) % ALPHA_SIZE;
            if(dbgOn) printf("R2: %c, ", (char)(ALPHA_BASE + rIdx));

            // select r2 pair output
            rIdx = (rotorTwo[rIdx].rightChar) - ALPHA_BASE;
            if(dbgOn) printf("%c\n", (char)(ALPHA_BASE + rIdx));

            // select r1 pair input
            rIdx = (r1Idx + findDist(r2Idx, rIdx)) % ALPHA_SIZE;
            if(dbgOn) printf("R1: %c, ", (char)(ALPHA_BASE + rIdx));

            // select r1 pair output
            rIdx = (rotorOne[rIdx].rightChar) - ALPHA_BASE;
            if(dbgOn) printf("%c\n", (char)(ALPHA_BASE + rIdx));

            // select reflector input
            rIdx = (findDist(r1Idx, rIdx)) % ALPHA_SIZE;
            if(dbgOn) printf("REF: %c, ", refString[rIdx]);

            // select reflector output
            rIdx = reflectorMap[rIdx].outPos;
            if(dbgOn) printf("%c\n", refString[rIdx]);

            // select r1 pair input
```

```
            rIdx = (r1Idx + rIdx) % ALPHA_SIZE;
            if(dbgOn) printf("R1: %c, ", (char)(ALPHA_BASE + rIdx));

            // select r1 pair output
            rIdx = findMatch(&rotorOne[0], (char)(ALPHA_BASE +
rIdx));
            if(dbgOn) printf("%c\n", (char)(ALPHA_BASE + rIdx));

            // select r2 pair input
            rIdx = (r2Idx + findDist(r1Idx, rIdx)) % ALPHA_SIZE;
            if(dbgOn) printf("R2: %c, ", (char)(ALPHA_BASE + rIdx));

            // select r2 pair output
            rIdx = findMatch(&rotorTwo[0], (char)(ALPHA_BASE +
rIdx));
            if(dbgOn) printf("%c\n", (char)(ALPHA_BASE + rIdx));

            // select r3 pair output - final output
            rIdx = (r3Idx + findDist(r2Idx, rIdx)) % ALPHA_SIZE;
            if(dbgOn) printf("R3: %c, ", (char)(ALPHA_BASE + rIdx));

            rIdx = findMatch(&rotorThree[0], (char)(ALPHA_BASE +
rIdx));
            if(dbgOn) printf("%c\n", (char)(ALPHA_BASE + rIdx));

            cipherTextLine[idx] = ALPHA_BASE + (findDist(r3Idx,
rIdx)  % ALPHA_SIZE);
            if(dbgOn) printf("output=%c\n", cipherTextLine[idx]);

            // rotate other two rotors if needed
            if(r3Idx == notch3Idx)
            {
               r2Idx++; // rotate Middle rotor

               if(r2Idx == notch2Idx)
               {
                  r1Idx++;
               }
            }
            r2Idx %= ALPHA_SIZE; r1Idx %= ALPHA_SIZE;
        }

        cipherTextLine[idx] = '\0';
        printf("Input = %s\n", plainTextLine); printf("Output = %s
\n",  cipherTextLine);

        if(!foundEnd)
        {
           fputs(cipherTextLine, fpOut);

           if((fgets(plainTextLine, lineSz, fpIn)) == (char *)0)
              foundEnd=1;
        }

     } while (!foundEnd);
}
```

I used the three-rotor Enigma machine whose logic was encapsulated in Listing 1 to encrypt file contents into an encrypted version, as shown below:

```
$ ./enigma3 plaintext.in cipher.out
Will run Enigma3 encode on input file=plaintext.in and produce
file=cipher.out
Input = ENIGMA REVEALED
Output = QMJIDO MZWZJFJR
Input = THE QUICK BROWN FOX JUMPED OVER THE FENCE
Output = YNF PVZNL JFHQX BUP XOSHNM GMSE DRM YKDDX
Input = WHY WAS ALAN TURING SO IMPORTANT TO THE EFFORT TO CRACK ENIGMA?
Output = ZCP BZV DQQS XWUBZV GX AWEBKYPWL BV HGA AREGCE VY IJXQS ZKFRQZ?
Input = WHY EMBEDDING ENCRYPTION IS NOT EASY
Output = TKU YHLNGZSMP HKQFDWPCHD DL TUV FTCV
```

A fun challenge: Break my simple cipher

Try to decipher the string below, which I generated with a simple substitution encryption scheme. It can be deciphered into an English sentence. It is a private key symmetric (single key) encryption method using a well-known algorithm. Be aware that this is not intended to be hard to crack, and is not strong encryption (I'm not trying to sell any snake oil). However, it should provide a nice challenge for someone new to cryptanalysis.

```
JI HXY EZBDN UIRE JJHWVTCPG ZGHVSH YPBT QU NY PCFPY
```

If you give up, you can download the solution code from the Downloads section.

A look ahead

As you can see from the code listing in this article, even a simple encryption scheme like Enigma can be computationally quite complex. In the next part of this series, you'll see just how complex cryptography can get, and see how a powerful coprocessor can help any application that needs to use crypto.

Downloads

Download the following code to test out the concepts discussed here:

- enigma3.c is the C code in Listing 1 that encapsulates the functionality of the Paper Enigma. You can also download a makefile for this code, along with input (plaintext.in) and output (cipher.out) examples.
- If you can't figure out the challenge, download prgcoding.c, which I used to encode the text, along with its accompanying makefile.

All C code was tested on Linux™, Mac OS X, and Microsoft® Windows with Cygwin.

Resources

- The Paper Enigma Machine can be downloaded and printed for a simple hand encryption tool that

will help you better understand Enigma encoding.

- The IPsec standards, developed through the IETF (Internet Engineering Task Force), are a collection of standards all related to IP security and cryptography.
- Check out this information on finding good random number generators, which are essential to encryption. You can learn more from this ACM paper, Random number generators: Good ones are hard to find, S.K. Park and K.W. Miller (October 1988).
- Applied Cryptography, Bruce Schneier (John Wiley and Sons, 1996) is one of the most comprehensive sources on encryption algorithms and theory.
- For historical information on Bletchley Park and Alan Turing's role in cracking Enigma, look into Tony Sale's Codes and Ciphers Web site in the U.K. The Enigma Code Breach by Jan Bury provides an excellent historical account including pictures and numerous details of the cryptanalysis done in Poland.
- See The Enigma Machine for a student-designed Java™ applet that simulates an Enigma machine with a nice graphical interface. Also find an Enigma machine written in Visual Basic.
- Privacy on the Line: The Politics of Wiretapping and Encryption, Whitfield Diffie and Susan Landau (MIT Press, 1999), provides a great historical overview of both the development of encryption technology and the U.S. government policies and organizations that have exercised legal controls and influence on the technologies.
- Philip Zimmermann's essay Beware of snake oil provides insight into what constitutes good encryption and why PGP got him into trouble with U.S. Customs. The PGP freeware version should only be downloaded by U.S. or Canadian citizens meeting all conditions required on the PGP download site at MIT. While PGP freeware is still available, it is not intended for commercial use; using it for commercial purposes is a violation of its license agreement. For commercial use, see the PGP Corporation's home page.
- The mcrypt open source password encryption program replaces the original UNIX `crypt` based upon Enigma, deprecated with AT&T UNIX Version 6 as well as crypt(3) introduced in AT&T UNIX Version 7, based upon the DES (Data Encryption Standard). Both can still be found in use on UNIX systems (especially `crypt(3)`), but both are highly susceptible to cryptanalysis attack by crack programs such as John The Ripper. Cracking programs have legitimate uses, such as testing password quality to test system security and for recovery of lost passwords. The Wikipedia page on cryptanalysis gives a great overview of the history of ciphers, deciphering, encryption, and cryptanalysis methods.
- Learn more about the Data Encryption Standard, or DES, including how the EFF (Electronic Frontier Foundation) cracked DES in 1998.

About the author

Dr. Sam Siewert is an embedded system design and firmware engineer who has worked in the aerospace, telecommunications, and storage industries. He also teaches at the University of Colorado at Boulder part-time in the Embedded Systems Certification Program, which he co-founded. His research interests include autonomic computing, firmware/hardware co-design, microprocessor/SoC architecture, and embedded real-time systems.

Trademarks