

Forum: [Hack Forums](#)

Thread: [\[Dynasty\] eLSB Steganography + Asymmetric Backdoors in Kleptography & Cryptovirology](#)

Keeper - October 18, 2013, 1:17 pm

Been quite some time since I last posted something stego oriented so here it goes. In this one I'm gonna explain a bit of in-depth steganography (as a matter of fact LSB shifting is considered a novice stage but not in the case of the tutorial) and sketchily go over basic kleptography aspects as well as cryptovirology ones as well. The tutorial is somewhat related to my old MLE one but with more advanced and major concepts being implemented in comparison to it. I had a notion of combining everything together and forming a unique tool or rather a virus but it's too complex for me so I'll just break down the information into different topics and sum it up in a single project that includes all the below listed stuff. As a matter of fact here's what I am gonna layout in it:

- Multiple hybrid layers (concealment)
- Steganography (enhanced LSB shifting)
- Kleptography (asymmetric backdoors)
- Cryptovirology (cryptotrojans)
- Python
- Esoteric language (Brainfuck/INTERCAL)
- Vigenere cipher
- Self-modifying source
- Polymorphism
- 2D models & Spectrums
- Temporal frequency analyses
- Graphical schemes (graphs)
- Datamatrix/QRcodes

Unlike the previous tutorial on this, here we're gonna develop a quintuple layer. The idea is to cover the above mentioned includes and especially the kleptographic attack where we're gonna spawn an asymmetric backdoor. It's in fact a monolayer but I'm gonna divide it into several sub-levels. We're gonna plant our cryptotrojan in a subliminal channel within the last sub-layer and then the next layer will take care of any eavesdropping or detection as a whole followed by three more for extra security and finalizing with a noised multi-pixel image. Here's a scheme of what each of them will represent:

So before we start lemme just mention the abbreviation of LSB which stands for least significant bit (this is basically the last bit in the sequence - rightmost). The fact that I wanted to meddle LSB steganography in this tutorial is mainly due to the fact that it's quite often mistaken with quantum steganography and images with a high level of noise. I myself had trouble solving a mission on HTS which has the same layer in the first place. The method I'm gonna apply is known as enhanced LSB shifting. The idea here is that each pixel is being replaced with a different value and hence makes the image totally unrecognizable. It is called enhanced because we're eliminated the high-level bits

for every pixel except for the last LSB one and this is the case where we can most often evaluate the layer by looking into the structure of an image and following let's say an IDAT of 9 blocks, last LSB will be either smaller or equal to the previous bits (rarely equal in fact) which means that the previous ones have been altered and there's literally no room for the last LSB.

One of the few techniques that can be used to detect eLSB steganography (and actually differentiate it from quantums) is statistical analyses. The most famous of which is the chi-square attack. There are a few tools to perform such analysis. I'm gonna use a Java implementation of one of them (not written by me). The first chi-square graph is the output of the analysis made on the enhanced image and the one below it - on the non-enhanced. For the record, the chi-square is calculated on 24-bit .BMPs. It's also known as steganalysis.

Lemme quote the author of the tool as to what the different curves represent:

Quote: The program will output a graph with two curves. The first one in red is the result of the chi-square test. If it's close to one, then the probability for a random embedded message is high. So, if there is a random message embedded, this green curve will stay around 0.5. On the graph network, every vertical blue line represents 1 kilobyte of embedded data.

This is a sample representation of how the LSBs are being enhanced and set to either 255 or 0. Basically, the noise level depends on how much data we want to steganography and of course the size of the image, the color capacity etc.

Below is a histogram of the image. The blue initiates with a high value and then settles to a certain level. Whenever something like this occurs after you analyses, you could be nearly certain that there is something steganographed. Magically said, there is somewhat 70%+ in which I've found something in an image after a histogram analysis which outputs such results (especially the heightened presence of one of the RGB values).

Here's a Java chi-square class. I've used this one to represent the output of the steganalysis above. Here is a direct [link to the .jar](#) if you happen to need it for testing purposes only and not going through the code.

```
Code public double[] getExpected() {
    double[] result = new double[pov.length / 2];
    for (int i = 0; i < result.length; i++) {
        double avg = (pov[2 * i] + pov[2 * i + 1]) / 2;
        result[i] = avg;
    }
    return result;
}

public void incPov(int i) {
    pov[i]++;
}

public long[] getPov() {
    long[] result = new long[pov.length / 2];
    for (int i = 0; i < result.length; i++) {
```

```
    result[i] = pov[2 * i + 1];  
}  
return result;
```

Frankly speaking there is a lot of software for embedding and extracting data but none is actually efficient when it comes to reversing the process. In this case the only possible way to reverse it will be to pull the least significant bit from every pixel channel, group them into words of 8 bits and convert back to text but that'd only be possible if we had any clue on which pixels have been altered (which we do not possess).

But let's say there is some sound or whatever audio file meddled. If you're good enough with steganography you could mix up both eLSB and audio rendering of an image and come up with an incredibly secure layer. Consider we have a file called fuke.wav which is somewhat altered and has some data within it. One of the ways to check for anything specific or whatever is to put the file under a frequency analysis and see whether there is something worth pursuing. First let's see a temporal analysis alongside a TFFT. Actually, the only difference between a FFT and a temporal analysis is that the TFFT studies both the time and frequency of the signal while the FFT one only the signal itself (in other words we need to define a spectrum in order to see the temporal frequencies).

If that doesn't suit you, you can use sox for Linux boxes and generate a similar spectrum. Note that sox works only with .wav files (which is pretty much the extension that most software worships). Now to output a spectrum we do the following:

```
Code sox fuke.wav -n spectrogram
```

You may have to use a converter like ffmpeg or similar to alter the extension if you have previously generated a different one than .wav. And so we end up with the following:

Similar to that spectrogram are the following. The first one of which is with a dBV² scale on a 1024-bit window at 85%+- and the second one a linear scale and a 2048-bit window at 90%+ with a log bin. Quite better visible as you can plainly see. Same would refer if we manage to scale the sox spectrogram and manipulate it as best as we can but I don't think sox offers such possibilities.

Ok, so much for our first layer - four more to go. Now let's get to the next one which is the datamatrix.

Let's start off with a bit of a challenge walkthrough so that we can get the the essence of this layer. So considering we have the following PHP file:

```
PHP Code  
<?php  
http://hastebin.com/ritecekafo.hs
```

So here we are supposed to have something embedded. First of all this is a normal image file just formatted into PHP, so let's save it as .PNG for instance and see what we come up with.

Using a converter we end up with the following string which is in fact a polyalphabetic substitution cipher (vigenere in this case).

Code

```
dtsfwqutisvqtesymkuvabbujwhfecuvlshwopcyeghguywjvlaibflcacyahckyqvypjntfhihgtvyxeqakjwouldlt  
uiuhbhjumgkxuugqahvwhotduqtahcknheypjetxpvlhxtlrpjagyjzcgijgfmcupsslkzpuxegaillytlfbygeptzjtuz  
lvlwkdznxqwpabbe
```

There are lots of applications for 1D barcodes, PDF417, datamatrixes, QRs and whatnot. So far [this one](#) does the job the best for me.

Let's start off with the source. So far it's not polymorphic just basic encryption functions:

Code #Vigenere Cipher Encrypter
#Programmed by Mateo Zoto

```
LETTERTOTAL = 26  
UPPERCASE_ASCII_START = 65  
LOWERCASE_ASCII_START = 97  
UPPERCASE_ASCII_END = 90  
LOWERCASE_ASCII_END = 122
```

#Function first displays a line of instruction to the user. Then it will ask for the user to input a key.
Runs a try in order to make sure that the key the user

#is inputting is actually alphabetical. Also checks if the user is actually inputting anything. For
example a user could be entering in nothing and it will throw

#a KeyError at the user. Throws a keyerror if the user tries to enter in anything thats not
alphabetical as well. Key is returned as lower in order to prevent

#any unexpected outcomes.

```
def getKey():  
    print ("Please enter the following requested inputs in order to encrypt your letter with a Vigenere  
cipher.")  
    print ("")  
    valid = False  
    while (not valid):  
        try:  
            key = raw_input("Please enter a key: ")  
            if len(key) == 0 or not key.isalpha():  
                raise KeyError("Key Error: Your key is not valid. Please enter a valid key which consists of ONLY  
alphabetic letters.")  
            valid = True  
        except KeyError:  
            print ("Key Error: Your key is not valid. Please enter a valid key which consists of ONLY  
alphabetic letters.")  
            return key.lower()
```

#Asks the user to input the word or sentence that they would like to see encrypted. Runs a check to
see if they actually entered anything or not.

```
def getPlaintext():
```

```

plaintext = raw_input("Please enter a word of your choice that you would like to get encrypted: ")
while len(plaintext) == 0:
plaintext = raw_input("Please enter a word of your choice that you would like to get encrypted: ")
return plaintext

```

```

#Where the actual encryption takes place. Key refreshes itself by the use of the for loop variable n
and a modulo of the length of the key. Checks to see if
#each separate string is upper or lower case or if it's
#something else such as a space. If the string is a space or anything not alphabetical it will simply
just leave it as it is whereas uppercase and lowercase
#are handled differently. I use += to combine the split strings into one sentence. I subtract by the
lowercase ascii value as that seems to work for whatever reason
#for both uppercase and lowercase. by -= i make sure the ord value stays within 26 values of the
uppercase lettering.

```

```

def doEncryption(key, plaintext):
    crypt = ""
    for n in range(0, len(plaintext), 1):
        if plaintext[n].isupper():
            upper_new = ord(plaintext[n]) + ord(key[n%len(key)]) - LOWERCASE_ASCII_START
            if (upper_new > UPPERCASE_ASCII_END) or (upper_new < UPPERCASE_ASCII_START):
                upper_new -= LETTERTOTAL
            #adds all uppercase letters into the resulting sentence in order with respect to lowercase and
            special characters/white space
            #also converts back from ascii to alpha
            crypt += chr(upper_new)
        elif plaintext[n].islower():
            lower_new = ord(plaintext[n]) + ord(key[n%len(key)]) - LOWERCASE_ASCII_START
            if (lower_new > LOWERCASE_ASCII_END) or (lower_new < LOWERCASE_ASCII_START):
                lower_new -= LETTERTOTAL
            #adds all lowercase letters into the resulting sentence in order with respect to the uppercase and
            special characters/whitespace
            #also converts back from ascii to alpha
            crypt += chr(lower_new)
        else:
            #adds all special characters/white space with respect to upper and lowercase letters. doesn't
            actually do anything to the white spaces so the
            #key just skips white spaces or anything not alphabetical.
            else_new = plaintext[n]
            crypt += else_new
    return crypt

```

```

#Displays the result to the user.

```

```

def display(crypt):
    print("The encrypted outcome of your inputted sentence or word is: ", crypt)

```

```

#God function, defining the order of the functions the program should run once started.

```

```

def main():
    key = getKey()
    plaintext = getPlaintext()
    crypt = doEncryption(key,plaintext)
    display(crypt)

```

```

#Where the program starts

```

main()

Little bit of background. Vigenere is a polyalphabetic cipher. Meaning that all attacks against such substitutions are applicable against it as well. I won't be explaining how to crack a vigenere since that's quite easily findable on the web. Anyway one of the key things is to estimate the key length. Below are the steps you need to take to manually crack a vigenere.

Code 1. Find long repeated sequences

2. Find often repeated sequences

3. Count the distances between the repeats

4. Work out which factors are common among these distances

5. Start by assuming a plausible keyword length and write down each letter on the position

6. Analyze the letters as if it were a Caesar's Shift

7. The current letter of the keyword will be the first letter of the Caesar's Shift cipher alphabet used

8. Repeat step 5 with the next position until you processed all letters or can make out the keyword

Those would include Kasiski examination and a lot of other cryptanalyses. I won't be narrating Phizo's words so here is what he advised me when I sought his help:

Quote:Polymorphism is dependent on the key alone, in your example. Notice that the ciphertext will not produce polymorphic results when the key consists of nothing but the same characters. Also notice that the actual key will be visible in the plaintext if the first character is used for the entire key-length. Imagine the decimal value of a character in your plaintext, multiplied by a completely different random decimal value of a character. To make the key of any importance, you're going to want to make it depend on the key (such as the one in the example you've shown), but with changes that will prevent the issues I pointed out earlier. For example, having the key be an operand in the multiplication along with random values, making the key and the resulting ciphertext polymorphic.

We'll start off with my favourite one - Malbolge. Practically, every esoteric language shouldn't be viewed that much from a programming point but rather a cryptographic one. Those languages are intentionally made sophisticated so that they hinder the writing of any code and obfuscate it immensely. Initially, they were invented to see how far can programming be extended and how many different ways of writing code can be implemented really. Malbolge is considered to be the hardest to write in simply because it's the most perplex of all of em. There are only three programs written in it ever since 1998 so yeah. We're not gonna write another one simply because we won't be able to but what the point here is to output the vigenere or whatever ciphertext we need using an esoteric language. So first off the 'accessories':

Interpreter:

Code #include <stdio.h>

const char* translation =

"5z]&gqtyfr\$(we4{WP)H-Zn,[%\3dL+Q;>U!pJS72FhOA1CB6v^=I_0/8]jsb9m<.TVac`uY*MK'X~xDI}R
EokN:#?G\"i@";

unsigned int crazy(unsigned int a, unsigned int d){

unsigned int crz[] = {1,0,0,1,0,2,2,1};

int position = 0;

unsigned int output = 0;

while (position < 10){

unsigned int i = a%3;

unsigned int j = d%3;

unsigned int out = crz[i+3*j];

```

    unsigned int multiple = 1;
    int k;
    for (k=0;k<position;k++)
        multiple *= 3;
    output += multiple*out;
    a /= 3;
    d /= 3;
    position++;
}
return output;
}

```

```

unsigned int rotateR(unsigned int d){
    unsigned int carry = d%3;
    d /= 3;
    d += 19683 * carry;
    return d;
}

```

```

int main(int argc, char* argv[]) {
    unsigned int memory[59050];
    unsigned int a, c, d, result;
    FILE* file;
    if (argc < 2) {
        // read program code from STDIN
        file = stdin;
    }else{
        file = fopen(argv[1],"rb");
    }
    if (file == NULL) {
        fprintf(stderr, "File not found: %s\n",argv[1]);
        return -1;
    }
    a=0;
    c=0;
    d=0;
    result = 0;
    while (!feof(file) && d < 59050){
        unsigned int instr;
        memory[d] = 0;
        result = fread(memory+d,1,1,file);
        if (result > 1)
            return -1;
        if (result == 0 || memory[d] == 0x1a || memory[d] == 0x04)
            break;
        instr = (memory[d] + d)%94;
        if (memory[d]==' ' || memory[d] == '\t' || memory[d] == '\r' || memory[d] == '\n');
        else if (memory[d] >= 33 && memory[d] < 127 && (instr == 4 || instr == 5 || instr == 23 || instr
== 39 || instr == 40 || instr == 62 || instr == 68 || instr == 81))
            d+=result;
        else{
            fprintf(stderr, "Invalid character 0x%02x at 0x%05x.\n",(char)memory[d],d);
            return -1; //invalid characters are not accepted.
        }
    }
}

```

```

    //that makes the "hacked" in-out-program unrunnable
}
}
if (file != stdin) {
    fclose(file);
}
if (d == 59050) {
    fprintf(stderr, "Maximum program length of 59049 exceeded.\n");
    return -1;
}
if (d < 2) {
    fprintf(stderr, "Minimal program length of 2 deceeded.\n");
    return -1;
}

while (d < 59049){
    memory[d] = crazy(memory[d-1], memory[d-2]);
    d++;
}
d = 0;

// execute program
while (1) {
    unsigned int instruction = memory[c];
    if (instruction < 33 || instruction > 126) {
        fprintf(stderr, "Invalid command 0x%05x at 0x%05x.\n", instruction, c);
        return -1;
    }
    instruction = (instruction+c)%94;
    switch (instruction){
        case 4:
            c = memory[d];
            break;
        case 5:
            {
                printf("%c", (char)(a));
            }
            break;
        case 23:
            a = getchar();
            if (a == EOF)
                a = 59048;
            break;
        case 39:
            a = (memory[d] = rotateR(memory[d]));
            break;
        case 40:
            d = memory[d];
            break;
        case 62:
            a = (memory[d] = crazy(a, memory[d]));
            break;
        case 81:

```

```

        return 0;
    case 68:
    default:
        break;
    }
    // if memory[c] has been modified by the command above bring it into valid range
    if (memory[c] < 33)
        memory[c] += 94;
    memory[c]-=33;
    if (memory[c] > 93)
        memory[c] %= 94;
    // encrypt command
    memory[c] = translation[memory[c]];

    c = (c+1)%59049;
    d = (d+1)%59049;
}
return 0;
}

```

Assembler --> [here](#)

Basic Malbolge generator:

Code #include <stdio.h>

#include <string.h>

#include <malloc.h>

```
unsigned int shortRotCrazyPrint(unsigned int C, char goal);
```

```
//first character: b -> cause a jump to address 98 which will be encrypted first
// then execution continues with data pointer at 1 and program counter at 99
```

```
const char* translation =
```

```
"5z]&gqtyfr$(we4{WP)H-Zn,[%\3dL+Q;>U!pJS72FhOA1CB6v^=I_0/8]jsb9m<.TVac`uY*MK'X~xDI}R
EokN:#?G\"i@";
```

```
unsigned int crazy(unsigned int a, unsigned int d){
    unsigned int crz[] = {1,0,0,1,0,2,2,2,1};
    int position = 0;
    unsigned int output = 0;
    while (position < 10){
        unsigned int i = a%3;
        unsigned int j = d%3;
        unsigned int out = crz[i+3*j];
        unsigned int multiple = 1;
        int k;
        for (k=0;k<position;k++)
            multiple *= 3;
        output += multiple*out;
        a /= 3;
        d /= 3;
        position++;
    }
    return output;
}
```

```
}
```

```
unsigned int rotateR(unsigned int d){  
    unsigned int carry = d%3;  
    d /= 3;  
    d += 19683 * carry;  
    return d;  
}
```

```
unsigned int memory[59050]; // program  
unsigned int memory_runtime[59050]; // memory at runtime: with encrypted commands  
unsigned int lastAval = 0; // last value of A register
```

```
int main(int a, char* b[]){  
    int outputCounter = 0;  
    int i;  
    unsigned char* out_s;  
    int C;  
    int firstIt;  
    int stop;  
    if (a > 1){  
        int pos = 0;  
        int sameLen = 0;  
        for (i=1;i<a;i++){  
            sameLen += strlen(b[i])+1;  
        }  
        if (sameLen == 0)  
            sameLen++;  
        out_s = (unsigned char*)(malloc(sameLen*sizeof(unsigned char)));  
        for (i=1;i<a;i++){  
            memcpy(out_s+pos,b[i],strlen(b[i]));  
            pos+=strlen(b[i]);  
            out_s[pos]=' '  
            pos++;  
        }  
        if (pos>0)  
            pos--;  
        out_s[pos]=0;  
    }  
    else{  
        // READ STRING FROM STDIN  
        // maximal 10.000 characters!  
        int pos = 0;  
        unsigned int result = 0;  
        out_s = (unsigned char*)malloc(10001*sizeof(unsigned char));  
        if (stdin == 0){  
            printf("Cannot access STDIN.\n");  
            return -1;  
        }  
        while (!feof(stdin) && pos < 10000){  
            result = fread(out_s+pos,1,1,stdin);  
            if (result<0){  
                printf("Error while reading from STDIN.\n");  
            }  
        }  
    }  
}
```

```

        return -1;
    }
    if (result==0)
        break;
    pos++;
}
out_s[pos]=0;
}

```

```
memory[0] = 'b'; //jmp to 99
```

```

for (i=1;i<99;i++){
    int nop = (68-(i%94)+94)%94;
    if (nop<33) nop+=94;
    memory[i]=nop;
}
memcpy(memory_runtime,memory,99*sizeof(unsigned int));
memory_runtime[98] = translation[memory[98]-33];
C = 99;
firstIt = 1;
while (*out_s != 0){
    int inc;

    // if character is still in A register just print it out
    if (!firstIt){
        if (*out_s == *(out_s-1)){
            if (C+1 >= 59049){
                goto finish;
            }else{
                int out = (5-C%94+94)%94;
                if (out<33) out+=94;
                memory[C] = out;
                memory_runtime[C] = translation[out-33];
                C++;
                out_s++;
                continue;
            }
        }
    }
}

```

```
// load character to A register
```

```

do {
    C += (inc = shortRotCrazyPrint(C, *out_s));
    if (inc==0)
        out_s++;
}while(inc == 0 && *out_s!=0);
if (inc == 0)
    break;
if (C+1 >= 59049){
    C-=inc;
    goto finish;
}else{
    // print A register

```

```

    int out = (5-C%94+94)%94;
    if (out<33) out+=94;
    memory[C] = out;
    memory_runtime[C] = translation[out-33];
    C++;
}
firstIt = 0;
out_s++;
}

```

finish:

```

    while (C+1 >= 59049){
        C-=1;
    }
    // halt
    stop = (81-C%94+94)%94;
    if (stop<33) stop+=94;
    memory[C] = stop;
    memory_runtime[C] = translation[stop-33];
    C++;
    for (i=0;i<C;i++){
        printf("%c", (char)memory[i]);
    }
    printf("\n");

    return 0;
}

```

unsigned int shortRotCrazyPrint(unsigned int C, char goal) {

```

    int maxwidth = 1;
    int i;

```

```

    if (((unsigned char)goal >= 154 && (unsigned char)goal <= 208) || goal == 43)

```

return 0; //TODO: 43 ('+') is possible with one shiftR and 2 crazies. At least a not-efficient + should be implemented

```

do{

```

```

    int rotPos = 0;
    int crazyInsteadRot = 0;

```

```

    if (maxwidth > 1000){
        // this should not occur
        return 0;
    }

```

```

    if (maxwidth + C >= 59049){
        //maximum space exceeded
        return 0;
    }

```

```

do{
    unsigned int shift_val;

```

```

unsigned int result;

for (i=C;i<C+rotPos;i++){
    int nop = (68-i%94+94)%94;
    if (nop<33) nop+=94;
    memory[i]=nop;
    if (memory[i] >= 33 && memory[i] <= 126)
if (i >= 98)
memory_runtime[i] = translation[memory[i]-33];
else
memory_runtime[i] = memory[i];
}

if (!crazyInsteadRot){
    int shift = (39-(C+rotPos)%94+94)%94;
    if (shift<33) shift+=94;
    shift_val = memory_runtime[C+rotPos-98];

    memory[C+rotPos] = shift;
    if (C+rotPos >= 98)
memory_runtime[C+rotPos] = translation[memory[C+rotPos]-33];
    else
memory_runtime[C+rotPos] = memory[C+rotPos];
}else{
    int cr = (62-(C+rotPos)%94+94)%94;
    if (cr<33) cr+=94;
    shift_val = memory_runtime[C+rotPos-98];

    memory[C+rotPos] = cr;
    if (C+rotPos >= 98)
memory_runtime[C+rotPos] = translation[memory[C+rotPos]-33];
    else
memory_runtime[C+rotPos] = memory[C+rotPos];
}

for (i=C+rotPos+1;i<(int)(C+maxwidth);i++){
    int nop = (68-i%94+94)%94;
    if (nop<33) nop+=94;
    memory[i]=nop;
    if (memory[i] >= 33 && memory[i] <= 126)
if (i >= 98)
memory_runtime[i] = translation[memory[i]-33];
else
memory_runtime[i] = memory[i];
}

if (rotPos + 1 < maxwidth){
    int crzy = (62-(C+maxwidth-1)%94+94)%94;
    unsigned int crzy_val = memory_runtime[C+maxwidth-1-98];
    if (crzy<33) crzy+=94;

    memory[C+maxwidth-1] = crzy;
    if (C+maxwidth-1 >= 98)

```

```

memory_runtime[C+maxwidth-1] = translation[memory[C+maxwidth-1]-33];
    else
memory_runtime[C+maxwidth-1] = memory[C+maxwidth-1];

    if (!crazyInsteadRot)
result = crazy(rotateR(shift_val), crzy_val);
    else
result = crazy(crazy(lastAval,shift_val), crzy_val);

    }else{
        if (!crazyInsteadRot)
result = rotateR(shift_val);
        else
result = crazy(lastAval,shift_val);
    }

    if ((unsigned char)result == (unsigned char)goal){
        //DONE!
        lastAval = result;
        return maxwidth;
    }

    if (crazyInsteadRot){
        rotPos++;
        crazyInsteadRot = 0;
    }else
        crazyInsteadRot = 1;
    }while (rotPos < maxwidth);

    maxwidth++;
}while(1);

}

```

Now we need to need to spawn a cryptotrojan in a subliminal channel through kleptography. The cryptotrojan is in fact gonna be an asymmetric backdoor that would append let's say a series of private keys coming from an internal system for encryption or similar into an array which in the scope of a temporary session - so we're getting only the last set of keys used. To start off with we'll need a cryptographic environment which we're gonna audit and plant a cryptovirological cryptotrojan. Our underlying cryptosystem will be simply encrypting a set of obviously fake missile launch codes just for the record. They'll be representing a block cipher and gonna get parsed through the RSA algorithm and stored in a file. Let's see how this is being done.

The following is a C code to implement the RSA encryption algorithm:

```

Code #include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>

long int p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
char msg[100];

```

```

int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
void main()
{
    clrscr();
    printf("\nEnter first prime number\n");
    scanf("%d",&p);
    flag=prime(p);
    if(flag==0)
    {
        printf("\nWrong input\n");
        getch();
        exit(1);
    }
    printf("\nEnter another prime number\n");
    scanf("%d",&q);
    flag=prime(q);
    if(flag==0||p==q)
    {
        printf("\nWrong input\n");
        getch();
        exit(1);
    }
    printf("\nEnter message\n");
    fflush(stdin);
    scanf("%s",msg);
    for(i=0;msg[i]!=NULL;i++)
    m[i]=msg[i];
    n=p*q;
    t=(p-1)*(q-1);
    ce();
    printf("\nPossible values of e and d are\n");
    for(i=0;i<j-1;i++)
    printf("\n%ld\t%ld",e[i],d[i]);
    encrypt();
    decrypt();
    getch();
}

int prime(long int pr)
{
    int i;
    j=sqrt(pr);
    for(i=2;i<=j;i++)
    {
        if(pr%i==0)
        return 0;
    }
    return 1;
}

void ce()

```

```

{
int k;
k=0;
for(i=2;i<t;i++)
{
    if(t%i==0)
        continue;
    flag=prime(i);
    if(flag==1&&i!=p&&i!=q)
    {
        e[k]=i;
        flag=cd(e[k]);
        if(flag>0)
        {
            d[k]=flag;
            k++;
        }
        if(k==99)
            break;
    }
}
}
long int cd(long int x)
{
    long int k=1;
    while(1)
    {
        k=k+t;
        if(k%x==0)
            return(k/x);
    }
}
void encrypt()
{
    long int pt,ct,key=e[0],k,len;
    i=0;
    len=strlen(msg);
    while(i!=len)
    {
        pt=m[i];
        pt=pt-96;
        k=1;
        for(j=0;j<key;j++)
        {
            k=k*pt;
            k=k%n;
        }
        temp[i]=k;
        ct=k+96;
        en[i]=ct;
        i++;
    }
    en[i]=-1;
}

```

```

printf("\nThe encrypted message is:\n");
for(i=0;en[i]!=-1;i++)
printf("%c",en[i]);
}
void decrypt()
{
long int pt,ct,key=d[0],k;
i=0;
while(en[i]!=-1)
{
    ct=temp[i];
    k=1;
    for(j=0;j<key;j++)
    {
        k=k*ct;
        k=k%n;
    }
    pt=k+96;
    m[i]=pt;
    i++;
}
m[i]=-1;
printf("\nThe decrypted message is:\n");
for(i=0;m[i]!=-1;i++)
printf("%c",m[i]);
}

```

-?Divine- - October 18, 2013, 1:20 pm

Lookin' good Keeper. My eyes hurt from this.

powerbk2 - October 18, 2013, 1:47 pm

Damn, I just looked through it, and my brain hurts.
Never the less, nice job on making this yo.

Downloading... - October 18, 2013, 1:54 pm

Great job Keeper, as always...

Although this looks a little abstract for me :D .

ida - October 18, 2013, 2:29 pm

Great tutorial, Keeper.

I have added it to the group thread.

Th3PonyWizard - October 18, 2013, 4:04 pm

Wow. How much time does it usually take to write something THIS long?

I've read out everything twice and I'm still having troubles understanding. Low-level is something I'll get a grasp on... Some day. c:

barfoo - October 18, 2013, 4:17 pm

Again, a thread like this, nevertheless how well it may be written, shouldn't belong in the Website & Forum Hacking section. This forum is expected to have high level web-based discussions.

-?Divine- - October 18, 2013, 4:21 pm

(Oct 18, 2013 04:17 PM)barfoo Wrote: [\[CITE\]](#) Again, a thread like this, nevertheless how well it may be written, shouldn't belong in the Website & Forum Hacking section. This forum is expected to have high level web-based discussions.

Regardless of the number of skids roaming around, WFH has some of the finest hackers in it. In Cryptography section this thread would've died in under a minute.

I post low level stuff here. We don't have a section for low level, so it's natural for me to pick. Half of the threads in the first page shouldn't be in WFH for many reasons, so if I was to pick between unrelated but quality content and retarded skiddness, you can guess which one I'd pick.

barfoo - October 18, 2013, 4:40 pm

(Oct 18, 2013 04:21 PM)-?Divine- Wrote: [\[CITE\]](#) Regardless of the number of skids roaming around, WFH has some of the finest hackers in it. In Cryptography section this thread would've died in under a minute.

I post low level stuff here. We don't have a section for low level, so it's natural for me to pick. Half of the threads in the first page shouldn't be in WFH for many reasons, so if I was to pick between unrelated but quality content and retarded skiddness, you can guess which one I'd pick.

Alrighty, at least you cared enough to give a decent reply. c:

Keeper - October 19, 2013, 3:18 am

(Oct 18, 2013 04:17 PM)barfoo Wrote: [\[CITE\]](#) This forum is expected to have high level web-based discussions.

It's been expected to have a lot of things but it never came round fulfilling its goals.

Xen0n- - October 19, 2013, 4:29 am

Brilliant tutorial, looking forward to more. ?

kalandershihabrocker - *October 19, 2013, 5:40 am*

Keeper,

Kindly tell me your qualifications please.

Are you a Ph.D in computers?

0.0.0.1 - October 19, 2013, 6:11 am

Nice tutorial Keeper .

Thanks for sharing

Keeper - *October 19, 2013, 6:33 am*

(Oct 19, 2013 05:40 AM)kalandershihabrocker Wrote: [\[CITE\]](#) Kindly tell me your qualifications please.

Secondary education.

kalandershihabrocker - *October 19, 2013, 7:22 am*

Not sure if it's a bug or a part of your secondary education. ?

Dyme???????????? - October 19, 2013, 7:11 pm

Loling @ whoever 1 starred this thread. Like literally, how pathetic are you.

Great thread Keeper - I don't know how you do it.

Im gone brah - October 19, 2013, 7:24 pm

Always good tutorials from you keepah.

Jay? - October 19, 2013, 7:28 pm

Hey man great tutorial! You can see quite a gap from what you and others in your group are posting then most people on the boards.

seg fault - October 20, 2013, 11:38 pm

(Oct 18, 2013 04:21 PM)-?Divine- Wrote: [\[CITE\]](#) Regardless of the number of skids roaming around, WFH has some of the finest hackers in it. In Cryptography section this thread would've died in under a minute.

I post low level stuff here. We don't have a section for low level, so it's natural for me to pick. Half of the threads in the first page shouldn't be in WFH for many reasons, so if I was to pick between unrelated but quality content and retarded skiddness, you can guess which one I'd pick. some of the finest hacker i hope u don't think ur on the list lol

Zixem - October 21, 2013, 8:24 am

Great thread as always, keep 'em coming.

Keeper - October 26, 2013, 12:32 pm

Second page,???????? buried already.

Viadecchi - October 27, 2013, 3:47 am

Please can you post some good book on this theme? I am also interested in this field.

Thanks in advance

Keeper - October 27, 2013, 3:53 am

(Oct 27, 2013 03:47 AM)Ventura123 Wrote: [\[CITE\]](#) Please can you post some good book on this theme? I am also interested in this field.

Thanks in advance

Yes. [This one](#) is the best but it's quite advanced and I hardly perceive anything out of it.

samrokzz - October 27, 2013, 6:49 am

Awesome tutorial as always , Thanks for posting .

Numinous - October 27, 2013, 6:52 am

I just tried this similar concept in MATLAB itself but its looking great here also.

Thanks for sharing here.

Nalum - October 27, 2013, 9:26 am

Great Read! Although i do not have the brainpower to understand and/or use this i really appreciate you take the time to write just an in depth tutorial and explanation.

Hats off to you my friend, this is highly motivating!

Cheers

Keeper - October 27, 2013, 9:45 am

(Oct 27, 2013 06:52 AM)Wordpress Security :) Wrote: [\[CITE\]](#) I just tried this similar concept in MATLAB itself but its looking great here also.

Thanks for sharing here.

Matlab or Mathematica. They both offer extensive tools for steganography. This [right here](#) is something I've ran into before. Though it's quite different since here we have no clue of the eLSBs initial values, the guy there declares his own variable to hold the message within the image.

Asp1r3-At0m - *October 27, 2013, 10:08 am*

My brain exploded, a lot of very nice and well explained information, I will bookmark this thread to read with more time this, It is awesome!

Great contribution man, If I had leet or uber I would give you all the rep I can :(

Keeper - *October 28, 2013, 11:46 am*

I intend to write a lot more about cryptovirology and kleptography since I barely mentioned them in this tutorial. As soon as I finish reading the book, I'll write another one.

Im gone brah - *November 1, 2013, 5:44 am*

Really well written tutorial.

Keeper mind sending me a pm?

Goodies - *January 16, 2014, 3:45 pm*

Bumping this awesome tut. I'm sure you spent a lot of time on it.

I'm fairly familiar with the basics of LSB cryptography, but this is an awesome thread and I appreciate the time you spent :P

Zapu - *April 14, 2014, 4:03 pm*

This looks like an interesting field to pursue, although my math isn't too good..which I feel like it'd be a hindrance.

Darck Loard - *April 16, 2014, 12:03 am*

thanks bro good topic?????

Amat3rasu - *March 5, 2015, 3:06 pm*

As always great tutorial, thanks a lot