Lolcipher Submission

technion@lolware.net

https://lolware.net (it uses https so it's secure)

github.com/technion/lolcipher

Introduction

This document lays out a submission to the Snake Oil Competition (http://snakeoil.cr.yp.to/). The design of this competition was found to be highly appropriate for the lolcipher algorithm, which has been in production in closed, air gapped systems for several years. As there has not been one case of compromise, the lolcipher is certified as unbreakable prior to its submission to this competition.

Lolcipher is on the Internet, and therefore demonstrated to be in the public domain.

Complexity Requirement

One of the issues identified with the AES cipher, and presumably similar block ciphers, is what I will refer to as the "in out complexity". A demonstration of this issue can be seen in the NIST published AES test vectors.

Key	2b7e151628aed2a6abf7158809cf4f3c
Block #1 Plaintext	6bc1bee22e409f96e93d7e117393172a
Input Block	6bc1bee22e409f96e93d7e117393172a
Output Block	3ad77bb40d7a3660a89ecaf32466ef97

An powerful application was created to perform a mathematical function against these vectors. It may be seen below.

```
[technion@goobyplz lolcipher]$ more lengths.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
        printf("Length of input is %lu\n",
            strlen("6bc1bee22e409f96e93d7e117393172a"));
    printf("Length of output is %lu\n",
            strlen("3ad77bb40d7a3660a89ecaf32466ef97"));

    exit(0);
}
```

```
[technion@goobyplz lolcipher]$ gcc -Wall -O9 -fomit-frame-pointer -fstack-protector lengths.c -o lengths
[technion@goobyplz lolcipher]$ ./lengths
Length of input is 32
Length of output is 32
```

For each byte of input, it is computed that the processor has a "complexity" of eight bits to process. This is the same for the output. This test application therefore demonstrates that data being fed into the AES algorithm has exactly the same mathematical complexity as the data going out. It must surely therefore be questioned whether this data is accurately being encrypted.

Whether this specific test vector demonstrates a NIST introduced backdoor not present in the original specification has not been examined.

Related Key Attack

It is my supposition that a related key attack exists against AES. Consider the following input, again, using a NIST test vector:

Input Block ae2d8a571e03ac9c9eb76fac45af8e51

As a demonstration of this attack, the following key has been created:

Key 2b7e151628aed2a6abf7158809cf4f3c

When the given input block is encrypted using the given key, the below output is seen:

Output Block f5d3d58503b9699de785895a96fdbaaf

Subsequently, the below key, which can clearly be seen to be related to the earlier key, is chosen:

Key 2b7e151628aed2a6abf7158809cf4f3c

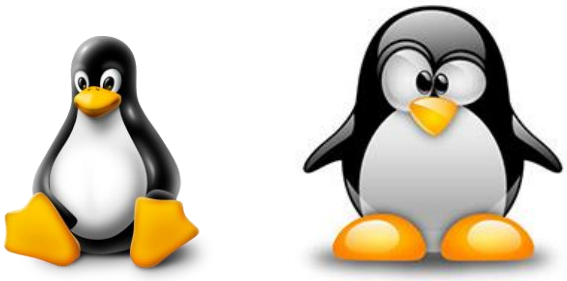
When using this key, which is related to the first, the below output is seen:

Output Block f5d3d58503b9699de785895a96fdbaaf

As has been demonstrated, related keys can be configured to produce identical output. This is a serious weakness in AES.

The penguin test

A common test of any encryption system demonstrates a picture of a penguin. The common picture can be seen below on the left, with the encrypted form on the right.



It has widely been posited that, due to the irreversible damage done to the penguin, he encryption should be considered insecure. I consider it a false assumption that this damage is irreversible and cite several sources which I will not link to which demonstrate:

The penguins weight gain may be addressed via an appearance on television's "The biggest loser"

Those eyes can be fixed with glasses. Maybe.

Lolcipher addresses those issues with a well formed structure detailed below. The key length in lolcipher is e^78 bit. Trust me it is.

The "blocksplit" function sits at its core. It takes two x eight bit inputs. A C implementation defines it well:

```
void blocksplit(int in, int inb)
{
    unsigned int a,b,c, d, e, f;
    printf("Input was %d %d\n", in, inb);
    a = rand()&0xF;
    b = in/a;
    c = in - (a*b);
    printf("Block is %d, %d, %d\n", a, b, c);


    d = rand()&0xF;
    e = inb/d;
    f = inb - (d*e);
    printf("Block is %d, %d, %d\n", d, e, f);

    int outa, outb, outc;
    outa = a<<4 | b;
    outb = c<<4 | d;
    outc = e<<4 | f;
    printf("Outputs are %d, %d, %d\n", outa, outb, outc);

}
```

The secure rand() source of entropy is called for each byte, to break it into a basic mathematical structure of multiple, base and remainder.

The three bytes output are different each call.

Conversely, the blocksplit function may be combined with the blockcombine function:

```
void blockcombine(int a, int b, int c)
{
   outa = (a>>4) * (a&0xF) + (b>>4);
   outb = (b&0xF) * (c>>4) + (c&0xF);
   printf("Combined block is %u, %u\n", outa, outb);
}
```

Proof of security

A proof regarding the complexity attack may be seen here. Input to the blockcombine function is 16 bit in size. Output is 24 bit in size. This 50% increase in complexity represents a 50% increase in security of the split function.

Related key attack

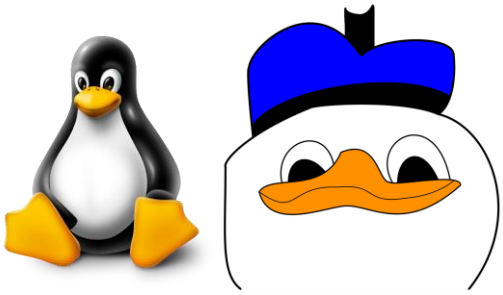
The below outputs may be seen across the same input presented multiple times:

```
[technion@goobyplz lolcipher]$ ./a.out
Input was 61 42
Outputs are 62, 17, 672
Combined block is 61, 42
[technion@goobyplz lolcipher]$
[technion@goobyplz lolcipher]$ ./a.out
Input was 61 42
Outputs are 166, 26, 66
Combined block is 61, 42
[technion@goobyplz lolcipher]$ ./a.out
Input was 61 42
Outputs are 120, 92, 54
Combined block is 61, 42
```

Although the input is the same, it can be seen that the encrypted block of three is different each call. This does not prevent the combine function from being able to resassemble as required. Therefore, even for related keys, statistical characteristics do not exist.

The penguin test

The results of the penguin test may be seen below.



It has been confirmed by an unnamed, prominent medical doctor, that the conversion from penguin to duck cannot be reversed.

It is therefore posited that, by using the lolcipher technique, all existing attacks, including related key, complexity based, and penguin based may be avoided.