CS 161 – Computer Security
Instructor: Tygar
6 October 2015

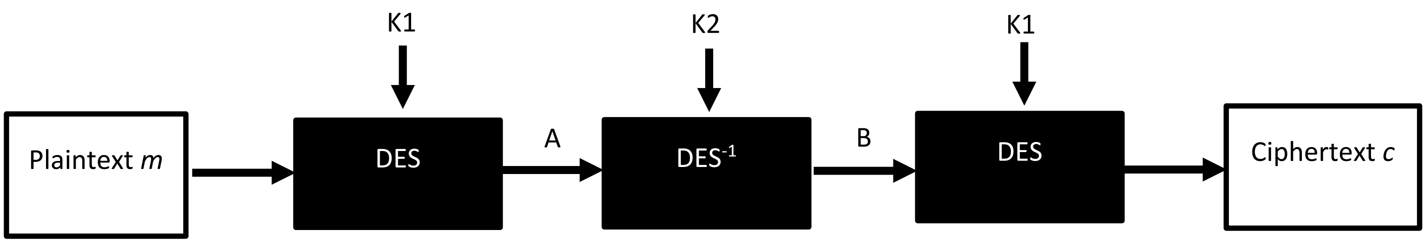
Homework 4 Answer Set

Notes

- Homework 4 is due on 13 October 2015 at 3PM.
- Please work on this homework individually – no collaboration allowed.
- Please keep your answers brief
- Submit this homework using Gradescope.

Please start the answer to each question (including subquestions) on a new page

1. We discussed how use a meet in the middle attack to break 2DES using a known-plaintext attack. Now we want to adapt it to break 3DES. We use the following diagram. Here, we are attempting to break $DES_{K1}\left(DES_{K2}^{-1}\left(DES_{K1}(m)\right)\right)$ by attempting to find $K1, K2$. Show that you can find $K1, K2$ with a ***chosen*** plaintext using attack 2^{56} chosen plaintexts and two tables of 2^{56} entries of DES inverse operations. (Hint: start by constructing a table with 2^{56} entries of all possible $K1$ and corresponding $DES_{K1}^{-1}(0)$ (that is, assume $A = 0$.) Using $A = 0$, meet in the middle.)



We compute a table of all 2^{56} key values f yielding A=0 by computing, i and $DES_i^{-1}(0)$. For each plaintext entry P_i in the table, we force a chosen plaintext attack $3DES(P_i)$ and observe the corresponding C_i. Now, we launch a man-in-the-middle attack of the sort described in lecture on the cipherpairs c = $DES_{K1}\left(DES_{K2}^{-1}(0)\right)$. Namely, we use our two tables

i	$\boldsymbol{DES_i^{-1}(0)}$
0	$DES_0^{-1}(0)$
1	$DES_1^{-1}(0)$
2	$DES_2^{-1}(0)$
…	…

j	$DES_j^{-1}(C_j)$
0	$DES_0^{-1}(C_0)$
1	$DES_1^{-1}(C_1)$
2	$DES_2^{-1}(C_2)$
…	$\dots$

When we have a match, we have a corresponding possible pair of $K1 = j$*,* $K2 = i$ *values.*

2. Suppose there is a transmission error in a block of ciphertext using CBC mode. Show that the error propagates for two blocks in decryption and then recovers.

Suppose the error is in block C_i*. Then* $(P_i = D(K, C_i)\ xor\ C_{i-1})$ *and* $(P_{i+1} = D(K, C_{i+1})\ xor\ C_i)$ *will not be properly recovered. But* $(P_{i+2} = D(K, C_{i+2})\ xor\ C_{i+1})$ *will be properly recovered*

3. Consider the following improvement to one time pad encryption, which we will call *super one time pad* encryption. As before our message and encryption key is a string of bits. But for super one time pad encryption we compute $c = m\ xor\ k\ xor\ k^R$ where k^R denotes key reversal (so, for example, $11010001^R = 10001011$). Is super one time pad encryption perfectly secure (that is, does it leak no information about the contents of the plaintext other than the length of the plaintext)?

No, it is not, because $(k\ xor\ k^R)$ *is limited in the values it can assume. The first half of k's bits determine the second half of k's bits. Suppose for example that the first bit of c is the same as the last bit of c, then the first bit of m is the same as the last bit of m. Conversely, if the first bit of c is different than the last bit of c, then the first bit of m is different than the last of bit of m.*

4. Let F be a single round of a Feistel cipher operating on 64 bit blocks, so that inputs to it are (a_L, a_R) where a_L and a_R are each 32 bits long, and F maps (a_L, a_R) to $(a_R, a_L\ xor\ f(a_R, K))$. Suppose that (a_L, a_R) and (b_L, b_R) are a pair of plaintexts such that $q = a_R\ xor\ b_R$. Consider two rounds of Feistel so that $(c_L, c_R) = F(F(a_L, a_R))$ and $(d_L, d_R) = F(F(b_L, b_R))$. Show that if $c_L = d_L$ then $c_R\ xor\ d_R = q$.

We compute $(c_L, c_R) = F\big(F(a_L, a_R)\big) = \big(a_L\ xor\ f(a_R, 0), a_R\ xor\ f(a_L\ xor\ f(a_R, 0), 1)\big)$.

Similarly $(d_L, d_R) = F\big(F(b_L, b_R)\big) = \big(b_L\ xor\ f(b_R, 0), b_R\ xor\ f(b_L\ xor\ f(b_R, 0), 1)\big)$

Since $c_L = d_L$ *we have* $(a_L\ xor\ f(a_R, 0)) = (b_L\ xor\ f(b_R, 0))$*, so*

$(d_L, d_R) = \big(a_L\ xor\ f(a_R, 0), b_R\ xor\ f(a_L\ xor\ f(a_R, 0), 1)\big)$ *so*

$c_R\ xor\ d_R = a_R\ xor\ f(a_L\ xor\ f(a_R, 0), 1)\ xor\ b_R\ xor\ f(a_L\ xor\ f(a_R, 0), 1) = a_R\ xor\ b_R = q$

5. Read Appendix A.1.1 on page 77 (page 85 in the PDF) of http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

Then go to http://cloud.sagemath.com and set up an account. Run the following as a Sage worksheet:

```
#modulus for P-256
p = 115792089210356248762697446949407573530086143415290314195533631308867097853951

#a and b values for P-256
a = -3
b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b

#order of P-256 (we do not use this)
#n = 115792089210356248762697446949407573529996955224135760342422259061068512044369

#create elliptic curve
FF = GF(p)
EC = EllipticCurve([FF(a),FF(b)])

#generator point P for P-256
Px = 0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
Py = 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5
P = EC(FF(Px),FF(Py))

#hush, hush, this is TOP SECRET
e = 123456 # eyes only!
Q = e*P
Qx = Q[0]
Qy = Q[1]
print "Qx =", hex(Integer(Qx)), "Qy =", hex(Integer(Qy))
```

a. Explain exactly what this program does. What would be the effect of publishing your point Q in the next edition of the NIST 800-90A standard? How long did Sage take to run this program?

   This program takes the government standard curve P-256 with the fixed generator "P," and calculates an "unknown" Q, for which we know $Q = eP$. If these values are used to generate random bits, then you can easily compute the "pseudo-random bits" generated by the corresponding generator. Sage runs this program in less than a second.

b. What is the output of this program?

```
Qx = 7a926a19fbdc7aa3e2e6c1476c3b8f0819e1d7cfdc2904c1adaa2ce73299e7b8
Qy = fc8929031165790a40adab6ce83e20786011473150e11a742ad46e68daaadf98
```

c. Use the same *e* and find similar output for Curve P-384 (Appendix A.1.2)

```
Qx = b89995a230041279c9cf06fa4eeaf7e95b10714dad42601038f1eaa
8e63407a99a42204d2833b80df1c95bfad53d0fab
Qy = 50ea7c117720729baba003e9c14e606e30ab3cc29f5ffd681379031
ffe464b110873ddabf8dc85037e580d3f5fde70c
```

d. Use the same *e* and find similar output for Curve P-521 (Appendix A.1.3)

```
Qx = f272381fd9b736ce6f9eb6810f98103919bafbd7b5538c3cbb785a9cc
6dd75693851415c5b132c25831aebc22a2f71684c51b15e9f468d73d690dfd
c437d997cc8
Qy = 9afecd5b35fdead12550fa9e99d1ec49c3ab79bd1a2eb7b25c81ca0de
```

*315e363a006de5db6c89421cbb8c59810f51756484583c2f758ddc15edc0be
92d8f511629*