Jeremy Barthélemy
ECE 746
Hw. 2

1. Find all irreducible polynomials:
(a) of degree 4 over *GF*(2),
From the textbook: "A polynomial f(x) over a field F is called irreducible if and only if f(x) cannot be expressed as a product of two polynomials both over F, and both of degree lower than that of f(x)."
We can determine whether the polynomials are irreducible by performing the test A(x) mod m(x) = 0? If it is equal to 0, then it is not an irreducible polynomial. If this is not the case, we can test various other irreducible polynomials m(x) of lower degree to further determine whether or not A(x) is reducible. If none of them successfully reduce A(x), we know that A(x) is an irreducible polynomial.
We can test with x, $x + 1$, $x^2 + x + 1$, $x^3 + x + 1$, $x^3 + x^2 + 1$
Below is a list of all monic polynomials of degree 4, along with their respective binary representations for simpler calculation.

Polynomial	Binary Rep.
$x^4 + x^3 + x^2 + x + 1$	11111
$x^4 + x^3 + x^2 + x$	11110
$x^4 + x^3 + x^2 + 1$	11101
$x^4 + x^3 + x^2$	11100
$x^4 + x^3 + x + 1$	11011
$x^4 + x^3 + x$	11010
$x^4 + x^3 + 1$	11001
$x^4 + x^3$	11000
$x^4 + x^2 + x + 1$	10111
$x^4 + x^2 + x$	10110
$x^4 + x^2 + 1$	10101
$x^4 + x^2$	10100
$x^4 + x + 1$	10011
$x^4 + x$	10010
$x^4 + 1$	10001
x^4	10000

Example test:
11111
<u>11000</u>
00111
<u>00110</u>
00001

Not divisible by x+1, now test with x:
11111
10000
01111
01000
00111
00100
00011
00010
00001
Again, we find that the result is not 0, thus $x^4 + x^3 + x^2 + x + 1$ is our first irreducible polynomial.

If we test $x^4 + x^3 + x^2 + x$, however, we can easily see that it is divisible by x:
11110
10000
01110
01000
00110
00100
00010
00010
00000

I found the following to be another irreducible polynomial by simple inspection: $x^4 + x + 1$
To verify:
10011
11000
01011
01100
00111
00110
00001

And if we perform the modulus operation using x:
10011
10000
00011
00010
00001

Finally, I found one more irreducible polynomial for this case: $x^4 + x^3 + 1$ (11001)
11001
10000
01001
01000

00001

And by using x+1,
11001
<u>11000</u>
00001
So, I found the following three irreducible polynomials, which were not reducible under (00010, 00011, 00111, 01011, $01101)_2$ polynomials:
$x^4 + x^3 + x^2 + x + 1$, $x^4 + x + 1$, and $x^4 + x^3 + 1$

b) of degree 2 over *GF*(3).
Below is a list of all monic polynomials of degree 2 over GF(3):

Polynomial	Ternary Rep.
$x^2 + 2x + 2$	122
$x^2 + 2x + 1$	121
$x^2 + 2x$	120
$x^2 + x + 2$	112
$x^2 + x + 1$	111
$x^2 + x$	110
$x^2 + 2$	102
$x^2 + 1$	101
x^2	100

Testing $x^2 + 2x + 2$:
122
<u>100</u>
022
<u>010</u>
012

122
<u>110</u>
012
<u>011</u>
001

We find that x2 + 2x + 2 is an irreducible polynomial.
Testing $x^2 + 2x + 1$:
121
<u>100</u>

021
010
011
010
001

121
110
011
011
000
$x^2 + 2x + 1$ is a reducible polynomial.

Testing 120:
120
100
020
010
000
Clearly, this is $x^2 + 2x$ is also a reducible polynomial.

Along with $x^2 + 2x + 2$, I found two other irreducible polynomials: $x^2 + 1$ and $x^2 + x + 2$.
Example:
112
110
002

112
100
012
010
002

2.
(a) One attack I was able to find was here: http://www.iacr.org/cryptodb/archive/2004/PKC/3421/3421.pdf, where the greatest field size to be successfully attacked was of $GF(2^{63})$, although another claims that the best is over $GF(2^{503})$! http://www.rsa.com/rsalabs/node.asp?id=2194
(b) Although some rather large attacks have been mounted successfully (the largest public one being of $GF(2^{97})$, it is generally considered secure in the range from $GF(2^{32})$ to $GF(2^{128})$, depending upon what purpose and how important the protection is. http://www.rsa.com/rsalabs/staff/bios/aoprea/publications/GF.pdf
(c) Yes, as it is becoming increasingly important to perform these operations quickly in a cryptographic perspective, there are many new and upcoming algorithms to improve upon the speed. For example, the algorithm presented here; http://www.princeton.edu/~rblee/ELE572Papers/Fall04Readings/NingYin-FiniteFieldMul.pdf,

which claims to offer a more efficient solution to software implementations of finite field multiplication.

3.
We generate the alog and log tables by taking a generator for this case to produce the values:
Taking 2 or $(10)_2$ and squaring mod P(x), we find the following values:
10 mod 11001 = 00010
10*10 mod 11001 = 00100
10*100 mod 11001 = 01000
10*1000 mod 11001 = 01001
10*1001 mod 11001 = 01011
10*1011 mod 11001 = 01111
10*1111 mod 11001 = 00111
10*111 mod 11001 = 01110
10*1110 mod 11001 = 00101
10*101 mod 11001 = 01010
10*1010 mod 11001 = 01101
10*1101 mod 11001 = 00011
10*11 mod 11001 = 00110
10*110 mod 11001 = 01100
10*1100 mod 11001 = 00001

Thus, the tables are as follows:

k	alog(k)
1	10
10	100
11	1000
100	1001
101	1011
110	1111
111	111
1000	1110
1001	101
1010	1010
1011	1101
1100	11
1101	110
1110	1100
1111	1

k	log(k)
1	1111

10	1
11	1100
100	10
101	1001
110	1101
111	111
1000	11
1001	100
1010	1010
1011	101
1100	1110
1101	1011
1110	1000
1111	110

(a) A * 7 = 10*7 = 1010 * 0111
= alog[log[1010]+log[0111] mod 10000]
= alog[1010 + 111 mod 10000]
= alog[10001 mod 10000] = alog[1] = 10 = 2
(b) 5*F = 5*15 = 0101 * 1111
= alog[log[0101] + log[1111] mod 10000] = alog[1001 + 110 mod 10000] = alog[1111] = 1
(c) $B^{-1} = 11^{-1}$ =
We calculate this first by determining the inverse of B:
$x*B^{-1}$ mod n = 1
alog[log[1011] + log[x] mod 10000] = 1 → B^{-1} = 1010, so to verify, 1010*1011 →
alog[log[1010] + log[1011] mod 10000] → alog[1010 + 101 mod 10000] = 1
(d) $C * D^{-1} = 12 * 13^{-1}$
alog[log[1101] + log[x] mod 10000] = 1
1011 + log[x] mod 10000 = 1111
log[x] mod 10000 = 100
x = 100, so → D^{-1} = 100 → alog[log[1100] + log[100] mod 10000] →
alog[1110 + 10 mod 10000] = 10

4.
(a) de Fermat Little Algorithm:
To determine the inverse of A(x), we use $A^{(2^m)-2}$ mod P(x).
Thus, with m = 4, we need to calculate $A(x)^{14}$.
$A(x) = x + 1$
$A(x)^2 = x^2 + 1$
Then multiplying by A(x) again, we obtain
$A(x)^3 = x^3 + x^2 + x + 1$
$A(x)^6 = x^3 + x$, which we square to obtain
$A(x)^{12} = x^3$
Then with $A(x)^{12} * A(x)^2$, we obtain $A(x)^{14} = x^3 + x^2 + x$.

To verify that this is indeed the inverse of A(x), we test it:
1110 * 0011 = 10010
10010
<u>10011</u>
00001
Thus, the inverse is $x^3 + x^2 + x$
(b) Extended Euclidean Algorithm for Polynomials:

i	qi	ri	xi
-2	-	10011	0
-1	1110	11	1
0	11	1	1110
1	-	0	

Where we can see that 1110 = $x^3 + x^2 + x$, which is the same result as obtained using Fermat's Little Theorem.

5.
a.
```
i    qi                                  ri                              xi
-2   -                                 x^7 + x + 1                        0
-1   x^6 + x^5 + x^4 + x^3 + x^2 + x   x + 1                              1
0    -                                       x^6 + x^5 + x^4 + x^3 + x^2 + x

('Result: ', x^6 + x^5 + x^4 + x^3 + x^2 + x)
```

b.

```
  i     qi               ri                   xi
  -2    -                x^5 + 2*x + 1        0
  -1    2*x^3 + 2*x      2*x^2 + 1            1
  0     -                                     x^3 + x

  ('Result: ', x^3 + x)
```

Sage:
```
P.<x> = GF(3^5,'z')[]
def Problem5(Px, Ax):
    (A1, A2, A3) = (1, 0, Px);
    (B1, B2, B3) = (0, 1, Ax);
    print'i', '          qi                ri              xi';
    i = -2;
    print i, '     ', '-', '      ', A3, '   ', A2;
    i = i + 1;
    qi = A3.quo_rem(B3)[0];
    print i, '     ', qi, '      ', B3, '      ', B2;
    Qtemp = 0;
```

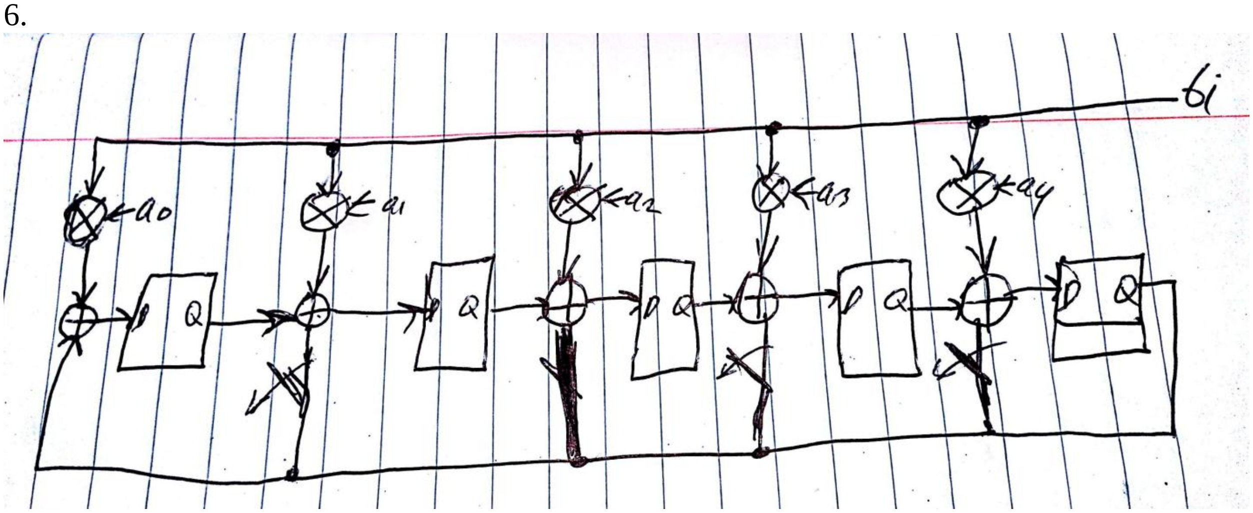
```
    while(True):
        i = i + 1;

        if(0 ==B3.degree()):
            return('Result: ', B2/B3);

        Qi = A3.quo_rem(B3)[0];

        (T1, T2, T3) = (A1 - qi*B1, A2-qi*B2, A3 - qi*B3);
        if(Qtemp == 0):
            print i, '        ','-', '            ', T2;
        if(Qtemp != 0):
            print i, '        ',Qtemp, '            ', T2;
        Qtemp = qi;
        (A1, A2, A3) = (B1, B2, B3);
        (B1, B2, B3) = (T1, T2, T3);
#Followed by the line below in a different code line to test:
Problem5(x^7+x+1, x+1)
```

6.



7.

(a) We have C(x) = A(x)B(x) mod P(x)

$c_1x + c_0 = (a_1x + a_0)(b_1x + b_0) \bmod x^2 + p_1x + p_0 =$

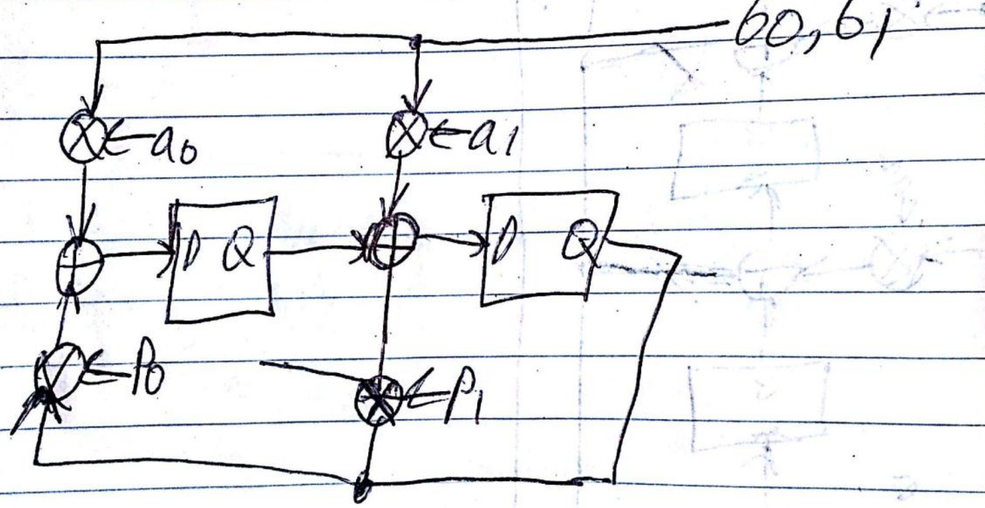
$a_1b_1x^2 + a_0b_1x + b_0a_1x + a_0b_0 \bmod x^2 + p_1x + p_0 =$

$$
\begin{array}{r}
a_1b_1 \\
x^2+p_1x+p_0 \,\big)\, a_1b_1x^2+(a_1b_0+a_0b_1)x+a_0b_0 \\
a_1b_1x^2+a_1b_1p_1x+a_1b_1p_0 \\
\hline
0+(a_1b_0+a_0b_1-a_1b_1p_1)x+a_0b_0-a_1b_1p_0
\end{array}
$$

So, $C_1 = a_1b_0 + a_0b_1 + a_1b_1p_1$

$C_0 = a_0b_0 + a_1b_1p_0$

6.



8.

We use multiplication clocked at 50MHz.

With this clock frequency, we have a $1/(50*10^6)$ = 20 ns clock period.

To perform the Diffie-Hellman Key Exchange, we assume each side simultaneously generates their public key, sends the key to each other, and then determines a shared secret key from this. In order to generate their public key using their private key in $GF(2^{593})$, this will take on average $(2*2^{296})$ clock cycles, which would require $(2*2^{296})*20$ns = $4.9*10^{77}$ seconds or 1.36 hours, which seems extremely large.