1 Problem 1

1.1 Part A

To demonstrate the square-and-multiply algorithm with $x^{937}$, let us start by breaking 937 down into binary. This gives us:

1 1 1 0 1 0 1 0 0 1

We now iterate over each bit from left to right. We initialize some value to $x^0$. For every bit, we square that value. If the bit we are on is a one, we additionally multiply that value by $x$.

An example is shown below.

<table>
<thead>
<tr>
<th>1 1 1 0 1 0 1 0 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^0$ $x^2$ $x^6$ $x^{14}$ $x^{28}$ $x^{56}$ $x^{112}$ $x^{234}$ $x^{468}$ $x^{936}$</td>
</tr>
<tr>
<td>$x^1$ $x^3$ $x^7$ $x^{15}$ $x^{29}$ $x^{58}$ $x^{116}$ $x^{234}$ $x^{468}$ $x^{937}$</td>
</tr>
</tbody>
</table>

1.2 Part B

Now let us find $2^{341} \mod 1001$. We follow the same steps as above, starting with converting 341 to binary. One we do that, we can perform the algorithm, additionally reducing each iteration by the modulus.

<table>
<thead>
<tr>
<th>1 0 1 0 1 0 1 0 1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4 16 23 529 246 456 914 561</td>
</tr>
<tr>
<td>2 32 57 912 123</td>
</tr>
</tbody>
</table>

Therefore, $2^{341} \equiv 123 \mod 1001$. 

1
2 Problem 2

2.1 Part A

To prove that encryption and decryption work with $\lambda(n)$, we can modify an existing proof in Stinson.

\[
(x^b)^a = x^{t\lambda(n)} x \mod n \\
= x^{t\frac{(p-1)(q-1)}{\gcd(p-1,q-1)}} x \mod n \\
= x^{t\frac{\phi(n)}{\gcd(p-1,q-1)}} x \mod n \\
= (x^{\phi(n)})^{t\frac{1}{\gcd(p-1,q-1)}} x \mod n \\
= 1^{t\frac{1}{\gcd(p-1,q-1)}} x \mod n \\
= x \ mod n
\]  

(1)

2.2 Part B

In the modified cryptosystem, we need to find $\lambda(n)$ given that $p = 37$ and $q = 79$.

\[
\lambda(n) = \frac{(p-1)(q-1)}{\gcd(p-1,q-1)} \\
= \frac{(36)(78)}{6} \\
= 468
\]  

(2)

Now that we have $\lambda(n)$, we can find $a$ knowing that $ab \equiv 1 \mod \lambda(n)$; $a$ is simply $b^{-1} \mod 468$.

Finding a multiplicative inverse can be done using the extended euclidian algorithm. A program was written to do exactly this. The C source code can be found appended to this report.

The program was used to find the inverse of 7 mod 468

\[
$ ./invert 7 468 \\
67$
\]

Therefore, $a = 67$.

Now we must find $a$ in the standard RSA cryptosystem. This involves solving for $a$ with $\phi(n)$ instead of $\lambda(n)$. Let us start by finding $\phi(n)$.  

\[
\phi(n) = (p - 1)(q - 1) \\
= (36)(78) \\
= 2808
\]

Using the same program, we can find \( a \).

\[
\$ \ ./invert 7 2808 \\
2407
\]

Therefore \( a = 2407 \).

3 Problem 3

A program was written that implements the RSA cryptosystem. The source code can be found appended to the end of this document.

The program consists of two parts: The first part performs exponentiation using the square-and-multiply algorithm, and the second part decodes a number into three characters from the specification given in Stinson. The first program is called \texttt{exp} and the second is called \texttt{decode}.

Both programs read from \texttt{stdin}. This allows the two programs to be piped together. A shell script is used to perform the decryption of ciphertext. The shell script is called \texttt{decrypt.sh} and can be executed as such:

\[
./decrypt.sh <a> <mod> < ciphertext
\]

where the ciphertext contains one number per line.

We can use these programs to decrypt ciphertext given that \( n = 18923 \) and \( b = 1261 \). To decrypt, we must find \( a \), which is simply \( b^{-1} \mod \phi(n) \).

Let’s start by finding \( \phi(n) \). Factoring \( n \) gives us 127 and 149. Therefore \( \phi(n) = 18648 \).

Using the inverse program mentioned earlier, we can find \( a \).

\[
./invert 1261 18648 \\
5797
\]

Using \( a \), we can find the plaintext using \texttt{decrypt.sh}

\[
./decrypt.sh 5797 18923 < ciphertext
ibecameinvolvedinaexegumentaboutmodern
paintingasubjectuponwhichiamspecta...
\]
The same procedure can be done for $n = 31313$ and $b = 4913$. We find that $\phi(n) = 30960$ and $a = 6497$

```
./decrypt.sh 6497 31313 < ciphertext2
lakewobegonismostlypoorsandysoilandevery
springtheearthheavesupanewcropof...
```
/*
* invert.c - a small program that finds the multiplicative
* inverse of a number given a modulus.
*
* author: John Murphy <jtmurphy@vt.edu>
*/

#include <stdio.h>
#include <stdlib.h>

static void
usage(FILE * stream, const char * prog)
{
    fprintf(stream, "usage: %s <number> <mod>\n", prog);
}

static void
iter(long * x, long * nx, long q)
{
    long tmp;
    tmp = *nx;
    *nx = *x - (q * *nx);
    *x = tmp;
}

static long
invert(long x, long m)
{
    long q;
    long t = 0, nt = 1;
    long r = m, nr = x;

    while (nr) {
        q = r / nr;
        iter(&t, &nt, q);
        iter(&r, &nr, q);
    }

    if (r > 1) /* not invertible */
        return 0;

    if (t < 0)
t = t + m;
return t;
}

int main(int argc, char * argv[])
{
    long x, mod, res;

    if (argc != 3) {
        usage(stderr, argv[0]);
        return 1;
    }
    x = strtol(argv[1], NULL, 10);
    mod = strtol(argv[2], NULL, 10);
    res = invert(x, mod);
    if (res == 0) {
        fprintf(stderr, "Not invertible.\n");
        return 1;
    }
    printf("%ld\n", res);
    return 0;
}
/* 
* decode.c - converts a number into three characters 
*       (as described in Stinson) 
*/

#include <stdio.h>

#define ascii(c) ((c) + 'a')

int
main(int argc, char * argv[])
{
    char msg[4];
    long in;

    scanf("%ld", &in);
    msg[2] = ascii(in % (26));
    msg[1] = ascii((in / 26) % (26));
    msg[0] = ascii((in / (26 * 26)) % (26));
    msg[3] = '\0';

    printf("%s", msg);
    return 0;
}
/* Exponentiates a number (shift and multiply) */
#include <stdio.h>
#include <stdlib.h>

/* save some trees */
typedef unsigned long ulong;

static void expmodimpl(ulong * res, ulong base, ulong pow, ulong mod)
{
    if (pow == 0)
        return;
    expmodimpl(res, base, pow >> 1, mod);
    *res = (*res * *res) % mod;
    if (pow & 0x1)
        *res = (*res * base) % mod;
}

static ulong expmod(ulong base, ulong pow, ulong mod)
{
    ulong res = 1;
    expmodimpl(&res, base, pow, mod);
    return res;
}

int main(int argc, char * argv[])
{
    ulong base, pow, mod;
    pow = strtol(argv[1], NULL, 10);
    mod = strtol(argv[2], NULL, 10);
    scanf("%lu", &base);
    printf("%ld\n", expmod(base, pow, mod));
    return 0;
}
#!/bin/sh

a=5797       # example $1
mod=18923    # example $2

while read num; do
  echo $num | ./exp $1 $2 | ./decode
done

echo      # clean prompt