**Math 319NT Fall 2024** **Name: Aby Soumare**
**Lab 8: Exploring ElGammal Encryption**
**Collaborators:**

---

**Introduction**

In this lab, we will be exploring the topic of the ElGammal crypto system. This is a public key crypto system which has its roots in Diffie-Hellman, which is a key exchange system that allows parties to securely send information to one another.

**Background**

To begin studying and exploring ElGammal, we should first explain how the system works. For this, we will be introducing to you two people who want to communicate with each other: Linda and Wanda. First, Linda needs to choose a parge prime number, $p$, and then a primitive number of that $p$, $g$.

**Definition 0.1.** *A primitive element is a generator of a finite field's multiplicative group. Every element can be written as a power of the primitive element.*

**Example 0.2.** *A primitive element of the prime number* $479$ *is* $13$.

After Linda chooses a primitive element, $g$, then she is to choose a private key $x$ such that $1 \leq x \leq p-1$. After choosing this $x$, she then computes $h = g^x$.
Linda will then send Wanda $p, g, h$ in order for Wanda to continue to decrypt the message.
Now, Wanda takes the information given by Linda and chooses her own secret $y$ such that $1leqy \leq p-1$. She will then compute $s = h^y$. After that, she will compute two numbers, $c_1, c_2$ such that $c_1 = g^y$ and $c_2 = M * s$. The letter $M$ in this case is just Wanda's message represented as an integer where $1 \leq M \leq p$. Now, after computing all of that, Wanda will send Linda $c_1$ and $c_2$.
To finish up, Linda will compute $s$ by computing $c_1^x$. Finally, she will compute $s^{-1}c_2$ to get the message $M$.

As one may observe, this is a clunky way to solve encryption problems, and we want to be able to compute these things more efficiently. In this lab, we will be tackling that problem, along with solving discrete logs using magma.

**Definition 0.3.** *The discrete log problem, describes the phenomenon where in which solving for logs,* $g^x = p$*, given a very large prime number* $p$ *and a generator* $g$ *becomes increasingly difficult the larger* $p$ *becomes.*

**Example 0.4.** *Discrete log computation:* $2^2 \equiv 4 \bmod 13$.

See how we have the generator $2$, the prime $13$, and the exponent $2$. We can continue to compute with all the elements in $\mathbb{Z}/13$ with the generator $2$ very easily. You can imagine how much harder it would be to do this same thing for a much larger prime.

**Analysis**

In this section, we will be formulating code in Magma to encrypt and decrypt a message for us. Afterward, we will be going into detail about the discrete log problem, and then finally going into an example of sorts. First off, we will jump into the coding section of this lab.
A template outlining the ingredients needed to solve this problem were given to us by Professor Day, and here we will be showcasing the code and describing each line. You can find the code sans comments in the appendix section.

**Code 0.5.** *Commented encryption code*

```
function ElGamalEc(p,g,h,M)
    R<x>:= PolynomialRing(FiniteField(p));
    y:= Random(1,p-1); %the function Random(x,y) chooses a number n
                       %such that x \leq n \leq y.
    s:= R!h^y; %putting h^y through our ring
    c_1:= R!g^y; %putting g^y through our ring
    c_2:= R!M*s; %putting M*s through our ring
return c_1,c_2; %returning the cypher text
end function;
```

Now we are to make a function to decrypt a message given the cypher text of the first function. We again used a template of code given by Professor Day. Here is the completed and explained code:

**Code 0.6.** *Commented decryption code*

```
function ElGamalDc(p,g,h,c1,c2)
    R<x>:= PolynomialRing(FiniteField(p));
    y:= Random(1,p-1); %the function Random(x,y) chooses a number n
                       %such that x \leq n \leq y.
    s:= R!c1^y; %putting c_1^x through our ring
    sinv:= InverseMod(s,p); %computing the inverse of s with the
                       %InverseMod(x,m) fuction in magma
    M:= R!sinv*c2; %putting sin*c_2 through our ring to get the message
return M; %returning our message
end function;
```

After this, we are to check the code works by using it in an example.

**Exercise 0.1.** *Use Code* $0.5$ *and Code* $0.6$ *to encrypt and decrypt a message of your choice to verify the functions work. This is what is going into ElGamalEc():*

1. $p = 101$

2. $g = 2$
3. $h = 64$
4. $M = 2$.

*This returned the two numbers* $20, 74$. *Now, we will use this information for ElGamalDc():*

1. $p = 101$
2. $g = 2$
3. $h = 64$
4. $x = 5$
5. $c_1 = 20$
6. $c_2 = 74$

*Which then returns* $2$.

Now we are to start studying the topic of the discrete log problem. In order to do this, we are again utilizing Magma to do the computations for us.

**Exercise 0.2.** *Use magma to solve* $2^x \equiv 99$ *in* $F_101$.

For this, we used the code $K := FiniteField(101)$ and the built-in function $Log(b, x)$ to find $k$ such that $b^k = x$.

**Code 0.7.**
```
> K:=FiniteField(101);
> b:= K!2;
> x:= K!99;
> k:= K!Log(b,x);
> print k;
51
```

From this, we now know that $2^51 = 99$ in the modulo space of $101$. Now we will try an example with different numbers:

**Code 0.8.**
```
> K:=FiniteField(7877);
> b:= K!2;
> x:= K!555;
> k:= K!Log(b,x);
> print k;
1716
```

The function $PrimitiveRoot(x)$ was used to calculate what $b$ should be used to ensure this process works. Try your own example using the same process and different numbers.

We want to analyze ElGamal to understand how it truly works. In order to do that, we will start with $M$ and how encrypting it and decrypting $c_1, c_2$ really does give us $M$.

**Exercise 0.3.** *Verify that ElGamal is correct:*
*The process of encrypting $M$ was outlined above, so refer to that outline when reading this section.*
*First, when we are encrypting $M$, we are multiplying it by $s = h^y = g^{xy}$, and then we are sending $g^y$ and $Mg^{xy}$ to someone else to decrypt it.*
*To decrypt it, we are first computing $c_1^x = g^{yx} = s$, and then computing $s^{-1}c_2 = s^{-1}Mg^{xy}$.*
*As one may see, all of these steps are the reverse of each other. We know that $s^{-1}$ will cancel with $g^{xy}$, so we are left with $M$ after simplifying our last step.*

Now, since we know why this encryption and decryption works, we have a question that arises:

**Exercise 0.4.** *What do you think the obvious attack on ElGamal would be? Why do you think ElGamal is resistant to that attack?*

The answer to this is simple, if we were to choose too small of a prime, then it would be very easy for someone to intercept the public information and decode the message. If we were to simply choose extremely large primes, say over $100$ digits, then it would be much harder for an unwanted party to decode the message.

The final thing we will be studying is this last question posed by Professor Day:

**Exercise 0.5.** *Consider an ElGamal cipher with the public keys*

$$p = 390914864699,$$
$$g = 2,$$
$$h = 105064723877.$$

*Suppose that you intercept the encoded message $(3169561325, 106700605787)$. This question has two parts:*

1. *What is the original message $M$?*
2. *What insight does this give you into the size of the numbers you might want to work on with here?*

In order to answer this question, we are given $p, g, h, c_1, c_2$ to work with. The only unknowns we have in this question are $x, y, M$, which is perfectly fine. All we have to do to decode is:

1. Compute $s$ by computing $c_1^x$,
2. Compute $x$ where $h = g^x$

3. Compute $M$ by computing $s^{-1}c_2$,

Since we essentially have all of these elements, other than $x$ before we compute our log, then finding our original $M$ will not be a challenging problem. As we have done before, we want to compute $105064723877 = 2^x$ for $x$, and to do that we can use the $Log(a, b)$ function in magma.

**Code 0.9.**
```
> K:=FiniteField(390914864699);
> b:=K!2;
> x:=K!105064723877;
> k:=K!Log(b,x);
> print k;
```

From doing this, we now know our $x = 284182014506$. Now we can compute $c_1^x = s = 386109946154$, and then we can compute our $M = s^{-1}c_2 = 47068479877$.
Given how easy it was for us to calculate the $M$ given this information, we can come to the conclusion that we should be choosing incredibly large primes and incredibly large $g$ in order for the calculations to become harder for our computers to compute.

**Future Directions**

Given all the information learned in this lab, some questions that come up are as such:

1. What are some more practical uses of this encryption methods?
2. Are there other methods that allow us to send word messages instead of just number messages?
3. What is the modern use of this encryption method? Are its setbacks as widely known as they should be?

The answers to this question I do not yet know, so beginning to answer them is beyond the scope of our learning for this lab. However, we have studied the topic of ElGamal encryption to a very fluent degree, nonetheless.

## Appendix

**Code 0.10.** *Uncommented code for encryption*

```
function ElGamalEc(p,g,h,M)
    R<x>:= PolynomialRing(FiniteField(p));
    y:= Random(1,p-1);
    s:= R!h^y;
    c_1:= R!g^y;
    c_2:= R!M*s;
return c_1,c_2;
end function;
```

**Code 0.11.** *Uncommented code for decryption*

```
function ElGamalDc(p,g,h,c1,c2)
    R<x>:= PolynomialRing(FiniteField(p));
    y:= Random(1,p-1);
    s:= R!c1^y;
    sinv:= InverseMod(s,p);
    M:= R!sinv*c2;
return M;
end function;
```