# Cryptography and Elementary Number Theory: A Brief Continuation From Math 592 - Cryptography Eastern Michigan University

David Moll
dmoll@emich.edu

April 21, 2017

# 1 Introduction

This is continuation of a set of notes for Math 592 - Cryptography taken at Eastern Michigan University in the Winter Semester of 2017 with Dr. J. Ramanathan. These pages round out a discussion of elementary number theory as it relates to cryptography and finish with a discussion of several cryptographic schemes, along with the number theoretic arguments for why the schemes work and what possible weaknesses they may have.

# 2 Primitive Roots in $\mathbb{Z}_n$

We are going to use our knowledge of units in $\mathbb{Z}_n$ and the Euler totient function to describe and derive some crucial facts about certain types of units, based on what we call the **Order** of an element in $\mathbb{Z}_n$.

## 2.1 Order of an element in $\mathbb{Z}_n$

**definition 1.** Let $n > 1$ be a given integer, and $a \in \mathbb{Z}_n$. The **order** of $a \mod n$ is:

$$\min\{k : k \geq 1 \text{ and } a^k \equiv 1 \mod n\}$$

Order is a property of the congruence class $a \mod n$. If this set is empty, we say that the order of a is $\infty$ or undefined. We generally write the order this way:

$$\text{ord}(a) = k$$

**Theorem 2.** *The order of $a \in \mathbb{Z}_n$ is finite precisely when* $\gcd(a, n) = 1$

Note that because this theorem is phrased in the manner of *precisely when*, we know that it is an "if an only if" scenario. So first we prove the "if" portion of the statement.

*Proof.* If the order of $a$ is finite $\mod n$, there is an integer $k \geq 1$ such that:

$$a^k \equiv 1 \mod n$$

Therefore, there exists some $x$ such that:

$$\begin{aligned} a^k - 1 &= x \cdot n (\text{By the definition of modulus}) \\ a^k - x \cdot n &= 1 \\ a^{k-1} \cdot a + (-x) \cdot n &= 1 (\text{This is a linear combination of } a,n) \\ \therefore \gcd(a, n) &= 1 (\text{By Bezout's Theorem}) \end{aligned}$$

We've shown $\Rightarrow$, now we'll show $\Leftarrow$:
Now, suppose $\gcd(a, n) = 1$. This implies that $a$ and $n$ do not share any common prime divisors.
$\therefore a^k$ and $n$ share no common prime divisors. i.e. $\gcd(a^k, n) = 1$. Since we have our $\mathbb{Z}_n$ glasses on, we know that the possible values for $a^k$ are:

$$a \mod n, a^2 mod n, a^3 mod n, \ldots, a^{\phi(n)} \mod n$$

Because there are only $\phi(n)$ possibilities for the members of this list (since by Euler's Theorem, $a^{\phi(n)} \equiv 1 \mod n$).
$\therefore$ There are positive integers $i < j$ :

$$\begin{aligned} a^i \mod n &\equiv a^j \mod n \\ a^i &\equiv a^j \mod n \\ a^i &\equiv a^i \cdot a^{j-i} \mod n \qquad (\text{Factor } a^i \text{ out from } a^j) \\ 1 &\equiv a^{j-i} \mod n \end{aligned}$$

$\therefore$ The order of $a$ must be finite (and in particular, $\text{ord}(a)$ must be $\leq j - i$) $\square$

**Theorem 3.** *Let $n > 1$ and $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$. Then the order of $a \mod n$ must divide $\phi(n)$. This is,*

$$\text{ord}(a) | \phi(n)$$

*Proof.* We will do a short proof by contradiction.
First, suppose that $\text{ord}(a) \not| \phi(n)$ Then:

$$\phi(n) = a \cdot \text{ord}(a) + r \text{ where } 0 < r < \text{ord}(a) \quad \text{(By the division algorithm)} \quad (1)$$

$$a^{\phi(n)} \equiv 1 \mod n \quad \text{(Euler's Theorem)} \quad (2)$$

$$a^{a \cdot \text{ord}(a) + r} \equiv 1 \mod n \quad \text{(Substitute value from line 1 in to line 2)} \quad (3)$$

$$a^{a \cdot \text{ord}(a)} \cdot a^r \equiv 1 \mod n \quad \text{(Simplifying)} \quad (4)$$

$$(a^{\text{ord}(a)})^a \cdot a^r \equiv 1 \mod n \quad \text{(Rearranging exponents)} \quad (5)$$

$$a^r \equiv 1 \mod n \quad (a^{\text{ord}(a)} \equiv 1 \mod n \text{ by the definition of order}) \quad (6)$$

(7)

And we can see line 6 is a contradiction, because $r < \text{ord}(a)$, and $\text{ord}(a)$ is the minimal integer that produces 1 when raising $a$ to that power. $\therefore \text{ord}(a) | \phi(n)$. □

**Example 4.** What is the order of $5$ in $\mathbb{Z}_{11}$?

$$5^2 = 25 \equiv 3 \mod 11 \quad (1)$$

$$5^3 = 3 \cdot 5 \equiv 4 \mod 11 \quad (2)$$

$$5^4 = 4 \cdot 5 \equiv 9 \mod 11 \quad (3)$$

$$5^5 = 9 \cdot 5 \equiv 1 \mod 11 \quad (4)$$

So we see that $\text{ord}(5) = 5$ in $\mathbb{Z}_{11}$.

## 2.2 Primitive roots

**definition 5.** A unit $x \mod n$ is a primitive root in $\mathbb{Z}_n$ if $\text{ord}_n(x) = \phi(n)$.

Before we really dive into primitive roots, let's look at some examples.

**Example 6.** For $\mathbb{Z}_6$, the set of units is $\mathbb{Z}_6^* = \{\underline{1}, \underline{5}\}$, and $\text{ord}_6(5) = 2$, because $5^2 = 25 \equiv 1 \mod 6$. We can compute $\phi(6)$ as follows:

$$\phi(6) = \phi(2) \cdot \phi(3) = 1 \cdot 2 = 2$$

$\therefore 5$ is a primitive root in $\mathbb{Z}_6$.

**Example 7.** For $\mathbb{Z}_8$, the set of units is $\mathbb{Z}_8^* = \{\underline{1}, \underline{3}, \underline{5}, \underline{7}\}$
We can see that for all $x \in \mathbb{Z}_8^*$, $x^2 \equiv 1 \mod 8$. We can compute $\phi(8)$ as follows:

$$\phi(8) = 2^3 - 2^2 = 4$$

$\therefore$ There are no primitive roots in $\mathbb{Z}_8$ Since $4 \neq 2$.

## 2.3 The Primitive Root Theorem

**Theorem 8.** *There always exists a primitive root in $\mathbb{Z}_p^*$, where $p$ is a prime.*[1]

This is a rather heavy idea, so we can't jump into it. We need some supporting facts first, that may not immediately appear to be related but will eventually come together to prove this theorem.

**Lemma 9.** *Suppose $d|n$. Then the number of solutions to:*

$$\begin{aligned} &\star \gcd(x, n) = d \\ &\textit{for } 0 \leq x < n \\ &\textit{is } \phi(\frac{n}{d}) \end{aligned}$$

*Proof.* Suppose $x$ is a solution to $\star$. Then $d|x$ and $d|n$. So we see that:

$$\begin{aligned} \gcd(x, n) &= \gcd(d \cdot \frac{x}{d}, d \cdot \frac{n}{d}) \\ \gcd(x, n) &= d \cdot \gcd(\frac{x}{d}, \frac{n}{d}) \\ \therefore \gcd(\frac{x}{d}, \frac{n}{d}) &= 1 \text{ and } 0 \leq \frac{x}{d} < \frac{n}{d} \end{aligned}$$

Note that this works because we have defined $\gcd(x, n) = d$, so we are dividing both sides by $d$ in the last step. We have actually just constructed a function taking us from $x \mapsto \frac{x}{d}$. We want to show that this function is a bijection.

Next, let $a$ have the property that:

$$0 \leq a < \frac{n}{d} \text{ and } \gcd(a, \frac{n}{d}) = 1$$

Then we can write:

$$\begin{aligned} d =& d \cdot 1 \\ d =& d \cdot \gcd(a, \frac{n}{d}) && \text{(From how we defined a.)} \\ d =& \gcd(a \cdot d, n) && \text{(From our gcd identities)} \end{aligned}$$

Note also that $0 \leq a \cdot d < n$, and $\therefore a \cdot d$ is a solution to $\star$.

$\therefore$ the solution set to $\star$ is bijective to the set $\{a : 0 \leq a < \frac{n}{d} \text{ and } \gcd(a, \frac{n}{d} = 1\}$. We can see that there are $\phi(\frac{n}{d})$ elements in this set. □

**Lemma 10.** *For any $n \geq 1$, $n = \sum_{d|n} \phi(d)$*

This is saying that if we sum up the Euler totient values of every $d$ that divides $n$, the resulting sum is simply $n$. Let's try and prove this.

*Proof.* Partition the list of numbers:

$$0, 1, 2, \ldots, n-1$$

according to their $\gcd$ with $n$.

So if $d|n$ we write $\mathbf{X}_d = \{x : 0 \leq x < n \ \& \ \gcd(x, n) = d\}$ And each integer in the list $0, 1, 2, \ldots, n-1$ belongs to exactly one of the sets $\mathbf{X}_d$, that specific set where $d|n$.

$\therefore n = \sum_{d|n} \#(\mathbf{X}_d) = \sum_{d|n} \phi(\frac{n}{d}) = \sum_{d|n} \phi(d)$ □

This feels a little obtuse, so let's do a quick example to illustrate exactly what's happening on that last line.

[1] This is a special case of a result from group theory, but since we're dealing with this in a cryptography course we won't talk about it because we only ever say "group theory" in whispers.

**Example 11.**

| d | List of $x$ with $\gcd(x,15)=d$ | Sum |
|---|---|---|
| 1 | $1,2,4,7,8,11,13,14$ | $\phi(\frac{15}{1})=8$ |
| 3 | 3.6.9.12 | $\phi(\frac{15}{3})=5$ |
| 5 | $5,10$ | $\phi(\frac{15}{5})=3$ |
| 15 | $0$ | $\phi(\frac{15}{15})=1$ |

And notice that if we sum the last column, we get $8+5+3+1=15$, which is what the Lemma states. One of the key items here is that if we sweep through $\phi(\frac{n}{d})$ with all possible values of $d$ that divide $n$, the resulting set is simple that of all the values of $d$ that divide $n$, which is why we can say that $\sum_{d|n}\phi(\frac{n}{d})=\sum_{d|n}\phi(d)$. We are essentially changing the variable of the Euler-$\phi$ function by swapping out the variable with its complementary divisor.

*Remark* 12. Let $p$ be a prime. Then:

$$\mathbb{Z}_p^*=\{\underline{1},\underline{2},\ldots,\underline{p-1}\}$$

Every non-zero element in $\mathbb{Z}_p$ has an inverse (i.e. is a unit). This makes $\mathbb{Z}_p$ a (field). In particular, a polynomial with coefficients in $\mathbb{Z}_p$ of degree $n$ can't have more than $n$ roots.

Before we jump into proving the Primitive Root Theorem, let's recap quickly the facts we have gathered:

1. Each element of $\mathbb{Z}_p$ has an order which is a divisor of $\phi(p)=p-1$.
2. For each $d|p-1$, we have partitioned the $d$'s into sets $\mathbf{X}_d=\{x\in\mathbb{Z}_p:\text{ord}(x)=d\}$
3. Each member of $\mathbb{Z}_p$ is in precisely one of the $\mathbf{X}_d$
4. $\sum_{d|p-1}\#(\mathbf{X}_d)=p-1$ (this is a restatement of Lemma 10 with n = p-1).

We have one final Lemma to prove before we move on to the Primitive Root Theorem.

**Lemma 13.** *Let $n$ be a positive integer, and let $g\in\mathbb{Z}_n^*$ and let $m=\text{ord}(g)$.*
*Then for any $a>0$, the order of $g^a$ is $\frac{m}{\gcd(m,a)}$*

*Proof.* Consider the list:

$$\underline{1},\underline{g},\underline{g^2},\ldots,\underline{g^{m-1}},\underline{g^m}=\underline{1},\ldots$$

And note that $g^a$ is in the middle ellipses, that is $\underline{g^2}<\underline{g^a}<\underline{g^{m-1}}$. The order of $g^a \mod n$ is the smallest positive power of $g^a$ that yields $1 \mod n$. Consider the following equation:

$$\begin{aligned}
(g^a)^k &\equiv 1 \mod n \\
g^{ak} &\equiv 1 \mod n && \text{(Rearrange exponents)}\\
&\Rightarrow m|ak && \text{(Because } g^{ml}\equiv 1 \mod m \text{ for } l\in\mathbb{Z})\\
&\Rightarrow ak=m\cdot s, && \text{( for some } s\in\mathbb{N})\\
&\Rightarrow \frac{a}{\gcd(a,m)}\cdot k=\frac{m}{\gcd(a,m)}\cdot s\\
&\Rightarrow \frac{m}{\gcd(a,m)}|k && \text{(Since } \gcd(\frac{a}{\gcd(a,m)},\frac{m}{\gcd(a,m)})=1)
\end{aligned}$$

The smallest $k$ with this property is simply $\frac{m}{\gcd(a,m)}$.
$\therefore$ the order of $g^a$ is $\frac{m}{\gcd(a,m)}$ □

We're finally ready to start proving the Primitive Root Theorem8.

*Proof.* Note that $\mathbb{Z}_p^*$ has $\phi(p) = p - 1$ elements.

Write, for each $d|p-1$,

$$\mathbf{C_d} = \{0 < k < p : \mathrm{ord}(k \mod p) = d\}$$

Now we're going to prove an intermediate Lemma, which will show that if there is at least one primitive root, then there are exactly $\phi(p)$ primitive roots:

**Lemma 14.** *For each $d|p-1$, we have either:*

$$\begin{cases} \#(\mathbf{C_d}) = 0 \\ \#(\mathbf{C_d}) = \phi(d) \end{cases}$$

*Proof.* Suppose that for the divisor $d|p-1$ we have that $\mathbf{C_d} \neq \emptyset$. Then we must show that $\#(\mathbf{C_d}) = \phi(d)$.

This means that there is some $\underline{g}$ in $\mathbf{C_d}$, and the elements $\underline{1}, \underline{g}, \underline{g^2}, \ldots, \underline{g^{d-1}}$ are all roots of the polynomial $x^d - 1$. These are all of the form $g^a$, and since $(g^a)^d = (g^d)^a \equiv 1 \mod p$, we see that $g^a$ is a root of $x^d - 1$.
So all the roots of $x^d - 1$ are in this list: $\underline{1}, \underline{g}, \underline{g^2}, \ldots, \underline{g^{d-1}}$
So every element in $\mathbb{Z}_p^*$ that has order $d$ must be in this list.

A power $\underline{g^a}$ has order $d$ precisely when:

$$\frac{d}{\gcd(a,d)} = d \text{ or } \gcd(a,d) = 1$$

And there are precisely $\phi(d)$ choices for $a$ with $0 \leq a < d$. $\therefore$ there are $\phi(d)$ elements in $\mathbf{C_d}$.

Note that $\mathbf{C_d} \cap \mathbf{C'_d} = \emptyset$ for any pair of distinct divisors of $p-1$. And every $u \in \mathbb{Z}^*$ must be in $\mathbf{C_d}$ for some $d|p-1$.

$$\therefore \sum_{d|p-1} \#(\mathbf{C_d}) = p - 1$$

On the other hand, $\#(\mathbf{C_d}) \leq \phi(d) \forall d|p-1$. But we know that $\sum_{d|p-1} \phi(d) = p - 1$

$$\therefore \#(\mathbf{C_d}) = \phi(d) \quad \forall \quad d|p-1$$

$$\therefore \#(\mathbf{C_{p-1}}) = \phi(p-1) > 0$$

□

And in proving this lemma we get the proof of the primitive root theorem. Q.E.D. □

# 3 Cryptography

## 3.1 What is Cryptography?

Cryptography is the art of writing or solving codes. This is the encoding and decoding of messages to keep information secret.[2] Cryptography has been used for thousands of years,[3] but this set of notes is rather too brief to go into a detailed history of the various encryption methods which have been used throughout the years. Instead, we will focus on a small section of ciphers which are built upon the fundamentals of number theory that have been covered in the earlier parts of these pages.

## 3.2 Why do we use Cryptography?

Cryptography is used to hide information. There are many times when we would want to transmit information in such a way that only the intended recipient is able to read the information. The existence of the cryptographic schemes described later in this section are the reason that we are able to buy goods on the Internet with our credit cards without (generally) worrying about our financial information being stolen.

## 3.3 Talking about Cryptography

Whenever someone is writing about Cryptography, the same cast of characters always appears. Alice and Bob want to communicate with each other. They want to be able to send messages to each other in such a way that no one is able to read the messages. However, there is a third character: Eve. Eve's goal is to listen in (or...eavesdrop!) on Alice and Bob's communications. Each description of a cryptographic scheme is a method for Alice and Bob to prevent Eve from listening in on their conversation.

# 4 Diffie-Hellman Key Exchange

The Diffie-Hellman Key Exchange is a cryptographic scheme that allows for Bob and Alice to agree on a shared, secret key over a public channel which they can then use to encode their communications so that no one else can read them.

## 4.1 Method

- A trusted party publishes a large prime $p$ and a unit $g \mod p$ with a very large order.
- Alice and Bob each choose random integers $a$ and $b$ modulo the order of $g$.
- Alice publishes $A = g^a \mod p$ and Bob publishes $B = g^b \mod p$.
- Alice computes $B^a \mod p$ and Bob computes $A^b \mod p$. This is the common key.

Let's walk through the computation to demonstrate that Alice and Bob both end up with the same shared secret key.

- Alice's computation: $B^a \equiv (g^b)^a \equiv g^{ab} \mod p$
- Bob's computation: $A^b \equiv (g^a)^b \equiv g^{ab} \mod p$

So we can see that Alice and Bob do in fact end up with the same number for the shared secret key.

## 4.2 Eavesdropping

Suppose Eve was listening in on the conversation. She was able to hear the large prime $p$, the unit $g$, and the numbers Alice and Bob have computed - $A$ and $B$ respectively. However, there is no easy way for Eve to compute $A^b$ or $B^a$ without knowledge of $a$ or $b$. This problem is called the Discrete Log problem.

[2] https://www.merriam-webster.com/dictionary/cryptography
[3] https://learncryptography.com/classical-encryption/caesar-cipher

### 4.3 Diffie-Hellman Example

**Example 15.**

1. A trusted party publishes a large prime 941089987 and a unit 48230119 with a large order.
2. Alice chooses a random number $a = 83729143$ and computes $g^a \mod p = 48230119^{83729143} \mod p = 9925483$
3. So $A = 9925483$
4. Bob chooses a random number $b = 702086117$ and computes $g^b \mod p = 48230119^{702086117} \mod p = 176248772$
5. So $B = 176248772$
6. Now Alice publishes A, and Bob publishes B.
7. So Alice computes $B^a \mod p$, which is $176248772^83729143 \mod p$ = 562558672
8. And Bob computes $A^b \mod p$, which is $9925483^702086117 \mod p$ = 562558672
   So Alice and Bob have the same shared secret key.

## 5 Discrete logarithm

**definition 16.** Suppose $g$ is a unit in $\mathbb{Z}_n$. The discrete log with respect to $g$ is defined as follows:

$$\mathrm{dlog}_g(x) = \begin{cases} \min\{k : g^k \equiv x \mod n\} & \text{If this set is non-empty} \\ undefined & \text{Otherwise} \end{cases}$$

Remark: If $g$ is a primitive root $\mod n$, then the domain of $\mathrm{dlog}_g$ is $\mathbb{Z}_n^*$

**Example 17.** Suppose we are in $\mathbb{Z}_6$. What is $\mathrm{dlog}_3(4)$? This is the $y$ that satisfies the equation $3^y \equiv 4 \mod 7$. For small values, we can just iterate through values of $y$ until we find the correct result. In this case, we find that $3^4 \equiv 4 \mod 7$. That is, $\mathrm{dlog}_3(4) = 4 \mod 6$.

Remark: $\mathrm{dlog}_g$ is a mapping from $\mathbb{Z}_n^* \mapsto \mathbb{Z}_{\phi(n)}$. When $g$ is a primitive root, the discrete log maps from the group of units in $\mathbb{Z}_n^*$ to $\mathbb{Z}_{\phi(n)}$.

### 5.1 Intractability

For large primes $p$ and typical primitive roots, the function $\mathrm{dlog}_g(x)$ is not computable in practical terms. What this means is that there is no known *efficient* method for finding the discrete logarithm. In computer science, it is considered an open question as to if the discrete logarithm can be computed in polynomial time on a classical computer. Unfortunately, a detailed discussion of P and NP is outside the scope of these notes.

### 5.2 Other notes

The discrete log behaves algebraically in much the same way as the natural log. Suppose we have $x \equiv g^a \mod n$ and $y \equiv g^b \mod n$ for some pair of integers $a, b$ with $0 \leq a, b < d$. Then:

$$\begin{aligned} \mathrm{dlog}_g(x) &= a \mod d \\ \mathrm{dlog}_g(y) &= b \mod d \\ x \cdot y &\equiv g^{a+b} \mod n \\ \therefore \mathrm{dlog}_g(x \cdot y) &= (a+b) \mod d \\ &= \mathrm{dlog}_g(x) + \mathrm{dlog}_g(y) \end{aligned}$$

So just like with the natural log, discrete log maps multiplication to division.

The same happens with exponentiation. If $k \geq 0$ is an integer, then:

$$\begin{aligned} x^k &\equiv (g^a)^k \mod n \\ &\equiv g^{ak} \mod n \\ \therefore \operatorname{dlog}_g(x^k) &= (k \cdot a) \mod d \\ &= k \cdot \operatorname{dlog}_g(x) \end{aligned}$$

And we see that exponentiation maps to multiplication.

# 6 ElGamal

The ElGamal cryptographic scheme is an example of what is called "public-key cryptography."

## 6.1 Objective

Alice wants to set up a system where anyone (e.g. Bob) can securely send her information over a public channel.[4]

## 6.2 Method

- A trusted agent publishes a prime $p$ and a unit $g \mod p$ with a large order.
- Alice choose an integer $a$ and publishes $A = g^a \mod p$.
- Bob wants to send $m \mod p$
- Bob randomly chooses an integer $k$ (an ephemeral key), modulo the order of $g$.
- Bob computes $A^k \mod p$.
- Bob sends the order pair $(c_1, c_2)$ where $c_1 \equiv m \cdot A^k \mod p$ and $c_2 \equiv g^k \mod p$.
- To decrypt, Alice computes $(c_2)^a \equiv (g^k)^a \equiv A^k \mod p$
- Since $(c_2)^a$ is a unit, Alice can compute $(c_2^a)^{-1}$
- $\therefore$ Alice can compute $c_1 \cdot (c_2^a)^{-1} \equiv m \cdot A^k \cdot A^{-k} \equiv m \mod p$.

## 6.3 ElGamal Example

**Example 18.**

1. A trusted party publishes the prime $p = 198769$ and a unit $g = 203$.
2. Alice chooses $a = 105231$ and computes $A = g^a \mod p$ to publish $A = 37580$
3. Bob wants to send the message $m = 72613$. So he computes $(c_1, c_2)$ where $c_1 \equiv m \cdot A^k \mod p$ and $c_2 \equiv g^k \mod p$. $c_1 = 12566986684$ and $c_2 = 188222$.
4. Alice computes the shared key $(c_2)^a = 173068$ and $(c_2^a)^{-1} = 63534$.
5. Alice then decrypts the message by computing $c_1 \cdot (c_2^a)^{-1} = 12566986684 \cdot 63534 \mod p = 72613$. And Alice has successfully decrypted the message that Bob transmitted.

[4] http://people.emich.edu/jramanath/docs/math409-592w17/cryptMeth01.pdf

# 7 RSA Cryptosystem

The RSA Cryptosystem is a public-key encryption method first published by Ronald Rivest, Adi Shamir and Len Adlemen in 1977.[5]. It relies on the idea that multiplying large numbers together is relatively easy, but factoring the product of large primes is difficult.

## 7.1 Objective

Alice wants to set up a system where anyone (*e.g.* Bob) can securely send her information over a public channel.

## 7.2 Method

- Alice chooses two secret primes, $p$ and $q$.
- She computes $N = p \cdot q$ and chooses $e$ such that $\gcd(e, (p-1)(q-1)) = 1$
- Alice then publishes $N, e$.
- Bob wants to send the message $m \mod N$ to Alice. To do this, he encrypts the message by computing $c \equiv m^e \mod N$ and sends Alice $c$, the ciphertext.
- To decode Bob's message, Alice computes $d$ such that $e \cdot d \equiv 1 \mod (p-1)(q-1)$.
- Thus, the message $m$ is recovered by computing $m = c^d \mod N$.

## 7.3 Factoring is intractable

In general, the only way for an adversary (let's call the adversary Ed this time) to find $(p-1)(q-1)$ is to factor $N$ and recover $p$ and $q$. Ed can't decrypt the ciphertext $c$ unless he can compute $d$, and he can't compute $d$ without knowing $(p-1)(q-1)$. Factoring is difficult in the same way that the Discrete Logarithm problem is difficult. There is not currently a known algorithm which can factor any integer in polynomial time. Because of this fact, RSA is secure in a vast number of cases.

## 7.4 RSA Example

**Example 19.**

1. Alice chooses $p = 373587883$ and $q = 776531401$, then calculates $N = 290102722182614083$ and $(p-1)(q-1) = 290102721032494800$.
2. Alice chooses $e = 2038074743$, which is relatively prime to $(p-1)(q-1)$.
3. Bob wants to send Alice the message $m = 85413055189023892$. So he computes $c = m^e \mod N = 222227017932548197$.
4. Alice has computed the decryption coefficient $d = 125710201374708407$, where $125710201374708407 \cdot 2038074743 \equiv 1 \mod (p-1)(q-1)$
5. Now, Alice can recover $m$ by calculating $c^d \mod N = 222227017932548197^{125710201374708407} \mod N = 85413055189023892$

Thus Alice has decrypted Bob's message successfully.

[5]https://people.csail.mit.edu/rivest/Rsapaper.pdf

### 7.5 Proof of RSA Cryptosystem

Setting
We have $p, q$ which are two primes.
Then we have $N = p \cdot q$ and there exists an $e$ such that $\gcd(e, (p-1)(q-1)) = 1$
$d$ satisfies $ed \equiv 1 \mod (p-1)(q-1)$
Let $m \mod N$ be a congruence class $\mod N$. We write $c \equiv m^e \mod N$.
And we claim that $c^d \equiv m \mod N$.
It is enough to show that

$$m^{ed} \equiv m \mod N$$

*Proof.* The proof is done by cases.
Case $m \mod N$ is a unit. In this case, $\gcd(m, N) = 1$. This is equivalent to $p \nmid m$ and $q \nmid m$.
Note also that $ed \equiv 1 \mod (p-1)(q-1)$, which means that $\exists k \in \mathbb{Z}$ such that $ed = k(p-1)(q-1) + 1$.

$$\begin{aligned}
m^{ed} &\equiv m^{k(p-1)(q-1)+1} && \text{(We're looking at } m^{ed} \text{ in } \mod p\text{)} \\
m^{ed} &\equiv m^{k(p-1)(q-1)} \cdot m \\
m^{ed} &\equiv (m^{(p-1)})^{k(q-1)} \cdot m \\
m^{ed} &\equiv (1)^{k(q-1)} \cdot m \mod p && \text{(by Fermat's Little Theorem)}
\end{aligned}$$

The same technique shows that $m^{ed} \equiv m \mod q$. Thus, $m^{ed}$ solves the system:

$$\begin{cases} x \equiv m \mod p \\ x \equiv m \mod q \end{cases}$$

But, so does $m$. $\therefore m^{ed} \equiv m \mod p \cdot q$, because $p \cdot q = N$ and the Chinese Remainder Theorem guarantees uniqueness $\mod N$.
Case $m = 0$... is trivial, because everything is 0.
Case $p|m$ and $q \nmid m$

$$\begin{aligned}
m^{ed} &\equiv 0^{ed} = 0 \mod p && \text{(because } p|m\text{)} \\
m^{ed} &\equiv m^{k(p-1)(q-1)+1} && \text{(Here we're in } \mod q\text{)} \\
m^{ed} &\equiv m^{k(p-1)(q-1)} \cdot m \\
m^{ed} &\equiv (m^{(q-1)})^{k(p-1)} \cdot m \\
m^{ed} &\equiv m \mod q && \text{(By Fermat's Little Theorem)}
\end{aligned}$$

So this shows that $m^{ed}$ satisfies the system:

$$\begin{cases} x \equiv 0 \mod p \\ x \equiv m \mod q \end{cases}$$

So $x = m$ also solves this, $\therefore m^{ed} \equiv m \mod N$.
Case $p \nmid m$ and $q|m$ proceeds the same as the above case, just with $p$ and $q$ switched.

□

So we have shown that the RSA cryptosystem makes sense using the number theory framework we have been working in.

# 8 Shank's Algorithm

Also sometimes referred to as the "big-step, little-step algorithm".

## 8.1 Method to solve discrete logs

Shank's Algorithm is a faster method for computing discrete logarithms than simply brute-forcing the result. The brute force method for computing $\mathrm{dlog}_g(x)$ is the method for finding a $y$ such that $g^y \equiv x \mod p$ where we list all of the powers of $g \mod p$ until we find the $x$ we are looking for. We could also store all the powers of $g \mod p$ in a dictionary/look-up table so that we would only have to calculate the list once, but that would still take a huge amount of time and memory. In Big-O notation, we would say that the brute-force method takes $\mathcal{O}(p)$ steps in time and $\mathcal{O}(p)$ amount of memory.

## 8.2 Algorithm description

Let $g$ be a unit modulo $n$ with order $d$. Shank's algorithm is a method to find solutions $y$ to the equation

$$g^y \equiv x \mod n$$

The complexity of this method if $\mathcal{O}(d)$. If such a $y$ is found, then $y = \mathrm{dlog}_g(x)$.

1. Set $\sigma = \lceil \sqrt{n} \rceil$. Note that every integer $y$ with $0 \leq y < n$ has a unique representation of the form $y = i\sigma + j$ (from the division algorithm). Where $i, j \in \mathbb{Z}$ and $0 \leq i, j < \sigma$.
2. To solve $g^y \equiv x \mod n$, it is enough to find $i$ and $j$ with $0 \leq i, j < \sigma$ and

   $$g^{i\sigma+j} \equiv x \mod n$$

   This is equivalent to

   $$g^j \equiv x \cdot g^{-i\sigma} \mod n$$

   And note that $g^{-i\sigma} \equiv (g^{-\sigma})^i \mod n$
3. It is sufficient to generate the two lists in $\mathbb{Z}_n$:

   $$x, xg^{-\sigma}, x(g^{-\sigma})^2, \ldots, x(g^{-\sigma})^{\sigma-1}$$

   and

   $$1, g, g^2, \ldots, g^{\sigma-1}$$

   and find a matching elements between the two sets.

This is Shank's algorithm. It requires $\mathcal{O}(\sqrt{n})$ memory. If this is implemented using dictionaries (association lists), then the execution time is also $\mathcal{O}(\sqrt{n})$.

## 8.3 Shank's Algorithm Example

**Example 20.** We want to find $\mathrm{dlog}_8(79)$ in $\mathbb{Z}_101$. That is, we want to find $y$ such that $8^y \equiv 79 \mod 101$

1. We set $y = 10q + r$
2. Then we use Bezout's Theorem to find $8^{-1} \mod 101 = 38$
3. Next, we generate our dictionary of $79 \cdot 38^{-10 \cdot q} \mod 101$ for $0 \leq q \leq 10$
4. Now we compute $q^r$, for $0 \leq q < 10$, and when $q^r$ is in our dictionary, we have found q.
5. Performing the calculations we find that $q = 8, r = 8$ and $y = 88$.
6. we verify that $8^{88} \mod 101 = 79$.

# 9 Pohlig-Hellman Theorem

## 9.1 Method for solving discrete logs

Let $n$ be a positive integer, and $g$ a unit modulo $n$ of order $d$. Consider the problem of finding, for a given $x \mod n$, and integer $y$ such that

$$G6y \equiv x \mod n$$

Note that any solution $y$ is only determined up to integral shifts by $d$. Hence only the congruence class of $y \mod d$ matters.
Pohlig-Hellman's result is a method for reducing the computation of discrete logarithm $\mod n$ to the computation of many discrete logarithms - one for each prime factor of $d$.

## 9.2 Brief notation interlude

Let's introduce some notation to help us describe the method. Let

$$d = q_1^{k_1} q_2^{k_2} \dots q_l^{k_l}$$

be the prime factorization of $d$. Also, for each $i = 1, \dots, l$ write

$$d_i = \frac{d}{q_i^{k_i}}$$

Finally, set

$$g_i \equiv g^{d_i} \mod n \quad i = 1, \dots, l$$

and

$$x_i \equiv x^{d_i} \mod n \quad i = 1, \dots, l$$

## 9.3 Method for reducing computation by prime factors of d

Pohlig and Hellman's method is a way to splice together solutions $y_i$ of the equations

$$g_i^{y_i} \equiv x_i \mod n \quad i = 1, \dots l$$

Because $g_i$ has order $q_i^{k_i}$, the only $y_i$ are determined modulo $q_i^{k_i}$.
Since the $q_i^{k_i}$ are pairwise relatively prime, the congruences

$$y \equiv y_i \mod q_i^{k_i} \quad i = 1, \dots l$$

have a simultaneous solution that is unique modulo $\prod_{i=1}^{l} q_i^{k_i}$, which is just $d$. (by the Chinese Remainder Theorem). And this $y$ solves the original discrete logarithm problem.

## 9.4 Pohlig-Hellman Example

As this is a relatively complicated algorithm, let's do an example before we dive into the proof.

**Example 21.** Let $p = 10337$, $p$ is prime and let $N = p - 1 = 10336 = 32 \cdot 17 \cdot 19$. $g = 51$ is a primitive root $\mod p$. We will use the Pohlig-Hellman algorithm to find $y = \mathrm{dlog}_g(5912)$. That is, we want to find $y$ such that

$$g^y = 5912 \mod p$$

This is equivalent to solving the series of equations:

a) $(g^{N/32})^y \equiv (5912)^{N/32} \equiv 7510 \mod p$

b) $(g^{N/17})^y \equiv (5912)^{N/17} \equiv 8449 \mod p$

c) $(g^{N/19})^y \equiv (5912)^{N/19} \equiv 1617 \mod p$

Next, we use Shank's algorithm to solve for $y$ in each of the above equations

$$\begin{aligned} \text{a)} \quad & y \equiv 9385 \mod 32 \\ \text{b)} \quad & y \equiv 10311 \mod 17 \\ \text{c)} \quad & y \equiv 9588 \mod 19 \end{aligned}$$

Now we use the Chinese Remainder Theorem to reassemble the original $y$ that we need.

$$\left.\begin{aligned} y &\equiv 9385 \mod 32 \\ y &\equiv 10311 \mod 17 \end{aligned}\right\} y \equiv 9 \mod 544$$

$$\left.\begin{aligned} y &\equiv 9 \mod 544 \\ y &\equiv 9588 \mod 19 \end{aligned}\right\} y \equiv 2729 \mod 10336$$

And we see that $51^{2729} \equiv 5912 \mod 10337$, which is what we were looking for.

## 9.5 Proof of Pohlig-Hellman

To prove Pohlig-Hellman, first we need a Lemma to assist with some of our algebra:

### 9.5.1 Lemma - Expanded Bezout's Theorem

**Lemma 22.** *Suppose $N_1, \ldots, N_l$ are positive integers whose gcd is 1. Then, there are integers $c_1, \ldots, c_l$ such that*

$$c_1 N_1 + c_2 N_2 + \ldots + c_l N_l = 1$$

*Proof.* Set $S^+ = \{\sum_{i=1}^{l} c_i N_i : c_i \in \mathbb{Z} \quad \& \quad \sum_{i=1}^{l} c_i N_i > 0\}$
Note that $S^+$ is not empty. By the least integer principle, $S^+$ contains a minimal element $m$.
We claim that $m | N_i$ for $i = 1, \ldots, l$.

Indeed, suppose (to the contrary), that $m \not| N_j$ for some particular $j$. By the division algorithm,

$$N_j = qm + r \quad 0 < r < m$$

So: $r = N_j - qm$. But $m \in S^+$ and hence $m = \sum_{i=1}^{l} a_i N_i$ for some $a_1, \ldots, a_l \in \mathbb{Z}$. This implies that

$$r = N_j - \sum_{i=1}^{l} a_i N_i$$

, and $r$ is an integer linear combination of $N_1, \ldots, N_l$. Since $r > 0, r \in S^+$. Moreover, $r < m$, contradicting the minimality of $m$.

This means that $m$ is a positive common divisor of the $N_i$.

$\therefore m = 1$. Since $m \in S^+$, 1 can be expressed as an integer linear combination of $N_1, \ldots, N_l$. □

### 9.5.2 Proof of Pohlig-Hellman

*Proof.* Suppose $p$ is a prime and $g$ is a primitive root. Let $p-1 = q_1^{k_1} q_2^{k_2} \dots q_l^{k_l}$ be the unique prime factorization of $p-1$.

- Set $N_i = \frac{p-1}{q_i^{k_i}}$ for $i = 1, \dots, l$
- $g_i \equiv g^{N_i} \mod p$
- $x_i \equiv x^{N_i} \mod p$

Consider the problem of finding, for a given $x \mod p$ a $y$ such that:

$$g^y \equiv x \mod p$$

We regard $y$ as an integer $\mod p-1$.
Suppose for each $i = 1, \dots, l$, the integer $y_i$ solves:

$$g_i^{y_i} \equiv x_i \mod p$$

What is $\mathrm{ord}(g_i)$? Recall that $g_i = g^{(\frac{p-1}{q_i^{k_i}})}$, and $(g^{(\frac{p-1}{q_i^{k_i}})})^{q_i^{k_i}} \equiv 1 \mod p$. So $\mathrm{ord}(g_i) = q_i^{k_i}$, and hence $y_i$ is an integer modulo $q_i^{k_i}$. By the Chinese Remainder Theorem, there is an integer $y$ such that:

$$y \equiv y_i \mod q_i^{k_i} \quad i = 1, \dots, l$$

and $y$ is unique modulo $q_1^{k_1} q_2^{k_2} \dots q_l^{k_l} = p-1$.

We know check that $y$ solves the original discrete log problem. So we write:

$$\begin{aligned} g^y &\equiv g^{y \cdot 1} \mod p \\ &\equiv g^{y \cdot \sum_{i=1}^{l} c_i N_i} && \text{(Since } 1 = \sum_{i=1}^{l} c_i N_i \text{ )} \end{aligned}$$

Now, recall that $N_i = \frac{p-1}{q_i^{k_i}}$. If $\gcd(N_1, \dots, N_l) > 1$, then there exists a prime $q | N_i, i = 1, \dots, l$ and $N_i | N$ for $i = 1, \dots, l$. $\therefore q | N \quad \therefore \underline{q} = q_j$ for some $j$. **But!** We know that $\underline{q} = q_j \not| N_j$, since we already factored $q$ out of the $N_i$, and we have a <u>contradiction</u>. $\therefore \gcd(N_1, \dots, N_l) = 1$. So, back to $g^{y \cdot \sum_{i=1}^{l} c_i N_i}$.

These operations are all being performed $\mod p$

$$
\begin{aligned}
g^{y\cdot\sum_{i=1}^{l} c_i N_i} &\equiv g^{\sum_{i=1}^{l} c_i N_i \cdot y} && \text{(bringing } y \text{ into the summation)} \\
&\equiv \prod_{i=1}^{l} g^{c_i N_i \cdot y} && \text{(Converting the sum to a product)} \\
&\equiv \prod_{i=1}^{l} (g^{N_i}) c_i \cdot y \\
&\equiv \prod_{i=1}^{l} (g_i)^{c_i \cdot y} && \text{(From the definition of } g_i\text{)} \\
&\equiv \prod_{i=1}^{l} (g_i^y)^{c_i} \equiv \prod_{i=1}^{l} (g_i^{y_i})^{c_i} && \text{(definition of } y_i\text{)} \\
&\equiv \prod_{i=1}^{l} (x_i)^{c_i} \equiv \prod_{i=1}^{l} (x_i^{N_i})^{c_i} && \text{(definition of } x_i \text{ and } N_i\text{)} \\
&\equiv \prod_{i=1}^{l} (x_i)^{N_i c_i} \equiv x^{\sum_{i=1}^{l} c_i N_i} && \text{(Switch back to sum} \\
&\equiv x && \text{(Because } \sum_{i=1}^{l} c_i N_i = 1\text{)}
\end{aligned}
$$

So we can see that we can in fact stitch together all of the $y_i$ to retrieve the original $y$ which gives us the $x$ we are looking for when the order of the prime modulus $p$ factors easily into small factors.

Moral: We should avoid cryptographic schemes depending on a discrete log mod $p$ where $p-1$ (the order of a primitive root in $p$) factors into a product of small prime powers. *i.e.* where all $q_i^{k_i}$ are small. □

# 10 Sophie Germain primes

## 10.1 Pair of primes ...

**definition 23.** A pair of primes $q$ and $p$ with $q < p$ are called *Sophie Germain* primes if $p = 2 \cdot q + 1$.

## 10.2 Theorem - primitive roots of Sophi Germain primes ...

**Theorem 24.** *Let $q < p$ be a pair of Sophie Germain primes. A unit $a \mod p$ is a primitive root modulo $p$ if $a \not\equiv -1 \mod p$ and $a^q \equiv -1 \mod p$*

*Proof.* We examine the questions - what are the possible orders of elements in $\mathbb{Z}_p^*$? Since $\phi(p) = 2q$, the possible orders are $1, 2, q, 2q$, since these are the divisors of $2q$.

Fermat's Little Theorem tells us that each element of $\mathbb{Z}_p^*$ is a root to the equation

$$x^{p-1} - 1 = x^{2q} - 1$$

Note: $x^{2q} - 1 = (x^q)^2 - 1 = (x^q - 1)(x^q + 1)$
Any element of order 1 or 2 must be a root of $x^2 - 1 = (x+1)(x-1)$. $\therefore x = \pm 1$ is the only element of order 1, and $x = -1$ is the only element of order 2.
Next, any element of order $q$ must be a root of:

$$x^q - 1 = (x-1) \cdot (x^{q-1} + x^{q-2} + \ldots + 1)$$

So:

- -1 is a root of $x^q + 1$, hence it can't be a root of $x^q - 1$.
- +1 is a root of $x - 1$, hence it cannot be a root of $x^{q-1} + x^{q-2} + \ldots + 1$
- No primitive root can be a root of $x^q - 1$ since the first positive power of a primitive root that yields 1 is $2q$.
- $\therefore$ the roots of $x^{q-1} + x^{q-2} + \ldots + 1$ coincide with the elements of $\mathbb{Z}_p^*$ of order $q$.

$\therefore$ every primitive root of $\mathbb{Z}_p^*$ must be a root of $x^q + 1$.

Because $q$ is odd, we have:

$$\begin{aligned} x^{2q} - 1 =& (x^q + 1)(x^q - 1) \\ =& (x - 1)(x^{q-1} + x^{q-2} + \ldots + 1)(x + 1)(x^{q-1} - x^{q-2} + x^{q-3} - \ldots + 1) \end{aligned}$$

$\therefore$ the roots of $x^{q-1} - x^{q-2} + x^{q-3} - \ldots + 1$ are the set of primitive roots.
$\therefore$ when $a$ is an element of $\mathbb{Z}_p^*$ such that $a \not\equiv -1 \mod p$ and $a^q \equiv -1 \mod p$, then $a$ is a primitive root. □

# 11 Miller-Rabin test

The Miller-Rabin test is a more robust test for "pseudoprimality" than the Fermat pseudoprime test.

## 11.1 Pseudoprimality test

In general, when testing if a particular number $a$ is pseudoprime with respect to a range of other numbers, the more numbers we can find that $a$ is pseudoprime to, the higher the likelihood that $a$ is actually prime.

## 11.2 Fermat's pseudoprimes

A positive integer $n$ is a Fermat pseudoprime with respect to $a \in \mathbb{Z}$ if $a^n \equiv a \mod n$. Note that if $n$ is prime, it is pseudoprime relative to every integer $a \in \mathbb{Z}$. If this fails for a particular integer $a$, then $n$ must be composite. In this case, $a$ is said to be a (Fermat) witness for the compositeness of $n$.

However, there are integers which are composite yet always pass Fermat's pseudoprimality test. These numbers are known as Carmichael numbers.

**definition 25.** A composite integer $n > 1$ is a Carmichael number if there is an $a \in \mathbb{Z}$ such that $a$ is a unit $\mod n$ and $a^n \equiv a \mod n$.

*Remark* 26. Such numbers exist! The first one is n = 561. There are infinitely many such numbers, which was proved in 1994 by Alford, Granville and Pomerance.[6]

## 11.3 Miller-Rabin pseudoprimality test

Let $n$ be an odd integer with $n > 1$. The integer $n - 1$ may be expressed uniquely as:

$$n - 1 = 2^l m$$

Where $m$ is odd.

Let $a$ be an integer with $\gcd(a, n) = 1$. The integer $n$ is a (Miller-Rabin) pseudoprime with respect to $a$ if the list:

$$a^m \mod n, a^{2m} \mod n, a^{4m} \mod n, \ldots, a^{2^l m} \mod n$$

has exactly one of the following properties:

- The list consists entirely of 1's
- It has -1 somewhere before the last entry

[6] https://math.dartmouth.edu/ carlp/PDF/paper95.pdf

## 11.4 Thm - every prime p is a Miller-Rabin pseudoprime with respect to any a not divisible by p

*Proof.* Let $n = p$, and odd prime, and $a \in \mathbb{Z}$ with $p \not| a$.
Recall: $p - 1 = 2^l m$ where $l \geq 1$ and $m$ is odd.
Consider the list:

$$\underline{a}^m, \underline{a}^{2m}, \underline{a}^{4m}, \ldots, \underline{a}^{2^l m}$$

Note that $\underline{a}^{2^l m} = a^{p-1} = 1 \mod p$
Write $k$ for the smallest integer with $0 \leq k \leq l$ and $\underline{a}^{2^k m} = 1 \mod p$
If $k = 0$, we are in the first case, since each subsequent entry in the list is the square of its predecessor, and the entire list is 1's. If $k > 0$, note that $1 = \underline{a}^{2^k m} = (\underline{a}^{2^{k-1} m})^2$. So

$$\underline{a}^{2^{k-1} m} = \pm 1 \mod p$$

But

$$\underline{a}^{2^{k-1} m} \neq +1 \mod p$$

Because of the definition of k as the minimal exponent that equals 1.

$$\therefore \underline{a}^{2^{k-1} m} = -1 \mod p$$

Since $\underline{a}^{2^{k-1} m}$ is in the list, we are in the second case, where -1 occurs somewhere before the last entry in the list. □

*Remark* 27. If $n$ fails the Miller-Rabin test for $a \in \mathbb{Z}$, then $a$ is said to be a (Miller-Rabin) witness for the compositeness of $n$. It is known that a composite $n$ (which is large) has approximately $0.75 \cdot n$ witnesses in $\mathbb{Z}_n^*$.

## 11.5 Heuristic Argument for Primality

We make the assumption that the events:

- $a$ is a witness for $n$ and
- $b$ is a witness for $n$

Are independent if $a \not\equiv b \mod n$. That is, if $a$ and $b$ are different congruence classes $\mod n$.

Suppose $n$ is composite and we have found that $a_1, \ldots, a_l$ are distinct $\mod n$ and none are witnesses for the compositeness of $n$. Then a rough estimate for the probability of this is:

$$\Pr(a_1, \ldots, a_l \text{ are not witnesses} \mid \text{n is composite }) = 0.25^l$$

We can see that the probability of a number being composite gets very small very quickly as subsequent tests fail. In fact, if five $a$ are chosen at random and none are witnesses, then the probability that $n$ is composite would be $0.25^5 = 0.00097$. However, this isn't deductive reasoning. It's seat-of-the-pants, plausible, engineering style reasoning. Which is basically the definition of a Heuristic! Appendix - Number Theory

Algorithms written by Dr. J. Ramanathan in Python for Math 592

```
In [1]: """

        Number Theory Routines
        (written in python3)

        """
        from math import sqrt, floor
        from fractions import gcd
        from matplotlib.pyplot import *

        # Fast version of the sieve that only filters multiples of
        # primes up to sqrt(n).

        def fastErat(n):
                """ Fast version of the sieve of Eratosthenes. Only multiple
                of primes up to sqrt of n are filtered."""
                primes = []
                if n < 1: return(primes)
                testbnd = sqrt(n + 0.000001)
                nxtpr = 1
                lst = list(range(2,n+1))
                while (lst != []) and (nxtpr <= testbnd) :
                        nxtpr = lst[0]
                        primes.append(nxtpr)
                        lst = [x for x in lst if (x%nxtpr) != 0]
                return(primes+lst)

        # Fast exponentiation mod p

        def modp(x,y,n):
                """ Fast modular exponentiation: computes x**y mod n"""
                acc, s, t = 1, x, y
                while t != 0:
                        if (t & 1) == 1: acc = (s*acc)%n
                        t = t >> 1
                        s = (s**2)%n
                return(acc)

        # Bezout's theorem

        def bezout(a,b):
                """ Computes the linear combination of a and b that
                yields the gcd."""
                if a<=0 or b<=0: return("Error") # a and b must be positive
                m00,m01 = 1,0
                m10,m11 = 0,1
                r0,r1 = a,b
                while r1 > 0:
                        q, rnext = r0//r1, r0%r1
                        m00,m10 = m10,m00-q*m10
                        m01,m11 = m11,m01-q*m11
                        r0,r1 = r1,rnext
```

```
        return(r0,m00,m01)

# Direct test for primality

def isPrime(n):
        """ Tests for primality by looking for a prime factor
        less than sqrt(n)."""
        sqn = floor(sqrt(n))
        smP = iter(fastErat(sqn))
        for p in smP:
            if n%p == 0:
                return(False)
        return(True)

# A simple factorization algorithm

def simpleFac(n):
        """ Computes the factorization of n."""
        sqn = floor(sqrt(n))
        res = n
        fac = dict({})
        smP = fastErat(sqn)
        for p in smP:
            while res%p == 0:
                if p in fac: fac[p] += 1
                else: fac[p] = 1
                res = res//p
        if res != 1:
            fac[res] = 1
        return(fac)

# Checks for Fermat pseudo-primality

def pseudoPrime(n,a):
    """ Is n a Fermat pseudoprime relative to a."""
    if ((modp(a,n,n) - a)%n == 0):
        return(True)
    else:
        return(False)

# Order of a unit

def order(g,n):
    """ Computes the order of a unit g modula n."""
    if gcd(g,n) != 1:
        return "g is not relavtively prime to n."
    k = 1
    a = g%n
    while a != 1:
        k += 1
        a = (a*g)%n
    return(k)

# Powers of a unit
```

```
def orbit(g,n):
    """ Returns the cycle generated by a unit  modulo n."""
    if gcd(g,n) != 1:
        return "g is not relavtively prime to n."
    k = 1
    val = [1]
    a = g
    while a != 1:
        val.append(a)
        k += 1
        a = (a*g)%n
    return(val)

# Computes the units

def units(n):
    return( [k for k in range(0,n) if gcd(k,n) == 1])

# Tiotient function

def tiotient(n):
    return(len(units(n)))

# Shank's Discrete Logarithm Algorithm

def dlog(x,g,n):
    (d,ginv,y) = bezout(g,n)
    if d != 1: return("Error.")
    sqn = int(sqrt(n)+1)
    A = modp(ginv,sqn,n)
    steps = {(x*modp(A,i,n))%n:i for i in range(0,sqn)}
    j,lhs=0,1
    while not lhs in steps.keys() and j < sqn:
        j,lhs = j+1,(lhs*g)%n
    if j == sqn: return("Error")
    return((steps[lhs]*sqn+j)%tiotient(n))
```