Fermat and Euler's Theorems

Definition: A *reduced set of residues* (RSR) *modulo* m is a set of integers R so that every integer relatively prime to m is congruent to exactly one integer in R.

Fact. $a \equiv b \pmod{m}$ implies $\gcd(a, m) = \gcd(b, m)$.

Fact. All RSR's modulo m have the same size.

Definition: $\phi(m)$ is the size of a RSR modulo m. ϕ is called the *Euler Phi or totient function*.

The standard CSR modulo m is $\{0, \ldots, m-1\}$.

The standard RSR modulo m is

$$\{1 \leq r \leq m; \gcd(r, m) = 1\}.$$

Example: $\phi(12) = 4$ because $\{1, 5, 7, 11\}$ is the standard RSR modulo 12.

Fact. ϕ is *multiplicative*, that is, $\phi(ab) = \phi(a)\phi(b)$ whenever $\gcd(a, b) = 1$.

Some special formulas for ϕ: Let p be prime. Then

$$\phi(p) = p - 1,$$

$$\phi(p^\alpha) = p^\alpha - p^{\alpha-1},$$

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right).$$

When $p \neq q$ are primes, we have

$$\phi(pq) = (p-1)(q-1).$$

Proof: Begin with the CSR $\{0, 1, \ldots, pq-1\}$. Delete all q multiples of p. Delete all p multiples of q. 0 was deleted twice, so add 1 back. We get $\phi(pq) = pq - p - q + 1 = (p-1)(q-1)$.

Fermat's "Little" Theorem

Theorem. Let p be prime and a be an integer which is not a multiple of p. Then

$$a^{p-1} \equiv 1 \pmod{p}.$$

Proof: Since $\gcd(a, p) = 1$, the set $\{ai \bmod p; i = 1, \ldots, p-1\}$ is the same as the set $\{1, \ldots, p-1\}$. Therefore,

$$a^{p-1} \prod_{i=1}^{p-1} i = \prod_{i=1}^{p-1} (ai) \equiv \left(\prod_{i=1}^{p-1} i \right) \cdot 1 \pmod{p}.$$

Since $\gcd\left(\prod_{i=1}^{p-1} i, p\right) = 1$, we can cancel and get $a^{p-1} \equiv 1 \pmod{p}$.

Example. 97 is prime and 2 is not a multiple of 97, so $2^{96} \equiv 1 \pmod{97}$.

Euler's Theorem

Theorem. Let $m > 1$ and $\gcd(a, m) = 1$. Then

$$a^{\phi(m)} \equiv 1 \pmod{m}.$$

Proof: Let $\{r_1, \ldots, r_{\phi(m)}\}$ be a RSR modulo m. Then $\{ar_1, \ldots, ar_{\phi(m)}\}$ is a RSR modulo m, too. Therefore, for all i, there is a unique j so that $r_i \equiv ar_j \pmod{m}$. Then

$$a^{\phi(m)} \prod_{i=1}^{\phi(m)} r_i = \prod_{i=1}^{\phi(m)} (ar_i) \equiv \left(\prod_{i=1}^{\phi(m)} r_i \right) \pmod{m}.$$

Since $\gcd\left(\prod_{i=1}^{\phi(m)} r_i, m\right) = 1$, we can cancel and get $a^{\phi(m)} \equiv 1 \pmod{m}$.

Example. Let $m = 13 \times 23 = 299$, where 13 and 23 are primes. Then

$\phi(m) = \phi(299) = (13-1)(23-1) = 12 \times 22 = 264.$

Note that $\gcd(5, 299) = 1$. Euler's Theorem says $5^{264} \equiv 1 \pmod{299}$, that is,
$299 \mid (5^{264} - 1)$.

Example of the use of Euler's theorem.

Find the two low-order decimal digits of 33862513^{119442}.

First, $33862513 \equiv 13 \pmod{100}$, so the answer is the same as the two low-order decimal digits of 13^{119442}
(because $(100k + 13)^n \equiv 13^n \pmod{100}$ and the two low-order decimal digits of m are m mod 100).

Second,

$$\phi(100) = \phi(2^2)\phi(5^2) = 2(2-1)\cdot 5(5-1) = 40.$$

Now $119442 \equiv 2 \pmod{40}$, so by Euler, $13^{119442} \equiv 13^2 \pmod{100}$.

Finally, $33862513^{119442} \equiv 13^{119442} \equiv 13^2 = 169 \equiv 69 \pmod{100}$, and the two low-order decimal digits of 33862513^{119442} are 69.

A Corollary of Euler's Theorem

Here is an alternate way to compute the multiplicative inverse a^{-1} of a modulo m: Recall that a^{-1} is the residue class mod m such that $a^{-1}a \equiv aa^{-1} \equiv 1 \pmod{m}$. It is defined only when $\gcd(a, m) = 1$. In that situation we have $a^{\phi(m)} \equiv 1 \pmod{m}$ by Euler's Theorem.

Factoring out one a gives

$$a \cdot a^{\phi(m)-1} \equiv 1 \pmod{m},$$

whence $a^{-1} \equiv a^{\phi(m)-1} \pmod{m}$. For a prime modulus p we have $a^{-1} \equiv a^{p-2} \pmod{p}$.

For large m, computing a^{-1} mod m by this formula requires roughly the same number of bit operations as computing a^{-1} mod m by the Extended Euclidean Algorithm. (The latter must be used if one does not know $\phi(m)$.)

How to compute a^n mod m swiftly

Here is an algorithm for computing a^n mod m in $O(\log_2 n)$ multiplications.

```
procedure power(a,n,m)
e = n;
y = 1;
z = a;
repeat {
         if (e is odd) y = (y*z)%m;
         if (e <= 1) return (y);
         z = (z*z)%m;
         e = floor(e/2);
         }
end power;
```

Finding large primes

Fermat's Little Theorem says that if p is prime and p does not divide a, then $a^{p-1} \equiv 1 \pmod{p}$.

This theorem gives a test for *compositeness*: If p is odd and p does not divide a and $a^{p-1} \not\equiv 1 \pmod{p}$, then p is not prime.

If the converse of Fermat's theorem were true, it would give a fast test for *primality*. The converse would say, if p is odd and p does not divide a and $a^{p-1} \equiv 1 \pmod{p}$, then p is prime.

Unfortunately, this converse is not a true statement, although it is true for most p and most a. Consider $p = 341 = 11 \times 31$ and $a = 2$. We have $2^{340} \equiv 1 \pmod{341}$.

The test, "Is $2^{m-1} \equiv 1 \pmod{m}$?" is widely used as a test for primality of very large odd numbers m, as the probability that it fails is incredibly small.

Now we will see some applications of number theory to cryptography.

A *cipher* is a way of converting ordinary text M, called *plaintext*, into meaningless symbols C, called *ciphertext*, and converting it back to plaintext under the control of a *key* K.

Here M and C are strings of letters or bits, and K is a number or a bit string.

The conversion of plaintext to ciphertext is called *encryption* and is written $C = E_K(M)$.

The conversion of ciphertext back to plaintext is called *decryption* and is written $M = D_K(C)$.

A basic property of ciphers is that $D_K(E_K(M)) = M$ for every M.

Another important property of ciphers is that if you know C, but not M or K, then it should be hard to find M or K.

The ciphers just described are the *one-key* or *symmetric* ciphers. They use the same key to decipher as to encipher. Until the 1970s, all known ciphers were of this type.

The Caesar rotate-the-alphabet cipher is a simple one-key cipher. The key K is the amount of rotation of the alphabet.

Modern one-key ciphers usually use bit operations, like shift and xor, to achieve high speed for the enciphering and deciphering algorithms.

Some modern one-key ciphers are the Data Encryption Standard, DES, and the Advanced Encryption Standard, AES.

Around 1980, *two-key* or *asymmetric* ciphers were invented.

They use different, but related, keys for enciphering and deciphering: $C = E_{K_1}(M)$ and $M = D_{K_2}(C)$.

Of course, if K_1 and K_2 are the correct keys, then $M = D_{K_2}(E_{K_1}(M))$ for every M.

The remarkable property of two-key ciphers is that if you know the enciphering key K_1, then you cannot easily find the deciphering key K_2. In fact, K_1 and the enciphering algorithm E are made public. Hence asymmetric ciphers are also called *public-key* ciphers.

Most public-key ciphers use arithmetic with large numbers and their algorithms are slow compared those of to one-key ciphers.

RSA (Rivest-Shamir-Adleman) and ElGamal are examples of public-key ciphers.

Here is a typical use of a public-key cipher.

Suppose Alice wants to email a long secret letter M to Bob. If the two have previously agreed on a secret AES key, Alice would just encipher M using AES with that key and send the ciphertext to Bob.

But if Alice and Bob did not share a secret AES key, then Alice could chose a random AES key K, encipher M using AES and K, and send the ciphertext C to Bob. She would then find Bob's public enciphering key K_1 from Bob's web page, say, and send $C_1 = E_{K_1}(K)$ to Bob.

Bob would decipher C_1 with $D_{K_2}(C_1) = K$, where K_2 is Bob's secret deciphering key. Then Bob would use AES and K to decipher C.

Note that it does not matter that the public-key cipher is slow because it used only to transmit the very short message K and not the long message M.

The first application is not a cipher but rather a way for Alice and Bob to choose a common AES key. It uses fast exponentiation and congruences, but not Euler's theorem.

Diffie-Hellman key-exchange protocol

This protocol allows two users to choose a common secret key, for DES or AES, say, while communicating over an insecure channel (with eavesdroppers).

The two users agree on a common large prime p and a constant value a, which may be publicly known and available to everyone. It is best if the smallest exponent $e > 0$ for which $a^e \equiv 1 \pmod{p}$ is $e = p - 1$, but the protocol will work if $e < p - 1$ provided e is still large.

Alice secretly chooses a random x_A in $0 < x_A < p-1$ and computes $y_A = a^{x_A} \bmod p$.
Bob secretly chooses a random x_B in $0 < x_B < p-1$ and computes $y_B = a^{x_B} \bmod p$.

Alice sends y_A to Bob. Bob sends y_B to Alice. An eavesdropper, knowing p and a, and seeing y_A and y_B, cannot compute x_A or x_B from this data unless he can solve the Discrete Logarithm Problem quickly. (See below.)

Alice computes $K_A = y_B^{x_A} \bmod p$.

Bob computes $K_B = y_A^{x_B} \bmod p$.

Then

$$K_A \equiv a^{x_A \cdot x_B} \equiv K_B \pmod{p}$$

and $0 < K_A, K_B < p$, so $K_A = K_B$.

Alice and Bob choose certain agreed-upon bits from K_A to use as their key for a single-key cipher like DES or AES.

Although this protocol provides secure communication between Alice and whoever is at the other end of the communication line, it does not prove that Bob is the other party. To guarantee that Bob is at the other end, they would have to use a signature system like RSA.

Discrete Logarithms

The Diffie-Hellman key exchange and several other crypto algorithms could all be broken if we could compute discrete logarithms quickly, that is, if we could easily solve the exponential congruence $a^x \equiv b \bmod p$.

By analogy to ordinary logarithms, we may write $x = \log_a b$ when p is understood from the context. These *discrete logarithms* enjoy many properties of ordinary logarithms, such as $\log_a bc = \log_a b + \log_a c$, except that the arithmetic with logarithms must be done modulo $p - 1$ because $a^{p-1} \equiv 1 \bmod p$.

The RSA public-key cipher

Rivest-Shamir-Adleman. Let $n = pq$ be the product of two large primes.
Then $\phi(n) = \phi(pq) = (p-1)(q-1)$. Choose a random e in $1 < e < n-1$ with $\gcd(e, \phi(n)) = 1$. Use Extended Euclid to compute $d = e^{-1} \bmod \phi(n)$, so that $ed \equiv 1 \pmod{(p-1)(q-1)}$.

Encode plaintext as (blocks) $0 \leq M < n$.
Encipher M as $C = E(M) = M^e \bmod n$.
Decipher C as $M = D(C) = C^d \bmod n$.

This works, that is, $D(E(M)) = M$ for all M in $0 \leq M < n$, provided that $ed \equiv 1 \pmod{\phi(n)}$. Write $ed = 1 + k\phi(n)$. Then $D(E(M)) \equiv (M^e)^d = M^{ed} = M^{1+k\phi(n)} = M \cdot (M^{\phi(n)})^k \equiv M \pmod{n}$, since $M^{\phi(n)} \equiv 1 \pmod{n}$ by Euler's Theorem.

Each user of RSA has her own set of keys: Make n and e public, but keep d secret. The factors p and q are not needed after e and d are computed, but in any case should not be revealed.

If many users wish to communicate securely in pairs, then RSA requires fewer total keys to be stored than DES or AES.

Cryptanalysis: Since n is public and one can easily compute d from e and the factors of n, a direct approach to breaking RSA is to factor n. Using the best currently-known methods, this is about as hard as solving the Discrete Logarithm Problem with the same sized modulus. For a modulus n of 400 decimal digits, this is too hard for current algorithms and computers.

Pohlig-Hellman cipher

This is NOT a public-key cipher.

Let $n = p =$ prime. Then $\phi(p) = p - 1$ and $ed \equiv 1 \pmod{p-1}$.

Keep all of p, e, d secret. All three are the "key". There is just one user or one pair of users.

Encode plaintext as (blocks) $0 \leq M < p$.
Encipher M as $C = E(M) = M^e \bmod p$.
Decipher C as $M = D(C) = C^d \bmod p$.

This works, that is, $D(E(M)) = M$ for all M in $0 \leq M < p$, provided that $ed \equiv 1 \pmod{\phi(p)}$. Write $ed = 1 + k\phi(p)$. Then $D(E(M)) \equiv (M^e)^d = M^{ed} = M^{1+k\phi(p)} = M \cdot (M^{\phi(p)})^k \equiv M \pmod{p}$, since $M^{\phi(p)} \equiv 1 \pmod{p}$ by Euler's Theorem.

RSA Signatures

RSA has no direct authentication: Anyone can send any message to you and claim it came from anyone. However, one can sign RSA messages as follows:

Suppose both Alice and Bob have complete RSA public-key ciphers, with different primes, moduli, and exponents. Write $E_A(M) = M^{e_A} \bmod n_A$ and $D_A(C) = C^{d_A} \bmod n_A$ for Alice's RSA enciphering and deciphering functions, where n_A is the product of Alice's two secret primes and e_A and d_A are Alice's enciphering and deciphering exponents. Likewise define Bob's enciphering function $E_B(\cdot)$ and his deciphering function $D_B(\cdot)$.

Suppose $n_A < n_B$. Then Alice can sign (and encipher) a message M to Bob by sending $C = E_B(D_A(M))$ to Bob. Bob can decipher C by applying D_B to it (to get $D_A(M)$) and then check the signature by applying E_A to the latter.

In case $n_A > n_B$, Alice can sign (and encipher) a message M to Bob by sending $C = D_A(E_B(M))$ to Bob. Bob checks it by applying E_A to C and then D_B to the result.

In both cases, Bob and only Bob knows $D_B(\cdot)$, so only Bob can do that part of the calculation. Bob obtains Alice's enciphering function $E_A(\cdot)$ from a public directory.

Discrete Logarithms

The Diffie-Hellman key exchange, the ElGamal public key cryptosystem, the Pohlig-Hellman private key cryptosystem and the Digital Signature Algorithm could all be broken if we could compute discrete logarithms quickly, that is, if we could solve the exponential congruence $a^x \equiv b \bmod p$ easily.

By analogy to ordinary logarithms, we may write $x = \log_a b$ when p is understood from the context. These discrete logarithms enjoy many properties of ordinary logarithms, such as $\log_a bc = \log_a b + \log_a c$, except that the arithmetic with logarithms must be done modulo $p - 1$ because $a^{p-1} \equiv 1 \bmod p$.

Neglecting powers of $\log p$, the congruence may be solved in $O(p)$ time and $O(1)$ space by raising a to successive powers modulo p and comparing each with b.

It may also be solved in $O(1)$ time and $O(p)$ space by looking up x in a precomputed table of pairs $(x, a^x \bmod p)$ sorted by the second coordinate.

Shanks' "giant step–baby step" algorithm is a meet-in-the-middle method which solves the congruence in $O(\sqrt{p})$ time and $O(\sqrt{p})$ space as follows.

Let $m = \lceil \sqrt{p-1} \rceil$.

Compute and sort the m ordered pairs $(j, a^{mj} \bmod p)$, for j from 0 to $m-1$, by the second coordinate.

Compute and sort the m ordered pairs $(i, ba^{-i} \bmod p)$, for i from 0 to $m-1$, by the second coordinate.

Find a pair (j, y) in the first list and a pair (i, y) in the second list.

This search will succeed because every integer between 0 and $p-1$ can be written as a two-digit number ji in radix m.

Finally, $x = mj + i \bmod p - 1$.

Example: Solve $5^x \equiv 44 \pmod{97}$.

We have $p = 97$, $m = \lceil \sqrt{97-1} \rceil = 10$, $a = 5$, $b = 44$. Then $a^m \equiv 5^{10} \equiv 53 \pmod{97}$ and $a^{-1} \equiv a^{95} \equiv 39 \pmod{97}$.

j	$a^{mj} \bmod p$	i	$b \cdot a^{-i} \bmod p$
j	$53^j \bmod 97$	i	$44 \cdot 39^i \bmod 97$
0	1	0	44
1	53	1	67
2	93	2	91
3	79	3	57
4	16	4	89
5	72	5	76
6	33	6	54
7	3	7	69
8	62	8	72
9	85	9	92

Note that 72 is in the second and fourth columns, so $5^{50} \equiv 53^5 \equiv 72 \equiv 44 \cdot 39^8 \equiv 44 \cdot 5^{-8} \pmod{97}$. This shows that $x = 5 \cdot 10 + 8 = 58$.