CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 8
February 1, 2012

Number Theory Needed for RSA

$\mathbf{Z}_n$: The integers mod n

- Modular arithmetic
- GCD
- Relatively prime numbers, $\mathbf{Z}_n^*$, and $\phi(n)$

Computing in $\mathbf{Z}_n$

- Modular multiplication
- Modular inverses
- Extended Euclidean algorithm

Generating RSA Encryption and Decryption Exponents

Number Theory Needed for RSA

Number theory needed for RSA

Here's a summary of the number theory needed to understand RSA and its associate algorithms.

- Greatest common divisor, $\mathbf{Z}_n$, mod n, $\phi(n)$, $\mathbf{Z}_n^*$, and how to add, subtract, multiply, and find inverses mod n.
- Euler's theorem: $a^{\phi(n)} \equiv 1 \pmod{n}$ for $a \in \mathbf{Z}_n^*$.
- How to generate large prime numbers: density of primes and testing primality.

How these facts apply to RSA

- The RSA key pair (e, d) is chosen to satisfy the *modular equation* $ed \equiv 1 \pmod{\phi(n)}$.
- To find (e, d), we repeatedly choose e at random from $\mathbf{Z}_n$ until we find one in $\mathbf{Z}_n^*$, and then *solve* the modular equation $ed \equiv 1 \pmod{\phi(n)}$ for d. We compute gcd to test for membership in $\mathbf{Z}_n^*$.
- Using *Euler's theorem*, we can show $m^{ed} \equiv m \pmod{n}$ for all $m \in \mathbf{Z}_n^*$. This implies $D_d(E_e(m)) = m$. To show that decryption works even in the rare case that $m \in \mathbf{Z}_n - \mathbf{Z}_n^*$ requires some more number theory that we will omit.
- To find p and q, we choose large numbers and *test each for primality* until we find two distinct primes. We must show that the *density of primes* is large enough for this procedure to be feasible.

$\mathbf{Z}_n$: The integers mod n

The mod relation

We just saw that mod is a binary operation on integers.

Mod is also used to denote a relationship on integers:

$$a \equiv b \pmod{n} \quad \text{iff} \quad n \mid (a - b).$$

That is, a and b have the same remainder when divided by n. An immediate consequence of this definition is that

$$a \equiv b \pmod{n} \quad \text{iff} \quad (a \bmod n) = (b \bmod n).$$

Thus, the two notions of mod aren't so different after all!

We sometimes write $a \equiv_n b$ to mean $a \equiv b \pmod{n}$.

Mod is an equivalence relation

The two-place relationship $\equiv_n$ is an *equivalence relation*.

Its equivalence classes are called *residue* classes modulo n and are denoted by $[b]_{\equiv_n} = \{a \mid a \equiv b \pmod{n}\}$ or simply by $[b]$.

For example, if $n = 7$, then $[10] = \{\ldots - 11, -4, 3, 10, 17, \ldots\}$.

Fact

$[a] = [b]$ iff $a \equiv b \pmod{n}$.

Canonical names

If $x \in [b]$, then x is said to be a *representative* or *name* of the equivalence class $[b]$. Obviously, b is a representative of $[b]$. Thus, $[-11]$, $[-4]$, $[3]$, $[10]$, $[17]$ are all names for the same equivalence class.

The *canonical* or preferred name for the class $[b]$ is the unique integer in $[b] \cap \{0, 1, \ldots, n-1\}$.

Thus, the canonical name for $[10]$ is $10 \bmod 7 = 3$.

Mod is a congruence relation

The relation $\equiv_n$ is a *congruence relation* with respect to addition, subtraction, and multiplication of integers.

Fact

For each arithmetic operation $\odot \in \{+, -, \times\}$, if $a \equiv a' \pmod{n}$ and $b \equiv b' \pmod{n}$, then

$$a \odot b \equiv a' \odot b' \pmod{n}.$$

The class containing the result of $a \odot b$ depends only on the classes to which a and b belong and not the particular representatives chosen.

Hence, we can perform arithmetic on equivalence classes by operating on their names.

Greatest common divisor

Definition

The *greatest common divisor* of two integers a and b, written $\gcd(a, b)$, is the largest integer d such that $d \mid a$ and $d \mid b$.

$\gcd(a, b)$ is always defined unless $a = b = 0$ since 1 is a divisor of every integer, and the divisor of a non-zero number cannot be larger (in absolute value) than the number itself.

Question: Why isn't $\gcd(0, 0)$ well defined?

Computing the GCD

$\gcd(a, b)$ is easily computed if a and b are given in factored form.

Namely, let p_i be the i^{th} prime. Write $a = \prod p_i^{e_i}$ and $b = \prod p_i^{f_i}$. Then

$$\gcd(a, b) = \prod p_i^{\min(e_i, f_i)}.$$

Example: $168 = 2^3 \cdot 3 \cdot 7$ and $450 = 2 \cdot 3^2 \cdot 5^2$, so $\gcd(168, 450) = 2 \cdot 3 = 6$.

However, factoring is believed to be a hard problem, and no polynomial-time factorization algorithm is currently known. (If it were easy, then Eve could use it to break RSA, and RSA would be of no interest as a cryptosystem.)

Euclidean algorithm

Fortunately, $\gcd(a, b)$ can be computed efficiently without the need to factor a and b using the famous *Euclidean algorithm*.

Euclid's algorithm is remarkable, not only because it was discovered a very long time ago, but also because it works without knowing the factorization of a and b.

Euclidean identities

The Euclidean algorithm relies on several identities satisfied by the gcd function. In the following, assume $a > 0$ and $a \geq b \geq 0$:

$$\begin{aligned} \gcd(a, b) &= \gcd(b, a) \quad (1) \\ \gcd(a, 0) &= a \quad (2) \\ \gcd(a, b) &= \gcd(a - b, b) \quad (3) \end{aligned}$$

Identity 1 is obvious from the definition of gcd. Identity 2 follows from the fact that every positive integer divides 0. Identity 3 follows from the basic fact relating divides and addition from lecture 7.

Computing GCD without factoring

The Euclidean identities allow the problem of computing $\gcd(a, b)$ to be reduced to the problem of computing $\gcd(a - b, b)$.

The new problem is "smaller" as long as $b > 0$.

The *size* of the problem $\gcd(a, b)$ is $|a| + |b|$, the sum of the two arguments. This leads to an easy recursive algorithm.

```
int gcd(int a, int b)
{
  if ( a < b ) return gcd(b, a);
  else if ( b == 0 ) return a;
  else return gcd(a-b, b);
}
```

Nevertheless, this algorithm is not very efficient, as you will quickly discover if you attempt to use it, say, to compute $\gcd(1000000, 2)$.

Repeated subtraction

Repeatedly applying identity (3) to the pair (a, b) until it can't be applied any more produces the sequence of pairs

$$(a, b), (a - b, b), (a - 2b, b), \ldots, (a - qb, b).$$

The sequence stops when $a - qb < b$.

How many times you can subtract b from a while remaining non-negative?

Using division in place of repeated subtractions

The number of times is the quotient $\lfloor a/b \rfloor$.

The amout $a - qb$ that is left after q subtractions is just the remainder $a \bmod b$.

Hence, one can go directly from the pair (a, b) to the pair $(a \bmod b, b)$, giving the identity

$$\gcd(a, b) = \gcd(a \bmod b, b). \tag{4}$$

Full Euclidean algorithm

Recall the inefficient GCD algorithm.

```
int gcd(int a, int b) {
  if ( a < b ) return gcd(b, a);
  else if ( b == 0 ) return a;
  else return gcd(a-b, b);
}
```

The following algorithm is exponentially faster.

```
int gcd(int a, int b) {
  if ( b == 0 ) return a;
  else return gcd(b, a%b);
}
```

Principal change: Replace `gcd(a-b,b)` with `gcd(b, a%b)`.

Besides collapsing repeated subtractions, we have $a \geq b$ for all but the top-level call on $\gcd(a, b)$. This eliminates roughly half of the remaining recursive calls.

Complexity of GCD

The new algorithm requires at most in $O(n)$ stages, where n is the sum of the lengths of a and b when written in binary notation, and each stage involves at most one remainder computation.

The following iterative version eliminates the stack overhead:

```
int gcd(int a, int b) {
  int aa;
  while (b > 0) {
    aa = a;
    a = b;
    b = aa % b;
  }
  return a;
}
```

Relatively prime numbers

Two integers a and b are *relatively prime* if they have no common prime factors.

Equivalently, a and b are *relatively prime* if $\gcd(a, b) = 1$.

Let $\mathbf{Z}_n^*$ be the set of integers in $\mathbf{Z}_n$ that are relatively prime to n, so

$$\mathbf{Z}_n^* = \{a \in \mathbf{Z}_n \mid \gcd(a, n) = 1\}.$$

Euler's totient function $\phi(n)$

$\phi(n)$ is the cardinality (number of elements) of $\mathbf{Z}_n^*$, i.e.,

$$\phi(n) = |\mathbf{Z}_n^*|.$$

Properties of $\phi(n)$:

1. If p is prime, then
$$\phi(p) = p - 1.$$
2. More generally, if p is prime and $k \geq 1$, then
$$\phi(p^k) = p^k - p^{k-1} = (p-1)p^{k-1}.$$
3. If $\gcd(m, n) = 1$, then
$$\phi(mn) = \phi(m)\phi(n).$$

Example: $\phi(26)$

Can compute $\phi(n)$ for all $n \geq 1$ given the factorization of n.

$$\begin{aligned}
\phi(126) &= \phi(2) \cdot \phi(3^2) \cdot \phi(7) \\
&= (2-1) \cdot (3-1)(3^{2-1}) \cdot (7-1) \\
&= 1 \cdot 2 \cdot 3 \cdot 6 = 36.
\end{aligned}$$

The 36 elements of $\mathbf{Z}_{126}^*$ are:

1, 5, 11, 13, 17, 19, 23, 25, 29, 31, 37, 41, 43, 47, 53, 55, 59, 61, 65, 67, 71, 73, 79, 83, 85, 89, 95, 97, 101, 103, 107, 109, 113, 115, 121, 125.

A formula for $\phi(n)$

Here is an explicit formula for $\phi(n)$.

Theorem

Write n in factored form, so $n = p_1^{e_1} \cdots p_k^{e_k}$, where $p_1, \ldots, p_k$ are distinct primes and $e_1, \ldots, e_k$ are positive integers.[1] Then

$$\phi(n) = (p_1 - 1) \cdot p_1^{e_1 - 1} \cdots (p_k - 1) \cdot p_k^{e_k - 1}.$$

For the product of distinct primes p and q,

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

---

[1] By the fundamental theorem of arithmetic, every integer can be written uniquely in this way up to the ordering of the factors.

Computing in $\mathbf{Z}_n$

Multiplication modulo n

Theorem

$\mathbf{Z}_n^*$ is closed under multiplication modulo n.

This says, if a and b are both in $\mathbf{Z}_n^*$, then $(ab \bmod n)$ is also in $\mathbf{Z}_n^*$.

Proof.

If neither a nor b share a prime factor with n, then neither does their product ab. $\square$

Example: Multiplication in $\mathbf{Z}^*_{26}$

Let $n = 26 = 2 \cdot 13$. Then

$$\mathbf{Z}^*_{26} = \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$$
$$\phi(26) = |\mathbf{Z}^*_{26}| = 12.$$

Multiplication examples:

$$5 \times 7 \bmod 26 = 35 \bmod 26 = 9.$$

$$3 \times 25 \bmod 26 = 75 \bmod 26 = 23.$$

$$9 \times 3 \bmod 26 = 27 \bmod 26 = 1.$$

We say that 3 is the *multiplicative inverse* of 9 in $\mathbf{Z}^*_{26}$.

Example: Inverses the elements in $\mathbf{Z}^*_{26}$.

x	1	3	5	7	9	11	15	17	19	21	23	25
x^{-1}	1	9	21	15	3	19	7	23	11	5	17	25
$\equiv_n$	1	9	-5	-11	3	-7	7	-3	11	5	-9	-1

Bottom row gives equivalent integers in range $[-12, \ldots, 13]$.

Note that $(26 - x)^{-1} = -x^{-1}$.

Hence, last row reads same back to front except for change of sign.

Once the inverses for the first six numbers are known, the rest of the table is easily filled in.

Finding modular inverses

Let $u \in \mathbf{Z}_n^*$. We wish to find u^{-1} modulo n.

By definition, u^{-1} is the element $v \in \mathbf{Z}_n^*$ (if it exists) such that

$$uv \equiv 1 \pmod{n}.$$

This equation holds iff $n \mid (uv - 1)$ iff $uv - 1 = qn$ for some integer q (positive or negative).

We can rewrite this equation as

$$uv - nq = 1. \tag{5}$$

u and n are given and v and q are unknowns. If we succeed in finding a solution over the integers, then v is the desired inverse u^{-1}.

Diophantine equations

A *Diophantine equation* is a linear equation in two unknowns over the integers.

$$ax + by = c \tag{6}$$

Here, a, b, c are given integers. A solution consists of integer values for the unknowns x and y that make (6) true.

We see that equation 5 fits the general form for a Diophantine equation, where

$$\begin{aligned} a &= u \\ b &= -n \\ c &= 1 \end{aligned} \tag{7}$$

Existence of solution

Theorem
The Diophantine equation

$$ax + by = c$$

has a solution over $\mathbf{Z}$ *(the integers) iff* $\gcd(a, b) \mid c$.

It can be solved by a process akin to the Euclidean algorithm, which we call the *Extended Euclidean algorithm*.

Extended Euclidean algorithm

The algorithm generates a sequence of triples of numbers $T_i = (r_i, u_i, v_i)$, each satisfying the invariant

$$r_i = au_i + bv_i \geq 0. \tag{8}$$

Extended Euclidean algorithm

The algorithm generates a sequence of triples of numbers $T_i = (r_i, u_i, v_i)$, each satisfying the invariant

$$r_i = au_i + bv_i \geq 0. \tag{8}$$

$$T_1 = \begin{cases} (a, 1, 0) & \text{if } a \geq 0 \\ (-a, -1, 0) & \text{if } a < 0 \end{cases}$$

$$T_2 = \begin{cases} (b, 0, 1) & \text{if } b \geq 0 \\ (-b, 0, -1) & \text{if } b < 0 \end{cases}$$

Extended Euclidean algorithm

$$r_i = au_i + bv_i \geq 0. \tag{8}$$

T_{i+2} is obtained by subtracting a multiple of T_{i+1} from from T_i so that $r_{i+2} < r_{i+1}$. This is similar to the way the Euclidean algorithm obtains $(a \bmod b)$ from a and b.

Extended Euclidean algorithm

$$r_i = au_i + bv_i \geq 0. \tag{8}$$

T_{i+2} is obtained by subtracting a multiple of T_{i+1} from from T_i so that $r_{i+2} < r_{i+1}$. This is similar to the way the Euclidean algorithm obtains $(a \bmod b)$ from a and b.

In detail, let $q_{i+1} = \lfloor r_i / r_{i+1} \rfloor$. Then $T_{i+2} = T_i - q_{i+1} T_{i+1}$, so

$$\begin{aligned} r_{i+2} &= r_i - q_{i+1} r_{i+1} = r_i \bmod r_{i+1} \\ u_{i+2} &= u_i - q_{i+1} u_{i+1} \\ v_{i+2} &= v_i - q_{i+1} v_{i+1} \end{aligned}$$

Extended Euclidean algorithm

$$r_i = au_i + bv_i \geq 0. \tag{8}$$

T_{i+2} is obtained by subtracting a multiple of T_{i+1} from from T_i so that $r_{i+2} < r_{i+1}$. This is similar to the way the Euclidean algorithm obtains $(a \bmod b)$ from a and b.

In detail, let $q_{i+1} = \lfloor r_i / r_{i+1} \rfloor$. Then $T_{i+2} = T_i - q_{i+1} T_{i+1}$, so

$$\begin{aligned} r_{i+2} &= r_i - q_{i+1} r_{i+1} = r_i \bmod r_{i+1} \\ u_{i+2} &= u_i - q_{i+1} u_{i+1} \\ v_{i+2} &= v_i - q_{i+1} v_{i+1} \end{aligned}$$

The sequence of generated pairs $(r_1, r_2), (r_2, r_3), (r_3, r_4), \ldots$ is exactly the same as the sequence generated by the Euclidean algorithm. We stop when $r_t = 0$. Then $r_{t-1} = \gcd(a, b)$.

Extended Euclidean algorithm

$$r_i = au_i + bv_i \geq 0. \tag{8}$$

From (8) it follows that

$$\gcd(a, b) = au_{t-1} + bv_{t-1} \tag{9}$$

Finding all solutions

Returning to the original equation,

$$ax + by = c \tag{6}$$

if $c = \gcd(a, b)$, then $x = u_{t-1}$ and $y = v_{t-1}$ is a solution.

If $c = k \cdot \gcd(a, b)$ is a multiple of $\gcd(a, b)$, then $x = ku_{t-1}$ and $y = kv_{t-1}$ is a solution.

Otherwise, $\gcd(a, b)$ does not divide c, and one can show that (6) has no solution.

Example of extended Euclidean algorithm

Suppose one wants to solve the equation

$$31x - 45y = 3 \tag{10}$$

Here, $a = 31$ and $b = -45$. We begin with the triples

$$\begin{aligned} T_1 &= (31, 1, 0) \\ T_2 &= (45, 0, -1) \end{aligned}$$

Computing the triples

The computation is shown in the following table:

i	r_i	u_i	v_i	q_i
1	31	1	0	
2	45	0	-1	0
3	31	1	0	1
4	14	-1	-1	2
5	3	3	2	4
6	2	-13	-9	1
7	1	16	11	2
8	0	-45	-31	

Extracting the solution

From $T_7 = (1, 16, 11)$, we obtain the solution $x = 16$ and $y = 11$ to the equation

$$1 = 31x - 45y$$

We can check this by substituting for x and y:

$$31 \cdot 16 + (-45) \cdot 11 = 496 - 495 = 1.$$

The solution to

$$31x - 45y = 3 \tag{10}$$

is then $x = 3 \cdot 16 = 48$ and $y = 3 \cdot 11 = 33$.

Generating RSA Encryption and Decryption Exponents

Recall RSA exponent requirement

Recall that the RSA encryption and decryption exponents must be chosen so that

$$ed \equiv 1 \pmod{\phi(n)}, \tag{11}$$

that is, d is e^{-1} in $\mathbf{Z}^*_{\phi(n)}$.

How does Alice choose e and d to satisfy (11)?

- Choose a random integer $e \in \mathbf{Z}^*_{\phi(n)}$.
- Solve (11) for d.

We know now how to solve (11), but how does Alice sample at random from $\mathbf{Z}^*_{\phi(n)}$?

Sampling from $\mathbf{Z}_n^*$

If $\mathbf{Z}_{\phi(n)}^*$ is large enough, Alice can just choose random elements from $\mathbf{Z}_{\phi(n)}$ until she encounters one that also lies in $\mathbf{Z}_{\phi(n)}^*$.

A candidate element e lies in $\mathbf{Z}_{\phi(n)}^*$ iff $\gcd(e, \phi(n)) = 1$, which can be computed efficiently using the Euclidean algorithm.[2]

[2] $\phi(n)$ itself is easily computed for an RSA modulus $n = pq$ since $\phi(n) = (p-1)(q-1)$ and Alice knows p and q.

How large is large enough?

If $\phi(\phi(n))$ (the size of $\mathbf{Z}^*_{\phi(n)}$) is much smaller than $\phi(n)$ (the size of $\mathbf{Z}_{\phi(n)}$), Alice might have to search for a long time before finding a suitable candidate for e.

In general, $\mathbf{Z}^*_m$ can be considerably smaller than m.
Example:

$$m = |\mathbf{Z}_m| = 2 \cdot 3 \cdot 5 \cdot 7 = 210$$
$$\phi(m) = |\mathbf{Z}^*_m| = 1 \cdot 2 \cdot 4 \cdot 6 = 48.$$

In this case, the probability that a randomly-chosen element of $\mathbf{Z}_m$ falls in $\mathbf{Z}^*_m$ is only $48/210 = 8/35 = 0.228\ldots$.

A lower bound on $\phi(m)/m$

The following theorem provides a crude lower bound on how small $\mathbf{Z}_m^*$ can be relative to the size of $\mathbf{Z}_m$.

Theorem

For all $m \geq 2$,

$$\frac{|\mathbf{Z}_m^*|}{|\mathbf{Z}_m|} \geq \frac{1}{1 + \lfloor \log_2 m \rfloor}.$$

Proof.

Write $m = \prod_{i=1}^{t} p_i^{e_i}$, where p_i is the i^{th} prime that divides m and $e_i \geq 1$. Then $\phi(m) = \prod_{i=1}^{t}(p_i - 1)p_i^{e_i-1}$, so

$$\frac{|\mathbf{Z}_m^*|}{|\mathbf{Z}_m|} = \frac{\phi(m)}{m} = \frac{\prod_{i=1}^{t}(p_i - 1)p_i^{e_i-1}}{\prod_{i=1}^{t} p_i^{e_i}} = \prod_{i=1}^{t}\left(\frac{p_i - 1}{p_i}\right). \qquad (12)$$

Proof.

$$\frac{|\mathbf{Z}_m^*|}{|\mathbf{Z}_m|} = \frac{\phi(m)}{m} = \frac{\prod_{i=1}^t (p_i - 1)p_i^{e_i-1}}{\prod_{i=1}^t p_i^{e_i}} = \prod_{i=1}^t \left(\frac{p_i - 1}{p_i}\right). \tag{12}$$

To estimate the size of $\prod_{i=1}^t (p_i - 1)/p_i$, note that

$$\left(\frac{p_i - 1}{p_i}\right) \geq \left(\frac{i}{i+1}\right).$$

This follows since $(x-1)/x$ is monotonic increasing in x, and $p_i \geq i+1$. Then

$$\prod_{i=1}^t \left(\frac{p_i - 1}{p_i}\right) \geq \prod_{i=1}^t \left(\frac{i}{i+1}\right) = \frac{1}{2}\cdot\frac{2}{3}\cdot\frac{3}{4}\cdots\frac{t}{t+1} = \frac{1}{t+1}. \tag{13}$$

Proof.

$$\frac{|\mathbf{Z}_m^*|}{|\mathbf{Z}_m|} = \frac{\phi(m)}{m} = \frac{\prod_{i=1}^{t}(p_i - 1)p_i^{e_i - 1}}{\prod_{i=1}^{t} p_i^{e_i}} = \prod_{i=1}^{t}\left(\frac{p_i - 1}{p_i}\right). \tag{12}$$

$$\prod_{i=1}^{t}\left(\frac{p_i - 1}{p_i}\right) \geq \prod_{i=1}^{t}\left(\frac{i}{i+1}\right) = \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} \cdots \frac{t}{t+1} = \frac{1}{t+1}. \tag{13}$$

Clearly $t \leq \lfloor \log_2 m \rfloor$ since $2^t \leq \prod_{i=1}^{t} p_i \leq m$ and t is an integer.

Combining this with equations (12) and (13) gives the desired result.

$$\frac{|\mathbf{Z}_m^*|}{|\mathbf{Z}_m|} \geq \frac{1}{t+1} \geq \frac{1}{1 + \lfloor \log_2 m \rfloor}. \tag{14}$$

Expected difficulty of choosing RSA exponent e

For n a 1024-bit integer, $\phi(n) < n < 2^{1024}$.

Hence, $\log_2(\phi(n)) < 1024$, so $\lfloor \log_2(\phi(n)) \rfloor \leq 1023$.

By the theorem, the fraction of elements in $\mathbf{Z}_{\phi(n)}$ that also lie in $\mathbf{Z}^*_{\phi(n)}$ is at least

$$\frac{1}{1 + \lfloor \log_2 \phi(n) \rfloor} \geq \frac{1}{1024}.$$

Therefore, the expected number of random trials before Alice finds a number in $\mathbf{Z}^*_{\phi(n)}$ is provably at most 1024 and is likely much smaller.