CPSC 467b: Cryptography and Computer Security
Lecture 9

Michael J. Fischer

Department of Computer Science
Yale University

February 8, 2010

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.

Ring of integers mod n

Let $\mathbf{Z}$ denote the set of all integers, positive and negative. Let $\mathbf{Z}_n \subseteq \mathbf{Z}$ contain the non-negative integers less than n, that is,

$$\mathbf{Z}_n = \{0, 1, \ldots, n-1\}.$$

We now define addition, subtraction, and multiplication operations directly on $\mathbf{Z}_n$:

$$\begin{aligned} a \oplus b &= (a+b) \bmod n \\ a \ominus b &= (a-b) \bmod n \\ a \otimes b &= (a \times b) \bmod n \end{aligned} \tag{1}$$

We will sometimes write $+, -, \times$ in place of $\oplus, \ominus, \otimes$, respectively, when it is clear from context that they are to be regarded as operations over $\mathbf{Z}_n$ rather than over $\mathbf{Z}$.

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 $a \geq b \geq 0$:

$$\begin{aligned} \gcd(a, b) &= \gcd(b, a) \qquad (2) \\ \gcd(a, 0) &= a \qquad (3) \\ \gcd(a, b) &= \gcd(a - b, b) \qquad (4) \end{aligned}$$

Identity 2 is obvious from the definition of gcd. Identity 3 follows from the fact that every positive integer divides 0. Identity 4 follows from the basic fact relating divides and addition from lecture 8.

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 (4) 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

$$\gcd(a, b) = \gcd(a \bmod b, b). \tag{5}$$

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 $\text{gcd}(a, b)$. 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, or equivalently, if $\gcd(a, b) = 1$. Let $\mathbf{Z}_n^* \subseteq \mathbf{Z}_n$ contain those integers 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 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.[a] Then

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

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

When p is prime, we have simply $\phi(p) = (p - 1)$, so for the product of two distinct primes, $\phi(pq) = (p - 1)(q - 1)$.

$\otimes$ on $\mathbf{Z}_n$

Recall the operation $\otimes$ is defined on all of $\mathbf{Z}_n$.

Theorem

$\mathbf{Z}_n^*$ is closed under $\otimes$.

In words, if a and b are both in $\mathbf{Z}_n^*$, then $a \otimes b$ is also in $\mathbf{Z}_n^*$.

Proof.

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

$\mathbf{Z}_n^*$ is an Abelean group

An *Abelean group* is a set with an associative and commutative binary operation having left and right inverses and an identity element.

$\mathbf{Z}_n^*$ under $\otimes$ is an Abelean group. This means that it satisfies the following properties:

Associativity	$\otimes$ is an associative binary operation on $\mathbf{Z}_n^*$. In particular, $\mathbf{Z}_n^*$ is closed under $\otimes$.
Identity	1 is an identity element for $\otimes$ in $\mathbf{Z}_n^*$, that is $1 \otimes x = x \otimes 1 = x$ for all $x \in \mathbf{Z}_n^*$.
Inverses	For all $x \in \mathbf{Z}_n^*$, there exists another element $x^{-1} \in \mathbf{Z}_n^*$ such that $x \otimes x^{-1} = x^{-1} \otimes x = 1$.
Commutativity	$\otimes$ is commutative, i.e., $x \otimes y = y \otimes x$ for all $x, y \in \mathbf{Z}_n^*$.

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

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

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

Inverses of 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 $[-12, \dots, 13]$.
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.

More to prove

It is not obvious from what I have said so far that inverses always exist for members of $\mathbf{Z}_n^*$.

The fact that they do will become apparent later when we show how to compute inverses.

Repeated multiplication in a finite group

General property of finite groups: If any element x is repeatedly multiplied by itself, the result is eventually 1.

Example, for $x = 5 \in \mathbf{Z}^*_{26}$: 5, 25, 21, 1, 5, 25, 21, 1,. . .

Let x^k denote the result of multiplying x by itself k times.
The *order of x*, written $\text{ord}(x)$, is the smallest integer $k \geq 1$ for which $x^k = 1$.

Note added after class: The first repeated element must be x. If not, then some $y \neq x$ is the first to repeat. The element immediately preceding each occurrence of y is yx^{-1}. But then yx^{-1} is the first to repeat, a contradiction. Hence, $x = x^{k+1}$ for some $k \geq 1$, so $x^k = x^{k+1}x^{-1} = xx^{-1} = 1$.

Euler's and Fermat's theorem

For any group, ord(x) divides the order (# elements) of the group.

For $\mathbf{Z}_n^*$, we therefore have $\mathrm{ord}(x) \mid \phi(n)$.

Theorem (Euler's theorem)

$x^{\phi(n)} \equiv 1 \pmod{n}$ for all $x \in \mathbf{Z}_n^*$.

As a special case, we have

Theorem (Fermat's theorem)

$x^{(p-1)} \equiv 1 \pmod{p}$ for all x, $1 \leq x \leq p-1$, where p is prime.

An important corollary

Corollary

Let $r \equiv s \pmod{\phi(n)}$. Then $a^r \equiv a^s \pmod{n}$ for all $a \in \mathbf{Z}_n^*$.

Proof.

If $r \equiv s \pmod{\phi(n)}$, then $r = s + u\phi(n)$ for some integer u. Then using Euler's theorem, we have

$$a^r \equiv a^{s+u\phi(n)} \equiv a^s \cdot (a^u)^{\phi(n)} \equiv a^s \cdot 1 \equiv a^s \pmod{n},$$

as desired. □

Application to RSA

Recall the RSA encryption and decryption functions

$$\begin{aligned} E_e(m) &= m^e \bmod n \\ D_d(c) &= c^d \bmod n \end{aligned}$$

where $n = pq$ is the product of two distinct large primes p and q.

This corollary gives a sufficient condition on e and d to ensure that the resulting cryptosystem works. That is, we require that

$$ed \equiv 1 \pmod{\phi(n)}.$$

Then $D_d(E_e(m)) \equiv m^{ed} \equiv m^1 \equiv m \pmod{n}$ for all messages $m \in \mathbf{Z}_n^*$.

Messages not in $\mathbf{Z}_n^*$

What about the case of messages $m \in \mathbf{Z}_n - \mathbf{Z}_n^*$?
There are several answers to this question.

1. Alice doesn't really want to send such messages if she can avoid it.
2. If Alice sends random messages, her probability of choosing a message not in $\mathbf{Z}_n^*$ is very small — only about $2/\sqrt{n}$.
3. RSA does in fact work for all $m \in \mathbf{Z}_n$, even though Euler's theorem fails for $m \notin \mathbf{Z}_n^*$.

Why Alice might want to avoid sending messages not in $\mathbf{Z}_n^*$

If $m \in \mathbf{Z}_n - \mathbf{Z}_n^*$, either $p \mid m$ or $q \mid m$ (but not both because $m < pq$).

If Alice ever sends such a message and Eve is astute enough to compute $\gcd(m, n)$ (which she can easily do), then Eve will succeed in breaking the cryptosystem.

Why?

Why a random message is likely to be in $\mathbf{Z}_n^*$

The number of messages in $\mathbf{Z}_n - \mathbf{Z}_n^*$ is only

$$n - \phi(n) = pq - (p-1)(q-1) = p + q - 1$$

out of a total of $n = pq$ messages altogether.

If p and q are both 512 bits long, then the probability of choosing a bad message is only about $2 \cdot 2^{512}/2^{1024} = 1/2^{511}$.

Such a low-probability event will likely never occur during the lifetime of the universe.

RSA works anyway

For $m \in \mathbf{Z}_n - \mathbf{Z}_n^*$, RSA works anyway, but for different reasons.

For example, if $m = 0$, it is clear that $(0^e)^d \equiv 0 \pmod{n}$, yet Euler's theorem fails since $0^{\phi(n)} \not\equiv 1 \pmod{n}$.

We omit the proof of this curiosity.