CS255: Cryptography and Computer Security	Winter 2000

Basic number theory fact sheet

Part II: Arithmetic modulo composites

Basic stuff

1. We are dealing with integers N on the order of 300 digits long, (1024 bits). Unless otherwise stated, we assume N is the product of two equal size primes, e.g. on the order of 150 digits each (512 bits).

2. For a composite N let $\mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$.
Elements of $\mathbb{Z}_N$ can be added and multiplied modulo N.

3. The inverse of $x \in \mathbb{Z}_N$ is an element $y \in \mathbb{Z}_N$ such that $x \cdot y = 1 \bmod N$.
An element $x \in \mathbb{Z}_N$ has an inverse if and only if x and N are relatively prime. In other words, $\gcd(x, N) = 1$.

4. Elements of $\mathbb{Z}_N$ can be efficiently inverted using Euclid's algorithm. If $\gcd(x, N) = 1$ then using Euclid's algorithm it is possible to efficiently construct two integers $a, b \in \mathbb{Z}$ such that $ax + bN = 1$. Reducing this relation modulo N leads to $ax = 1 \bmod N$. Hence $a = x^{-1} \bmod N$.
Note: this inversion algorithm also works in $\mathbb{Z}_p$ for a prime p and is more efficient than inverting x by computing $x^{p-2} \bmod p$.

5. Denote by $\mathbb{Z}_N^*$ the set of invertible elements in $\mathbb{Z}_N$.

6. We now have an algorithm for solving linear equations: $\quad a \cdot x = b \bmod N$.
Solution: $\quad x = b \cdot a^{-1}$ where a^{-1} is computed using Euclid's algorithm.

7. How many elements are in $\mathbb{Z}_N^*$? We denote by $\varphi(N)$ the number of elements in $\mathbb{Z}_N^*$.
We already know that $\varphi(p) = p - 1$ for a prime p.

8. One can show that if $N = p_1^{e_1} \cdots p_m^{e_m}$ then $\varphi(N) = N \cdot \prod_{i=1}^{m} \left(1 - \frac{1}{p_i}\right)$.
In particular, when $N = pq$ we have that $\varphi(N) = (p-1)(q-1) = N - p - q + 1$.
Example: $\varphi(15) = |\{1, 2, 4, 7, 8, 11, 13, 14\}| = 8 = 2 * 4$.

9. Euler's theorem: for any $a \in \mathbb{Z}_N^*$ we have that $a^{\varphi(N)} = 1 \bmod N$.
Note: Euler's theorem implies that for a prime p we have $a^{\varphi(p)} = a^{p-1} = 1 \bmod p$ for all $a \in \mathbb{Z}_p^*$. Hence, Euler's theorem is a generalization of Fermat's theorem.

Structure of $\mathbb{Z}_N$

1. The Chinese Remainder Theorem (CRT): Let p, q be relatively primes integers and let $N = pq$. Given $r_1 \in \mathbb{Z}_p$ and $r_2 \in \mathbb{Z}_q$ there *exists a unique* element $s \in \mathbb{Z}_N$ such that $s = r_1 \bmod p$ and $s = r_2 \bmod q$. Furthermore, s can be computed efficiently.

2. The CRT shows that each element $s \in \mathbb{Z}_N$ can be viewed as a pair (s_1, s_2) where $s_1 = s \bmod p$ and $s_2 = s \bmod q$. The uniqueness guarantee shows that each pair $(s_1, s_2) \in \mathbb{Z}_p \times \mathbb{Z}_q$ corresponds to one element of $\mathbb{Z}_N$. For example, the pair $(1, 1)$ corresponds to $1 \in \mathbb{Z}_N$.

3. Note that by the CRT if $x = y \bmod p$ and $x = y \bmod q$ then $x = y \bmod N$.

4. An element $s \in \mathbb{Z}_N$ is invertible if and only if $s \bmod p$ in invertible in $\mathbb{Z}_p$ and $s \bmod q$ is invertible in $\mathbb{Z}_q$. Hence, the number of invertible elements in $\mathbb{Z}_N$ is $\varphi(N) = (p-1)(q-1)$.

5. An element $s \in \mathbb{Z}_N^*$ is a Q.R. if and only if $s \bmod p$ is a Q.R. in $\mathbb{Z}_p$ and $s \bmod q$ is a Q.R. in $\mathbb{Z}_q$. Hence, the number of Q.R. in $\mathbb{Z}_N$ is $\frac{p-1}{2} \cdot \frac{q-1}{2} = \frac{\varphi(N)}{4}$.

6. Jacobi symbol: for $x \in Z_N$ define $\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right) \cdot \left(\frac{x}{q}\right)$.
   As it turns out, there is en efficient algorithm to compute the Jacobi symbol of $x \in \mathbb{Z}_N$ without knowing the factorization of N.

7. Consider the RSA function $f(x) = x^e \bmod N$. When e is odd we have that:

$$\left(\frac{x^e}{N}\right) = \left(\frac{x^e}{p}\right) \cdot \left(\frac{x^e}{q}\right) = \left(\frac{x}{p}\right) \cdot \left(\frac{x}{q}\right) = \left(\frac{x}{N}\right)$$

   Hence, given an RSA ciphertext $C = x^e \bmod N$ the Jacobi symbol of C reveals the Jacobi symbol of x.

Computing in $\mathbb{Z}_N$

1. Since N is a huge prime (e.g. 1024 bits long) it cannot be stored in a single register.

2. Elements of $\mathbb{Z}_N$ are stored in buckets where each bucket is 32 or 64 bits long depending on the processor's register size.

3. Adding two elements $x, y \in \mathbb{Z}_N$ can be done in linear time in the *length* of N.

4. Multiplying two elements $x, y \in \mathbb{Z}_N$ can be done in quadratic time in the *length* of N. For an n bit integer N faster multiplication algorithms work in time $O(n^{1.7})$ (rather than $O(n^2)$).

5. Inverting an element $x \in \mathbb{Z}_N$ can be done in quadratic time in the length of N using Euclid's algorithm.

6. Using the repeated squaring algorithm, $x^r \bmod N$ can be computed in time $(\log_2 r)O(n^2)$ where N is n bits long. Note, the algorithm takes linear time in the length of r.

7. Efficient exponentiation modulo $N = pq$ when the factorization of N is known: to compute $a = x^s \bmod N$ one does the following:

   (a) Compute $a_1 = x^s \bmod p$ and $a_2 = x^s \bmod q$. Note that it suffices to compute $a_1 = x^{s \bmod p-1} \bmod p$ and $a_2 = x^{s \bmod q-1} \bmod q$.

   (b) Use the Chinese Remainder Theorem to construct $a \in \mathbb{Z}_N$ such that $a = a_1 \bmod p$ and $a = a_2 \bmod q$. Then $a = x^s \bmod N$ since this relation holds modulo p and modulo q.

   Since p and q are half the size of N arithmetic modulo p and q is four times as fast (recall, multiplication takes quadratic time). Furthermore, $s \bmod p-1$ and $s \bmod q-1$ are each roughly half that size of s (we are assuming s is as large as N). Hence, computing of $a_1 = x^{s \bmod p-1} \bmod p$ is eight times faster than computing $a = x^s \bmod N$. Since we repeat this step twice, once for p and once for q, exponentiation using CRT is four times faster overall.

Summary

Let N be a 1024 bit integer which is a product of two 512 bit primes. Easy problems in $\mathbb{Z}_N$:

1. Generating a random element. Adding and multiplying elements.
2. Computing $g^r \bmod N$ is easy even if r is very large.
3. Inverting an element. Solving linear systems.

Problems that are believed to be hard if the factorization of N is unknown, but become easy if the factorization of N is known:

1. Finding the prime factors of N.
2. Testing if an element is a QR in $\mathbb{Z}_N$.
3. Computing the square root of a QR in $\mathbb{Z}_N$. This is provably as hard as factoring N. When the factorization of $N = pq$ is known one computes the square root of $x \in \mathbb{Z}_N^*$ by first computing the square root in $\mathbb{Z}_p$ of $x \bmod p$ and the square root in $\mathbb{Z}_q$ of $x \bmod q$ and then using the CRT to obtain the square root of x in $\mathbb{Z}_N$.
4. Computing e'th roots modulo N when $\gcd(e, \varphi(N)) = 1$.
5. More generally, solving polynomial equations of degree d. This is believed to be hard when the factorization of N is unknown, but can be done in polynomial time in d when the factorization is given. When the factorization of N is given one solves the polynomial equation by first solving it modulo p and q and then using the CRT to obtain the roots in $\mathbb{Z}_N$.

Problems that are believed to be hard in $\mathbb{Z}_N$:

1. Let g be a generator of $\mathbb{Z}_N^*$. Given $x \in \mathbb{Z}_N^*$ find an r such that $x = g^r \bmod N$. This is known as the *discrete log problem.*

2. Let g be a generator of $\mathbb{Z}_N^*$. Given $x, y \in \mathbb{Z}_N^*$ where $x = g^{r_1}$ and $y = g^{r_2}$. Find $z = g^{r_1 r_2}$. This is known as the *Diffie-Hellman problem.*

One-way functions

Recall: a function $f : \{0,1\}^n \to \{0,1\}^m$ is a (t, ϵ) one-way function if

1. There is an efficient algorithm that for any $x \in \{0,1\}^n$ outputs $f(x)$.

2. The function is hard to invert. More precisely, for any algorithm $\mathcal{A}$ whose running time is at most t we have

$$\Pr_{x \in \{0,1\}^n} \Big[f(\mathcal{A}(f(x))) = f(x) \Big] < \epsilon$$

   In other words, when given $f(x)$ as input algorithm $\mathcal{A}$ is unlikely to output a y such that $f(y) = f(x)$.

Based on block ciphers If $E(M, k)$ is a block cipher secure against a chosen ciphertext attack then $f(k) = E(0, k)$ is a one way function. Such general one-way functions can be used for symmetric encryption, but cannot be used for efficient key-exchange.

Discrete log Fix a prime p and an element $g \in \mathbb{Z}_p^*$ of "large" order.
Define $f_{Dlog}(x) = g^x \bmod p$.
Main property: *linear*: Given $a \in \mathbb{Z}$ and $f(x)$, $f(y)$ one can easily compute $f(a \cdot x)$ and $f(x + y)$.
The one-wayness of this function is essential for the security of the Diffie-Hellman protocol and ElGamal public key system.

RSA Let $N = pq$ be a product of two large primes. Let e be an integer relatively prime to $\varphi(N)$. Define $f_{RSA}(x) = x^e \bmod N$.
Main property: *trapdoor*. Given the factorization of N the function can be inverted efficiently.
The one wayness of this function is essential to the security of the RSA public key system.

Rabin Let $N = pq$ be a product of two large primes. Define $f_{Rabin}(x) = x^2 \bmod N$. This function is one-way if there is no efficient algorithm to factor integers of the form $N = pq$. As in the case of RSA, the factorization of N enables efficient inversion. The one wayness of this function is essential to the security of Rabin's signature scheme.