Midterm Study Guide

The midterm will be an 80-minute long exam consisting of 13 problems, 10 of which will be short-answer, requiring little or no justification.

In what follows below I delineate exactly what I expect you to know for the exam. You do not need to know any proofs except where specifically mentioned. More emphasis will be placed on knowing how algorithms work than the sometimes-deep mathematics behind them.

Number Theoretical Foundations

Basic number theory and modular arithmetic (1.2–1.5 and 2.8)

• You should know the definition of divisor, the divisibility relation |, greatest common divisor (GCD), and what division with remainder means.

• You should be able to prove basic results about divisors, divisibility, and GCDs, including but not limited to:

  – Proposition 1.4; and
  – If $a, b$ are integers $\geq 0$ and $r$ is the remainder of $a$ divided by $b$, then $\gcd(a, b) = \gcd(a, r)$.

• You should know that the Euclidean algorithm is based on the second fact above, though you will not be asked to prove that the algorithm works.

• Given $a, b, n$, you should know under what conditions $ax + by = n$ can be solved (iff $\gcd(a, b) \mid n$), how to find one solution (extended Euclidean algorithm), and how to find all solutions (Exercise 1.11d) (without proofs).

• You should know what prime numbers are, their important properties (Props 1.20–22, for example), and that every positive integer factors uniquely as a product of prime numbers. You should know what $\text{ord}_p(n)$ means.

• You should know the content of the “modular arithmetic cheat sheet” on the course website (without proofs) and be able to apply that knowledge deftly. This includes facts about congruence mod $m$, $\mathbb{Z}/m\mathbb{Z}$ as a ring, arithmetic operations modulo $m$, the units of $\mathbb{Z}/m\mathbb{Z}$ and how to find inverses, the definition of the $\phi$ function, the finite fields $\mathbb{F}_p$, exponentiation in $\mathbb{Z}/m\mathbb{Z}$, the definition of order mod $m$, Fermat’s little theorem, the Fermat–Euler theorem, and primitive roots in $\mathbb{F}_p$.

• You should know that $a^k \equiv 1 \mod m$ iff the order of $a$ (denoted $|a|_m$) divides $k$. 
• You should be able to find simultaneous solutions to systems of congruences mod $m_1, \ldots, m_t$ given that $\gcd(m_i, m_j) = 1$ for all $i \neq j$ using the Chinese remainder theorem.

You should know also that if $\gcd(M, N) = 1$, the CRT sets up a bijection $\mathbb{Z}/MN\mathbb{Z} \rightarrow \mathbb{Z}/M\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$ given by

$$(a \mod MN) \mapsto (a \mod M, a \mod N)$$

and that this restricts to a bijection on units: $(\mathbb{Z}/MN\mathbb{Z})^* \rightarrow (\mathbb{Z}/M\mathbb{Z})^* \times (\mathbb{Z}/N\mathbb{Z})^*$.

• You should be able to compute $\phi(n)$ quickly given the prime factorization of $n$. Using the formulas $\phi(1) = 1$, $\phi(p^e) = p^e - p^{e-1} = p^{e-1}(p - 1)$ (if $p^e$ is a prime power) and $\phi(mn) = \phi(m)\phi(n)$ when $\gcd(m, n) = 1$. (You should know that this last rule is based on the CRT bijection for units, above.)

Discrete logarithms and roots in modular arithmetic (2.2, 3.1, 3.9)

• Given a prime $p$ and a primitive root $g$, you should know the definition of the discrete logarithm $\log_g : \mathbb{F}_p^* \rightarrow \mathbb{Z}/(p - 1)\mathbb{Z}$, and you should be able to explain why its codomain is $\mathbb{Z}/(p - 1)\mathbb{Z}$.

• More generally, you should know that if $a^k \equiv b \mod p$ (with $a$ not necessarily a primitive root), then $k = \log_a(b)$ is “determined up to” congruence modulo the order of $a$: In particular, $a^i \equiv a^j \mod p$ iff $i \equiv j \mod |a|^p$.

• You should know the basic properties of the discrete logarithm (the ones analogous to the “real” logarithm) as in Exercise 2.3.

• You should know that if $a^e \equiv b \mod m$, then $a \equiv b^d \mod m$ where $ed \equiv 1 \mod \phi(m)$.

• You should know under what conditions a quadratic congruence $ax^2 + bx + c \equiv 0 \mod p$ can be solved (when $p$ is prime, $p > 2$, $a \neq 0 \mod p$), namely that $b^2 - 4ac$ must be a quadratic residue modulo $p$.

• You should be able to compute $\left( \frac{a}{p} \right)$ quickly (using the various rules in 3.9 and in particular 3.61) in order to determine whether $x^2 \equiv a \mod p$ has a solution or not.

Algorithms for Number Theory

You should know how the algorithms below work, some ideas behind why they work, and their runtimes. You may be asked to implement them in simple cases:

• The Euclidean algorithm (GCDs and modular inversion): Reduces computing $\gcd(a, b)$ recursively to computing $\gcd(a, a \% b)$. Requires $O(\log N)$ divisions, so $O((\log N)^2)$ time.

• Fast powering (modular exponentiation and inversion): Reduces computing $a^a \mod m$ to computing the binary expansion of $n$ and repeatedly squaring $a$ modulo $m$. Requires $O(\log N)$ time (all operations are multiplication). Can be used to invert mod $p$ using $a^{b-2} \equiv a^{-1} \mod p$. 

2
• **Baby step–giant step** (discrete log): Solves $g^x \equiv h \mod m$ by making two lists (mod $m$), \( \{g^j\}_{j=0}^n \) and \( \{hg^{-jn}\}_{j=0}^n \) where $n = \lceil \sqrt{N} \rceil + 1$ and $N$ is the order of $g \mod m$. A collision is guaranteed (by looking at the division of $N$ by $n$ with remainder), and if $g^i \equiv hg^{-jn} \mod m$, then $x = i + jn$ is a solution. This requires $O(\sqrt{N} \log N)$ time, and so it is exponential in input size.

• **Pohlig–Hellman** (discrete log): Speeds up a DLP-solving algorithm by applying the Chinese remainder theorem to congruences modulo $N = q_1^{e_1} \cdots q_t^{e_t}$ in two stages:
  - The first stage, which we covered, reduces the runtime to $O(\sqrt{Q} \log Q + \log N)$ where $Q$ is the largest prime power dividing $N$. We covered this, so you are responsible for knowing how it works.
  - The second stage, which we covered, reduces the runtime to $O(\sqrt{q} \log q + \log N)$ where $q$ is the largest prime dividing $N$. We did not cover this second improvement—you only need to know its runtime and the security implications ($p - 1$ should have a large prime factor, ideally $\frac{p-1}{2}$ should be prime).

• **Miller–Rabin** (primality testing): Let $n > 1$ be the number to test and let $a > 1$ be relatively prime to $n$ (otherwise either $n \mid a$ or $\text{gcd}(a, n)$ is a nontrivial proper factor of $n$ and so $n$ is composite). Let $n - 1 = 2^k q$ where $k \geq 0$ and $q$ is odd. If $a^q \not\equiv 1 \mod n$ and $a^{2^k q} \not\equiv -1 \mod n$ for all $i = 0, \ldots, k - 1$, then $n$ is composite.
  - An $a$ that satisfies the above is a *Miller–Rabin witness* for (the compositeness of) $n$.
  - If $n$ is composite, $\geq 75\%$ of all numbers ($a$ with $1 < a < n$) are MR witnesses for it.
  - If $a$ is *not* a MR witness for $n$ for “many” values of $a$, then $n$ is a *probable prime*.
  - If the Riemann Hypothesis is true, then there is a MR witness $a \leq 2(\log n)^2$.

• **Agrawal–Kayal–Saxena/AKS** (primality testing): You need only know that this is a *deterministic* primality test that runs in polynomial time. This is in contrast to the MR primality test, which is either probabilistic (it could potentially give false positives) or conditional (requires the truth of the Riemann hypothesis, which is unproven). MR is faster in practice.

• **Pollard** $p - 1$ (prime factorization): Factors $N = pq$ quickly if either $p - 1$ or $q - 1$ has only small prime factors. In particular, if $p - 1 \mid n!$ but $q - 1 \nmid n!$, then $p$ can (usually) be recovered as $p = \text{gcd}(N, 2^n - 1)$ (with the factorial and exponential computed mod $N$ of course). The security implication for factorization-based cryptosystems is that $p - 1$ and $q - 1$ should each have at least one relatively large prime factor.

• **Congruences of squares** (prime factorization): Algorithms that factor numbers into primes by finding $X, Y \mod N$ such that $X^2 \equiv Y^2 \mod N$ but $X \not\equiv Y \mod N$. Then $\text{gcd}(X - Y, N)$ is a nontrivial proper divisor of $N$.

Such algorithms proceed in a few steps: (0) Choosing a factor base / smoothness bound and a starting point; (1) *(Relation gathering)* Computing $(k^2)\%N$ for some values of $k$ and recording those that only have factors in the factor base; (2) *(Elimination)* Using linear algebra over $\mathbb{F}_2$ to use the relations to generate a (nontrivial) congruence of squares mod
— guaranteed to work once you have more relations than primes in the factor base; (3)
Finally, take the GCD of \(X - Y\) and \(N\).

We discussed three implementations, in sharply decreasing detail:

- **Dixon’s method**: In step (0), let \(B = e^{\sqrt{\log N \log \log N} / \sqrt{2}}\) and let the factor base consist of all primes \(p \leq B\). In step (1) Choose \(k \mod N\) at random. The expected number of \(k\) required to factor \(N\) is \(e^{(\sqrt{2} + o(1))\sqrt{\log N \log \log N}}\).

- **Quadratic sieve**: In step (1), start at \(k = \lfloor \sqrt{N} \rfloor + 1\) and work your way up. This is advantageous because \((k^2) \mod N = k^2 - N\) is small (and therefore more likely to have small prime factors in the chosen factor base) when \(k\) is only a little bigger than \(\sqrt{N}\). We now expect to have to look at \(e^{(1 + o(1))\sqrt{\log N \log \log N}}\) values of \(k\) to factor \(N\).

- **Number field sieve**: With some fairly advanced theory, we can reduce the runtime of this kind of algorithm further, to \(e^{(1 + o(1))\sqrt{\log N \log \log N}}\). The NFS is the fastest general-purpose algorithm for factoring integers, but its advantage over the QS is not seen until fairly large values of \(N\).

In summary, the expected runtimes of the algorithms above are \(L_N[\frac{1}{2}, \sqrt{2}]\), \(L_N[\frac{1}{2}, 1]\), and \(L_N[\frac{1}{3}, 1]\), where
\[
L_N[\alpha, c] = e^{(c + o(1))\log N^{\alpha} \log \log N^{1-\alpha}}
\]
and \(o(1)\) represents a function that \(\to 0\) as \(N \to \infty\). (You do not have to know the \(L\)-notation here, but it is obviously more compact than the alternative!)

**Cryptographic Protocols**

You are expected to know how the following cryptosystems work:

- **Ciphers**: One-time pads, affine ciphers (1.7)
- **Diffie–Hellman key exchange** (2.3)
  You should know that if one can solve the DLP quickly, one can solve the DHP quickly (but it is unknown if the converse is true or false).
- **Elgamal public-key cryptosystem** (2.4)
  You should know that breaking Elgamal is equivalent to solving a DLP.
- **RSA public-key cryptosystem** (3.2)
  You should know that factorizing \(N = pq\) breaks RSA (and why/how), and that knowing either a decryption exponent or \(\phi(N)\) can also allow you to factor \(N\).

We have not gone over attacks on these systems in great detail (e.g. we have not yet covered material like 3.3), but there are some basic ones you should be able to figure out without much trouble on the exam (B3 on the practice exam), and you should know some basic security concerns that arise when choosing moduli for DHKE/Elgamal and RSA.