Problem Set 4

Due in class on Thursday, March 24, 2005.

Problem 11: Primitive roots

Does the modulus $n = pq$ in RSA have primitive roots? Explain why or why not.

Problem 12: Quadratic Residues

Problem 6.13 in the textbook:

> "Let $n = pq$ with p and q being distinct primes. Under what condition $-1 \in \mathrm{QR}_n$? Under what condition $\left(\frac{-1}{n}\right) = -1$?"

Problem 13: Generators over $\mathbf{Z}_p^*$

Prove or disprove the following: Let g, h be generators for $\mathbf{Z}_p^*$, where p is an odd prime. Suppose $x \equiv g^{2u} \equiv h^{2v} \pmod{p}$. Then $g^u \equiv h^v \pmod{p}$.

Problem 14: DSA Signature Scheme Implementation

In this problem, you will implement the key generation, signature, and verification functions of the DSA Signature Scheme. (See lecture notes week 8, §6.) In particular, you should write the following three programs that take the indicated command line arguments:

`dsa_key` *<keyfile>*

This program generates random DSA parameters p, q, g, x. The primes p and q should be exactly 1024 bits long and 160 bits long, respectively. Next it computes a. It then creates two files named `<keyfile>.pub` and `<keyfile>.prv`, where `<keyfile>` is the command line argument. It writes p, q, g, a to `<keyfile>.pub`, in decimal, one number per line. It writes p, q, g, x to `<keyfile>.prv`, also one decimal number per line. The lines should contain only digits (no labels) so that the key file may be easily read back by another program.

`dsa_sign` *<keyfile> <msgfile>*

Reads p, q, g, x from the private key file, `<keyfile>.prv`. Reads a decimal integer $m \in Z_p$ (the message) from `<msgfile>`. Signs m using the DSA signature scheme to produce a signature $s = (b, c)$, which it writes to `<msgifle>.sig` as two decimal numbers, one per line.

`dsa_verify` *<keyfile> <msgfile>*

Reads p, q, g, a from the public key file, `<keyfile>.pub`. Reads m from `<msgfile>`. Reads the signature $s = (b, c)$ from `<msgifle>.sig`. Checks the signature of m using DSA signature verification and writes "good" or "bad" to standard output accordingly.

(continued on next page)

You may use either the GMP or ln3 bignum package, as well as any of the functions that they provide.

The hardest part of this assignment is probably the generation of large random primes of the desired length. The GMP functions `mpz_random()` and `mpz_random2()` are obsolete and should not be used. Use `mpz_urandomb()` or `mpz_urandomn()` instead. In ln3, you might use either `Random()` or `RandomPrime()`, but note that the desired size arguments to both of them is in terms of decimal digits rather than bits. If you'd prefer, you can call the standard Unix random number generator `random()` repeatedly and shift the bits in to form a large number. Note that `random()` returns 31 bits, not 32.

Of course, code for DSA signatures is undoubtedly available from various places on the web. You're welcome to read it but not to copy it. In any case, I expect you to use the routines in the GMP or ln3 packages rather than solve the problem in some other way. Start early on this assignment, and feel free to ask me for help if there are things you don't understand. This is not a difficult assignment, but whenever you program, there are bound to be hidden pitfalls.