Public Key Cryptography

- Every participant has a key pair & publishes their **public** key.
- Encryption: $A \rightarrow B$, $c = E_{b\_pub}(m)$. Attacker may know m, c, b_pub.
- Authentication: $A \rightarrow B$, $C = E_{a\_priv}(M)$. B can verify with $E_{Apub}(C)$. No confidentiality.
- Can combine authentication with encryption. Schneier recommends sign, then encrypt.
- Multiplying two n bit numbers is $O(n^2)$ ops [Knu98, Section 4.3.1, Algorithm M].
- Exponentiating an n bit number mod (n bit #) is approx $O(n^3)$ bit operations. For a 1024 bit #, this is $\approx 1G$ bitops. On a 1ns instruction issue machine, this is $\approx 1s$!

RSA

By definition $\phi(n)$ is the number of integers $0 < x < n$ that are relatively prime to n. Consider

$$n = p \times q$$

where p and q are distinct primes. Then

$$\phi(n) = (n-1) - (p-1) - (q-1) = (p-1) \times (q-1)$$

1. Choose large primes p and q differing by a few digits. Say one of 75 digits, the other of 100 digits. Both $(p-1)$ and $(q-1)$ should contain a large prime factor.
2. Compute $n = p \times q$. No one can factor n easily *la*.
3. Choose e to be, say 65537.
4. Compute $d \equiv e^{-1} \mod \phi(n)$.
5. Public key $= (e, n)$.
6. Private key $= (d, n)$. Infeasible to get d given (e, n).
7. For a given message m, its encryption is $c = m^e \mod n$. And to decrypt a cipher text c, compute $m = c^d \mod n$.
8. $m^{e^d} \equiv m^{ed} \equiv m^1 \bmod \phi(n) \equiv m$.

An example of RSA

Let $p = 577487293141428113323$ and $q = 5295757044745316310341$. Then,

$$n = 305823240090462151745038276856407276791143$$

$\phi(n) = 305823240090462151739684771082347817669480.$

Choose $e = 65537$, then $d = e^{-1} \mod \phi(n)$

$$= 599448455407186291903503451382248205713 13$$

Encode a message "NUS" as its binary encoding (for example) to get $0x4E5553 = 5133651$.

To encrypt, find

$5133651^{65537} \mod 305823240090462151745038276856407276791143$
$= 217657393729141588774828799917624500652607$

To decrypt, compute c^d to get the original message.

$217657393729141588774828799917624500652607^{599448455407186291903503451382248205713}$
$=$ run time error in bc
$= 5133651$

Breaking RSA

- Brute force. Try all possible values of d. Given an (m, c) pair, find a d such that $c^d = m$. From this get $\phi(n)$ [Wel01, Section 16.2] and verify that this $\phi(n)$ yields the correct factors of n.
- Mathematical attacks.
  - Factor n.
  - Find $\phi(n)$. But knowing $\phi(n)$ is equivalent to factoring n. Because $n = pq,\ \phi(n) = n - (p + q) + 1$ and we have

$$\boxed{\begin{array}{rcl} p + q & = & n + 1 - \phi(n) \\ p - q & = & \sqrt{(p+q)^2 - 4n} \end{array}}$$

    This gives equations for $p + q$ and $p - q$.
- Timing attacks.
- Do we need factorization to solve the RSA problem which is finding the e^{th} root modulo n [MvOV96, Section 3.3]?

Show [Sta99, Fig. 6.9] on MIPS years needed to factor large n.

Miller-Rabin Primality Test

Thm: If p is an odd prime, then $x^2 \equiv 1 \mod p$ has only two solutions, namely $x = 1$ and $x = -1$.

Proof: $x^2 \equiv 1 \mod p$ means that $p \mid (x^2 - 1)$, or $p \mid (x-1)(x+1)$. Because p is prime, it divides either $(x-1)$ or $(x+1)$. It cannot divide both because then it'd divide their difference which is $(x+1)-(x-1) = 2$.

Example: $5^2 \mod 6 = 1$ because $5^2 - 1 = \frac{(5-1)(5+1)}{2 \times 3}$.

Miller-Rabin primality test

By Fermat's theorem, $x^{p-1} \equiv 1 \mod p$ if p is prime. So to test a number n for primality, try Fermat's for $x = 2, 3, 4$. Now, let $n - 1 = 2^e \cdot y$. Find x^y. We ultimately want to find $x^{y^{2^e}}$. Repeatedly square x^y but make sure that you never have $z^2 = 1$ when $z \neq \pm 1$.

Discrete Logs

Let $Z_p = \{0, 1, \ldots, p-1\}$, p is prime and

$$Z_p^* = \{1, 2, \ldots, p-1\}$$

For $0 < g < p$ let's study the sequence

$$g^1, g^2, g^3, \ldots$$

We know that $g^{p-1} \equiv 1 \pmod{p}$. Show [Sta99, Table 7.6].

The sequence $g^1, g^2, g^3, \ldots$ ends in 1. If the sequence ends in 1, it clearly repeats itself after that. If it does not, let $g^m = g^x$. Then, since g^{-1} exists, $g^{x-m} \equiv 1 \bmod m$, which is a contradiction. Or looked differently, $g^m(g^{x-m} - 1) \equiv 0$ which means that $p \mid g^{x-m} - 1$, or that $g^{x-m} \equiv 1 \bmod m$.

Def: g is a primitive root of n if ord$(g) = \phi(n)$. Not all integers have primitive roots. Integers with primitive roots are of the form: $2, 4, p^\alpha, 2p^\alpha$, p odd prime.

Thm: Z_p^* is a cyclic group. Not every element of Z_p^* is a generator. For e.g., $<2>$ mod $7 = \{1, 2, 4\}$.

Logarithms are the inverse of exponentiation.

Reals	$\mathbf{Z_p^*}$
$\log_x 1 = 0$	$\log_g 1 \pmod{p} \equiv 0$
$\log_x x = 1$	$\log_g g \pmod{p} \equiv 1$
$\log_x(yz) = \log_x y + \log_x z$	$\log_g(yz) \pmod{p} \equiv \log_g y + \log_g z \pmod{\phi(p)}$
$\log_x(y^r) = r \log_x y$	$\log_g(y^r) \pmod{p} \equiv r \log_g y \pmod{\phi(p)}$

Diffie-Hellman Key Exchange

prime p, generator α.

A $\rightarrow$ B: α^a
B $\rightarrow$ A: α^b
From this both compute shared secret α^{ab}.

DH: Given α^a, α^b, compute α^{ab}. Certainly no harder than Dlog. Does Dlog hard $\Rightarrow$ DH "secure"? Open problem. (Strong evidence).

Dlog is randomly self reducible. Susceptible to *person-in-the-middle* attack.

Diffie-Hellman in practice: $p = 1024$ bit prime, $g \in Z_p^*$, an element of order q where q is a prime such that $q \mid (p-1)$ and $q \approx 2^{160}$ (160 bits).

Now $a \in \{0, 1, \ldots, q-1\}$ and $b \in \{0, 1, \ldots, q-1\}$. Since q is 160 bits, g^a mod p only needs 160 multiplies rather than 1024. A seven fold improvement!

CRT

See [Knu98, Section 4.3.2].

Alternative for doing arithmetic on large numbers. Have several moduli $m_1, m_2, \ldots, m_r$ relatively prime in pairs and work on residues u mod m_i instead of with u.

Regard $(u_1, u_2, \ldots, u_r)$ as a new type of internal representation for u.

Disadvantage: Can't test for $>$, overflow, do division.

Advantage: Parallelizes multiplication.

$$(u_1, u_2, \ldots, u_r)+(v_1, v_2, \ldots, v_r) = ((u_1+v_1) \text{ mod } m_1, \ldots, (u_r+v_r) \text{ mod } m_r)$$

$$(u_1, u_2, \ldots, u_r)-(v_1, v_2, \ldots, v_r) = ((u_1-v_1) \text{ mod } m_1, \ldots, (u_r-v_r) \text{ mod } m_r)$$

$$(u_1, u_2, \ldots, u_r)\times(v_1, v_2, \ldots, v_r) = ((u_1\times v_1) \text{ mod } m_1, \ldots, (u_r\times v_r) \text{ mod } m_r)$$

You can see the above because $uv \mod x = (u \mod x)(v \mod x)$.

Proof: Let $m = m_1 m_2 \cdots m_r$ and let $u_1, u_2, \ldots, u_r$ be integers. Then there is exactly one integer u such that $0 \leq u < m$ and $u \equiv u_j \pmod{m_j}$ for $1 \leq j \leq r$.

Let $M_k = m/m_k$. Then GCD$(M_k, m_k) = 1$. So M_k^{-1} mod m_k exists. Let this be y_k. Then

$$u = u_1 M_1 y_1 + u_2 M_2 y_2 + \cdots + u_r M_r y_r$$

is the solution of the simultaneous congruences.

CRT Example

Let $m_1 = 9, m_2 = 10, m_3 = 11$. Then $m = 990$. Suppose you wanted to find 889^{899} mod 990.

- Find the representation of 889 in the new system $= (7, 9, 9)$.
- Now $889^{899} = (7, 9, 9)^{899} = (7^{899} \mod 9, 9^{899} \mod 10, 9^{899} \mod 11)$.
- That is $= (4, 9, 5)$.
- Convert this back to the integer $= 49$.

References

[Knu98] Donald E. Knuth. *The Art of Computer Programming Volume 2 — Seminumerical Algorithms*. Addison Wesley, 3rd edition, 1998.

[MvOV96] Alfred Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Sta99] William Stallings. *Cryptography and Network Security*. Prentice-Hall Inc., 2nd edition, 1999.

[Wel01] Michael Welschenbach. *Cryptography in C and C++*. Springer Verlag, 2001. Translated by David Kramer.