RSA Algorithm

Sources:

- *Applied Cryptography* by B. Schneier
- *Twenty Years of Attacks on the RSA Cryptosystem* by D. Boneh, *Notices of the AMS*, February 1999
- *RSA Security WebFAQ*, www.rsasecurity.com/company/webfaq
- Cryptography FAQ, www.x5.net/faqs/crypto
- *Modern Cryptography* by W. Mao

RSA is the easiest public-key algorithm to understand and implement.

It is also the most popular.

It is named after its inventors: Ron Rivest, Adi Shamir, and Leonard Adleman (1977).

RSA is widely used:

- Public Key Infrastructure (PKI)
- SSL/TLS certificates and key exchange
- Secure email

Its security comes from the *difficulty of factoring large numbers*.

The public and private keys are functions of a pair of large prime numbers (e.g., 1024 bits).

Recovering the plaintext from the public key and the ciphertext is thought to be equivalent to factoring the product of the two prime numbers.

To generate a key pair, choose two large random prime numbers, p and q (e.g., 512 bits).

For maximum security, p and q should be of equal length.

Compute the product

$$n = pq$$

Then randomly choose the encryption key e such that e and $(p - 1)(q - 1)$ are relatively prime.

Use the extended Euclidean algorithm to compute the decryption key d, such that

$$ed \equiv 1 \bmod (p - 1)(q - 1)$$

That is,

$$d = e^{-1} \bmod (p - 1)(q - 1)$$

Note that d and n are also relatively prime.

The pair (n, e) is the public key.

The number d is the private key.

To encrypt a message m, divide it into numerical blocks smaller than and relatively prime to n.

The encrypted message c will be made up of message blocks c_i of about the same length.

The encryption formula is

$c_i = m_i^e \bmod n$ – Trapdoor 1-1 function

To decrypt a message, take each encrypted block c_i and compute

$m_i = c_i^d \bmod n$

Since

$c_i^d = (m_i^e)^d = m_i^{k(p-1)(q-1)+1} = m_i\, m_i^{k(p-1)(q-1)} = m_i * 1 = m_i$; all (mod n)

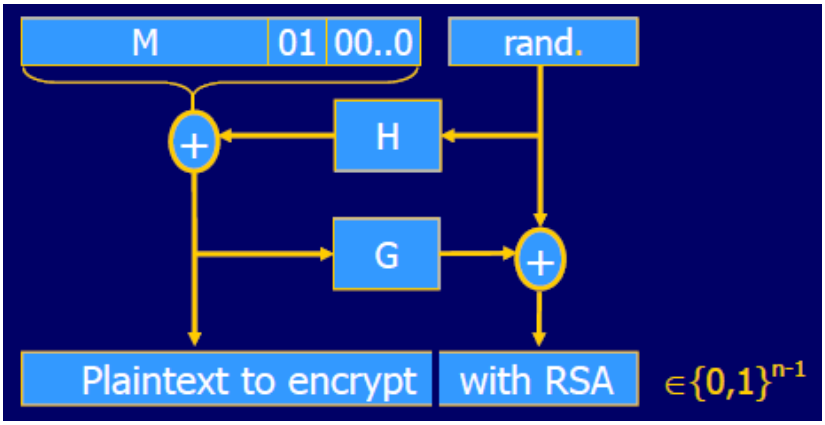
($m_i^{k(p-1)(q-1)} \equiv 1 \bmod n$ follows from Euler's Theorem.)

this recovers the message.

"Textbook" RSA is *insecure*.

Preprocessing of message is necessary.

A *random pad* is added to the message [Boneh]:



[crypto.stanford.edu/~dabo/courses/cs255_winter07/rsa.ppt]

Speed of RSA

An "RSA operation" is essentially a modular exponentiation, which can be performed by a series of modular multiplications.

It is common to choose a small public exponent for the public key.

Entire groups of users can use the same public exponent, each with a different modulus.

This makes encryption faster than decryption and verification faster than signing.

With typical modular exponentiation algorithms,

- Public-key operations take $O(k^2)$ steps
- Private-key operations take $O(k^3)$ steps
- Key generation takes $O(k^4)$ steps

where k is the number of bits in the modulus.

"Fast multiplication" techniques, such as FFT-based methods, require asymptotically fewer steps.

They are not commonly used because:

- They are complex to implement.
- They may actually be slower for typical key sizes.

In hardware, RSA is more than 1000 times slower than DES.

In software, RSA is about 100 times slower than DES.

Factoring Large Integers

If an attacker can factor the modulus n, they can easily determine the decryption exponent d.

Although factoring algorithms have been improving, they still don't compromise the security of RSA, *if it is used properly*.

The current fastest factoring algorithm is the *General Number Field Sieve*.

Its running time on n-bit integers is

$$\exp\left(c + o(1)\, n^{1/3} \log^{2/3} n\right)$$

for some $c < 2$.

Some sparse sets of RSA moduli can be easily factored.

Example: If $p - 1$ is a product of prime factors less than B, then n can be factored in time less than B^3.

As of 2010, the largest number known to be factored with a general purpose factoring algorithm was 768 bits long.

RSA keys are typically 1024 or 2048 bits long.

It is currently recommended that *n* be at least 2048 bits long.

Chosen Ciphertext Attacks Against RSA

Some attacks work against the way RSA is used in cryptographic protocols.

Example: Suppose Trent is a computer notary public, who signs documents with an RSA digital signature and returns them.

Mallory wants Trent to sign a message m' that purports to be from another person.

First, Mallory chooses an arbitrary value x and computes $y = x^e \bmod n$, where e is Trent's public key.

Then he computes $m = y\,m' \bmod n$, and he sends m to Trent.

Trent returns $m^d \bmod n$.

Now Mallory calculates $(m^d \bmod n)x^{-1} \bmod n$.

This equals $m'^d \bmod n$ and is the signature of m'.

This technique is called *blinding*.

Moral: never use RSA to sign an arbitrary document presented to you by a stranger.

A one-way hash function should be used first.

Common Modulus Attack on RSA

To avoid generating a different modulus for each user, one may wish to fix n once and for all.

The same n is used by all users.

A trusted authority could provide user i with a unique pair e_i, d_i from which user i forms a public key $<n, e_i>$ and a secret key $<n, d_i>$.

However, the resulting system is insecure.

Fact 1: Let $<n, e>$ be an RSA public key. Given the private key d, one can efficiently factor the modulus $n = pq$. Conversely, given the factorization of n, one can efficiently recover d.

Proof. See [Boneh].

By Fact 1, Bob can use his own exponents e_b, d_b to factor the modulus n.

Once n is factored, Bob can recover Alice's private key d_a from her public key e_a.

Low Encryption Exponent Attack on RSA

To reduce encryption or signature-verification time, it is customary to use a small public exponent e.

However, this makes certain attacks possible.

For example, Hastad showed that if you encrypt more than $e(e + 1)/2$ linearly dependent messages with different public keys having the same value of e, Mallory can efficiently recover the plaintext (see [Boneh]).

To defeat such attacks, the value $e = 2^{16} + 1 = 65537$ is recommended.

When this value is used, signature verification requires 17 multiplications, as opposed to ~1000 when a random $e < (p - 1)(q - 1)$ is used.

Lessons Learned

Judith Moore lists restrictions in RSA suggested by such attacks:

- Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to factor the modulus.

- Knowledge of one encryption/decryption pair of exponents for a given modulus enables an attacker to calculate other encryption/decryption pairs without having to factor *n*.

- A common modulus should not be used in a protocol using RSA in a network.

- Messages should be padded with random values to prevent attacks on low encryption exponents.

- The decryption exponent should be large.

Attack on Encrypting and Signing with RSA

It is wise to sign a message before encrypting it.

With RSA there is an attack against protocols that do the reverse – encrypt before signing.

Suppose Alice wants to send a message to Bob.

She first encrypts it with Bob's public key, then signs it with her private key.

Her encrypted and signed message looks like this:

$$(m^{e_b} \bmod n_b)^{d_a} \bmod n_a$$

Bob can claim Alice sent him m' instead of m.

Since Bob knows the factorization of n_b, he can calculate discrete logarithms with respect to n_b.

Therefore, he need only find an x such that

$$m'^x = m \bmod n_b$$

Suppose Bob can publish xe_b as his new public exponent and keep n_b as his modulus.

Then he can claim that Alice sent him m' encrypted in n_b.