Introduction to Encryption

RSA Algorithm and Poker

Introduction

We will spend more time on computer security issues next term if students and TAs pursue research projects in the area. This term, we only have time for a very brief introduction.

There are many aspects to computer security including:

- **Confidentiality:** one process should not be able to access another's data. In distributed systems, messages sent from a process X to a process Y must not be accessible to a process other than X or Y.
- **Authentication:** an attacker must not be able to pretend to be someone else.
- **Integrity:** an attacker must not be able to substitute a false message for a legitimate one.
- **Nonrepudiation:** a process that sent a message m should not be able to deny that it sent m (or equivalently, if it does deny that it sent the message then other processes must be able to identify this process as one that is not obeying the protocol).

Our purpose in this term is to discuss confidentiality, and give you a brief description of the interesting problem of developing attacks, and proving that a protocol is immune to attacks.

The simplest encryption problem is this. A sender applies an encryption function f to a message M, and sends the encrypted message f.m to a receiver. The receiver applies a decrypting function g to get g.f.m. We require that g.f = I where I is the identity function. Furthermore, computing the decrypting function g, given the intercepted messagesf.m (or f.m and f.m' and f.m'' for a sequence of messages m, m', m'') should be very difficult if not impossible. Likewise, computing the plaintext messages m, m', m'' (even if the attacker does not have the decrypting function g) must also be difficult if not impossible.

In a public-key protocol, the interceptor has access to f. So, the interceptor has both f, and f.m, f.m', f.m'', ... Nevertheless, the interceptor should be unable to compute g or m, m', m'' ... Public key protocols can be more difficult because the interceptor has access to the encryption function. On the other hand, if a process X publishes its encryption f, but keeps the corresponding decryption function g secret, then other processes can communicate with X securely merely by applying f to all the messages that they send X. Thus, public key encryption simplifies the overall communication protocol.

In a symmetric protocol the interceptor does not have access to f. In symmetric protocols, g can be computed from f; usually there is a common key used for both encryption and decryption. (Public-key protocols are sometimes called asymmetric protocols.) Symmetric protocols have the advantage of relative simplicity, but the disadvantage that the communicators must agree on a shared secret, namely the function f.

An assumption made in cryptanalysis is that the attacker knows the protocol, but does not know the decryption function. In practise, the attacker may not know which of several possible protocols are being used by the communicators; nevertheless, the usual assumption is that they do know the protocol.

The RSA Algorithm

The RSA is a public-key algorithm named after its inventors: Rivest, Shamir and Adleman. The algorithm is as follows.

- Choose two large random primes p,q of equal length. For purposes of illustration we use very small primes: p = 7 and q = 11

- Let n = p * q, and let s = (p-1) * (q-1).
  For our example, n = 77 and s = 60.

- Choose an encryption key e (a positive integer) so that e is relatively prime to s, i.e., the greatest common divisor of e and s is 1. For example, choose e to be 13.

- ```
  Compute the decryption key d to satisfy the formula:
    (e*d = 1) mod s,
  which is equivalent to: there exists an integer c such that
     e*d = 1 + c*s.
  Since e is relatively prime to s, such a decryption key d exists.
  For our example, we want to compute d so that there exists an integer
  c such that:
     13*d = 1 + c*60
  We use Euclid's algorithm to compute d to be 37 because
     13*37 = 1 + 8*60
  ```

- Encrypt a message m by raising it to the power of the encryption key e, and applying mod n. Usually, a message m will be a block of several bytes, but for illustration let m be the integer 3. Then the encrypted message, f.m is 3 raised to the 13th power mod 77.

  ```
  13 in binary is 1101; hence
  3**13 mod 77=
  (3**8) * (3**4) *(3**1) mod 77 =
  38
  ```

  We apply mod 77 at every step to keep the numbers below 77. For example,
    - 3**1 mod 77 = 3
    - 3**2 mod 77 = 9
    - 3**4 mod 77 = 81 mod 77 = 4
    - 3**9 mod 77 = 4*4 mod 77 = 16

  We maintain the product mod 77, as in
    - product = 1
    - product = product * (3**1) = 3 mod 77 = 3
    - product = product * (3**4) = 3*4 mod 77 = 12
    - product = product * (3**8) = 12*16 mod 77 = 192 mod 77 = 192-154 = 38

- Decrypt a message x by raising it to the power of the decryption key d and applying mod n.

  ```
  In our example, the ciphertext message is 38, and so the plaintext message
  received is 38**37 mod 77.
  37 in binary is 100101. Hence
  ```
  ```

```
38**37 mod 77 =
(38**32)*(38**4)*(38**1) mod 77 = 3
```

We apply mod 77 at every step to keep the numbers below 77. For example,

- 38**1 mod 77 = 38
- 38**2 mod 77 = 1444 mod 77 = 1444 - 77*18 = 1444 - 1386 = 58
- 38**4 mod 77 = 58*58 mod 77 = 3364 mod 77 = 3364 - 43* 77 = 53
- 38**8 mod 77 = 53*53 mod 77 = 2809 mod 77 = 2809 - 36* 77 = 37
- 38**16 mod 77 = 37*37 mod 77 = 1369 mod 77 = 1369 - 17* 77 = 60
- 38**32 mod 77 = 60*60 mod 77 = 3600 mod 77 = 3600 - 46* 77 = 58

We maintain the product mod 77, as in

- product = 1
- product = product * (38**1) = 38 mod 77 = 38
- product = product * (38**4) = 38*53 mod 77 = 2014 mod 77 = 12
- product = product * (38**32) = 12*58 mod 77 = 696 mod 77 = 696-693 = 3

Outline: Proof of Correctness

We first prove that g.f is the identity function. To carry out this proof, we show that for any positive integer M, where M is less than e, (M**e)**d mod n = M. The proof uses Fermat's little theorem.

Fermat's little theorem says that for any M which is not a multiple of p, and any prime p: M**(p-1) mod p = 1.

```
For instance,
4**(7-1) mod 7 = 4096 mod 7 = 4096 - 7*585 = 1
3**(7-1) mod 7 = 729 mod 7 = 729 - 7*104 = 1
```

Let p and q be primes.

```
From Fermat's little theorem,
(M**(p-1))**(q-1)mod p = (M**(p-1)mod p)**(q-1) = 1**(q-1) = 1.
Therefore M**s - 1 is divisible by p. (Recall that s = (p-1)*(q-1).)
By symmetry, M**s - 1 is divisible by q.
Hence, M**s - 1 is divisible by p*q which is n.
Hence, M**s mod n = 1.

Recall that e, d are chosen so that there exists an integer c such that:
e*d = 1 + c*s.
Hence, M**(e*d) mod n =
(M**1) * (M**s)**C mod n =
M * (M**s mod n)**C mod n =
M * (1 ** C) mod n = M.
```

Attacks

An attacker has e and n. The attacker attempts to compute d. So, the attacker can attempt to find a d such that e*d mod s = 1. But, the attacker does not have s. How can the attacker compute s from n. One approach is to factor n into p and q. The problem is that there is no generally known algorithm to factor large numbers rapidly. (Perhaps the National Security Agency knows of a way, but they aren't telling.)

From CS 141.