COMP 200: Elements of Computer Science
Fall 2004
Lecture 32: November 17, 2004

Public Key Cryptography

On the Board

Final Projects — get working …

Requirements for Digital Signatures & Digital Commerce

To enable secure and trustworthy digital communication, we need an encryption system that has several key properties:

- Encrypting and decrypting messages (given the key) should be relatively inexpensive — at least, a low-order polynomial algorithm.
- Breaking the encryption — that is, decrypting it without a key — should involve solving an intractable problem — exponential cost or higher.
- The system should offer assurances to both sender and receiver
    - Message stands alone as a voucher of the transaction — that is, neither the sender nor the receiver can forge it.
    - Receiver can establish sender's unique identity
    - Sender can establish that receiver has not modified message
    - The encrypted message conveys no knowledge that would let the sender impersonate the receiver — even by forwarding the same message to a third party.
- Sender can initiate a secure communication without exchanging secret information, such as the keys required in a *single-use code* or a *substitution code*. (See last lecture.)

Each of these properties plays an important role in establishing widespread, secure digital communication. (Examples include https, ssh, ssl, & tls.)

The widely-used scheme for "public-key cryptography" was introduced in 1976 by Whitfield Diffle and Martin Helman. The best known and most widely used cryptosystem (for public-key systems) is the RSA algorithm, die to Rivest, Sharir, and Adelman.

The Model

Assume that our encryption system works through two functions:

ciphertext = *Encr* (message), and

message = *Decr* (ciphertext)

Further, we will assume that *Encr* and *Decr* are related, but that holding one does not reveal the other. That is, given *Encr* and an encrypted message, the problem of deducing *Decr* remains computationally intractable — hidden behind a problem of exponential or higher complexity. Now, if we ensure that *Encr* and *Decr* are commutative, that is

$$Decr(Encr(M)) = M \quad \text{and} \quad Encr(Decr(M)) = M$$

Then we can create a workable public-key encryption system as follows:

1. Each person has a unique *Encr* and *Decr* function, designated with a subscript.
2. Each person publishes their *Encr* function in a public registry.

As long as generating new *Encr* and *Decr* functions is relatively easy, cracking them requires solving a computationally intractable problem, and we can create a public registry of *Encr* functions, we can make this system work.

For Fred to send an encrypted message to Jane, he can simply encrypt it using Jane's "public key" — $Encr_{Jane}$ — and she can read it using her "private key" — $Decr_{Jane}$. Anyone with access to the registry can locate Jane's public key and send her a message that only she can read.

Remember, *Encr* and *Decr* must be functional inverses ***and*** knowing one cannot give you the other — unless you solve a computationally intractable problem.

So far, this scheme works. However, anyone can send Jane a message and claim to be Fred. The encrypted message depends on the message and on Jane's encryption keys, but has no context that makes it unique to Fred. Remember, the *Encr* key is published in a registry.

To create a message that can only come from Fred (a ***signed*** message), Fred can use his *Decr* process on the message to create a ciphertext that can be decoded with his publicly available *Encr*. Next, he encodes the ciphertext with Jane's *Encr*. Jane decodes the ciphertext that she receives with her

Decr process, then uses Fred's publicly registered *Encr* process to recover the original plaintext message.

The message that Fred transmits is a unique function of Fred's public and private keys (ensuring that it cannot be forged), of Jane's public and private keys (ensuring that no one but Jane can decrypt and read it), and (of course) of the original plaintext. As long as our cryptosystem (*Encr* and *Decr*) cannot be cracked (knowing *Encr*, derive *Decr*), this method securely sends a signed message from Fred to Jane.

Assume that the receiver is less than scrupulous and wants to forge the communication. Jane could create the ciphertext $Encr_{Jane}(M)$, but she cannot create the signature, since that required Fred's private secret, $Decr_{Fred}$.

Jane could send the message to someone else, by taking the $Decr_{Jane}$ of Fred's original message and encrypting it with the new recipient's publicly-registered *Encr* function. That would create a message that looked as if it were sent securely from Fred to the new recipient. This particular problem is easily avoided. If Fred includes in his encoded message enough context to make this kind of forgery obvious — such as "Jane – This message is from Fred and is intended to convey the following information: …" — then the forgery will be obvious.

(Jane cannot modify the message that Fred sent to change the name, since she needs $Decr_{Fred}(M)$ to fake his signature. If she modifies the message, it will decrypt into garbage rather than into the correct plaintext.)

RSA — How does it work?

The key to RSA cryptography is a clever use of algebra and number theory to create a system that can only be cracked by factoring very large numbers. Finding very large numbers is not that hard. Factoring them is much harder; researchers believe that no polynomial-time algorithm exists, even using randomization or probabilistic attacks (see Chapter 11).

To set up her *Encr* and *Decr* functions, Jane chooses two large (300 digit) prime numbers, *P* and *Q*, along with a public exponent *G*. Let $N = P \times Q$. The three numbers must have the properties that $(P-1)$ and *G* are relatively prime, as are $(Q-1)$ and *G*. Using a prime as *G* ensures the property, but nonprimes will work, too. A favorite choice for *G* is $2^{16} + 1$, or 65,537.

Jane chooses her private exponent, *K*, as the multiplicative inverse of

$$G \text{ modulo } (P-1) \times (Q-1),$$

so that K x $G \div ((P\text{-}1) \text{ x } (Q\text{-}1)) = 1$, or K x $G = 1$, modulo $((P\text{-}1) \text{ x } (Q\text{-}1))$. Now, she can use a simple and efficient algorithm. She posts the pair $<G,N>$ as her public key and keeps K as her private key. Now, the public *Encr* function breaks the message into numbers between 0 and $N\text{-}1$. For each such number M, *Encr* computes

M^G modulo N.

For a ciphertext H encrypted this way, the decryption process breaks it into the appropriate number of bits and computes

H^K modulo N.

Number theory and algebra show that these two functions are inverses.

To crack the private function, one must find K. However, we chose K as a function of G and the two numbers $(P\text{-}1)$ and $(Q\text{-}1)$. Since P and Q are the prime factors of G (and, thus, G has about 600 digits), we would need a major theoretical breakthrough in factoring to find P and Q in any reasonable amount of time.

The best known methods for factoring take time exponential in the length of G, so the system is considered uncrackable. As computers get faster, we can simply use longer prime numbers and gain an exponential increase in difficulty.

Public Registries and Usability

Of course, all of this choosing and factoring is something that the average computer user (such as my eleven year old daughter) does not want to do. Fortunately, software exists to easily choose the appropriate numbers, record the public keys in a registry, and build the private key into a software form where the user does not need to remember 300 digit prime numbers or their products. You may see terms such as SSL certificate — as in, a message from a mail system or an ISP that someone's SSL certificate has expired. The certificate is nothing more than a public-key pair $<G,N>$, provided by one of the many registry services that exist on the Internet. (When you register for wireless at Rice, you are asked to accept such a certificate into your browser — which conveniently keeps track of it.)