6.857 Computer and Network Security **Fall Term, 1997**

Lecture 6 : September 23, 1997

Lecturer: Ron Rivest *Scribe: Elliot Schwartz*

1 Topics Covered

- Concepts of Public-Key Cryptography
- Number Theory

2 Concepts of Public-Key Cryptography

So far our cryptographic techniques have utilized a shared secret; because of this they operate in a symmetric and point-to-point manner. Because it breaks the symmetry and allows one-to-many and many-to-one operation, public key cryptography is also called *asymmetric cryptography.*

The main idea is to separate the capability of encryption from decryption. Thus, the system has the following requirements:

- an encryption key E_A, such that $E_A(M) = C'$
- a decryption key D_A, such that $D_A(E_A(M)) = M$
- knowing E_A does not imply knowing D_A (i.e. knowing how to encrypt does not imply knowing how to decrypt)

With such a system, E_A can be published in a directory as Alice's *public key*, and people can use the key to create messages that only Alice can decrypt.

Open question: How do we organize such a directory?

The encryption function could be a deterministic, one-to-one function, but this would allow a *guess and check* attack: An adversary could compute $E_A(M')$ for a guessed M', and see if the result matches an overheard C'. Therefore, E_A should be (and is) randomized in practice.

2.1 Digital Signatures

We can use a corresponding idea to public-key encryption for providing authentication: we separate the process of signing from verifying. This allows anyone to verify that an individual signed a message. Since this is like signing a paper document, the technique is called a *digital signature*.

Requirements:

- public key verifies signature: $V_A(M, S) = \left\{ \begin{smallmatrix} true \\ false \end{smallmatrix} \right.$
- private key signs: $\sigma_A(M) = S$; transmit (M, S)
- knowing verify does not imply knowing how to sign

2.2 Miscellany

We can combine digital signatures and public-key encryption to get both privacy and authentication.

Under the Diffie-Hellman paradigm, D_A can be used to **encrypt** and E_A to **decrypt**. This is a special case, and the way RSA works. Mathematically, $\sigma_A(M) = D_A(M)$, and verification uses E_A.

A signature once valid is always valid, therefore replay attacks are possible. Attaching sequence numbers or the date to a message are common ways of allowing a verifier to remember whether he's seen the message before.

3 Number Theory

Public-key cryptography can be implemented using number theory.

For a prime p,
$Z_p = \{0, 1, \ldots, p-1\}$ forms a group under $+$ (identity is 0, associative, inverses exist)
$Z_p^\star = \{1, 2, \ldots, p-1\}$ forms a group under $\times$ (identity is 1)
Z_p actually forms a field with $+ - \times \div$ (distributive law holds, etc.)

If p is not a prime, say 10, not all numbers between 1 and 9 have multiplicative inverses modulo 10 (for example, 2 and 5 do not). So, $Z_n^\star = \{a : gcd(a, n) = 1\}, a \in \{1, 2, \ldots, n-1\}$ (a's that are relatively prime to n). $Z_{10}^\star = \{1, 3, 7, 9\}$.

Theorem 1 *If p is prime, then $Z_p^\star$ is cyclic.*

This means, there exists a generator g such that $\{g^1, g^2, \ldots, g^{p-1}\} = Z_p^\star$.
For example, in modulo 7, take $g = 3$: $g^2 = 2, g^3 = 6 = -1, g^4 = 4, g^5 = 5, g^6 = 1$.
Note that $(g^a)^b = (g^b)^a = g^{ab} \pmod{p}$.

Theorem 2 *If p is prime, then $(\forall a \in Z_p^\star) a^{p-1} \equiv 1 \pmod{p}$. (Fermat's Little Theorem)*

Lemma 1 *If p is prime, and g is a generator of $Z_p^\star$, then $g^{p-1} \equiv 1 \pmod{p}$.*

Proof: For some generator g, g generates all the elements in $Z_p^\star$. Therefore, the generator must generate a 1. If a 1 was generated before $p-1$ exponentiations, the sequence would repeat and be missing some elements. If a 1 was generated after $p-1$ exponentiations, then the sequence would have repeated before generating a 1, which is not possible. Therefore, $g^{p-1} \equiv 1 \pmod{p}$. ■

Proof: $(\forall a \in Z_p^\star) \exists x : a \equiv g^x \pmod{p}$.
Therefore, $a^{p-1} \equiv (g^x)^{p-1} \equiv (g^{p-1})^x \equiv 1^x \equiv 1 \pmod{p}$. ■

Corollary 1 $a^{-1} \equiv a^{p-2} \pmod{p}$

3.1 Fast Modular Exponentiation

To find a/b, we calculate $a \cdot b^{-1}$ using exponentiation. How do we compute $a^b \pmod{p}$ efficiently?

$$a^b = \begin{cases} 1 & b = 0 \\ (a^{b/2})^2 \pmod{p} & b \neq 0 \wedge b \equiv 0 \pmod{2} \\ a \cdot (a^{\lfloor b/2 \rfloor})^2 \pmod{p} & b \neq 0 \wedge b \equiv 1 \pmod{2} \end{cases}$$

Note that modulus p doesn't need to be prime here, since were are only doing multiplication and squarings.

Number of recursive calls is $O(\lg b)$. Total work is $O(k^3)$ for k-bit numbers a, b, p, and a algorithm for multiplying in modulo p that's $O(k^2)$. This is reasonable to do on a PC.

3.2 Finding Primes

How do we find large primes? Pick a random number, and then check with some tests.

Rabin-Miller Primality Test:

1. Given a large (odd) number p, to test for primality:
2. Pick at random $a \in \{1, 2, \ldots, p-1\}$.
3. Test if $a^{p-1} \equiv 1 \pmod{p}$. If $\neq 1$ then halt; p is not prime.
4. Test if we ever saw an r, $r \neq \pm 1$ such that $r^2 \equiv 1 \pmod{p}$ in step 3. If so, halt; p is not prime.
5. Repeat steps 2-4 20 times, or until happy.
6. Declare p to be probably prime.

Each pass has a chance of at least 1/2 of declairing p to be non-prime, if p is indeed non-prime. If p is prime, it is never declaired non-prime.

Density: About 1/1000 1000-bit numbers are prime, so this is reasonable to do on a PC.

4 References

- W. Biffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644-654, November 1976.