CS641

Modern Cryptology

Lecture 11

Key Exchange

- Exchanging keys in private-key cryptography is very challenging.
- Methods adopted include:
  - Exchange of keys in bulk at one time
  - Transfer of keys through a highly secure mechanism
- Is there a way to exchange keys without complicated methods?

Key Exchange

- If following is possible, we can exchange keys remotely:
  - There is an efficiently computable function $f : D \times D' \mapsto D$, where D, D' are suitably large sets of elements,
  - There is a $\alpha \in D$ such that for every $c, d \in D'$, $f(f(\alpha, c), d) = f(f(\alpha, d), c)$, and $f(\alpha, \cdot)$ is 1-1,
  - From α and $f(\alpha, c)$ for any $c \in D'$, it is hard to compute c.
- Given such a function f and $\alpha \in D$, a new key can be computed as follows:
  - Anubha randomly chooses $c \in D'$ and sends $f(\alpha, c)$ to Braj.
  - Braj randomly chooses $d \in D'$ and sends $f(\alpha, d)$ to Anubha.
  - Anubha computes $f(f(\alpha, d), c)$ and Braj computes $f(f(\alpha, c), d)$ as common key.
  - Ela knows α, $f(\alpha, c)$ and $f(\alpha, d)$, but cannot compute either c or d easily.

Exponentiation Function

- Function f has the property that it is easy to compute but hard to invert.
- Are there any such functions?
- Exponentiation in finite fields is believed to be one such function:
  - Let F be a finite field of size s.
  - Define $\text{Exp}(a, c) = a^c$ for $a \in F$ and $c \in [1, s-2]$.
- Function Exp is easy to compute:
  - Given a and c, compute a^2, a^4, a^8, $\ldots$ by repeated squaring.
  - Multiply a subset of these as specified by number c.
- There is no easy way to compute inverse of Exp:
  - Given a and a^c, the only way appears to compute a^2, a^3, a^4, $\ldots$ and compare with a^c.
  - The inverse problem is called Discrete Log problem.

Multiplication Function

- Multiplication of integers is believed to be another such function:
  - Define $\text{Mult}(c, d) = c * d$ for $c, d \in \mathbb{Z}$.
  - It is clearly easy to compute.
  - Given a product cd, it is not easy to compute c from it when both c and d are large prime numbers.
- Can any of these be used for key exchange?

Diffie-Hellman Key Exchange

- Proposed by Diffie and Hellman in 1976.
- Uses exponentiation in finite fields.
- First method to make remote key-exchange feasible.

Diffie-Hellman Key Exchange

- Choose a large prime number p, and consider field F_p.
- Find a generator $g \in F_p$ of cyclic group F_p^*.
- Numbers p and g are known publicly.
- Key exchange protocol runs as follows:
  - Anubha chooses a random number $c \in [1, p-2]$ and sends $g^c \pmod{p}$ to Braj.
  - Braj chooses a random number $d \in [1, p-2]$ and sends $g^d \pmod{p}$ to Anubha.
  - Anubha computes $(g^d)^c \pmod{p}$ and Braj computes $(g^c)^d \pmod{p}$ as common key.

Security of Diffie-Hellman Key Exchange

- Ela knows g, p, g^c and g^d and wants to compute g^{cd}.
- The obvious way is compute either c or d and use it to compute g^{cd}.
- Computing c or d requires solving Discrete Log Problem in F_p, believed to be hard.
- The fastest known algorithm for solving Discrete Log Problem takes time $2^{\Omega((\log p)^{1/3}(\log\log p)^{2/3})}$.
- Secure provided p is chosen to be at least 1024 bits long.

Public-Key Encryption

- Proposed by Diffie-Hellman inspired by key exchange protocol.
- In this, encryption key k_E is publicly known. Only decryption key k_D is secret.
- No need for key exchange now, as k_E is known to everyone.
- To communicate to Braj remotely, Anubha needs to ask Braj for his public key and use it to encrypt the secret message.

Public-Key Encryption

- Following makes public-key encryption possible:
  - There is an efficiently computable function $f : D \mapsto D$, where D is a suitably large set of elements.
  - Function f is 1-1, and hard to invert.
  - There are efficiently computable functions E and D such that $D(E(m, f(k_D)), k_D) = m$ for every m and k_D.
  - Function $E(\cdot, f(k_D))$ is also hard to invert.
- We set $k_E = f(k_D)$ and use E and D for encryption and decryption.
- This requires two hard to invert functions: f and E.

RSA Public Key Algorithm

- Proposed by Rivest, Shamir, and Adleman in 1977.
- Uses both exponentiation and multiplication functions:
  - Multiplication to define function f,
  - Exponentiation to define E and D.

Group Z_n^*

- Let $n \in \mathbb{Z}$, $n > 1$.
- Let $(Z_n, +, *)$ be the ring with $Z_n = \{0, 1, \ldots, n-1\}$, and addition and multiplication modulo n.
  - Division may not always be possible in Z_n for composite n.
  - For example, division by 2 is not possible in Z_6 since $2 * 3 = 0$.
- Let $Z_n^* = \{a \mid \gcd(a, n) = 1\}$.
- Z_n^* is a commutative group under multiplication inside Z_n:
  - For any $a \in Z_n^*$, $\gcd(a, n) = 1$, that is, $ab + kn = 1$ for some $b, k \in \mathbb{Z}$.
  - Therefore, $a * b = 1 \pmod{n}$, implying that division by a is possible in Z_n.
  - If $\gcd(a_1, n) = 1 = \gcd(a_2, n)$, then $\gcd(a_1 a_2, n) = 1$ too.
- If $\gcd(a, n) = k > 1$, then $a * \frac{n}{k} = 0$ in Z_n, implying that division by a is not possible.

Group Z_n^*

- Let $\phi(n)$ be the count of numbers $< n$ that are relatively prime to n.
- Then, $|Z_n^*| = \phi(n)$.
- If $n = \prod_{i=1}^{r} p_i^{e_i}$, then $\phi(n) = \prod_{i=1}^{r} p_i^{e_i - 1}(p_i - 1)$.
- Specifically, if $n = pq$ for primes p and q, then $\phi(n) = (p-1)(q-1)$.

RSA Key Generation

- Choose two large prime numbers p and q, each of ℓ bits long.
- Compute $n = pq$.
- Randomly choose a number e such that:
  - $1 < e < \phi(n)$, and
  - $\gcd(e, \phi(n)) = 1$.
- Compute d such that:
  - $1 < d < \phi(n)$, and
  - $de = 1 \pmod{\phi(n)}$.
- Encryption key $k_E = (e, n)$. Also called public key.
- Decryption key $k_D = (d, n)$. Also called private key.

RSA Encryption

- Given a plaintext, break it into blocks of size $2\ell - 1$ bits. Recall size of n is 2ℓ bits.
- Treat each block as a number in Z_n.
- Let m be any such block.
- Define $c = E(m, (e, n)) = m^e \pmod{n}$.
- Define $D(c, (d, n)) = c^d \pmod{n}$.
- Suppose $m \in Z_n^*$. Then:

$$c^d = m^{ed} = m^{1+k\phi(n)} = m * (m^{\phi(n)})^k = m \pmod{n},$$

since $a^{\phi(n)} = 1 \pmod{n}$ for every $a \in Z_n^*$.

RSA Encryption

- Suppose $m \notin Z_n^*$.
- This implies that $\gcd(m, n)$ equals either p or q.
- Suppose $\gcd(m, n) = p$.
- Then $\phi(n) = (p-1)(q-1)$ can be computed easily.
- Then $d = e^{-1} \pmod{\phi(n)}$ can be computed easily.
- This breaks the encryption!