Asymmetric Cryptography

Daniel Halperin
Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...
1 secret key, shared between sender/receiver

Repeat fast and simple operations lots of times (rounds) to mix up key and ciphertext

Why do we think it is secure? (simplistic)

- If we do lots and lots and lots of mixing, no simple formula (and reversible) describing the whole process (cryptographic weakness).
- Mix in ways we think it’s hard to short-circuit all the rounds. Especially non-linear mixing, e.g., S-boxes.
- Some math gives us confidence in these assumptions
Public Key Cryptography
Basic Problem

**Given:** Everybody knows Bob’s public key
Only Bob knows the corresponding private key

**Goals:**
1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate himself
Public-Key Cryptography

- Everyone has 1 private key and 1 public key
- Mathematical relationship between private and public keys
- Why do we think it is secure? (simplistic)
  - Relies entirely on problems we believe are “hard”
Applications of Public-Key Crypto

♦ Encryption for confidentiality
  • Anyone can encrypt a message
    – With symmetric crypto, must know secret key to encrypt
  • Only someone who knows private key can decrypt
  • Key management is simpler (or at least different)
    – Secret is stored only at one site: good for open environments

♦ Digital signatures for authentication
  • Can “sign” a message with your private key

♦ Session key establishment
  • Exchange messages to create a secret session key
  • Then switch to symmetric cryptography (why?)
**Diffie-Hellman Protocol (1976)**

- Alice and Bob never met and share no secrets
- **Public info:** $p$ and $g$
  - $p$ is a large prime number, $g$ is a generator of $\mathbb{Z}_p^*$
    - $\mathbb{Z}_p^* = \{1, 2 \ldots p-1\}$; $\forall a \in \mathbb{Z}_p^* \exists i$ such that $a = g^i \mod p$
    - Modular arithmetic: numbers “wrap around” after they reach $p$

Alice

- Pick secret, random $X$
- Compute $k = (g^y)^x = g^{xy} \mod p$

Bob

- Pick secret, random $Y$
- Compute $k = (g^x)^y = g^{xy} \mod p$
Why Is Diffie-Hellman Secure?

- **Discrete Logarithm (DL) problem:**
  - given $g^x \mod p$, it’s hard to extract $x$
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure!

- **Computational Diffie-Hellman (CDH) problem:**
  - given $g^x$ and $g^y$, it’s hard to compute $g^{xy} \mod p$
  - ... unless you know $x$ or $y$, in which case it’s easy

- **Decisional Diffie-Hellman (DDH) problem:**
  - given $g^x$ and $g^y$, it’s hard to tell the difference between $g^{xy} \mod p$ and $g^r \mod p$ where $r$ is random
Properties of Diffie-Hellman

- Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Eavesdropper can’t tell the difference between established key and a random value
  - Can use new key for symmetric cryptography
    - Approx. 1000 times faster than modular exponentiation
- Diffie-Hellman protocol (by itself) does not provide authentication
Properties of Diffie-Hellman

- DDH: not true for integers mod p, but true for other groups
- DL problem in p can be broken down into DL problems for subgroups, if factorization of p-1 is known.
- Common recommendation:
  - Choose $p = 2q+1$ where q is also a large prime.
  - Pick a $g$ that generates a subgroup of order q in $\mathbb{Z}_p^*$
    - DDH is hard for this group
    - (OK to not know all the details of why for this course.)
  - Hash output of DH key exchange to get the key
Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: $p$ and $g$
  - $p$, $q$ are large prime numbers, $p=2q+1$, $g$ a generator for the subgroup of order $q$
    - Modular arithmetic: numbers “wrap around” after they reach $p$

\[ \text{Compute } k = H((g^y)^x) = H(g^{xy} \mod p) \]
\[ \text{Compute } k = H((g^x)^y) = H(g^{xy} \mod p) \]
Requirements for Public-Key Encryption

- **Key generation**: computationally easy to generate a pair (public key PK, private key SK)
  - Computationally infeasible to determine private key SK given only public key PK

- **Encryption**: given plaintext M and public key PK, easy to compute ciphertext C=E_{PK}(M)

- **Decryption**: given ciphertext C=E_{PK}(M) and private key SK, easy to compute plaintext M
  - Infeasible to compute M from C without SK
  - Even infeasible to learn partial information about M
  - **Trapdoor** function: Decrypt(SK,Encrypt(PK,M))=M
Some Number Theory Facts

- Euler totient function $\varphi(n)$ where $n \geq 1$ is the number of integers in the $[1, n]$ interval that are relatively prime to $n$
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
- Euler’s theorem:
  if $a \in \mathbb{Z}_n^*$, then $a^{\varphi(n)} = 1 \mod n$
  $\mathbb{Z}_n^*$: multiplicative group of integers mod $n$ (integers relatively prime to $n$)
- Special case: Fermat’s Little Theorem
  if $p$ is prime and $\gcd(a, p) = 1$, then $a^{p-1} = 1 \mod p$
RSA Cryptosystem

[ Rivest, Shamir, Adleman 1977 ]

Key generation:
- Generate large primes p, q
  - Say, 1024 bits each (need primality testing, too)
- Compute n=pq and \( \varphi(n) = (p-1)(q-1) \)
- Choose small e, relatively prime to \( \varphi(n) \)
  - Typically, e=3 or e=2^{16}+1=65537 (why?)
- Compute unique d such that ed = 1 mod \( \varphi(n) \)
- Public key = (e,n); private key = (d,n)

Encryption of m: \( c = m^e \mod n \)
- Modular exponentiation by repeated squaring

Decryption of c: \( c^d \mod n = (m^e)^d \mod n = m \)