ECE 443/518 – Computer Cyber Security
Lecture 09 The RSA Cryptosystem

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

September 18, 2024

Outline

Reading Assignment

- This lecture: UC 6,7, except 7.6
- Next lecture: UC 8.1,8.5,13.3.1

Outline

Modular n-th Root

Public-Key Cryptography

RSA

Modular n-th Root where m is not Prime

$$x^n \equiv a \pmod{m}.$$

- What if m is not a prime number?
- Consider $m = pq$ where $p \neq q$ are both prime numbers.
- Idea
  - Solve the equation for p and q individually.
  - Then combine the results.

Solve Modular n-th Root

$$x^n \equiv a \pmod{m}, \quad \text{where} \quad m = pq.$$

- By Fermat's Little Theorem,
  - For $ny \equiv 1 \pmod{p-1}$, $(a^y)^n \equiv a \pmod{p}$.
  - For $ny' \equiv 1 \pmod{q-1}$, $(a^{y'})^n \equiv a \pmod{q}$.
- By Chinese Remainder Theorem,
  - If we can choose $y = y'$, then $(a^y)^n \equiv a \pmod{pq}$.
  - This is possible if $gcd(n, (p-1)(q-1)) = 1$.
  - Solve $ny \equiv 1 \pmod{(p-1)(q-1)}$ to obtain y.
- We can solve $x^n \equiv a \pmod{pq}$ if $gcd(n, (p-1)(q-1)) = 1$.
  - Solution: $x \equiv a^y \pmod{m}$, or practically $x = a^y \bmod m$.
  - Time complexity: $O(N^3)$
- Note that you cannot use this method to solve the seemingly very simple case of $x^2 \equiv a \pmod{pq}$.

Example

- Solve $x^5 \equiv 197 \pmod{221}$.
  - $221 = 13 * 17$
- Apply EEA to solve $5y \equiv 1 \pmod{192}$
  - $y \equiv 77 \pmod{192}$
- To compute $x \equiv 197^{77} \pmod{221}$,
  - Use Chinese Remainder Theorem to simplify computation.
  - $x \equiv 197^{77} \equiv 2^{77} \equiv 2^5 \equiv 6 \pmod{13}$
  - $x \equiv 197^{77} \equiv 10^{77} \equiv 10^{13} \equiv 11 \pmod{17}$
  - $x \equiv 45 \pmod{221}$

An Observation

$$x^n \equiv a \pmod{m}.$$

- But what if you don't know p and q for $m = pq$?
  - Factor m into pq first, or
  - Brute force: try $x = 1, 2, \ldots, m-1$
- What are their time complexities?
  - Any better algorithms?
- Is this observation of any practical importance?

Outline

Modular n-th Root

Public-Key Cryptography

RSA

Symmetric Cryptography Revisited

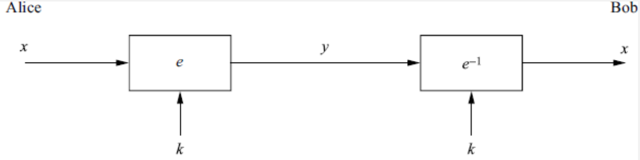

Fig. 6.1 Principle of symmetric-key encryption

(Paar and Pelzl)

- With the use of MAC as needed.
- Issue with Key Distribution: to establish a secret channel using symmetric cryptography, Alice and Bob need a secret channel to share the secret key k.
- Issue with Number of Keys: for a group of n people to communicate securely among each two of them, each people need to manage n keys and a total of $\frac{n(n-1)}{2}$ keys are needed.
- Issue with Nonrepudiation: Alice cannot prove to a third party that a ciphertext (with MAC) was sent by Bob as she also know the secret key k to generate the ciphertext.

Public-Key Cryptography

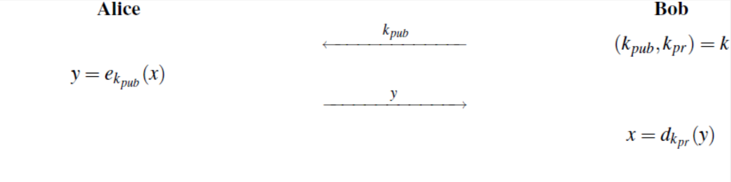

Fig. 6.4 Basic protocol for public-key encryption

(Paar and Pelzl)

- Key pair k: a public k_{pub} and a private (secret) k_{pr}.
  - No one should be able to derive k_{pr} from k_{pub}.
- Key Distribution: to establish a secret channel, Alice only need to obtain Bob's k_{pub} via an authentic channel.
- Number of Keys: each people just need to manage 1 key no matter how many people are there in the group.
- Nonrepudiation: via digital signatures if roles of k_{pr} and k_{pub} can be exchanged.
- Only if we could find such a cipher ...
  - For computationally unbounded adversaries?
  - For computationally bounded adversaries?

A Simple Hybrid Protocol

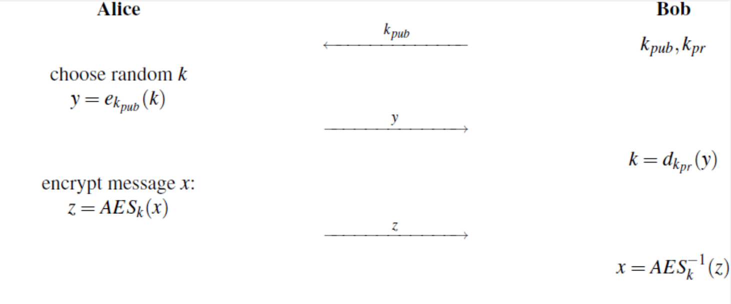

Fig. 6.5 Basic key transport protocol with AES as an example of a symmetric cipher

(Paar and Pelzl)

- In practice, symmetric ciphers remain very useful as public-key ciphers are usually orders of magnitude slower.
  - Use public-key ciphers to create a "slower" secure channel from an authentic channel between Alice and Bob.
  - Then Alice and Bob can use this "slower" secure channel to establish the secret key for symmetric ciphers, and thus create a "faster" secure channel.

Outline

Modular n-th Root

Public-Key Cryptography

RSA

History of RSA

- 1977: created by Ronald Rivest, Adi Shamir and Leonard Adleman
- 1983: RSA patent granted in US
- 1997: Clifford Cocks' equivalent system when working in the British intelligence agency GCHQ in 1973 was declassified.
- 2000: RSA patent expired in US

RSA Key Generation

- Choose two prime numbers p and q.
- Compute $n = pq$.
- Choose a positive integer e such that $gcd(e, (p-1)(q-1)) = 1$.
- Solve $de \equiv 1 \pmod{(p-1)(q-1)}$ for a positive integer d.
- Public key: $k_{pub} = (n, e)$
- Private key: $k_{pr} = (p, q, d)$

RSA Encryption

- Public key: $k_{pub} = (n, e)$
- Plaintext: $x \in \{0, 1, \ldots, n-1\}$.
- Encryption: $y = e_{k_{pub}}(x) = x^e \bmod n$.
  - Ciphertext: $y \in \{0, 1, \ldots, n-1\}$.
- Example: $k_{pub} = (n = 221, e = 5)$
  - $x = 45$, $y = 45^5 \bmod 221 = 197$.

RSA Decryption

- Private key: $k_{pr} = (p, q, d)$
- Decryption: $x = d_{k_{pr}}(y) = y^d \bmod pq$.
- Example: $k_{pr} = (p = 13, q = 17, d = 77)$
  - $y = 197$, $x = 197^{77} \bmod 221 = 45$.
- Use a public key from *Bob*, Alice can only encrypt the message but cannot decrypt the message.
  - Why? What are our assumptions?

Oscar's Attacks

- Oscar knows $k_{pub} = (n, e)$ and the ciphertext y.
  - Assume n to be N bits.
- Apply brute force to find x
  - Need $O(2^N)$ time.
- Factor n into p and q
  - Apply integer factorization.
  - If p and q are chosen to be around $\frac{N}{2}$-bit, then this will take Oscar $O(2^{\frac{N}{2}})$ time.
- Both are not practical for large N.
  - At least $N = 2048$ to be secure in long term.

Padding

- Oscar may derive useful statistics about plaintext from ciphertext since RSA is deterministic.
- Oscar may recover small x if e is small by trying to compute $\sqrt[e]{y}$, $\sqrt[e]{y+n}$, etc. using usual (non-modular) math.
- Oscar may modify y to change the plaintext in predictable ways: for any chosen s, if $y' = s^e y$, then $x' = d_{k_{pr}}(y') = sx$.
- Use padding to introduce random structure into plaintext.
- E.g. Optimal Asymmetric Encryption Padding (OAEP) in Public Key Cryptography Standard #1 (PKCS #1).
- A lot of other considerations for both security and performance.

Summary

- RSA
  - Key generation: by Bob, $k_{pub} = (n, e)$, $k_{pr} = (p, q, d)$
  - Encryption: everyone, $y = e_{k_{pub}}(x) = x^e \bmod n$.
  - Decryption: Bob only, $x = d_{k_{pr}}(y) = y^d \bmod pq$.
  - Assumption: Oscar cannot factorize n into p and q in polynomial time.
- Similar to other cryptosystems, there are a lot of pitfalls for actual implementaion – you should follow documented standards exactly or use an established library instead.