1.264 Lecture 28

Cryptography:
Asymmetric keys

Next class: Anderson chapters 20. Exercise due <u>before</u> class
(Reading doesn't cover same topics as lecture)

Asymmetric or “public key” encryption

- Key pairs: public key for encryption, private key for decryption
  - RSA: 1024-2048 bit, in common use for Web and email
  - Patent expired in 2005
- Problem with public key algorithms
  - Speed: RSA is 1000 times slower than symmetric algorithms
    - Problem avoided by using RSA to exchange a symmetric session key and then using symmetric encryption method for the rest of the session.
    - Use a different symmetric key each session to limit damage if key is broken

Public key (RSA) concept

- Public key P is pair of integers (N, p)
- Secret or private key S is pair of integers (N, s)
- Generate 3 large random prime numbers (Fermat's Little Thm)
  - Largest is s. Call the other two x and y.
  - N= xy
  - p= smallest integer such that (ps) mod (x-1)(y-1)= 1
- Break message into a series of chunks m_i
- Encrypt message chunk m_i to ciphertext chunk c_i by:
  - $c = m^p \bmod N$
- Decrypt ciphertext chunk c_i to plaintext m_i by:
  - $m_i = c^s \bmod N$
- s is hard to compute from N and p
  - Requires knowledge of x and y, which requires factoring N
  - Factoring is exponential time algorithm, so if the number to be factored is big enough, it takes a very long time...

Exercise (again, very simplified)

- Code GETURL as A=01, B=02…E=05,G=07,L=12,R=18,T=20,U=21:
  - ______________________________
- Generate 3 random primes: 47, 79, 97 (way too small in real life!)
- Use s= __, x= __, y= __. Verify that p= 37 using
  - (ps) mod (x-1)(y-1) = 1
- Compute N= xy: ______
- Break the message into three 4-digit chunks:
  - _____ _____ _____
- Create ciphertext: raise each chunk to the p power % N: (% =mod)
  - _____ _____ _____
  - 0705^{37} % 3713=0564, 2021^{37} % 3713= 1645, 1812^{37} % 3713= 3378
- Retrieve plaintext: raise each chunk to s power % N:
  - _____ _____ _____
  - 0564^{97} % 3713=0705, 1645^{97} % 3713= 2021, 3378^{97} % 3713= 1812

Solution

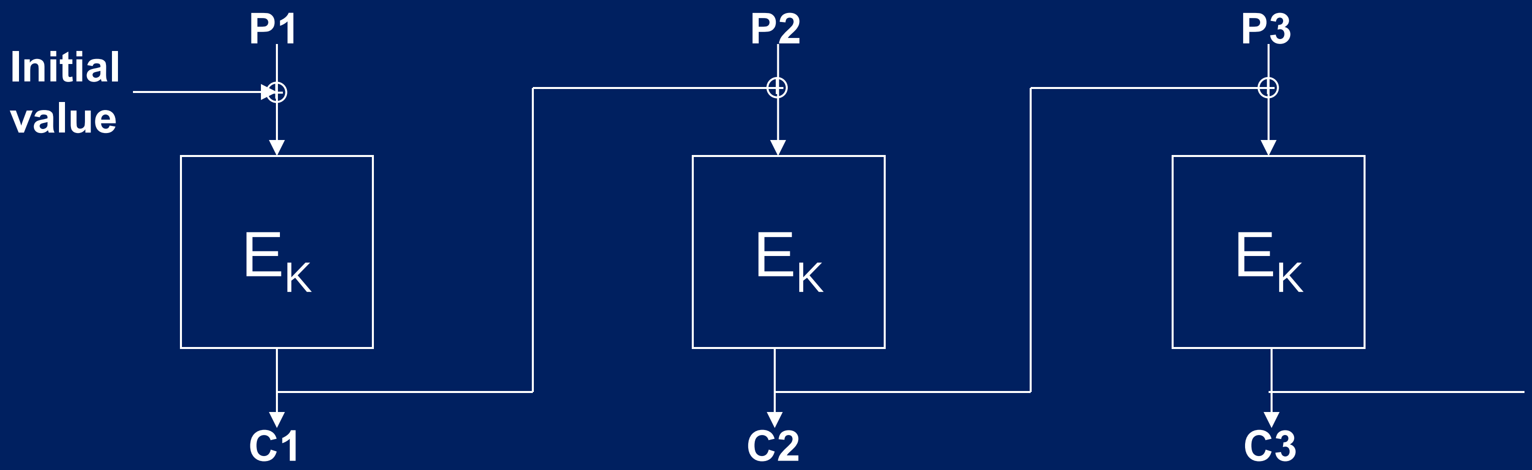
- Code GETURL as A=01, B=02…E=05,G=07,L=12,R=18,T=20,U=21:
  - 070520211812
- Generate 3 random primes: 47, 79, 97 (way too small in real life!)
- Use s= 97, x= 47, y= 79. Verify that p= 37 using
  - (ps) mod (x-1)(y-1) = 1
- Compute N= xy: 3713
- Break the message into three 4-digit chunks:
  - 0705 2012 1812
- Create ciphertext: raise each chunk to the p power % N: (% =mod)
  - 0564 1645 3378
  - 0705^{37} % 3713=0564, 2021^{37} % 3713= 1645, 1812^{37} % 3713= 3378
- Retrieve plaintext: raise each chunk to s power % N:
  - 0705 2021 1812
  - 0564^{97} % 3713=0705, 1645^{97} % 3713= 2021, 3378^{97} % 3713= 1812

How secure are AES and RSA?

- Questions to ask:
  - Is algorithm correct?
    - Yes, though if we learn to factor large numbers, RSA is dead
  - Is algorithm coded correctly, including chaining, digests, padding of short blocks, etc?
    - Often a vulnerability: 2010 ASP.NET break
  - Is key management correct and secure?
    - Often a vulnerability; keys protected by weak passwords, revocation lists not checked, etc.
  - Can a message be cracked by brute force?
    - Only if key is too short: 128-256 bits for symmetric is ok, 1024-2048 for RSA is ok.
    - Only keys for high value assets (defense, nuclear, etc.) merit the effort to crack them

Ciphers

- When a message is longer than the key (the usual case)
  - We exclusive-or (add bits without carrying) each block of plaintext with the previous block of ciphertext before encrypting it
  - This disguises any patterns in plaintext
    - Repeated plaintext is coded differently each time it appears



Message digests

- Cryptographic hashes are a one-way function that creates a short number (128 to 160 bits, often) that is very unlikely to be generated by any other message
  - Many hashes are the last (chained) block cipher of a message, so it depends on the entire message
  - It's used to verify that the message has not been altered
- Common message digests:
  - MD4: 3 rounds, 128 bit hash
  - MD5: 4 rounds, 128 bit hash
  - sha1: 5 rounds, 160 bit hash
  - sha256: 64 rounds, 256 bit hash
  - sha512: 80 rounds, 512 bit hash

Digital signatures/certificates

- Use public/private key in opposite fashion from message encryption to provide sender authentication
  - Sender signs document with her private key
  - Receiver decrypts with sender's public key
  - If the decryption is correct, message must have been sent by sender
- Compare:
  - Encryption:
    - Sender encrypts message with receiver public key and sends
    - Receiver decrypts with receiver's private key
    - This allows any sender to send secure messages to any receiver
    - Secure Sockets Layer(SSL) distributes public keys– covered next
  - Digital signature:
    - Sender signs message with own private key and sends
    - Receiver decrypts with sender's public key
    - This allows any receiver to verify the sender of any message

Digital signatures/certificates, cont.

- Digital signatures are implemented using <u>certificates</u>
  - These are the MIT certificates we all have on our computers
- Problems with digital signatures
  - Spoofer can cut and paste encrypted signature from old message to new faked message.
    - One solution is for receiver to send ‘challenge phrase’ to sender
    - Sender then encrypts with sender private key and sends to receiver, who can check if it’s what she sent initially
  - Spoofer can alter parts of the message
    - Solution is message digest functions to provide integrity check
      - Message digest is function run on entire message that produces short digest, often 128 bits (note that 2^{128} is a very big number of combinations!)
      - Send digest and message. Receiver runs digest algorithm on message and checks if same value

Glossary

- RSA: Rivest-Shamir-Adleman: asymmtric encryption algorithm
- AES: American Encryption Standard: symmetric encryption algorithm
- MDx, sha-x: hash or message digest functions to ensure message integrity
- SSL: Secure Sockets Layer, protocol for entire transaction
- TLS: Transport Layer Security, successor to SSL