Securing Communication: Cryptography

EE 122: Intro to Communication Networks

Fall 2007 (WF 4-5:30 in **Cory 277**)

Vern Paxson

TAs: Lisa Fowler, Daniel Killebrew & Jorge Ortiz

http://inst.eecs.berkeley.edu/~ee122/

Materials with thanks to Jennifer Rexford, Ion Stoica, and colleagues at Princeton and UC Berkeley

1

Announcements

- I will have extra office hours Mon Dec 3, 3-4PM
- What particular review topics would you like to have covered in the final lecture?

2

Goals of Today's Lecture

- How can we secure our use of networks?
- Requirements for secure communication
- Technology for secure communication: cryptography
  - Symmetric encryption (secret key)
  - Asymmetric encryption (public key)
  - Cryptographic hash functions (integrity, signatures)
- Classes of attacks on cryptosystems

3

Requirements for Secure Communication

- Authentication: who is this actor?
  - Attacker counterpart: *spoofing*
- Authorization: is this actor allowed to do what they request?
  - Attacker counterpart: *compromise*
- Accountability/Attribution: who did this activity?
  - For messages, *non-repudiation*
    - Sender can't later claim didn't send it
    - Receiver can't claim didn't receive it
  - Attacker counterpart: *framing*
- Integrity: do messages arrive in their original form?

4

Requirements for Secure Communication

- Confidentiality: is communication free from eavesdropping?
  - Attacker counterpart: *sniffing, man-in-the-middle*
- Availability: can you use the network / a service when you want to?
  - Attacker counterpart: *Denial-of-Service* (DoS), *theft-of-service*
- Audit/forensics: what occurred in the past?
  - A broader notion of accountability/attribution
- Appropriate use: policies regarding use of resources
  - E.g., no spam; no games during business hours; etc.

5

Securing Communication: Cryptography

- Cryptography: *communication in the presence of adversaries*
- Studied for thousands of years
  - See the Simon Singh's *The Code Book* for an excellent, highly readable history
- Central goal: how to encode information so that an adversary can't extract it …
  - … but a friend can
- General premise: there is a **key**, possession of which allows decoding, but without which decoding is infeasible
  - Thus, key must be kept **secret** and not guessable

6

Symmetric Key Encryption

- Same key for encryption and decryption
- When used for communication, central problem is key distribution
  - How do the parties agree on the key?
- How big should the key be?
- What can you do with a huge key?
- **One-time pad**: huge key of random bits
  - To encrypt: just XOR with the key! (same to decrypt)
  - *Provably secure!* …. provided:
    - You **never** reuse the key …
    - … and it really is random/unpredictable
  - Spies actually use these

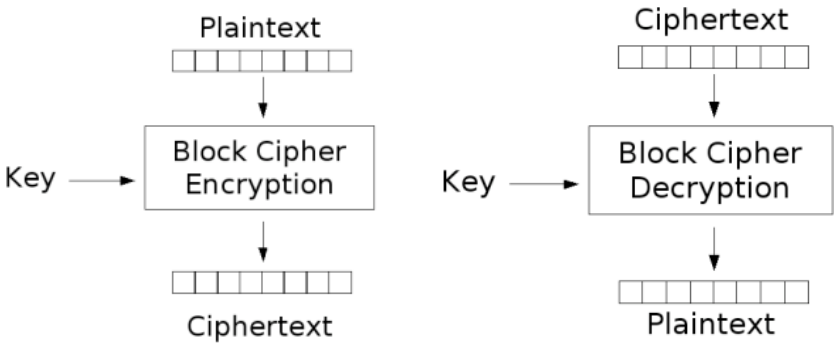
7

Shorter Symmetric Keys

- One way to *approximate* a one-time pad: generate a (very good) pseudo-random number stream
  - And XOR the plaintext with it to get the ciphertext
  - Key is the "seed" used to initialize the generator
- More general: algorithms that produce keyed *permutations* of their input
  - Permutation = different inputs mapped to different outputs
  - Necessary so that decryption recovers a unique original
  - Key selects between zillions of possible permutations
  - Works with a *block size* (e.g., 64 bits)
    - To encrypt a stream, can encrypt blocks separately, or link them
  - Note: output is same size as input (other than *padding*)

8

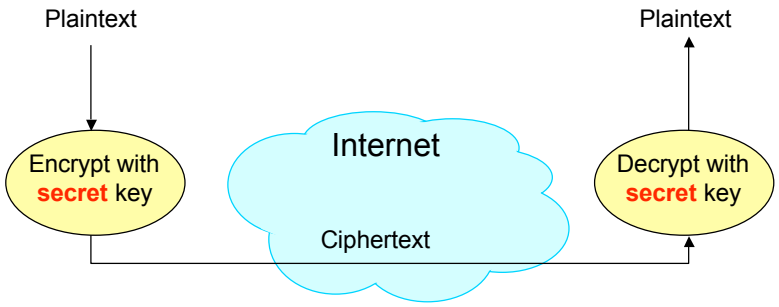
Operation of Symmetric Key Cipher



9

Using Symmetric Keys

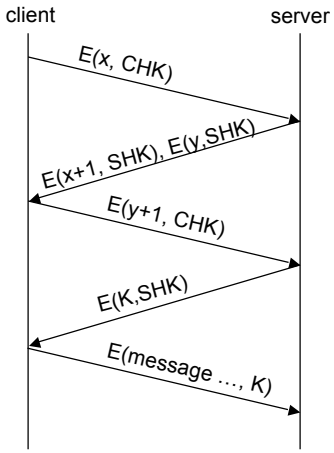
- Both the sender and the receiver use the same secret keys



10

Symmetric Crypto for Authentication

- Client's secret key: CHK
- Server's secret key: SHK
- Does CHK = SHK?
- Notation: E(m,k) – encrypt message m with key k
- x, y: **nonces** (random values)
  - Avoid replay attacks, e.g., attacker impersonating client or server
- K – *session key* used for data communication
  - minimize # of messages containing CHK / SHK



11

Symmetric Key Ciphers - DES & AES

- Data Encryption Standard (DES)
  - Developed by IBM in 1970s, standardized by NBS/NIST
  - 56-bit key (decreased from 64 bits at NSA's request)
  - Still fairly strong other than brute-forcing the key space
    - But custom hardware can crack a key in < 24 hours
  - Today many financial institutions use Triple DES
    - = DES applied 3 times, with 3 keys totaling 168 bits
- Advanced Encryption Standard (AES)
  - Replacement for DES standardized in 2002
  - Key size: 128, 192 or 256 bits
- How fundamentally strong are they?
  - **No one knows** (no proofs exist)

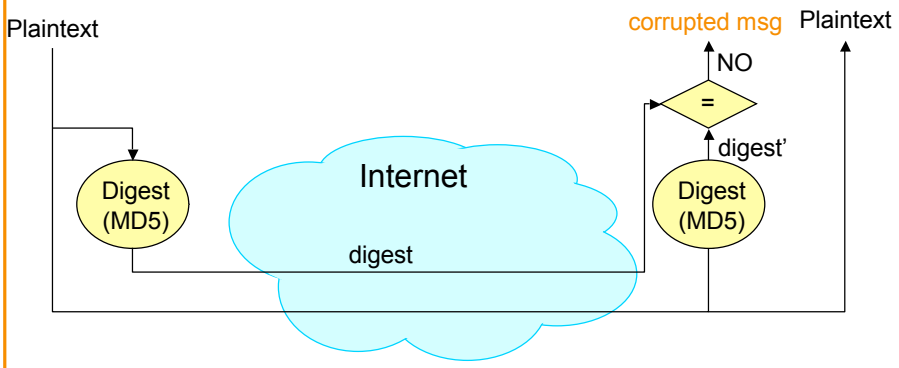
12

Integrity: Cryptographic Hashes

- Basic building block for integrity: *hashing*
  - Associate hash with byte-stream, receiver verifies match
    - Assures data hasn't been modified, either accidentally - or maliciously
  - TCP checksum a very simple (weak) such hash
- Allows us to succinctly refer to large data items
- Approach:
  - Sender computes a *digest* of message **m**, i.e., H(m)
    - H() is a publicly known *hash function*
  - Send digest (**d** = H(**m**)) to receiver in a secure way, e.g.,
    - Using another physical channel
    - Using encryption
  - Upon receiving **m** and **d**, receiver re-computes H(**m**) to see whether result agrees with **d**

13

Operation of Hashing for Integrity



14

Cryptographically Strong Hashes

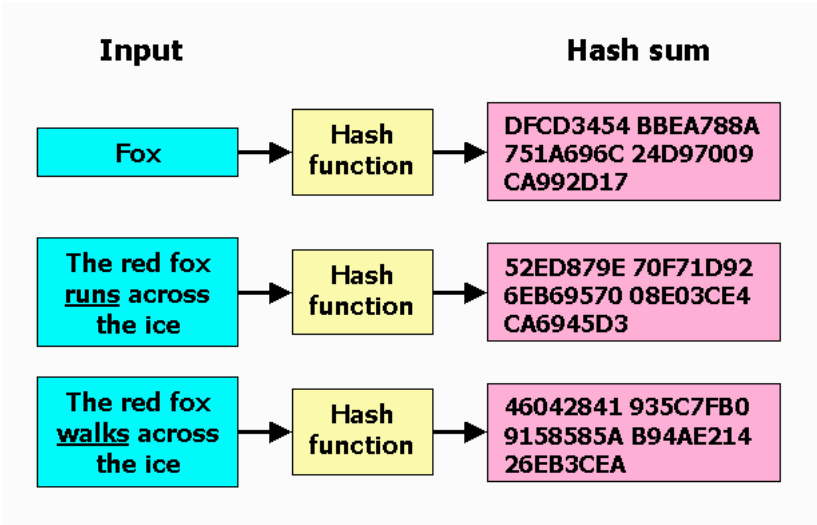
- Desired properties when faced with an adversary:
  - Hard to invert
    - Given hash, adversary can't find input that produces it
  - Hard to find collisions
    - Adversary can't find two inputs that produce the same hash

⇒ Someone cannot alter the message without modifying the digest

- Hashes let us
  - Succinctly refer to large objects
  - Obliquely refer to private objects (e.g., passwords)
    - Send hash of object rather than object itself (since hard to invert)
    - Can prepend a (secret) key so that hashes of known items is unpredictable

15

Effects of Cryptographic Hashing



Standard Cryptographic Hash Functions

- MD5 (Message Digest version 5)
  - Developed in 1991 (Rivest)
  - Produces 128 bit hashes
  - Widely used (RFC 1321)
  - **Broken**:
    - Recent work quickly finds collisions
- SHA-1 (Secure Hash Algorithm)
  - Developed by NSA in 1995 as successor to MD5
  - Produces 160 bit hashes
  - Widely used (SSL/TLS, SSH, PGP, IPSEC)
  - **Broken**:
    - Recent work finds collisions, though not really quickly … yet

17

5 Minute Break

Questions Before We Proceed?

18

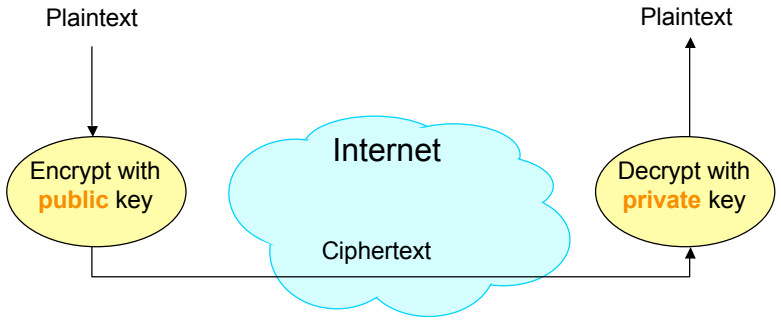
Asymmetric Encryption *(Public Key)*

- Idea: use two *different* keys, one to encrypt (***e***) and one to decrypt (***d***)
  - A key pair
- Crucial property: knowing ***e*** does not give away ***d***
- Therefore ***e*** can be public: everyone knows it!
- If Alice wants to send to Bob, she fetches Bob's public key (say from Bob's home page) and encrypts with it
  - Alice can't decrypt what she's sending to Bob …
  - … but then, neither can anyone else (except Bob)

19

Public Key / Asymmetric Encryption

- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret



20

Realizing Public Key Cryptography

- Invented in the 1970s
  - *Revolutionized* cryptography
  - (Was actually invented earlier by British intelligence)
- How can we construct an encryption/decryption algorithm using a key pair with the public/private properties?
  - Answer: Number Theory
- Most fully developed approach: RSA
  - Rivest / Shamir / Adleman, 1977; RFC 3447
  - Based on modular multiplication of very large integers
  - Very widely used (e.g., SSL/TLS for **`https`**)

21

RSA Public / Private Key Pairs

- Choose two large prime numbers p and q (~ 256-512 bits long) and multiply them: $n = p \cdot q$
- Choose encryption exponent e such that e and $(p-1) \cdot (q-1)$ are *relatively prime*
- Compute decryption key d as

  $d \equiv e^{-1} \mod ((p-1) \cdot (q-1))$

  (equivalent to $d \cdot e \equiv 1 \mod ((p-1) \cdot (q-1))$)
- Public key consists of pair (n, e)
  - Often e takes on one of a few common values
    - e.g., 65537 ….. Or even just 3!
- Private key consists of pair (d, n)

22

RSA Encryption and Decryption

- Encryption of message block *m*:

$$c = E(m, e) = m^e \bmod n$$

- Decryption of ciphertext *c*:

$$m = D(c, d) = c^d \bmod n$$

  - –Works due to number-theoretic properties
  - –Note: D(E(x, e), d) = E(D(x, d), e) = x
    - o I.e., D & E are **inverses**

23

RSA Example: Deriving Keys

- Choose p = 7 and q = 11 ⇒ n = p•q = 77
- Compute encryption key ***e***:
  (p-1)•(q-1) = 6•10 = 60 ⇒
  chose e = 13 (13 & 60 relatively prime)
- Compute decryption key ***d*** such that
  13•d ≡ 1 mod 60 ⇒
  d = 37 (37•13 = 481 = 60•8 + 1)

24

RSA Example: Encrypt/Decrypt

- Public key: (n = 77, e = 13); private: (n = 77, d = 37)
- Suppose the message we want to encrypt is the bitstring 0x0000111 - i.e., m = 7
- Encryption: **$c = m^e \bmod n$**
  $= 7^{13} \bmod 77$
  $= 96{,}889{,}010{,}407 \bmod 77$
  $= 35$ (**ciphertext**)
- Decryption: **$m = c^d \bmod n$**
  $= 35^{37} \bmod 77$
  $= 1350571686708832152540938380931547726504504680633544921875 \bmod 77$
  $= 7$ (**recovered plaintext**)

25

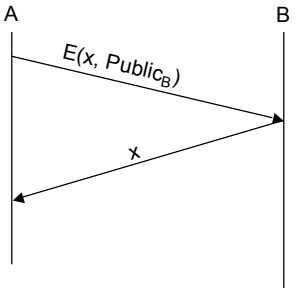
Properties of RSA

- Requires generating large, random prime numbers
  - Algorithms exist for quickly finding these (probabilistic!)
- Requires exponentiating very large numbers
  - Again, fairly fast algorithms exist
- Overall, much slower than symmetric key crypto
  - One general strategy: use public key crypto to exchange a (short) symmetric session key
    - Use that key then with AES or such
- How difficult is recovering d, the private key?
  - Recall $d \equiv e^{-1} \bmod ((p-1)\bullet(q-1))$
  - To find d given n and e, need to **factor** n into p and q
    - Many have tried - believed to be **very hard** (= brute force only)
    - (Though *quantum computers* can do so in polynomial time!)

26

Public Key Authentication

- Each side need only to know the other side's public key
  - No secret key need be shared
- **A** encrypts a nonce (random number) **x**
- **B** proves it can recover **x**
- **A** can authenticate itself to **B** in the same way



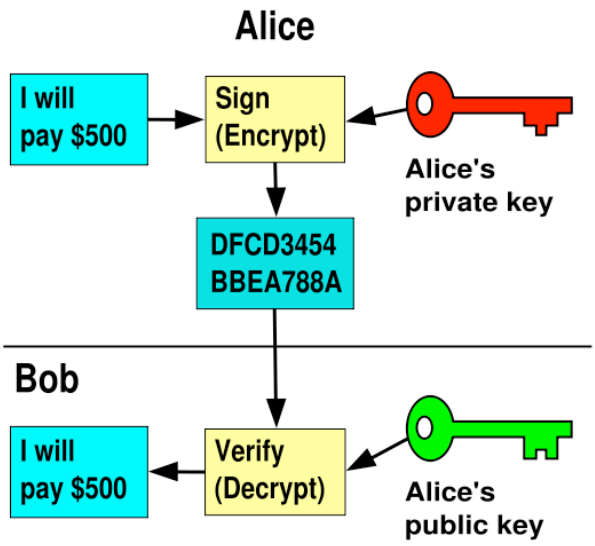
27

RSA Crypto & Signatures

- Suppose Alice has published public key K_E
- If she wishes to prove who she is, she can send a message x encrypted with her **private** key K_D (i.e., she sends $D(x,K_D)$)
  - Recall: $E(x,K_E)$ and $D(x,K_D)$ are inverses
  - Therefore: anyone w/ public key K_E can recover x, verify that Alice must have sent the message
    - It provides a signature
  - Alice can't deny it ⇒ non-repudiation

28

RSA Crypto & Signatures, con't



29

RSA Crypto & Signatures

- Suppose Alice has published public key K_E
- If she wishes to prove who she is, she can send a message x encrypted with her **private** key K_D (i.e., she sends $D(x,K_D)$)
  - Recall: $E(x,K_E)$ and $D(x,K_D)$ are inverses
  - Therefore: anyone w/ public key K_E can recover x, verify that Alice must have sent the message
  - And: Alice can't deny it ⇒ non-repudiation
- In practice, for efficiency Alice signs a digest (e.g., SHA-1) of the message rather than the whole thing (oops - not great if SHA-1 can be broken!)

30

Summary of Our Crypto Toolkit

- **If** we can securely distribute a key, then
  - Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality
- Public key cryptography does away with (potentially major) problem of secure key distribution
  - But: not as computationally efficient
    - Often addressed by using public key crypto to exchange a session key
  - And: also not guaranteed secure (but **major** result if not)

31

Summary of Our Crypto Toolkit, con't

- Cryptographically strong hash functions provide major building block for integrity (e.g., SHA-1)
  - As well as providing concise digests
  - And providing a way to prove you know something (e.g., passwords) without revealing it (non-invertibility)
  - But: worrisome recent results regarding their strength
- Public key also gives us signatures
  - Including sender non-repudiation
- Turns out there's a crypto trick based on similar algorithms that allows two parties *who don't know each other's public key* to securely negotiate a secret key even in the presence of eavesdroppers

32

Types of Attacks on Crypto Systems

- Guess the key
  - From knowledge about the user picking it
  - By trying every entry in a dictionary
  - By figuring out the algorithm the user used to generate it
    - E.g., if they use a predictable pseudo-random number generator
- Brute-force the key
  - Try every possible key
  - Perhaps exploiting extra info to rule out some
- Steal the key
  - It has to be stored somewhere for system to use it
  - Or: perhaps it's reused in another context
  - Or: "rubber hose cryptanalysis", i.e., force user to divulge

33

Attacks on Crypto Systems, con't

- Deduce the key
  - Known plaintext attack: if Eve sees crypto output and knows what the input is, can she solve for the key?
    - Requires an attacker who can snoop
  - Chosen plaintext attack: if Eve can select what gets encrypted and sees the crypto output, can she solve for the key?
    - Requires an attacker who can inject

34

Attacks on Crypto Systems, con't

- Replay
  - Eve records a previously transmitted encrypted message, sends again at later, judicious time …
    - o …. even though she can't directly read it
  - How can this benefit the attacker?
    - o E.g., Eve replays Alice's non-repudiable "**I will pay $500**"
    - o E.g., Eve replays Alice's "**my password is *icecream***" while Eve claims to be Alice
  - Defenses:
    - o Both parties exchange a token unique to each dialog
      - Replay will have the old token in it
    - o Include timestamps (must be careful to synch. clocks)
    - o Include context (e.g., "**I will pay *Bob* $500**")

35

Attacks on Crypto Systems, con't

- Man-in-the-middle
  - Suppose Bob doesn't know Alice's public key in advance
  - Alice sends her public key to Bob, but Eve intercepts
    - o Because Eve is on the path between the two, and can alter messages
  - Eve sends her own public key to Bob, claiming it's Alice's
    - o Eve likewise sends her key to Alice, claiming it's Bob's
  - Suppose Alice now sends a session key to Bob encrypted with "Bob's" public key (really, Eve's)
  - Eve recovers the key, repackages it for Bob using Bob's actual public key …
    - o … and reads (and/or modifies) entire subsequent communication
- This is why SSH will tell you:

```
The authenticity of host XYZ can't be established.
Are you sure you want to continue connecting?
```

36

Attacks on Crypto Systems, con't

- Side-channel attacks: deduce key from an incidental property of the system's implementation
- E.g.: OpenSSL used modulo-exponentiation operation for which execution time changes as input (crypto text) nears multiple of p or q
  - So feed an **`HTTPS`** server (many) different crypto texts …
  - … and use reply time variation to drive search for p & q
  - **It Works**: private key extracted over LAN in a few hours
  - Fix (for this particular attack): break dependence of CPU execution time on details of input
- Even wilder (demonstrated) side-channel attacks:
  - What someone's typing from **sound** keyboard makes
  - What's on a CRT based on brightness of screen's **flicker**

37

Summary

- Secure communication has many requirements
  - Authentication, authorization, …
- Workhorse for many of these: cryptography
  - Symmetric encryption: fast, but requires shared secret
  - Public key encryption: no need for shared secret
- Hash functions provide integrity and signatures
- There are a range of attacks on cryptosystems
  - However, crypto is in fact our most mature security technology
- Next lecture: attacks & defenses

38