eecs489 COMPUTER NETWORKS

Lecture 14:
Network Security and
Cryptographic Algorithms

Security and the Internet

Original key design goals of Internet protocols:
- resiliency
- availability
- scalability

Security has not been a priority until mid 1990s

Designed for simplicity: "on"-by-default

Unfortunately, readily available zombie machines

Attacks look like normal traffic

Internet's federated operation obstructs cooperation for diagnosis/mitigation

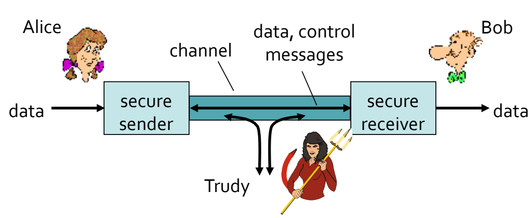
[after Rexford]

Security Attacks

Cast of characters: Alice, Bob, and Trudy, three well-known characters in network security world

Bob and Alice want to communicate "securely"

Trudy (intruder) may intercept, delete, and/or add messages



Security Requirements

Attack against content:

Data integrity: sender and receiver(s) want to ensure that data is not altered (in transit, or afterwards) or if altered, detectable

Confidentiality/secrecy: only parties involved, the sender and the intended receiver(s) should know of (the content of) the transaction

Security Requirements

Attack against content:

Authentication: sender, receiver(s) want to confirm each other's identity

- compare: authorization (what's the difference between authentication and authorization?)

Non-repudiation: involved parties cannot deny participation afterwards

Security Requirements

Attack against infrastructure:

Access and availability: services must be accessible and available to (authorized) users

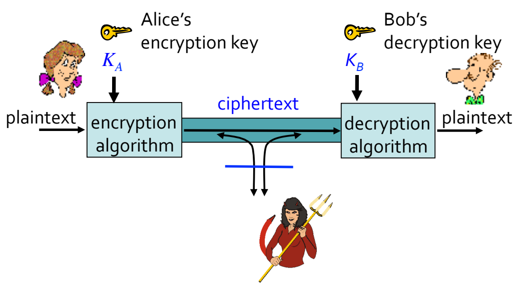
- destroy hardware (cutting fiber) or software
- modify software in a subtle way
- corrupt packets in transit
- denial of service (DoS)attack:
  - crashing the server
  - overwhelm the server (use up its resource)

Countermeasure: Cryptography

The fundamental tool for achieving network security

Origin: Greek word for "secret"

Cryptographers invent secret codes (cipher) to try to hide messages from unauthorized observers



Two Types of Encryption Algorithms

Symmetric key cryptography:

- both parties share a secret key that is used for both encryption and decryption

Public-key cryptography:

- asymmetric cryptography: involves use of two keys: a public key and a private (secret) key, data encrypted with the public key can be decrypted by the private key and vice versa

Kerckhoff's Principle: "The security of a cryptosystem must not depend on keeping secret the crypto-algorithm. The security depends only on keeping secret the key."

– *La cryptographie militaire* (1883)

Symmetric-key Cryptography

Both parties share a secret key that is used for both encryption and decryption

Assumes encryption algorithm is known to both parties

Implies a secure channel to distribute key

Was the only type of encryption prior to the invention of public-key cryptography in 1970's

Typically more computationally efficient, often used in conjunction with public-key cryptography

Example system: Kerberos Authentication Service

Key Escrow

Symmetric key cryptography requires participants to know shared secret key

Q: how to agree on shared key in the first place (particularly if the participants never "met")?

Shared key can be distributed by key escrow or key distribution center (KDC):

- escrow shares secret keys with both parties
- generates a session key for each session between the two parties
- $K_{A\text{-}KDC}(K_{A\text{-}B}, K_{B\text{-}KDC}(A, K_{A\text{-}B}))$ sent to Alice to be passed to Bob

Authentication

Fundamental trade-off: security vs. convenience

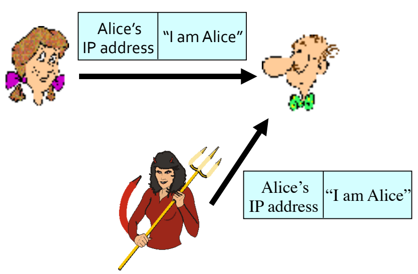
Most secure, least convenient: not networked, placed in a secure locked room

Two options in access control:

1. challenge the users each time they want to use a service
2. authenticate them once and grant them tickets to use several services without further (user-level) challenge for a duration of time (Kerberos)

Authentication: IP Spoofing

Bob wants Alice to "prove" her identity to him

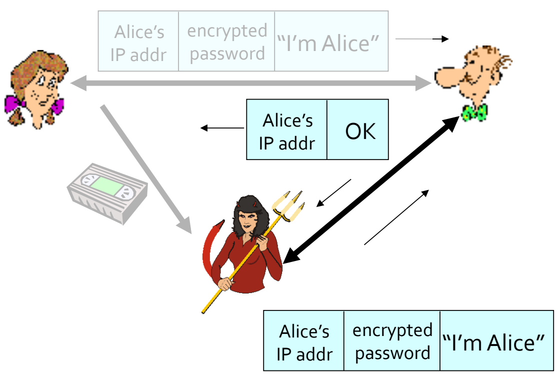


Threat model:

Trudy can create a packet, "spoofing" Alice's address

Authentication: Playback Attack

Alice says "I am Alice" and sends her encrypted secret password to "prove" it

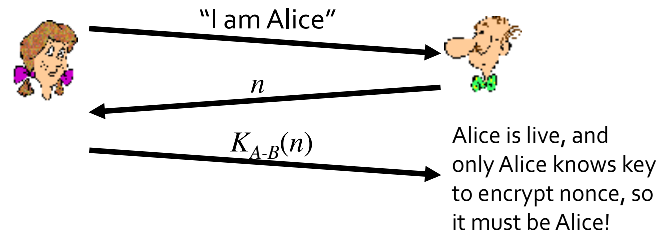


Threat model:

Playback attack: Trudy records Alice's packet and later spoofs Alice's IP address and plays back the recorded packet to Bob

Authentication: Use of Nonce

Nonce used to avoid playback attack
Nonce: a number (n) used only once-in-a-lifetime
To prove Alice "live", Bob sends Alice nonce, n
Alice must return n, encrypted with shared secret key



Kerberos: an Authentication Service

Kerberos generates a shared symmetric key for each user-service pair

- key is valid for only a limited period of time

Three parties:

1. Authentication Server (AS)
2. Principal: party whose ID is to be verified, usually a client application (c)
3. Verifier: party requesting verification, typically servers (v) for various services, e.g., name server, file server, print server, etc.

Kerberos Authentication Protocol

Authentication Server (AS):

1. keeps a list of all clients' passwords (K_c's)
2. shares a key with each service (K_v)

Client (c):

1. asks AS for a session key for a specific server (v) for a period of time, provides nonce (n)
2. gets back (a) a session key ($K_{c,v}$) with expiration time, and nonce, encrypted with client's password (K_c) and (b) a ticket ($T_{c,v}$) for server v, encrypted using server's key (K_v), $T_{c,v} = K_v(K_{c,v}, c, \text{time}_{\text{exp}}, \ldots)$
3. sends data (encrypted with session key), along with ticket and authenticator (a timestamp/nonce and an optional sub-session key, encrypted using session key)

Kerberos Authentication Protocol

Server (v):

1. decrypts and "unpacks" $T_{c,v}$ to obtain $K_{c,v}$, makes sure it belongs to c and time hasn't expired
2. decrypts authenticator ($K_{c,v}(t_s, K_{\text{subsession}})$), checks that nonce, t_s, is within window (5 min) and has not been used
3. decrypts data using $K_{\text{subsession}}$ (optional)
4. responds with $\{t_s\}K_{c,v}$ (optional)

Kerberos Authentication Protocol

Inconvenience:

- each service requires a separate ticket
- client prompts user for password for each ticket

More convenient: use a ticket-granting service with TGS ticket that lives for a "short" period of time (8 hours)

Kerberos still relies on password, which could be "spoofed"

One-time Passcode

Protection against password spoofing

- generates a random number as passcode
- each passcode is good for 4 minutes
- login challenge comprises user's password plus the random number

RSA SecurID tokens
(has a built-in accurate clock)

Public-key Cryptography

Symmetric key cryptography requires participants to know a shared secret key

Two "key" issues:

- key distribution: how to secure communication if you won't trust a key distribution center with your key?
- digital signatures: how to verify message arrives intact from claimed sender (w/o prior authentication)

[after Rexford]

Public-key Cryptography

A radically different approach [Diffie-Hellman76, RSA78]

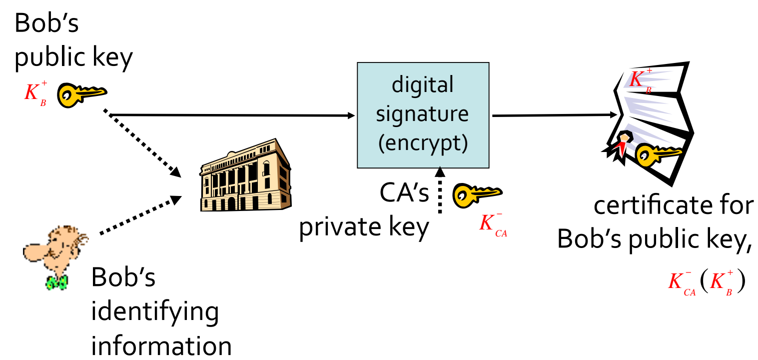
- known earlier in classified community
- example algorithm: RSA (Rivest, Shamir, Adelson)

Sender and receiver do not share a secret key

- public key (K^{+}) known to all
- private key (K^{-}) known only to owner
- given public key, it should be impossible to compute private key
- ciphertext encrypted using the public key can be decrypted using the private key $K^{-}(K^{+}(M)) = M$, used for message integrity, secrecy
- data encrypted with private key, can be decrypted with public key $K^{+}(K^{-}(M)) = M$, used for digital signature, sender verification, non-repudiation

How to Obtain Public Key?

Certificates and Certification Authorities (CAs)

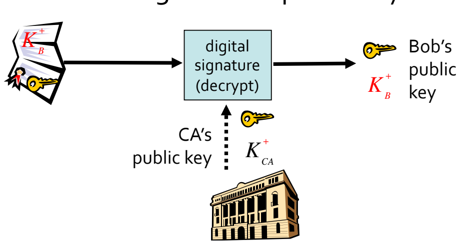


CA is in effect asserting that "this is Bob's public key"

Public Key Distribution

When Alice wants Bob's public key:

- Alice obtains CA's public key in an offline, secure manner (comes with browser code download, how secure is that?)
- Alice gets Bob's certificate (from Bob or from elsewhere, doesn't have to be secure channel, why not?)
- Alice decrypts Bob's certificate using the CA's public key to get Bob's public key



Certificate Revocation

CA periodically publishes a Certification Revocation List (CRL) for revoked public-keys

- not currently done

How to revoke CA's public key?

- currently as part of browser updates

Public Key Infrastructure (PKI)

A hierarchy of CAs

Relies on a chain of trust (speak-for relationship)

Examples:

- Verisign, Entrust, thawte, Symantec, GlobalSign, Visa, DigiCert, etc.
- see Chrome→Settings→Advanced Settings→HTTPS/SSL→Manage certificates… →System Roots

Performance of Public-key Schemes

Like symmetric key schemes brute force exhaustive search attack is theoretically possible

- but keys used are so large (e.g., ≥ 1024bits) as to be impractical to crack
- the requirement to use very large numbers makes public-key cryptography slow compared to symmetric key schemes

For example [`cryptopp.com/becnhmarks.html`]: on a 1.83 GHz Intel Core 2 running 32-bit Windows Vista,

- symmetric AES 128-bit key performs at 109 MB/s (1.2 μs/Mbits)
- RSA 1024-bit key encrypt speed: 1.56 MB/s (80 μs/Mbits)
- RSA 1024-bit key decrypt speed: 85.6 KB/s ($1{,}460$ μs/Mbits)
- RSA decryption is 12-19 times slower than encryption, depending on key size

[after Rexford]

Security of Public-key Schemes

Symmetric keys are also more resistant to brute-force attacks

Common practice, due to message size and algorithm performance: use public-key to distribute symmetric session key

- generate random symmetric key r
- use public key encryption to encrypt and distribute r
- use symmetric key encryption under r to encrypt message M

Symmetric- and Public-key Key Lengths with Similar Resistances to Brute-Force Attacks [Schneier]

symmetric	public
56 bits	384 bits
64 bits	512 bits
80 bits	768 bits
112 bits	1,792 bits
128 bits	2,304 bits

[after Rexford]

Digital Signatures

Cryptographic technique analogous to hand-written signatures

Sender (Bob) digitally signs document by encrypting the document using his private key, establishing he is the document owner/creator

Verifiable, non-forgeable: recipient (Alice) can prove to anyone that only Bob, and no one else (including Alice), could have signed the document

Non-repudiation: Alice can take message M, and signature $K_B^-(M)$ to court and proves that Bob signed M

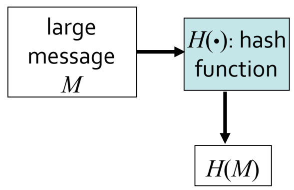
Message Digest

But it is computationally expensive to encrypt long messages with public-key cryptography

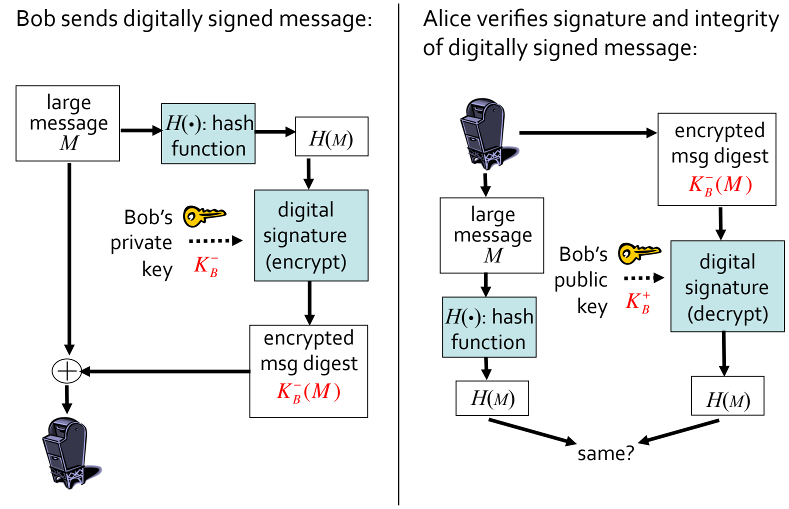
For purposes of authentication and certification, it is sufficient to encrypt a digest of the original message

Want: a fixed-length, easy-to-compute digital "fingerprint" or digest to uniquely represent a message

Solution: apply a one-way hash function $H(\cdot)$ to M to get a fixed-size message digest, $H(M)$



Digital Signature = Signed Message Digest



Hash Function Criteria

Required criteria of the hash function:

1. many-to-1 "compression", but 1-1 mapping
2. produces fixed-size message digest (fingerprint), fast
3. "one-way": given message digest h, computationally infeasible to find M such that $h = H(M)$

Checksum would not be a good one-way hash function:

message	ASCII format	message	ASCII format
I O U 1	49 4F 55 31	I O U 9	49 4F 55 39
0 0 . 9	30 30 2E 39	0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42	9 B O B	39 42 D2 42
	B2 C1 D2 AC		B2 C1 D2 AC

different messages but identical checksums!

Birthday Paradox

What is the smallest number of people in a room for a better-than-even odds (probability ≥ 0.5) that two persons share the same birthday?

Assumptions:

- 366 days to a year
- birthdays are independent (no twins)
- birthdays are uniformly distributed (equally likely, in reality, more likely 9 months after a holiday)

Birthday Paradox

Probability that each person in the room has a birthday different from all the other persons in the room:

Probability for the 1^{st} person: 1

Probability for the 2^{nd} person: $\frac{366-1}{366}$

Probability for the 3^{rd} person: $\frac{366-2}{366}$

…

Probability for the j-th person: $\frac{366-(j-1)}{366} = \frac{367-j}{366}$

Birthday Paradox

Assuming independence, the probability that all k people in the room have different birthdays is:

$$p_k = 1 \cdot \frac{365}{366} \cdot \frac{364}{366} \cdot \ldots \cdot \frac{367-k}{366}$$

The probability that not "all k people in the room have different birthdays", i.e., at least 2 out of the k persons have the same birthday is: $\varepsilon = 1 - p_k$

Birthday Paradox

$\varepsilon = 1 - p_k$

By brute force calculations, we find that:
for $k = 22$, $\varepsilon \approx 0.475$, for $k = 23$, $\varepsilon \approx 0.506$

So you only need 23 people in a room for 2 persons to share the same birthday!

More generally, $k \approx \sqrt{2M \log \frac{1}{1-\varepsilon}}$

for $\varepsilon = 0.5$, $k \approx 1.17\sqrt{M}$

For the birthday paradox, $M = 366$

Hashing Collision

How many items (k) does it take to hash two items into the same bucket, with probability ≥ 0.5, for a table of size M?

Assuming:
- items are independent
- all possible items are equally likely
  (clearly not true for English words, for example)
- hash function hashes uniformly

For:
$M = 7, k = 4$
$M = 9, k = 4$
$M = 11, k = 4$
$M = 2^{40}, k = 1\ 226\ 834$
$M = 2^n$, it takes on the order of $\sqrt{M}$ or $2^{n/2}$

For SHA-1, $n = 160$, it takes 2^{80} items to have a collision with probability ≥ 0.5

Example Hash Function Algorithms

MD5 (Message Digest):

- MD4 developed by Rivest (the 'R' in RSA) in 1990, MD5 in 1992
- computes 128-bit message digest in 4-step process
- collision found in 2^{18} calculations (< 1 sec) in 2013
- cryptographically broken

RIPEMD-160

- developed by a team of European researchers at RIPE
- produces 160-bit hash
- less popular, less scrutinized

Speed: MD5 > RIPEMD-160 > SHA-1

Example Hash Function Algorithms

SHA-1 (Secure Hash Algorithm):

- SHA developed by the NSA (1993), revised SHA-1 in 1995
- produces 160-bit message digest
- collisions in SHA-1 can be found in 2^{69} (not 2^{80}) calculations

SHA-2 family: SHA-256, SHA-512, and truncated versions (2001)

- SHA-256 and SHA-512 are structurally identical, differ only in rounds (but different from SHA-1)
- produce 256- and 512-bit digests respectively
- successful attacks on reduced round, none extends to full round of the hash functions

Example Hash Function Algorithms

SHA-3:

- winner of the NIST hash function competition 2012
- not to replace SHA-2, but as an alternative, dissimilar cryptographic hash
- standardized by NIST on Aug. 5th, 2015
- SHA3-224, SHA3-256, SHA3-384, SHA3-512 are the drop-in replacements for SHA2, with identical security claims

Other Uses of One-Way Hash

Password hashing:

- can't store passwords in a file that could be read
- how to compare input passwords to stored passwords?
  - solution: hash(input) == hash(stored) ?
- often "salt" is used: hash(input||salt)
  - known as hash message authentication code (HMAC)
- can also be used to generate a different password for each web account from one password
- don't use MD5 or SHA-1

Integrity of downloaded file:

- file tagged with hash(data)
- users verify that hash(downloaded) == hash(data)

[after Rexford]