Certificates,
Authentication & Identity,
Design Principles
Network Security

Lecture 8

October 23, 2003

Project

- Survey type paper
  - Comparative/tradeoff studies
  - Current trends, challenges, possible approaches
  - At most two people
  - Number of references should be large
- Implementation
  - Reasonable sophistication
  - Up to 3 people
- New research ??
- Others: Case studies??

Project Topics (not limited to these only!)

- XML and security
- Security policies
- RBAC
- Cryptographic protocols
- Database security
- Ad hoc network security
- Cyber Security
- Privacy
- Java security
- Intrusion detection schemes
- Auditing
- Security and ethics
- Smartcards and standards for smartcards
- Security standards
- E-commerce security

Project Schedule

- Proposal (by Nov 15)
  - Up to 2 pages (identify a group)
  - State the goals
  - State the significance
- Final project report
  - By the last day of the semester
  - Article format, or conference format
    - Each person should state his contribution
  - Implementation projects should demonstrate to TA and/or me

Cryptographic Key Infrastructure

- Goal: bind identity to key
- Classical Crypto:
  - Not possible as all keys are shared
- Public key Crypto:
  - Bind identity to public key
  - Crucial as people will use key to communicate with principal whose identity is bound to key
  - Erroneous binding means no secrecy between principals
  - Assume principal identified by an acceptable name

Certificates

- Create token (message) containing
  - Identity of principal (here, Alice)
  - Corresponding public key
  - Timestamp (when issued)
  - Other information (perhaps identity of signer)

  signed by trusted authority (here, Cathy)

$$C_A = \{ e_A \mid\mid \text{Alice} \mid\mid T \} \, d_C$$

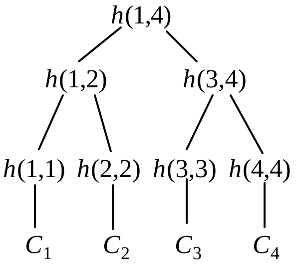
C_A is A's certificate

Use

- Bob gets Alice's certificate
  - If he knows Cathy's public key, he can decipher the certificate
    - When was certificate issued?
    - Is the principal Alice?
  - Now Bob has Alice's public key
- Problem: Bob needs Cathy's public key to validate certificate
  - Problem pushed "up" a level
  - Two approaches: Merkle's tree, signature chains

Merkle's Tree Scheme

- Keep certificates in a file
  - Changing any certificate changes the file
  - Use crypto hash functions to detect this (data integrity)
- Define hashes recursively
  - *h* is hash function
  - C_i is certificate *i*
- Hash of file (*h*(1,4) in example) known to all

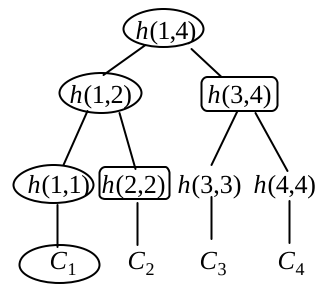


Details

- f: $D \times D$ ® D maps bit strings to bit strings
- h: $N \times N$ ® D maps integers to bit strings
  - if $i = j$, $h(i, j) = f(C_i, C_j)$
  - if $i < j$,

    $h(i, j) = f(h(i, \lfloor (i+j)/2 \rfloor), h(\lfloor (i+j)/2 \rfloor + 1, j))$

Validation



- To validate C_1:
  - Compute $h(1, 1)$
  - Obtain $h(2, 2)$
  - Compute $h(1, 2)$
  - Obtain $h(3, 4)$
  - Compute $h(1,4)$
  - Compare to known $h(1, 4)$
- Need to know hashes of children of nodes on path that are not computed

Problem

- File must be available for validation
  - Otherwise, can't recompute hash at root of tree
  - Intermediate hashes would do
- Not practical in most circumstances
  - Too many certificates and users
  - Users and certificates distributed over widely separated systems

Certificate Signature Chains

- Create certificate
  - Generate hash of certificate
  - Encipher hash with issuer's private key
- Validate
  - Obtain issuer's public key
  - Decipher enciphered hash
  - Recompute hash from certificate and compare
- Problem:
  - Validating the certificate of the issuer and getting issuer's public key

X.509 Chains

- Key certificate fields in X.509v3:
  - Version
  - Serial number (unique)
  - Signature algorithm identifier: hash algorithm
  - Issuer's name; uniquely identifies issuer
  - Interval of validity
  - Subject's name; uniquely identifies subject
  - Subject's public key
  - Signature:
    - Identifies algorithm used to sign the certificate
    - Signature (enciphered hash)

X.509 Certificate Validation

- Obtain issuer's public key
  - The one for the particular signature algorithm
- Decipher signature
  - Gives hash of certificate
- Recompute hash from certificate and compare
  - If they differ, there's a problem
- Check interval of validity
  - This confirms that certificate is current

Issuers

- *Certification Authority (CA)*: entity that issues certificates
  - Multiple issuers pose validation problem
  - Alice's CA is Cathy; Bob's CA is Don; how can Alice validate Bob's certificate?
  - Have Cathy and Don cross-certify
    - Each issues certificate for the other

Validation and Cross-Certifying

- Certificates:
  - Cathy<<Alice>>
    - represents the certificate that C has generated for A
  - Dan<<Bob>
  - Cathy<<Dan>>
  - Dan<<Cathy>>
- Alice validates Bob's certificate
  - Alice obtains Cathy<<Dan>>
  - Alice uses (known) public key of Cathy to validate Cathy<<Dan>>
  - Alice uses Cathy<<Dan>> to validate Dan<<Bob>>
    - Cathy<<Dan>> Dan<<Bob>> is a signature chain
  - How about Bob validating Alice?

PGP Chains

- Pretty Good Privacy:
  - Widely used to provide privacy for electronic mail
  - Sign files digitally
- OpenPGP certificates structured into packets
  - One public key packet
  - Zero or more signature packets
- Public key packet:
  - Version (3 or 4; 3 compatible with all versions of PGP, 4 not compatible with older versions of PGP)
  - Creation time
  - Validity period (not present in version 3)
  - Public key algorithm, associated parameters
  - Public key

OpenPGP Signature Packet

- Version 3 signature packet
  - Version (3)
  - Signature type (level of trust)
  - Creation time (when next fields hashed)
  - Signer's key identifier (identifies key to encipher hash)
  - Public key algorithm (used to encipher hash)
  - Hash algorithm
  - Part of signed hash (used for quick check)
  - Signature (enciphered hash using signer's private key)
- Version 4 packet more complex

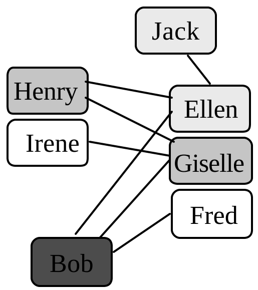
Signing

- Single certificate may have multiple signatures
- Notion of “trust” embedded in each signature
  - Range from “untrusted” to “ultimate trust”
  - Signer defines meaning of trust level (no standards!)
- All version 4 keys signed by subject
  - Called “self-signing”

Validating Certificates

- Alice needs to validate Bob’s OpenPGP cert
  - Does not know Fred, Giselle, or Ellen
- Alice gets Giselle’s cert
  - Knows Henry slightly, but his signature is at “casual” level of trust
- Alice gets Ellen’s cert
  - Knows Jack, so uses his cert to validate Ellen’s, then hers to validate Bob’s

Arrows show signatures
Self signatures not shown



Stream and Block Cipher

- Block cipher (E_k is encryption)
  - $m = b_1b_2\ldots$, where b_i *is of a fixed length*
  - $E_k(m) = E_k(b_1)E_k(b_2)\ldots$
  - DES is a block cipher (64 bit blocks)
- Stream cipher (E_k is encryption)
  - $m = b_1b_2\ldots$, where b_i *is of a fixed length*
  - $k = k_1k_2\ldots$
  - $E_k(m) = E_{k1}(b_1)E_{k2}(b_2)\ldots$
  - Vinegere cipher, one-time pad

Authentication and Identity

What is Authentication?

- Authentication:
  - Binding of identity to subject
- How do we do it?
  - Entity *knows* something (secret)
    - Passwords, id numbers
  - Entity *has* something
    - Badge, smart card
  - Entity *is* something
    - Biometrics: fingerprints or retinal characteristics
  - Entity is in *someplace*
    - Source IP, restricted area terminal

Authentication System: Formal Definition

- *A*: Set of *authentication information*
  - used by entities to prove their identities (e.g., password)
- *C*: Set of *complementary information*
  - used by system to validate authentication information (e.g., hash or a password or the password itself)
- *F*: Set of *complementation functions* (to generate *C)*
  - $f : A ? C$
  - Generate appropriate $c \in C$ given $a \in A$
- *L*: set of *authentication functions*
  - $l: A \times C ?$ { **true, false** }
  - verify identity
- *S*: set of *selection functions*
  - Generate/alter *A* and *C*
  - e.g., commands to change password

Authentication System: Passwords

- Example: plaintext passwords
  - *A* = *C* = alphabet*
  - *f* returns argument: *f*(*a*) returns *a*
  - *I* is string equivalence: *I*(*a*, *b*) is true if *a* = *b*
- Complementation Function
  - Null (return the argument as above)
    - requires that *c* be protected; i.e. password file needs to be protected
  - One-way hash – function such that
    - *Complementary information* $c = f(a)$ easy to compute
    - $f^{-1}(c)$ difficult to compute

Passwords

- Example: Original Unix
  - A password is up to eight characters each character could be one of 127 possible characters;
  - *A* contains approx. 6.9 x 10^{16} passwords
  - Password is hashed using one of 4096 functions into a 11 character string
  - 2 characters pre-pended to indicate the hash function used
  - *C* contains passwords of size 13 characters, each character from an alphabet of 64 characters
    - Approximately 3.0 x 10^{23} strings
  - Stored in file */etc/passwd* (all can read)

Authentication System

- Goal of (*A*, *C*, *F*, *L*, *S*)
  - For all $a \in A$, $c \neq f(a) \in C$
    - $\exists$ (*f*, *l*), $f \in F$, $\forall\, l \in L$ in the system such that
      - *l*(*a*, *f*(*a*)) ? **true**
      - *l*(*a*, *c*) ? **false** (with high probability)
- Approaches
  - Hide enough information so that one of *a*, *c* or *f* cannot be found
    - Make C readable only to root (*use shadow password files*)
    - Make F unknown
  - Prevent access to the authentication functions *L*
    - *root* cannot log in over the network (*L* exist but fails)

Attacks on Passwords

- Dictionary attack: Trial and error guessing
  - Type 1: attacker knows *A*, *f*, *c*
    - Guess *g* and compute *f*(*g*) for each *f* in *F*
  - Type 2: attacker knows *A*, *l*
    - *l* returns **True** for guess *g*
  - Difficulty based on |*A*|, Time
    - Probability *P* of breaking in time *T*
    - *G* be the number of guesses that can be tested in one time unit
    - $P = TG/|A|$
    - Assumptions: time constant; all passwords are equally likely

Password Selection

- Random
  - Depends on the quality of random number generator; size of legal passwords
  - 8 characters: humans can remember only one
  - Will need to write somewhere
- Pronounceable nonsense
  - Based on unit of sound (phoneme)
    - "Helgoret" vs "pxnftr"
  - Easier to remember
- User selection (proactive selection)
  - Controls on allowable
  - Reasonably good:
    - At least 1 digit, 1 letter, 1 punctuation, 1 control character
    - Obscure poem verse

Password Selection

- Reusable Passwords susceptible to dictionary attack (type 1)
  - *Salting* can be used to increase effort needed
    - makes the choice of complementation function a function of randomly selected data
    - Random data is different for different user
    - Authentication function is chosen on the basis of the salt
    - Many Unix systems:
      - A salt is randomly chosen from 0..4095
      - Complementation function depends on the salt

Password Selection

- Password aging
  - Change password after some time: based on expected time to guess a password
  - Disallow change to previous *n* passwords
- Fundamental problem is *reusability*
  - Replay attack is easy
  - Solution:
    - Authenticate in such a way that the transmitted password changes each time

Authentication Systems: Challenge-Response

- Pass algorithm
  - authenticator sends message *m*
  - subject responds with *f*(*m*)
    - *f* is a secret encryption function
    - In practice: key known only to subject
  - Example: ask for second input based on some algorithm

Authentication Systems: Challenge-Response

- One-time password: *invalidated after use*
  - *f* changes after use
    - Challenge is the number of authentication attempt
    - Response is the one-time password
- S/Key uses a hash function (MD4/MD5)
  - User chooses an initial seed *k*
  - Key generator calculates
    - $h(k) = k_1, h(k_1) = k_2 \ldots, h(k_{n-1}) = k_n$
  - Passwords used in the order
    - $p_1 = k_n, p_2 = k_{n-1}, \ldots, p_n = k_1$
  - Suppose $p_1 = k_n$ is intercepted;
    - the next password is $p_2 = k_{n-1}$
    - Since $h(k_{n-1}) = k_n$, the attacker needs to know *h* to determine the next password

Authentication Systems: Biometrics

- Used for human subject identification based on physical characteristics that are tough to copy
  - Fingerprint (optical scanning)
    - Camera's needed (bulky)
  - Voice
    - Speaker-verification (identity) or speaker-recognition (info content)
  - Iris/retina patterns (unique for each person)
    - Laser beaming is intrusive
  - Face recognition
    - Facial features can make this difficult
  - Keystroke interval/timing/pressure

Attacks on Biometrics

- Fake biometrics
  - fingerprint “mask”
  - copy keystroke pattern
- Fake the interaction between device and system
  - Replay attack
  - Requires careful design of entire authentication system

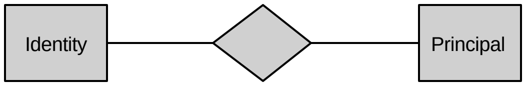
Authentication Systems: Location

- Based on knowing physical location of subject
- Example: Secured area
  - Assumes separate authentication for subject to enter area
  - In practice: early implementation of challenge/response and biometrics
- What about generalizing this?
  - Assume subject allowed access from limited geographic area
    - I can work from (near) home
  - Issue GPS Smart-Card
  - Authentication tests if smart-card generated signature within spatio/temporal constraints
  - Key: authorized locations known/approved in advance

Authentication vs. Identity

- Principal: Unique Entity
  - Subject
  - Object
- Identity: Specifies a principal
  - Used for accountability
  - Used for access control
- Authentication
  - Binds a principal to a representation of identity internal to the system

Identity = Principal?



- Identity to Principal may be many-to-one
  - Given identity, know principal
  - Other direction unimportant?
- Examples: Unix
  - User identity
  - File identity

Users, Groups, Roles

- Files/Objects
  - Identity depends on the system
  - Names may be used for human use (file names) or file descriptors/handle (process use) etc.
- User
  - An identity tied to a single entity
  - Unix: UID is an integer – identifies a user (0 is root)
- Entity may also be a set of entities referred to a single identity
  - Examples:
    - Groups: defined collection of users with common privileges
    - Roles: membership tied to function

Representing Identity

- Randomly chosen: not useful to humans
- User-chosen: probably not unique
  - At least globally
- Hierarchical: Disambiguate based on levels
  - File systems
  - X.503v3 certificates use identifiers called Distinguished Names
    - /O=University of Pittsburgh/OU=Information and Telecommunications/CN=Alice

Validating Identity

- Authentication: Does subject match purported identity?
- Problem: Does identity match principal?
- Solution: *certificates*
  - Certificate: Identity validated to belong to known principal
  - Certification Authority: Certificate Issuer
    - Authentication Policy: describes authentication required to ensure principal correct
    - Issuance policy: Who certificates will be issued to
  - CA is *trusted*

Certificate Implementation

- Is a certificate real?
  - Digital signatures
  - Certificate = Identity + $E_{IssuerPrivateKey}$(Identity)
    - Correct if Identity = $D_{IssuerPublicKey}$(Signature)
- Can I trust it?
  - Hierarchy of issuers
    - Certificate includes certificate of issuer chain
  - Higher levels place (contractual) conditions on lower level issuance
    - Common issuance, authentication policy

Certificate Examples

- Verisign
  - Independently verifies identity of principal
  - Levels of certification
    - Email address verified (Class 1 CA)
    - Name/address verified (Class 2 CA)
    - Legal identity verified (Class 3 CA)
  - More common: *corporate* identity
    - Is this really PayTuition.EDU I'm giving my bank account number to?
- PGP (Pretty Good Privacy): "Web of Trust"
  - Users verify/sign certificates of other users
  - Do I trust the signer?
    - *Or someone who signed their certificate?*

Internet Identity

- Host Identity: Who is this on the network?
- Ethernet: MAC address
  - Guarantees uniqueness
- IP address: aaa.bbb.ccc.ddd
  - Provides hierarchy to ease location
- Issues: Spoofing
  - Attacker spoofs the identity of another host
  - All protocol that rely on that identity are being spoofed

Domain Name Service

- Associates host names with IP addresses
- Forward records
  - Map host names into IP addresses
- Reverse records
  - Map IP addresses into host names
- DNS attacks alter the association of host name and an IP address

Anonymity

- What if identity not needed?
  - Web browsing
  - Complaints through emails
- Removing identity not as easy as it sounds
  - I can send email without my userid
  - But it still traces back to my machine
- Solution: anonymizer
  - Strips identity from message
  - Replaces with (generated) id
  - Send to original destination
  - Response: map generated id back to original identity

Anonymity

- Problem: Anonymizer knows identity
  - Can it be trusted?
  - *Courts say no!*
- Solution: multiple anonymizers
  - Need to attack each node in the chain
- Anonymity also protects privacy
  - Against user profiling
- Various social uses

  (read 14.6.3.1)

Design Principles

Design Principles for Security Mechanisms

- Principles
  - Least Privilege
  - Fail-Safe Defaults
  - Economy of Mechanism
  - Complete Mediation
  - Open Design
  - Separation of Privilege
  - Least Common Mechanism
  - Psychological Acceptability
- Based on the idea of *simplicity* and *restriction*

Overview

- Simplicity
  - Less to go wrong
  - Fewer possible inconsistencies
  - Easy to understand
- Restriction
  - Minimize access power (need to know)
  - Inhibit communication

Least Privilege

- A subject should be given only those privileges necessary to complete its task
  - Function, not identity, controls
    - RBAC!
  - Rights added as needed, discarded after use
    - Active sessions and dynamic separation of duty
  - Minimal protection domain
    - A subject should not have a right if the task does not need it

Fail-Safe Defaults

- Default action is to deny access
- If action fails, system as secure as when action began
  - Undo changes if actions do not complete
  - Transactions (commit)

Economy of Mechanism

- Keep the design and implementation as simple as possible
  - KISS Principle (Keep It Simple, Silly!)
- Simpler means less can go wrong
  - And when errors occur, they are easier to understand and fix
- Interfaces and interactions

Complete Mediation

- Check every access to an object to ensure that access is allowed
- Usually done once, on first action
  - UNIX: Access checked on open, not checked thereafter
- If permissions change after, may get unauthorized access

Open Design

- Security should not depend on secrecy of design or implementation
  - Popularly misunderstood to mean that source code should be public
  - “Security through obscurity”
  - Does not apply to information such as passwords or cryptographic keys

Separation of Privilege

- Require multiple conditions to grant privilege
  - Example: Checks of $70000 must be signed by two people
  - Separation of duty
  - Defense in depth
    - Multiple levels of protection

Least Common Mechanism

- Mechanisms should not be shared
  - Information can flow along shared channels
  - Covert channels
- Isolation
  - Virtual machines
  - Sandboxes

Psychological Acceptability

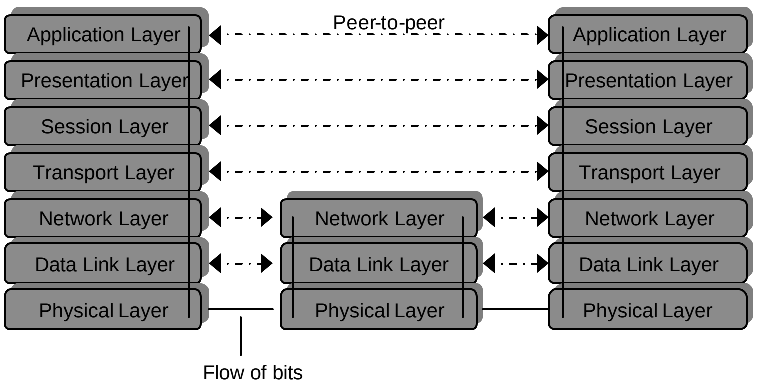
- Security mechanisms should not add to difficulty of accessing resource
  - Hide complexity introduced by security mechanisms
  - Ease of installation, configuration, use
  - Human factors critical here

Key Points

- Principles of secure design underlie all security-related mechanisms
- Require:
  - Good understanding of goal of mechanism and environment in which it is to be used
  - Careful analysis and design
  - Careful implementation

Network Security

ISO/OSI Model

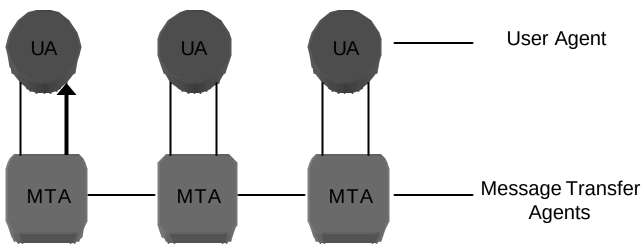


Protocols

- End-to-end protocol
  - Communication protocol that involves end systems with one or more intermediate systems
  - Intermediate host play no part other than forwarding messages
    - Example: telnet
- Link protocol
  - Protocol between every directly connected systems
    - Example: IP – guides messages from a host to one of its immediate host
- Link encryption
  - Encipher messages between intermediate host
  - Each host share a cryptographic key with its neighbor
    - Attackers at the intermediate host will be able to read the message
- End-to-end encryption
  - Example: telnet with messages encrypted/decrypted at the client and server
  - Attackers on the intermediate hosts cannot read the message

Electronic Mail

- UA interacts with the sender
- UA hands it to a MTA
- Attacker can read email on any of the computer with MTA
- Forgery possible



Security at the Application Layer: Privacy-enhanced Electronic Mail (PEM)

- Study by Internet Research Task Force on Privacy or Privacy Research Group to develop protocols with following services
  - Confidentiality, by making the message unreadable except to the sender and recipients
  - Origin authentication, by identifying the sender precisely
  - Data integrity, by ensuring that any changes In the message are easy to detect
  - Non-repudiation of the origin (if possible)

Design Considerations/goals for PEM

- Not to redesign existing mail system protocols
- To be compatible with a range of MTAs, UAs and other computers
- To make privacy enhancements available separately so they are not required
- To enable parties to use the protocol to communicate without prearrangement

PEM Basic Design

- Defines two keys
  - Data Encipherment Key (DEK) to encipher the message sent
    - Generated randomly
    - Used only once
    - Sent to the recipient
  - Interchange key: to encipher DEK
    - Must be obtained some other way than the through the message

Protocols

- Confidential message (DEK: k_s)

Alice —— $\{m\}k_s \,\|\, \{k_s\}k_{Bob}$ —— Bob

- Authenticated, integrity-checked message

Alice —— $m \,\|\, \{h(m)\}k_{Alice}$ —— Bob

- Enciphered, authenticated, integrity checked message

Alice —— $\{m\}k_s \,\|\, \{h(m)\}k_{Alice} \,\|\, \{k_s\}k_{Bob}$ —— Bob