Lecture 11
October 20, 2023
Adding Security to Email

• Goal: provide privacy (confidentiality), authentication of origin, and integrity checking for email

• Two systems
  • Privacy-Enhanced Electronic Mail (PEM)
  • PGP, GPG, OpenPGP — all basically the same

• Ideas underlying both protocols are the same
  • PEM is older and simpler; not used much today
  • PGP/GPG/OpenPGP newer, used widely

• Here, discuss PEM and show differences between it and OpenPGP
Design Principles

• Do not change related existing protocols
  • Cannot alter SMTP

• Do not change existing software
  • Need compatibility with existing software

• Make use of PEM optional
  • Available if desired, but email still works without them
  • Some recipients may use it, others not

• Enable communication without prearrangement
  • Out-of-bands authentication, key exchange problematic
Basic Design: Keys

- Two keys
  - *Interchange keys* tied to sender, recipients and is static (for some set of messages)
    - Like a public/private key pair (indeed, may be a public/private key pair)
    - Must be available *before* messages sent
  - *Data exchange keys* generated for each message
    - Like a session key, session being the message
Confidentiality:

- \( m \) message
- \( k_B \) Bob’s interchange key (his public key, in a public key system)
Basic Design: Integrity

Integrity and authentication:

- $m$ message
- $h(m)$ hash of message $m$ —Message Integrity Check (MIC)
- $k_A$ Alice’s interchange key (her private key, in a public key system)

$$m \{ h(m) \} k_A$$

Alice $\rightarrow$ Bob

Non-repudiation: if $k_A$ is Alice’s private key, this establishes that Alice’s private key was used to sign the message
Basic Design: Everything

Confidentiality, integrity, authentication:
• Notations as in previous slides
• If $k_A$ is Alice’s private key, get non-repudiation too

$$\{ m \} k_s \; || \; \{ h(m) \} k_A \; || \; \{ k_s \} k_B$$

Alice $\rightarrow$ Bob
Practical Considerations

• Limits of SMTP
  • Only ASCII characters, limited length lines

• Use encoding procedure
  1. Map local char representation into canonical format
     – Format meets SMTP requirements
  2. Compute and encipher MIC over the canonical format; encipher message if needed
  3. Map each 6 bits of result into a character; insert newline after every 64th character
  4. Add delimiters around this ASCII message
Problem

- Recipient without PEM-compliant software cannot read it
  - If only integrity and authentication used, should be able to read it
- Mode MIC-CLEAR allows this
  - Skip step 3 in encoding procedure
  - Problem: some MTAs add blank lines, delete trailing white space, or change end of line character
  - Result: PEM-compliant software reports integrity failure
PEM vs. OpenPGP

• Use different ciphers
  • PGP allows several ciphers
    • Public key: RSA, El Gamal, DSA, Diffie-Hellman, Elliptic curve
    • Symmetric key: IDEA, Triple DES, CAST5, Blowfish, AES-128, AES-192, AES-256, Twofish-256
    • Hash algorithms: MD5, SHA-1, RIPE-MD/160, SHA256, SHA384, SHA512, SHA224
  • PEM allows RSA as public key algorithm, DES in CBC mode to encipher messages, MD2, MD5 as hash functions
PEM vs. OpenPGP

• Use different key distribution models
  - PGP uses general “web of trust”
  - PEM uses hierarchical structure

• Handle end of line differently
  - PGP remaps end of line if message tagged “text”, but leaves them alone if message tagged “binary”
  - PEM always remaps end of line
Authentication Basics

• Authentication: binding of identity to subject
  • Identity is that of external entity (my identity, Matt, etc.)
  • Subject is computer entity (process, etc.)
Establishing Identity

• One or more of the following
  • What entity knows (*eg.* password)
  • What entity has (*eg.* badge, smart card)
  • What entity is (*eg.* fingerprints, retinal characteristics)
  • Where entity is (*eg.* In front of a particular terminal)
Authentication System

• \((A, C, F, L, S)\)
  • \(A\) information that proves identity
  • \(C\) information stored on computer and used to validate authentication information
  • \(F\) complementation function; for \(f \in F, f : A \rightarrow C\)
  • \(L\) functions that prove identity; for \(l \in L, l : A \times C \rightarrow \{\text{true, false}\}\)
    • \(l\) is lowercase “\(L\)”
  • \(S\) functions enabling entity to create, alter information in \(A\) or \(C\)
Example

• Password system, with passwords stored on line in clear text
  • A set of strings making up passwords
  • $C = A$
  • $F$ singleton set of identity function \{ $I$ \}
  • $L$ single equality test function \{ $eq$ \}
  • $S$ function to set/change password
Passwords

• Sequence of characters
  • Examples: 10 digits, a string of letters, *etc.*
  • Generated randomly, by user, by computer with user input

• Sequence of words
  • Examples: pass-phrases

• Algorithms
  • Examples: challenge-response, one-time passwords
Storage

- Store as cleartext
  - If password file compromised, *all* passwords revealed
- Encipher file
  - Need to have decipherment, encipherment keys in memory
  - Reduces to previous problem
- Store one-way hash of password
  - If file read, attacker must still guess passwords or invert the hash
Example

• UNIX system original hash function
  • Hashes password into 11 char string using one of 4096 hash functions

• As authentication system:
  • $A = \{ \text{strings of 8 chars or less} \}$
  • $C = \{ 2 \text{ char hash id } \| \ 11 \text{ char hash} \}$
  • $F = \{ 4096 \text{ versions of modified DES} \}$
  • $L = \{ \text{login, su, ...} \}$
  • $S = \{ \text{passwd, nispasswd, passwd+, ...} \}$
Anatomy of Attacking

• Goal: find $a \in A$ such that:
  • For some $f \in F$, $f(a) = c \in C$
  • $c$ is associated with entity

• Two ways to determine whether $a$ meets these requirements:
  • Direct approach: as above
  • Indirect approach: as $l(a)$ succeeds iff $f(a) = c \in C$ for some $c$ associated with an entity, compute $l(a)$
Preventing Attacks

• How to prevent this:
  • Hide one of $a, f, \text{ or } c$
    • Prevents obvious attack from above
    • Example: UNIX/Linux shadow password files hides $c$’s
  • Block access to all $l \in L$ or result of $l(a)$
    • Prevents attacker from knowing if guess succeeded
    • Example: preventing any logins to an account from a network
      • Prevents knowing results of $l$ (or accessing $l$)
Picking Good Passwords

• “WtBvStHbChCsLm?TbWtF.+FSK”
  • Intermingling of letters from Star Spangled Banner, some punctuation, and author’s initials

• What’s good somewhere may be bad somewhere else
  • “DCHNH,DMC/MHmh” bad at Dartmouth (“Dartmouth College Hanover NH, Dartmouth Medical Center/Mary Hitchcock memorial hospital”), ok elsewhere (probably)

• Why are these now bad passwords? 😞
Passphrases

• A password composed of multiple words and, possibly, other characters

• Examples:
  • “home country terror flight gloom grave”
    • From Star Spangled Banner, third verse, third and sixth line
  • “correct horse battery staple”
    • From xkcd

• Caution: the above are no longer good passphrases
Remembering Passphrases

- Memorability is a good example of how environment affects security
  - Study of web browsing shows average user has 6-7 passwords, sharing each among about 4 sites (from people who opted into a study of web passwords)
    - Researchers used an add-on to a browser that recorded information about the web passwords but not the password itself
- Users tend not to change password until they know it has been compromised
  - And when they do, the new passwords tend to be as short as allowed
- Passphrases seem as easy to remember as passwords
  - More susceptible to typographical errors
  - If passphrases are text as found in normal documents, error rate drops
Password Manager (Wallet)

- A mechanism that encrypts a set of user’s passwords
- User need only remember the encryption key
  - Sometimes called “master password”
  - Enter it, and then you can access all other passwords
- Many password managers integrated with browsers, cell phone apps
  - So you enter the master password, and password manager displays the appropriate password entry
  - When it does so, it shows what the password logs you into, such as the institution with the server, and hides the password; you can then have it enter the password for you
Salting

• Goal: slow dictionary attacks
• Method: perturb hash function so that:
  • Parameter controls *which* hash function is used
  • Parameter differs for each password
  • So given $n$ password hashes, and therefore $n$ salts, need to hash guess $n$
Example

• password: hello, there!1

• stored version (no line breaks in password file):
  $6$1BSRcuVLmWnV6LET$dJf2kPCM9Pj0yEvxAtyp8ZJIcgtNY7QEY4J/nDc8iYx9NR610XxCFI7gewN2yduSMu2z4BOAemTOVAn/R0yQV/

• interpretation ($ separates parts of the password):
  • $6$ indicates modular password format and hashing algorithm
    • SHA-512 (1=MD5, 2=Blowfish, 3=NT-Hash [doesn’t use salt, use discouraged, 5=SHA-256)
  • 1BSRcuVLmWnV6LET is salt
  • dJf2kPCM9PjOyEvxAtyp8ZJIcgtNY7QEY4J/nDc8iYx9NR610XxCFI7gewN2yduSMu2z4BOAemTOVAn/R0yQV/ is hash of password and salt
Challenge-Response

User, system share a secret function $f$ (in practice, $f$ is a known function with unknown parameters, such as a cryptographic key)

- User request to authenticate
- System responds with $f(r)$
- User sends $f(r)$ to system
- System verifies $f(r)$

The challenge is $r$ (random message)
One-Time Passwords

• Password that can be used exactly *once*
  • After use, it is immediately invalidated

• Challenge-response mechanism
  • Challenge is number of authentications; response is password for that particular number

• Problems
  • Synchronization of user, system
  • Generation of good random passwords
  • Password distribution problem
S/Key

• One-time password scheme based on idea of Lamport
• $h$ one-way hash function (MD5 or SHA-1, for example)
• User chooses initial seed $k$
• System calculates:
  \[ h(k) = k_1, h(k_1) = k_2, \ldots, h(k_{n-1}) = k_n \]
• Passwords are reverse order:
  \[ p_1 = k_n, p_2 = k_{n-1}, \ldots, p_{n-1} = k_2, p_n = k_1 \]
S/Key Protocol

System stores maximum number of authentications $n$, number of next authentication $i$, last correctly supplied password $p_{i-1}$.

System computes $h(p_i) = h(k_{n-i+1}) = k_{n-i} = p_{i-1}$. If match with what is stored, system replaces $p_{i-1}$ with $p_i$ and increments $i$. 
Hardware Support

• Token-based
  • Used to compute response to challenge
    • May encipher or hash challenge
    • May require PIN from user

• Temporally-based
  • Every minute (or so) different number shown
    • Computer knows what number to expect when
  • User enters number and fixed password
Biometrics

- Automated measurement of biological, behavioral features that identify a person
  - Fingerprints: optical or electrical techniques
  - Voices: speaker verification or recognition
  - Eyes: patterns in irises unique
  - Faces: image, or specific characteristics like distance from nose to chin
  - Keystroke dynamics: believed to be unique
Location

• If you know where user is, validate identity by seeing if person is where the user is
  • Requires a device saying where the user is, like a smart phone
Multi-Factor Authentication

- Example: “where you are” also requires entity to have LSS and GPS, so also “what you have”
- Can assign different methods to different tasks
  - As users perform more and more sensitive tasks, must authenticate in more and more ways (presumably, more stringently) File describes authentication required
    - Also includes controls on access (time of day, etc.), resources, and requests to change passwords
  - Pluggable Authentication Modules
Multi-Factor Authentication

- Example: “where you are” also requires entity to have LSS and GPS, so also “what you have”
- Can assign different methods to different tasks
  - As users perform more and more sensitive tasks, must authenticate in more and more ways (presumably, more stringently) File describes authentication required
    - Also includes controls on access (time of day, etc.), resources, and requests to change passwords
  - Pluggable Authentication Modules
Identity

• *Principal*: a unique entity
• *Identity*: specifies a principal
• *Authentication*: binding of a principal to a representation of identity internal to the system
  • All access, resource allocation decisions assume binding is correct
Files and Objects

• Identity depends on system containing object

• Different names for one object
  • Human use, *eg.* file name
  • Process use, *eg.* file descriptor or handle
  • Kernel use, *eg.* file allocation table entry, inode
More Names

• Different names for one context
  • Human: aliases, relative vs. absolute path names
  • Kernel: deleting a file identified by name can mean two things:
    • Delete the object that the name identifies
    • Delete the name given, and do not delete actual object until all names have been deleted

• Semantics of names may differ
Example: Names and Descriptors

• Interpretation of UNIX file name
  • Kernel maps name into an inode using iterative procedure
  • Same name can refer to different objects at different times without being deallocated
    • Causes race conditions

• Interpretation of UNIX file descriptor
  • Refers to a specific inode
  • Refers to same inode from creation to deallocation
Example: Different Systems

• Object name must encode location or pointer to location
  • *SSH* style: *host:*object
  • URLs: *protocol://host/object*

• Need not name actual object
  • *SSH* style may name pointer (link) to actual object
  • URL may forward to another host
Users

• Exact representation tied to system

• Example: UNIX/Linux systems
  • Login name: used to log in to system
    • Logging usually uses this name
  • User identification number (UID): unique integer assigned to user
    • Kernel uses UID to identify users
    • One UID per login name, but multiple login names may have a common UID
Multiple Identities

• UNIX systems again
  • Real UID: user identity at login, but changeable
  • Effective UID: user identity used for access control
    • Setuid changes effective UID
  • Saved UID: UID before last change of UID
    • Used to implement least privilege
    • Work with privileges, drop them, reclaim them later
  • Audit/Login UID: user identity used to track original UID
    • Cannot be altered; used to tie actions to login identity
Groups

• Used to share access privileges

• First model: alias for set of principals
  • Processes assigned to groups
  • Processes stay in those groups for their lifetime

• Second model: principals can change groups
  • Rights due to old group discarded; rights due to new group added
Roles

- Group with membership tied to function
  - Rights given are consistent with rights needed to perform function
- Uses second model of groups
- Example: DG/UX
  - User *root* does not have administration functionality
  - System administrator privileges are in *sysadmin* role
  - Network administration privileges are in *netadmin* role
  - Users can assume either role as needed
Naming and Certificates

• Certificates issued to a principal
  • Principal uniquely identified to avoid confusion

• Problem: names may be ambiguous
  • Does the name “Matt Bishop” refer to:
    • The author of this book?
    • A programmer in Australia?
    • A stock car driver in Muncie, Indiana?
    • Someone else who was named “Matt Bishop”