Lecture 9
October 16, 2024

Fast Exponentiation

- Idea: compute 2^5 mod 9
- 5 = 101 in binary so …
  - base = 2
  - 1 bit gives 1 * base mod 9 = 1 * 2 mod 9 = 2
- New base is $base^2$ mod 9 = 4
- 101 shifted right 1 bit, so look at 10 in binary (0 bit) …
  - base = 4
  - 0 bit means don't multiply
- New base is $base^2$ mod 9 = 16 mod 9 = 7
- 10 shifted with 1 bit, so look at 1 in binary (1 bit) …
  - base = 7
  - 1 bit gives 2 * 7 mod 9 = 14 mod 9 = 5
- New base is $base^2$ mod 9 = 49 mod 9 = 4
- 1 shifted right 1 bit is 0, so done; result is 5

Another Example

- Compute 12^{64} mod 93
- In bits, 64 = $1_7 0_6 0_5 0_4 0_3 0_2 0_1$

1. So as rightmost bit is 0, base = 12^2 mod 93 = 51
2. Shift right, rightmost bit is 0, so base is 51^2 mod 93 = 90
3. Shift right, rightmost bit is 0, so base is 90^2 mod 93 = 9
4. Shift right, rightmost bit is 0, so base is 9^2 mod 93 = 81
5. Shift right, rightmost bit is 0, so base is 81^2 mod 93 = 51
6. Shift right, rightmost bit is 0, so base is 51^2 mod 93 = 90
7. Shift right, rightmost bit is 1, so 1 * base mod 93 = 90

- So 12^{64} mod 93 = 90

And to Verify, We Use *dc*(1)

A multi-precision calculator on UNIX-like systems that uses postfix (reverse Polish) notation:

```
bishop — dc — 80×24
Last login: Tue Oct 15 15:33:57 on ttys000
bishop@campus-056-054 ~ % dc
2 5 ^ 9 % p
5
12 64 ^ 93 % p
90
```

Algorithm (in Python)

```
# compute g^k mod n
def fastexp(g, n, k):
    retval = 1
    base = g
    while k != 0:
        r = k % 2
        if r == 1:
            retval = (retval * base) % n
        k = k // 2
        base = (base * base) % n
    return retval
```

Algorithm (in C)

```
# compute g^k mod n
int fastexp(int g, int n, int k)
{
    retval = 1;
    base = g
    do{
        if (k&01)
            retval = (retval * base) % n;
        k >>= 1;
        base = (base * base) % n;
    }while (k);
    return retval;
}
```

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

Basic Design: Confidentiality

Confidentiality:

- *m* message
- k_B Bob's interchange key (his public key, in a public key system)

$$\text{Alice} \xrightarrow{\{\, m \,\}\, k_B} \text{Bob}$$

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)

$$\text{Alice} \xrightarrow{m \{ h(m) \} k_A} \text{Bob}$$

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 ⟶ 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 $l \in L,\ l : A \times C \rightarrow \{$ 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* = { strings of 8 chars or less }
  - *C* = { 2 char hash id || 11 char hash }
  - *F* = { 4096 versions of modified DES }
  - *L* = { *login*, *su*, … }
  - *S* = { *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*, 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 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

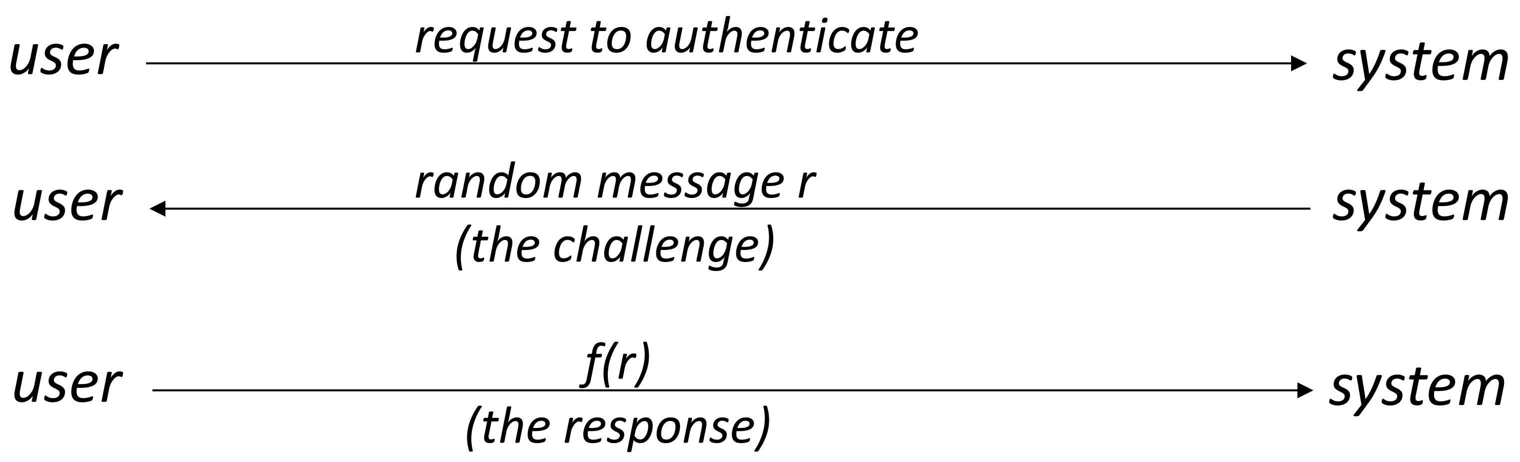
- Goal: slow dictionary attacks
- Method: perturb hash function so that:
  - Parameter controls *which* hash function is used
  - Parameter differs for each password
  - So given *n* password hashes, and therefore *n* salts, need to hash guess *n*

Example

- password: hello,there!1
- stored version (no line breaks in password file):
  `$6$1BSRcuVLmWnV6LET$dJf2kPCM9PjOyEvxAtyp8ZJIcgtNY7QEY4J/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)



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 (SHA-256, for example)
- User chooses initial seed *k*
- System calculates:

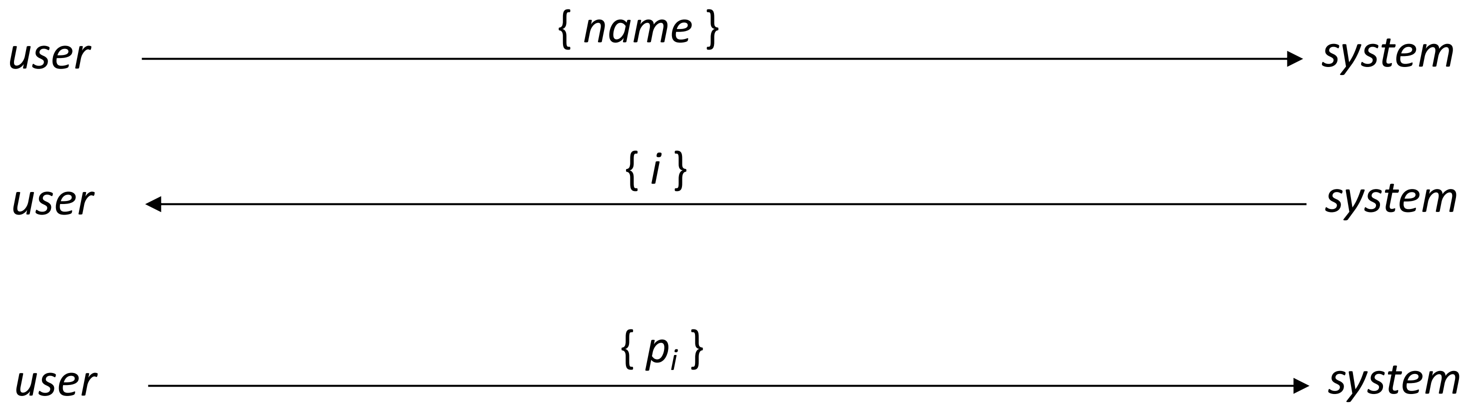
$$h(k) = k_1, h(k_1) = k_2, \ldots, h(k_{n-1}) = k_n$$

- 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