



slides kindly provided by:

Tam Vu, Ph.D
Professor of Computer Science
Director, Mobile & Networked Systems Lab
Department of Computer Science
University of Colorado Boulder

CSCI 3753 Operating Systems Spring 2019

Christopher Godley
PhD Student
Department of Computer Science
University of Colorado Boulder



Security in Operating Systems

Relevance of security to operating systems:

- OS must keep track of rights a user has to each file, object, and service
 - this is a form of *authorization*
- Users have to provide a password to login
 - this is a form of *authentication*
- Some data is sensitive and be encrypted.
 - this is a form of *confidentiality*.
- Networked services to remote users may invite malicious adversaries
 - Who may wish to prevent access to these services, (DDOS)
 - this is a form of (un)*availability*.
- Detect whether data has been tampered with.
 - this form of *data integrity*.

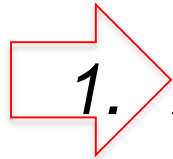
Security concepts

1. Authentication – proving you are who you say you are, e.g. passwords
2. Authorization – managing access to resources, e.g. files
3. Confidentiality – only allow authorized viewing of data - encrypting files and communication
4. Data Integrity – detecting tampering with digital data
5. Non-repudiation – proving an event happened
6. Availability – ensuring a service is available (despite denial of service attacks)

Defense In Depth

- Standard security philosophy is *defense-in-depth*
 - employ multiple layers of security
- For each layer, identify:
 - What is the threat model?
 - e.g. eavesdropping, replay, MIM, DDOS, etc.
 - What resources does the attacker have available to them?
 - One attacker or many?
 - A laptop or a supercomputer?
 - What resources do you have to defend at that layer?

6 Main Areas of Security



1. *Authorization* – managing access to resources, e.g. files
2. *Confidentiality* – only allow authorized viewing of data - encrypting files and communication
3. *Authentication* – proving you are who you say you are, e.g. passwords
4. Data Integrity – detecting tampering with digital data
5. Non-repudiation – proving an event happened
6. Availability – ensuring a service is available (despite denial of service attacks)

Authorization

- First authenticate a user with a login password
- Then, OS must determine what files/services the user/process is authorized to access
 - login shell or process operates in a *protection domain* that specifies which resources it may access
 - a domain is a collection of access rights, each of which is an ordered pair <object, set of rights>
 - rights can include read, write, execute, print privileges, etc.
 - in UNIX, a domain is associated with a user
- can collect object and access rights into an *access matrix*

Authorization

access
matrix

objects

domains,
e.g. users

| | file F1 | file F2 | file F3 | printer | D1 | D2 | D3 | D4 |
|----|----------------|---------------|----------------|---------|----|--------|----|-------------------|
| D1 | read | | read | | | switch | | |
| D2 | | owner read | | print | | | | switch control |
| D3 | | read | execute | | | | | |
| D4 | read, write | | read, write | | | | | |

- a process executing in protection domain D1, e.g. as user U1, has permission to read file F1, read F3, and *switch* to another domain D2
- a process in domain D2 has *control* right to modify permissions in *row* D4 and *owner* right to modify permissions in the *column* for file F2

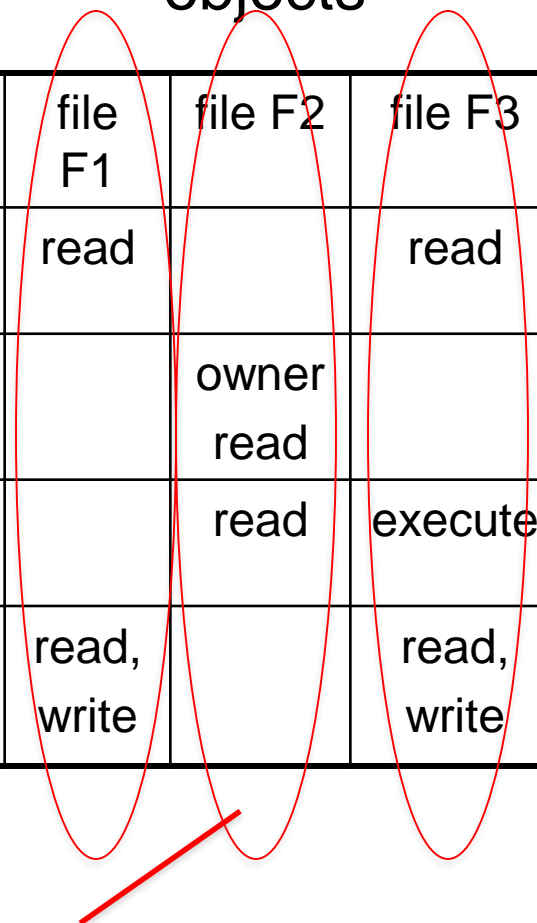
Authorization

- Implementation 1: **access matrix as 1 global table**
 - large, may be difficult to keep it all in memory
 - could use VM-like demand paging to keep only active portions of access matrix in memory
 - difficult to exploit relationships
 - e.g. changing the read access to a given file for an entire group of users - have to change each entry in the matrix
 - waste of space
 - matrix may be very sparse, with few entries filled in, yet would have to allocate space for the matrix entry anyway

Access Control Lists

access
matrix

objects



| | file F1 | file F2 | file F3 |
|----|-------------|------------|-------------|
| D1 | read | | read |
| D2 | | owner read | |
| D3 | | read | execute |
| D4 | read, write | | read, write |

domains,
e.g. users

- Implementation of an access matrix as an *access control list (ACL)*
 - each *column* of the access matrix defines access rights to a particular object, e.g. a file
 - store the access permissions in an ACL with the file header

All access permissions to file F2 are stored in F2's file header, forming an ACL for F2

Access Control Lists

- When a process tries to access the file, search the ACL for the proper permissions
 - define a **default set of permissions** when a process in a domain with an empty entry tries to access the file
- Advantages:
 - Can use existing data structures of file headers – just add a field for ACLs
 - Only keep in memory ACLs of active files/directories
 - empty entries can be discarded to save space (then use Default setting)
- Disadvantage:
 - Determining the set of access rights across a domain is difficult, while determining the set of access rights for a given file is easy

Capability Lists

access
matrix

objects

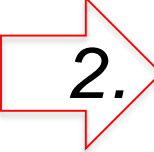
| | file F1 | file F2 | file F3 |
|----|----------------|---------------|----------------|
| D1 | read | | read |
| D2 | | owner read | |
| D3 | | read | execute |
| D4 | read, write | | read, write |

domains,
e.g. users

- each row of the access matrix defines access permissions for a particular user/domain
 - creates a *capability list* for each user
 - Store the capability list with each user

All access permissions for user D4 are stored with D4's account information, forming a list of capabilities for user D4

6 Main Areas of Security

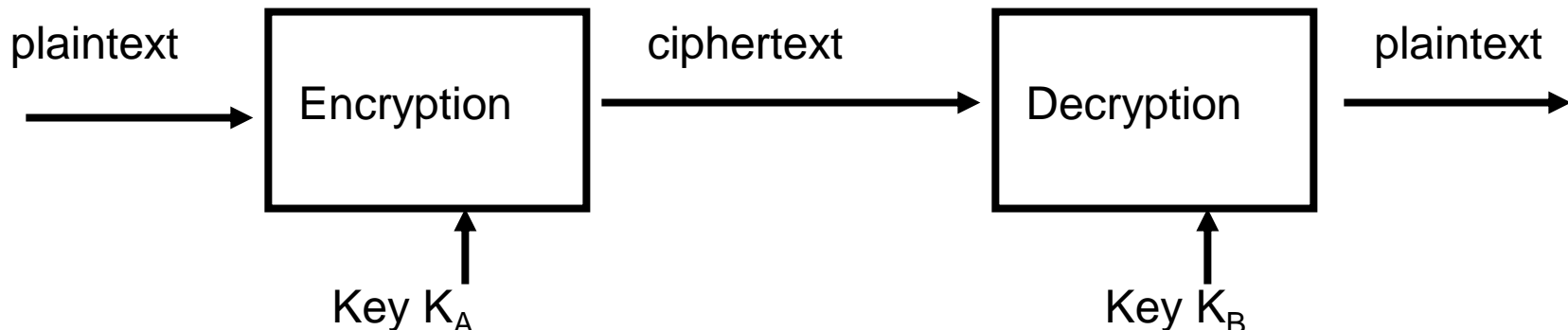
1. *Authorization* – managing access to resources, e.g. files
-  2. *Confidentiality* – only allow authorized viewing of data - encrypting files and communication
3. *Authentication* – proving you are who you say you are, e.g. passwords
4. Data Integrity – detecting tampering with digital data
5. Non-repudiation – proving an event happened
6. Availability – ensuring a service is available (despite denial of service attacks)

Confidentiality

- Encrypt
 - Files to protect the confidentiality of the data
 - Communication messages to protect the confidentiality of the messages
- Only designated decryptors can view the data
- Given a string “*secret message*”, how would you encrypt it?
 - Substitute other letters for given letters
 - Permute order of letters
- Remembering the pattern of substitution and permutation = the *key* for encryption/decryption

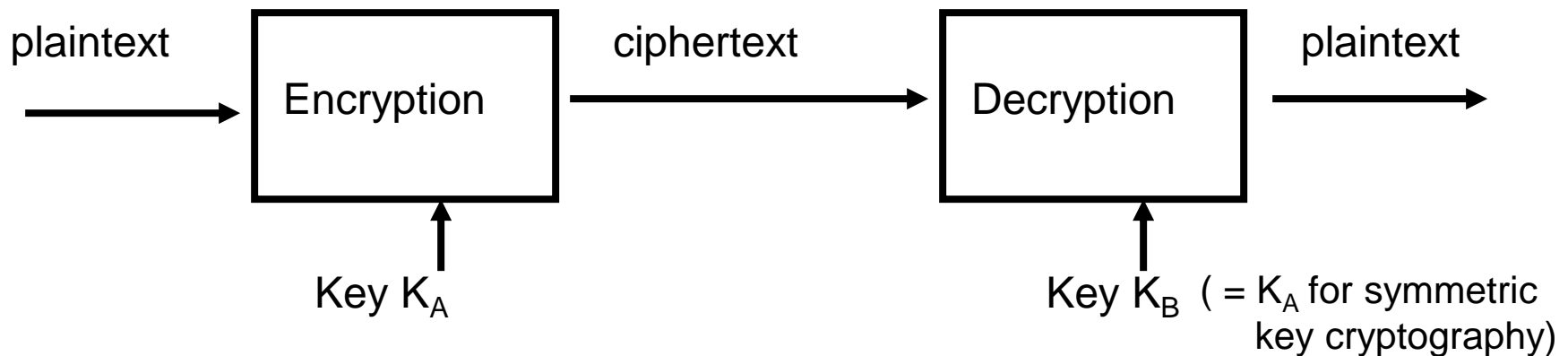
Confidentiality

- Modern cryptography uses
 - keys to encrypt and decrypt
 - Complex combinations of substitution & permutation
- Only have to protect the keys from discovery
 - Don't have to protect the algorithm for encryption and decryption from discovery – this can be publicly known!



Symmetric Key Cryptography

- Symmetric key cryptography: $K_A = K_B$
 - Use the same key for encryption & decryption
 - Has been used since the times of the Romans
 - Also called secret key or private key cryptography
- AES (Advanced Encryption Standard) uses symmetric key cryptography



Symmetric Key Cryptography

- Encrypted file systems use symmetric key cryptography
 - EFS (Encrypting File System for Windows)
 - and EncFS (for Linux, uses FUSE)
- The symmetric key used to encrypt/decrypt files is itself stored in encrypted form
 - Don't want the symmetric key stored in plaintext in a file for attacker to steal from file system
 - You enter a password (more accurately, a *passphrase*) to decrypt this key at run time, which is then used to encrypt/decrypt files

Confidentiality

- Suppose the encryptor and decryptor are physically separate (different locations)
- How does the decryptor securely obtain the symmetric key K ?
 - common to any remote login problem
 - this is the classic *symmetric key distribution problem*
 - one way is to “securely” transport the key to the destination
 - but there’s no guarantee that a spy won’t intercept the key K
 - even worse, the spy could copy the key K without letting the decryptor or encryptor know, and then eavesdrop on all future encrypted communications!

Confidentiality

- Public key cryptography emerged in the 1970s, invented by Diffie and Hellman (and Merkle)
 - endpoints exchange *public* quantities with each other
 - Each endpoint then calculates its symmetric key from these publicly exchanged quantities
 - The symmetric keys calculated are the same
 - even though an attacker could eavesdrop on all the public communications, it cannot calculate the symmetric key!
 - this solves the classic *symmetric key distribution problem* (with a caveat explained later), and was the foundation for public key cryptography

Diffie-Hellman Key Exchange

Host X



Choose a , g , and p

Calculate $A = g^a$

$\text{mod } p$

Send A , g , and p in
the clear

A, g, p

Host Y



Choose b

Calculate $B = g^b \text{ mod } p$

Send B in the clear

B

$K = B^a \text{ mod } p$

$K = A^b \text{ mod } p$

*Two mathematical properties: $B^a \text{ mod } p = A^b \text{ mod } p$
And $A(x) = g^a \text{ mod } p$ is not easily invertible, i.e. can't
find a from $A(x)$*

$$B^a \text{ mod } p = A^b \text{ mod } p$$
$$(g^b)^a \text{ mod } p = (g^a)^b \text{ mod } p$$

Diffie-Hellman Key Exchange

Host X



Choose a , g , and p

Calculate $A = g^a$

$\text{mod } p$

Send A , g , and p in
the clear

A, g, p

Host Y



Choose b

Calculate $B = g^b \text{ mod } p$

Send B in the clear

B

$K = B^a \text{ mod } p$

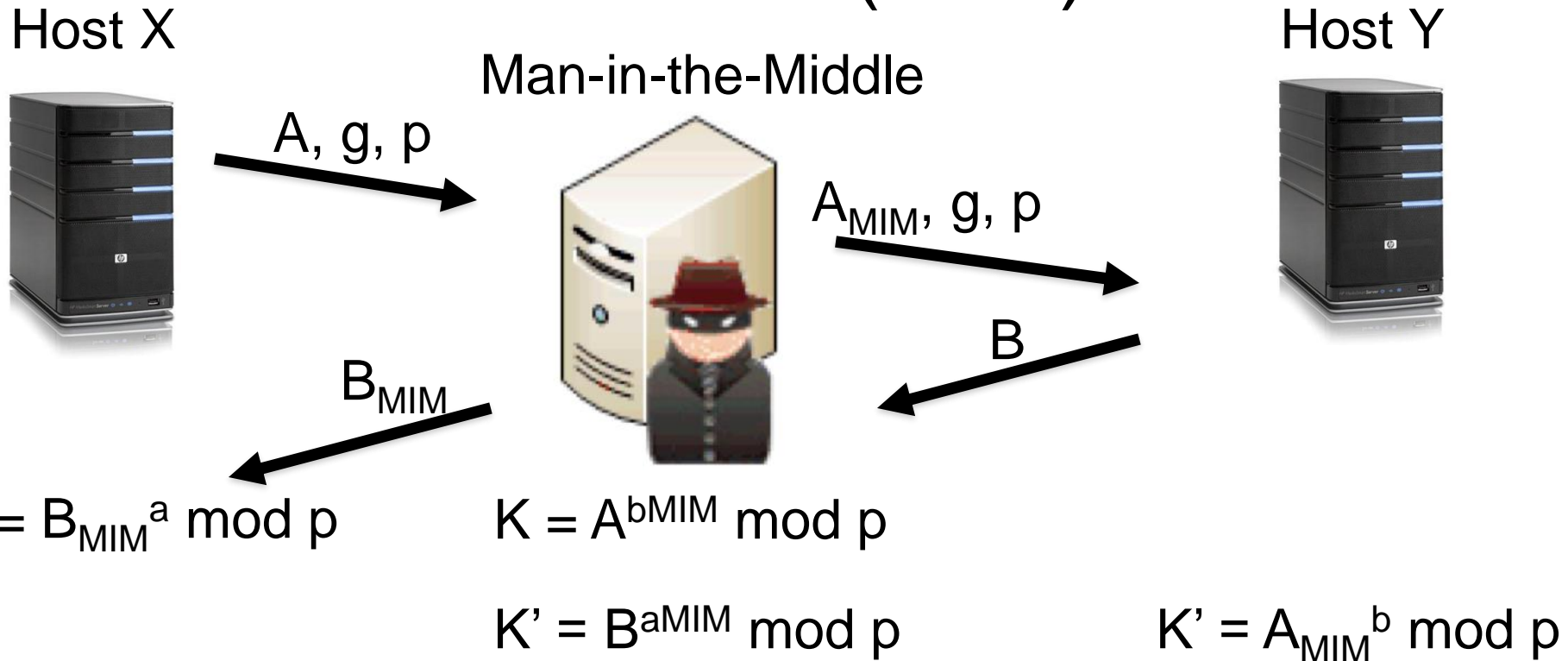
$K = A^b \text{ mod } p$

- Even if an attacker knows A , g , p , B , & algorithm $f(x) = g^x \text{ mod } p$, they cannot compute K
 - would have to invert f to find a or b , then K can be calculated
 - But inverting f is not computationally feasible

Diffie-Hellman Key Exchange

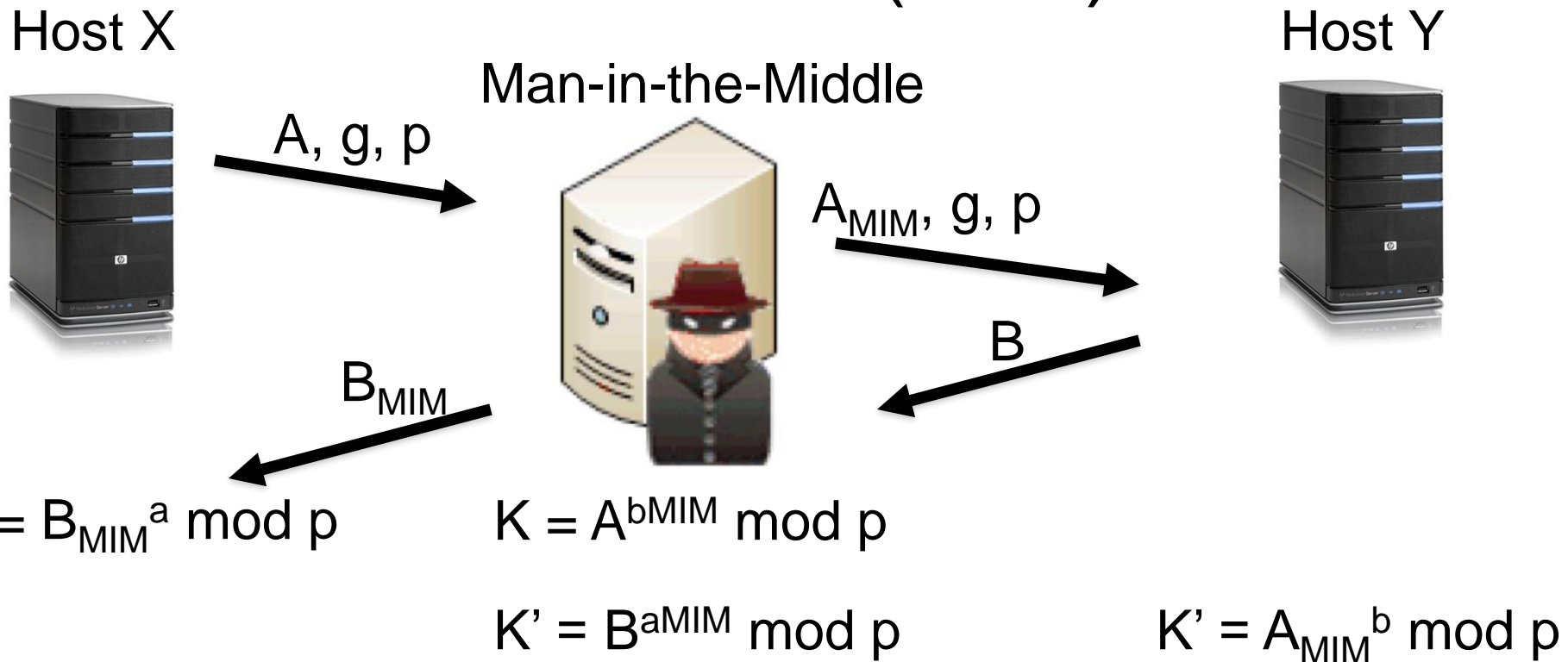
- Diffie-Hellman can also be used for encryption, i.e. public key encryption, not just key establishment
 - This led to the field of public key cryptography
 - Other algorithms like RSA are typically used for public key cryptography, and Diffie-Hellman is used more for key establishment

Diffie-Hellman Vulnerable to a Man-in-the-Middle (MIM) Attack



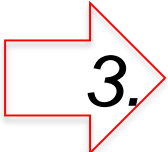
- Man-in-the-middle can compute both K and K' !
 - MIM can decrypt & observe all messages between X and Y !

Diffie-Hellman Vulnerable to a Man-in-the-Middle (MIM) Attack



- MIM can re-encrypt messages with K' or K so neither X nor Y know their communication has been compromised!
- Solution is to use certified public key infrastructure

6 Main Areas of Security

1. *Authorization* – managing access to resources, e.g. files
2. *Confidentiality* – only allow authorized viewing of data - encrypting files and communication
-  3. *Authentication* – proving you are who you say you are, e.g. passwords
4. *Data Integrity* – detecting tampering with digital data
5. *Non-repudiation* – proving an event happened
6. *Availability* – ensuring a service is available (despite denial of service attacks)

Authentication

- Prove you are who you say you are
 - e.g. Logging into your laptop or smartphone
- Password is a form of authentication
 - Providing the correct password is seen as authenticating the user to the OS
- Biometric authentication on smartphones
- For text-based authentication:
 - Attacker can try to guess your password, using common words, etc.
 - OS can block or slow down access after too many login attempts

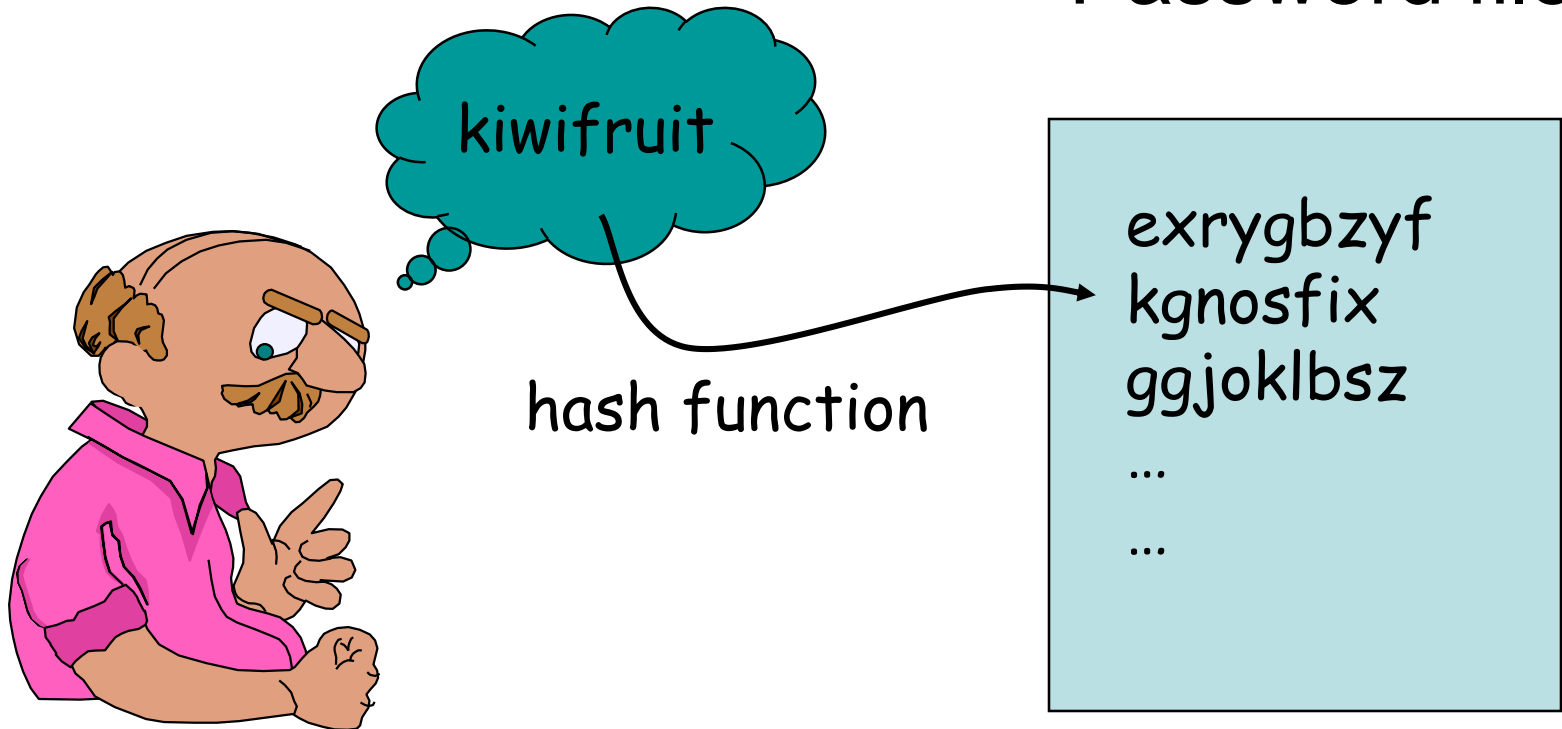
Password authentication

- Basic idea
 - User has a secret password
 - System checks password to authenticate user
- Issues
 - How is password stored?
 - How does system check password?
 - How easy is it to guess a password?
 - Difficult to keep password file secret, so best if it is hard to guess password even if you have the password file

Basic password scheme

User

Password file



Basic password scheme

- Hash function $h : \text{strings} \rightarrow \text{strings}$
 - Given $h(\text{password})$, hard to find password
 - No known algorithm better than trial and error
- User password stored as $h(\text{password})$
- When user enters password
 - System computes $h(\text{password})$
 - Compares with entry in password file
- No passwords stored on disk

Dictionary Attack – some numbers

- Typical password dictionary
 - 1,000,000 entries of common passwords
 - people's names, common pet names, and ordinary words.
 - Suppose you generate and analyze 10 guesses per second
 - This may be reasonable for a web site; offline is *much* faster
 - Dictionary attack in at most 100,000 seconds = 28 hours, or 14 hours on average
- If passwords were random
 - Assume six-character password
 - Upper- and lowercase letters, digits, 32 punctuation characters
 - 689,869,781,056 password combinations.
 - Exhaustive search requires 1,093 years on average

Challenge-response Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”



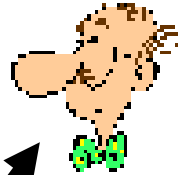
Failure scenario??



Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”

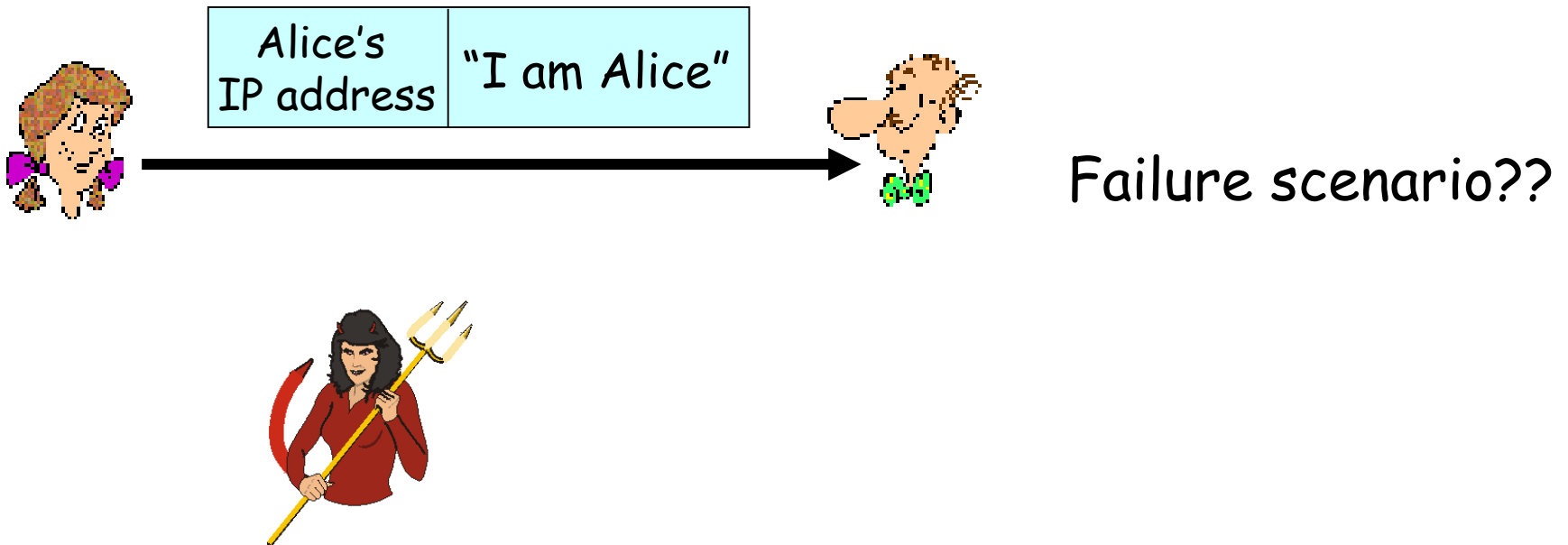


“I am Alice”

in a network,
Bob can not “see”
Alice, so Trudy simply
declares
herself to be Alice

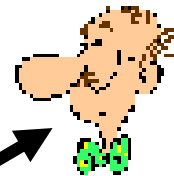
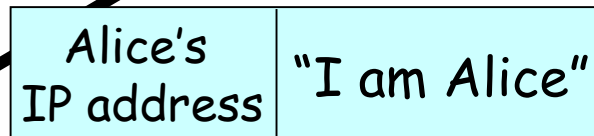
Authentication: another try

Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Authentication: another try

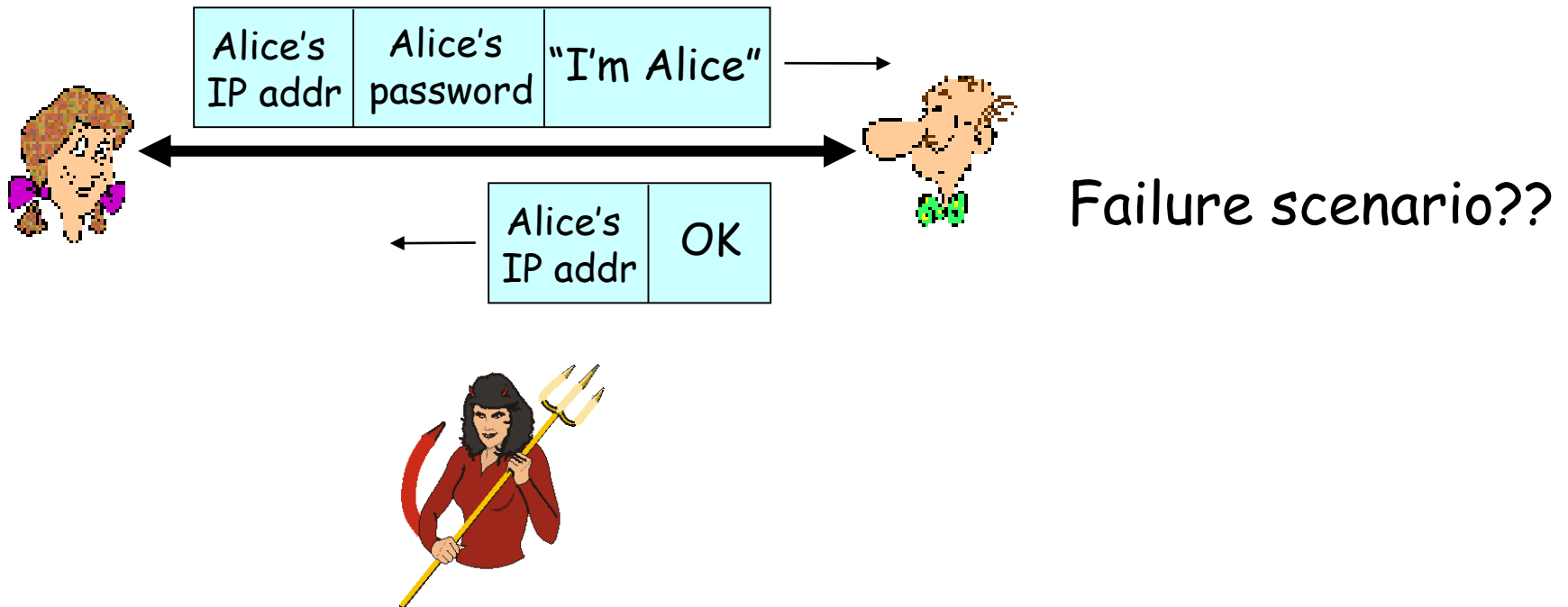
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



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

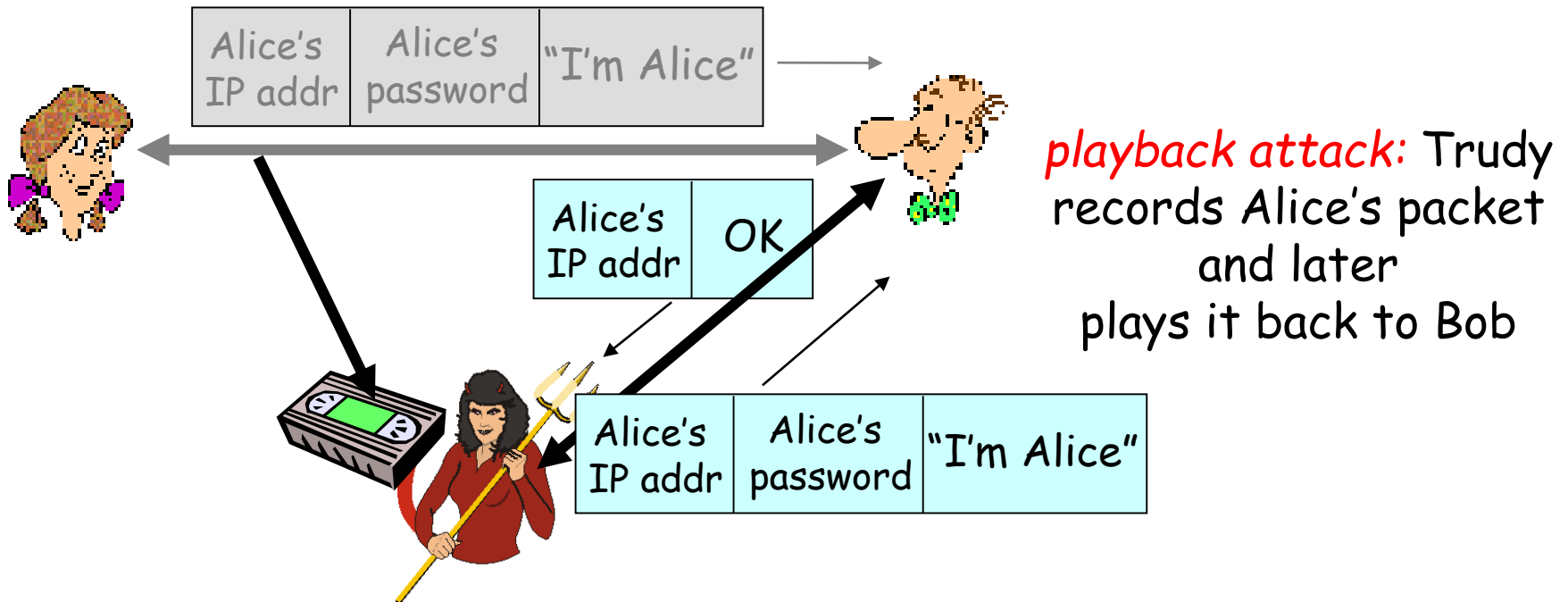
Authentication: another try

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



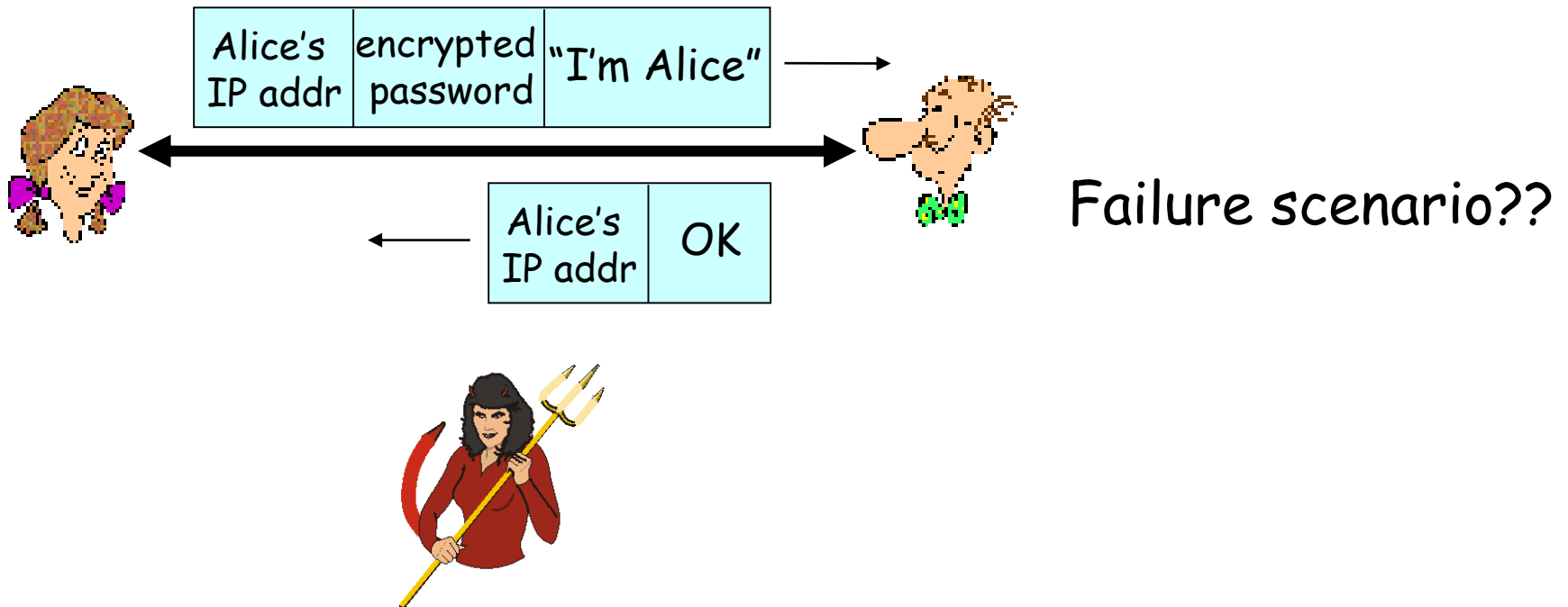
Authentication: another try

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.



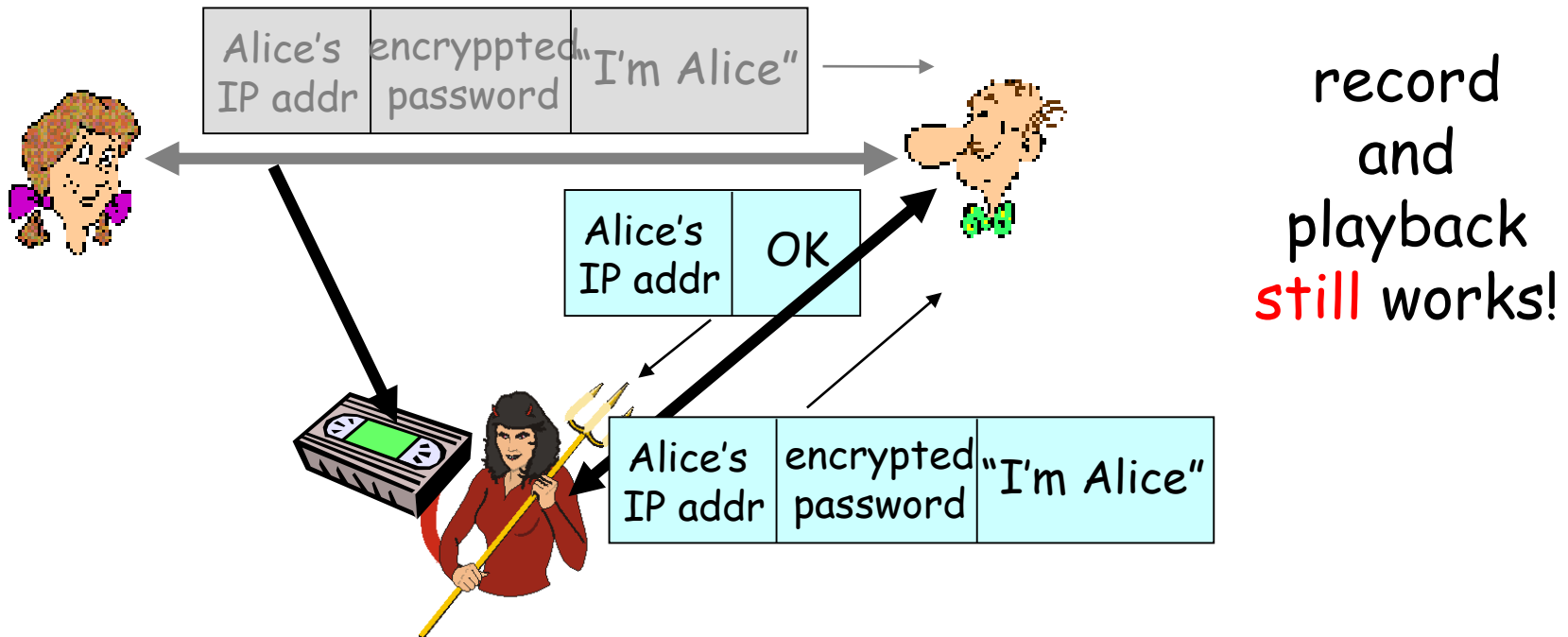
Authentication: yet another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Authentication: another try

Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.

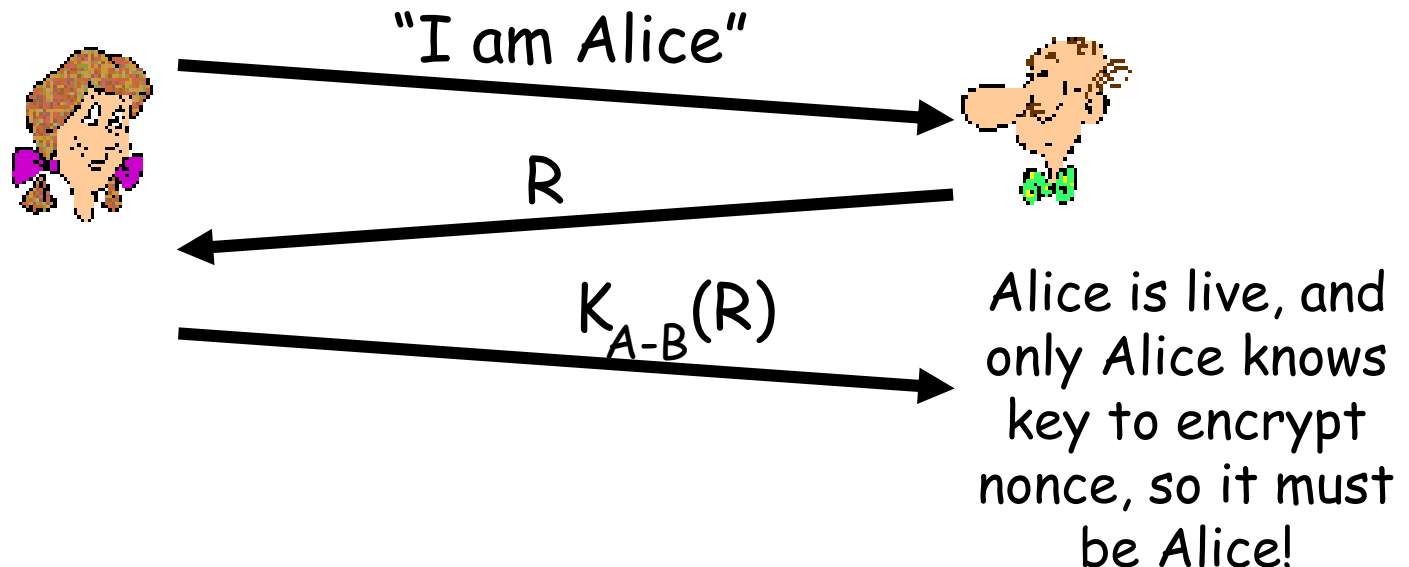


Authentication: yet another try

Goal: avoid playback attack

Nonce: number (R) used only *once -in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice **nonce**, R. Alice must return R, encrypted with shared secret key

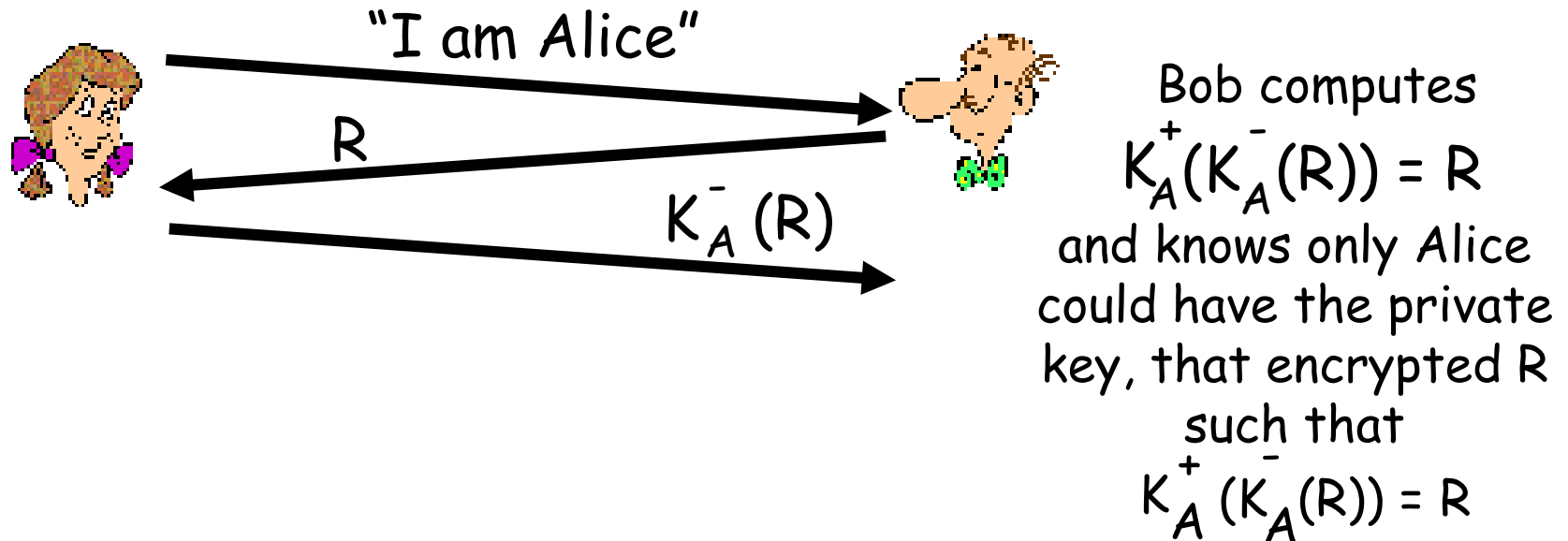


Authentication: ap5.0

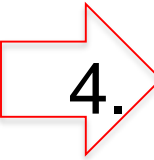
ap4.0 doesn't protect against server database reading

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



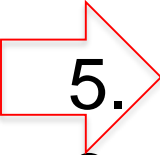
6 Main Areas of Security

1. *Authorization* – managing access to resources, e.g. files
2. *Confidentiality* – only allow authorized viewing of data - encrypting files and communication
3. *Authentication* – proving you are who you say you are, e.g. passwords
-  4. *Data Integrity* – detecting tampering with digital data
5. *Non-repudiation* – proving an event happened
6. *Availability* – ensuring a service is available (despite denial of service attacks)

Data Integrity

- Refers to the overall completeness, accuracy and consistency of the data
- Physical integrity
 - Challenges of correctly storing and retrieving the data
 - Hardware faults
- Logical integrity
 - Software bugs (corrupting data)
 - Human errors

6 Main Areas of Security

1. *Authorization* – managing access to resources, e.g. files
2. *Confidentiality* – only allow authorized viewing of data - encrypting files and communication
3. *Authentication* – proving you are who you say you are, e.g. passwords
4. Data Integrity – detecting tampering with digital data
-  5. Non-repudiation – proving an event happened
6. Availability – ensuring a service is available (despite denial of service attacks)

Non-repudiation

- In digital security, non-repudiation means
 - Involves associating actions or charges with a unique individual
 - A service that provides proof of the integrity and origin of data
 - An authentication that can be said to be genuine with high confidence
- Usually requires:
 - Authentication
 - Authorization
 - Data Integrity
 - Confidentiality

6 Main Areas of Security

1. *Authorization* – managing access to resources, e.g. files
2. *Confidentiality* – only allow authorized viewing of data - encrypting files and communication
3. *Authentication* – proving you are who you say you are, e.g. passwords
4. Data Integrity – detecting tampering with digital data
5. Non-repudiation – proving an event happened
6. Availability – ensuring a service is available (despite denial of service attacks)

Denial-of-service attacks (DoS)

Designed to make data, a machine, or network resource unavailable to its intended users

- Denial-of-service attacks (DoS)
 - Either a single source or distributed attack to prevent access by authorized users
- Direct-access attacks
 - An unauthorized user gaining physical access to a computer
- Tampering
 - Malicious modification of information

Denial-of-service attacks (DoS)

Designed to make data, a machine, or network resource unavailable to its intended users

- Eavesdropping
 - Act of surreptitiously listening to a private conversation, typically between hosts on a network
- Phishing
 - Attempt to acquire sensitive information such as usernames and passwords
- Privilege escalation
 - Attacker with some level of restricted access is able to, without authorization, elevate their privileges or access level
- Spoofing attack
 - Spoofing is the act of masquerading as a valid entity through falsification of data (such as an IP address or username), in order to gain access to information or resources that one is otherwise unauthorized to obtain