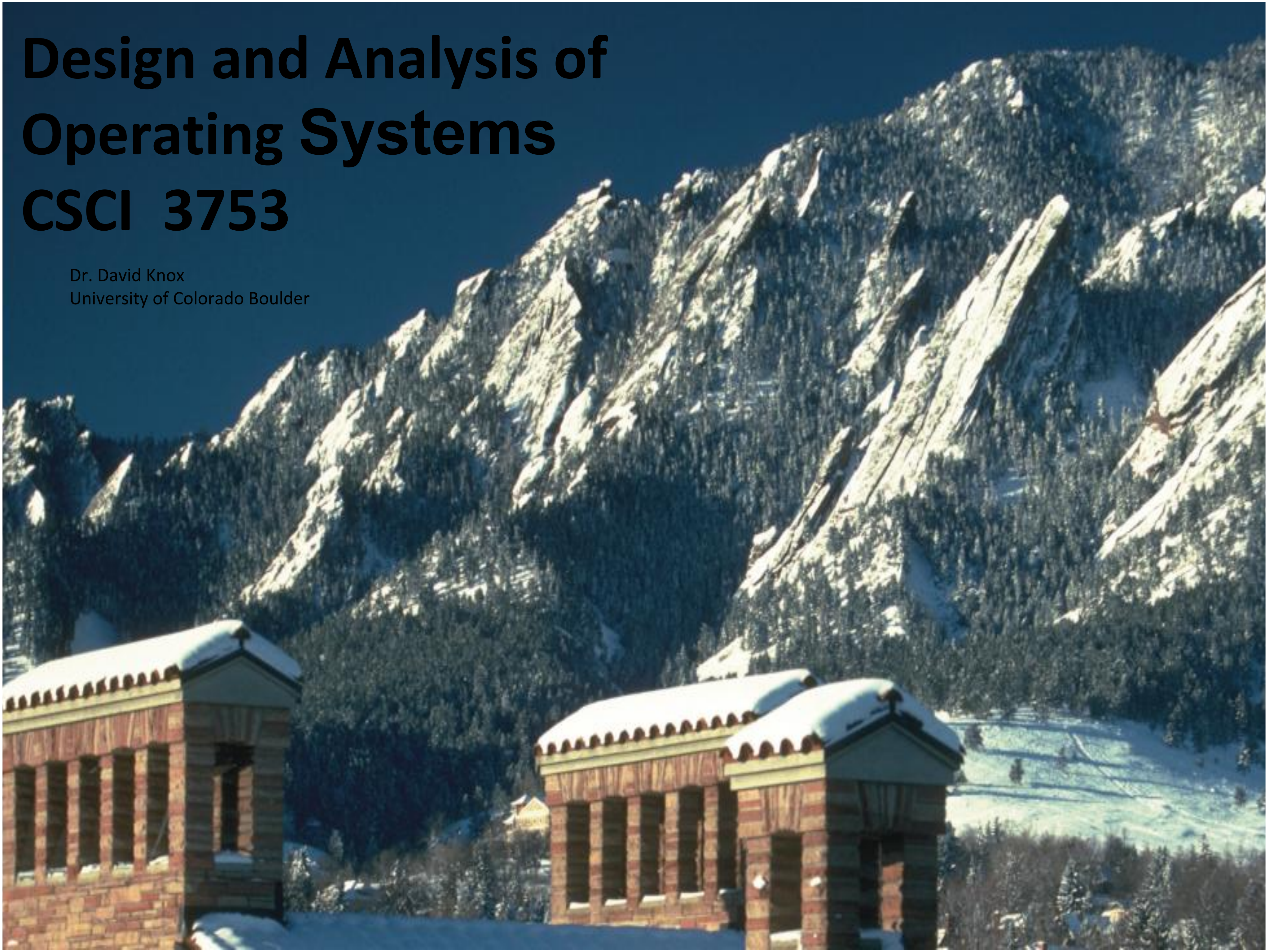
Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder



Design and Analysis of Operating Systems CSCI 3753

Security in Operating Systems

Dr. David Knox

University of Colorado Boulder

Security in Operating Systems

Authorization

Confidentiality

6 Main Areas of Security

1. *Authorization* – managing access to resources
2. *Confidentiality* – only allow authorized viewing of data - encrypting files and communication
3. *Authentication* – proving you are who you say you are
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 of an 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
  - still 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
  - difficult to compress
    - 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

domains, e.g. users

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

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

- 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

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
- **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

Access Control Lists

- **UNIX and Windows NT/2000 use a form of ACL**
  - access permissions stored with the file header/FCB
  - in UNIX, *ls -lg* will reveal the file permissions
    - "-rwxrwxrwxs filename" is the format returned
    - the first 3 fields specify read/write/execute permissions for the file for this user
    - the next 3 fields specify r/w/x permissions for the group,
    - and the last 3 fields specify r/w/x permissions for the "world"
  - *chmod* will change file permissions to files that the user owns
    - *chmod 700 data.txt*
      - changes the rwx permissions to on for the owner, and off for group and world

Capability Lists

access matrix

objects

domains, e.g. users

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

- 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

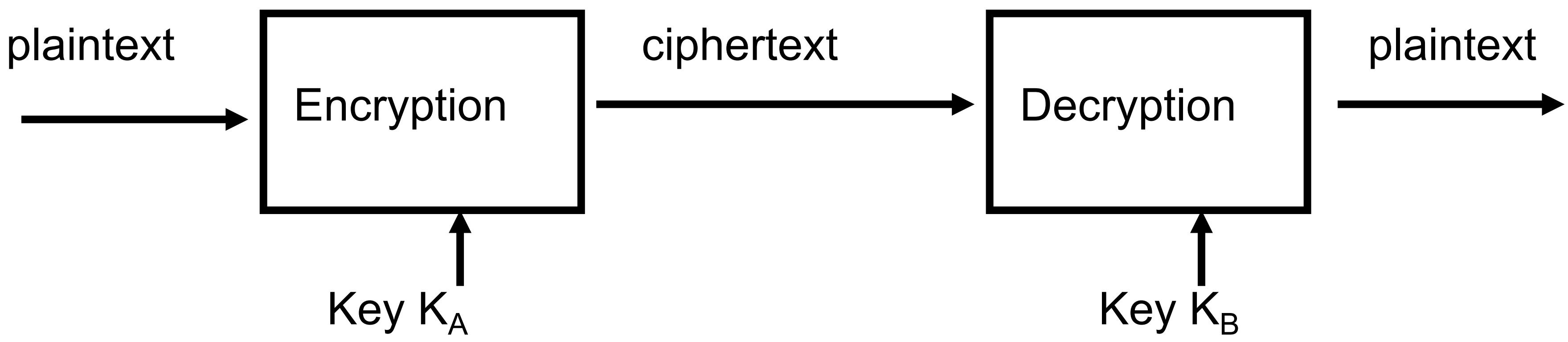
1. ***Authorization*** **– managing access to resources**
2. ***Confidentiality*** **– only allow authorized viewing of data - encrypting files and communication**
3. ***Authentication*** **– proving you are who you say you are**
4. ***Data Integrity*** **– detecting tampering with digital data**
5. ***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**

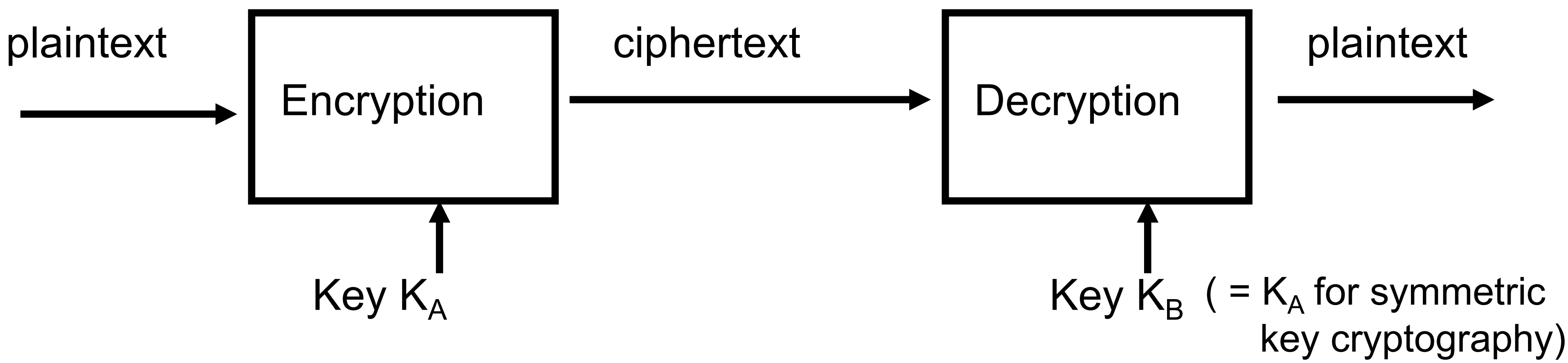
Confidentiality

- **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 $A = g^a \bmod p$
Send A, g, and p in the clear

A, g, p

Host Y

Choose b
Calculate $B=g^b \bmod p$
Send B in the clear

B

$K = B^a \bmod p$

$K = A^b \bmod p$

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

Diffie-Hellman Key Exchange

Host X

Choose a, g, and p
Calculate $A = g^a$ mod p
Send A, g, and p in the clear

A, g, p

Host Y

Choose b
Calculate $B = g^b$ mod p
Send B in the clear

B

$K = B^a$ mod p

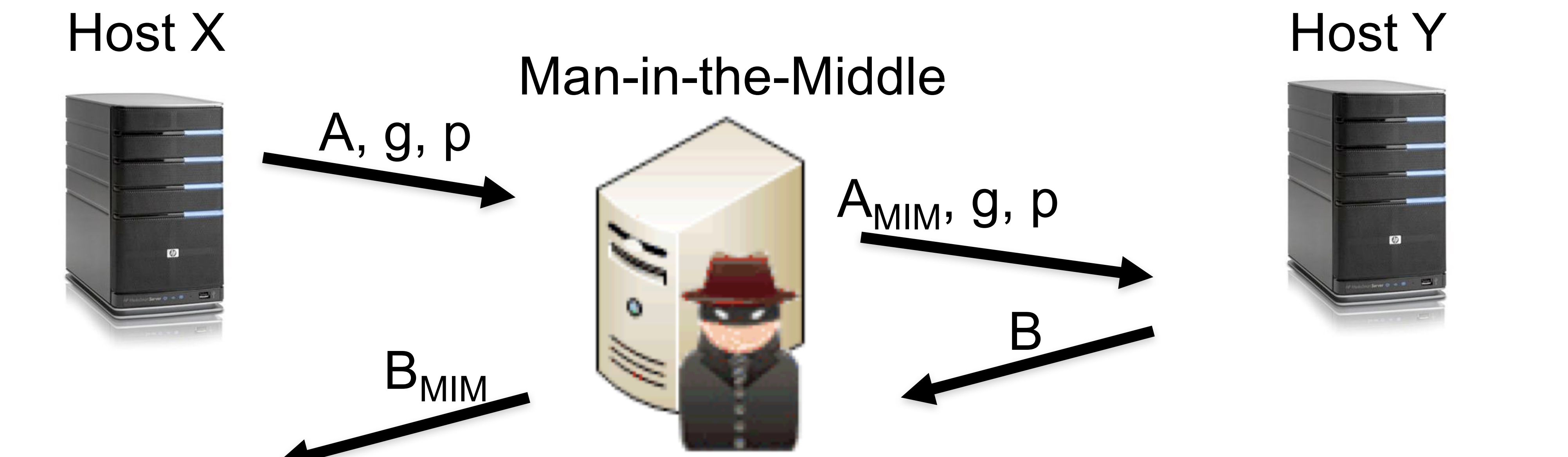
$K = A^b$ mod p

- Even if an attacker knows A, g, p, B, & algorithm $f(x) = g^x$ 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



$K = B_{MIM}^{a} \bmod p$

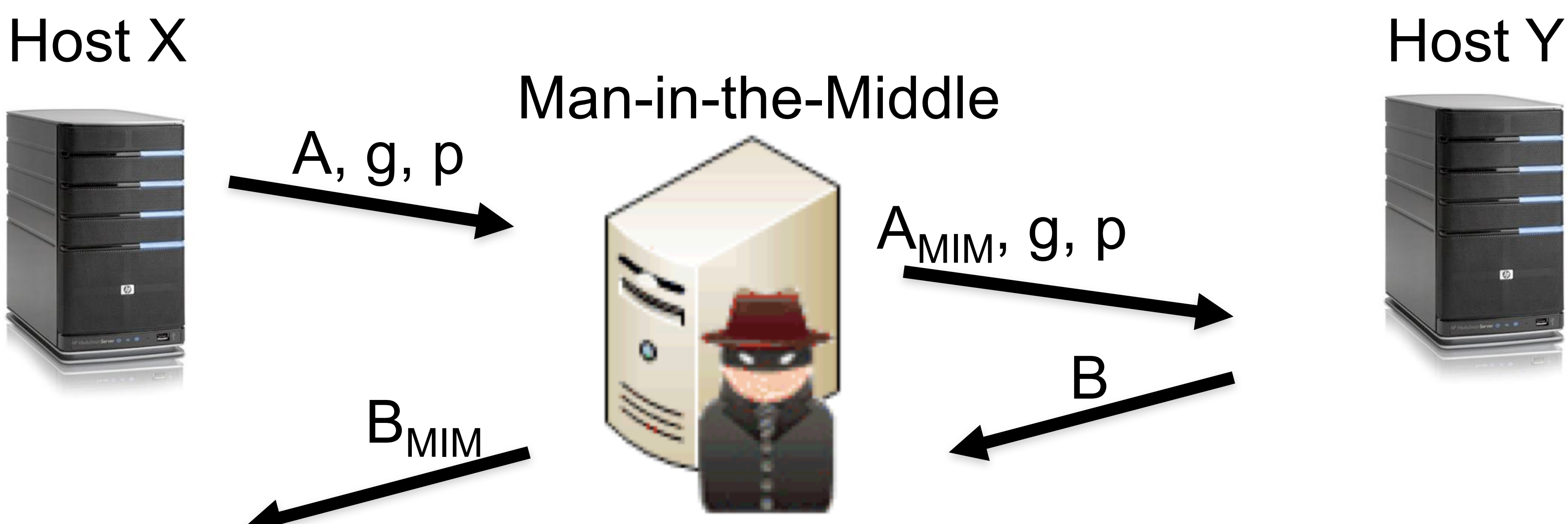
$K = A^{bMIM} \bmod p$

$K' = B^{aMIM} \bmod p$

$K' = A_{MIM}^{b} \bmod p$

- 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



$K = B_{MIM}^{a} \bmod p$

$K = A^{bMIM} \bmod p$

$K' = B^{aMIM} \bmod p$

$K' = A_{MIM}^{b} \bmod p$

- 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 (next slides)

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder