ECE560
Computer and Information Security

Fall 2020

User Authentication and Access Control

Tyler Bletsch

Duke University

User Authentication

Determining if a user is who they say they are before giving them access.

The four means of authenticating user identity are based on:

Something the individual knows	Something the individual possesses (token)	Something the individual is (static biometrics)	Something the individual does (dynamic biometrics)
• Password, PIN, answers to prearranged questions	• Smartcard, electronic keycard, physical key	• Fingerprint, retina, face	• Voice pattern, handwriting, typing rhythm

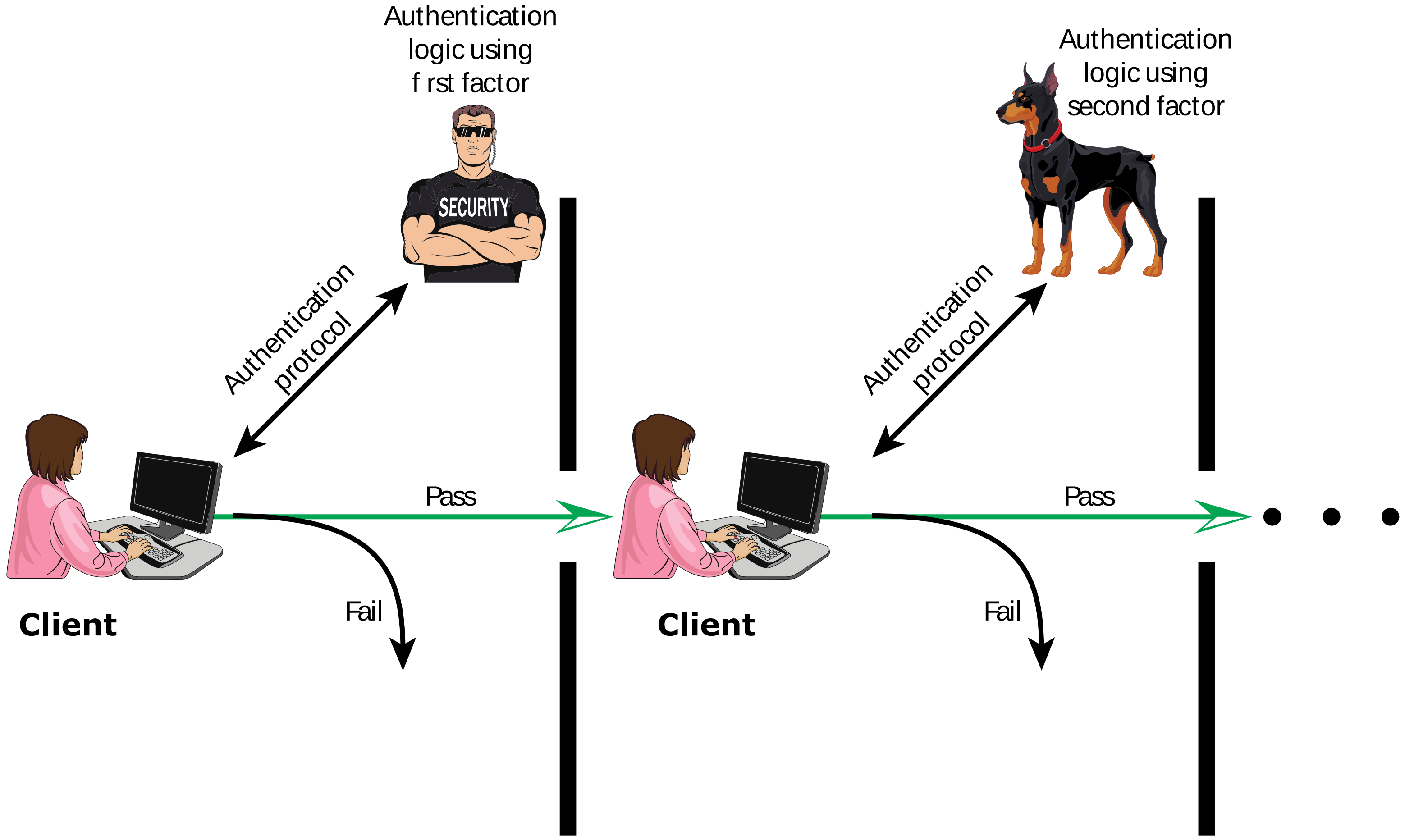


Figure 3.2 Multifactor Authentication

The four means of authenticating user identity are based on:

Something the individual knows	Something the individual possesses (token)	Something the individual is (static biometrics)	Something the individual does (dynamic biometrics)
• Password, PIN, answers to prearranged questions	• Smartcard, electronic keycard, physical key	• Fingerprint, retina, face	• Voice pattern, handwriting, typing rhythm

Password-Based Authentication

- Widely used line of defense against intruders
  - User provides name/login and password
  - System compares password with the one stored for that specified login
- The user ID:
  - Determines that the user is authorized to access the system
  - Determines the user's privileges
  - Is used in discretionary access control

Hash

- Threat model:
  - Database of password hashes is compromised (happens a LOT)
  - Attacker wants to figure out password

- Hashing:
  - Don’t store the plaintext password, store a hash
  - Compare hashes

- Why?
  - So the attacker can’t just look at the database and see passwords

Salt

- Threat model:
  - Database of password hashes is compromised (happens a LOT)
  - Attacker wants to figure out password for a given hash

- Salting:
  - Add a bit of random stuff (“salt”) to password before hashing
  - Random stuff differs per record
  - Store the salt with the hash so we can use it when verifying given passwords

- Why?
  - If I hash many possible passwords and find that “c00ldude” hashes to a53d677656e7bcb216b9ef6e38bb7ab1, then *anyone* with that hash must have that password
  - With a salt, I need to brute-force search *per user* instead of *once-for-everyone*

Iteration count

- Threat model:
  - Database of password hashes is compromised (happens a LOT)
  - Attacker wants to figure out password for a given hash
  - Attacker has lots of fast computers

- Iteration count:
  - Instead of just using H(data), do H(H(H(… H(data) …)))
  - Increase iteration count to make it very hard for attacker while still being feasible for login checks
  - Makes our hash function “slow” (configurably so!)

- Why?
  - If default hashing has speed of X, then an iteration count of 1000 gives a speed of X/1000. Login is a tiny amount of time in normal use, but it makes the attacker’s job 1000x harder for very little cost.

Password Vulnerabilities

- **Offline dictionary attack** (e.g., cracking a hashed password)
  - Defense: Make harder by salting, iteration count
- **Specific account attack** (e.g., dictionary attack on account)
  - Defense: Max attempt counter, password complexity requirements
- **Popular password attack** (try few passwords on many accounts)
  - Defense: Password complexity requirements
- **Password guessing against single user** (do research then guess)
  - Defense: User training, password complexity requirements
- **Workstation hijacking** (physically use logged-in workstation)
  - Defense: Physical security, auto-lock timers
- **Exploiting user mistakes** (Post-Its, sharing, unchanged defaults, ...)
  - Defense: Training, single-use expiring passwords for new accounts
- **Exploiting multiple password use**
  - Defense for individual: Password managers with strong crypto
  - Defense for organization: ?????
- **Electronic monitoring** (sniffing network, keylogger, etc.)
  - Defense: Encryption, challenge-response schemes, training

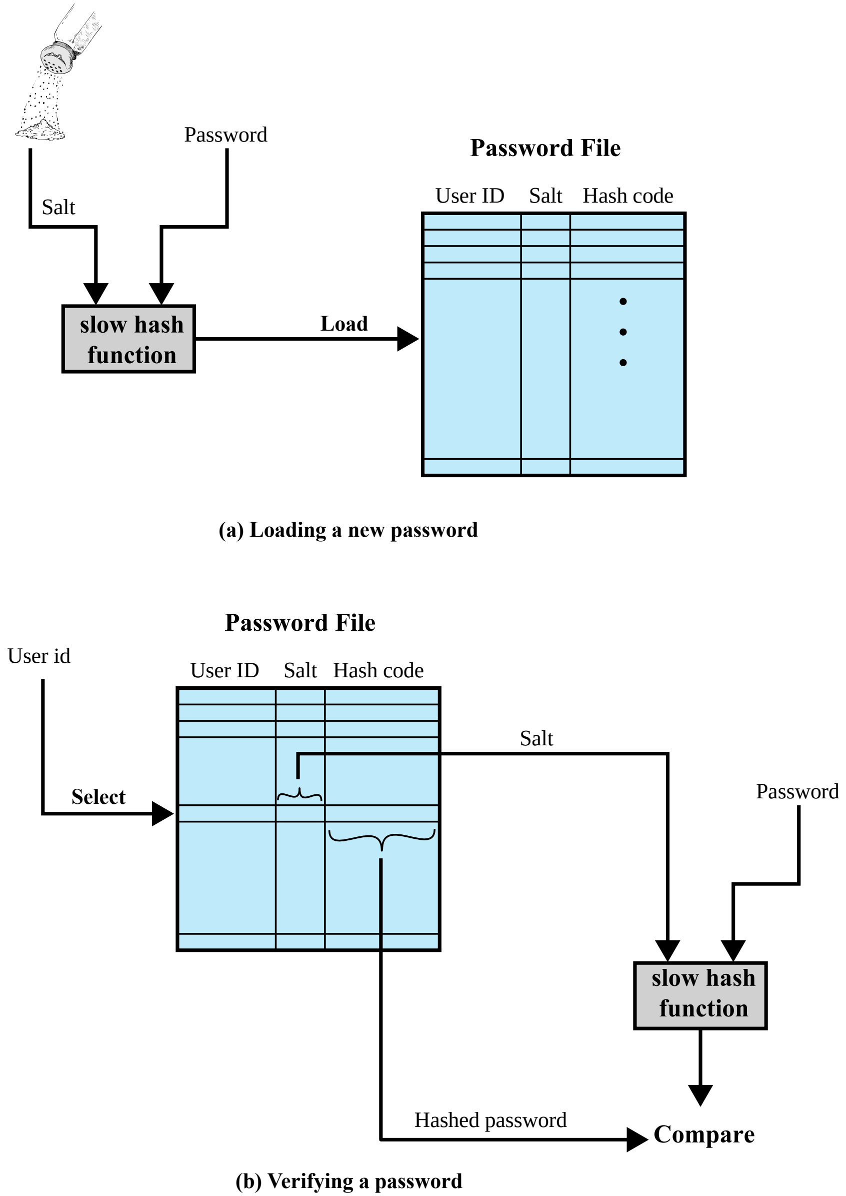


Figure 3.3 UNIX Password Scheme

Evolution of UNIX scheme

- Originally: hash stored in public-readable /etc/passwd file
- Now: hash stored in separate root-readable /etc/shadow file

- Originally: small hash, few iterations
- Later: MD5 hash, more iterations
- Now: SHA 512 hash, configurable iterations

Password Cracking

- Dictionary attacks
  - Develop a large dictionary of possible passwords and try each against the password file
  - Each password must be hashed using each salt value and then compared to stored hash values
- Rainbow table attacks
  - Pre-compute tables of hash values for all salts
  - A mammoth table of hash values
  - Can be countered by using a sufficiently large salt value and a sufficiently large hash length
- Password crackers exploit the fact that people choose easily guessable passwords
  - Shorter password lengths are also easier to crack

Storing passwords correctly

- Storing password plaintext (or encrypted)

T-Mobile [Link](#)
SONY [Link](#)
Adobe [Link](#)

- Storing hashed password

Windows [Link](#)
LinkedIn [Link](#)

- Storing salted hash of password

DISQUS [Link](#)

- Hash function has iteration count

I couldn't find anyone who bothered to do this yet didn't just use one of the functions below

- Just use PBKDF2, scrypt, bcrypt, etc.

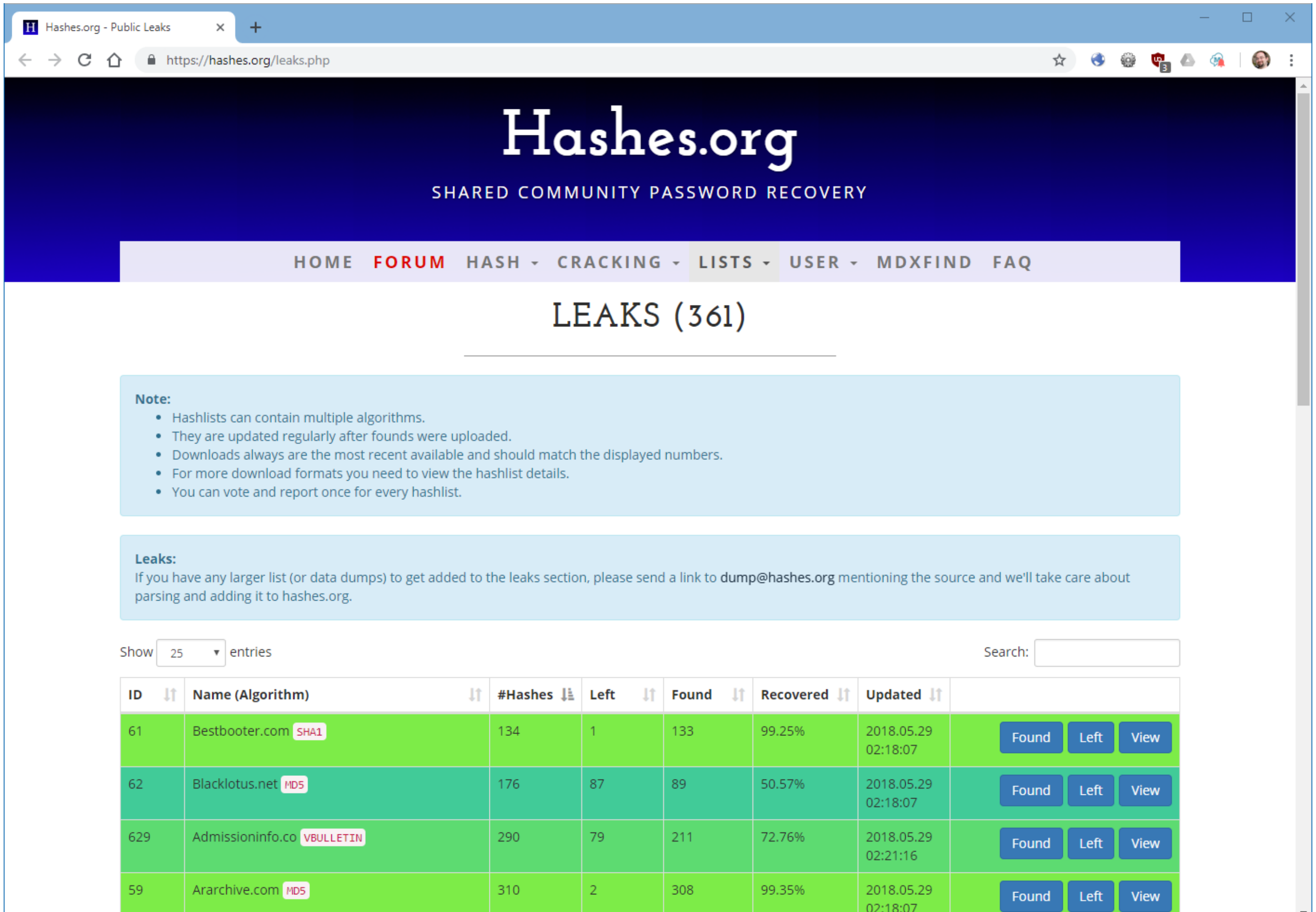
Dropbox [Link](#)
Linux [Link](#)

- Have a user management library handle it

django
node.js

Where do stolen hashes go?

- Attacker uses directly, sells on black market, or they leak
- Often, eventually, they hit the public internet:



Hashes.org - Public Leaks

https://hashes.org/leaks.php

Hashes.org

SHARED COMMUNITY PASSWORD RECOVERY

HOME FORUM HASH CRACKING LISTS USER MDXFIND FAQ

LEAKS (361)

Note:

- Hashlists can contain multiple algorithms.
- They are updated regularly after founds were uploaded.
- Downloads always are the most recent available and should match the displayed numbers.
- For more download formats you need to view the hashlist details.
- You can vote and report once for every hashlist.

Leaks:
If you have any larger list (or data dumps) to get added to the leaks section, please send a link to dump@hashes.org mentioning the source and we'll take care about parsing and adding it to hashes.org.

Show 25 entries Search:

ID	Name (Algorithm)	#Hashes	Left	Found	Recovered	Updated	
61	Bestbooter.com SHA1	134	1	133	99.25%	2018.05.29 02:18:07	Found Left View
62	Blacklotus.net MD5	176	87	89	50.57%	2018.05.29 02:18:07	Found Left View
629	Admissioninfo.co VBULLETIN	290	79	211	72.76%	2018.05.29 02:21:16	Found Left View
59	Ararchive.com MD5	310	2	308	99.35%	2018.05.29 02:18:07	Found Left View

Importance of password storage illustrated (1)

- Plaintext passwords: 100% are “recovered” by attacker (obviously)
- Sorted hashes.org by “percent recovered” – all are unsalted!

ID	Name (Algorithm)	#Hashes	Left	Found	Recovered	Updated	
780	Pingpong.su MD5	32'394	0	32'394	100%	2018.05.31 19:45:34	Found Left View
606	Shadi.com SHA1	1'136'091	35	1'136'056	100%	2018.09.28 11:57:53	Found Left View
35	Zoosk.com MD5	29'013'020	266	29'012'754	100%	2018.09.10 13:08:06	Found Left View
70	Have I been Pwned V1 SHA1	320'294'464	75'523	320'218'941	99.98%	2018.09.25 13:34:22	Found Left View
26	Op Northkorea MD5	6'393	4	6'389	99.94%	2018.05.29 02:18:03	Found Left View
698	Fon MD5	85'033	84	84'949	99.9%	2018.09.12 14:41:54	Found Left View

- Scroll to lower percent – almost all are salted.

849	Xronize.com MYBB	43'795	17'106	26'689	60.94%	2018.09.14 16:58:06	Found Left View
783	Politicalforum.com VBULLETIN	31'588	12'396	19'192	60.76%	2018.09.01 08:56:03	Found Left View
115	DayZ.com MYBB/IPB	208'236	81'736	126'500	60.75%	2018.05.29 02:18:30	Found Left View
630	Adult-forum.org VBULLETIN	7'853	3'094	4'759	60.6%	2018.08.28 18:42:52	Found Left View
812	Snowandmud.com VBULLETIN	53'722	21'259	32'463	60.43%	2018.09.01 08:56:03	Found Left View
660	Bodyweb.com VBULLETIN	79'696	31'800	47'896	60.1%	2018.09.01 08:55:58	Found Left View
625	vectorlinux.com SHA1(SALTPLAIN)	18'343	7'402	10'941	59.65%	2018.05.29 02:21:16	Found Left View

Importance of password storage illustrated (2)

- Scroll to very low percentages...most use bcrypt or similar, which has an iteration count

971	Forum.lightshope.org BCRYPT	28'721	28'570	151	0.53%	2018.08.17 12:26:32	Found Left View
802	Scufgaming.com WORDPRESS / MD5	2'809	2'801	8	0.28%	2018.05.30 00:34:17	Found Left View
778	Pesfan.com VBULLETIN	426'495	425'704	791	0.19%	2018.08.28 08:28:50	Found Left View
558	Dailymotion BCRYPT	16'147'134	16'139'263	7'871	0.05%	2018.05.29 02:20:48	Found Left View
810	Siriusforum.net BCRYPT	1'284	1'284	0	0%	2018.05.29 16:17:02	Found Left View
751	Legion.cm BCRYPT	23'113	23'113	0	0%	2018.05.29 02:21:30	Found Left View
749	Krolop-gerst.com BCRYPT	27'748	27'748	0	0%	2018.05.29 02:21:30	Found Left View
972	Totaljerkface.com BCRYPT	188'055	188'055	0	0%	2018.08.16 23:58:37	Found Left View

- **Conclusion: How you store password has HUGE effect on what happens if (when) they are breached!**

Password Selection Strategies

- **User education**
  - Users can be told the importance of using hard to guess passwords and can be provided with guidelines for selecting strong passwords
- **Computer generated passwords**
  - Users have trouble remembering them (good for single-use, bad for long-term)
- **Reactive password checking**
  - System periodically runs its own password cracker to find guessable passwords
- **Complex password policy**
  - User is allowed to select their own password, however the system checks to see if the password is allowable, and if not, rejects it
  - Goal is to eliminate guessable passwords while allowing the user to select a password that is memorable

The four means of authenticating user identity are based on:

Something the individual knows

- Password, PIN, answers to prearranged questions

Something the individual possesses (token)

- Smartcard, electronic keycard, physical key

Something the individual is (static biometrics)

- Fingerprint, retina, face

Something the individual does (dynamic biometrics)

- Voice pattern, handwriting, typing rhythm

Table 3.3

Card Type	Defining Feature	Example
Embossed	Raised characters only, on front	Old credit card
Magnetic stripe	Magnetic bar on back, characters on front	Bank card
Memory	Electronic memory inside	Prepaid phone card
Smart Contact Contactless	Electronic memory and processor inside Electrical contacts exposed on surface Radio antenna embedded inside	Biometric ID card

Types of Cards Used as Tokens

Memory Cards

- Can store but do not process data
- The most common is the magnetic stripe card
- Can include an internal electronic memory
- Can be used alone for physical access
  - Hotel room
  - ATM
- Provides significantly greater security when combined with a password or PIN
- Drawbacks of memory cards include:
  - Requires a special reader
  - Loss of token
  - User dissatisfaction

Smart Tokens

- Physical characteristics:
  - Include an embedded microprocessor
  - A smart token that looks like a bank card
  - Can look like calculators, keys, small portable objects
- User interface:
  - Manual interfaces include a keypad and display for human/token interaction
- Electronic interface
  - A smart card or other token requires an electronic interface to communicate with a compatible reader/writer
  - Contact and contactless interfaces
- Authentication protocol:
  - Classified into three categories:
    - Static
    - Dynamic password generator
    - Challenge-response

Smart Cards

- Most important category of smart token
  - Has the appearance of a credit card
  - Has an electronic interface
  - May use any of the smart token protocols
- Contain:
  - An entire microprocessor
    - Processor
    - Memory
    - I/O ports
- Typically include three types of memory:
  - Read-only memory (ROM)
    - Stores data that does not change during the card's life
  - Electrically erasable programmable ROM (EEPROM)
    - Holds application data and programs
  - Random access memory (RAM)
    - Holds temporary data generated when applications are executed

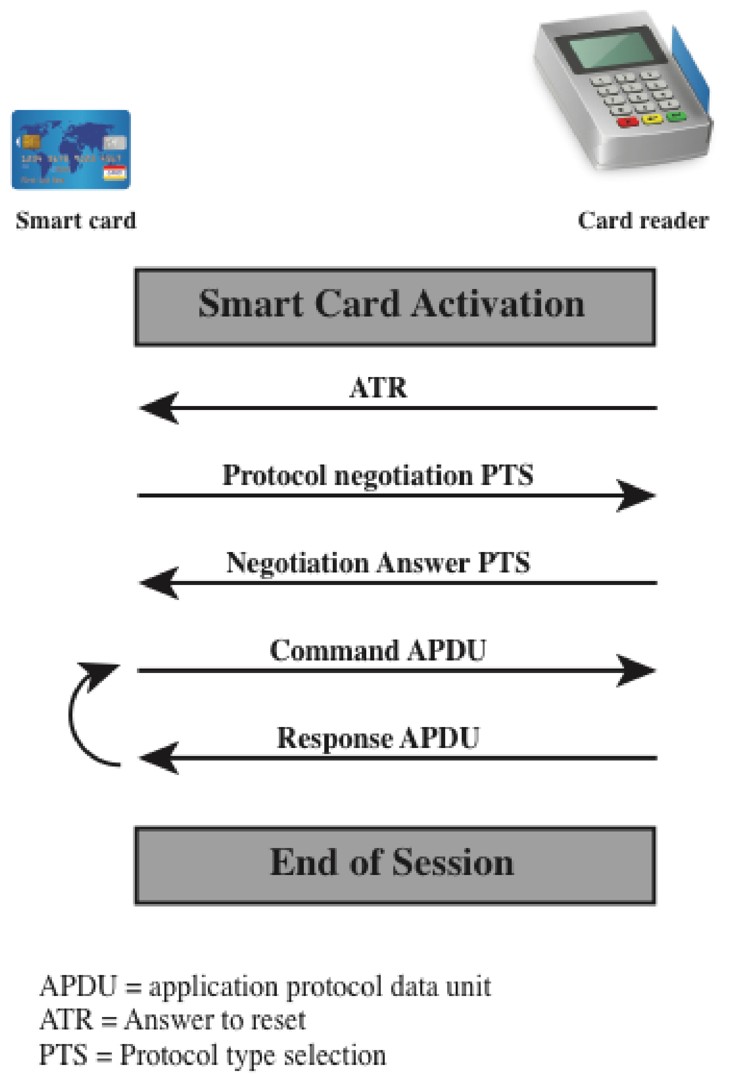


Figure 3.6 Smart Card/Reader Exchange

The four means of authenticating user identity are based on:

Something the individual knows

- Password, PIN, answers to prearranged questions

Something the individual possesses (token)

- Smartcard, electronic keycard, physical key

Something the individual is (static biometrics)

- Fingerprint, retina, face

Something the individual does (dynamic biometrics)

- Voice pattern, handwriting, typing rhythm

Biometric Authentication

- Attempts to authenticate an individual based on unique physical characteristics
- Based on pattern recognition
- Is technically complex and expensive when compared to passwords and tokens
- Physical characteristics used include:
    - Facial characteristics
    - Fingerprints
    - Hand geometry
    - Retinal pattern
    - Iris
    - Signature
    - Voice

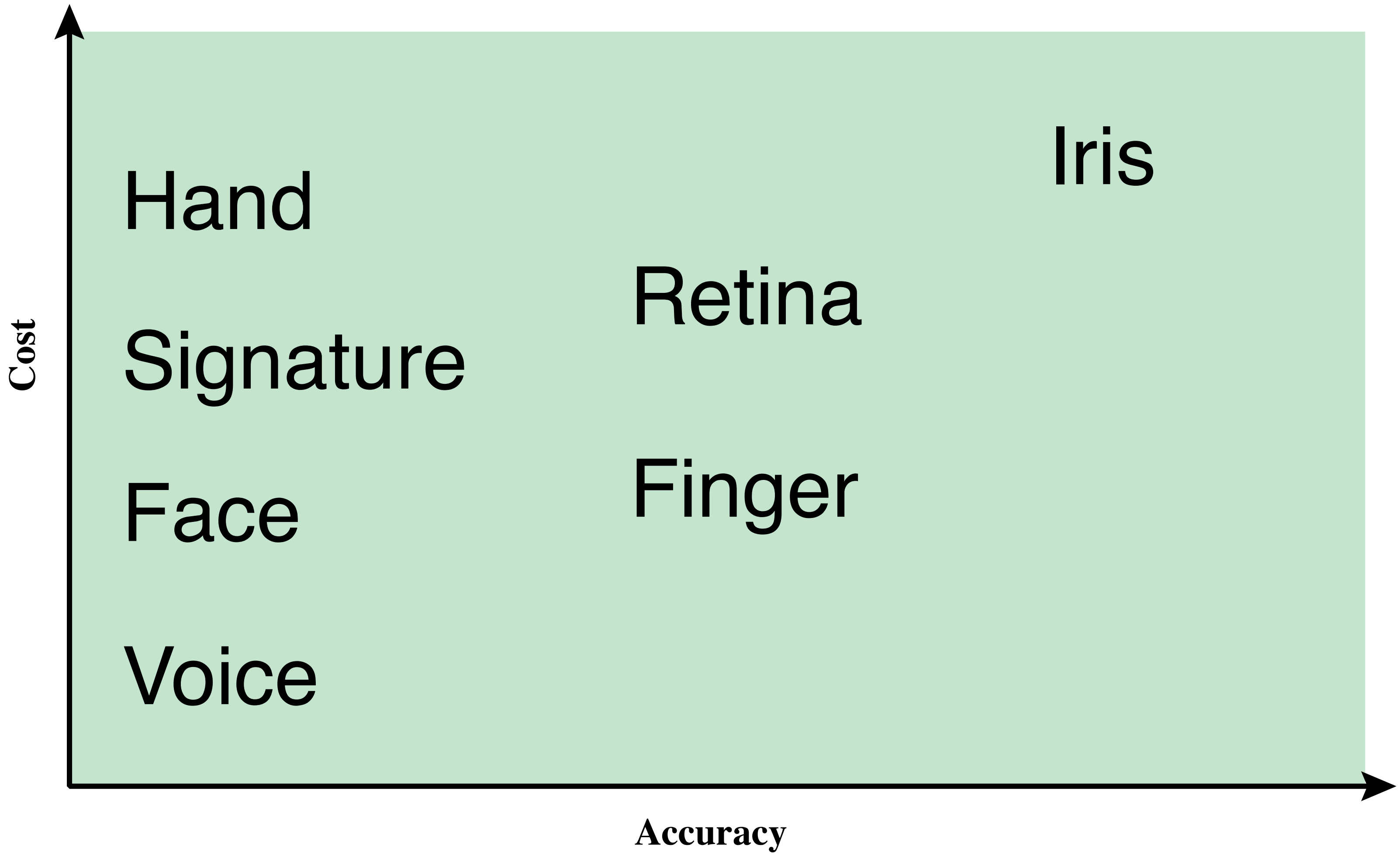


Figure 3.8 Cost Versus Accuracy of Various Biometric Characteristics in User Authentication Schemes.

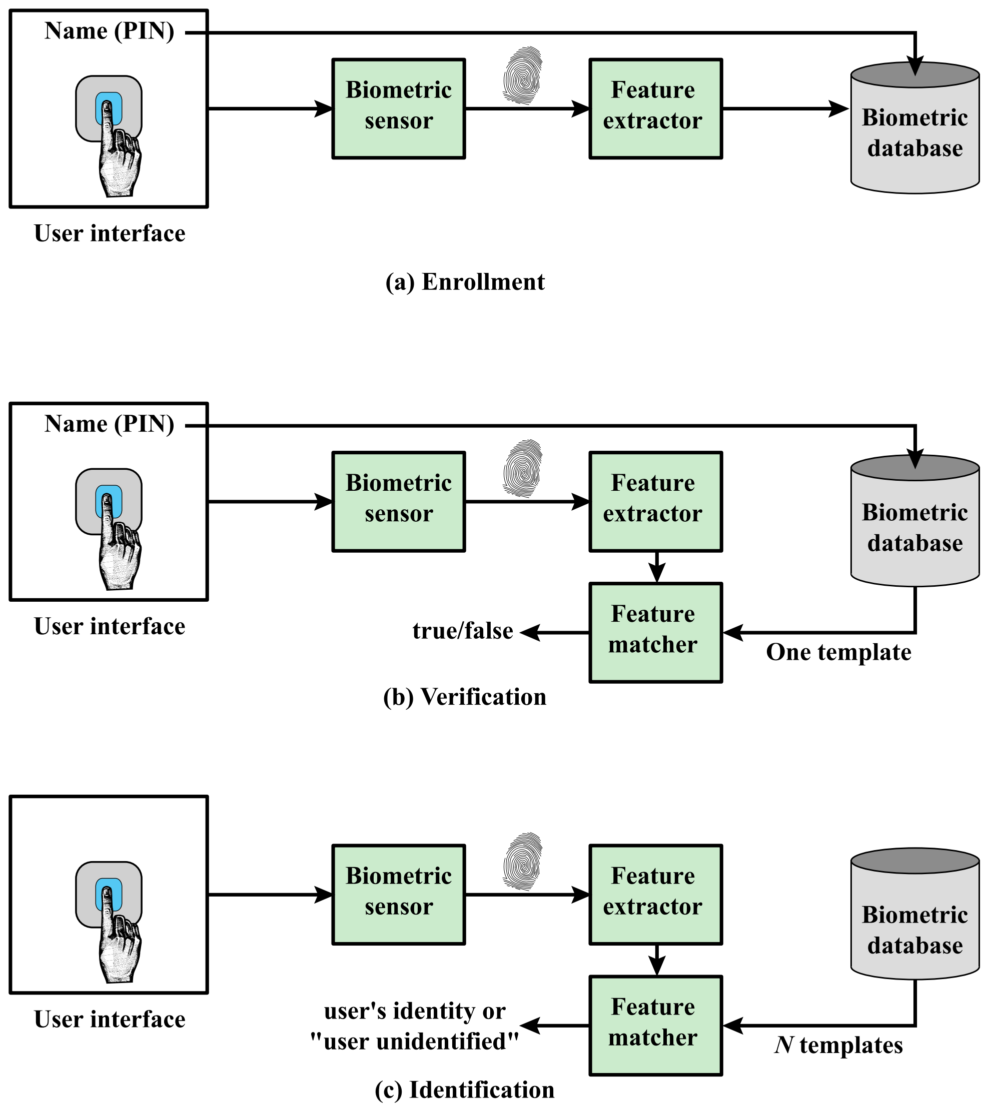


Figure 3.9 A Generic Biometric System. Enrollment creates an association between a user and the user's biometric characteristics. Depending on the application, user authentication either involves verifying that a claimed user is the actual user or identifying an unknown user.

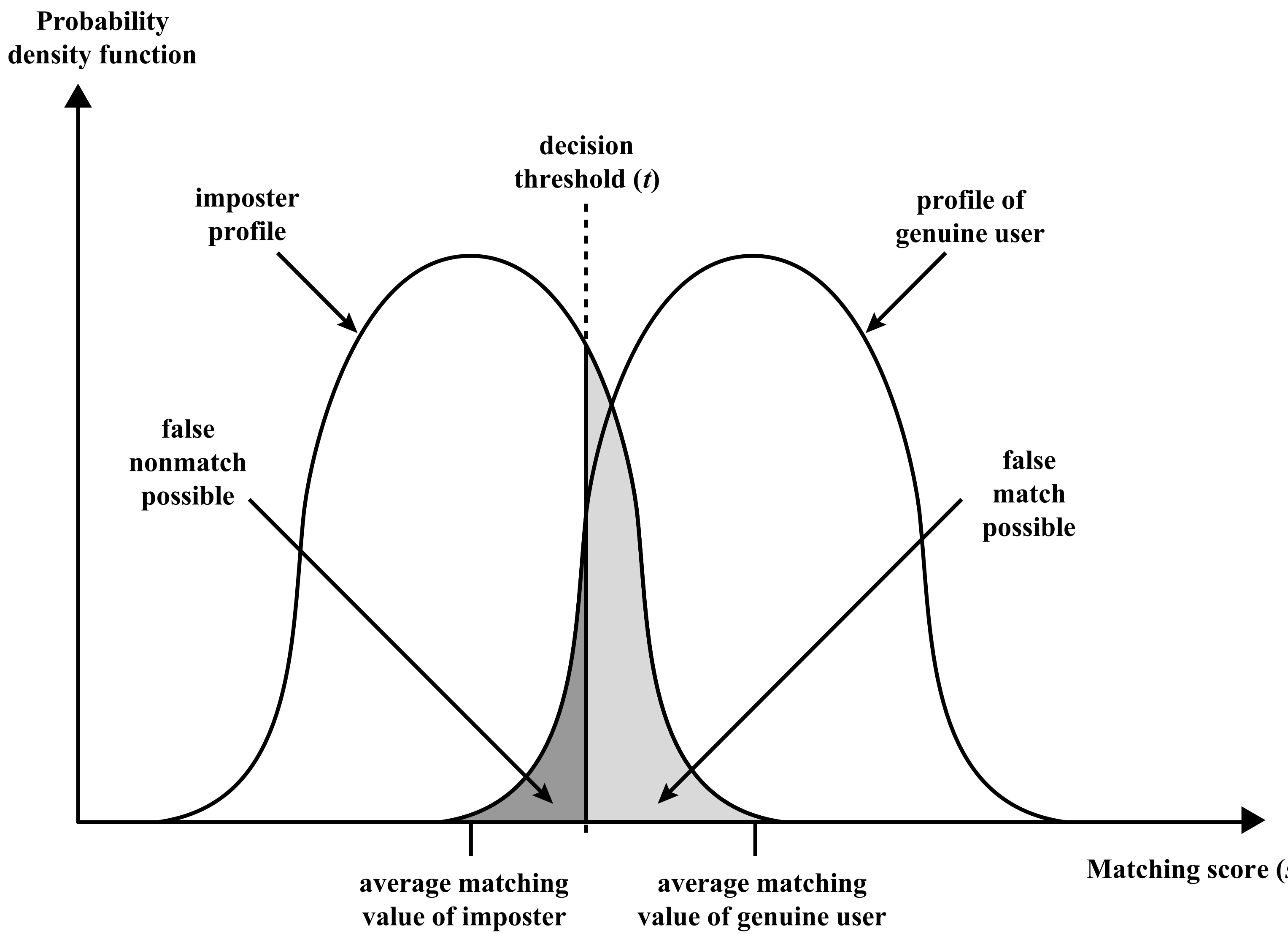


Figure 3.10 Profiles of a Biometric Characteristic of an Imposter and an Authorized Users In this depiction, the comparison between presented feature and a reference feature is reduced to a single numeric value. If the input value (s) is greater than a preassigned threshold (t), a match is declared.

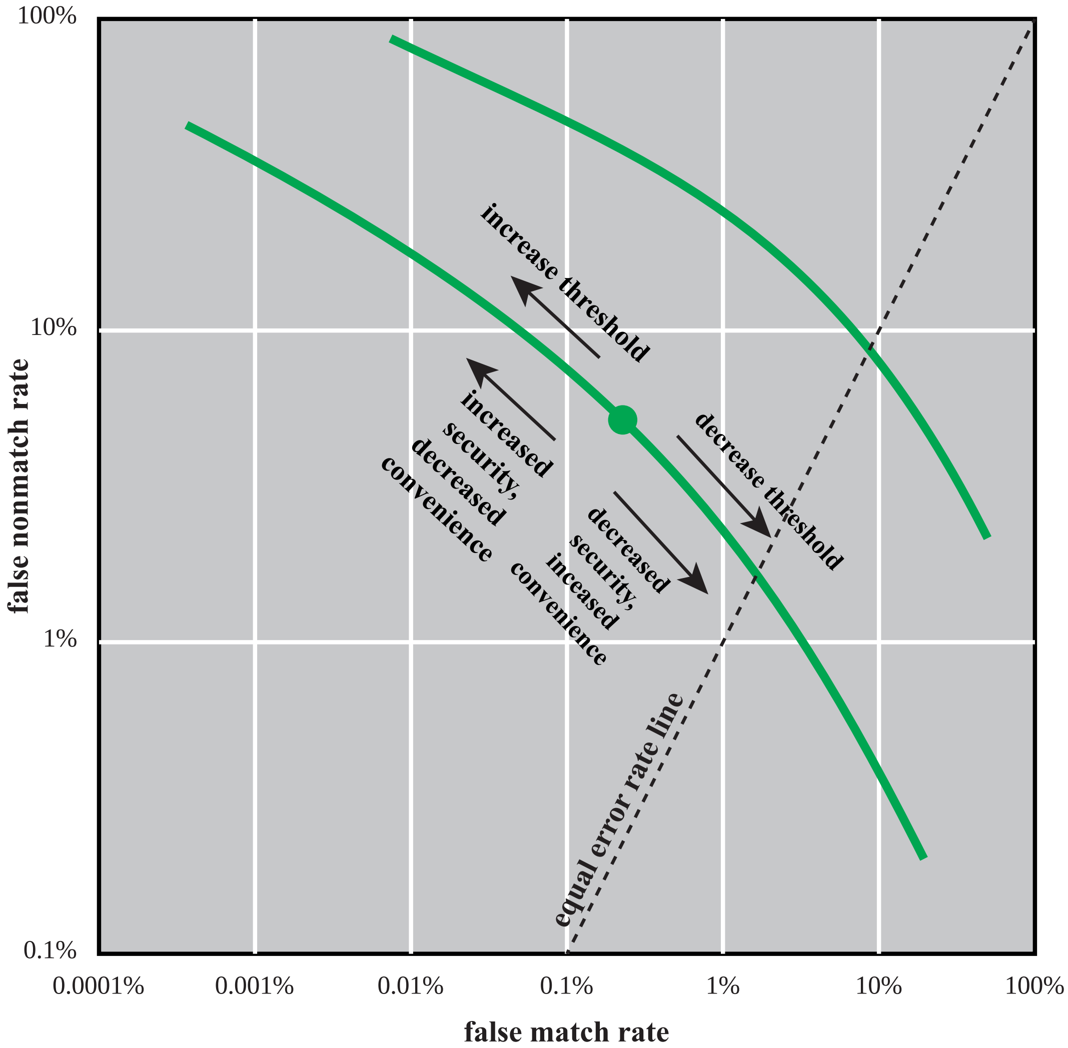


Figure 3.11 Idealized Biometric Measurement Operating Characteristic Curves (log-log scale)

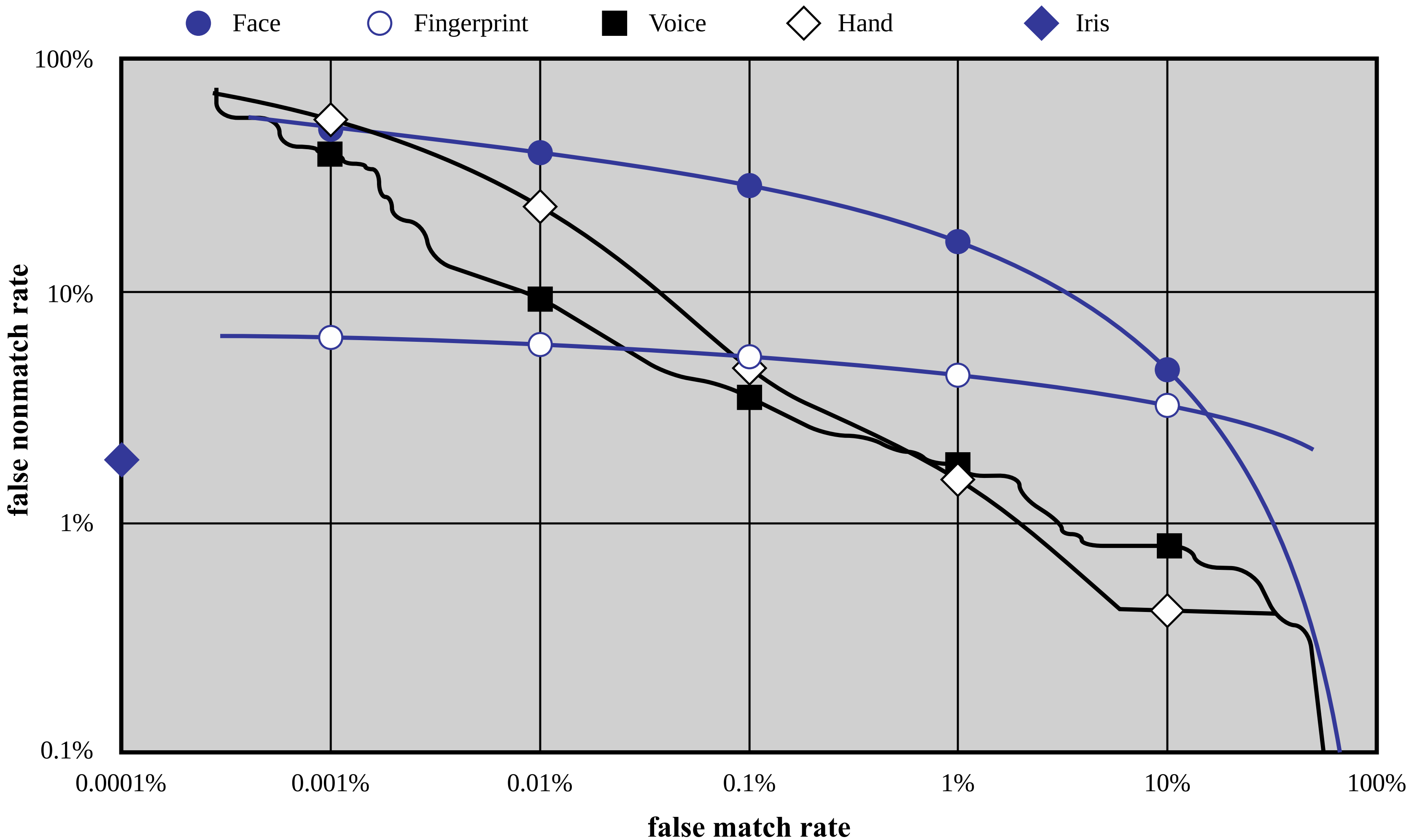


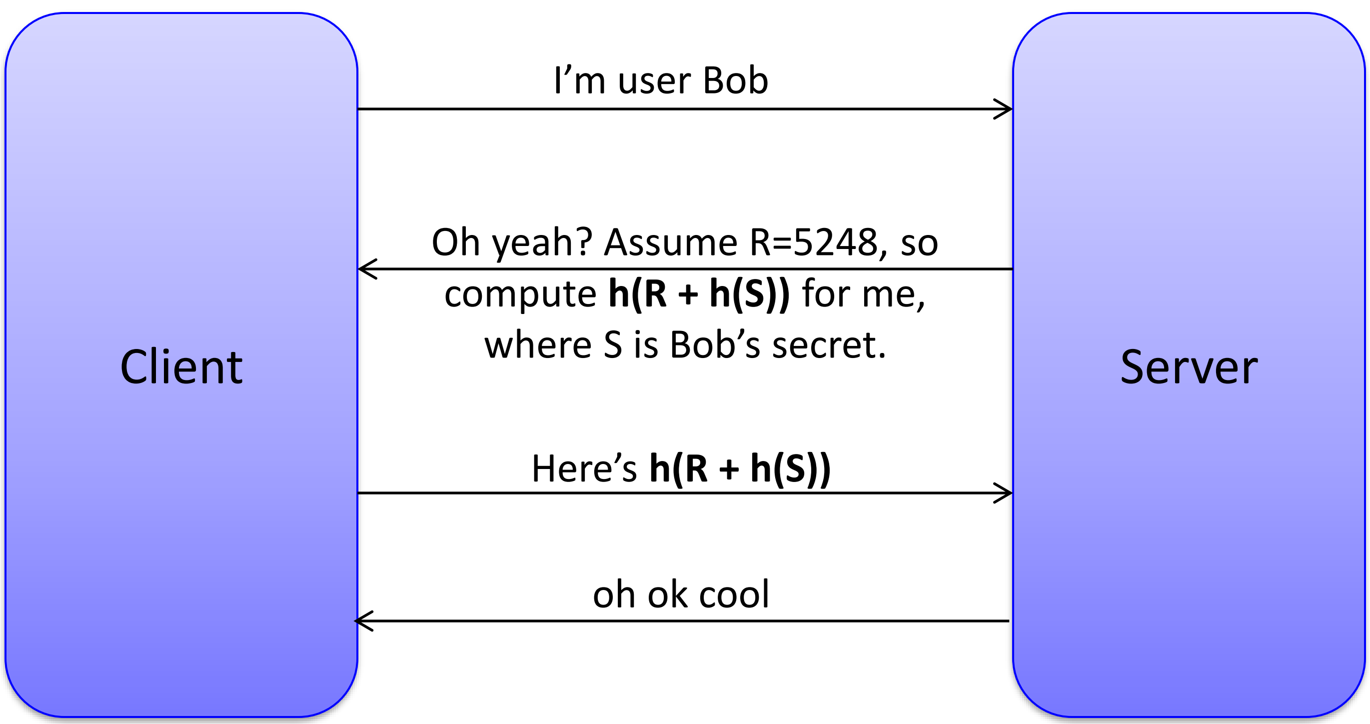
Figure 3.12 Actual Biometric Measurement Operating Characteristic Curves, reported in [MANS01]. To clarify differences among systems, a log-log scale is used.

Remote User Authentication

- Authentication over a network, the Internet, or a communications link is more complex
- Additional security threats such as:
  - Eavesdropping, capturing a password, replaying an authentication sequence that has been observed
- Generally rely on some form of a challenge-response protocol to counter threats

Challenge-Response scheme

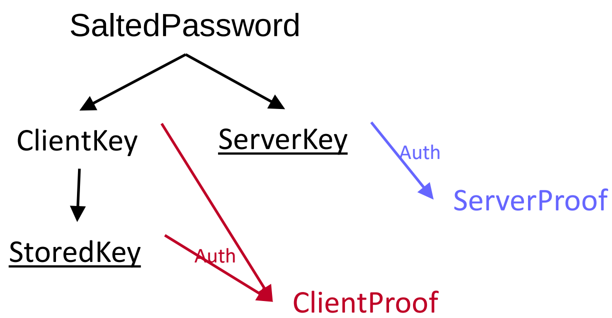
- Assume we have some authentication secret S
  - Token value, biometric signature, etc...
- Don't want to send it (*or even its hash!*)
- Instead, server issues a *challenge* (random value *R*) to client that can only be answered if it has S, but which doesn't reveal S.



Challenge-Response: What about passwords?

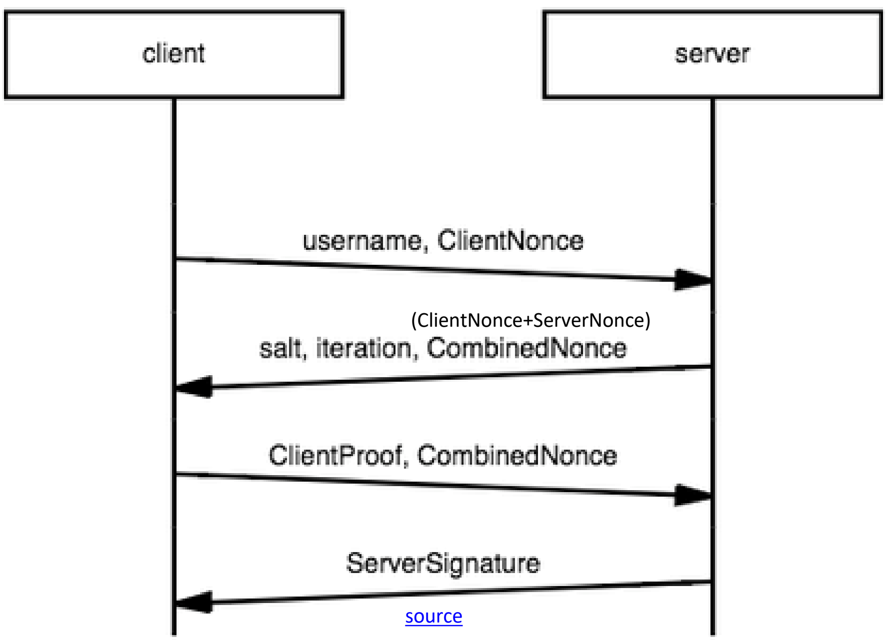
- In the scheme shown, if the password hash is leaked, it's *equivalent* to having the actual password, since we only need h(S)!
- Other challenge-response schemes avoid this issue, e.g. **Salted Challenge Response Authentication Mechanism** (**SCRAM**)

Mutations done to the salted password



Black = computed by server when account is created
Underline = stored by server
Red = computed by client during auth
Blue = computed by server during auth

Communications sequence



source

SaltedPassword = {salted hash of password}
ClientKey = HMAC(SaltedPassword, "Client Key")
StoredKey = H(ClientKey)
ServerKey = HMAC(SaltedPassword, "Server Key")

Auth = {username, salt, iteration, CombinedNonce}

ClientProof = ClientKey ^ HMAC(StoredKey, Auth)
ServerProof = HMAC(ServerKey, Auth)

For more, see Wikipedia or this article

Table 3.5

Some Potential Attacks, Susceptible Authenticators, and Typical Defenses

Attacks	Authenticators	Examples	Typical defenses
Client attack	Password	Guessing, exhaustive search	Large entropy; limited attempts
	Token	Exhaustive search	Large entropy; limited attempts, theft of object requires presence
	Biometric	False match	Large entropy; limited attempts
Host attack	Password	Plaintext theft, dictionary/exhaustive search	Hashing; large entropy; protection of password database
	Token	Passcode theft	Same as password; 1-time passcode
	Biometric	Template theft	Capture device authentication; challenge response
Eavesdropping, theft, and copying	Password	"Shoulder surfing"	User diligence to keep secret; administrator diligence to quickly revoke compromised passwords; multifactor authentication
	Token	Theft, counterfeiting hardware	Multifactor authentication; tamper resistant/evident token
	Biometric	Copying (spoofing) biometric	Copy detection at capture device and capture device authentication
Replay	Password	Replay stolen password response	Challenge-response protocol
	Token	Replay stolen passcode response	Challenge-response protocol; 1-time passcode
	Biometric	Replay stolen biometric template response	Copy detection at capture device and capture device authentication via challenge-response protocol
Trojan horse	Password, token, biometric	Installation of rogue client or capture device	Authentication of client or capture device within trusted security perimeter
Denial of service	Password, token, biometric	Lockout by multiple failed authentications	Multifactor with token

(Table is on page 96 in the textbook)

Access control

So you've proven who you are, but what are you allowed to do?

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC
  - ABAC
- Identity federation

Subjects, Objects, Actions, and Rights

Subject
(initiator)

- The thing making the request (e.g. the user)

Verb
(request)

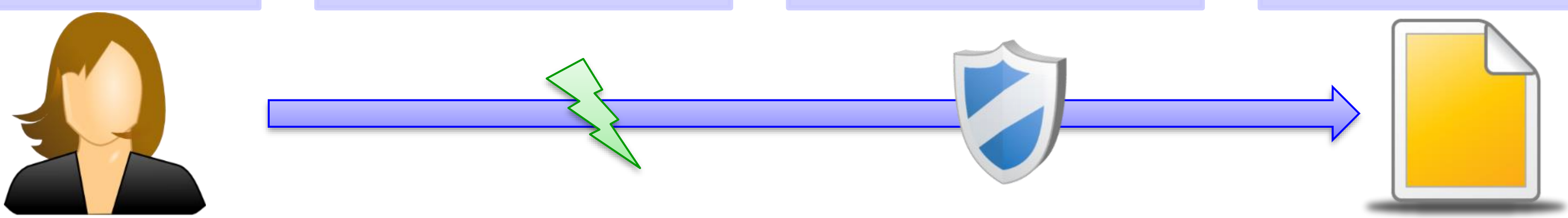
- The operation to perform (e.g., read, delete, etc.)

Right
(permission)

- A specific ability for the subject to do the action to the object.

Object
(target)

- The thing that's being hit by the request (e.g., a file).



Categories of Access Control Policies

- **Discretionary AC (DAC)**: There's a list of permissions attached to the subject or object (or possibly a giant heap of global rules).
- **Mandatory AC (MAC)**: Objects have classifications, subjects have clearances, subjects cannot give additional permissions.
  - An overused/abused term
- **Role-Based AC (RBAC)**: Subjects belong to roles, and roles have all the permissions.
  - The current Enterprise IT buzzword meaning "good" security
- **Attribute-Based AC (ABAC)**: Subjects and objects have attributes, rules engine applies predicates to these to determine access
  - Allows fine-grained expression
  - Usually complex, seldom implemented

Discretionary Access Control (DAC)

- Scheme in which an entity may enable another entity to access some resource
- Often provided using an access matrix
    - One dimension consists of identified subjects that may attempt data access to the resources
    - The other dimension lists the objects that may be accessed
- Each entry in the matrix indicates the access rights of a particular subject for a particular object

DAC model

```
bool IsActionAllowed(subject, object, action) {
  if (action ∈ get_permissions(subject,object))
    return true
}
```

Implementation

- Can use various data structures, none of which should surprise you

Matrix

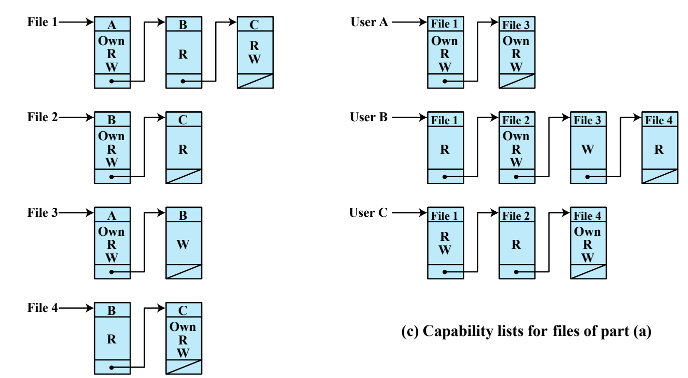
		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

(a) Access matrix

Flat list

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
A	Write	File 1
A	Own	File 3
A	Read	File 3
A	Write	File 3
B	Read	File 1
B	Own	File 2
B	Read	File 2
B	Write	File 2
B	Write	File 3
B	Read	File 4
C	Read	File 1
C	Write	File 1
C	Read	File 2
C	Own	File 4
C	Read	File 4
C	Write	File 4

Linked list



(b) Access control lists for files of part (a)

(c) Capability lists for files of part (a)

Figure 4.2 Example of Access Control Structures

UNIX File Access Control

UNIX files are administered using inodes (index nodes)

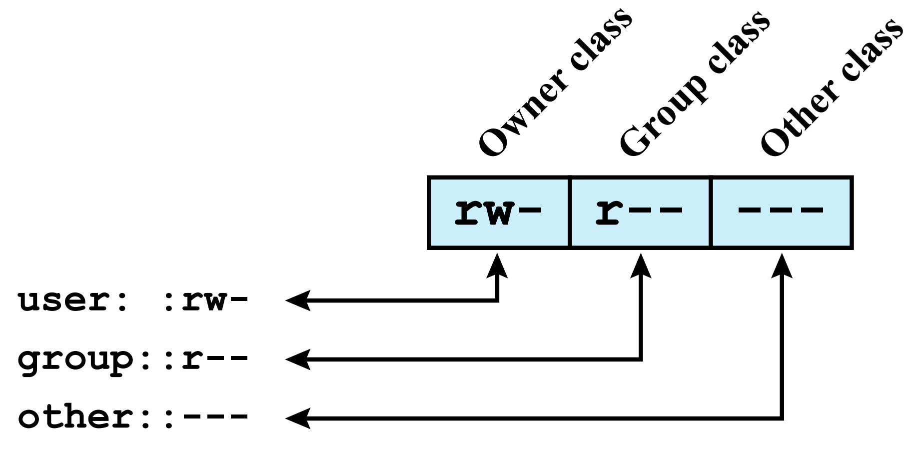
- Control structures with key information needed for a particular file
- Several file names may be associated with a single inode
- An active inode is associated with exactly one file
- File attributes, permissions and control information are sorted in the inode
- On the disk there is an inode table, or inode list, that contains the inodes of all the files in the file system
- When a file is opened its inode is brought into main memory and stored in a memory resident inode table

Directories are structured in a hierarchical tree

- May contain files and/or other directories
- Contains file names plus pointers to associated inodes

UNIX
File Access Control

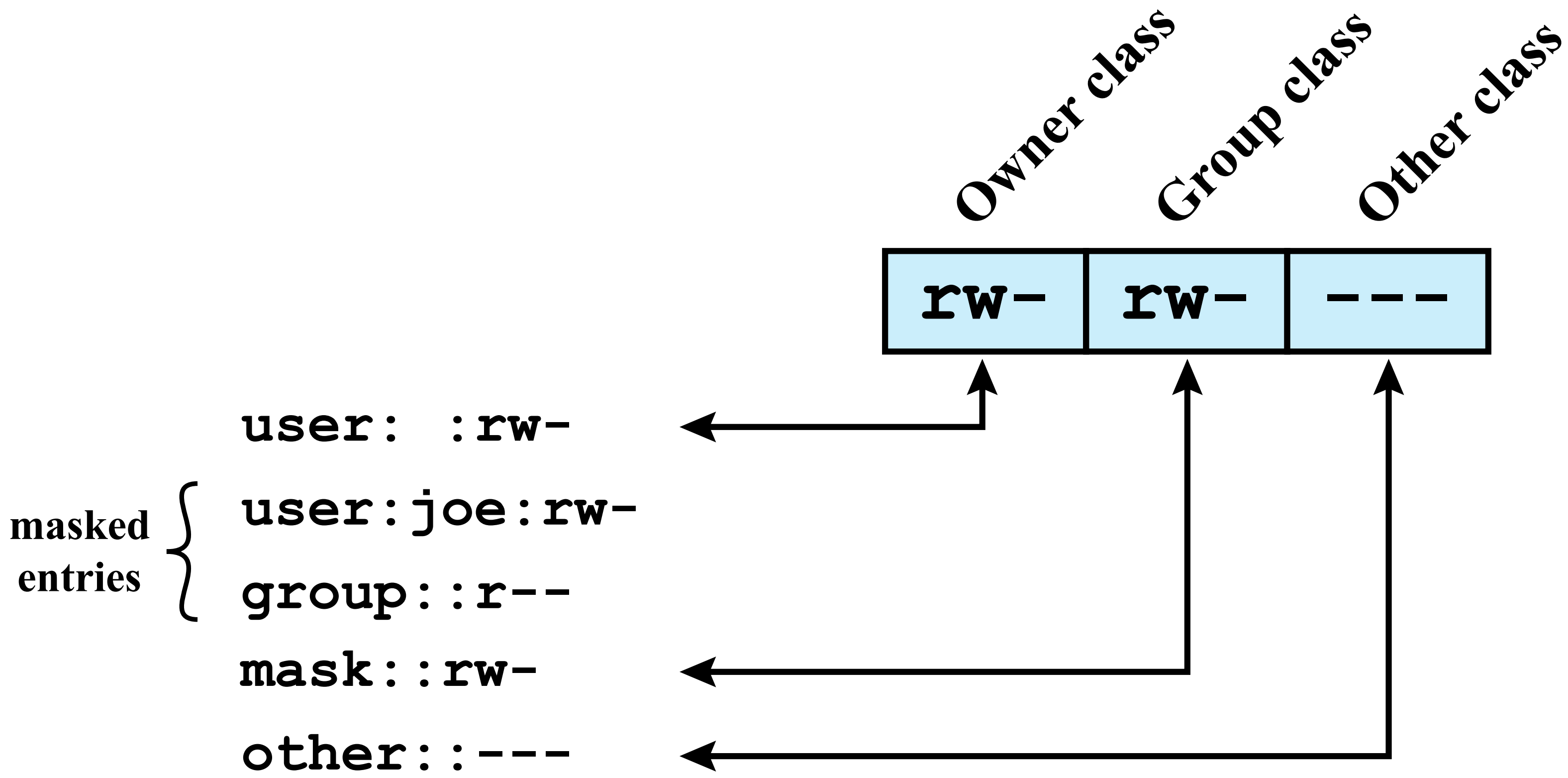
- Unique user identification number (user ID)
- Member of a primary group identified by a group ID
- Belongs to a specific group
- 12 protection bits
  - Specify read, write, and execute permission for the owner of the file, members of the group and all other users
- The owner ID, group ID, and protection bits are part of the file's inode



(a) Traditional UNIX approach (minimal access control list)

Traditional UNIX File Access Control

- “Set user ID”(SetUID)
- “Set group ID”(SetGID)
  - System temporarily uses rights of the file owner/group in addition to the real user’s rights when making access control decisions
  - Enables privileged programs to access files/resources not generally accessible
- Sticky bit
  - When applied to a directory it specifies that only the owner of any file in the directory can rename, move, or delete that file
- Superuser
  - Is exempt from usual access control restrictions
  - Has system-wide access



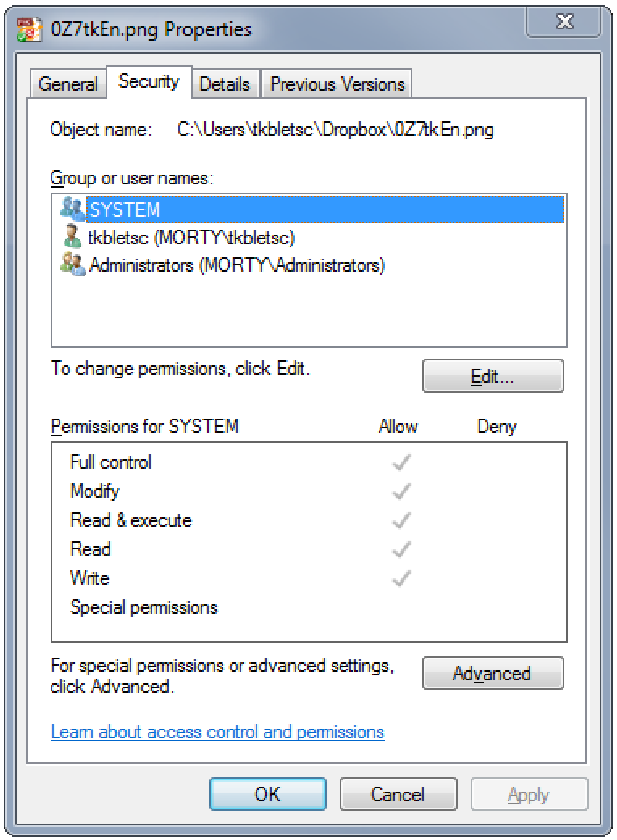
(b) Extended access control list

Figure 4.5 UNIX File Access Control

File system access control lists (ACLs)

- Arbitrary list of rules governing access per-file/directory
- More flexible than classic UNIX permissions, but more metadata to store/check

Windows ACL UI



Examples of Linux ACL commands

Set all permissions for user johny to file named "abc":

```
# setfacl -m "u:johny:rwx" abc
```

Check permissions

```
# getfacl abc
```

```
# file: abc
# owner: someone
# group: someone
user::rw-
user:johny:rwx
group::r--
mask::rwx
other::r--
```

Change permissions for user johny:

```
# setfacl -m "u:johny:r-x" abc
```

Check permissions

```
# getfacl abc
```

```
# file: abc
# owner: someone
# group: someone
user::rw-
user:johny:r-x
group::r--
mask::r-x
other::r--
```

Remove all extended ACL entries:

```
# setfacl -b abc
```

From Arch Wiki

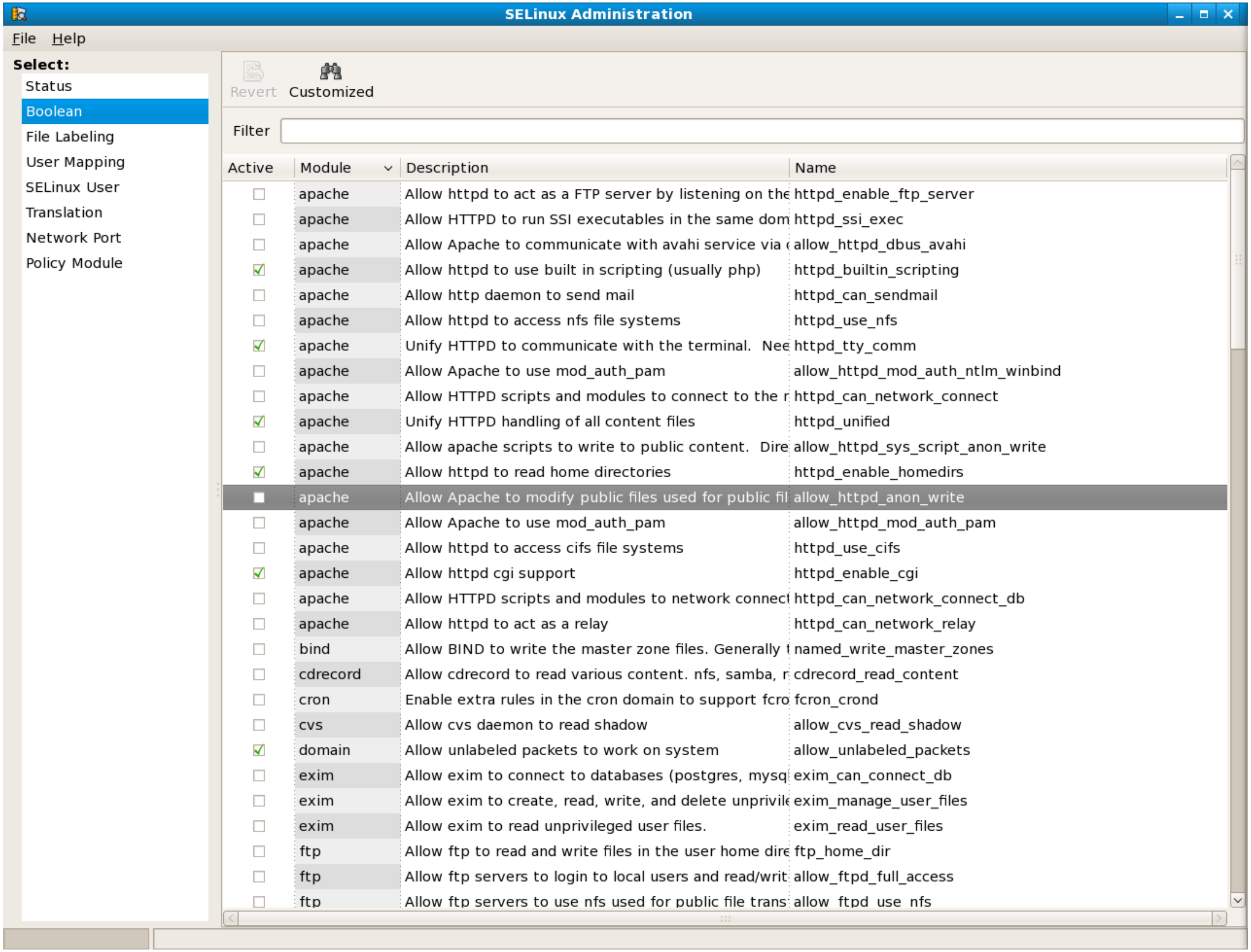
Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC
  - ABAC
- Identity federation

MAC example: SELinux

- Developed by U.S. Dept of Defense
- General deployment starting 2003
- Can apply rules to virtually every user/process/hardware pair
- Rules are governed by system administrator only
  - No such thing as “selinux_chmod” for users

MAC example: SELinux



MAC model

```
bool IsActionAllowed(subject, object, action) {
 for each rule in rules:
  if rule allows (subject,object,action) return true
 return false
}
```

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC
  - ABAC
- Identity federation

RBAC: The thing you invent if you spend enough time doing access control

- Scenario:
  - Frank: “Bob just got hired, please given him access.”
  - Admin: “What permissions does he need?”
  - Frank : “Same as me.”
- Later, a new system is added
  - Bob: “Why can’t I access the new system?!”
  - Admin: “Oh, I didn’t know you needed it too…”
  - Bob: “I need everything Frank has!”
- Later, Frank is promoted to CTO
  - Admin: “Welp, looks like Bob also needs access to our private earnings, since this post-it says he gets everything Frank has…”
- The admin is later fired amidst allegations of conspiracy to commit insider trading with Bob. He dies in prison. ☹

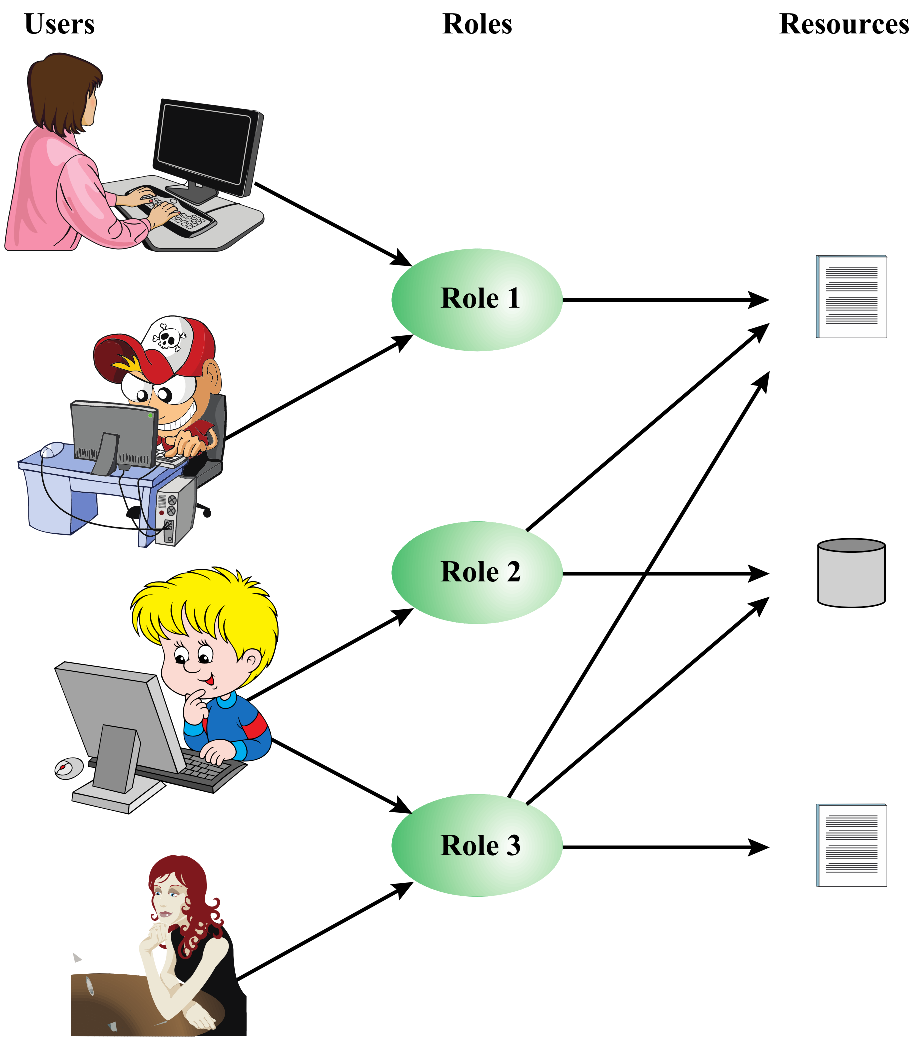


Figure 4.6 Users, Roles, and Resources

RBAC

- Decide what KINDS of users you have (**roles**)
- Assign **permission** to **roles**.
- Assign **users** to **roles**.

- When a role changes, everyone gets the change.
- When a user's role changes, that user gets a whole new set of permissions.
- No more special unique snowflakes.

- Roles may be partially ordered, e.g. "Production developer" inherits from "Developer" and adds access to the production servers

RBAC implementation

- Unsurprisingly, you can represent this using various data structures.
  - Anything that can represent two matrices:

	R_1	R_2	• • •	R_n
U_1	✖			
U_2	✖			
U_3		✖		✖
U_4				✖
U_5				✖
U_6				✖
⋮				
U_m	✖			

OBJECTS

ROLES	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
⋮									
R_n			control		write	stop			

Figure 4.7 Access Control Matrix Representation of RBAC

RBAC model

```
bool IsActionAllowed(subject, object, action) {
  if (action ∈ get_permissions(subject.role,object))
    return true
}
```

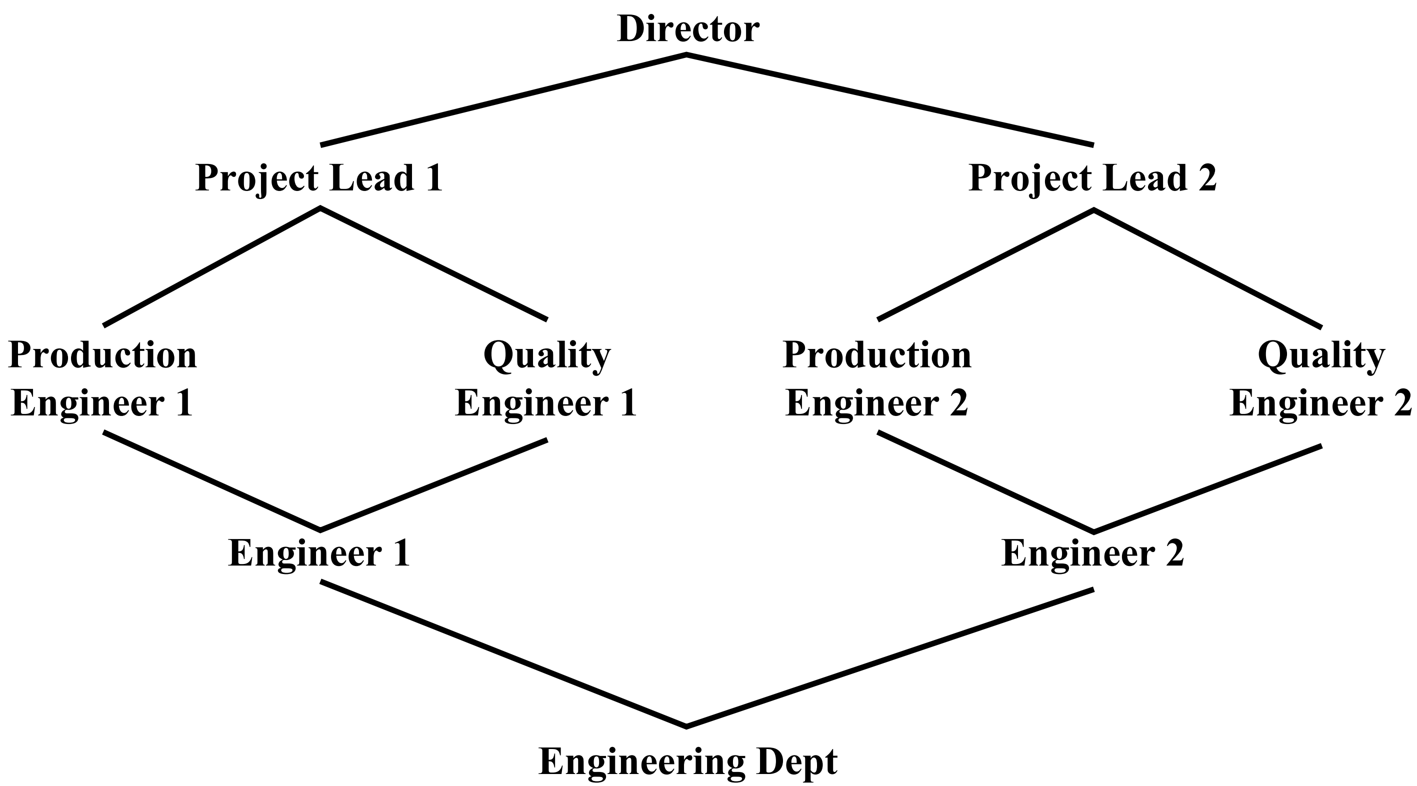


Figure 4.9 Example of Role Hierarchy

Constraints - RBAC

- Provide a means of adapting RBAC to the specifics of administrative and security policies of an organization
- A defined relationship among roles or a condition related to roles
- Types:

Mutually exclusive roles	Cardinality	Prerequisite roles
• A user can only be assigned to one role in the set (either during a session or statically) • Any permission (access right) can be granted to only one role in the set	• Setting a maximum number with respect to roles	• Dictates that a user can only be assigned to a particular role if it is already assigned to some other specified role

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC
  - ABAC
- Identity federation

Attribute-Based Access Control (ABAC)

- Authorizations based on conditions on properties of ***both*** the resource and the subject
- Strength is its flexibility and expressive power
- Main obstacle: complexity to administer (and understand)

I have no evidence that any non-academic person has ever used this, so we're skipping it. Slides included in case you care to learn more.

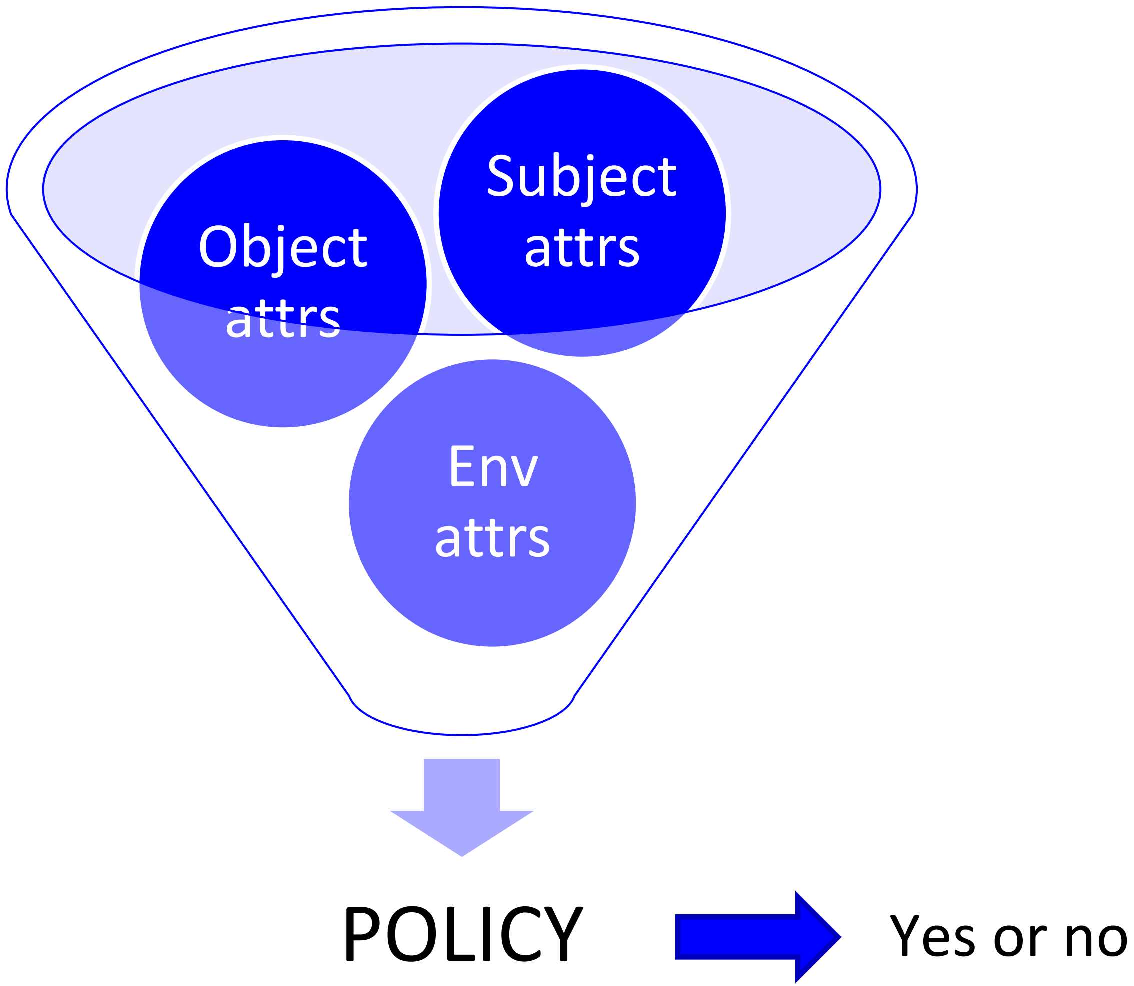
Skip

ABAC Model: Attributes

Subject attributes	Object attributes	Environment attributes
• A subject is an active entity that causes information to flow among objects or changes the system state • Attributes define the identity and characteristics of the subject	• An object (or resource) is a passive information system-related entity containing or receiving information • Objects have attributes that can be leverages to make access control decisions	• Describe the operational, technical, and even situational environment or context in which the information access occurs • These attributes have so far been largely ignored in most access control policies

ABAC in a nutshell

Skip



ABAC

Skip

Distinguishable because it controls access to objects by evaluating rules against the attributes of entities, operations, and the environment relevant to a request

Relies upon the evaluation of attributes of the subject, attributes of the object, and a formal relationship or access control rule defining the allowable operations for subject-object attribute combinations in a given environment

Systems are capable of enforcing DAC, RBAC, and MAC concepts

Allows an unlimited number of attributes to be combined to satisfy any access control rule

Skip

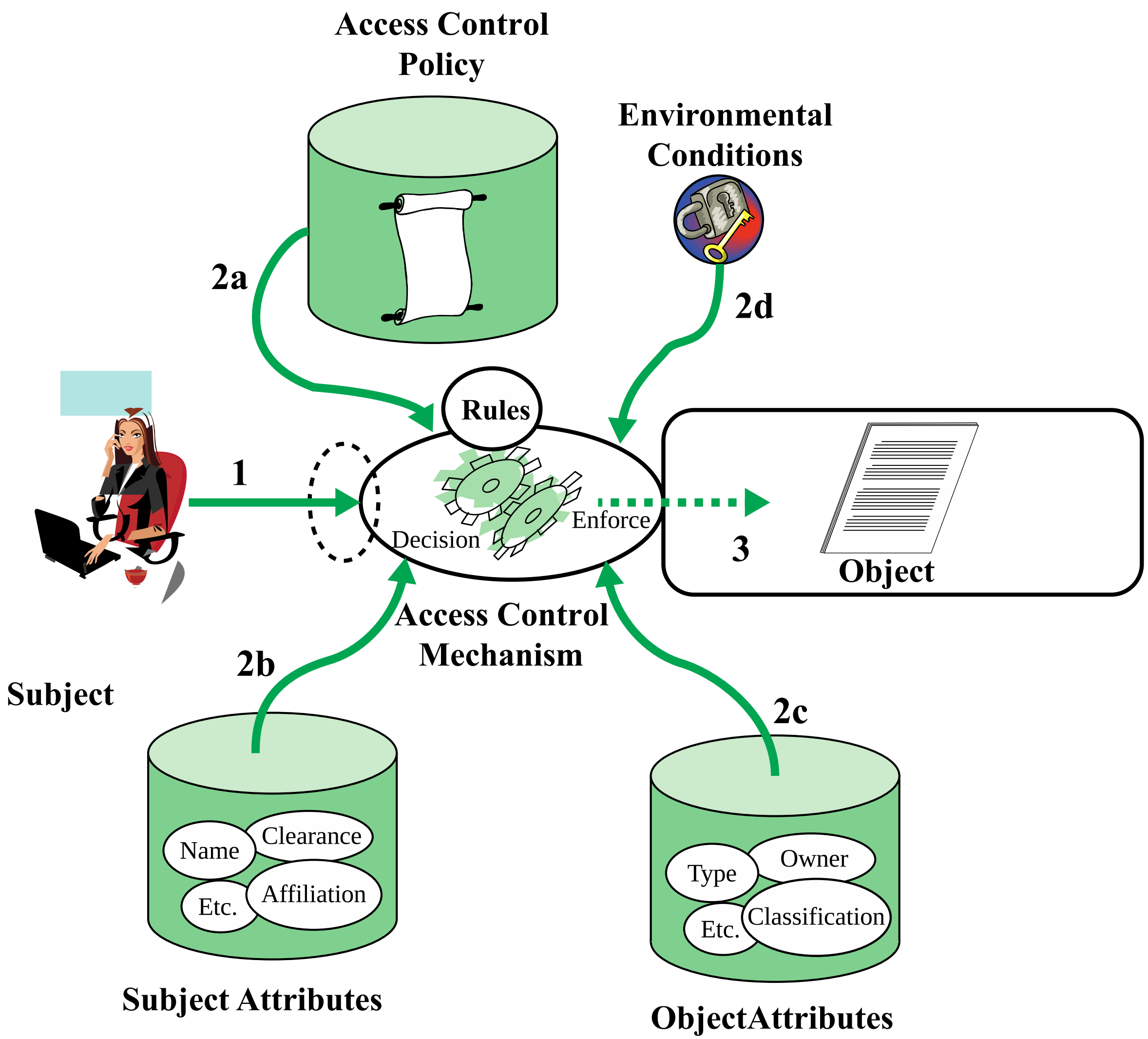


Figure 4.10 Simple ABAC Scenario

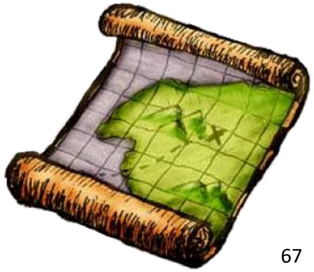
ABAC model

Skip

```
bool IsActionAllowed(subject, object, action) {
  for each rule in rules {
    The rule is basically code that examines all attributes of subject and
object as well as the global environment; the rule is highly expressive,
and so could basically do anything.  If it says yes, return true
  }
  return false
}
```

Topics

- Core concepts
- Access control policies:
  - DAC
    - UNIX file system
  - MAC
  - RBAC
  - ABAC
- Identity federation

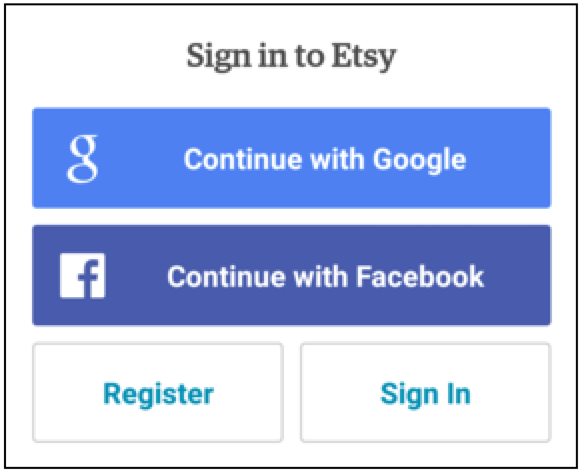


Identity Federation

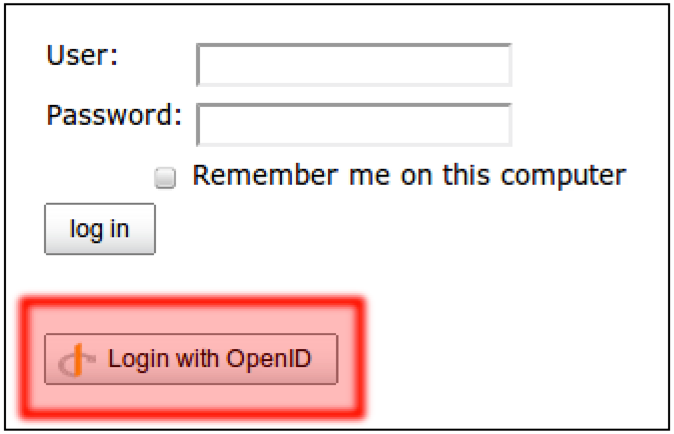
- Term used to describe the technology, standards, policies, and processes that allow an organization to trust digital identities, identity attributes, and credentials created and issued by another organization
- Addresses two questions:
  - How do you trust identities of individuals from external organizations who need access to your systems
  - How do you vouch for identities of individuals in your organization when they need to collaborate with external organizations

Identity Federation made simple

- Translation:



Corporate providers: Google/Facebook



Open provider framework: OpenID

- Allow one entity to manage the concept of “logging in” (credentials, etc.), and communicate that to another entity on behalf of the user

- Want a standard to support federation from any provider? **OAuth**
- Duke has an authentication system: **Duke NetID**
  - You can write apps that use OAuth to allow login via Duke NetID

Any questions?