



Technische
Universität
Braunschweig

IAS

INSTITUTE FOR
APPLICATION
SECURITY



Authentication & Access Control

Vorlesung “Einführung in die IT-Sicherheit”

Prof. Dr. Martin Johns

Overview

- **Topic of the unit**
 - Authentication and Access Control
- **Parts of the unit**
 - Part #1: Basics of authentication
 - Part #2: Passwords and their problems
 - Part #3: Challenge-response authentication
 - Part #4: Access control techniques



Authentication

- **Authentication** = binding of an identity to a subject



- **Confirmation of identity by ...**

- Knowledge factors = what the subject knows
- Ownership factors = what the subject has
- Human factors = what the subject is



Authentication by Knowledge

- **Subject confirms identity by knowledge**
 - e.g. shared secrets, individual knowledge, ...

+ Advantages

- Simple deployment without special hardware
- Replacement of secrets easy in case of compromise

- Disadvantages

- Memory capabilities of human brain limited
- Confidentiality not guaranteed, e.g. sticky notes



Authentication by Ownership

- **Subject confirms identity by ownership**
 - e.g. chip cards, security tokens, smartphones, ...

+ Advantages



- Handling similar to physical keys, e.g. Yubikey
- Cloning and replication usually difficult

- Disadvantages

- Loss and theft of device problematic
- Often need for special hardware



Authentication by Biometry

- **Subject confirms identity through human factors**
 - e.g., fingerprint, iris patterns, typing behaviour ...

+ Advantages

- Ideal binding between subject and identity
- Biometric features naturally available to subject



- Disadvantages

- Replacement of biometric features difficult ( Privacy)
- Evasion resistance still open research question



Multi-Factor Authentication

- **Authentication using multiple factors**
 - Example: Scene from the movie “Mission Impossible”



Ethan Hunt needs to

1. ... use a stolen chip card
(ownership factor)
2. ... fake an IRIS scan
(human factor)
3. ... enter a password
(knowledge factor)



Multi-Factor Authentication

- **Authentication with two factors**
 - No science fiction — we all use it regularly
- **Payment with credit card**
 - Scan of credit card (Ownership)
 - Hand-written signature (Human factor)
- **Payment with debit card**
 - Scan of debit card (Ownership)
 - Input of card PIN (Knowledge)



Overview

- **Topic of the unit**
 - Authentication and Access Control
- **Parts of the unit**
 - Part #1: Basics of authentication
 - Part #2: Passwords and their problems
 - Part #3: Challenge-response authentication
 - Part #4: Access control techniques



Passwords

- **Password = information confirming the identity of user**
 - Knowledge of a secret word, phrase or number



- **Often combination with (a)symmetric cryptography**
 - e.g. password is mapped to key of symmetric cipher
 - e.g. password protects private key of public-key algorithm
 - ~~Passwords are just great. Wait, it's not that easy...~~



Problems with Passwords

- **Password snooping**
 - Eavesdropping of passwords in network traffic
 - Retrieval of passwords from hosts (e.g. via malware)
- **Password guessing (online) or cracking (offline)**
 - Dictionary attacks = guessing using dictionary of words
 - Brute-force attacks = guessing using all possible strings
- **Human deficiencies**
 - Weak and often re-used passwords

Let's fix some of these problems



Password Storage

- **Passwords should never be stored in clear**
 - Application of cryptographic one-way functions
 - Only encoded (hashed) passwords are stored
- **Example:** `$hashed_pw = hash($password);`
 - Simple to validate: `hash($input) == $hashed_pw?`
 - Hard to deduce password from strong hash function
- **Efficient cracking of stored passwords still possible**
 - Brute-force or dictionary attack using hashed strings



Salted Passwords

- **Encoding of password with random string (salt)**
 - Example: `$hashed_pw = hash($password+$salt);`
 - Salt stored in clear with password `[$salt, $hashed_pw]`
 - **Cracking of multiple passwords more expensive**
 - Same password maps to different hash values
 - Pre-computation of hashes not possible anymore
 - No difference when cracking a single password
 - Security depends on quality of password, hash and salt



Slowing-down Crackers

- **Key stretching**
 - Costly encoding of passwords (e.g. through iterations)
 - Useful for password storage and derivation of keys
 - Direct effect on speed of cracking attempts
- **Some examples**
 - CRYPT: 25 iterations with DES variant and salt
 - PBKDF2: >1000 iterations with PRF (e.g. hash) and salt

```
def simple(password, salt, iter, key=''):    for i in iter:        key = hash(key + password + salt)
```



Good Passwords?

- **Testing for insecure passwords is very easy**
 - A laptop can test millions of MD5 hashes per hour
- **Passwords should be very hard to guess**
 - No dictionary words, names, dates and patterns
 - Simple transformations (e.g. reversing) not sufficient
 - Minimum length and diversity of passwords
- **Study by Mazurek et al. (CCS 2013)**
 - ~50% of 25k passwords cracked within one week



Choosing a Good Password

- **Not that easy**
 - Simple translations well-known (e.g., leet speak)
 - Human memory vs. cracking capabilities
 - Partial solution using password managers
- Hint: **simply pick a phrase instead of a word**
 - “I see dead people.” = **iSeeDeadPpl.**
- However: **Avoidance of too common phrases**
 - **2bon2b** found in 4 out of 30 million passwords



Overview

- **Topic of the unit**
 - Authentication and Access Control
- **Parts of the unit**
 - Part #1: Basics of authentication
 - Part #2: Passwords and their problems
 - Part #3: Challenge-response authentication
 - Part #4: Access control techniques



One-Time Passwords

- **Security of passwords “weakens” over time**
 - Password aging = enforced changing of passwords
 - One-time passwords = passwords used exactly once
- **Example: S/KEY Algorithm from the 1980s**
 - User chooses initial key K_1
 - Recursive hashing: $H(K_1) = K_2$, $H(K_2) = K_3$, ... $H(K_{n-1}) = K_n$
 - One-time passwords: $P_1 = K_n$, $P_2 = K_{n-1}$, ... $P_n = K_1$
 - Hard to deduce next password P_i from previous P_{i-1}



Example: RSA SecurID

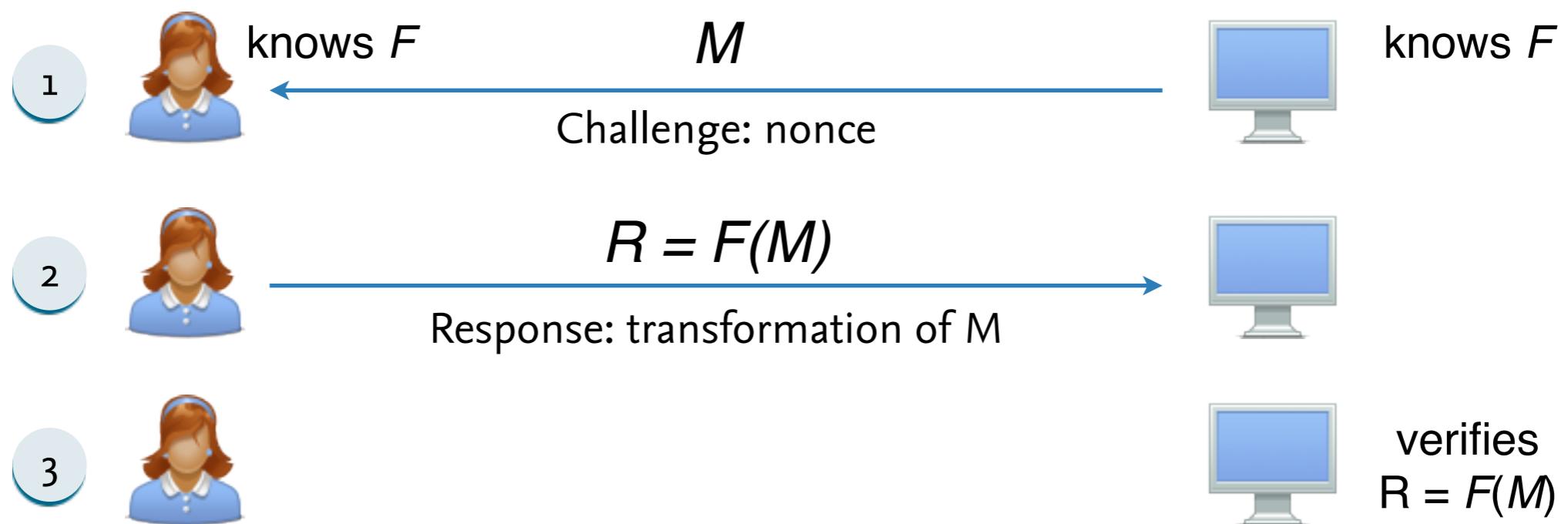
- **Security system using two-factor authentication**
 - Factors: knowledge (password) and ownership (device)
 - Device generates authentication code every 60 seconds
 - Authentication using password and displayed code
- **Code Generation**
 - Device initialized for each user with seed (random number)
 - Code computed from seed and current time (~one-time password)



Challenge-Response Authentication

- **Generic protocol scheme for authentication**

- System and user share a secret function F
- Knowledge of function F “challenged” using nonce M
- Implicit confirmation using response $R = F(M)$



Challenge-Response Authentication

- **Secret function often parameterized by password**
 - $F = H(M + P)$ hash function H and password P
 - $F = E_P(M)$ encryption function E and password P
 - Hard to deduce P if F is cryptographically strong
- **Several methods related to challenge-response scheme**
 - One-time passwords
= challenge (index of password); response (password)
 - RSA SecurID
= challenge (current time); response (authentication code)



Overview

- **Topic of the unit**
 - Authentication and Access Control
- **Parts of the unit**
 - Part #1: Basics of authentication
 - Part #2: Passwords and their problems
 - Part #3: Challenge-response authentication
 - **Part #4: Access control techniques**



Access Control



- **Authorization and access control**
 - Control of what a subject is allowed to do
 - Management of permissions and capabilities
 - Often tight coupling with authentication
- **Examples**
 - Execution of programs, reading of files, ...

Access Control Matrix

- Classic and simple representation for access control
 - Mapping from subjects and objects to permissions

		Objects			
		File 1	File 2	Process 1	Process 2
Subjects	User 1	read		control, send	receive
	User 2	write	read	send	
	User 3	read, execute	read	control	control

Access Control Matrix

- Classic and simple representation for access control
 - Mapping from subjects and objects to permissions

		Objects			
		File 1	File 2	Process 1	Process 2
Subjects	User 1	read		control, send	receive
	User 2	write	read	send	
	User 3	read, execute	read	control	control

Access control lists



Access Control Matrix

- Classic and simple representation for access control
 - Mapping from subjects and objects to permissions

		Objects			
		File 1	File 2	Process 1	Process 2
Subjects	User 1	read		control, send	receive
	User 2	write	read	send	
	User 3	read, execute	read	control	control

Access control lists **Capabilities**



Access Control Models

- **Access control non-trivial in practice**
 - Complex systems ↪ complex access control models
- **Some characteristics of access control models**
 - Definition of objects and subjects
 - e.g. subjects can be users, processes or hosts
 - Representation of permissions
 - e.g. columns (access control lists), rows (capabilities)
 - Management of permissions
 - e.g. discretionary, mandatory or role-based access control



Representation: Access Control Lists

- **Access control lists (ACL)**
 - Attachment of permissions to objects (columns)
 - + Efficient and decentral organization of permissions
 - Listing of subject permissions very involved
- Example: **OpenBSD packet filter**
 - Deny access to the SSH service from any host
 - block in quick proto tcp from any to any port ssh



Representation: Capabilities

- **Capabilities**
 - Attachment of permissions to subjects (rows)
 - + Listing and control of subject permissions simple
 - Fine-grained permissions difficult to implement
- **Example: Linux capabilities**
 - Restrict permissions to reboot system and load modules
 - `lcap -z CAP_SYS_BOOT CAP_SYS_MODULE`



Management of Permissions

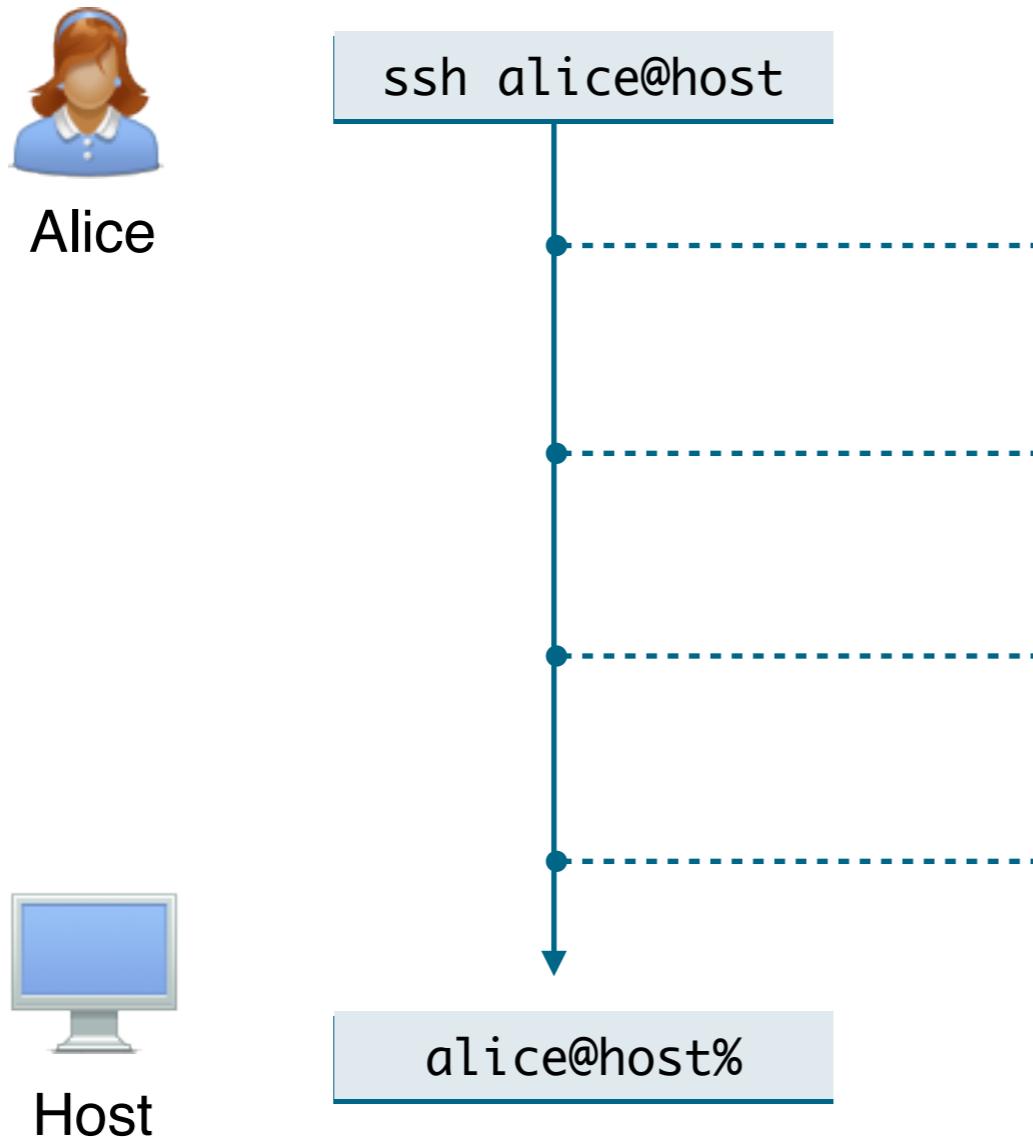
- **Discretionary Access Control (DAC)**
 - Owner of an object controls access
 - Convenient but insecure if object changes owner
- **Mandatory Access Control (MAC)**
 - System globally enforces access control
 - Very secure but tedious to design and operate
- **Role-based Access Control (RBAC)**
 - System enforces access control using roles
 - In-between DAC and MAC models



Summary



... putting it together



Exchange of session key
using public-key cryptography

Efficient encryption using
symmetric-key cryptography

Authentication using
password (or private key)

Access control using
UNIX permissions



Summary

- **Authentication**
 - Key to binding identities to subjects
 - Binding relies on authentication factors
 - Passwords common but weak factor
- **Access Control and Authorization**
 - Definition and management of access control
 - Realizations using ACLs and capabilities

