Database Management Systems

Lecture 6

Security (II)

* *data security* and *data integrity* - similarities

- the system enforces certain constraints (i.e., users cannot violate these constraints)
- constraints are:
  - specified in a declarative language
  - saved in the system catalog
- the system monitors users' actions to enforce the specified constraints

* the DBMS's *authorization subsystem* (*security subsystem*)

- the authorization subsystem checks any given *access request* against the applicable constraints
- *access request*
  - requested object + requested operation + requesting user
- identifying the *applicable constraints*
  - the system must recognize the source of the request, i.e., the requesting user
    - *authentication* mechanism
      - password scheme
        - users supply their user ID (they say who they are) and their password (they prove they are who they say they are)
      - fingerprint readers, voice verifiers, retina scanners, etc

* main approaches to data security in a DBMS

- *discretionary* control
- *mandatory* control

- *discretionary control*
  - users have different access rights on different objects
  - the operations users are allowed to perform are explicitly specified
  - anything not explicitly authorized is implicitly forbidden

* main approaches to data security in a DBMS

- *discretionary* control
- *mandatory* control

- *mandatory* control
  - every object has a classification level (e.g., top secret, secret, etc)
  - every user has a clearance level (same options as for classification levels)
  - levels – strict ordering

  - Bell and La Padula rules:
    - user *x* can retrieve object *y* only if the clearance level of *x* is greater than or equal to the classification level of *y*
    - user *x* can update object *y* only if the clearance level of *x* is equal to the classification level of *y*

* database user management

- user:
  - username
  - password
  - additional information: time intervals during which access is allowed; privileges; answers to some questions

- SQL statements:
  - CREATE / ALTER / DROP user

* database user management

- giving privileges to users
  - every component in the database (table, view, procedure, trigger, etc) can have certain privileges that can be given to users
  - a privilege allows the execution of a SQL statement
  - e.g.,
    - `SELECT [(column_list)]`
    - `UPDATE [(column_list)]`
    - `INSERT`
    - `DELETE`
    - …
  - the names of privileges vary among DBMSs
  - SQL statements:

```
GRANT privilege_list ON component TO user_list [options]
REVOKE privilege_list ON component FROM user_list [options]
```

* recommendations

- using views
- the database administrator is monitoring the database and keeping track of all operations performed by users via *audit trails* that contain:
  - executed SQL statements
  - user names, IP addresses, dates and times
  - affected objects
  - old values and new values for changed records

* SQL injection

- application -> execution of a SQL statement: fixed statement, statement that is generated using input data from the user
- a statement can be changed (when being generated) due to data supplied by the user
- such a change is attempted when the user is trying to obtain additional access rights
- the user enters code into input variables; the code is concatenated with SQL statements and executed
- the DBMS executes all syntactically valid statements

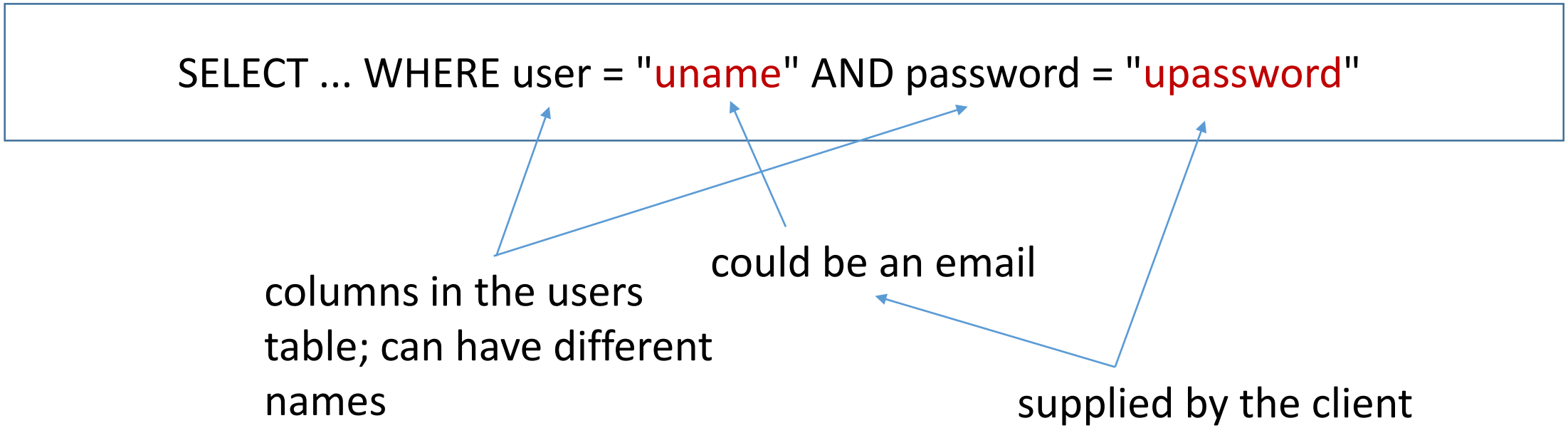
* SQL injection

- obs.
  - string separators:
    - single quotes, double quotes
  - statement separator (if a command can include multiple statements):
    - semicolon
  - comments in SQL:
    - `statement --comment`

* SQL injection

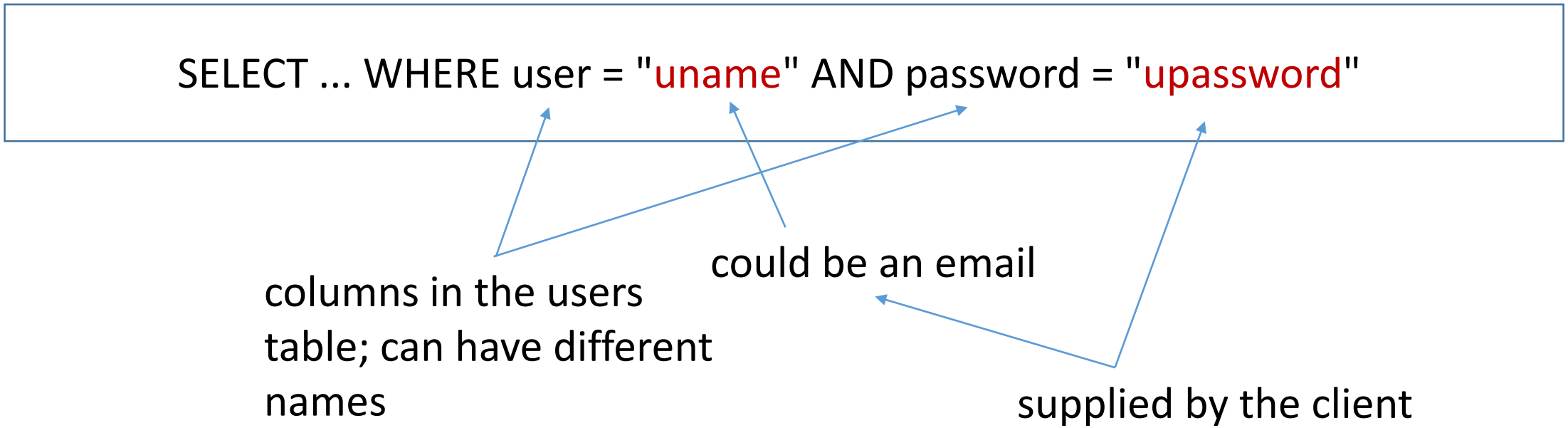
1. changing an authentication statement

- errors when executing the statement -> use error messages in subsequent executions
- authenticating with a username and a password
  - in many cases, the following statement is executed:



* SQL injection

1. changing an authentication statement



- if the statement is successfully executed and returns at least one record, the authentication is successful

* SQL injection

1. changing an authentication statement

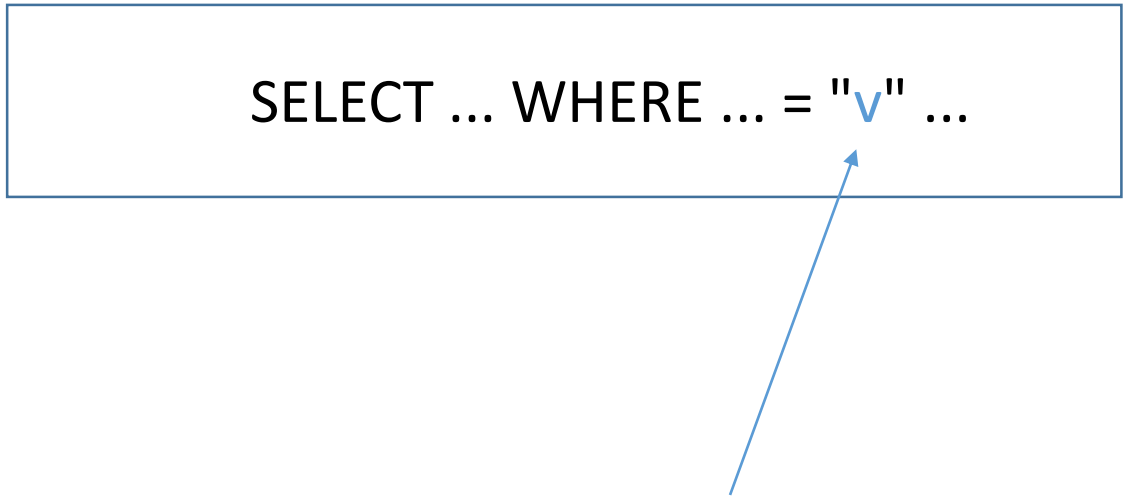
- values that change the SQL statement (modify the action intended by the programmer)

uname	upassword	condition in the statement
a	" OR 1 = 1 --	user="a" AND password="" OR 1=1 -- "
admin" --	?	user="admin" -- AND ... (other values besides *admin* can be chosen from various lists on the internet)
" OR 0=0 --	?	user="" OR 0=0 -- ...

* SQL injection

2. obtaining information from the database

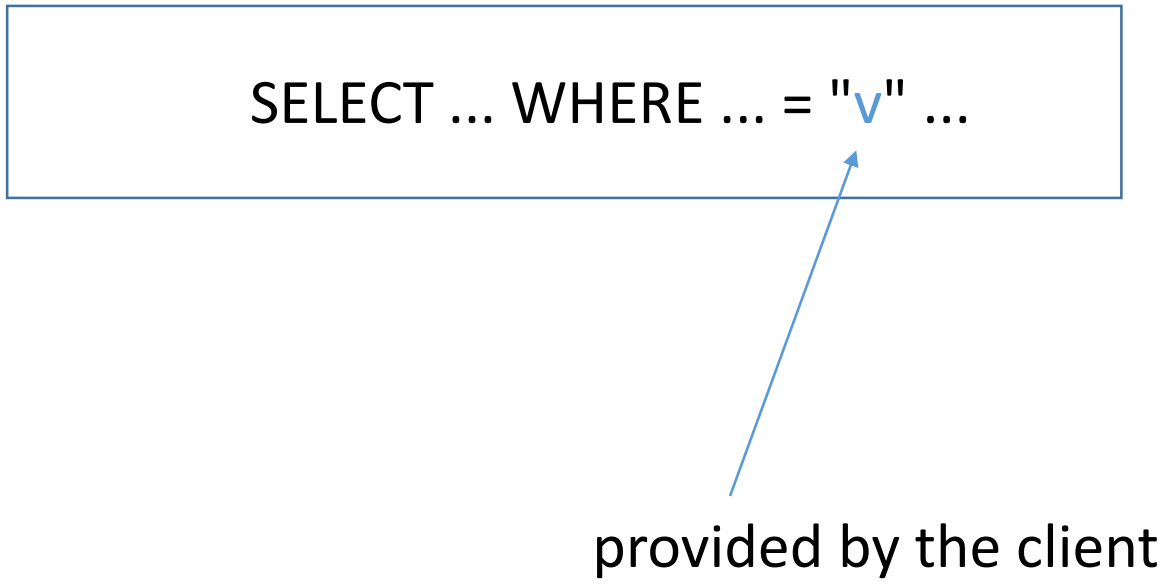
- the client is asked to provide value *v*:



- possible values entered by the user:
  - `x" OR 1 = (SELECT COUNT(*) FROM table) --`
    - error => change table name
    - one can obtain the name of a table (e.g., a table with passwords)

* SQL injection

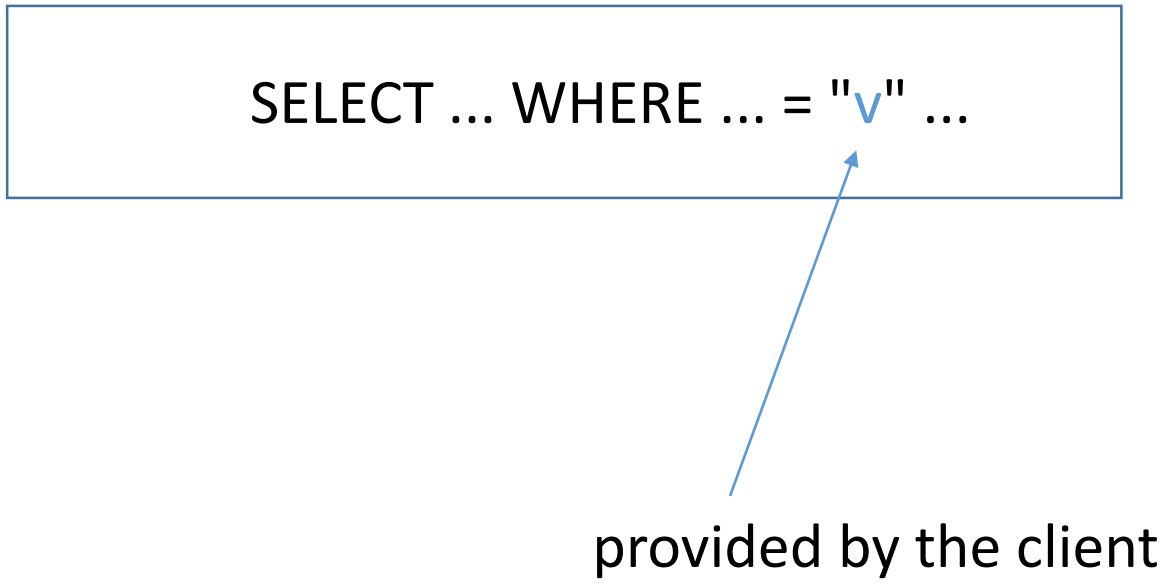
2. obtaining information from the database



- possible values entered by the user:
  - `x" AND user IS NULL --`
    - error => change the name of the column
    - one can obtain the name of a column

* SQL injection

2. obtaining information from the database

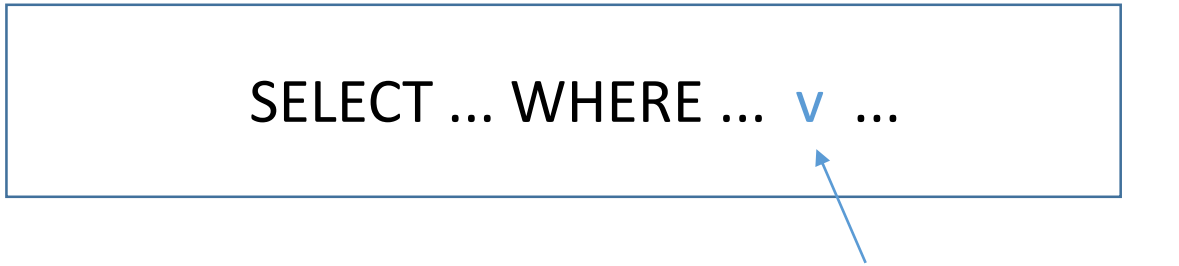


- possible values entered by the user:
  - `x" AND user LIKE "%pop%" --`
    - error / no data => change the name of the column or the string in the condition
    - one can obtain the name of a user

* SQL injection

3. changing data in tables

- the client is asked to provide value *v*:

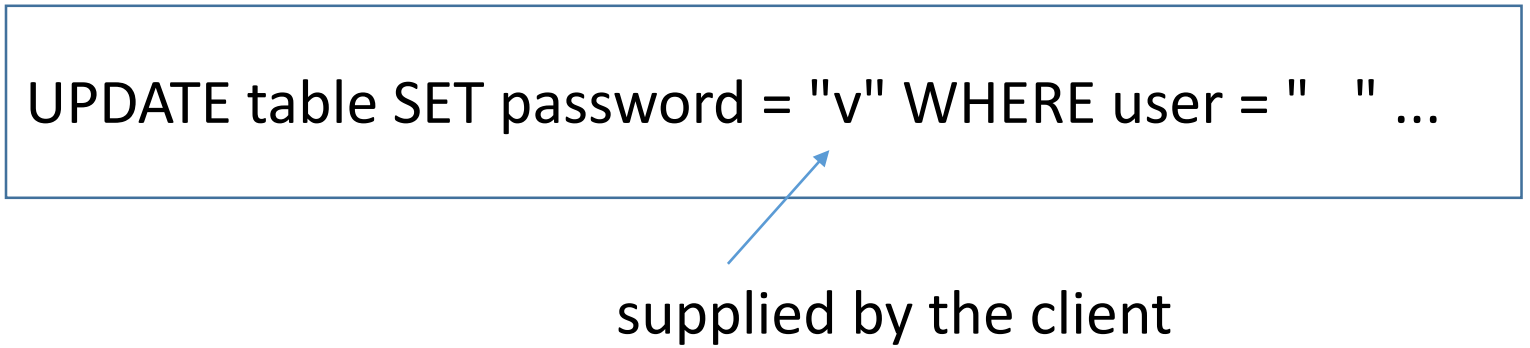


- possible values entered by the user:
  - `0; INSERT INTO users ...`
  - `0; DROP TABLE users`

* SQL injection

4. changing a user's password

- the client is asked to provide value *v*:



- possible values entered by the user for *v* (the new password):
  - `x" --`
  - `x" WHERE USER LIKE "%admin%" --`

* SQL injection

- prevention
  - data validation
    - regular expressions, allow only certain types of characters in the input data
  - modify problematic characters
    - double the: single quotes, double quotes
    - precede single and double quotes with "\"
    - use functions for these changes

    => the single and double quotes in the statement won't modify the statement (like in the previous examples)

* SQL injection

- prevention
  - use parameterized statements

    ```
    SELECT ... WHERE user=? AND password=?
    ```

    , where “?” is a parameter
  - such a statement has a collection of parameters

* data encoding

- protecting the data when the normal security mechanisms in the DBMS are insufficient (e.g., an intruder gets physical access to the server or the data center and steals the drives containing the data, an enemy taps into a communication line, etc)
- storing / transmitting encoded data => the stolen data is illegible for the intruder
- some DBMSs store data (all data or only a part of the data) in an encoded form; if the files containing the database can be copied, using the data in these files is impossible (or extremely difficult, as the decoding cost is enormous)

* data encoding

- *codes* and *ciphers*
  - code
    - replace one word or phrase with another word / number / symbol
  - cipher
    - replace letters with a letter / number / symbol

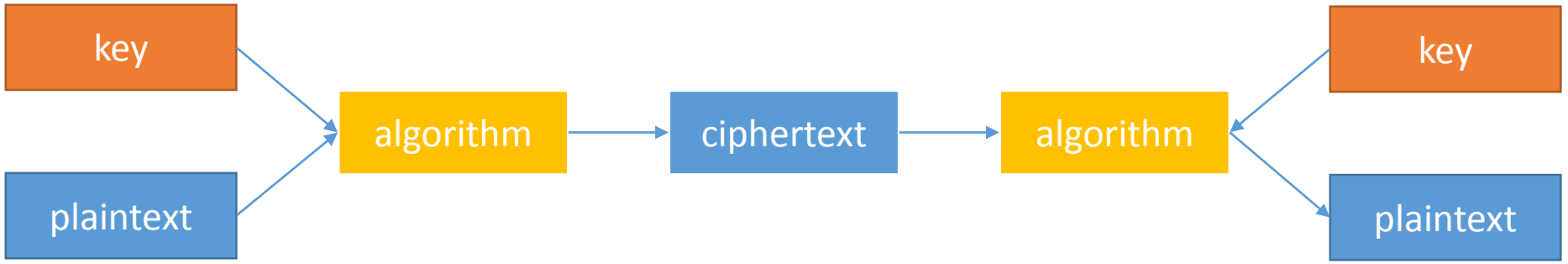
- *steganography* and *cryptography*
  - steganography
    - hide the existence of the message
  - cryptography
    - hide the meaning of the message – *encryption*
    - *transposition* and *substitution*

* data encoding
- cryptography
  - *transposition* and *substitution*
    - transposition
      - rearrange letters in the message (create anagrams)
      - every letter retains its identity, but changes its position
    - substitution
      - pair the letters of the alphabet (randomly)
      - replace each letter in the message with its pair
      - every letter retains its position, but changes its identity

* data encoding

- substitution
  - *plaintext*
    - message before encryption
  - *ciphertext*
    - message after encryption
  - *plain alphabet*
    - alphabet used to write the message
  - *cipher alphabet*
    - alphabet used to encrypt the message
    - examples:
      - shift the original alphabet by 1 position (replace *A* with *B*, *B* with *C*, etc), 2 positions, etc
        => 25 possible ciphers
      - any rearrangement of the plain alphabet
        => over 400.000.000.000.000.000.000.000.000 possible ciphers

* data encoding
- *algorithms* and *keys*



  - algorithm
    - general encryption method
    - not secret
  - key
    - details of a particular encryption
    - must be kept secret!
    - large number of keys
      - e.g., checking 25 versus 400.000.000.000.000.000.000.000.000 keys

* data encoding

- the *key distribution* problem
  - if two parties want to exchange a message, they must first exchange a secret key
    - meet in person, use couriers
    - logistical difficulties, costs way too high

- Whitfield Diffie, Martin Hellman, Ralph Merkle
  - one-way functions in modular arithmetic

->

one-way function: $Y^x \pmod P$ e.g., $7^x \pmod{11}$ 7, 11 – chosen by Alice & Bob	Alice	Bob
1	Alice chooses a number A = 3 A - secret	Bob chooses a number B = 6 B - secret
2	Alice computes: $\alpha = 7^A \pmod{11} = 2$	Bob computes: $\beta = 7^B \pmod{11} = 4$
3	Alice sends α to Bob	Bob sends β to Alice
exchange	An eavesdropper can intercept α and β. This poses no problems, since these numbers are not the key!	
4	Alice computes: $\beta^A \pmod{11} = 4^3 \pmod{11} = 9$	Bob computes: $\alpha^B \pmod{11} = 2^6 \pmod{11} = 9$
the key	Alice and Bob obtained the same number: 9 - the key.	

* data encoding methods - examples

1. use a secret encryption key

- example

data:	disciplina baze de date
secret key:	student

- algorithm

a. create a table of codes

- every character is associated with a number, for instance:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

- *n* - number of characters in the table

* data encoding methods - examples

1.

b. divide the message into blocks of length *L*, where *L* is the number of characters in the key

d	i	s	c	i	p	l	i	n	a		b	a	z	e		d	e		d	a	t	e

s	t	u	d	e	n	t

- replace every character in the message and every character in the key with their associated codes

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

s	t	u	d	e	n	t
19	20	21	04	05	14	20

d	i	s	c	i	p	l	i	n	a		b	a	z	e		d	e		d	a	t	e
04	09	19	03	09	16	12	09	14	01	00	02	01	26	05	00	04	05	00	04	01	20	05

* data encoding methods - examples

1.

b. add every number that corresponds to a character in a block with the number of the corresponding character in the key

- if the obtained value is greater than *n* (in the example, n = 27), compute the remainder of the division by *n*

d	i	s	c	i	p	l	i	n	a		b	a	z	e		d	e		d	a	t	e
04	09	19	03	09	16	12	09	14	01	00	02	01	26	05	00	04	05	00	04	01	20	05
19	20	21	04	05	14	20	19	20	21	04	05	14	20	19	20	21	04	05	14	20	19	20
23	02	13	07	14	03	05	01	07	22	04	07	15	19	24	20	25	09	05	18	21	12	25

c. replace the obtained numbers with their corresponding characters in the table of codes

=> a string representing the result of the encoding algorithm

- this string can be stored / transmitted
- the obtained code in the example: *wbmgnceagvdgosxtyieruly*

* data encoding methods - examples

1.

- decoding
  - similar
  - in step b, perform *subtraction (modulo n)* (instead of addition)

2. permute the original message and add values (to the characters' codes) for every position

3. combine methods **1** and **2**

* data encoding methods - examples

4. DES (Data Encryption Standard)

- US standard - 1977
- secret key
- specify the encryption key K - blocks of 64 bits
  - 56 bits used for encoding => 2^{56} possible combinations (possible keys)
- message divided into 7 character blocks (corresponding to the 56 bits used for encoding)
- characters in the blocks - permuted
- 16 successive encryption rounds with encryption key K_i <- (K, step i)
- decryption algorithm
  - identical to the encryption algorithm
  - keys K_i used in reverse order

* data encoding methods - examples

obs.

- some encryption algorithms use 128 bits for the key
- AES (Advanced Encryption Standard)
  - US - new standard (2000)
  - based on the Rijndael algorithm
  - keys - 128, 192, 256 bits (16, 24, 32 bytes)

- .NET
  - classes that implement encryption algorithms with secret key (*symmetric encryption algorithms*)

* data encoding methods - examples

5. the **RSA** encoding scheme

- Ron Rivest, Adi Shamir, Leonard Adleman

1. randomly choose two large prime numbers *p* and *q*

- compute their product:

$$r = p*q$$

- value *r* can be publicly announced
- *p* and *q* are kept secret
- estimation
  - *p*, *q* – numbers with 63 decimal digits

  => determining *p* and *q* from *r* would require $40*10^{15}$ years on a powerful computer

* data encoding methods - examples

5. the **RSA** encoding scheme

2. choose number *c* such that:

c co-prime with *(p-1)(q-1)*

c > (p-1)(q-1)

- *c* is used as a public encryption key

3. determine a secret decryption key *d* that verifies:

*d*c ≡ 1 modulo (p-1)(q-1)*

* data encoding methods - examples

5. the **RSA** encoding scheme

4. *m* – message (integer)

- encryption
  - determine code *v* using the public key *c* and the value *r*

$$v \equiv m^c \ modulo \ r$$

5. decryption

$$v^d \ modulo \ r$$

=> value *m*

* data encoding methods - examples

5. the **RSA** encoding scheme

- example
- p=3; q=5 ⇒ r = 15
- choose c = 11 > (p-1)(q-1) = 8
- determine **d** such that: d * 11 ≡ 1 modulo 8

  ⇒ d = 3 or d = 11 or d = 19 ... (we'll use **d = 3**)
- let **m = 13** (text to encode)

⇒

v = the code = m^c modulo r = 13^{11} modulo 15 = 1.792.160.394.037 modulo 15 = 7

- decryption: v^d modulo r = 7^3 modulo 15 = 343 modulo 15 = **13**

References

- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Si05] SINGH, Simon, Cartea codurilor – istoria secretă a codurilor și a spargerii lor, Humanitas, 2005
- [Ra07] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, McGraw-Hill, 2007, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Ra00] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (2nd Edition), McGraw-Hill, 2000
- [WWW1] The Open Web Application Security Project, https://www.owasp.org/index.php/SQL_Injection
- [WWW4] SQL injection, http://en.wikipedia.org/wiki/Sql_injection