Database Management Systems

Lecture 5

Security

* database <u>protection</u>
- security & integrity

- <u>security</u>
  - protecting the data against unauthorized users (who may want to read, modify, destroy the data)
  - i.e., users have the right to do what they are trying to do
- <u>integrity</u>
  - protecting the data against authorized users
  - i.e., the operations that users are trying to execute are correct
  - the consistency of the database must be maintained at all times

* aspects related to <u>information security</u>:
- <u>legal, ethical</u> aspects
    - certain rules are being broken
    - e.g., a person searches for a password or accidentally finds such a password and uses it
- <u>physical controls</u>
    - e.g., the computer room is locked (or guarded)
- <u>software controls</u>
    - files are protected (unauthorized access is not allowed)
    - there are no programs intercepting actions of authorized users
- <u>operational problems</u>
    - if a password mechanism is used, how are the passwords kept secret and how often should a user change his / her password?

* data security and data integrity – similarities:

- the system enforces certain <u>constraints</u> (users cannot violate these constraints)
- <u>security constraints</u>, <u>integrity constraints</u>
- constraints are:
  - specified in a declarative language
  - saved in the system catalog

  example: *primary key constraint*, specified in SQL and saved in the system catalog (retrieving info about PK constraints in SQL Server:

  ```
  SELECT *
  FROM sys.objects
  WHERE type = 'PK')
  ```

- the system monitors users' actions to enforce the specified constraints

  example: INSERT, UPDATE statements cannot violate a PK constraint

* the DBMS's <u>authorization subsystem</u> (security subsystem)

- checks any given access request against the applicable constraints
- <u>access request</u>:
  - requested object + requested operation + requesting user

  example: *Alice wants to delete 10 rows from table Customers.*
- identifying the applicable constraints for an access request:
  - the system must recognize the source of the request, i.e., the requesting user
  - an <u>authentication</u> mechanism is used for this purpose, e.g.:
    - password scheme
      - users supply their user ID (users say who they are) and their password (users prove they are who they say they are)
    - fingerprint readers, voice verifiers, retinal scanners, etc

* main approaches to data security in a DBMS:
- discretionary control & mandatory control
- <u>discretionary control</u>
  - users have different access rights (<u>privileges</u>) on different DB components
    - 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

* main approaches to data security in a DBMS:
- <u>discretionary control</u>
  - the operations users are allowed to perform are explicitly specified
  - anything not explicitly authorized is implicitly forbidden
    examples:
    Alice and Bob are both allowed to SELECT from table Customers.
    Jane is allowed to SELECT, INSERT, DELETE from / into table Customers.
  - SQL statements:

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

* main approaches to data security in a DBMS:

- mandatory control
  - every component has a classification level (e.g., *top secret*, *secret*, etc)
    - levels – strict ordering (e.g., *top secret > secret*)
  - every user has a clearance level (same options as for classification levels)
  - Bell and La Padula rules:
    - user *x* can retrieve component *y* only if the clearance level of *x* is >= the classification level of *y*
      example: *Alice's clearance level is top secret, Bob's is secret. Component C1's classification level is top secret, C2's is secret. Alice can retrieve both C1 and C2, Bob can only retrieve C2.*
    - user *x* can update component *y* only if the clearance level of *x* is equal to the classification level of *y*
      example: *Alice can only update C1, Bob can only update C2.*
    Q: why do you think Alice is not allowed to update C2?

* 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

* recommendations

- using <u>views</u>
  - external structures that hide the conceptual structure*
  - the user is allowed to see only a certain subset of the data
- database administrator should monitor the database and keep track of all operations performed by users via an <u>audit trail</u>
  - if there is any suspicion of wrongdoing, the audit trail can be analyzed
  - an audit trail can contain:
    - the executed SQL statement, with the corresponding: user name, IP address, date and time, affected object, old values and new values for changed records

* see the *Databases* course in the 1st semester (lecture 1)

* <u>SQL injection</u>
- 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 statements that are syntactically valid
- 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 a user's authentication statement

- errors when executing the statement => use the error messages in subsequent executions

- authenticating with a username and a password - in many cases, the following statement is executed:

> SELECT ... WHERE user = "uname" AND password = "upassword"

columns in the *users* table (can have various other names)

could be an email

supplied by the client

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

* SQL injection

1. changing a user's 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 password ... (other values besides *admin* can be chosen from various lists on the internet)
" OR 0=0 --	?	user="" OR 0=0 -- ...

SELECT ... WHERE user = "uname" AND password = "upassword"

• for the values in the 1st row, the statement becomes:

SELECT ... WHERE user = "a" AND password = "" OR 1 = 1 --"

SELECT ... WHERE user = "a" AND password = "" OR 1 = 1 --" => all rows are retrieved from the DB, successful authentication

* SQL injection

2. obtaining information from the database

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

- possible values entered by the user:

> provided by the client
>
> SELECT … WHERE … = "v" …

  - `x" OR 1 = (SELECT COUNT(*) FROM table) --`
    - if an error occurs, the statement can be executed again, with a different table name; a malevolent user can obtain, in this way, the name of a table in the DB (for instance, a table with passwords)
  - `x" AND user IS NULL --`
    - if an error occurs, the statement can be executed again, with a different column name (instead of *user* in the example); a malevolent user can obtain the name of a column
  - `x" AND user LIKE "%pop%" --`
    - if an error occurs, or no data is returned => change the column name or the string in the condition (*%pop%*); one can obtain the name of a user

* SQL injection

3. changing data in tables

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

SELECT … WHERE … v …

- possible values entered by the user:
  - `0; INSERT INTO users ...`
    - the SELECT statement is executed, then data about a new user is inserted into the *users* table
  - `0 UNION SELECT CONCAT(name, password) FROM users`
    - retrieving all usernames and passwords while executing the SELECT statement
  - `0; DROP TABLE users`
    - the SELECT statement is executed, then table *users* is dropped

* 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" --`

    - setting the password for all users to *x*

  - `x" WHERE user LIKE "%admin%" --`

    - setting the password for all usernames matching the criterion specified by the LIKE operator

UPDATE table
SET password = "v"
WHERE user = " " ...

supplied by the client

* SQL injection
- prevention
  - data validation
    - use regular expressions to validate data
    - allow the client to use only certain types of characters in the input data
  - modify problematic characters
    - double single quotes and double quotes; or precede single and double quotes with "\" (one can use functions for such changes)
    => single and double quotes in the statement won't modify the statement (like in the previous examples)

example: SELECT ... WHERE user = "uname" AND password = "upassword"

user enters: *a* and *" OR 1 = 1 --* for *uname* and *upassword* (like in a prev. ex.)

double the "

=> statement: SELECT ... WHERE user = "a" AND password = " " " OR 1 = 1 -- ", which

retrieves rows with user *a* and password *" OR 1 = 1 --*

* SQL injection
- <u>prevention</u>
  - use <u>parameterized statements</u>
    `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
  - <u>code</u>
    - replace one word or phrase with another word / number / symbol
    - e.g., replace expression *The enemy is moving* with code word *Binary*
  - <u>cipher</u>
    - replace each letter with another letter (number / symbol)
    - e.g., replace each letter in the alphabet with the previous one (B with A, C with B, etc); *The enemy is moving* becomes *Sgd dmdlx hr lnuhmf*

- steganography and cryptography
  - <u>steganography</u>
    - hide the *existence* of the message
    - e.g., shave messenger's head, write message on scalp, wait for the hair to grow back; use invisible ink; etc

* data encoding
- steganography and cryptography
  - <u>cryptography</u>
    - hide the *meaning* of the message – <u>encryption</u>
    - branches in cryptography:
      - transposition & substitution

- <u>transposition</u>
  - rearrange letters in the message (create anagrams)
  - every letter retains its identity, but changes its position
  - e.g., 6 ways of rearranging 3 letters: *cat, cta, act, atc, tca, tac*
- <u>substitution</u>
  - pair the letters of the alphabet (randomly, e.g., A's pair could be X)
  - replace each letter in the message with its pair
  - every letter retains its position, but changes its identity   ->

* data encoding
- <u>substitution</u>
  - *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)
      - by 2 positions (replace *A* with *C*, *B* with *D*, etc), etc
    => there are 25 possible ciphers
    - if any rearrangement of the plain alphabet is allowed (don't restrict to shift-based rearrangements) => over $4*10^{26}$ possible rearrangements (i.e., over $4*10^{26}$ ciphers)
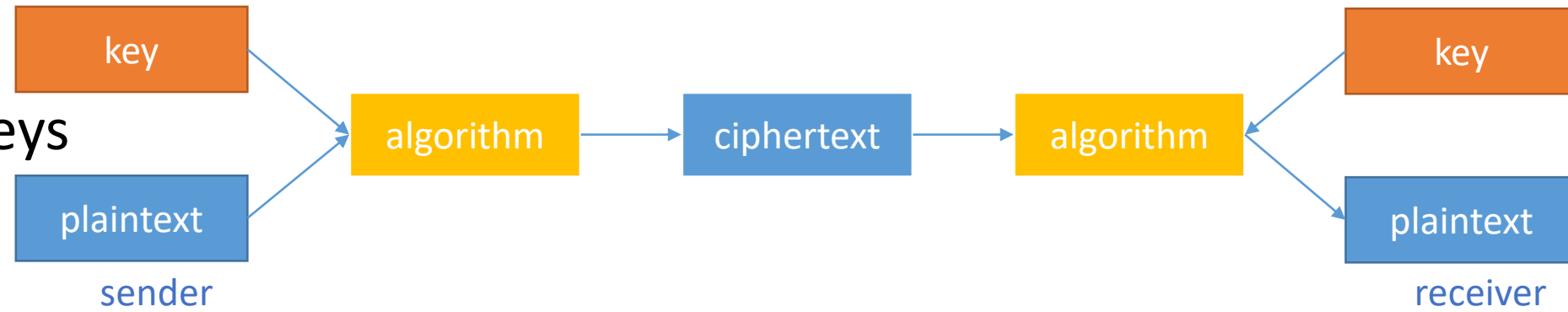
* data encoding

- <u>substitution</u>
  example:
  - *plain alphabet:*

a	b	c	d	...
B	C	D	E	...

  - *cipher alphabet:*

    (shift the original alphabet by one position)
  - *plaintext*:   *the enemy is moving*
  - *ciphertext*:   *UIF FOFNZ JT NPWJOH*

* data encoding
- algorithms and keys

- <u>algorithm</u> - not secret
  - general encryption method (e.g., replace each letter in the plain alphabet with a letter from the cipher alphabet; cipher alphabet can be any rearrangement of the plain alphabet)
- <u>key</u> - must be kept secret!
  - details of a particular encryption (e.g., the chosen cipher alphabet)
  - important - large number of keys:
    - suppose the enemy intercepts a ciphertext; if they know a shift cypher has been used, they only need to check 25 keys; but if any rearrangement of the original alphabet is possible, they need to check $4*10^{26}$ keys!

* data encoding

- the key distribution problem - optional
  - if two parties want to exchange a message, they must first exchange a secret key:
    - meet in person, use couriers (banks used such couriers back in the '70s to communicate securely with their customers)
      - logistical difficulties, costs way too high
  - Whitfield Diffie, Martin Hellman, Ralph Merkle
    - came up with a solution that would enable 2 people to exchange a key without meeting in person and without relying on a 3rd party
    - solution based on one-way functions in modular arithmetic

->

* data encoding

- the key distribution problem - optional
- examples from *SINGH, Simon, Cartea codurilor – istoria secretă a codurilor și a spargerii lor, Humanitas, 2005*
- two-way functions - they are easy to do and to undo
  example: doubling a number is a two-way function
  double 8 => 16; given 16, it's easy to get back to the original number, 8
  example: turning on a light switch can also be thought of as a function
  turn on switch => light bulb turned on; can easily turn off the bulb by turning off the switch
- one-way functions - easy to do and extremely difficult to undo
  example: mixing yellow and blue paint to obtain green paint is a one-way function
  example: modular arithmetic function $453^x \pmod{21997}$

one-way function: Y^x (mod P) e.g., 7^x (mod 11) 7, 11 – numbers chosen by Alice & Bob; not secret	Alice	Bob
1	Alice chooses a number A = 3 A - secret	Bob chooses a number B = 6 B - secret
2	Alice computes: $\alpha = 7^A$ (mod 11) = 2	Bob computes: $\beta = 7^B$ (mod 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: β^A (mod 11) = 4^3 (mod 11) = 9	Bob computes: α^B (mod 11) = 2^6 (mod 11) = 9
the key	Alice and Bob obtained the same number: 9 - the key.	

- Alice and Bob are able to establish a key without meeting beforehand
- Mark (eavesdropper, trying to intercept exchanged messages) only knows the 7^x (mod 11) function, $\alpha = 2, \beta = 4$
- Mark doesn't know the key and cannot obtain it, since:
  - he doesn't know A and B (these numbers are kept secret);
  - it's extremely difficult for him obtain A from α (or B from β), since 7^x (mod 11) is a one-way function; this is especially true if the chosen numbers are very large

* 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 - optional

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 - optional

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:

$$c \text{ co-prime with } (p-1)(q-1)$$

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

• *c* is used as a public encryption key

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

$$d*c \equiv 1 \text{ modulo } (p-1)(q-1)$$

* 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 $\Rightarrow$ r = 15

- choose c = 11 > (p-1)(q-1) = 8

- determine **d** such that: d * 11 ≡ 1 modulo 8

  $\Rightarrow$ d = 3 or d = 11 or d = 19 ... (we'll use **d = 3**)

- let **m = 13** (text to encode)

$\Rightarrow$

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