Database Management Systems

Lecture 5

Security

Sabina S. CS

* database protection
  - security & integrity

- security
  - 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
- integrity
  - 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 security:

- legal, ethical aspects
  - certain rules are being broken
  - e.g., a person searches for a password or accidentally finds a password and uses it
- physical controls
  - e.g., the computer room is locked (or guarded)
- software controls
  - files are protected (unauthorized access is not allowed)
  - there are no programs intercepting the actions of authorized users
- operational problems
  - 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 constraints (users cannot violate these constraints)
- security constraints, integrity constraints
- 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 authorization subsystem (security subsystem)

- checks any given access request against the applicable constraints
- access request:
  - 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 authentication 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
- discretionary control
  - users have different access rights (privileges) on different database objects
  - very flexible schemes, e.g.:
    - user U1 sees object O1, but doesn't see object O2
    - user U2 sees object O2, but doesn't see object O1

* main approaches to data security in a DBMS:

- discretionary control & mandatory control
- mandatory control
  - each object has a classification level
  - each user has a clearance level
  - an object O can be accessed by user U only if U has the right clearance
  - schemes - more rigid, e.g.:
    - user U1 can see object O1, but cannot see object O2
      - i.e., object O2's classification level is higher than O1's classification level
    - so no user U2 can see O2, but not O1

* discretionary control

- 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 Orders.

* discretionary control

- possible language to define *authorities*

AUTHORITY CA1
GRANT RETRIEVE
ON Customers
TO Alice, Bob

AUTHORITY CA2
GRANT RETRIEVE {CID, LastName, FirstName, DOB}, DELETE
ON Customers
TO Jane

- an authority has:
  - a name
  - a set of privileges (GRANT clause)
  - the relation to which it applies (ON clause)
  - a set of users (user IDs) who will be granted the specified privileges on the specified relation (TO clause)

- privileges
  - RETRIEVE [attr_list]
  - INSERT [attr_list]
  - DELETE
  - UPDATE [attr_list]

* discretionary control

- SQL support
  - views
    - users are allowed to see only a certain subset of the data
  - authorization subsystem
    - users can grant privileges to / revoke privileges from other users

  - user U created object O => U automatically receives all privileges associated with O
    - e.g., O is a table => U is automatically granted the following privileges: SELECT, INSERT, UPDATE, DELETE, REFERENCES, TRIGGER
      - TRIGGER - allows the user to create a trigger on O
      - REFERENCES - allows the user to refer to O in an integrity constraint
    - U can also grant these privileges to other users

* discretionary control

- SQL support
  - granting privileges - GRANT statement

    GRANT privilege_list
        ON object
        TO user_list
      [WITH GRANT OPTION]

    - WITH GRANT OPTION - users (in user_list) can grant the specified privileges on the specified object to other users

* discretionary control

- SQL support
  - revoking privileges - REVOKE statement

REVOKE [GRANT OPTION FOR] privilege_list
ON object
FROM user_list
behavior

REVOKE SELECT, DELETE, UPDATE {FirstName, LastName}
ON Customers
FROM Jane RESTRICT

REVOKE SELECT
ON Customers
FROM Alice, Bob CASCADE

* discretionary control

- SQL support
  - revoking privileges - REVOKE statement
    - GRANT OPTION FOR
      - revoke the grant authority
    - behavior - RESTRICT / CASCADE
      - let p be a privilege on object O
      - user U1 grants privilege p to user U2, user U2 grants privilege p to user U3; U1 tries to revoke p from U2 => privilege p of U3 would be *abandoned*
      - RESTRICT / CASCADE - avoid abandoned privileges
        - RESTRICT - operation fails
        - CASCADE - abandoned privileges are also revoked

* discretionary control

- SQL support
  - revoking privileges - REVOKE statement
    - behavior - RESTRICT / CASCADE
      - user U1 grants privilege p to user U2, user U2 grants privilege p to user U3
        - U1 tries to revoke p from U2 with RESTRICT => REVOKE fails
        - U1 tries to revoke p from U2 with CASCADE => p is revoked from U2 and U3

* mandatory control
  - every object 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 object *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.*
      Object O1's classification level is top secret, O2's is secret.
      Alice can retrieve both O1 and O2, Bob can only retrieve O2.
    - user *x* can update object *y* only if the clearance level of *x* is equal to the classification level of *y*
      example: *Alice can only update O1, Bob can only update O2*.

* audit trail

- database administrator should monitor the database and keep track of all operations performed by users via an audit trail
  - if there is any suspicion of wrongdoing, the audit trail can be analyzed
  - an audit trail can contain:
    - executed SQL statement, user name of the user who executed the statement, IP address, date and time, affected object, old values and new values for changed records

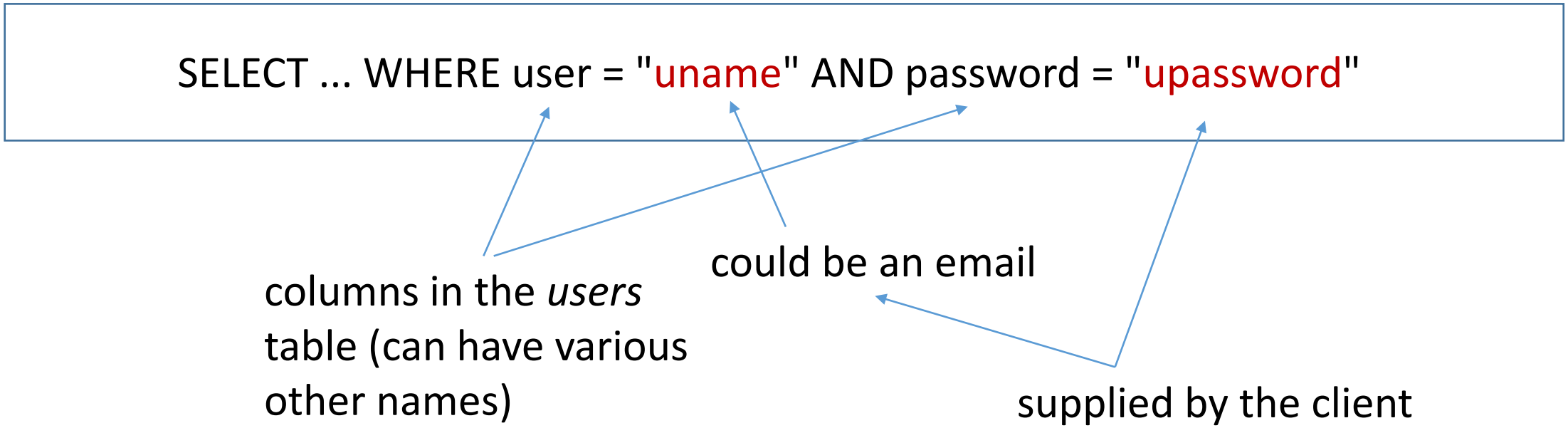
* SQL injection

- 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 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:



- 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"

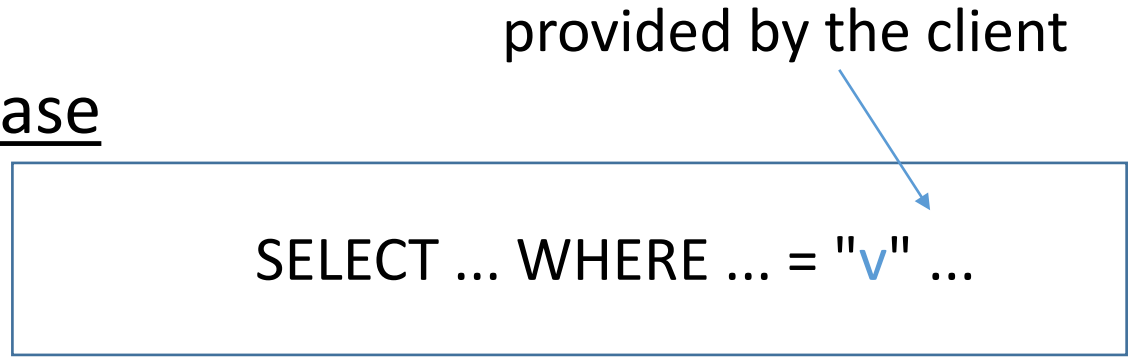
- for the values in the 1st row, the statement becomes:

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

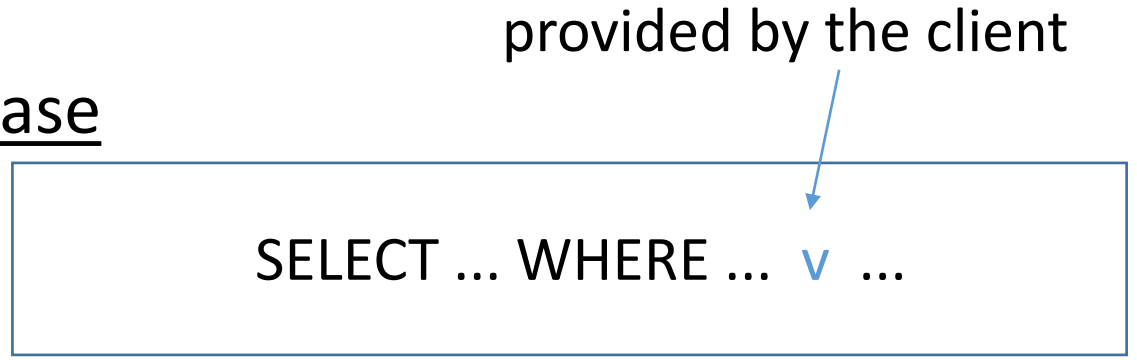
2. obtaining information from the database



- the client is asked to provide value *v*:
- possible values entered by the user:
  - `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

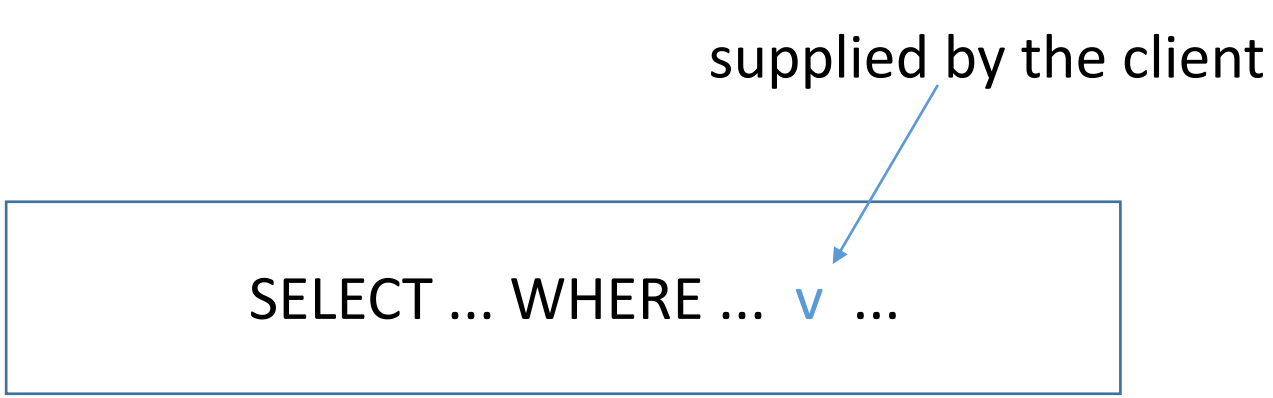
2. obtaining information from the database



- the client is asked to provide value *v*:
- possible value entered by the user:
  - `0 UNION SELECT CONCAT(name, password) FROM users`
    - retrieving all usernames and passwords while executing the SELECT statement

* SQL injection

3. changing data in tables

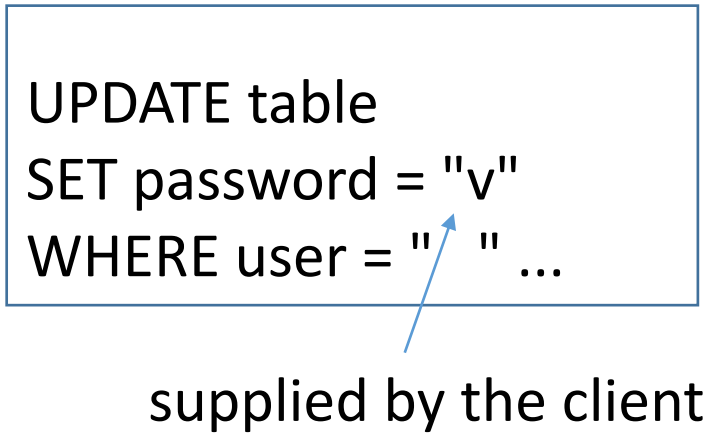


- the client is asked to provide value *v*:
- possible values entered by the user:
  - `0; INSERT INTO users ...`
    - the SELECT statement is executed, then data about a new user is inserted into the *users* table
  - `0; DROP TABLE users`
    - the SELECT statement is executed, then table *users* 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 specified criterion



* 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

- prevention
  - use parameterized statements

    `SELECT ... WHERE user=? AND password=?`, where "?" is a parameter
  - such a statement has a collection of parameters

* data encryption

- 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 encrypted data => the stolen data is illegible for the intruder
- some DBMSs store data (all data or only a part of the data) in an encrypted form; if the files containing the database can be copied, using the data in these files is impossible (or extremely difficult, as the decryption cost is enormous)

* codes and ciphers

- code
  - replace one word or phrase with another word / number / symbol
  - e.g., replace expression *The enemy is moving* with code word *Binary*
- cipher
  - 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

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

* 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.)
    - 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)

* substitution

example:

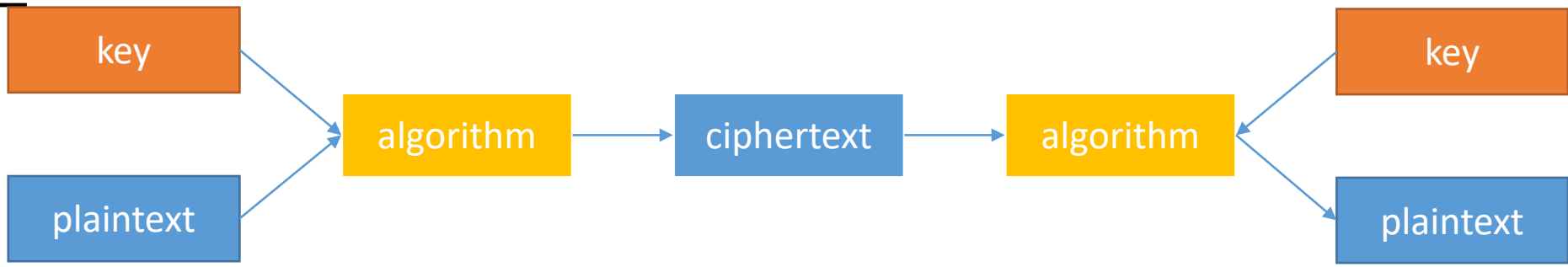
- *plain alphabet:*

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*

* algorithms and keys



- algorithm - 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)
- key - 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!

* 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
  - one-way functions - easy to do and extremely difficult to undo

one-way function: $Y^x \pmod P$ e.g., $7^x \pmod{11}$ 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: $\alpha = 7^A \pmod{11} = 2$	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.	

- Alice and Bob are able to establish a key without meeting beforehand
- Mark (eavesdropper, trying to intercept exchanged messages) only knows the $7^x \pmod{11}$ 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 to obtain A from α (or B from β), since $7^x \pmod{11}$ is a one-way function; this is especially true if the chosen numbers are very large

* encryption algorithm - example

- use a secret encryption key

data:	disciplina baze de date
secret key:	student

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

* encryption algorithm - example

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

* encryption algorithm - example

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 that can be stored / transmitted

- the obtained string in the example: *wbmgnceagvdgosxtyieruly*

* encryption algorithm - example

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

- one could also permute the original message and add values (to the characters' codes) for every position; or combine this method with the previous one

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
- [Ra02] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (3rd Edition), McGraw-Hill, 2002
- [Ra02S] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, Slides for the 3rd Edition, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [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