CS 4604: Introduction to Database Management Systems

Security & SQL injection

Virginia Tech CS 4604 Sprint 2021

Instructor: Yinlin Chen

Today’s Topics

- DB Security
- Application Security

Introduction to DB Security

- Secrecy: Users should not be able to see things they are not supposed to
  - E.g., A student can't see other students' grades.
- Integrity: Users should not be able to modify things they are not supposed to
  - E.g., Only instructors can assign grades.
- Availability: Users should be able to see and modify things they are allowed to.

Access Controls

- A security policy specifies who is authorized to do what
- A security mechanism allows us to enforce a chosen security policy
- Two main mechanisms at the DBMS level:
  - Discretionary access control: allows each user to control access to their own data
  - Mandatory access control: access to system resources is controlled by the operating system

Mandatory Access Control

- Based on system-wide policies that cannot be changed by individual users.
  - Each DB object is assigned a security class.
  - Each subject (user or user program) is assigned a clearance for a security class.
  - Rules based on security classes and clearances govern who can read/write which objects.
- Most commercial systems do not support mandatory access control. Versions of some DBMSs do support it; used for specialized (e.g., military) applications.

Why Mandatory Control?

- Discretionary control has some flaws, e.g., the Trojan horse problem:
  - Dickinson creates Horsie and gives INSERT privileges to Justin (who doesn't know about this).
  - Dickinson modifies the code of an application program used by Justin to additionally write some secret data to table Horsie.
  - Now, Justin can see the secret info.
- The modification of the code is beyond the DBMS's control, but it can try and prevent the use of the database as a channel for secret information.

Discretionary Access Control

- Based on the concept of access rights or privileges for objects (tables and views), and mechanisms for giving users privileges (and revoking privileges)
- Creator of a table or a view automatically gets all privileges on it
  - DMBS keeps track of who subsequently gains and loses privileges and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.

GRANT Command

```
GRANT privileges ON object TO users [WITH GRANT OPTION]
```

- The following privileges can be specified:
  - SELECT: Can read all columns (including those added later via ALTER TABLE command)
  - INSERT(col-name): Can insert tuples with non-null or non-default values in this column
  - DELETE: Can delete tuples.
  - REFERENCES (col-name): Can define foreign keys (in other tables) that refer to this column.
- If a user has a privilege with the GRANT OPTION, can pass privilege on to other users (with or without passing on the GRANT OPTION).
- Only owner can execute CREATE, ALTER, and DROP.

GRANT and REVOKE of Privileges

- GRANT INSERT, SELECT ON Sailors TO Horatio
  - Horatio can query Sailors and insert tuples into it.
- GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION
  - Yuppy can delete tuples, and also authorize others to do so.
- GRANT UPDATE (rating) ON Sailors TO Dustin
  - Dustin can update (only) the rating field of Sailors tuples.
- GRANT SELECT ON ActiveSailors TO Guppy, Yuppy
  - The 'uppies can query ActiveSailors directly
- REVOKE: When a privilege is revoked from X, it is also revoked from all users who got it solely from X.

GRANT/REVOKE on Views

- If the creator of a view loses the SELECT privilege on an underlying table, the view is dropped (on some databases)!

- If the creator of a view loses a privilege held with the grant option on an underlying table, (s)he loses the privilege on the view as well; so do users who were granted that privilege on the view!

Views and Security

- Views can be used to present necessary information (or a summary), while **hiding details** in underlying relation(s).
  - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the bid's of boats that have been reserved.
- Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
- Together with GRANT/REVOKE commands, views are a very powerful access control tool.

Role-Based Authorization

- In SQL-92, privileges are actually assigned to authorization ids, which can denote a single user or a group of users.
- In SQL:1999 (and in many current systems), privileges are assigned to roles.
  - Roles can then be granted to users and to other roles.
  - Reflects how real organizations work.
  - Illustrates how standards often catch up with “de facto” standards embodied in popular systems.

Security to the Level of a Field!

- Can create a view that only returns one field of one tuple. (How?, Why?)
- Then grant access to that view accordingly.
- Allows for arbitrary granularity of control, but:
  - Clumsy to specify, though this can be hidden under a good UI
  - Performance is unacceptable if we need to define field-granularity access frequently. (Too many view creations and look-ups.)

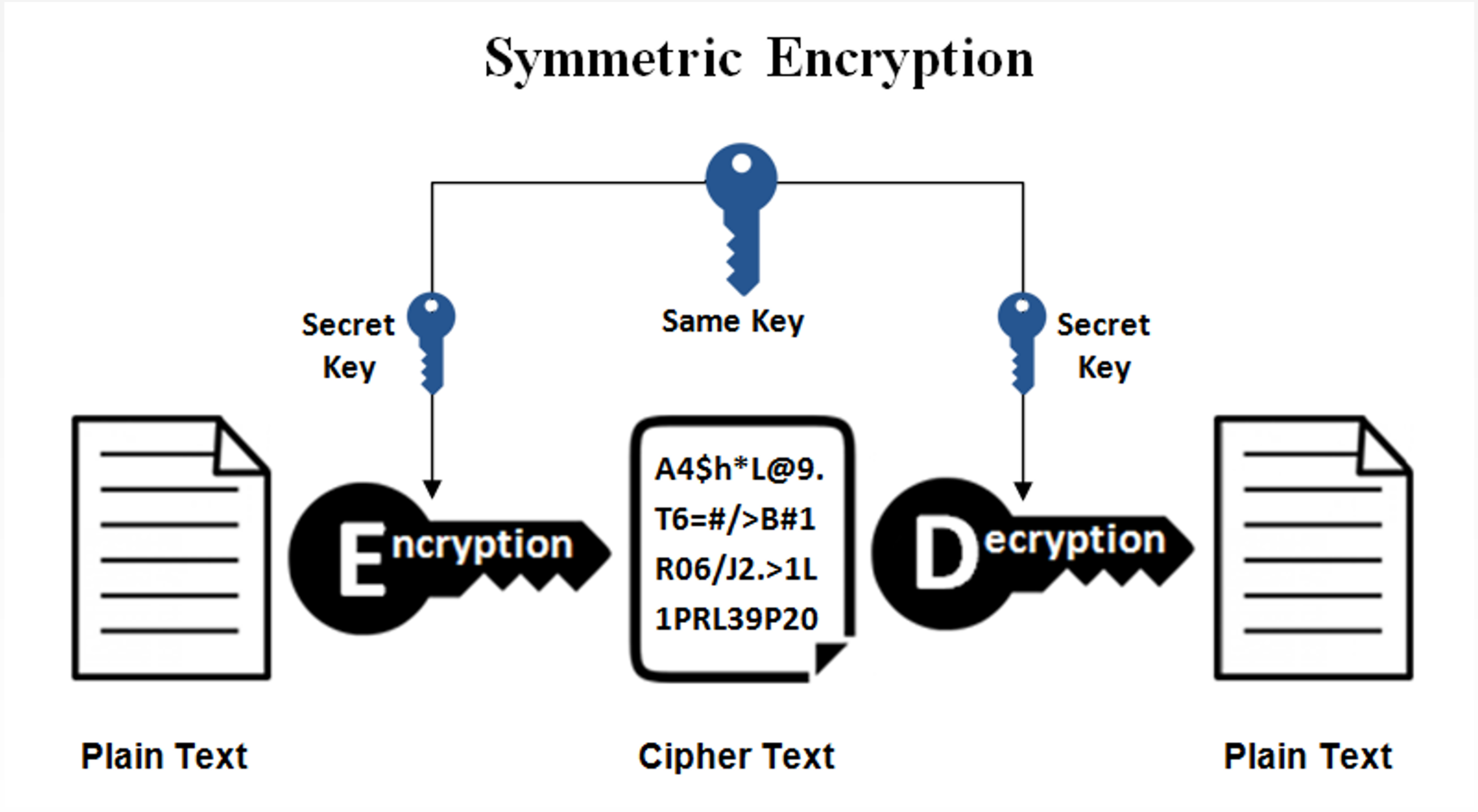
Internet-Oriented Security

- Key Issues: User authentication and trust.
  - When DB must be accessed from a secure location, password-based schemes are usually adequate.
- For access over an external network, trust is hard to achieve.
  - If someone with Sam's credit card wants to buy from you, how can you be sure it is not someone who stole his card?
  - How can Sam be sure that the screen for entering his credit card information is indeed yours, and not some rogue site spoofing you (to steal such information)?
  - How can he be sure that sensitive information is not "sniffed" while it is being sent over the network to you?
- Encryption is a technique used to address these issues.

Encryption

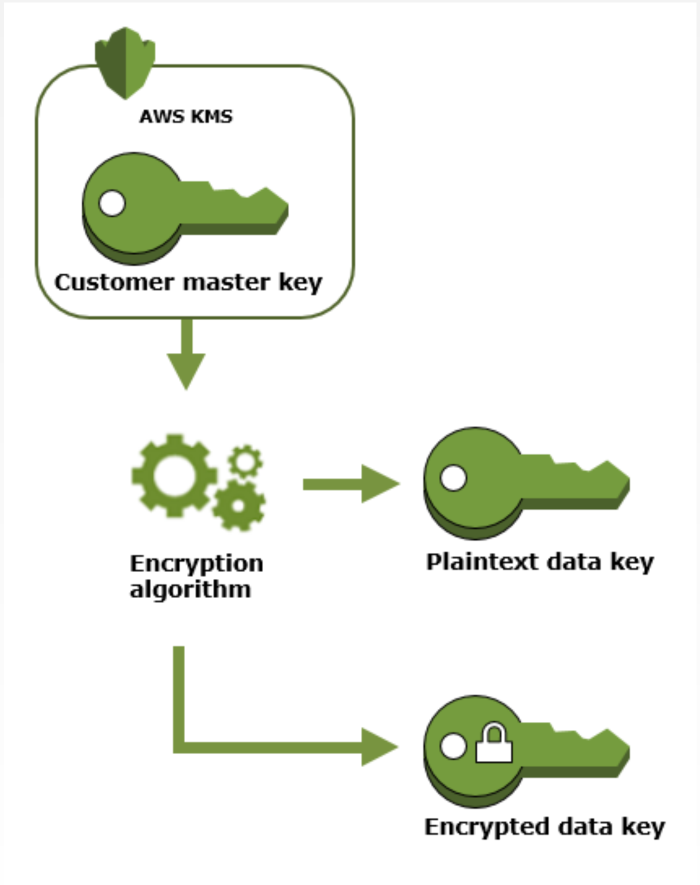
- “Masks” data for secure transmission or storage
  - Encrypt(data, encryption key) = encrypted data
  - Decrypt(encrypted data, decryption key) = original data
  - Without decryption key, the encrypted data is meaningless gibberish
- Symmetric Encryption:
  - Encryption key = decryption key; all authorized users know decryption key (a weakness).
  - 3DES, AES
- Public-Key Encryption: Each user has two keys:
  - User’s public encryption key: Known to all
  - Decryption key: Known only to this user
  - Used in RSA, DSA, PKCS

Symmetric Encryption

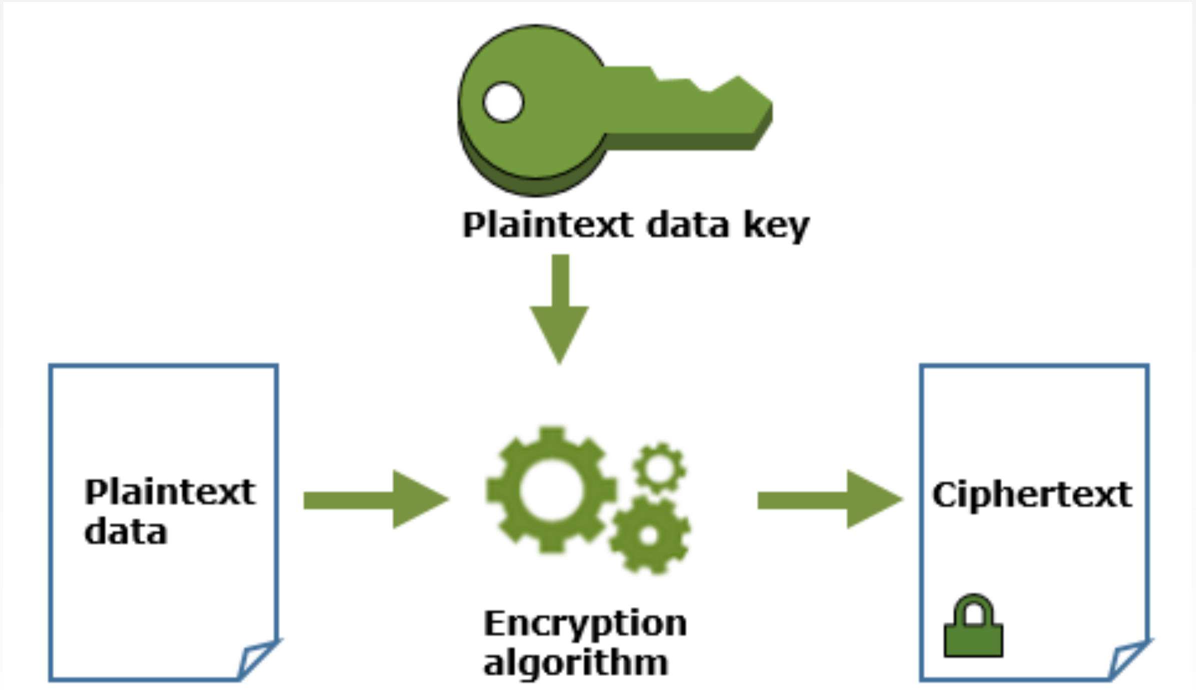


Data keys

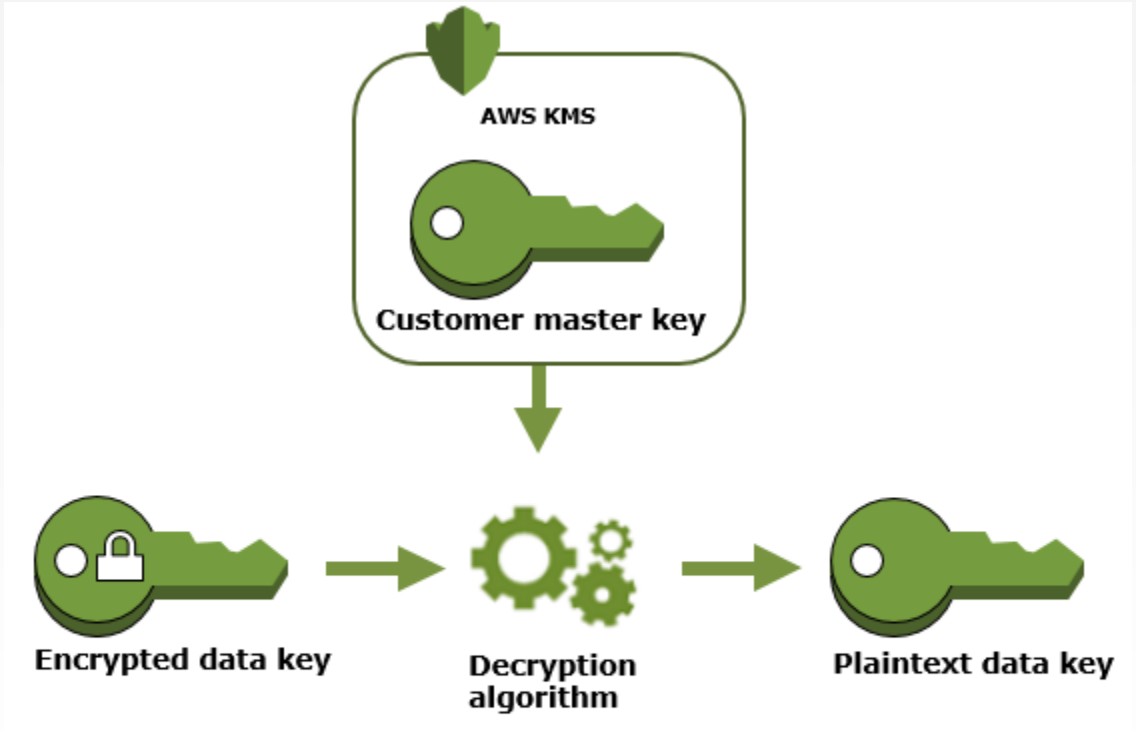
- Data keys are encryption keys that you can use to encrypt data, including large amounts of data and other data encryption keys



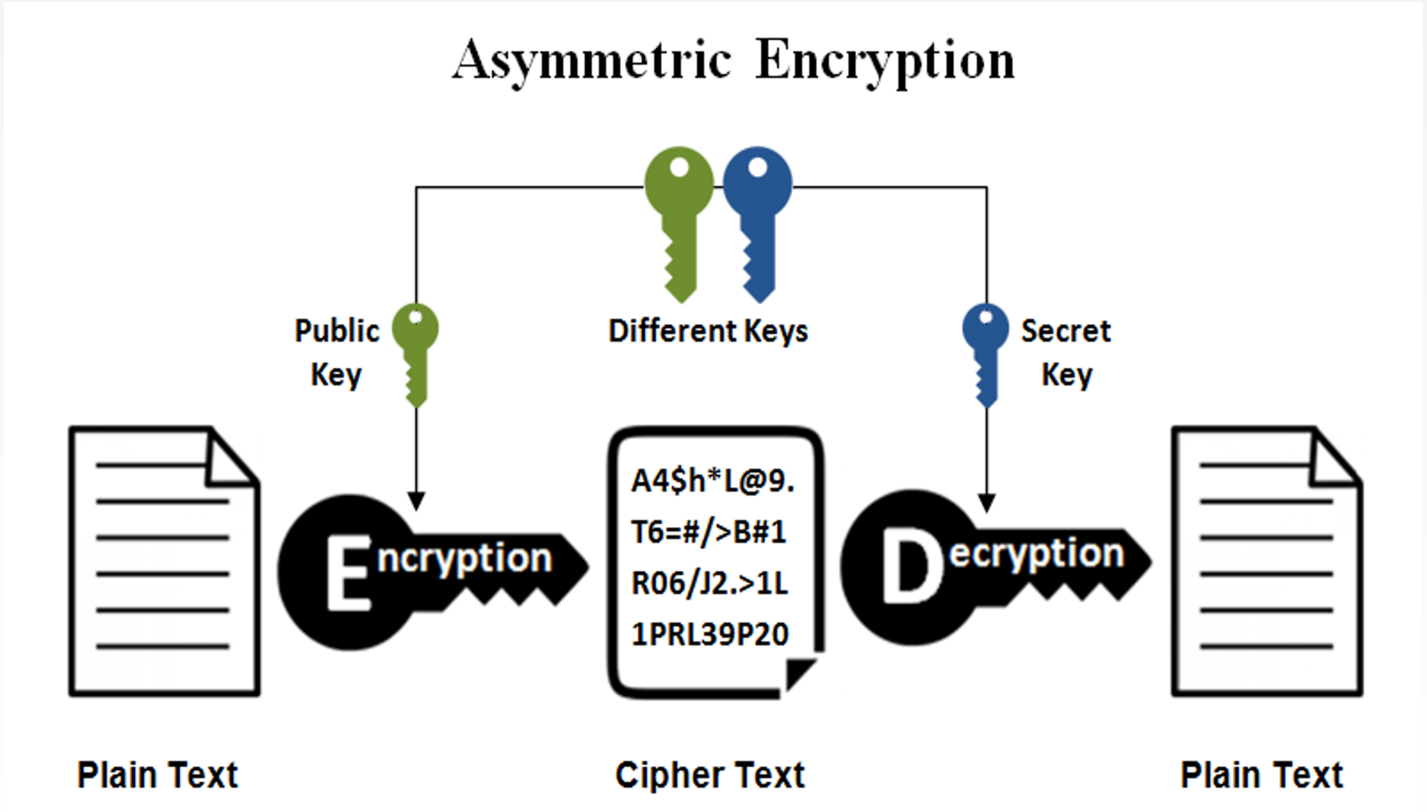
Encrypt data with a data key



Decrypt data with a data key

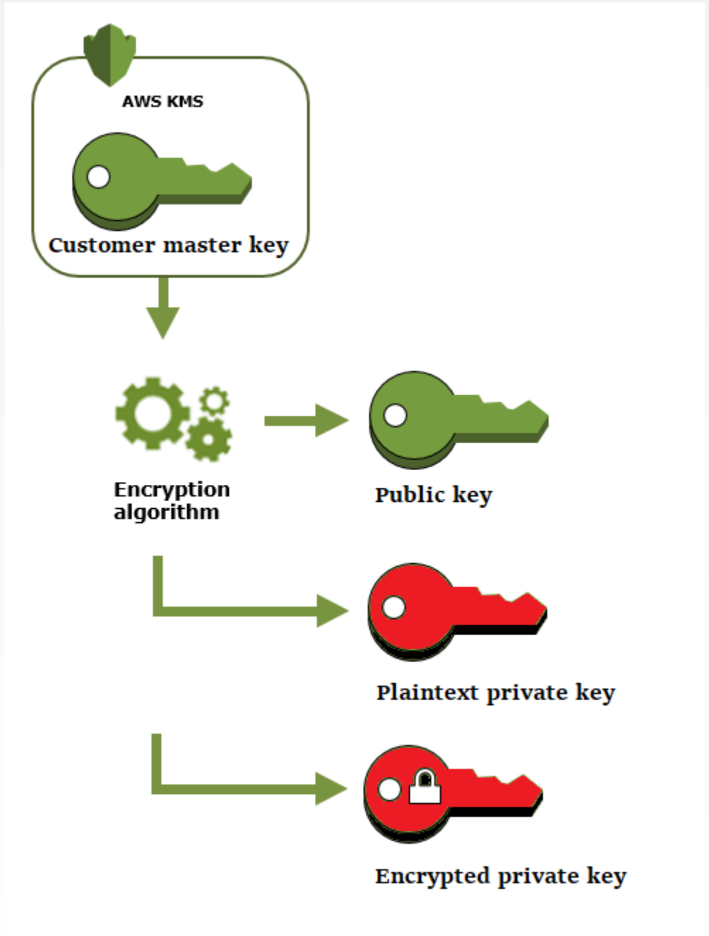


Asymmetric Encryption

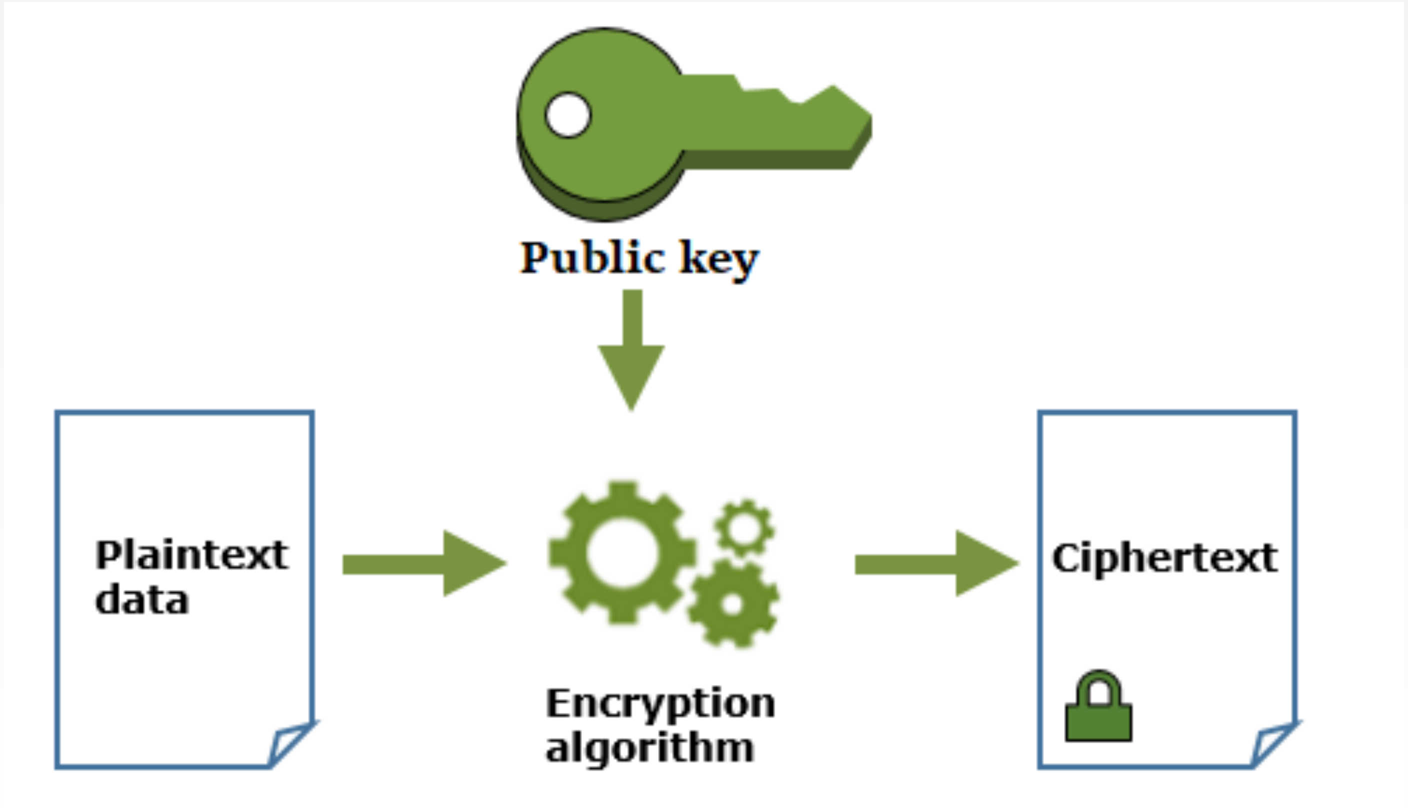


Data key pairs

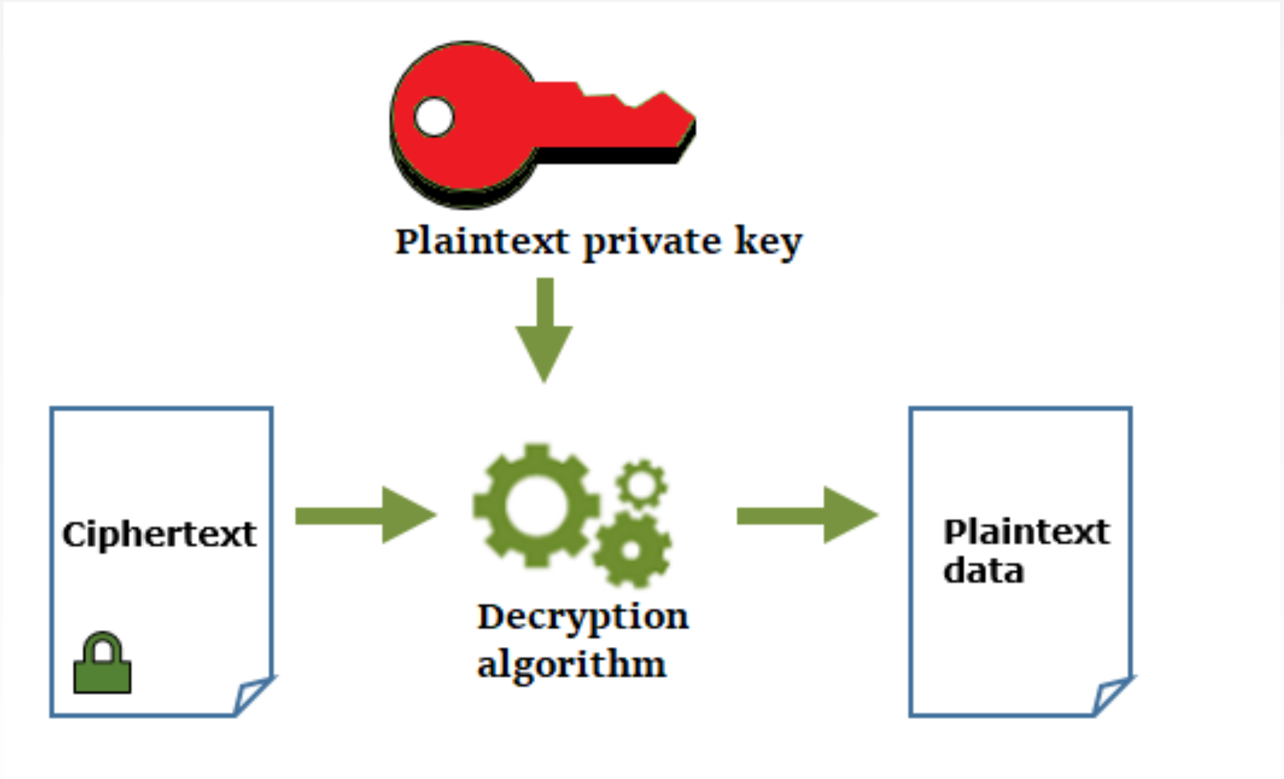
- Data key pairs are asymmetric data keys that consist of a mathematically-related public key and private key.



Encrypt data with a data key pair



Decrypt data with a data key pair



Application Security

Application Authentication

- Application Managed
  - Application logs into the database using an application/service account
  - User logs into the application - application verifies credentials and manages privileges
- Delegated
  - User logs into a delegated authentication provider (login.vt.edu) and is issued a ticket
  - User (browser) presents ticket to application
  - application validates ticket with authentication provider

Application Authentication

- Proxied
  - Application logs into the database using an application/service account
  - User login is application managed or delegated
  - Application connects to the database proxying as the user (which must exist in the database). Privileges are obtained through database grants.

SQL Injection

- A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application
- A successful SQL injection exploit can
  - Read sensitive data from the database
  - Modify database data
  - Execute administration operations on the database (such as shutdown the DBMS)
  - Recover the content of a given file present on the DBMS file system
  - Issue commands to the operating system.

SQL Injection Threat Modeling

- Spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.
- Very common with PHP and ASP applications. J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.
- In general, consider SQL Injection a high impact severity.

SQL Injection

- HTML Form:

```
<html>
      <body>

            <form action="welcome.php" method="post">
                  Username: <input type="text" name="user"><br>
                  Password: <input type="text" name="password"><br>
                  <input type="submit">
            </form>

      </body>
</html>
```

SQL Injection

- welcome.php

```
<?php
…
// Query to check the username and password
$sql = "SELECT user FROM users WHERE username = ‘“ . $_GET[“user"] . “’ AND
password = ‘” . $_GET[“password"] . “’”;

// Execute the query
$result = $conn->query($sql);
…
```

SQL Injection

- welcome.php continued

```
…
if ($result->num_rows > 0) {
        echo “Welcome “ . $_GET[“user"] . “!”
} else {
        echo “Access denied.”
}
```

SQL Injection

- If a user enters a valid username & password, access is granted. Otherwise they get an access denied message.
  - User: Alice
  - Password: MyPassword

```
SELECT user FROM users WHERE username = ‘Alice’ AND password = ‘MyPassword’;
```

SQL Injection

- What if the user enters
  - User: Alice
  - Password: a’ or 1=1; --

```
SELECT user FROM users WHERE username = ‘Alice’ AND password = ‘a’ or 1=1; --’;
```

Bind Variables!

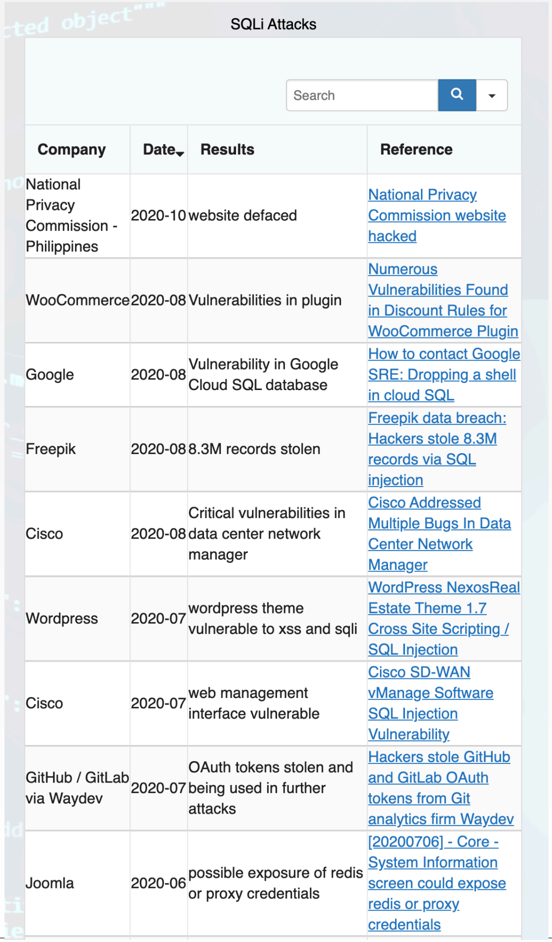
```
$sql = "SELECT user FROM users WHERE username = :user AND password = :pw”;
$sth = $conn->prepare($sql)
$sth->bindValue(':user', $_GET[“user"], PDO::PARAM_STR);
$sth>bindValue(':pw', $_GET[“password"], PDO::PARAM_STR);
$conn>execute();
```

OWASP SQLI Prevention Cheat Sheet

- Primary Defenses:
  - Option 1: Use of Prepared Statements (with Parameterized Queries) [aka bind variables]
  - Option 2: Use of Stored Procedures
  - Option 3: Whitelist Input Validation
  - Option 4: Escaping All User Supplied Input
- Check this out!
  - SQL Injection Prevention Cheat Sheet

SQL Injection Hall-of-Shame

- SQL injection is about 22 years old
- It is a threat even now
- https://codecurmudgeon.com/wp/sql-injection-hall-of-shame/


SQLi Attacks

Company	Date	Results	Reference
National Privacy Commission - Philippines	2020-10	website defaced	National Privacy Commission website hacked
WooCommerce	2020-08	Vulnerabilities in plugin	Numerous Vulnerabilities Found in Discount Rules for WooCommerce Plugin
Google	2020-08	Vulnerability in Google Cloud SQL database	How to contact Google SRE: Dropping a shell in cloud SQL
Freepik	2020-08	8.3M records stolen	Freepik data breach: Hackers stole 8.3M records via SQL injection
Cisco	2020-08	Critical vulnerabilities in data center network manager	Cisco Addressed Multiple Bugs In Data Center Network Manager
Wordpress	2020-07	wordpress theme vulnerable to xss and sqli	WordPress NexosReal Estate Theme 1.7 Cross Site Scripting / SQL Injection
Cisco	2020-07	web management interface vulnerable	Cisco SD-WAN vManage Software SQL Injection Vulnerability
GitHub / GitLab via Waydev	2020-07	OAuth tokens stolen and being used in further attacks	Hackers stole GitHub and GitLab OAuth tokens from Git analytics firm Waydev
Joomla	2020-06	possible exposure of redis or proxy credentials	[20200706] - Core - System Information screen could expose redis or proxy credentials

Summary

- Three main security objectives: secrecy, integrity, availability.
- DB admins are responsible for overall security.
  - Design security policy, maintains an audit trail, or history of users' accesses to DB.
- Two main approaches to DBMS security: discretionary and mandatory access control.
  - Discretionary control based on notion of privileges.
- SQL Injection (SQLi) is still a threat. Treat it seriously!
- When writing applications that interact with a database, use bind variables at least!

SQL Injection Demo

Reading and Next Class

- Security: Ch 21
- Next: Functional Dependencies Ch 19.1-19.3