SQL:

-Structured Query Language, azaz strukturált lekérdezési nyelv.
-Legtöbb relációs adatbázis által használt nyelv.
-Egyszerű, angol kifejezéseket használ.
-Információt táblázatok segítségével kezelhetők. Táblázat sorokra, oszlopokra, cellákra és rekordokra bontható.
-Relációs adatmodell, a táblázatok között kapcsolat létrehozása lehetséges. A táblázatok egymással valamilyen viszonyban vannak. Nagyvonalakban, egy táblázat egy adata utalhat egy másik táblázat adatára.
-A nyelv segítségével adatokat lehet beszúrni, törölni és frissíteni a táblázatokba. Különösen
-Sok rekord elérhető egyetlen paranccsal.
-Sok programozási nyelvvel ellentétben, rekordok elérhetőek indexelés nélkül is.

XML:

-Extensible Markup Language, azaz kiterjeszthető jelölőnyelv
-Szövegalapú jelölőnyelv, SGML-ből származik (Standard Generalized Markup Language)
-Hierarchikus (fa) felépítésű.
-Tartalomelemek úgynevezett nyitó és záró tag-eken (címkéken) belül értelmezhetőek. A tag-ek elnevezése a felhasználótól függ. Nincsenek előre definiált tag-ek.
-Olvasható, önleíró formátumú. Robosztus méreteket is elérhet. Rengeteg nyitó- és záró-tag, köztes tartalom és egymásba ágyazódás megnehezítheti az értelmezést.
-Célja az adatok szerializálása, azaz hordozása, tárolása és rekonstruálása.
-Általában két vagy több különböző rendszer közötti adatátvitelre használják.
-XML dokumentum struktúrájának leírására és elemeinek korlátozására, validálására XML Schema használható.

XMLSchema:
-XML dokumentum struktúrájának leírására szolgál.
-Feladata az XML dokumentum korlátozásainak meghatározása. Ilyen korlátozás lehet:
    -milyen elemek és tulajdonságok jelenhetnek meg az xml dokumentumban
    -gyerekelemek száma és sorrende
    -elem és tulajdonság adattípusai
    -elem és tulajdonság alapértelmezett és meghatározott (fixed) értékei.
-Megkönnyíti: -az xml dokumentumban megengedett tartalom meghatározását
    -adat helyességének validálását
    -adat minták (pattern) meghatározását
    -az adatokat különböző adattípusok között konvertálni.
-XML szintaktikát használ.
-Létrehozhatók saját típusok szabványos típusokból származtatva.
-Több sémára is lehet hivatkozni ugyanabban a dokumentumban.
-Adatátvitel esetén, az xml schema segítségével a feladó leírhatja úgy az adatokat, hogy a címzett megértse azokat.

Relációs és XML adatmodell:

1. XML dokumentum elemei közötti kapcsolat a dokumentum felépítési hierarchiáján keresztül értelmezhező.

Relációs modellben az adatok összefüggése a táblák közötti kapcsolatokkal határozható meg.

2. XML dokumentum önleíró, formátumó. A tartalmat befoglaló címkék elmondják, hogy milyen típusú adatról van szó. Egy dokumentumban több típusú adat is lehet.
   Relációs modellben az adatok típusát az oszlopdefiníció határozza meg. Egy oszlopban lévő összes adatnak egyező típusúnak kell lennie.
3. XML adatmodell jobban kezeli a 'dizájn változtatásokat', rugalmasabb felépítésű, mint egy relációs adatmodell.
   Relációs adatmodell eléggé merev formát követ, nehezebb lehet egy egész adatbázis felépítését, beleértve táblákat, oszlopokat, kapcsolatokat megváltoztatni, újakat létrehozni, mint XML adatmodell esetében, még ha szerkezete XML Schema-val kontrollált akkor is.
   Tehát ha az adatmodell gyakori változására számítunk, alkalmasabb lehet az XML használata.
4. XML adatmodell szerializálása / értelmezése költségesebb feladat, mint egy relációs adatmodellé.
5. XML dokumentum adatainak módosítása csak a teljes dokumentum cseréjével lehetséges. Ha egy nagyobb dokumentum kisebb részeit nagy gyakorisággal módosítjuk, akkor ésszerűbb lehet relációs adatmodellt alkalmazni, ha viszont kisebb dokumentumot frissítünk az XML adatmodell szintén hatékony lehet.
6. Komplexebb felépítésű adat XML dokumentumban könnyebben leírható (egymásba ágyazódás) , mint egy relációs adatbázisban (sok tábla). Komplex felépítésű adat alatt főképp olyan adatokra gondolok, amik felépítése között feltűnik például egymásba ágyazódás, fa felépítés, tömb szerkezet, stb…

<u>XML dokumentum tárolására alkalmas adatbázisok közül egy pár példa:</u>

1. IBM DB2
2. Microsoft SQL Server
3. Oracle Database
4. PostgreSQL

Léteznek úgynevezett natív XML adatbázisok is. Ezekről nem szeretnék részletesen írni, csak megemlítés szinten írok róluk a következő sorokban.

<u>Natív XML adatbázisok közül egy pár példa:</u>

1. BaseX
2. eXist
3. MarkLogic Server
4. Qizx
5. Sedna

Natív XML adatbázisok XML dokumentumokat, adatokat tárolnak. Nem SQL lekérdező parancsokkal működnek, hanem XPath utasításokkal.

Előny:

-Nagy mennyiségű XML típusú adat tárolására, lekérdezésére sokkal hatékonyabbak, mint relációs adatbázisok.

-Beállításukhoz nincs szükség táblák és egyéb bonyolultabb struktúrák létrehozásához.

Hátrány:

-Kevésbé elterjedtek, a relációs adatbázisok sok éve használtak, jól bevált adatbázisok.
-XPath nehezen tanulható, bonyolult szintaktikájú, míg az SQL olvasható, könnyebben értelmezhető, átlátható.

Az adatokat valahogy tárolni kell, vagy fájlként, vagy adatbázisba. Tegyük fel, hogy két eszközön használjuk ugyanazt az alkalmazást, három tárolási módot fogok ebben az esetben jellemezni.

Mivel személyes használatról beszélünk, feltételezhetjük, hogy az adat mennyisége nem hatalmas méretű, ezért a tárterületből való kifogyás lehetőségét minden esetben elvethetjük.

Lokális fájl

Az alkalmazás adatai a számítógépen tárolódnak. Szöveges formátumú adat tárolása esetén a leggyakrabban használt fájl formátumok a CSV, JSON, XML és YAML.

-Gyors és egyszerű hozzáférés az adatokhoz lokális tárolásból kifolyólag.

-Könnyedén implementálható megoldás, évtizedek óta alkalmazott módszer az adatok lokális fájlként való tárolása.

-Könnyedén átmásolható fizikai adathordozóra, így potenciálisan növelve a védelmet.

-Adathordozó elhagyása / lopása, vagy fájlok véletlen törlése biztonsági mentés hiánya esetén adatvesztést eredményezhet.

-Több fájl esetén gyakori jelenség az adatduplikáció.

Lokális adatbázis

Hasonló előnyei vannak, mint a helyi fájlnak.

-Egyszerű hozzáférés az adatokhoz, teljes adat kontrollálás.

-Nincs szükség internetkapcsolatra.

-Gyorsabb, mint egy távoli adatbázis, mivel gyorsabb a helyi lemez elérése, mint egy távoli esetében a hálózaton keresztüli kommunikáció.

-Adatok lekérdezése parancsokkal érhető el, így egyszerűen megtalálható a keresett információ nagy adatmennyiség esetén is. Ezzel szemben nagyobb méretű fájlok esetén nehezebb lehet a megfelelő adatok kiszűrése.

Hátrány:

-Nehéz az adatmegosztás külső résztvevővel (másik számítógép lokális adatbázisa).

-Ha az eszköz olyan állapotba kerül, hogy a fájlokhoz nem lehet hozzáférni, akkor elveszlik minden adat.

Távoli (felhőalapú / cloud) adatbázis

-Bárhonnan elérhető, internet elérés szükséges. Negatívuma ugyanebből származik, ha nincs internet, nem lehet elérni.

-Adatok nem helyileg a számítógépen tárolódnak, emiatt lassabb lehet a hozzáférés.Figyelembe kell venni, hogy más szerverek / alkalmazások is használhatják ugyanazt a hálózatot, ami szintén befolyásolhatja az adatok elérését.

-Teknikai hibák ellenére (számítógép elromlik) az információ biztonságba marad a távoli adatbázison.

Szóba került egy probléma többgépes alkalmazás készítése esetében, mégpedig az adatok szinkronizálása különböző eszközökön.

Az adatszinkronizálás egy olyan folyamat, ami alatt a különböző eszközökön eltárolt információkat megpróbáljuk össze egyeztetni, összhangba hozni.

Szükséges megoldani ezt a problémát alkalmazások készítésekor, hiszen nem szeretnénk ugyanazt az információt bevinni a rendszerünkbe, amit már egy másik eszközön egyszer megtettünk. Legtöbb esetben szeretnénk azonos adatokat elérni minden készüléken.

Programozási szempontból ez adatbázisok és/vagy fájlok szinkronizálását jelenti.

---Saját gondolat, nem forrásokból kiemelt tartalom---

A felsorolt tárolási módszerekből következtetve három esetet fogok jellemezni.

Tegyük fel, hogy két eszköz esetén az alábbi módon tárolódnak az adatok:

Lokális fájl – lokális fájl, lokális adatbázis – lokális adatbázis, távoli adatbázis.

Lokális fájl – lokális fájl

Első probléma két eszközön a fájlok megosztása.

Megoszthatók fizikai adathordozók használatával.

Ha ugyanazon a hálózaton találhatóak az eszközök, a megosztása egyszerűen megoldható.

Ha nem, szükség van valamilyen szolgáltatásra, ami mindkét eszköz esetében használható. Ilyen szolgáltatás internetelérés esetében lehet felhőalapú tárhely(pl.: Dropbox, Google Drive). Erről részletesebb leírás egy kicsivel lentebb található.

Ahhoz, hogy a fájlok tartalmát szinkronizáljuk, vagy egy külső alkalmazást veszünk segítségül, vagy saját alkalmazást készítünk külön erre a feladatra.

További problémák lehetnek:

-Ha az egyik fájlba benne van bizonyos adat, a másikba pedig nem, akkor az törlésre került, vagy újonnan létrehozott információ? Valamilyen megkülönböztetés szükséges.

-Két fájlban ugyanazok az adatok szerepelnek? Itt is valamilyen megkülönböztetés szükséges. Jó megoldás lehet az utolsó módosítási dátum hozzáadása minden fontosabb információhoz.

-Hogyha a két fájl különböző, nem azonos módon lett tárolva (pl.: egyik XML, másik JSON), valamilyen közös értelmezési megoldás szükséges. Ez ugyanazon alkalmazás használata esetében nem jellemző, inkább cégeknél, ha egy régebbi alkalmazás és egy újabb adatait kell egyeztetni.

Lokális adatbázis – lokális adatbázis

Néhány adatbázis-kezelő rendszer esetén létezik beépített funkció azonos típusú adatbázisok szinkronizálására.

Manuálisan ez egy nehezebb feladat és végrehajtását manuálisan tudom elképzelni, hasonló módon a két fájl szinkronizálásához.

Ha a fájlhoz hasonló adatmegosztási problémák sikeresen megoldhatók, a szinkronizációs program / script működését így tudnám elképzelni:

-Az adatbázisokba minden adat rendelkezik egyedi azonosítóval, illetve új adat bevitelekor vagy módosításakor egy tulajdonság mindig frissül. Például bevitelkor és szerkesztéskor lehet az adott időpontot megadni. Törléskor lehet egy boolean mezőbe eltárolni az adott elem állapotát, egy törölt elem például true lenne.

-Az adatokat XML / JSON vagy más adathordozó állománnyá konvertáljuk.

-Egyik fél megosztja az adatait ezen állomány segítségével a másik féllel.

-A befogadó fél szintén átkonvertálja az adatait az adott formátumra, majd egy összehasonlítás történik. Egyező azonosító esetén az adott elem tulajdonságai az utolsó módosításnak megfelelően változnak. Ha valami törlésre került, akkor itt is true legyen a törlési állapota.

-Ha ezek a lépések megtörténtek, a befogadó adatbázison elvégezhetőek a változtatások a dokumentum alapján.

Távoli adatbázis

Elegendő egy távoli adatbázis létrehozásaegy, kettő vagy akár több eszköz esetén is.

Megkönnyíti az adatszinkronizálást, hiszen mindegyik kliens(?) ugyanazzal az adatbázissal dolgozik. Ebből következik, hogy az adatok mindig szinkronizálva lesznek. Ha egy cella értékét megváltoztatja X felhasználó, akkor Y felhasználó is látni fogja azt a módosítást.

Ebben az esetben inkább az adattitkosítási problémák a jelentősek. Ennek kifejtése következő data-in-motion résznél.

---Saját gondolat vége---

---Talált szinkronizációs tartalom---

Szinkronizáció lehet egy- vagy kétirányú (one-way, two-way).

Egyirányú szinkronizáció:

Szokás még fájltükrözésnek (file mirroring), fájlreplikációnak (file replication) és fájlmentésnek (file backup) nevezni.

A fájlok várhatóan csak egy helyen változnak. A változtatások egyeztetése érdekében a szinkronizálási folyamat egy irányba másolja a fájlokat. A két tárolási helyszín nem tekinthető egyenértékűnek. Az egyik helyszín a forrás (source), a másik pedig a cél (target). Bármilyen változtatás a forrásba tükröződni fog a célba. A célon elvégzett változtatások nem fognak a forráson replikálódni.Ha ez a folyamat végigmegy, azt lehet mondani, hogy a forrás tükrözve van a célba.

Ez a módszer a forrás pontos másolatát hozza létre a célba. Hasznos és hatékony biztonsági mentés szempontjából, mivel csak a változtatott / új fájlok másolódnak.

Kétirányú szinkronizáció:

Gyors szinkronizálásnak is szokás nevezni (fast sync).

Ez a folyamat mindkét irányba másolja a fájlokat. A fájlok várhatóan mindkét helyen változnak, a két (vagy több) hely egyenértékűnek tekinthető.

Célja, hogy két vagy több hely azonos legyen egymással.

---Talált szink. tartalom vége---

Data-at-rest és data-in-motion / data-in-transit

Data-at-rest, szószerint nyugvó adatra lehetne fordítani olyan adatokat jelent, amelyek nem mozognak eszközről eszközre, vagy hálózatról hálózatra. Általában merevlemezen vagy pendrive-on tárolódnak.

A nyugalmi adatvédelem célja a bármilyen eszközön vagy hálózaton tárolt inaktív adatok védelme.

A nyugvó adatokat általában kevésbé sebezhetőnek tartják, mint a mozgásban lévő adatokat (data-in-motion), gyakran értékesebbnek is találják ezek tartalmát. Nyugvó adatok esetében az ellopható információ mennyisége sokkal nagyobb lehet mint az éppen úton levő adatoké.

Adatok biztonsága a szükséges óvintézkedések megtételétől függ.

Egy cég legtöbb esetben adatit saját hálózatán belül tárolja, azok mégis veszélyben lehetnek rosszindulatú külső és belső fenyegetések miatt. Egy betolakodó könnyedén hozzáférhet egy cég adataihoz, ha sikerül jogtalanul hozzáférnie egy számítógépükhöz vagy egy lopott eszközt feltör.

Data-at-rest típusú adatok védelme érdekében az egyik legjobb és legegyszerűbb módszer ezek titkosítása.

További biztonsági lépések lehetnek:

-Operációs rendszerek natív adattitkosítási eszközeinek használata (Windows BitLocker, macOS FileVault) merevlemez titkosítására, hogy lopás esetén a megfelelő kulcs ismerete nélkül se tudjanak hozzáférni az adatokhoz.

-Úgynevezett DLP (Data Loss Prevention) megoldások is alkalmazhatóak. Letilthatóak vagy korlátozhatóak az USB-k, mobilok vagy tároló meghajtók csatlakozása. Így nem lehet rosszindulató USB-ket csatlakoztatni az eszközhöz, hogy megfertőzzék, meggátolják az adatok adattárolón keresztüli szivárgását is.

Data-in-motion / Data-in-transit, magyarul mozgásban lévő adatnak vagy tranzitadatnak olyan adatokat nevezünk, amelyek aktívan mozgásban vannak egyik helyről a másikra vagy az interneten vagy magánhálózaton keresztül.

Mivel az adatok mozgásban vannak, ezért kevésbé biztonságosnak tekinthetők. Célja olyan adatok védelme amelyek például belső hálózaton belül mozognak vagy helyi tárolóeszközről felhőtípusú tárolóeszközre.

A dolgozat szempontjából ez kimondottan a helyi tárolóeszközről felhőtípusú tárolóeszközre való átvitel miatt fontos.

Tranzitadat esetében is egy kiváló biztonsági intézkedés az adatok titkosítása. Védi az adatokat, ha a két fél között a kommunikációt 'lehallgatják', miközben az adat a felhasználó és a felhőszolgáltató között mozog.

Ez a védelem az adatok titkosításával érhető él még mozgatás előtt, ez lehet talán a legfontosabb része a mozgásban lévő adatok védelmének, illetve a megfelelő kulcskezelés, amiről lejjebb található leírás. Végpontok hitelesítése és adat érkezésekor való visszafejtése és ellenőrzése is tovább fokozhatja a védelmet.

Adatbázis titkosítás

(11.link)

Egy olyan folyamatot nevezhetünk adatbázis titkosításnak, ami egy algoritmus segítségével az adatbázisban tárolt adatokat titkosított szöveggé (cipher text) alakítja, ami értelmezhetetlen a megfelelő kulcs ismerete nélkül.

Célja, hogy megvédje az adatainkat a potenciális fenyegetések ellen. Hogyha egy hacker valahogy sikeresen feltöri az adatbázist, akkor számára értéktelen, értelmezhetetlen szöveggel fog találkozni.

Többféle titkosítási technika is létezik, melyek közül a legleterjedtebbek:

Transparent Data Encryption (TDE) (Átlátható adat titkosítás):

(github linkek)

-Teljes adatbázist, úgynevezett 'nyugvó adatokat' (data-at-rest) titkosít, merevlemezen és a biztonsági mentési adathordozón is. Használatban és szállításban lévő adatokat nem véd (data-in-use, data-in-transit).

-A módszer biztosítja, hogyha még el is lopják a fizikai adathordozót, akkor sem férnek hozzá a tolvajok a rajta lévő adatokhoz.

-Mivel az összes adatot titkosítja, ezért nem szükséges speciális módon rendezni az adatokat.

-Adatok titkosítási a tároláskor történik, visszafejtésük pedig rendszer memóriába való hívásakor történik.

-Szimmetrikus kulcsot használ a kódoláshoz.

-„A vállalatok jellemzően a TDE-t alkalmazzák az olyan megfelelési problémák megoldására, mint például a PCI DSS, amely megköveteli a nyugalmi adatok védelmét."

-Microsoft, Oracle, IBM is alkalmazza ezt a módszert az adatbázis fájlok titkosítása érdekében

Column Level Encryption (Oszlop szintű titkosítás)

(github linkek)

-Relációs adatmodell esetén egy adatbázis táblákból, oszlopokból, sorokból és cellákból vagy mezőkből áll. Ahogy a nevéből is következik, ez a módszer egy ilyen adatbázis egy sorát titkoítja.

-Független oszlopokat lehet titkosítani. Az oszlop összes adatát titkosítja kivétel nélkül.

-Előnye, hogy könnyedén megkülönböztethető az érzékeny és a nem érzékeny adat, illetve külön kulcs használható minden oszlop titkosításához, így növelve a biztonságot. Sokkal rugalmasabb, mint a teljes adatbázist titkosító TDE.

-Hátrány az előnyeiből fakad. Több oszlop több kulccsal való titkosítása az adatbázis teljesítményének csökkenéséhez vezethet. Lassabban lehet keresni és indexelni is.

-Akkor használatos, ha nincs szükség teljes adatbázis titkosításra, hanem megkülönböztethető, hogy mely oszlopok tárolnak érzékeny adatot és melyek nem.

-Microsoft, Oracle, IBM , MyDiamo és még sok más cég használja ezt a titkosítási módszert.

Field Level Encryption (Cella szintű titkosítás)

(github linkek)

-Relációs adatmodell esetén használható.

-Kiválasztható, hogy pontosan melyik mezőt szeretnénk titkosítani.

-Előnyei és hátrányai megegyeznek az előbb ismertetett Column Level Encryption-nel.

-Nincs minden esetben szükség a mezők dekódolására, lehetőség van egyenlőség vizsgálatra.

-Akkor használatos, ha nincs szükség teljes adatbázis titkosításra, hanem megkülönböztethető, hogy mely cellák tárolnak érzékeny adatot és melyek nem.

-Microsoft, Oracle, IBM , MyDiamo és még sok más cég használja ezt a titkosítási módszert.

További titkosítási formák, amik már nem pontosan az adatbázis titkosításhozsorolhatók.

Filesystem Encryption

(8.link)

-Fájlrendszer titkosítás, szokás még fájl / mappa titkosításnak, FBE-nek (file-based encryption) is nevezni.

-Célja adott fájl tartalmának titkosítása

-Előnye, hogy minden fájlt külön kulccsal lehet titkosítani, így növelve a biztonságot.

-A kriptográfiai kulcs addig van a memóriában, amíg az adott fájl meg van nyitva.

-Aki fizikailag hozzáfér a tároló számítógéphez, az láthatja, hogy milyen nevű fájlok találhatóak a rendszeren, holott a tartalmukat nem tudja megnézni, amíg nem ismeri a kulcsot.

-Olyan adatokat is képes titkosítani, amelyek nem részei az adatbázis rendszernek.

-Csökkenti a teljesítményt és operációs rendszer hozzáférést is kíván a használatához.

-Teljesítményproblémák miatt nem igazán alkalmazzák, de ennek elenére kis felhasználószámú rendszerek esetében ajánlják.

Full Disk Encryption

(7.link)

-Teljes merevlemez titkosítás

-Az egész merevlemez tartalma titkosításra kerül.

-Általában ugyanazt a kulcsot használja az egész meghajtó titkosításához, ezért futásidőben az összes adat visszafejthető. Néhány módszer több kulcsot is használ különböző 'fejezetek'(?) titkosításához.

-Nagy hátránya, ha a támadó futásidőben fér hozzá a számítógéphez, minden fájl elérhető számára.

Application Level Encryption

-Kódolás és dekódolás adatátvitel és tárolás előtt történik.

-Maga az alkalmazás végzi a titkosítási folyamatot.

-Az adat csak a megfelelő alklamazáson keresztül érhető el. Egy hacker-nek szüksége van az adatbázis és az adatokat használó alkalmazásra is az adatok visszafejtéséhez.

-Negatívuma lehet, ha ezt a titkosítási módszert szeretné alkalmazni egy cég, akkor maguknak kell implementálniuk, ami egy nem informatikai cég esetében nehéz lehet.

-Másik hátránya, a kulcsok kezelésének az összetettsége is megnövekedhet, ha több különböző alkalmazásnak kell egy adatbázishoz hozzáférnie, írnia, olvasnia.

Mindegyik korábban bemutatott módszer valamilyen algoritmust használ az adatok titkosításához. Egy titkosítási algoritmus lehet szimmetrikus vagy aszimmetrikus. Ez a két fogalom a kulcs és a titkosított szöveg közötti kapcsolatot írja le.

(13.link)

Szimmetrikus titkosítás:

-Az adatok titkosítása az adatbázisba történő mentéskor és visszafejtéskor történik.

-Adat titkosítására és dekódolására egy privát kulcsot használ.

-Az adatok megosztásához a címzettnek rendelkeznie kell a visszafejtési kulcs másolatával. Ez a titkosítás legegyszerűbb, legrégebbi és legismertebb fajtája.

-A hátránya a kulcs megosztásából ered, ha a visszafejtési kulcs kiderül, az adatszivárgáshoz (data leak) vezethet.

-Előnye a sebesség.

Aszimmetrikus titkosítás:

-Egy privát és egy publikus kulcsot használ.

- A publikus kulcs lehetővé teszi, hogy bárki titkosítsa az adatokat.

-A privát kulcs szükséges az adatok dekódolásához. Ez minden felhasználó esetében különböző.

-Biztonságosabbnak vélhető, mivel a privát kulcsok nem kerülnek megosztásra, de ugyanakkor nagyobb a számítási költsége is.

Kulcskezelés / Key-management

Ez a fogalom a kulcs előállítását, cseréjét, tárolását és használatát jelenti.

Kulcsok nem megfelelő tárolása és kezelése adatszivárgást eredményezhet. Ha a kulcskezelő rendszer valamilyen oknál fogva elveszti vagy törli a kulcsokat, akkor a titkosított adatok is elvesznek.

Komplexitás szempontjából, minél több alkalmazás adata kerül titkosításra, úgy nő a tárolandó és kezelendő kulcsok száma is.

Kulcsok tárolása a leggyakoribb, hogy egy titkosítási alkalmazás kezeli a kulcsokat és a hozzáférés jelszóval megadásával lehetséges.

Kimondottan a kulcskezelésre számos úgynevezett kulcskezelő rendszert (KMS) létrehoztak. Egy ilyen rendszer magába foglalja a kulcsok generálását, cseréjét, tárolását.

Szimmetrikus vagy aszimmetrikus titkosítási algoritmusnak olyan matematikai folyamatot nevezünk, amikor egy szimpla szöveget (plain text) az adott módszer kódolt szöveggé transzformál (cipher text).

Pár oldallal ez előtt bemutatott módszerek ilyen algoritmusokat használnak adat titkosításához.

Az, hogy egy algoritmus szimmetrikus vagy aszimmetrikus itt is a kulcs és a kódolt szöveg közötti kapcsolatot jelenti.

Szimmetrikus titkosítási algoritmusok közül gyakoriak:

Advanced Encryption Standard (AES)

-Bevált, megbízható titkosítási módszer. Komplex, több fázisból álló matematikai számításokat hajt végre.

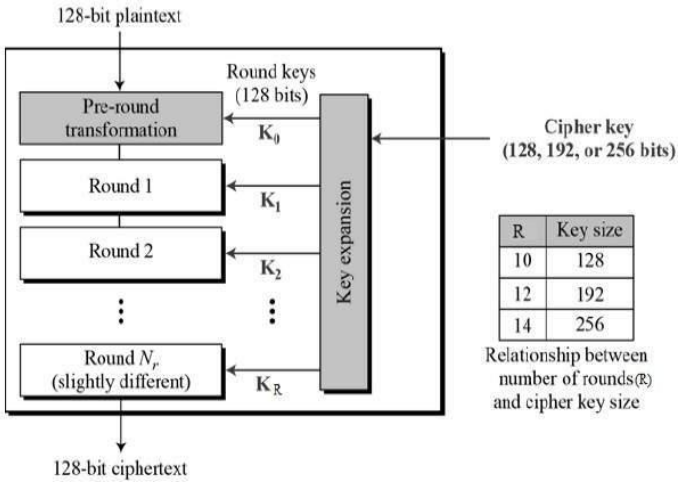
-Hátránya, hogy szoftveresen nehezen implementálható, illetve komplexitása és nagysága miatt teljesítményi ideje is nagy.

-Úgynevezett block cipher, azaz blokk titkosítás. Blokkok mérete 128 bit.

-3 fajtája van:
AES-128 – 128 bit nagyságú titkosítási kulcsot alklamaz.
AES-192 – 192 bit nagyságú titkosítási kulcsot alkalmaz.
AES-256 – 256 bit nagyságú titkosítási kulcsot alkalmaz.

-Számítást bájtokon nem pedig biteken végez. 128 bites blokkot az algoritmus 16 bájtként kezel. Ezt a 16 bájtot 4 sorba és oszlopba rendezi a mátrixműveletekhez.

-Kulcs méretétől függ, hogy hány kört fog az algoritmus elvégezni. 128 bites kulcs – 10 kör, 192 bites kulcs – 12 kör, 256 bites kulcs – 14 kör.



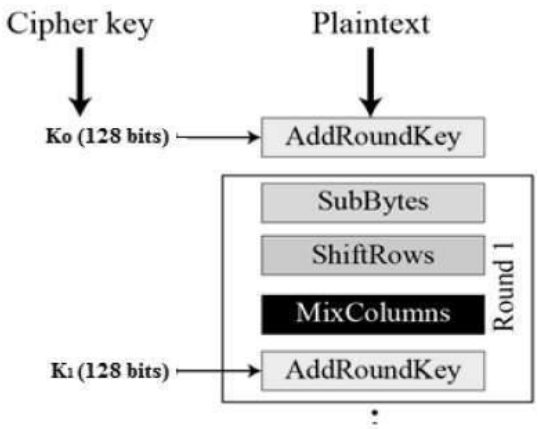
Mindegyik kör egy másik 128 bites kulcsot használ, amit az eredeti AESkulcsból számolnak ki.

-A titkosítási folyamat minden köre a további négy lépésből áll:

1. Byte helyettesítés (SubBytes)

A 16 bemeneti bájtot a megadott fix táblázat (Rijndael S-box) megfelelő értékeivel helyettesíti. Az eredmény egy négy sorból és négy oszlopból álló mátrix.

2. Sorok eltolása (ShiftRows)



A mátrix minden sora balra tolódik. A baloldalt kieső értékeket a mátrix jobb oldalára helyezi. Az eltolás menete:
Első sort nem toljuk el.
A második sor egy byte pozícióval balra tolódik.
A harmadik sor két byte pozícióval balra tolódik.
A negyedik sor három bytepozícióval balra tolódik (azaz eggyel jobbra).
A byte eltolások adják meg az eredmény mátrixot.

3.Oszlopok összekeverése (Mix Columns)

Minden oszlop négy bájtját egy invertálható lineáris transzformáció szerint módosítja. A függvény bemeneteke az adott négy bájt, kimenetként négy teljesen új bájtot ad. Ez a 4 bájt az előzőek helyettesítésére szolgál, az egész mátrixon ezt elvégezve megkapjuk a 16 új byte-ból álló eredmény mátrixot.

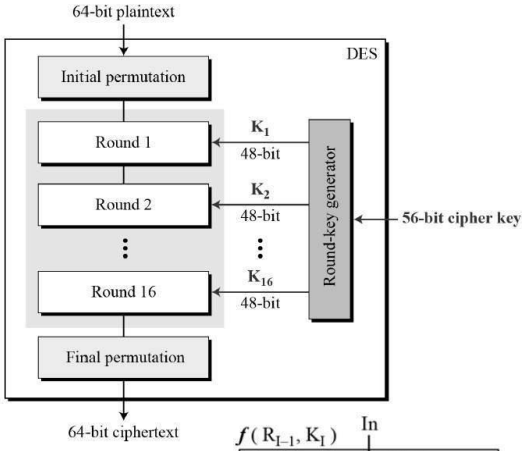
Ez alatt a lépés alatt minden oszlop megszorzásra kerül egy előre meghatározott mátrixszal.

4.Kör kulcs hozzáadás (AddRoundKey)

A mátrix 16 bájtját most 128 bitnek tekinti majd XOR-ozza a kör 128 bites kulcsával. Ha ez az utolsó kör, akkor a kimeneti értéket kapjuk, ha nem, akkor a kapott 128 bitet megint 16 bájtként értelmezi és egy újabb kört kezd.

-Visszafejtés a titkosítási folyamat fordított sorrendben történő elvégzése. A bemutatott négy lépést kell visszafele elvégezni, AddRoundKey-MixColumns-ShiftRows-ByteSub

Data Encryption Standard (DES)



-Block cipher, egy blokk 64 bit nagyságú.

-64 bit nagyságú sima szöveg titkosítás után 64bit nagyságú titkosított szöveget eredményez.

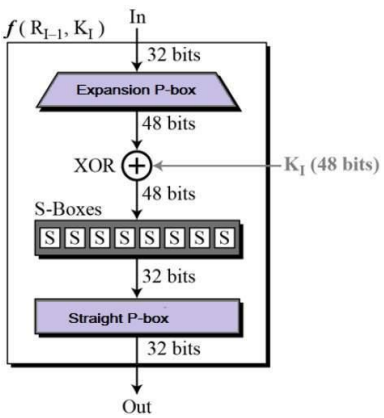
-16 sorozatot véget el matematikai számításokból, mindegyikhez külön titkosítási kulcsot használ.

-Kulcsok mérete 56 bit (64 bit, de 8-at közülük nem használ).

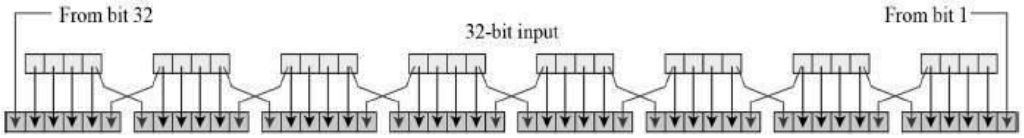
-Hátránya lehet, hogy a titkosítás és visszafejtés ugyanazzal az algoritmussal és kulcsokkal történik.

-Feistel kódon alapul.

-Kör funkciók (Round function): A titkosítás lényege itt történik. A DES függvény egy 48 bites kulcsot alkalmaz a jobb szélső 32 bitre, hogy 32 bites kimenetet kapjon.



Bővítési permutációs doboz (Extensible Permutation Box) - Mivel a jobb bemenet 32 bites, a kör kulcsa pedig 48 bites, először a jobb bemenetet 48 bitesre kell bővítenünk.

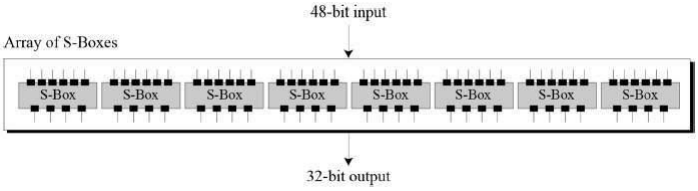


A permutációs logika általában táblázatként van leírva a DES specifikációban:

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

Következő lépés a XOR: Az elvégzett bővítés után az algoritmus XOR műveletet végez a kibővített jobboldali részen és a kör kulcsán. A kör kulcsát csak ebben a műveletben használja.

Helyettesítő dobozok (Substitution boxes): Az S-dobozok végzik az összekeverést. A DES 8 S-dobozt használ, mindegyik 6 bites bemenettel és 4 bites kimenettel.



Egyenes permutáció (Straight Permutation): Az S-dobozok 32 bites kimenetét ezután egyenes permutációnak veti alá.

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Triple DES

-Működése megegyezik a DES működésével.

-3x16 sorozatot végez, így biztonságosabbnak mondható.

-Hátránya lehet, hogy a titkosítás és visszafejtés ugyanazzal az algoritmussal és kulcsokkal történik.

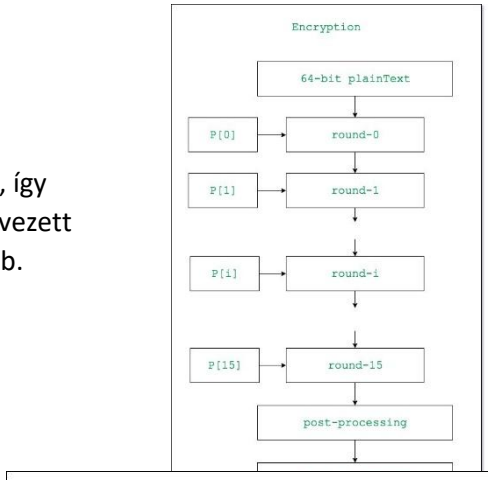
Blowfish

-Block cipher, 64 bit méretű blokkok.

-Kulcsok mérete 32 bittől 448 bit-ig terjed, változó méretűek, így lehetőséget nyújt személyes és ipari felhasználásra is. Úgynevezett alkulcsokat is használ (subkey), számszerint 18-at. Ez a P tömb.

-Sokkal gyorsabb mint a DES és Triple DES.

-Lépések:



32-bit hexadecimal representation of initial values of sub-keys

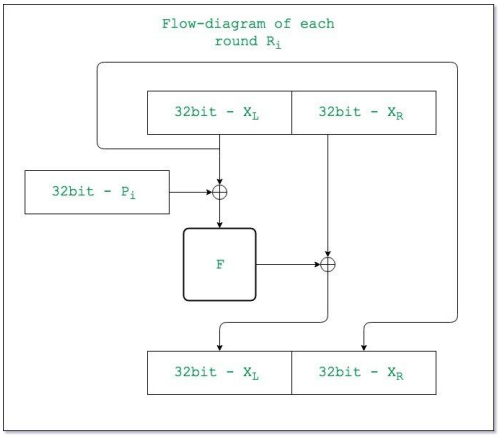
P[0] : 243f6a88	P[9] : 38d01377
P[1] : 85a308d3	P[10] : be5466cf
P[2] : 13198a2e	P[11] : 34e90c6c
P[3] : 03707344	P[12] : c0ac29b7
P[4] : a4093822	P[13] : c97c50dd
P[5] : 299f31d0	P[14] : 3f84d5b5
P[6] : 082efa98	P[15] : b5470917
P[7] : ec4e6c89	P[16] : 9216d5d9
P[8] : 452821e6	P[17] : 8979fb1b

1.Alkulcsok generálása – 18 darab (P[0] – P[17]), mind a titkosítási és visszafejtési folyamathoz is szükségesek. A 18 alkulcsot egy P-táblázatban tároljuk, amelynek minden egyes eleme egy 32 bites bemenet.

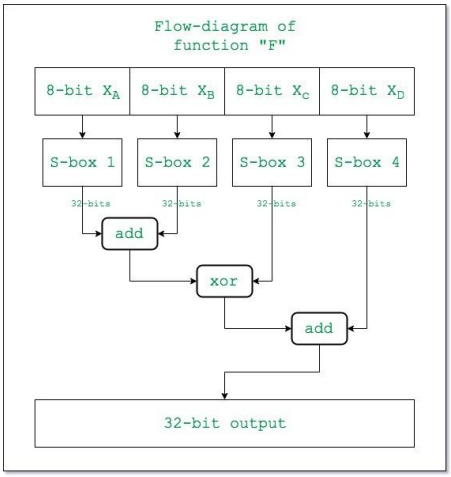
Minden egyes alkulcsot megváltoztatunk a megfelelő bemeneti kulcshoz képest. (XOR művelet)

Ennek eredményeképpen megkapjuk a P sorozatot, amit a teljes titkosítási folyamat alatt használ az algoritmus.

2.Titkosítás – 16 körből áll, minden kör (Ri) az előző körből származó sima szöveg (P.T.) és a megfelelő alkulcs (Pi) bemenetét veszi.



Az F függvény leírása:



Aszimmetrikus titkosítási algoritmusok közül gyakoriak:

RSA (Rivest–Shamir–Adleman)

-Azon az elven alapul, hogy a nagy számok szorzása könnyű, de a nagy számok tényezőkre bontása nehéz.

-A publikus kulcs két számból áll, ami közül az egyik két nagy prímszám szorzata.

-A privát kulcs egyik száma ugyanennek a két prímnek a szorzata.

-Lépések:

1. RSA modul generálása – Két prímszám, p és q kiválasztásával kezdődik. Ezek szorzatából adódik n értéke (n = p * q). N a megadott nagy szám.

2. Származtatott számok (derived numbers)

Tekintsük az e számot származtatott számnak, amelynek nagyobbnak kell lennie, mint 1 és kisebbnek, mint (p-1) és (q-1). A fő feltétel az, hogy a (p-1)-nek és a (q-1)-nek ne legyen közös tényezője, kivéve a 1-et.

3. Nyilvános kulcs (public key) – A kapott n és e számpár alkotja a publikus RSA kulcsot.

4. Privát kulcs (private key) – A d privát kulcsot a p, q, e számokból számítja ki a következő módon:

ed = 1 mod (p-1) (q-1), amíg másik értéke az n.

Titkosítás: C = m^e mod n, ahol C atitkosítandó szöveg, m a sima szöveg, e és n a publikus kulcs számpárai.

Visszafejtés: Plaintext = C^d mod n, ahol C a titkosított szöveg, d és n a privát kulcs értékei. Plaintext a sima visszafejtett szöveg.

Digital Signature Algorithm (DSA)

-Digitális aláírások titkosítására használt szabvány.Digitális üzenet hitelesítésére is alkalmas.

-Moduláris exponenciáláson (modular exponentiation) alapul, a diszkrét logaritmus (discrete logarithm) problémájával együtt.

-Privát kulcsot egy üzenet digitális aláírásának generálására használják. Ellenőrizni az aláíró publikus kulcsával lehetséges.

Elliptic Curve Cryptography (ECC)

(15.link)

-RSA-val vetélkedő aszimmetrikus titkosítási módszer.

-Kulcs generálása matematikai elliptikus görbék segítségével történik.

-„Véges mező feletti sík görbe, amely az egyenletet kielégítő pontokból áll: $y^2=x^3 + ax + b$.”

-Alapvető kulcs méret 256 bit, de görbétől függően változhat.

A titkosítási módszerek közül kimaradt, az úgynevezett **hashing**. Ez nem sorolható kimondottan sem az adatbázis sem a fájlrendszerek titkosításához. A hashing jellemzői a következők:

-Érzékeny adatok tárolására használják, mint a jelszavak.

-Egyediek és ismételhetőek, ami azt jelenti, hogy egy szó ugyanazzal a hash algoritmussal transzformálva ugyanazt a titkosított szöveget fogja eredményezni.

-Egy algoritmus mindig ugyanolyan méretű kimenetet állít elő.

-Nagyon nehéz két olyan különböző szót találni, amelyek ugyanazon hash algoritmussal transzformálva ugyanazt a titkosított szöveget eredményeznék.

-Egy hash algoritmussal transzformált szöveget visszaalakítani egyszerű, értelmes szöveggé (plain text) már nem lehet, éppen ezért használják jelszavak titkosítására.

-Nem visszaalakíthatósága miatt egyezőség vizsgálatát lehet megnézni két ugyanazon hash algoritmussal transzformált , kódolt szöveg között. Ha két hash egyezik, akkor ugyan az volt a bemeneti szöveg, ha nem egyeznek, akkor különbözőek voltak.

-Úgynevezett salt és pepper –rel szokás a hash-eket ellátni.

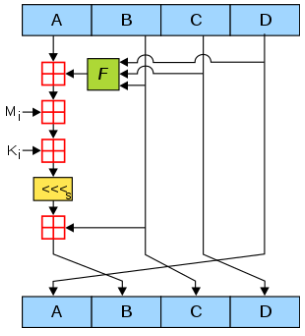
-Salt: A titkosítandó szövegrészhez extra szöveg csatolása bonyolultság növeléséhez. Regisztrációkor lehet például az email címet és jelszót kombinálni, majd a kombinált szövegen elvégezni a hash algoritmust. Ilyen adatokat salted hash-nek nevezik.

-Pepper: Salted hash adathoz , tehát már titkosított adathoz fűz további értékeket. Egy adatbázis esetébe ez általában megegyező értéket jelent, azaz minden hash-elt adathoz ugyanaz a pepper érték lesz hozzáadva.

A legismertebb hash algoritmusok közül bemutatok párat:

-MD5 (Message Digest 5):
Régebben gyakran használt funkció, ami egy 128 bites kimenetet állít elő.

Kiderült róla, hogy nem ütközésálló (collision resistant), emiatt kriptográfusok más hash algoritmusokat ajánlanak. Egy hash függvényre akkor mondjuk, hogy ütközésálló, hogyha nehéz olyan két különböző bemenetet találni, amely ugyanazt a kimenetet eredményezi.



Az algoritmus először a bemeneti üzenetet 512 bites (16 32bites szó) blokkokra osztja. A bemenetet kitölti, hogy hossza mindig osztható legyen 512-vel. Ez a következő módon történik: először egy bit 1-est csatol a szöveg végére. Ezután nullákat szúr be az egyes után, amíg a blokk mérete el nem éri az 512-64, azaz 448 bitet. A maradék 64 bit helyére az eredeti üzenet hossza modulo 2^64 megoldása kerül.

Az algoritmus fő része egy 128 bites állapoton (?state) dolgozik, ami négy 32 bites szóra oszt fel (A,B,C,D). Ezek előre meghatározott konstansokkal

inicializáltak.

Ezután minden 512 bites blokkot felhasznál az állapot megváltoztatására.

Az üzenetblokk feldolgozása négy hasonló körből áll, minden kör 16 hasonló műveletből, amelyek egy F nemlineáris függvényen, moduláris összeadáson és balra forgatáson alapulnak.

Négy lehetséges függvény van, minden forduló mást használ:

$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$
$$H(B,C,D) = B \oplus C \oplus D$$
$$I(B,C,D) = C \oplus (B \vee \neg D)$$

-SHA (Secure Hashing Algorithm) Family

Hat különböző hash függvényből áll: SHA-0, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512.

Változó méretű bemenetet alakítanak fix méretű kimentté.

Az első négy 512 bites blokkokat használ 32 bites szavakra osztva, az utolsó kettő pedig 1024 bites blokkokat 64 bites szavakra bontva.

Kimenet mérete SHA-0 és 1 esetén 160 bit, SHA-224 esetén 224 bit, SHA-256 esetén 256 bit, SHA-384 esetén 384 bit, SHA-512 esetén 512 bit nagyságú.

Mindegyik algoritmus hasonlóképpen működik.

Eredmény mindig 160 bit hosszú. Eredeti üzenet hosszának kevesebbnek kell lennie, mint 2^64-en bit.

A SHA-2 családot, aminek a 256, 384, 512 is a tagja, szélés körben implementálták biztonsági alkalmazásokban és protokollokban, mint a TLS, SSL, PGP, stb.

A SHA-256-ot a Debian szoftvercsomag hitelesítésére használják és DKIM üzenetaláírási standard. Linux és Unix gyártók 256 és 512 bites SHA-2 használatára tértek át a biztonságos jelszótároláshoz.

Számos kriptovaluta, köztük a Bitcoin is használja a SHA-256-ot.