# Kriptográfia összefoglaló - ez az amit már kérdezett mástól

## Kis fermat

**Tétel**
Let $p$ be a prime which does not divide the integer $a$, then $a^{p-1} \equiv 1 \pmod{p}$.

**Bizonyítás**
Start by listing the first $p$-1 positive multiples of $a$:

$a$, $2a$, $3a$, ... $(p-1)a$

Suppose that $ra$ and $sa$ are the same modulo $p$, then we have $r = s \pmod{p}$, so the $p$-1 multiples of $a$ above are distinct and nonzero; that is, they must be congruent to 1, 2, 3, ..., $p$-1 in some order. Multiply all these congruences together and we find

$a \cdot (2a) \cdot (3a) \cdot ... \cdot ((p-1)a) \equiv 1 \cdot 2 \cdot 3 \cdot ... \cdot (p-1) \pmod{p}$

which is, $a^{(p-1)}(p-1)! \equiv (p-1)! \pmod{p}$. Divide both side by $(p-1)!$ to complete the proof.

**Alkalmazás**
Prímteszteléshez lehet használni. Ha egy szám kielégíti, akkor abból még nem következik, hogy prím, de ha nem elégíti ki, akkor az már kizárja. RSA-nál lesz később használva.
https://hu.wikipedia.org/wiki/Fermat-pr%C3%ADmteszt
Ezt használjuk. A lényeg, hogy a-t folyamatosan változgattatjuk, és egy idő után elég biztosak lehetünk benne, hogy p prím (valószínűségi teszt, tehát sosem teljesen biztosak)

## Diffie-Hellman

A lényeg, hogy hogyan tudunk úgy kitalálni egy közös kulcsot, hogy ezt nem küldjük ki igazából egy közös csatornán, vagy legalábbis az azon keresztül menő információk alapján egy harmadik fél nem tudja kitalálni.
És akkor itt az az ötlet, hogy egyirányú függvényeket használunk.
Az egyirányú fv ugye:

- $y = f(x)$ számítása "könnyű"
- x számítása adott $y = f(x)$-hez "nehéz"
- példa: Logaritmus számítás véges testek fölött (modulo algebra)

Számelméleti probléma: Az egyirányúként alkalmazott függvények nem biztos, hogy tényleg azok. Nincs bizonyítva, hogy egyáltalán létezik egyirányú függvény (a "könnyű" és "nehéz" fogalmak ésszerű definiálása mellett). Mégis bátran használjuk őket. Ez igaz a csapóajtó függvényekre is.

A lényeg, hogy van két közös információ:
- n (egy kurvanagy szám kb 1024 bit) és
- g (egy prím, 100 számjegy elég)

És mind Alicenek meg Bobnak van egy privát kulcsa is: X, és Y.

Aztán Alice azt mondja, hogy kiszámolja, hogy

$$p_A = g^X \pmod{\mathrm{n}}$$

Ezt elküldi Bobnak. Majd Bob kiszámlolja full ugyanígy, hogy

$$p_B = g^Y \pmod{\mathrm{n}},$$

Ő is elküldi ezt Alicenek. Ezt követően ezt csinálják külön külön:

$$A : k_{AB} = p_B^X \bmod \mathrm{n}$$
$$B : k_{BA} = p_A^Y \bmod \mathrm{n}$$

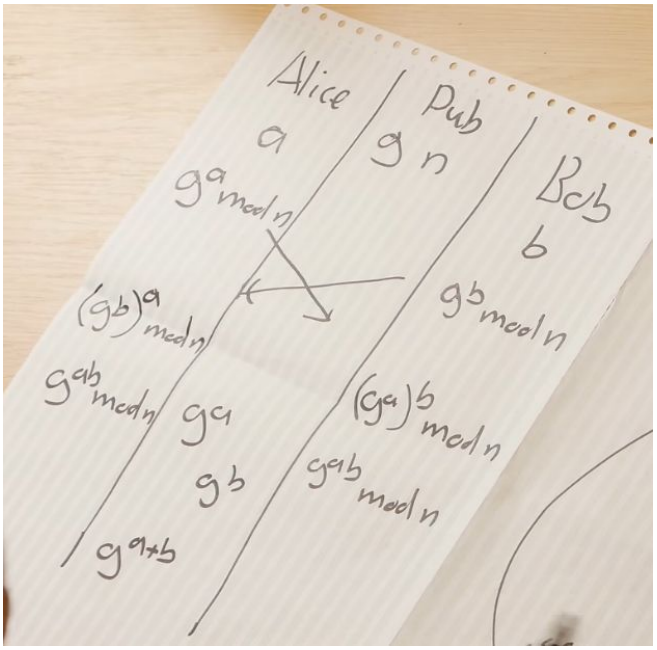
És ez a k ez tökre közös érték lesz:

$$k_{AB} = k_{BA} = g^{XY} \pmod{\mathrm{n}}$$

Ez az egész emiatt az összefüggés miatt működik:

$$[g^Y \pmod{\mathrm{n}}]^X = [g^X \pmod{\mathrm{n}}]^Y = g^{XY} \pmod{\mathrm{n}}$$

Lerajzolni így kell, de a és b nálunk X és Y.




**Diszkrét logaritmus probléma**
Ezt a modulos problémát diszkrét logaritmus problémának nevezzük. A lényeg, hogy azokból az értékekből amiket átküldünk nagyon nehéz megtudni a privátkulcsokat.

# Elliptikus görbék

Szereti a témát a bácsi. A matekos témákat szereti amúgy azt mondja.

**Mese**
Ez is a nyilvános kulcsú eljárások kategóriájában érdekes igazából. A diffie hellmannal ellentétben aláírásra is lehet használni. Titkosítási változat nincsen.
Azért elterjedt, mert sokkal kisebb kulcsméretet használ mint a sima RSA. Ha jól tudom erre a táblázatra áll a rudi, szóval ezt majd elmondom nektek:

| NIST·javaslata·az·egyes·kriptorendszerek· kulcshosszának·összehasonlítására:¤ | | | |
|---|---|---|---|
| ECC· modulus¤ | AES¤ | RSA↵ modulus¤ | RSA:ECC¤ |
| 112¤ | 56¤ | 512¤ | 5:1¤ |
| 161¤ | 80¤ | 1024¤ | 6:1¤ |
| 256¤ | 128¤ | 3072¤ | 12:1¤ |
| 384¤ | 192¤ | 7680¤ | 20:1¤ |
| 512¤ | 256¤ | 15630¤ | 30:1¤ |

A lényeg, hogy ez azt mutatja, hogy hogyan aránylanak egymáshoz a különböző technikák kulcsméretei ugyanazt a biztonságot garantálva. Azért szerepelnek olyan értékek, amiket nem használunk. pl az 56 bites AES az már DES.
Látjuk hogy az utóbbi években egyre nagyobb RSA kulcsot kell használnunk. Túl nagy. Túl nagy számokkal kell dolgozni, sokáig tart stb.
ECC: elliptic curve cryptohgraphy. Az ECC szerencséje hogy inkább az AES kimerítő kulcskeresésére hajaz és nem az RSA faktorizációjára.
Van egy határ amikortól már inkább ezt használják. Az új személyiken ilyen az aláírás amúgy.

**Definíció**
Mi az egyszerűbbet használjuk, mert az is bőven elég lesz.
Itt x,y valós, a,b egész.

Elliptikus görbe: $y^2 = f(x)$ alakú egyenlettel definiált síkbeli görbe, ahol
- f(x) egy $x^3+ax+b$ alakú kifejezés
- x, y valós változók (egyelőre)
- a, b egész számok (egyelőre)
- a görbe nemszinguláris

Általános képlet: $y^2+a_1xy+a_2y=x^3+a_3x^2+a_4x+a_5$
Izomorfizmus erejéig visszavezethető az előbbire.

Ez pedig a diszkrimináns:

$$: D = 4*a^3 + 27*b^2$$

Ha ez 0, akkor a görbe szinguláris, azok nekünk nem jók.

A kriptográfiában nem valós elliptikus szép folytonos görbéket használunk, hanem nyilván diszkrétet (azt mondta el szokták baszni, hallgatók, szla jegyezzük meg)

Van ugye egy ideális pont, ami a végtelenben van az y tengelyen. Ezt már elmondtam. Azt is a görbe részének tekintjük. Ez egy algebrai absztrakció.

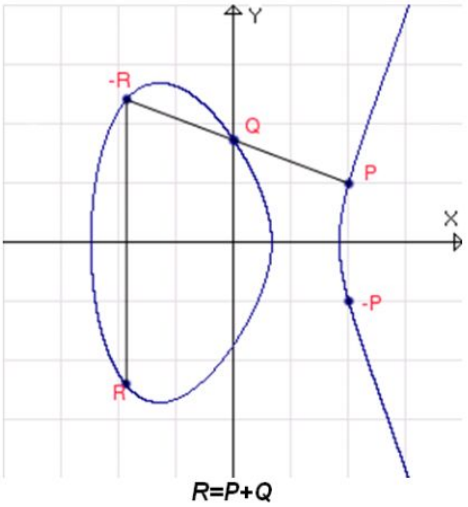
**Összeadás**

Az ideális pontról feltételezzük, hogy rajta van minden függőleges (az y-tengellyel párhuzamos) egyenesen és hogy az x-tengelyre vonatkozó tükörképe önmaga.

A görbe P és Q pontjainak P + Q összege:

a P, Q pontokon átmenő egyenes és a görbe harmadik metszéspontja -R; ennek az x-tengelyre való R tükörképe (ami szintén pontja a görbének) a P + Q összeg.

R + -R =?
az ideális pont (O)




Ez az egész **csoportot** fog alkotni. Van egység, inverz és asszociatív és zárt.
https://hu.wikipedia.org/wiki/Csoport_(matematika)
Ez a sok lózung mind ezért van, hogy csoport legyen. Mint az igazi összeadás. Az asszociativitás azért fontos, mert a szorzatokat (igazából hatvány, de sokszori összeadás na érted) sokkal gyorsabb kiszámolni.

Vannak algebrai formulák a számolásokra, de még senkitől sem kérdezte.

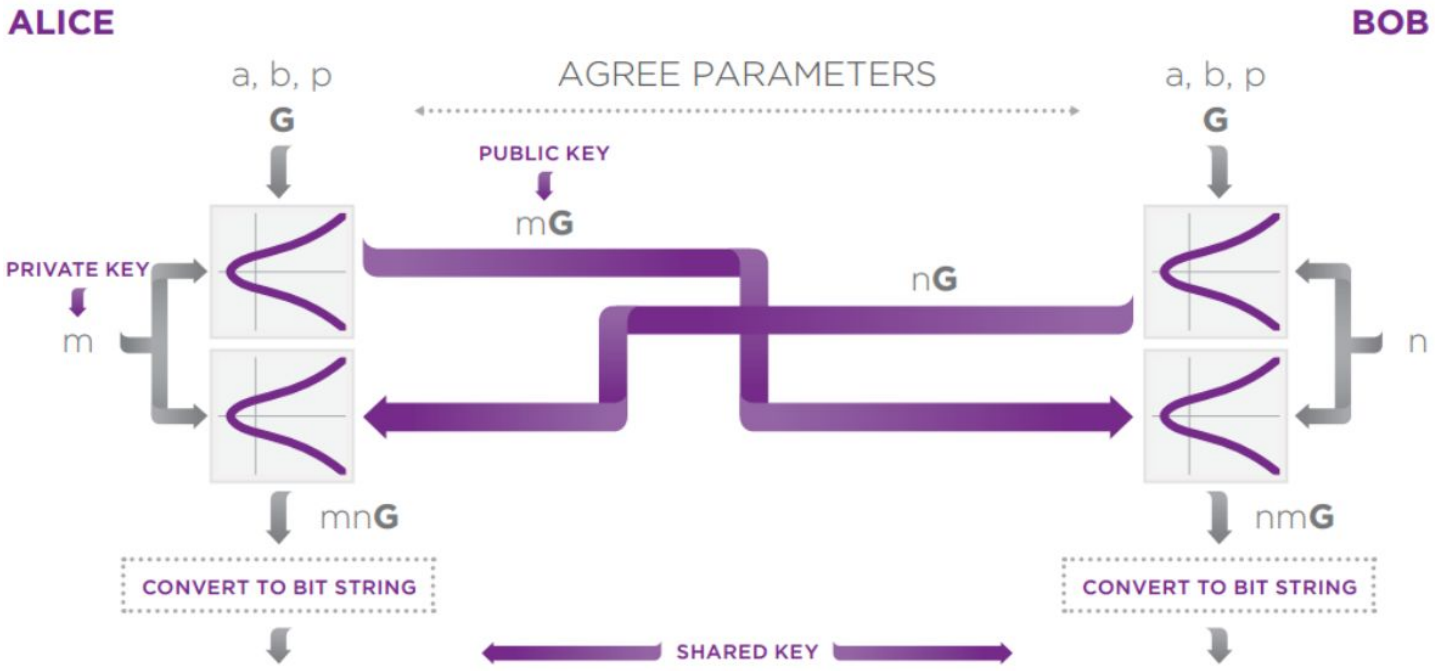
**mod p**
Ez csak azért fontos, mert egész számokat fogunk használni 0 és p-1 között.
De minek? Hát mert számítógépet használunk, szal nem tudunk valósokkal dolgozni.
De használhatjuk majd az algebrai formulákat.

Egész számos görbét úgy ábrázolsz, hogy felírod az egész számokat 1-től n-ig, aztán ezeket behelyettesítve x-be megnézed, hogy melyikre kapsz egész y-t. Ezek a pontok tartoznak a görbéhez, illetve a (0,0).

**Generátor pont**
Ezt önmagával adogatjuk össze és előállítja az elliptikus görbe összes pontját amúgy.

Vannak hagyományos paraméterek, amiket használunk, ezek ilyen szakmai megállapodások.

# Elliptikus Diffie-Hellman




Ugye látszódik a lényeg, hogy a, b, p, G közös paraméterek. Ugye az a, b írja le a curvet a p a modulóhoz kell, a G pedig egy generátor pont. az n meg az m a privát kulcsok. Akkor Alice megcsinálja az mG-t azt elküldi, Bob az nG-t, azt küldi el, aztán ebből mindketten meg tudják csinálni az mnGt. juhúú
Itt a diszkrét logaritmus probléma az az, hogy nagyon nehéz vissza tudni az mG-t. Tehát tudod a G-t meg az mG-t de kurva nehéz kitalálni az m-t.

De az extra, hogy tudunk aláírni is ezzel. A részletek nem fontosak, de az azért elég fontos, hogy legyen egy random szám amit használunk. Playstationnél pl ezt baszták el anno, rá lehet guglizni.

# Szinkron, önszinkron rejtjelezők

**Adatfolyam rejtjelező**
Szimbólumonként (bit/karakter) rejtjelez. Szimbólumról szimbólumra változik a kulcs. A dolgokat úgy jelöljük benne, hogy:

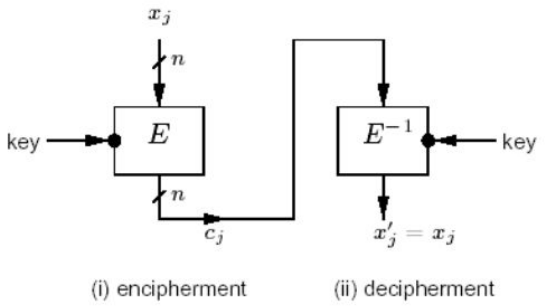
kulcsfolyam: $e_1\, e_2\, e_3 \ldots e_n$
nyílt szöveg: $m_1\, m_2\, m_3 \ldots m_n$ rejtjelezés: $c_i = E_{ei}(m_i)$
rejtjelezett szöveg: $c_1\, c_2\, c_3 \ldots c_n$ dekódolás: $m_i = D_{di}(c_i)$

Magát a transzformációt könnyen végezzük el, de a kulcsot kín legenerálni.
Itt érdemes megemlíteni a Vernam rejtjelezőt, aminél ezek a dolgok binárisak, és a rejtjelezés abból áll, hogy össze XOR-olom a kulcsot a szöveggel.
Egy speciális Vernam rejtjelező a One Time Pad (OTP), aminek a lényege, hogy teljesen random a kulcsfolyam. Ez bizonyíthatóan feltörhetetlen.

Akkor használd, ha kis méretű dolgokat fogsz csinálni és gyorsan akarod hardveresen. ha bitenként akarsz kódolni mert tetszik, vagy ha tudod hogy sok a noise, mert itt nem lesz hibaterjedés. pl. Távközlésben használják.

**Blokk rejtjelező**
t hosszú elemekre bontjuk a szöveget és egyszerre egy ilyen blokkot rejtjelezünk. Lehet használni pl helyettesítéssel vagy felcseréléssel, ezekkel együtt stb. ezek fix méretű blokkok 64,128,256 bit, ilyesmi.

**ECB**

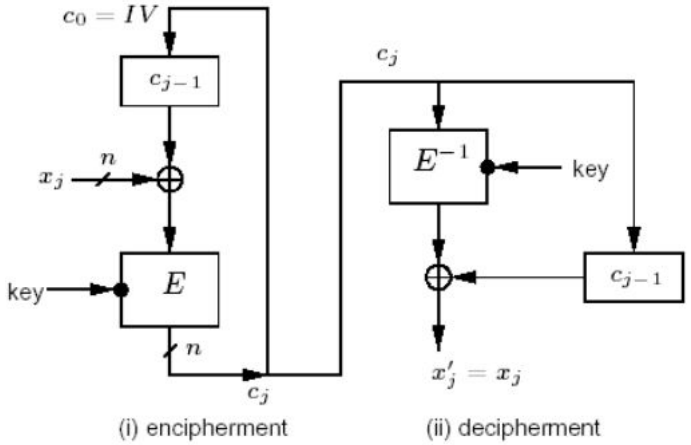



Beszélt valami ECB (Electronic Codebookból) amiből így vissza lehet fejteni a dolgokat.

- Fontos, hogy a blokkok ugye függetlenek, ezért nem lesz hibaterjedés.
- Probléma, hogy azonos nyíltszöveg esetén azonos lesz a rejtjelezett szöveg.
    - Ez ellen úgy védekezünk, hogy csinálunk ilyen paddingot, hogy az üzenetet kisebb darabokra vágjuk mint muszáj és a maradék helyet feltöltjük random szarral, amit dekódolásnál majd ellehet hagyni. De így nem lesz mindig ugyanaz a szöveg.

E:blokkrejtejelő E-1:inverz (ezekre figyelni a többi ábrán), ez azért fontos mert az AES majd ezért tud más tempóban számolni különböző irányokban.

**CBC**

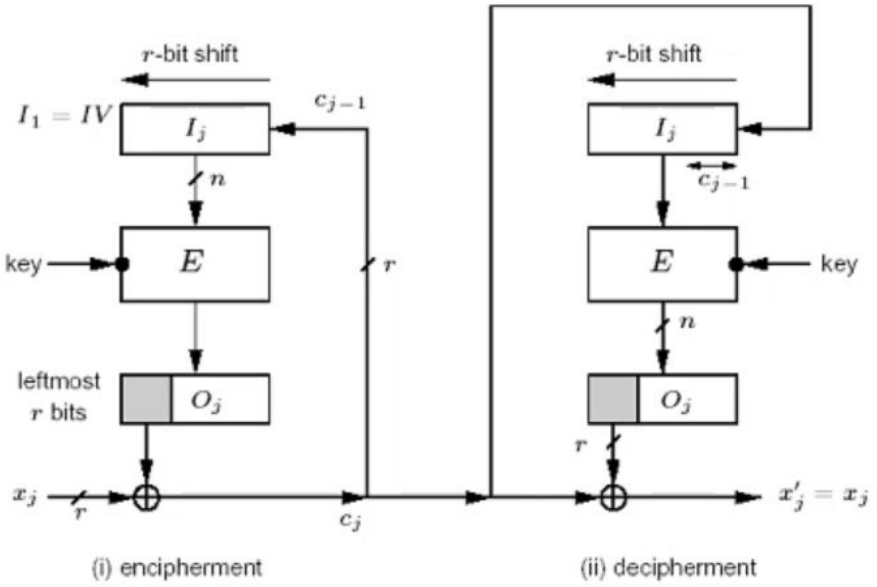



Aztán pedig valami CBC (Cipher block Chaining): a rejtjelező blokkot láncoljuk, hozzáadjuk a kövi blokk tartalmához. (tehát a az előző rejtjelező blokkot felhasználjuk az újban). A regiszter kezdeti értéke az IV (initialization vector - *a szerk*), tipikusan random ugye.

- Itt már nem lesz ugyanaz az üzi.
- Itt van hiba terjedés az előző hat a következőre.

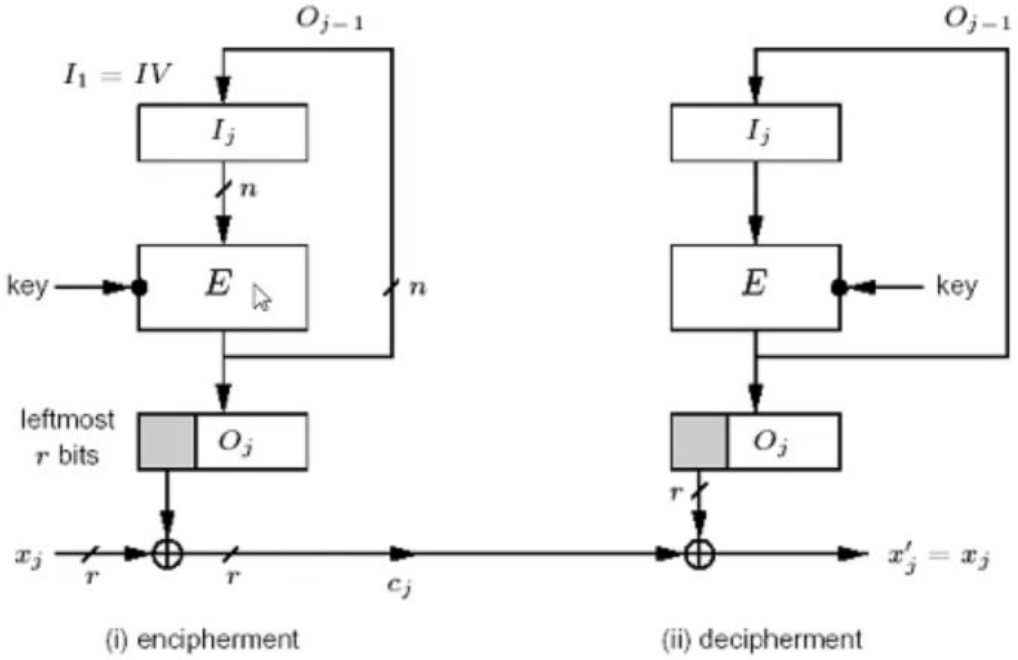
**CFB**

A CFB meg az OFB tökre hasonlít. Ezeket arra használjuk, hogy blokk rejtjelezőkből adatfolyam rejtjelezőt csináljunk. Mivel az adatfolyam rejtjelezők kakik, ezért szeretnénk csinálni blokkrejtjelezőből (ez régen a DES volt, mostanság az AES), és ezekkel csinálunk egy adatfolyam rejtjelezőt.




Itt is az a lényeg, hogy van visszacsatolás, mert így elshifteljük az előző cipher textet és felhasználjuk a kulcsfolyamnál. IV miatt pedig eltérő a rejtjelezett szöveg ugyanazon szövegnél is.
Ha van valami hiba, vagy bitsérülés, akkor hibás adatok töltődnek a regiszterbe, de ez a hiba egy idő után eltűnik, a shiftelés miatt a hibás bitek elmennek egy idő után.

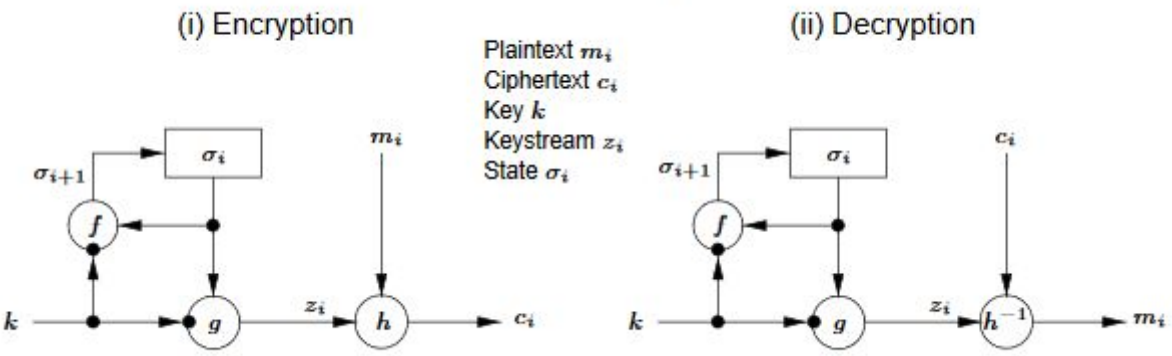
**OFB**




Az a különbség hogy a blokk rejtjelező outputja a visszacsatolás és nem a cipher text. Ha valami bit itt változik, akkor az a dekódoláskor okoz egy változást. Ha azonban bit elvész, akkor elcsúsznak egymástól a szinkronból és rábaszarintottak.

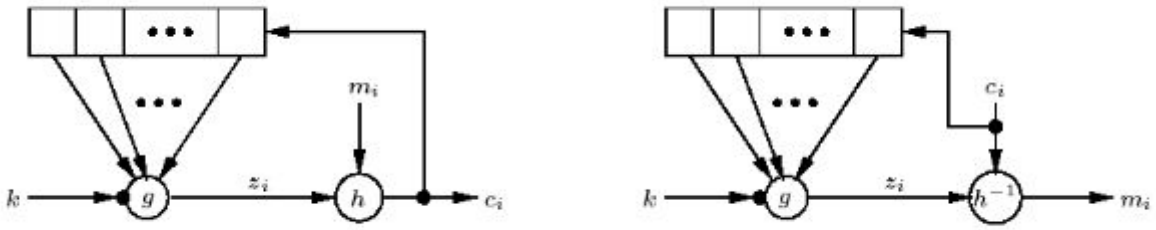
Lehet változtatás, hogy a counter módot bekapcsoljuk, amikor nem a blokkrejtjelező kimenetét csatoljuk vissza hanem az IV-t vlamai számlálóval változtatjuk. Ez is jó, de nem gyakori.

**Szinkron**
Álvéletlen bitsorozat a rejtjelezett szövegtől független. Ilyen pl az OFB (Output Feedback)




- Szinkronizálni kell: reinicializációval, markerekkel vagy offszetekkel
  Ez azért van mert van ez a szép nagy kulcsban ugyanott kell lennünk mint a másiknak, hogy értsük amit pofázik, e ha elcsúszunk valamerre valmaiért akkor full katyvaszság lesz minden.

**Aszinkron (önszinkron)**

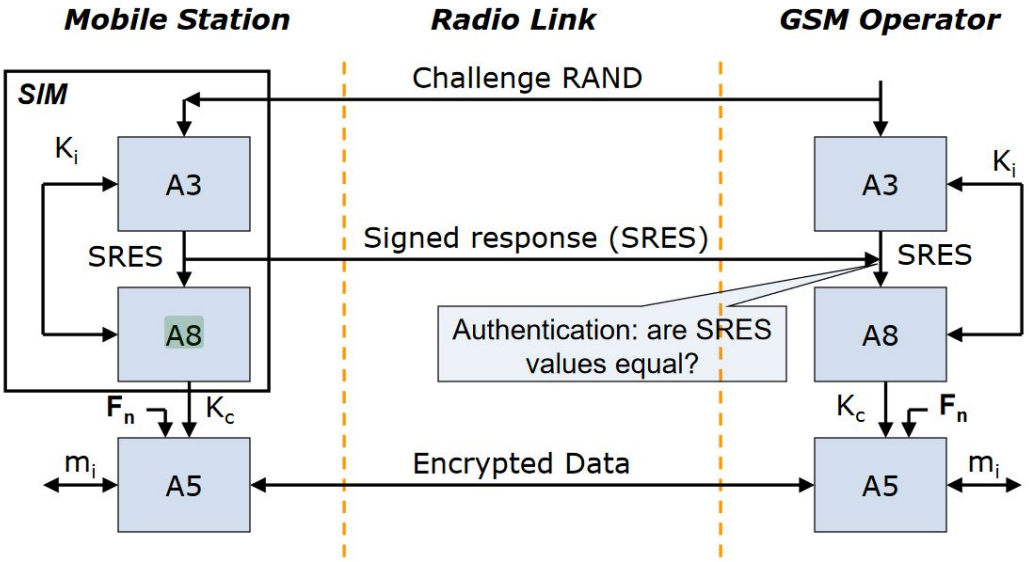



Álvéletlen bitsorozat a rejtjelezett üzenetből. Ilyen pl a CFB.

- Ez szinkronizálja magát
- korlátozott hiba terjedés van benne

**LFSR**
könnyű hardveresen, a lényeg, hogy tolódik benne a szám. pl A5-ben használjuk.

# A3, A5, A8




**A3:** MS autentikáció
**A8:** privát kulcsokat generál (64 bit)
Ezek a SIM-en vannak implementálva

**COMP128:** ezt használjuk mindenütt

The COMP128 algorithms are implementations of the A3 and A8 functions defined in the GSM standard. A3 is used to authenticate the mobile station to the network. A8 is used to generate the session key used by **A5 to encrypt the data transmitted between the mobile station and the BTS.**

**Működés**

- https://www.youtube.com/watch?v=LgZAI3DdUA4
- 3 LFSR, non clock, clock, majority bit
- A5 tud autentikálni és enkriptálni.
- GSM-nél baszatjuk

**Verziók**

- A5/1 erős (USA, nyugat EU)
- A5/2 direkt kaka
- A5/3 3G-hez japán szar

# WEP

- WEP: Wired Equivalent Privacy
- RC4-et használ a kulcsfolyam generálására (ami ugye kakipisi)
- Eredetileg 40 bit kulcs (5 karakter) + 24 bit IV (INITIAL VECTOR) (export korlátozások miatt)
- Arra van ugye, hogy a WIFI-det védi, de van egyébként jelszavas meg nem jelszavas módja is. A jelszavas módja az SKA (Shared Key Authentication)

| | WEP Encryption | No Encryption |
|---|---|---|
| **Open System Authentication** | Encryption<br>No Authentication | No Encryption<br>No Authentication |
| **Shared Key Authentication** | Encryption<br>Authentication | No Encryption<br>Authentication |

**SKA**

Na ez úgy működik, hogy

1. szólunk, hogy kapcsolódni akarunk
2. kapunk egy véletlen challenget (1024 bit) (elvileg hasonló a gsm-hez?)
3. kliens elküldi: IV, WEP([challenge, checksum], [IV, shared key])

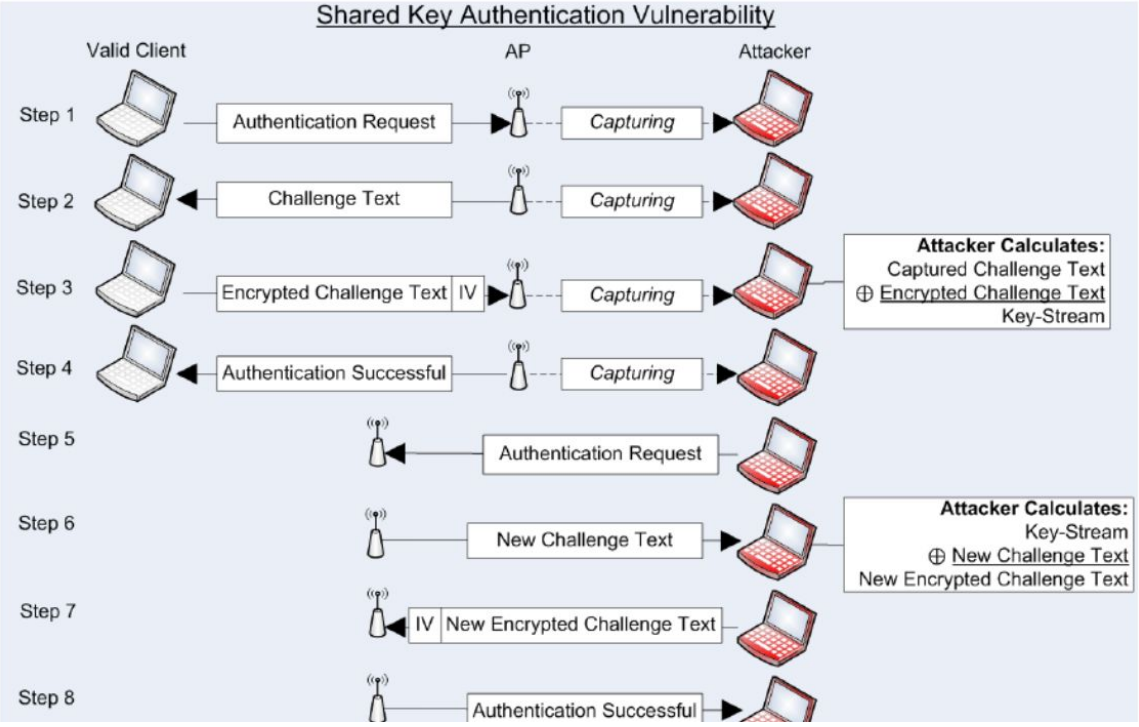
mi generálunk (kliens) egy IV-t, ezt összefűzzük a jelszóval (shared key).
a kapott challenget pedig egy ellenőrző chesummal
ezt megcsinálja a router is, és le tudja ellenőrizni, hogy ugyanaz jött e ki

IV: 24 bites kezdeti érték, mi legeneráljuk
[challenge, checksum] összefűzve a kihívás érték meg a generált checksum
[IV, shared key]: összefűzve a kezdeti érték és a megosztott kulcs

Ez egyébként csak akkor para, ha nem titkosított. Ha titkosítva van, akkor hiába megy fel, nem fogja tudni látni.

**WEP-SKA sérülékenység**

Az a lényeg, hogy kaka sajni.
Azért mert, ha a támadó lehallgatja a Challenge-t meg a WEP([challenge, checksum], [IV, shared key]) üzenetet, akkor abból ki tudja számolni a kulcsfolyamot a Vernam miatt.
Az egész baj azért létezik, mert a kliens találja ki az IV-t és nyíltan elküldi. Ezért ha szépen ezt lehallgatja egy mókus, akkor rábaszás van.
Ez azt jelenti, hogy az autentikáció kaka. (nem kell jelszó)



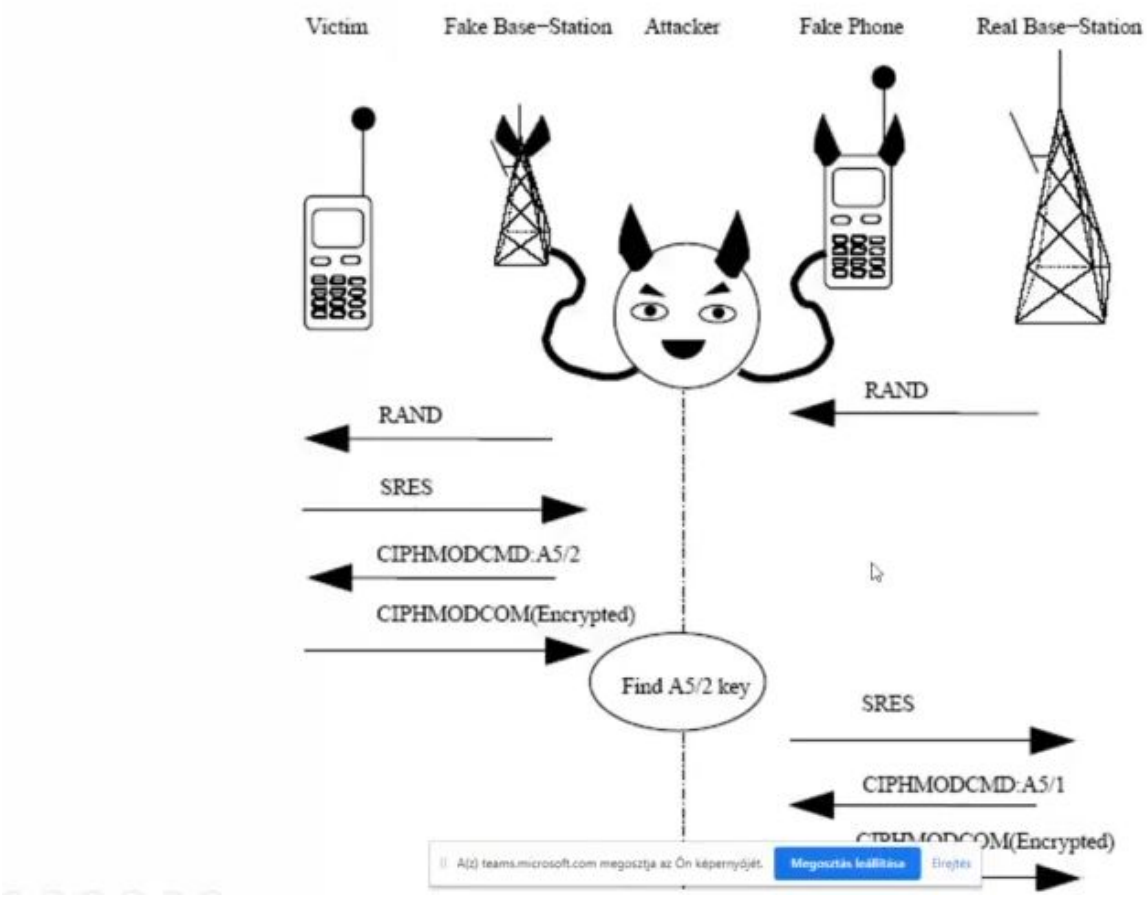
**WEP checksum sérülékenység**
Itt viszont az a baj, hogy a checksum nem védi az üzenet integritását (lehet rajta változtatni). Tudunk néhány bitet átfordítani (anélkül hogy megismernénk őket), de a hitelessége sérül az üzinek.

**WEP születésnapi paradoxon**
24 bit az IV, ami nem OLYAN nagy. ugye ez a szar ez folyamatosan változik, minden falcsatlakozásnál. viszont azt jelenti, hogy pár ezer frameben lesz valszeg két azonos IV.
Azonos IV-nél azonos a kulcsfolyam, ebből már látszik a baj: a két kódolt szöveg különbsége ugyanaz lesz, mint a két eredeti szöveg különbsége.

Pont ezek miatt nem nagyon használjuk már a WEP-et, mert ugye mint mondtam kaka. Ezt turbózták fel jól az lett a WPA (ott is RC4 van, de már jobban véd), abból aztán a WPA2, ami már AES alapú.

# Man in the middle

Ne keverd a meet in the middlel az más. Az a lényeg, hogy a kisköcsög azt hazudja a bázisállomásnak, hogy ő a teló, a telódnak meg hogy ő a bázisállomás. megváltoztatja a paramétereket. Pl. az A5/2-t használ A5/1 helyett gonoszan. ezzel így majd a dekódolást könnyen el tudja végezni.




Egyébként aláírással lehet védekezni.

# DES, Meet in the middle

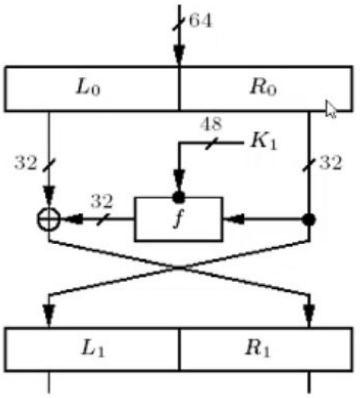
**DES** = Data Encryption Standard

**Feistel rejtjelező**

Szétvágjuk a blokkot egy bal és egy jobb oldali részre. Majd a jobb oldali blokkból valami módon előállítunk egy értéket, amit hozzáadunk a baloldalihoz. Majd a két oldalt felcseréljük.

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$




Ez attól érdekes, hogy nem fontos a dekódoláshoz, hogy az f invertálható legyen. Úgy tudjuk visszafejteni, hogy visszafele haladunk. bal oldalt visszatesszük jobbra, előállítjuk belőle a szart, és aztán innen már logikus.

A DES ezt fogja végrehajtani sokszor.

**DES működése:**

1. Bemenetek: (blokk méret) 64 bites plain text és 64 bites kulcs (ami valójában 56 bit + 8 paritás bit)
2. 16 iteratív lépés
3. Kimenet: 64 bites cipher

**Típusai:**

- sima DES : nem biztonságos, túl kicsi a kulcs -> brute forceolható

Hogyan lehetne javítani? hát használjuk többször! ennek akár értelme is lehet, mert két des művelet ne mhelyettesíthető 1-gyel.

- double DES: nincs értelme (bácsi mondta nem én: azért mondta, mert alig magasabb a számítási igénye: 2^56 2^55 helyett. ennek a feltörése a meet in the middle)
- triple DES: 2-3 kulcsot használ , sajnos lassú, de néha még használjuk (ahol nincs meg az AES-hez a technikai feltételek)

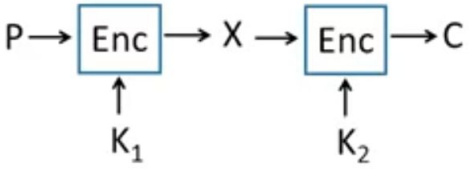
Double és triple DES-nél lehet Meet in the middle támadás

**Meet in the middle**

https://www.youtube.com/watch?v=FDgx055JA_Y&t=280s

Ez nem a man in the middle. Itt a lényeg, hogy a nyílt üzenet és a rejtezett üzenet között találkozik az a köztes érték, ami a két kulcsal történő des titkosítás során keletkezik.

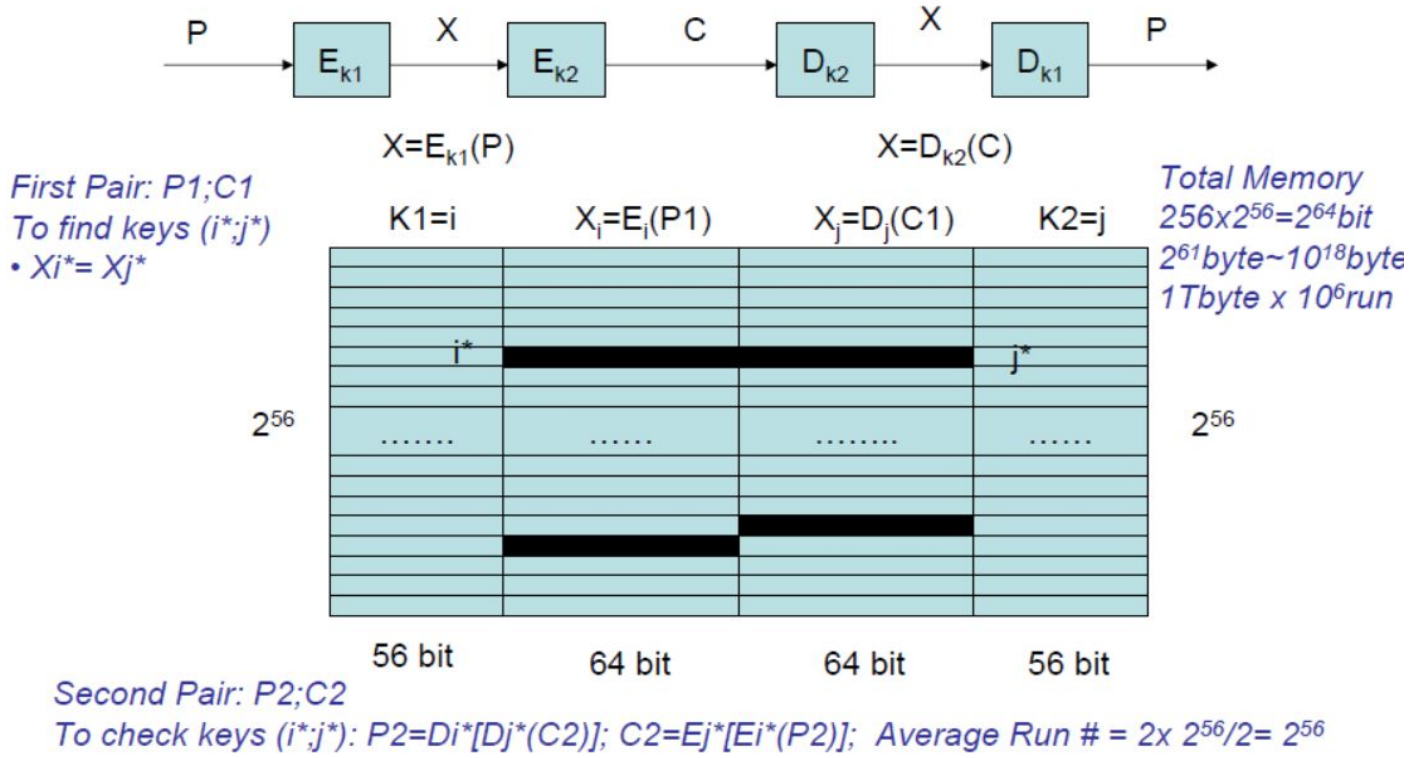

Az az egész lényege, hogy a támadó kihasználja, hogy lesz egy középső érték a két tikosítás közt.




Azt feltételezzük hogy két nyílt üzenet párt is ismerünk. (P és C a második ábrán). Készítünk egy nagy táblázatot (néhány terabyte amúgy). A lehetséges kulcsokat belírogatjuk mindet.Aztán mindegyikre legyártjuk a Cből a dekódoltat, meg P-ből a kódoltat. majd elmondom am szóban.

És ahol megegyezik az X ott lehet hogy jó nekünk az élet. Több ilyen is lehet ugye.

VISZONT, ha már van MÉGEGY nyílt üzenet párunk, és megint találtunk ugyanott egy találkozást, akkor már jó. A számítási igény már nem sok a másodiknál.




Triple DES-nél is van, de ott már nagyobb a számítási igény azért.

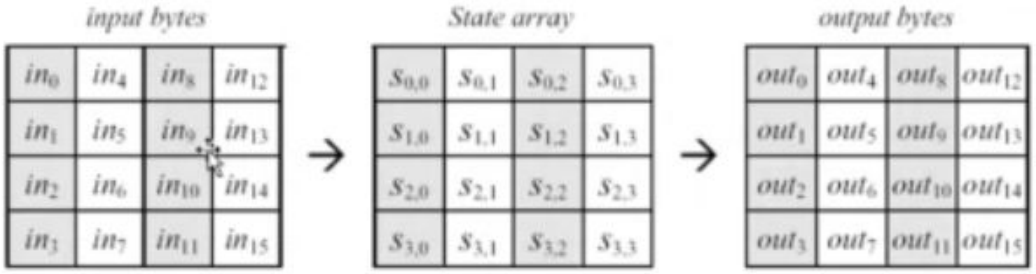
Double DES tehát O(2^56) komplexitású, a triple DES viszont már O(2^112), a sima DES csak O(2^55)

**Szumm**
Összességében jó konstrukció, mert nem lehet megjósolni, hogy a rejtjel szöveg változása milyen hatással lesz az egészre. Viszont azzal, hogy a kulcsmérete pici, elalvulttá vált.
Elavult és szar, ehelyett lett az AES az új standard, amiben nagyobb a kulcs és kurva gyors és marha ügyes:))))))

# AES

edvenszd enkripsön sztenderd

- DES utódja: 56 bit kevés volt, ezért itt 128 bitet használunk. A kulcsmérete változó a rétegszámtól függően (128,192,256)
- Fontos, hogy a dekódolás meg a kódolás nem ugyanúgy zajlik: az egyik inverz. Éppen ezért az AES sokkal gyorsabban kódol mind dekódol.




- 128 bites blokk rejtjelező (4x4-es táblázat/mátrix ahogy érzed; 1 oszlop 4 bájt)
- ez fontos: nem bitekkel dolgozik, hanem bájtokkal

**Bináris polinomok**

- a bájtokat bináris polinomként kezeli (bináris polinom: olyan polinom ahol az együtthatók egyesek vagy nullák)
- A bináris polinomok együtthatóit bináris számokként adjuk  össze. példák:

$$x^2+x+1 \;+\; x+1 \;=\; x^2+0+0 = x^2$$

$$(x^2+x+1)(x+1) \Rightarrow x^3+1$$

$$x^3+x^2+x \;+\; x^2+x+1 \;=\; x^3+0+0+1$$

**Irreducibilis polinomok**

Azt jelenti, hogy a polinomot nem lehet felbontani más polinok szorzatára. Ön maga osztja csak meg 1. pl. az x^2 +x +1 az irrdeucibilis.

Általános szabályok, ha ezt vizsgálod:

- ha nincs benne 1-es csak x-ek akkor tuti osztható x-el, szal azokat egyből kizárjuk, fel sem írod őket.
- megvizsgálod, hogy hogyan lehetne osztható. pl harmadfokú egy másodfokú meg egy első fokú szorzata
- Az az általános módszer, hogy fel sem írod ami x-szel vagy x+1-gyel osztható.
    - Azt onnan látod, hogy x+1-gyel osztható, hogyha páros számíú tagja van.
- ha jól értem amúgy gecire nincs más szabály szóval ennyi, utána próbálod kitalálni csuklóból.

**Multiplikatív inverz**

Adott egy m(x) polinom, meg egy másik aminek kisebb a foka és annak keressük az inverzét. Ez azt jelenti, h: Olyan polinomot keresünk, amivel megszorozva az eredetit, 1-et kapunk moduló m(x).

$$x^3+x^2+1 = m(x)$$

$$(x+1)^{-1} = ? \mod m(x)$$

$$(x^2)(x+1) \equiv 1 \mod m(x)$$

$$(x^3+x^2)-(x^3+x^2+1) = 1$$

Aztán itt most abba nem megyek bele, de euklideszi algoritmussal lehet itt baszakodni még, ez azért kell amiért az RSA-ban is használunk inverzet, és azt használjuk majd a sok invertálásnál.

Ezeket a polinokoat binárisan úgy ábrázoljuk, hogy pl az x^7+x+1 az 1000011 stb.

AES-ben a modulus:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

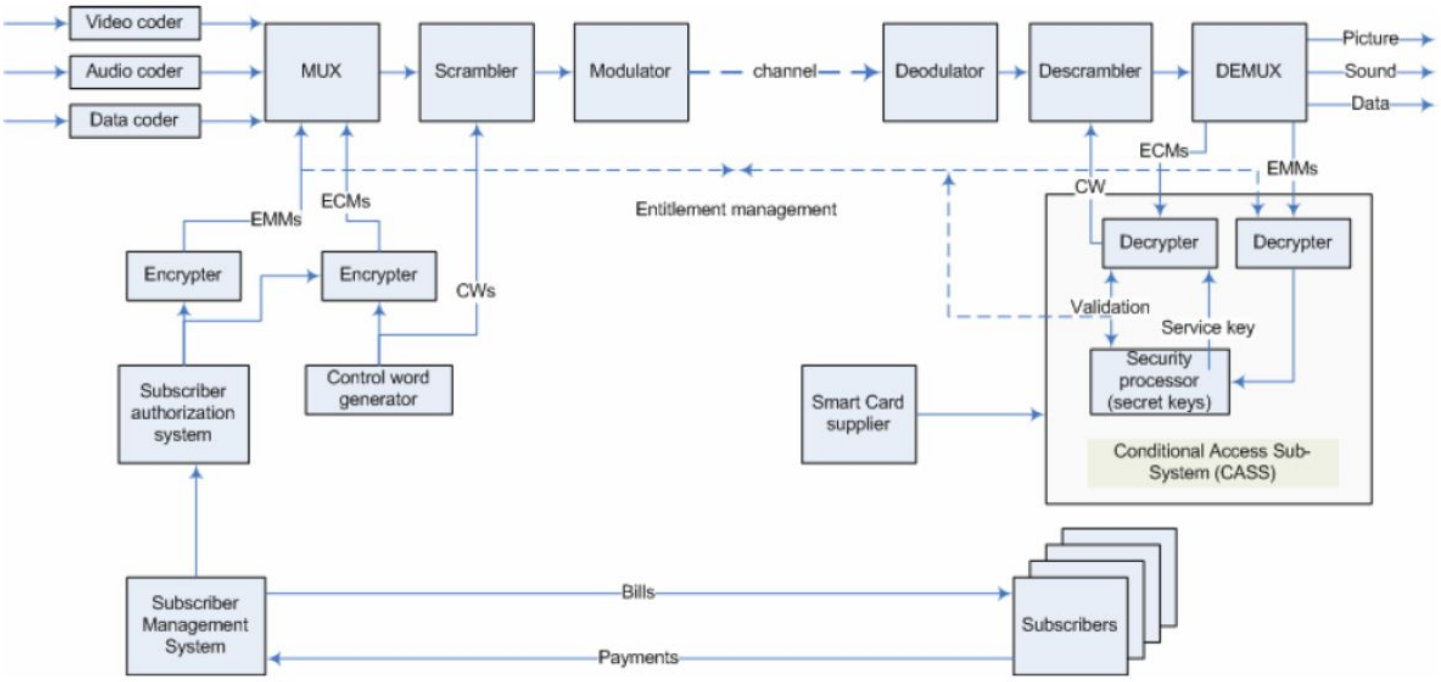
**Lényeg**

- két féle művelet: polinom vagy oszlop
- egy oszlop = 4 bájtból álló speciális bináris polinom, aminek az együtthatói is bináris polinomok (erre lehet az oszlop keverést használni)
- fog egy 128 bites üzit és a speckó műveleteivel titkosítja
- 4 művelete: (https://youtu.be/lnKPoWZnNNM?t=518 )
    - helyettesítés (invertálást használ, majd lineáris leképezés, és hozzáad egy vektort, ne kérdezd mit)
        - Abból a táblából kikeresed, hogy mi az érték. (videóban látszik)
    - sor eltolás
        - Itt kb az a lényeg, hogy a sorokat tolod így overflowol mint egy shifttregiszter (videóban jól látszik szintén)
    - oszlop keverés
        - ez elég egyértelmű-nek tűnik, de valójában nem az aminek gondolod. valami mátrixszal beszorozza az oszlopot (videóban látszik)
    - kulcs hozzáadás
        - hát itt van egy round key és az oszlopokat elemenként XOR-oljuk (ez is a videón látszik)
- iterációs rejtjelező = egy belső invertálható funkció ismétlése (ezeknek a fenti szaroknak a valamilyen ismétlése)
- van segédtábla az intvertáláshoz, mert hosszú lenne minden elemhez kiszámolni
- Próbáéják feltörni, egyelőre jó ,szeretjük.

# DVB

DVB: Digital Video Broadcasting
Lényeg: fizetős csatornák anyaga titkosítva megy, hogy csak az előfizetők lássák a műsort.
Van egy TV-hez csatlakoztatott szar (conditional acces subsytem), ez oldja meg ezt.
Az az izgalmas probléma, hogy nem tudunk hitelesíteni: itt nem tudunk jogosultságot ellenőrizni, hanem a hozzáférési kulcsot probálom úgy terjeszteni, hogy csak az kapja meg, aki jogosult. De honnan tudjam, hogy csak az kapja meg?
Megoldás: egy kártya tartalmazza ezt a kulcsot. Ez a kód sűrűn változhat amúgy (pl nem fizette be) Erre ilyen vezérlő üzeneteket használ.
Ez a megoldás meg ez a mechanizmus a DVB-CA (Conditional Access)

**Card sharing támadás**
A lényeg, hogy egy jogosult felhasználó ugye tudja dekódolni a CW (Control Word) kulcsot, és ezt megoszthatja nem jogosult felhasználókkal. Ez a mai napig létezik amúgy. Nem beszélt róla egyébként sokat.




DVB - Common Interface, az a lényeg, hogy az ellenőrzéses szarok egy részét leszedik a TV-ről, hogy pl könnyebb legyen a dolog szolgáltatás váltásnál.

**Biztonsági problémák**
Jobb mint az analóg időkben, mikor konkrétan elég volt egy kalóz kártya. Azért a kulcsok folyamato cseréje és a különböző felhasználói kulcsok egész valid védelmet adnak.

A baj, hogy: mindenki megkapja az adást, nincs hitelesítés ezért fogalmunk sincs hányan használják ugyanazt a kulcsot, a set-top-boxra pl akár lehet telepíteni bazmeg programokat is.

**Lehetséges megoldás**
**CW kódolása**: EMM dekódolása és a K0 tárolása a smart cardon van, de a CW elhagyja a kártyát, mert a dekódolás maga nem a kártyán történik. Ilyenkor el lehet csípni a CW-t. Hát ezt akadályozza meg ha a kártya és CAM (ami dekódolja az adást) között használunk egy extra titkosítást
**DVB CSA3**: nem tudjuk hogy működik, mert nem public.
**CI+**: Másolásvédelemre is jó. UPC pl ilyet ad. Ha a TV támogatja a szabványt a CAM és a TV hitelesítik egymást, és titkosított közöttük az adatáramlás.

# RSA

Az a lényeg, hogy ez egy olyan eljárás, ahol nem közös kulcsot használunk (mint pl Diffie-nél), hanem külön kulcs van a dekriptálásra, meg a titkosításra. Mindenkinek lesz egy publikus meg egy privát kulcsa, és a publikussal kell majd titkosítani, de azt csak a priváttal lehet kinyitni.

3 komponens: kulcsgenerálás, kódolás, dekódolás.

**KulcsgenerálaS**
Kell találnunk két nagy prímet: p, q (nem ugyanazok). Ebből kiszámoljuk, azt, hogy $n = pq$ és $\phi(n) = (p-1)(q-1)$. Aztán szedünk egy e-t is, úgy, hogy $1 < e < \phi(n)$ és $\phi(n)$-nel relatív prím. Aztán d pedig legyen $1 < d < \phi(n)$ úgy, hogy $ed = 1 \ (mod\ \phi(n))$ . Ekkor a nyilvános kulcs lesz az $(e, n)$ a titkos kulcs pedig a d.
Ezt jó gyorsan ki lehet számolni (a d számolásához a kiterjesztett euklideszi algoritmust használjuk).

**Kódolás**
Küldő megszerzi a vevő nyilvános kulcsát (e, n). Aztán kódoljuk az üzentet, úgy hogy egy n-nél kisebb x számmá konvertáljuk. Ha túl nagy az üzenet, akkor blokkokra bontjuk az üzenetet, és a blokkokat külön konvertáljuk egésszé. Aztán ebből kiszámuljuk, hogy $y = x^e \ mod\ n$. Ez lesz a titkosított üzenet.

**Dekódolás**
Fogja a vevő a privátkulcsát: d és az y-nal azt varázsolja, hogy $x = y^d mod\ n$, és vissza is kapta az üzenetet

A kódolást és a dekódolást is gyorsan végre tudju hajtan az **ismételt négyzetre emelés** műveletével. Ezzel az algoritmussal gyorsan kiszámíthatjuk az e-dik hatványt. Ilyesmódon:

$342^2 \equiv 285$ $\qquad$ $342 \cdot 342^2 \equiv 342 \cdot 285 \equiv 1261$
$342^4 \equiv 285^2 \equiv 1392$
$342^8 \equiv 1392^2 \equiv 1202$
$342^{16} \equiv 1202^2 \equiv 1669$ $\qquad$ $1261 \cdot 342^{16} \equiv 1261 \cdot 1669 \equiv 293$
$342^{32} \equiv 1669^2 \equiv 1641$ $\qquad$ $293 \cdot 342^{32} \equiv 293 \cdot 1641 \equiv 1815$
$342^{64} \equiv 1641^2 \equiv 1076$
$342^{128} \equiv 1076^2 \equiv 1221$ $\qquad$ $1815 \cdot 342^{128} \equiv 1815 \cdot 1221 \equiv 1261$

(A kongruenciák modulusa mindenütt 2047.) Végeredményül kapjuk tehát, hogy $342^{179} \equiv 1261 \pmod{2047}$.

forrás: http://www.math.u-szeged.hu/~kaman/titkosiras/doc/RSA.pdf

**Miért jó?**
Ezért:

$$(x^e)^d \equiv x \pmod{n}$$

**Biztonság**
Na az RSAa pont annyira nehéz mint egész számokat prímtényezőkre bontanti. Tehát a d privátkulcs kiszámítása a nyilvános (e,n) kulcsból ekvivalens az n prímtényezős felbontásával. Ez pedig baszott nehéz. Itt van egy bizonyítás amit nem értek.
Az RSA feltörése nem feltétlenül a d megfejtése, hanem az $y = x^e \ mod\ n$ -ből az (e,n) ismeretével az x meghatározása. Ez az RSA probléma, és igazából nem tudjuk, hogy ekvivalens e a prímfaktorizációból. Viszont azt gondoljuk, hogy nehéz megcsinálni.
Konkrét megvalósításnál sok múlik a biztonságon.

# SSO

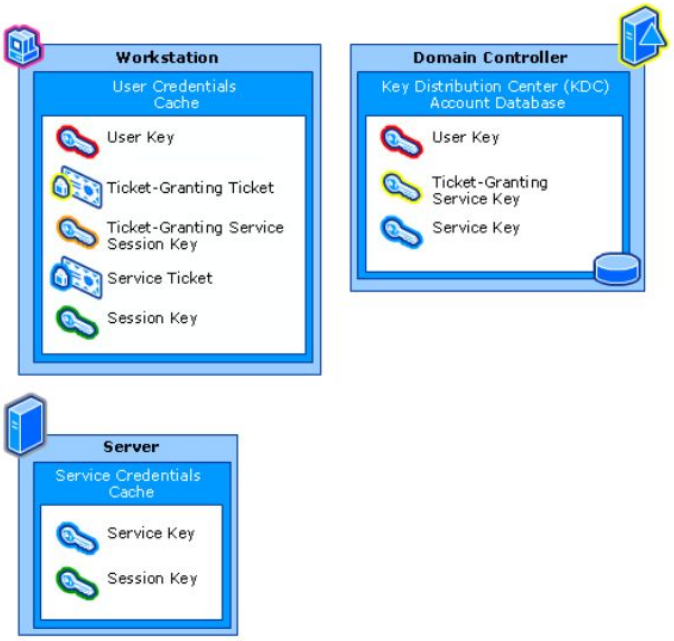
**SSO:** Single Sign On : a lényeg, hogy egyszer bejelentkezel aztán mindent tudsz hasnálni, nem kell folyton újra azonostani magad. Ilyen pl a Kerberos, vagy a weben ugye bármi

**Kerberos**

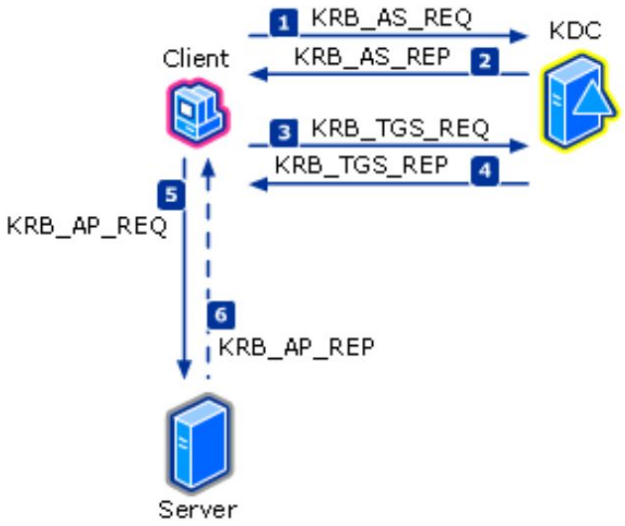
Ez a Windows bejelentkezéshez van használva.
**Fogalmak**
**realm/domain**: tartomány, melyben a kliensek egyetlen autentikálós szerverhez csatlakoznak
principal: felhasználó/ szolgáltatás
**ticket**: jegy, amit a kliens használ majd az autentikációhoz. van benne pl név, IP, érvényességi időtartam, session kulcs
**session key**: na ez van a ticketben, a ticket kibocsátója hozza létre, ennek ismerete az autentikáció alapja.
**authenticator:** hitelesítő adat
**FONTOS: Key Distribution Center:** tartomány autentikációs szolgáltatása**.** ennek részei:

- database: ebben tároljuk az ügyfeleket
- autentikációs szerver: ő hajtja végre az ellenőrzést. ő egy olyan ticketet csinál, amivel a ticket granting tickethez lehet majd hozzáférni.
- ticket granting server: a domainben elérhető szolgaáltatásokhoz csinálja meg a ticketeket.

**Szereplők**: (van egy megosztott szimmetrikus kulcs ugye, amivel nyilvántartanak minket)




**Hozzáférési folyamat**



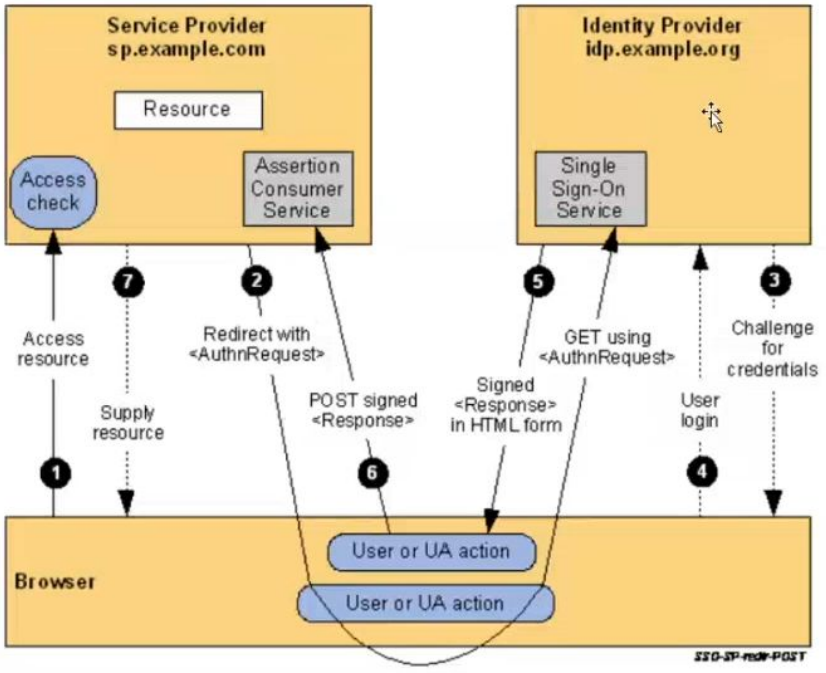

Először autentikáció aztán a magic ticket, atán azzal kér hozzáférést a szolgáltatáshoz, aztán ha megkapja akkor azt küldi a szervernek.

Ez le van írva, hogy ezek pontosan miből vannak a diában de csak nem kell…

**SAML**

Security Assertion Markup Language

A lényeg, hogy akarsz szolgáltatást, átküld hogy identityt provideolj, aztán ő visszaküld ha jó vagy. Nem kell ismerni ANNYIRA a részleteit. Aláírást, titkosítást használgatunk benne. Itt fontos ez a böngésző átirányításos dolog. Egy XML struktúrában megy, hogy miket várnak el. Azt is elmondhatjuk, hogy mennyire szigorú legyen, pl akar e OTP-t meg ilyesmi. Autentikáción is aláírás van.



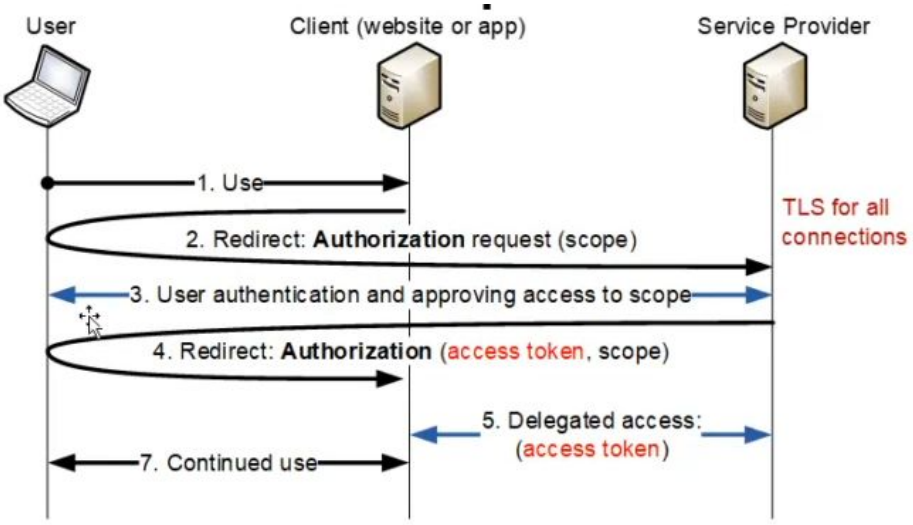

**OAuth 2.0**

Open Authorization

Majd az OpenID-nál lesz használva, ami azért fontos, mert a személyinél van használva.

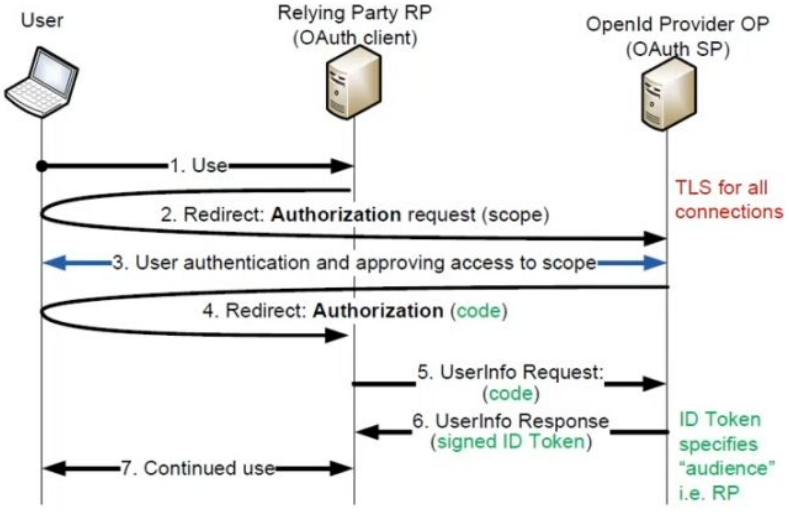
A Nagy cégek együtt találták ki. Arra van, hogy hozzáférési jegyeket biztosítson az ügyfél szarjai alapján különböző szolgáltatásokhoz. 
Authentication vs Authorazitation: bejelnetkeztetés vs mire van felhatalamaztvva

Pl arra van, hogy vásárolok egy kabátot és ők megosztanak valamit a fb-mben mert megengedtem




Tehát ő az én nevemben csinálhat valamit.
És akkor ehhez csináltak egy SSO-t.

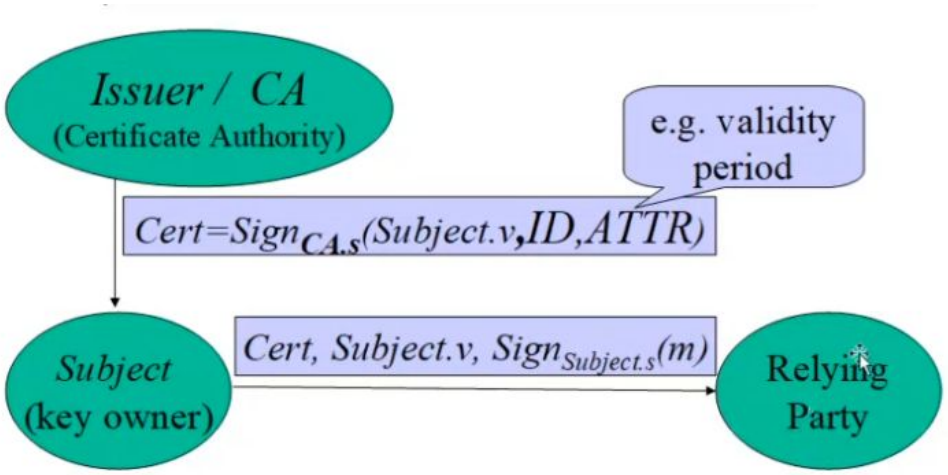
**OpenID Connect**
ugyanaz kb mint az SAML, de flexibilisebb az egész, inkább üzlethez nem pedig közigazgatáshoz.




annyit csinál ugye, hogy olyan ticketet kap a szolgáltató az átirányításkor, amellyel lekérdezheti az ügyfélnek a máshol regisztrált adatait. ez egy ilyen jsonban mászkál.

# PKI

Nyilvánoskulcsú eljárásnál legnagyobb probléma az az, hogy a publikus kulcs tényleg azé akié gondolom. Erről lehet csacsogni picit. Hogy ezzel mi a baj a lényeg, hogy szükség lesz ilyen tanúsítványokra, amikkel igazoljuk hogy kik vagyunk.
https://www.youtube.com/watch?v=i-rtxrEz_E8&list=PLmkD7YZEG8ykhT9s7PoCwG6naWe-Ir2o5&index=9
PKI: Public Key Infrastructure




Relying Party: elfogadnak tanúsítványt felhasználó/böngésző
Subject: magánkulcs birtokosai pl szerver
Kibocsátó: hitelességi szolgáltató, ő biztosítja a megfelelő tanúsítványokat

RP akarja a tanúsítványt, lát egy üzenetet, amit aláír a subject, és passzol mellé egy tanúsítványt. A tanúsítványban meg ugye a subject adatai, meg érvényességi idő van. A hitelesítő szolgálat aláírása benne van.

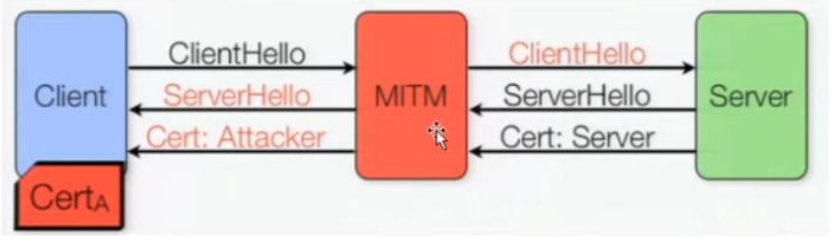
Ezek a hitelesítési szolgáltatók hierarchikus rendszerben vannak. Root CA, aki saját magának ad tanusítványt, és aztán ő ad az alatta lévőnek, meg ő az alatta lévőnek.

Egy böngésző összeveti a tanúsítványokat. Az URL is belekerül a tanúsítványba.
A tanúsítványok lejárnak kb 2 évente. Kulcsok is elavulnak időről időre (pl 1024 bites RSA már szar lesz kb 2 év múlva). pl atz is baj lehet, hogy a tanusítvány nem csak a személyazonosságom, hanem a munkahelyem adatait is tartalmaz, ha felmondok, akkor ezt revokeolni kell nyilván, mert már nem dolgozok ott, ne beszélhesse ki cég nevében.

A hitelesítési szolgáltatók tudnak oylat, hogy
- listában tartják azokat amiket visszavontak
- biztosítanak egy lekérdezési felületet, ahol le lehet kérdezni, hogy érvényes e egy tanúsítvány

Vannak short term certek is.

DigiNotar: valamelyik alkalmazott köcsög volt és a DigiNotar szolgáltató, ami benne volt midnenütt mint hitelesítési szolgáltató. és kiadott speckó tanúítványokat, anélkül hogy az kérve lett volna. A lényeg, hogy Man In The Middle-t csináltak az iraki/iráni szarfaszúak mindenhez.
Ez volt a TLS Interception



# SSL, TLS

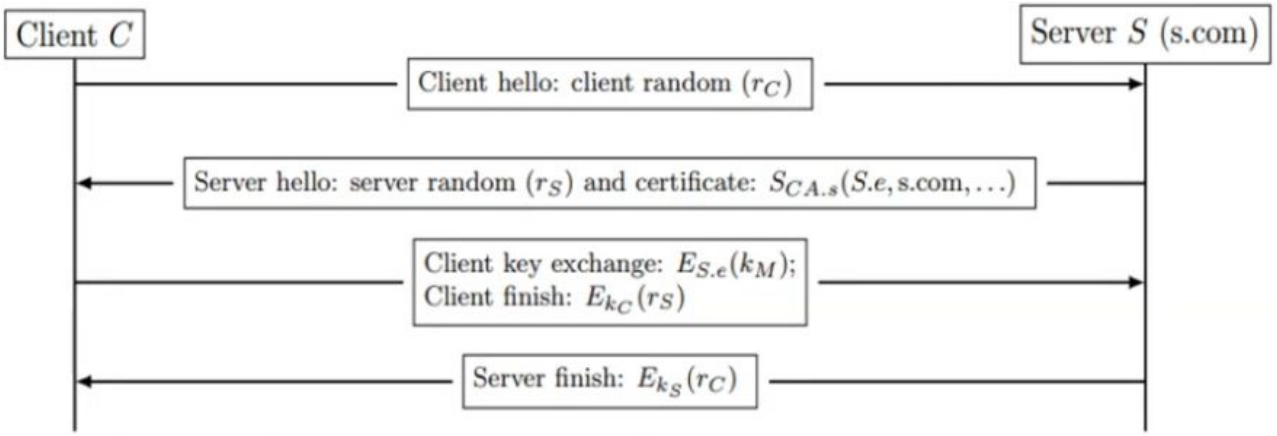
PKI egyik elterjedt használata az SSL/TLS kapcsolat.

Több verzió: fontos, a visszafele kompatibilitás (itt is azt választjuk, amelyik a maxból a kisebb)
SSL, TLS 1.0

- szar mert RC4

SSL és TLS különböző algoritmusokat támogatnak.

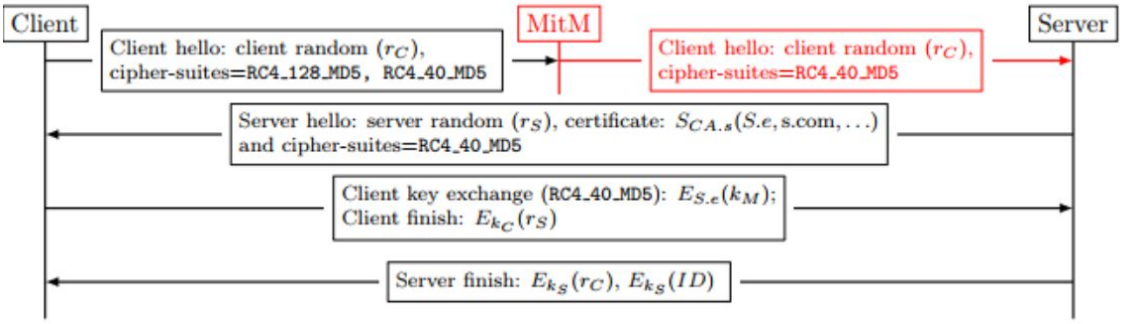
**SSL TLS handshake**
https://www.youtube.com/watch?v=sEkw8ZcxtFk&list=PLmkD7YZEG8ykhT9s7PoCwG6naWe-Ir2o5&index=7




1. Kliens küld egy helót, ebben infó van pl arról, hogy milyen verziót akar hasnzálni, milyen kriptografiai eljárások, meg valami randomot stb.
2. Szerver küld egy helót, illetve a saját certificatejét, meg a publikus kulcsát, és egy saját szerver randomot (ezek a randomok a kulcs genhezlesznek használni)
3. Itt a kliens igazából megkérdezi a certificate authorityt, hogy valid e a cert ->ezután alakul ki a bizalom, ezután kulcsot cserélhetünk
4. A kliens elküldi a titkos kulcsot, amit használni fognak a kommunikációban a szerver publikus kulcsával titkosítva
5. Aztán a klien küld egy finisht, amit már ezzel a hatred secrettel titkosít.
6. EZtán a server váalszol, úgy hogy ő is encriptálja a finish mesaget ezzel.

Aztán sessiönként már ezt nem kel lmindig végig csinálni, mert aztán már session keyt az izéből generálunk.

**Downgrade attack**
Itt is lehet úgy MITM-t csinálni, hogy gyengébb algoritmut csináljjunk az egyik irányba.




# Kiegészítő cuccosok (nem kérdés, de már leírtam szal anyway)

# Euklideszi algoritmus

**Lényeg**
LNKO-t kapod meg vele két számról. Legyen a és b két nem nulla egész, és mondjuk a > b. Ekkor az algoritmus így megy:

1. lépés: maradékosan osztjuk $a$-t $b$-vel. Ekkor:

$$a = b \cdot q_1 + r_1$$

2. lépés: maradékosan osztjuk $b$-t az előző osztás maradékával, azaz $r_1$-gyel. Ekkor:

$$b = r_1 \cdot q_2 + r_2$$

i. lépés: maradékosan osztjuk $r_i$-t az előző osztás maradékával $r_{i-1}$-gyel. Ekkor:

$$r_i = r_{i-1} \cdot q_{i+1} + r_{i+1}$$

Látszik, hogy faszán lemegy egészen nulláig majd a maradék, és ha ez bekövetkezik, akkor az utolsó nem nulla maradék lesz az LNKO.

**Videón**
https://www.youtube.com/watch?v=JUzYI1TYMcU (ez maga a számolás bemutatása, hogy hogyan kell használni az algoritmust)

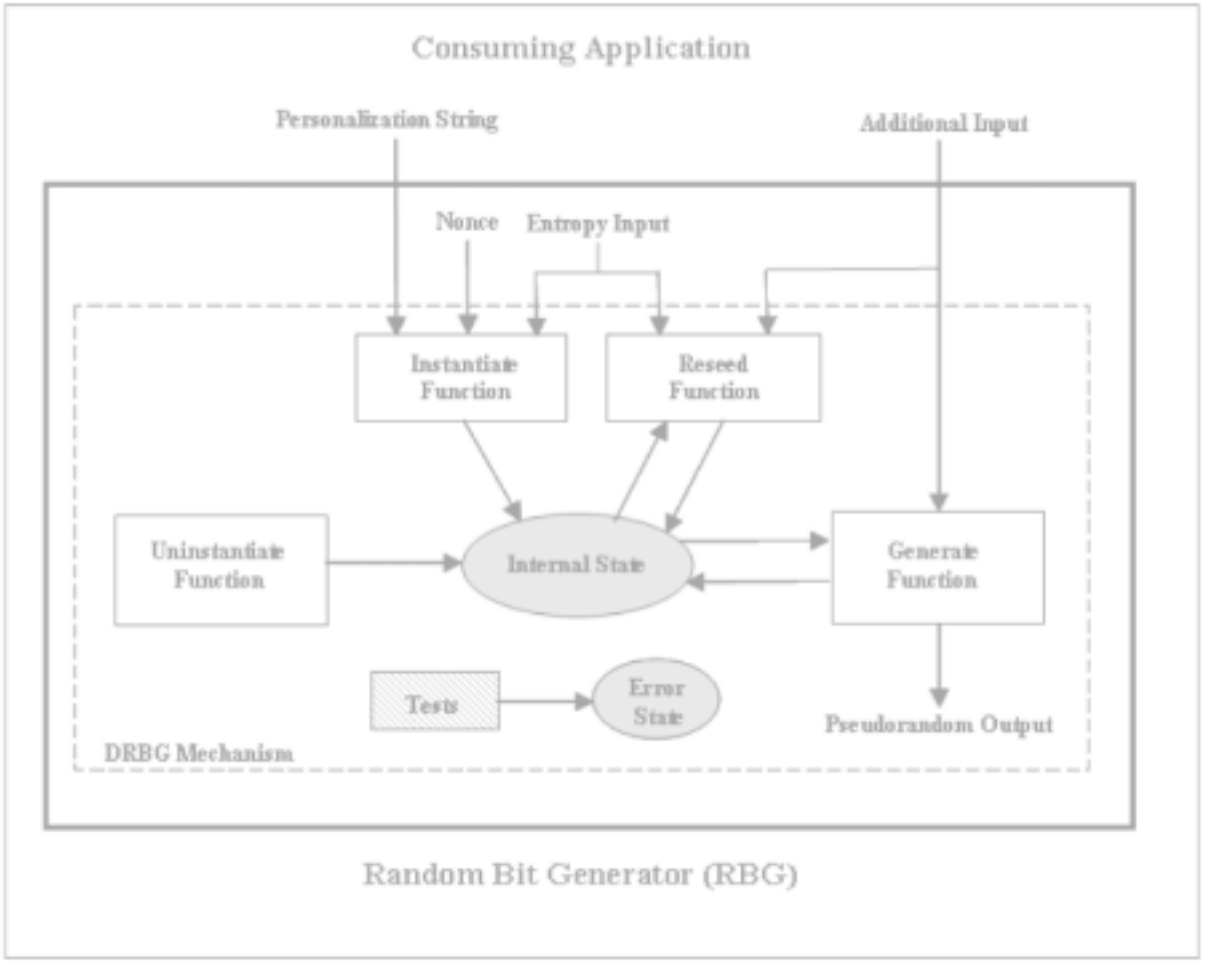
**Kiterjesztett euklideszi algoritmus**
Meghatározza két szám LNKO-ját r-et és a legkisebb t-t, amelyre tb = r (mod n).

# Véletlen generálás

**Fogalmak**
- RBG: Random Bit Generator
- NRBG: Non-deterministic Random Bit Generator

- DRBG: Deterministic Random Bit Generator (Pseudorandom Bit Generator)
- Cryptographically Secure DRBG: ez nem csak hogy jól működik, de hatékony is ezért lehet használni kriptográfiában

**Funkcionális modell**




Azt kell tudni, amik az ábrán vannak kb. ezek:

- Entrópia
  - Magérték (seed) létrehozása
  - titkos
  - nem a hossza meghatározott, hanem a bizonytalansága
- Egyéb bemenet - nonce
  - személyre szabott érték
  - nem feltétlenül titkos
- Állapot
  - Minden belső adat, ami a következő állapot kiszámításához kell
- Műveletek:
  - Egyed létrehozása (instantiate)
  - Új magérték létrehozása (reseed)
  - Törlés (uninstantiate)
  - Gyártás (generate)
- Teszt