Adatbiztonság 2. kisZH

Kolbay Zsolt által elkezdett összefoglaló befejezése.
A tartalomért felelősséget nem vállalok, mindenki saját felelősségére használja!

Proto 2 (5. ea.)

Integritásvédelem

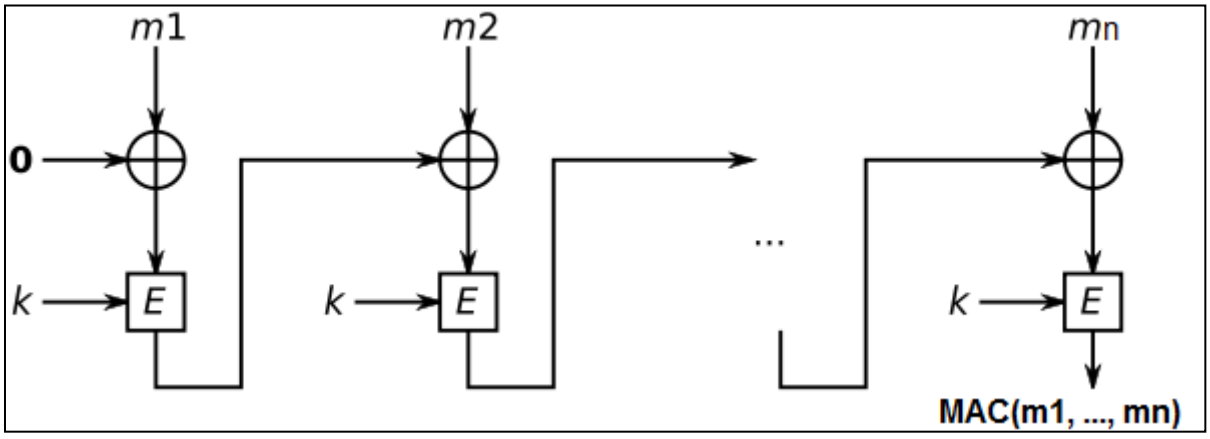
Az üzenethitelesítés (integritásvédelem) feladata az, hogy a vételi oldalon detektálhatóvá tegyük azon eseményeket, amelyek során az átviteli úton az üzenet valamilyen módosulást szenvedett el. Leggyakrabban üzenethitelesítő kódok (Message Authentication Code – MAC) alkalmazásával oldják meg. Ez abban különbözik a hibadetektáló kódoktól (pl. CRC), hogy értéke nem csak az üzenettől függ, hanem egy a küldő és a vevő által megosztott titkos információtól (kulcs) is, így nemcsak a bithibákat, hanem a rosszindulatú módosításokat is képes detektálni.
A MAC függvényekkel szemben támasztott követelmények:

1. A MAC függvény szűkítő transzformáció kell hogy legyen, amely képes tetszőleges hosszú üzeneteket egy fix hosszúságú, n bites blokkba leképezni.
2. A K kulcs ismeretében tetszőleges m üzenetre könnyű legyen kiszámolni $MAC_K(m)$-et.
3. A K kulcs ismeretének hiányában $MAC_K(m)$ kiszámítása legyen nehéz feladat, még akkor is ha nagy számú $\{m_i \neq m,\ MAC_K(m_i)\}$ párt ismerünk.
4. A K kulcs meghatározása nehéz feladat, még nagy számú $\{m_i,\ MAC_K(m_i)\}$ pár ismerete esetén is.

Módszerek integritásvédelemre:

- kriptográfiai ellenőrző összeg (CBC-MAC)
- kulcsolt hash
- rejtjelezés
- digitális aláírás
- speciális alkalmazások (pl: Zero Knowledge Protocol)

CBC-MAC (Message Authentication Code)



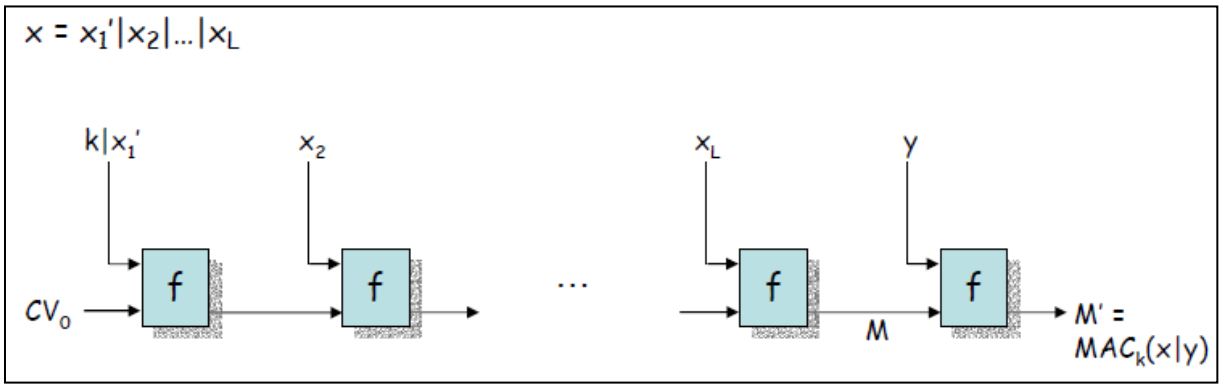
Az m üzenetet szeretnénk integritásvédelemmel ellátni. Először n darab egyforma hosszú blokkra osztjuk fel: $m = (m_1, \ldots, m_n)$, az utolsó blokkot szükség szerint kiegészítjük, hogy ugyanolyan hosszú legyen, mint a többi.

Az üzenetet CBC módban rejtjelezzük (IV = 0 értékkel) és a rejtjelezett üzenetnek az utolsó blokkját használjuk MAC értékként. A vevő félnek átküldött adat: $[(m_1, \ldots, m_n) \mid MAC_K(m_1, \ldots, m_n)]$.

Hash függvényre épülő MAC függvények: titkos prefix, titkos suffix és a szendvics módszer

Az üzenethez (x) valamilyen módon hozzávesszük a titkos kulcsot (K), és egy kiválasztott hash függvénnyel (h) kiszámítjuk a hash értékét, ami a MAC érték lesz.

Titkos prefix módszer: az üzenet elé fűzzük a kulcsot, vagyis $MAC_K(x) = h(K \mid x)$.
Nem biztonságos: iterált hash függvény esetén, ha a támadó ismeri az x üzenet MAC értékét, akkor kiszámíthatja tetszőleges y-ra $MAC_K(x \mid y) = h(K \mid x \mid y)$ értéket, vagyis képes lesz az eredeti üzenet után tetszőleges saját üzenetet beszúrni és hitelesíteni.



Titkos suffix módszer: az üzenet után fűzzük a kulcsot, vagyis $MAC_K(x) = h(x \mid K)$.
Nem biztonságos ha iterált hash függvényt használunk, ami nem ütközésmentes. Ekkor a támadó találhat egy olyan $x' \neq x$ üzenetet (pl: születésnapi paradoxon módszerével), amelyre h(x') = h(x). Ilyenkor h(x' | pad') = h(x | pad). Tehát x üzenet lecserélhető egy x' üzenettel, úgy hogy a vevő fél hitelesnek találja azt.

Szendvics módszer: az üzenet elé és után is berakjuk a kulcsot, vagyis $MAC_K(x) = h(K \mid x \mid K)$. Ha két kulcs van (vagy a kulcs két részre bontható): $MAC_{(K, K')}(x) = h(K \mid x \mid K')$.
Biztonságosabb az előző két megoldásnál.

HMAC hitelesítő kód

$$HMAC_K = h(K^+ opad \mid h(K^+ \oplus ipad \mid m))$$

h egy iteratív hash függvény, amely az üzenetet B bájtos blokkokban dolgozza fel
ipad (inner pad) egy B bájt hosszú konstans blokk, amelyben minden bájt értéke 36
opad (outer pad) egy B bájt hosszú konstans blokk, amelyben minden bájt értéke 5C
K^+ a K kulcs csupa 0 bittel kiegészítve, hogy hossza elérje a B bájtot

Kulcscsere

Alapvetően két fajtája van*: kulcsszállító* és *kulcsmegegyezés* protokollok.
Feladatai: generálás, tárolás, csere, frissítés, visszavonás.
Szolgáltatásai:

- Implicit kulcshitelesítés: a fél meg van győződve arról, hogy rajta kívül csak a másik fél, illetve megbízható harmadik fél fér hozzá a kulcshoz.
- Kulcskonfirmáció: a fél meg van győződve arról, hogy a másik fél valóban birtokában van a létrehozott kulcsnak.
- Explicit kulcshitelesítés: implicit + kulcskonfirmáció együtt.
- Kulcsfrissesség: a fél meg van győződve róla, hogy a kulcs új, nem egy korábban használt.
- Partnerhitelesítés: a fél meg van győződve arról, hogy a másik fél valóban részt vett a protokoll futásában.

Támadás: célja a kulcs megszerzése, illetve annak elhitetése, hogy a fél azt a másik megbízható féllel hozta létre.

Támadás típusai:

- passzív lehallgatás
- protokoll üzenet törlései, módosítása, visszajátszása (replay)
- megszemélyesítés (parallel session)

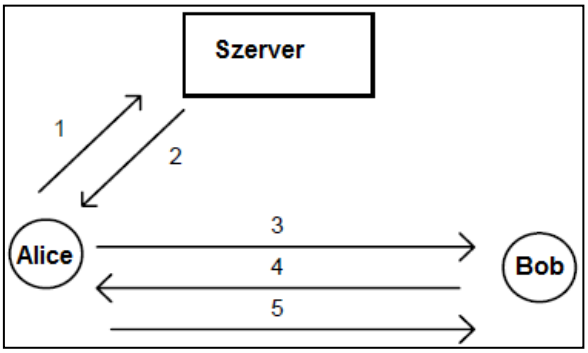
Wide-Mouth Frog protokoll

Kulcsszállító protokoll, amelynek résztvevői: *Alice* és *Bob*, akik szeretnének egy kapcsolatkulcsot létrehozni egy harmadik megbízható fél (*Szerver*) segítségével. A protokoll feltételezi, hogy Alicenek van egy közös titkos kulcsa a Szerverrel (K_{AS}), és Bobnak is (K_{BS}). Így Alice a K_{AS} kulcsot felhasználva el tudja juttatni a Szerveren keresztül Bobnak a k kapcsolatkulcsot.

1. Alice -> Szerver: $ID_{Alice} \mid \{ID_{Bob} \mid k \mid T_{Alice}\}_{K_{AS}}$	Alice elküldi a Szervernek Bob azonosítóját (ID_{Bob}), a frissen generált kapcsolatkulcsot (k) és egy friss időbélyeget (T_{Alice}), mindezt a K_{AS} kulccsal titkosítva.
2. Szerver -> Bob: $\{ID_{Alice} \mid k \mid T_{Szerver}\}_{K_{BS}}$	A Szerver elküldi Bobnak Alice azonosítóját (ID_{Alice}), az Alice által generált kapcsolatkulcsot (k) és egy friss időbélyeget ($T_{Szerver}$), mindezt a K_{BS} kulccsal titkosítva.

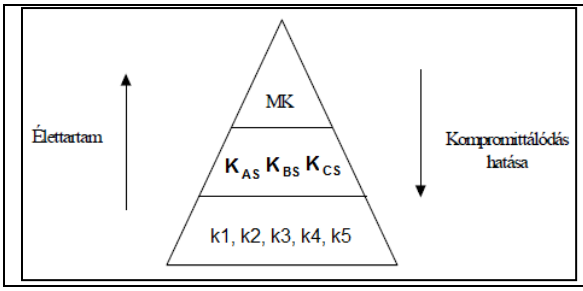
Needham-Schroeder protokoll (szimmetrikus kulcsú)

Kulcsszállító protokoll, amelynek résztvevői: *Alice* és *Bob*, akik szeretnének egy kapcsolatkulcsot létrehozni egy harmadik megbízható fél (*Szerver*) segítségével. A protokoll feltételezi, hogy Alicenek van egy közös titkos kulcsa a Szerverrel (K_{AS}), és Bobnak is (K_{BS}).



1. Alice -> Szerver: $\mathrm{ID_{Alice}}$ |$\mathrm{ID_{Bob}}$| $\mathrm{Rnd_{Alice}}$	Alice generál egy véletlenszámot ($\mathrm{Rnd_{Alice}}$) és elküldi a Szervernek a saját azonosítójával és Bob azonosítójával együtt.
2. Szerver -> Alice: $\{\mathrm{Rnd_{Alice}}$ |$\mathrm{ID_{Bob}}$ |k | $\{k \mid ID_{Alice}\}_{K_{BS}}\}_{\mathrm{K_{AS}}}$	A Szerver generál egy kapcsolatkulcsot(k), majd válaszol Alicenek. Az Alice és a Szerver közös kulcsával (K_{AS}) titkosított üzenet két részből áll: 1. Alice véletlenszáma, Bob azonosítója és a kapcsolatkulcs. 2. A kapcsolatkulcs és Alice azonosítója titkosítva Bob és a Szerver közös kulcsával (K_{BS}).
3. Alice -> Bob: $\{k \mid ID_{Alice}\}_{K_{BS}}$	Alice elküldi Bobnak a Szervertől kapott üzenet második felét: Bob és a Szerver közös kulcsával titkosított kapcsolatkulcsot és Alice azonosítóját.
4. Bob -> Alice: $\{\mathrm{Rnd_{Bob}}\}_{\mathrm{k}}$	Bob (felhasználva a közös kulcsát a Szerverrel (K_{BS})) kiolvassa az üzenetből a kapcsolatkulcsot, és azzal titkosít egy általa generált véletlenszámot (Rnd_{Bob}), majd elküldi Alicenek.
5. Alice -> Bob: $\{\mathrm{Rnd_{Bob}} - 1\}_{\mathrm{k}}$	Alice kiolvassa a kapcsolatkulcs segítségével a véletlenszámot, levon belőle egyet, majd visszaküldi Bobnak. Bob ellenőrzi, hogy a kapott szám eggyel kisebb-e annál amit Alicenek küldött.

Kulcshierarchia:

	MK: mesterkulcs K_{AS}, K_{BS}, K_{CS}: viszonykulcsok egy kliens (pl. Alice) és a szerverközött k1, k2, k3, k4, k5: kapcsolatkulcsok két kliens (pl: Alice és Bob) között

Diffie-Hellman protokoll

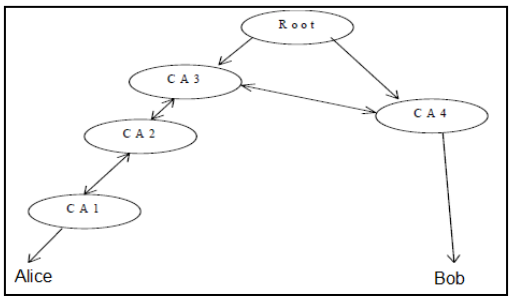
Kulcsmegegyezés protokoll (mindkét kommunikáló fél részt vesz a kapcsolatkulcs generálásában). Feltesszük, hogy Alice és Bob birtokában vannak egy nagy *p* prímszám, és az általa generált véges Z_p multiplikatív csoport egy *g* generátor eleme. A protokoll a diszkrét logaritmus problémát használja ki ($g^x(mod\ p) = M$ esetén, ha g és M adott, akkor x kiszámítása nehéz feladat). A kulcshitelességet nem biztosítja, ezért csak passzív támadás ellen véd.

1. Alice -> Bob: $g^x(mod\ p)$	Alice generál egy *1 < x < p-1* véletlenszámot, majd kiszámítja $g^x(mod\ p)$ értéket és elküldi Bobnak.
2. Bob -> Alice: $g^y(mod\ p)$	Bob generál egy *1 < y < p-1* véletlenszámot, majd kiszámítja $g^y(mod\ p)$ értéket és elküldi Alicenek.
3.a) Alice: $k = (g^y)^x(mod\ p) = g^{x \cdot y}(mod\ p)$ 3.b) Bob: $k = (g^x)^y(mod\ p) = g^{x \cdot y}(mod\ p)$	Mindkét fél a kapott számot a saját véletlenszámának hatványára emeli, a kapott eredmény adja a közös kulcsot (ami a hatványozás kommutativitása miatt valóban mindkét oldalon ugyanaz).

Az X.509 tanúsítvány komponensei

- verziószám
- sorozatszám
- algoritmus azonosító
- a tanúsítvány kibocsátó (CA) azonosítója
- tanúsítvány érvényességi időtartam (Not Before Date, Not After Date)
- a tanúsítvány tulajdonosának azonosítója
- a tanúsítandó **publikus kulcs**
- **digitális aláírás** (a fenti adatokra)

Tanúsítvány szolgáltatók hierarchiája



Nyilvános kulcsú tanusítványokat hitelesítés-szolgáltatók (Certification Authority – CA) generálják, akik egymást is hitelesítik. A hierarchia csúcsán a gyökér hitelesítés szolgáltató áll (Root CA), amelynek publikus kulcsa nem tanusítvánnyal hitelesített, hanem mindenki hitelesnek feltételezi.
Amikor Alice és Bob egymással egy nyilvános kulcsú kriptográfiára épülő protokollt szeretne futtatni, be kell szereznie a másik fél nyilvános kulcsát igazoló tanúsítványokat.
Alice az alábbi tanusítványokat küldi el ($\{K^p_{Alice}\}$CA1 azt jelenti, hogy Alice publikus kulcsát CA1 tanúsítja):
Alice -> Bob: $\{K^p_{Alice}\}$CA1, $\{K^p_{CA1}\}$CA2, $\{K^p_{CA2}\}$CA3, $\{K^p_{CA3}\}$CA4
Bob ezt fordított sorrendben dolgozza fel: először CA4 alapján megállapítja CA3 hiteles publikus kulcsát, CA3 alapján CA2 hiteles kulcsát, és így tovább, egészen addig amíg Alice publikus kulcsát nem hitelesíti egy hitelesítés-szolgáltató.

Titokmegosztás, Shamir módszere

Egy (m-1)-edfokú $y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{m-1} \cdot x^{m-1} (mod\ p)$ egyenlet által leírt görbét egyértelműen meghatározza annak m darab tetszőlegesen felvett különböző pontja (<x, y> pár).
Legyen a titok $s = (a_0, a_1, \dots, a_{m-1})$.
N darab résztvevő fél esetén kiszámoljuk N különböző pontját a görbének, ezeket kiosztjuk a résztvevőknek. Így a titok felfedéséhez legalább M résztvevő együttműködése szükséges.

Zero-Knowledge protokollok

Legyen $n = p \cdot q$, ahol p és q két nagy prímszám.
$Ha\ \exists x, amelyre\ x^2 = c\ (mod\ n), akkor\ a\ c\ számot\ kvadratikus\ maradéknak,$
$x\ megoldást\ pedig\ c\ négyzetgyökének\ nevezzük\ modulon\ n.$
Ha csak a c és n értékeket ismerjük, akkor x meghatározása nehéz feladat, de ha n faktorait (p és q) is ismerjük, akkor már könnyű.
Ez felhasználható Zero-Knowledge protokolloknál, ahol az egyik fél (Alice) azt szeretné bizonyítani a másiknak (Bob), hogy birtokában van egy titoknak, anélkül hogy felfedné azt a titkot.

A Fiat-Shamir partner-hitelesítési protokoll egy kulcskiosztó központot használ, amely generál két véletlen prímszámot, p-t és q-t, majd az $n = p \cdot q$ modulust kiadja Alice-nek és Bobnak (de p-t és q-t nem). Ezután sorsol Alice-nek egy u titkos kulcsot (véletlenszám), és egy $v = u^2 (mod\ n)$ publikus kulcsot.

Ezután ha Alice hitelesíteni akarja magát Bob előtt:

1. Alice kisorsol egy 0 < R < n véletlenszámot, majd elküldi $z = R^2 (mod\ n)$ számot Bob-nak
2. Bob visszaküld egy véletlen b bitet (b = 0 vagy b = 1) Alice-nek

Ha b = 0:

3. Alice elküldi az R számot
4. Bob ellenőrzi a $z = R^2 (mod\ n)$ értéket

Ha b = 1

3. Alice elküldi Bob-nak a $w = R \cdot u (mod\ n)$
4. Bob ellenőrzi a $z \cdot v = w^2 (mod\ n)$ értéket

Alice így bizonyítja, hogy ismeri u értékét, Bob azonban csak R és z számokkal együtt tudná u-t kiszámítani, ezek közül csak egyet kap meg.
Ha Alice birtokában van az u titkos kulcsnak, akkor mindkét esetben probléma nélkül el tudja küldeni a megfelelő választ. Ha nem rendelkezik vele és át akarja verni Bob-ot, akkor meg kell próbálnia megjósolni b értékét:

1. Ha b = 0-ra tippel, akkor generál egy tetszőleges R számot, az első lépésben elküldi a négyzetes maradékát Bob-nak, és ha valóban b = 0-t kap vissza, akkor a harmadik lépésben elküldi R-t és sikerült az átverés. Ha b = 1, akkor nem tudja elhitetni Bob-bal. A siker esélye így $^1/_2$.
2. Ha b = 1-re tippel, akkor az első lépésben a $z = R \cdot v^{-1} (mod\ n)$ értéket küldi át, és ha valóban b = 1-et kap vissza, akkor a harmadik lépésben R-t. Ha b = 0, akkor az átverés nem sikerült, mert Alice nem tudja kiszámítani z modulo négyzetgyökét. A siker esélye így $^1/_2$.

Tehát Alice $^1/_2$ eséllyel verheti át Bob-ot. Azonban ha Bob k alkalommal játssza le az 1-4 lépéseket, akkor Alice esélye 2^{-k}-ra csökken.

Vak aláírás

RSA titkosítást alkalmazva, ahol

- e: kódoló kulcs (nyilvános)
- d: dekódoló kulcs (titkos)
- n: két nagy prímszám szorzata (nyilvános, $e \cdot d = 1\ (mod\ \varphi(n))$)

- Véglegesíti a dokumentumot: X
- Behelyezi egy átlátszatlan borítékba:
  - egy r < n véletlen érték kiválasztása, ahol (r, n) = 1
  - $Y = r^e \cdot X (mod\ n)$
- Y átküldése Bob-nak

- Aláírás: $S = Y^d (mod\ n)$
- Bob nem tudja mit ír alá
- S visszaküldése Alice-nek

Aláírás ellenőrzése:

$r^{-1} \cdot S =$
$r^{-1} \cdot Y^d =$
$r^{-1} \cdot r^{e \cdot d} \cdot X^e =$
$r^{-1} \cdot r \cdot X^d = X^d$

Felhasználó hitelesítés (6. ea.)

A kommunikációban résztvevő fél azonosítja magát (pl: felhasználói név megadásával), majd megpróbálja bizonyítani kilétét, ezt tipikusan háromféleképpen teheti meg:

- rendelkezik valamivel (azonosító kártya, mobiltelefon, stb.)
- tud valamit (PIN-kód, szöveges jelszó, stb.)
- valamilyen tőle elválaszthatatlan azonosítóval (ujjlenyomat, retinaminta, stb.)

Jelszó alapú azonosítás

Előnyei:

- egyszerű és intuitív (a felhasználók könnyen megértik a használatát)
- olcsóbb implementáció

Hátrányai:

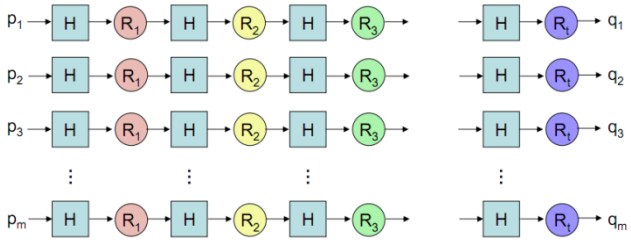
- a felhasználóknak meg kell jegyezniük a jelszavakat, ezért sokszor egyszerű jelszavakat választanak (gyakori jelszavakat (pl: '12345' vagy olyan jelszavakat amelyek a személyes adataikhoz kapcsolódnak (pl: születési dátum)) és/vagy ugyanazt a jelszót használják több helyen is
- a hitelesítő rendszernek tárolnia kell a jelszavakat vagy a hash értéküket, ezért ha valaki bejut a rendszerbe és ellopja, akkor off-line elemezheti és szótár alapú támadást indíthat
- a jelszót elkaphatja a támadó amíg az eljut a felhasználótól a hitelesítő rendszerhez (key logger alkalmazása, hálózati csatorna lehallgatása, visszajátszásos támadás)
- a felhasználó maga is akaratlanul felfedheti a jelszavát (pl: felírja egy cetlire és otthagyja az asztalán)

Támadási módok:

- guessing: gyakran használt jelszavak (qwerty, 1234), illetve a felhasználó tulajdonságaiból (születési dátum) kitalálható jelszavakkal próbálkozás
- dictionary: a jelszavak jelentős százaléka a weben is fellelhető szótárakból épül fel, ezt használják ki az alkalmazások
- brute force: minden lehetséges bemenet kipróbálása

Rainbow táblák

H hash fgv, és R redukciós fgv, ami hash értékeket valós jelszavakká képez le (nem a H megfordítása, csak megfelelő kimenetet ad, ami akár jelszó is lehetne!)
Választunk tetszőleges jelszavakat, és láncokat készítünk H és R váltakozásával. A láncoknak csak az elejét és végét tároljuk (lookup table), ezzel csökkentve a tárigényt.



Ha megszereztük a feltörni kívánt jelszó *h* hash értékét, akkor megnézzük, hogy szerepel-e a lookup table-ben. Ha nem, akkor kiszámoljuk $R_t(h)$ értékét, majd ellenőrizzük. Ha így sincs találat, kiszámoljuk $R_t(H(R_{t-1}(h)))$ értékét és ellenőrizzük … így tovább, amíg találunk egy megegyező q_n értéket.
Ekkor elindulunk az n. láncon, azaz $h_0=H(p_n)$, $h_1=H(R_1(h_0))$ … amíg $h_i=H(R_i(h_{i-1}))$ == h.
Így a keresett jelszó $R_i(h_{i-1})$.

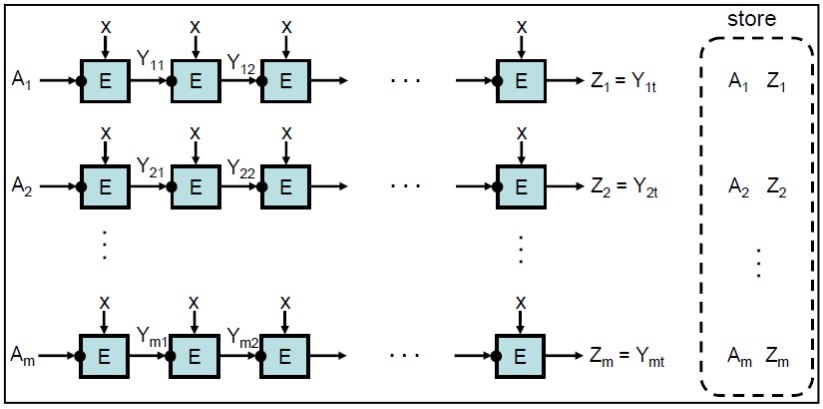
Brute Force támadás felgyorsítása, Hellman módszere

Választott üzenet alapú támadást feltételezünk: egy általunk választott X üzenethez ismerjük az $Y=E_K(X)$ rejtjelezett üzenetet. Ennek alapján akarjuk a k bites K kulcs értékét kiszámolni.

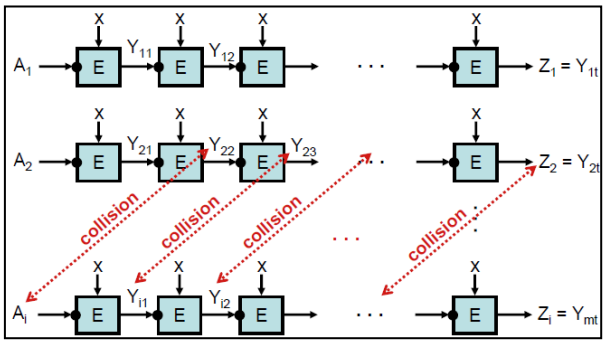
Ha a kimerítő keresést választjuk: kb. 2^{k-1} számítást kell végrehajtani, külön memóriát nem igényel.

Ha kiszámoljuk $E_K(X)$ értéket az összes lehetséges K kulcsra: egyetlen keresést és kb. 2^k darab érték tárolásához szükséges memóriát igényel.

Egy harmadik megoldás: számoljunk $m \cdot t \approx 2^k$ darab Y_{ij} értéket adott X-re az ábrán látható módon és tároljuk el az A_i-Z_i párokat.



Adott $Y=E_K(X)$ érték esetén kiszámoljuk $Y_1=E_Y(X)$, $Y_2=E_{Y1}(X)$, ..., $Y_n=E_{Y(n-1)}(X)$ értékeket egészen addig amíg valamelyik i,n-re $Y_i=Z_n$ vagy i > t. Ezután a megfelelő Z_n érték alapján kiszámoljuk $Y_{n1} = E_{An}(X)$, $Y_{n2} = E_{Yn1}(X)$, … egészen addig amíg $Y = E_{Ynj}(X)$. Ekkor megtaláltuk a keresett kulcsot: $K = Y_{nj}$.
Ez nagyjából 2t számítást és 2m memóriát igényel.
A megoldás hátránya, hogy minél nagyobb táblát használunk, annál több ütközés lesz benne (ha egy Y_{ij} és egy Y_{pr} érték ütközik akkor a i. és p. sorokban a j. és r. értékek után következő blokkok is ütközni fognak).



Hellman megoldása: alkalmazzunk kisebb táblákat (kisebb a valószínűsége az ütközéseknek), de egyszerre nem egy hanem r darabot.

Jelszó erősségének mérése

A brute force támadás esetén mérhető a jelszó erőssége az alábbi formulával:

$$H = L * log_2 N$$

,ahol L a jelszó hossza, míg N a lehetséges karakterek száma. H mértékegysége bit.

Természetesen nem lehet teljes precizitással mérni az erősséget. Becslés a karaketerek entrópiájára: 1. karakter: 4 bit 2-8 karakter: 2 bit (karakterenként) 9-20 karakter: 1.5 bit e fölött 1 bit/karakter.

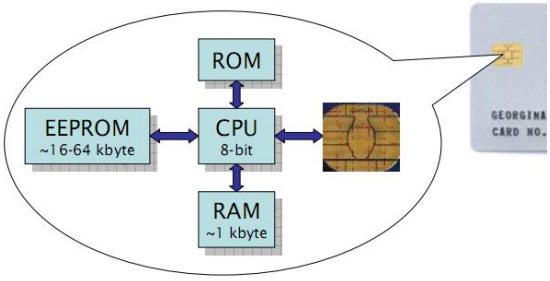
Autentikáció HW alapon

One-time password generálás:

- OTP küldése SMS-ben: napjainkban a bankok ezt használják pl. átutalásokhoz.
- off-line OTP generátorok: periodikusan generál új OTP-t az időből illetve egy titokból. A tokennek szinkronban kell lennie az ellenőrző rendszerrel.
- általában kombinált használat jelszóval, PINnel

Smart kártyák

Támogatják a kriptográfiai algoritmusokat.



Hozzáférés védelem: egyszerű interfészük van, melyhez a hozzáférést az OS ellenőrzi. PIN alapú ellenőrzés, bizonyos számú rossz kísérlet után tiltás.

Fizikai védelem: feszültség védelem glitch attack ellen, frekvencia szenzor a cpu lassítása/gyorsítása ellen, belső buszrendszer az adatok titkosítására, az analízis megnehezítése érdekében, stb.

Veszélyforrás: kártyán nincs UI, így egy terminálnak kell megadni a PIN-t. Azonban lehet, hogy a terminált rosszindulatú támadó üzemelteti, aki megszerezheti a kódot.

Kijelzővel ellátott smart kártya

A kártya egy véletlen számsorozatot jelez. A felhasználó növelheti, illetve csökkentheti az értéket számjegyenként, egészen addig, amíg el nem éri a PIN-t.

Biometrikus azonosítás

Bármely emberi élettani vagy viselkedési jellemző alapja lehet biometrikus azonosításnak, ha a következő tulajdonságokkal rendelkezik:

- általános: minden emberben fellelhető a karakterisztika
- egyedi: nincs két olyan ember, akinél megegyezne a tulajdonság
- permanens: időben nem változik
- ellenőrizhető: a tulajdonság mérhető valamilyen módon
- kijátszhatatlan: a rendszert nehéz átverni

Példák:

- ujjlenyomat
- írisz (szivárványhártya)
- arc
- fül
- retina
- hang
- kéz geometriája
- billentyűleütés
- thermo

Ujjlenyomat

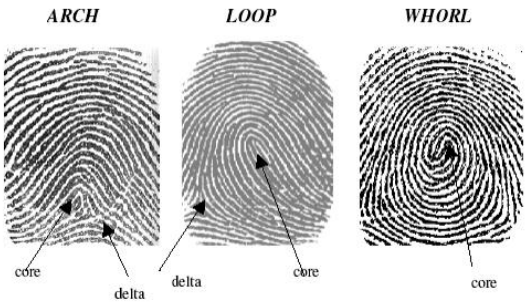
A bűnügyi nyomozásokban évszázados múltja van, mára már - hála a fejlett technológiának – a nem bűnügyi alkalmazások is használják.

Két féle ellenőrzési probléma:

- one-to-one: ujjlenyomat verifikáció, két ujjlenyomat azonos-e
- one-to-many: ujjlenyomat azonosítás, adatbázisból

Felépítés:

- ridge: az ujjlenyomat mintázata
- valley: két ridge közötti távolság
- minutiae
    - ending: véget ér egy ridge
    - bifurcation: ketté válik egy ridge (Y elágazás)



3 féle típus

Feldolgozás lépései:

- ujjlenyomat vételezése
- zajcsökkentés, képjavítás
- adaptive thresholding: binarizálás
- keskenyítés: ridge egy pixelesre változtatása
- minutiea detektálás

Ujjlenyomat verifikáció: Leggyakoribb a minutiae alapú.

Teljesítmény: Természetesen adódnak problémák az ujjlenyomat olvasásakor, pl: nem illeszkedik jól, más az orientációja, al-régiók nem illeszkednek (bőr tulajdonsága miatt), stb.

CAPTCHA

Completely Automated Public Turing test to tell Computers and Humans Apart

Cél: nem azonosítani a felhasználót, csak meggyőződni arról, hogy ember, és nem bot.
Tipikus megoldás: probléma, ami az ember számára könnyű, de gép számára nehéz, pl: kép feldolgozása

Alkalmazás

- védelem a spam kommentektől fórumokban, blogokban
- regisztrációk védelme
- e-mail címek védelme
- szavazatok védelme
- kereső robotok elleni védekezés
- on-line brute force megakadályozása

Támadási lehetőségek

- tervezési hibák kiaknázása
    - ismert kép session id-jának újbóli használata
    - kevés kép alkalmazása
- karakterfelismerés fejlesztése
    - pre-processing: háttér, és zaj eltávolítása
    - szegmentálás: karakterenként szétvágjuk a képet, és azt próbáljuk felismerni
- olcsó emberi munkaerő alkalmazása

reCAPTCHA

Egy project, ami a könyvek digitalizálást segíti, míg megvédi az oldalakat a spam botoktól.

Működése: a digitalizált szövegen két OCR programot is futtatnak. Amennyiben nem egyeznek meg a feldolgozásban, úgy a szóból egy CAPTCHA képet generálnak, és egy már ismert control szóval együtt jelenítik meg. Ha a felhasználó azt helyesen írja be, akkor feltételezhető, hogy a kérdéses szó is helyes.

MailHide: a projekt felkínál egy html kódot, melyet az oldalba illesztve a böngésző a reCAPTCHA szerverére irányítja, és csak akkor derül ki az e-mail cím, ha az ott elhelyezett CAPTCHA-t helyesen megoldja.

Internet biztonsági protokollok (7. ea.)

Ezen protokollokat két vagy több fél által nem biztonságos csatornán (Internet, wifi hálózat) folytatott kommunikáció biztosítására használják.
Általában úgy tekintjük, hogy a csatorna teljesen a támadó hatásköre alatt áll (képes módosítani, törölni, beszúrni, visszajátszani üzeneteket, stb.)

ARP: Address Resolution Protocol – címfeloldási protokoll. (IP → MAC)
Probléma: ARP Spoofing: bárki válaszolhat az ARP kérésre

Célok:

- Felhasználó és/vagy üzenet feladójának hitelesítése
- Üzenet integritás és visszajátszás-védelem
- Üzenetek megbízhatóságának biztosítása

TLS – Transport Layer Security

Cél: biztonságos átviteli kapcsolat nyújtása (általában kliens szerver között, https)

- Kölcsönös hitelesítés a felek közt
- Üzenetek titkosítása, integritás- és visszajátszás-védelme

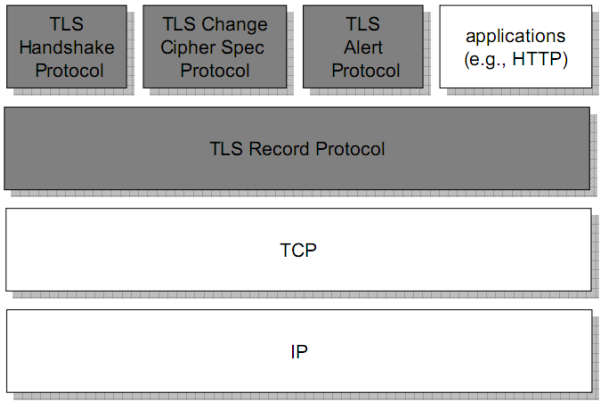
Architektúra

Record: Kommunikáció védelme, tördelés, tömörítés, titkosítás a feladata.

Handshake: Itt zajlik a kriptográfiai algoritmusok és paraméterek egyeztetése, valamint a kulcscsere.

Cipher Change: legegyszerűbb protokoll, feladata a nyugtázás egy handshake után.

Alert: rendszerüzeneteket továbbít az esetleges figyelmeztetésekről, hibákról.

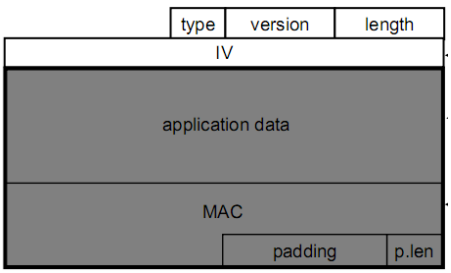


Session-ök, kapcsolatok: egy session tartalmazhat több párhuzamos kapcsolatot is. A sessionnek van állapota (ebben tárolja a handshake alatt megosztott közös titkot, a kriptográfiai protokollokat, azok paraméterét, illetve az esetleges tanúsítványokat), ami azonos minden tartalmazott kapcsolatra. A kapcsolatoknak a közös titokból származtatott saját kulcsuk van.

TLS Record Protocol

MAC: HMAC with MD5, SHA1, SHA256, titkosítás előtt számolt

Rejtjelezett rész (szürke): alapértelmezett rejtjelező az RC4 kulcsfolyam rejtjelező, de támogatott CBC módú blokkrejtjelezés is (3DES, AES)



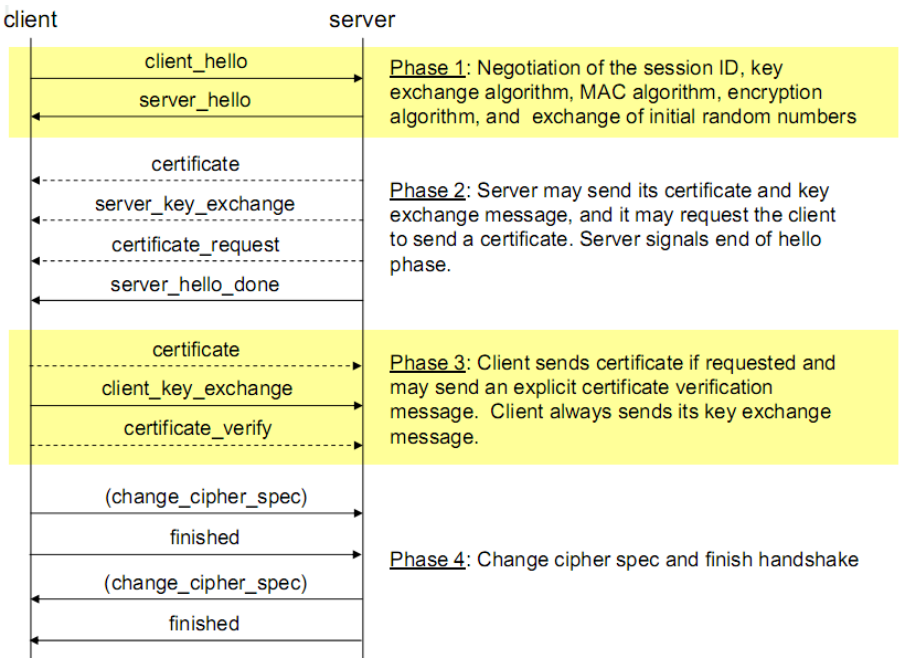
TLS Alert Protocol

2 byte, első: „warning/fatal”, második pedig az altípus.

Amennyiben „fatal” üzenet jön, a session érvénytelenítésre kerül és a kapcsolat megszakad.

TLS Handshake Protocol

client-hello: kliens verzió, véletlen szám, session id (ha új kapcsolatot akar akkor jelenlegi session id-t teszi bele, ha új session-t akar, akkor üres), lehetséges biztonsági algoritmusok felsorolása, tömörítő algoritmusok
server hello: szerver verzió, véletlen szám, session id (ha kliens kért, akkor kap új kapcsolatot [amennyiben nincs hely aktuális session-ben, új session-t kap], ha üres volt, új session-t), választott biztonsági és tömörítő algoritmus.



Támogatott kulcsrendszerek: RSA alapú, fix DH, egyszer használatos DH, anonim DH, Fortezza

certificate: anonim DH kivételével mindig kell, X.509 tanúsítványlánc
server_key_exchange: ha az előző tanúsítvány nem tartalmazott elég információt a kulcscsere befejezéséhez
certificate_request: ha a kliensnek is azonosítania kell magát
server_hello_done: szerver jelzi, hogy végzett

certificate: csak kérésre (szerver mindig azonosításra kerül, míg a kliens csak kimondottan kérésre!)
client_key_exchange: mindig elküldésre kerül, vagy egy one-time DH érték, vagy a szerver RSA kulcsával rejtjelezett véletlen blokk
certificate_verify: ha a tanúsítványt küldött. Tartalma az eddigi Handshake üzenetek hash értéke digitálisan aláírva

finished: az összes Handshake üzenet hash értékén és a mestertitkon alkalmazott pszeudo-random funkcióval számolt hitelesítő üzenet

IPSec

A hálózati réteg szabványosított biztonsági protokollja, az IIP és minden fölötte levő protokoll számára védelmet biztosít.

Alprotokollok:

- AH (Authentication Header): IP csomagok integritásvédelme, eredetének hitelesítése és visszajátszások detektálása
- ESP (Encapsulated Security Payload): IP csomagok rejtjelezése
- IKEv2: kulcscsere protokoll (nem tárgyaljuk)

Implementálási lehetőségek:

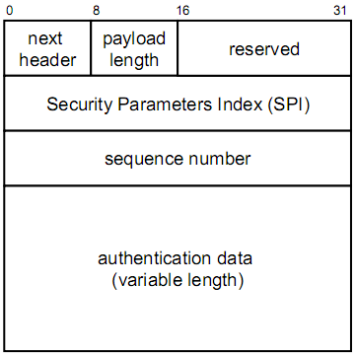
- natív IP stackbe integrálás
- bump-in-the-stacks (BITS)
- bump-in-the-wire (BITW)

Működési módok (lásd lenti ábrák):

- Transport: az eredeti IP fejléc marad, és az AH és ESP fejléc az IP csomag payload-ját burkolja be. Védelmet a felsőbb rétegbeli protokolloknak biztosít.
- Tunnel: eredeti IP fejléc beágyazásra kerül, és az AH és ESP fejléc az új és az eredeti IP fejléc közé kerül. Az egész IP csomagot védi, tipikusan biztonsági átjáróknál használják.

Authentication Header

Integritásvédelmet, eredethitelesítést és visszajátszás védelmet biztosít.



next header: következő fejlécet adja meg, azaz au IP csomag tartalmára utal

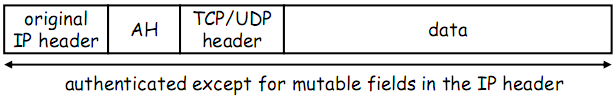
length: AH fejléc hossza 32 bites szavakban

SPI: azt jelzi a fogadónak, hogy milyen módon kell az AH fejlécet feldolgozni (amiről már korábban megegyeztek

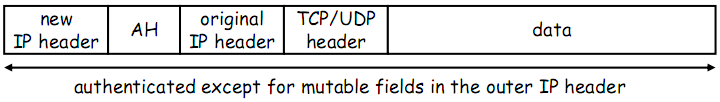
sequence num: sorszám

authenticated data: MAC, a teljes IP csomagra számolva (Az IP fejléc változó mezőit és az AH fejléc MAC mezőjét nullával töltjük ki, így számoljuk)

AH in transport mode



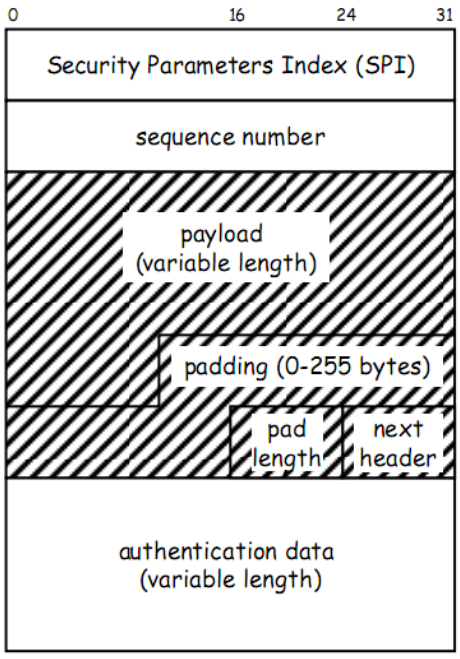
AH in tunnel mode



Visszajátszás detektálás: vevő oldalon egy konstans W méretű ablakkal

Encapsulated Security Payload

Az ESP protokoll feladata az IP csomag tartalmának elrejtése, opcionálisan integritásának védelme. Előbbit rejtjelezéssel, utóbbit a csomag tartalmára számolt MAC csatolásával oldja meg.

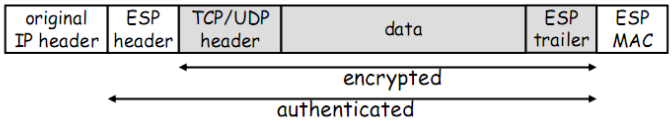


SPI: milyen módon és milyen kulcsokkal kell feldolgozni a csomagot

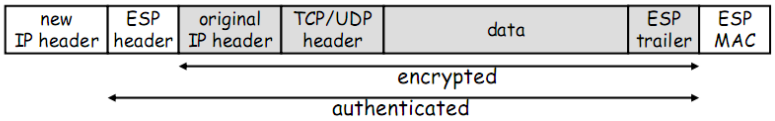
A csíkozott rész a csomag rejtjelezett része. Blokkrejtjelezés (CBC mód, nyílt IV) esetén természetesen ki kell tölteni, ESP MAC zárja a csomagot.

Transport és tunnel mód analóg AH esettel:

ESP in transport mode



ESP in tunnel mode



Szállítási módú alapkombináció (transport adjacency): IP csomagon először az ESP feldolgozást végezzük el hitelesítés (MAC) nélkül, majd az így kapott csomagra végezzük el az AH feldolgozást.

Alagutak egymásba ágyazása:

- az alagutak két végpont azonos (tipikusan hosztok között)
- az alagutak egyik végpont azonos (tipikusan hoszt és tűzfal között)
- az alagutak egyik végpontja sem közös (tipikusan tűzfalak között)

WiFi security

SSID = Service Set IDentifier (hálózat neve) 32 karakteres egyedi azonosítója a hálózatnak. Csomagok fejlécében található és „jelszó"ként viselkedik csatlakozáskor, de könnyen megszerezhető, így nem biztosít valós védelmet.

MAC cím alapú szűrés: csak előre beregisztrált címmel csatlakozhatunk. Könnyen megszerezhető és beállítható ilyen cím.

Problémák:

- nincs fizikai védelem: fizikai kapcsolatokról nem beszélhetünk, nincsenek kábelek, stb.
- broadcast kommunikáció: bárki lehallgathatja és generálhat üzenetet, zavarhatja az adást

Következmények:

- lehallgatás egyszerű
- hamis üzenetek beszúrása egyszerű
- üzenet visszajátszása egyszerű
- a hálózathoz és szolgáltatásaihoz a hozzáférés egyszerű
- DoS egyszerűen megvalósítható zavarással (jamming)

Biztonsági követelmények:

- titkosítás: a hálózaton küldött üzenetek legyenek rejtjelezve
- hitelesség: az üzenetek származása ellenőrizhető legyen
- visszajátszás detektálás: az üzenetek frissességét ellenőrizni kell
- integritásvédelem: on-thy-fly módosítás lehetséges, ezért ellenőrzés szükséges
- hozzáférés védelem: csak legitim felhasználóknak, ellenőrzés nem csak csatlakozáskor, hanem bizonyos időközönként is
- zavarás elleni védelem

WEP –Wired Equivalent Privacy

IEEE 802.11 szabvány része. Célja: WiFi-t biztonságossá tenni (nem jött össze…)

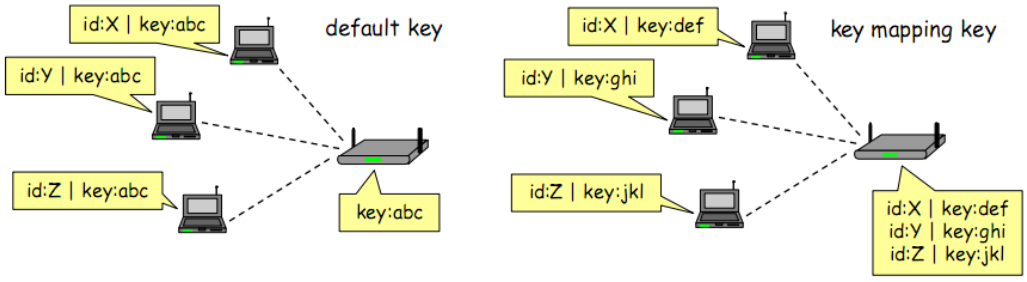
Hozzáférés védelem

Kapcsolódáskor STA-nak azonosítani kell magát AP felé. Ez egy egyszerű kihívás-válasz protokoll. Autentikáció után van lehetőség csatlakozási kérelemre.

Integritásvédelem és üzenethitelesítés

RC4 kulcsfolyam rejtjelezőt használ: minden üzenet más kulcsfolyammal van kódolva. Integritáshoz egy kódolt IntegrityCheckValue-t használ, ami együtt kerül titkosításra az üzenettel.
A titkosítás úgy történik, hogy az RC4 előállít egy pszeudo-random bitsorozatot (az IV és a kulcs alapján) majd ezt XOR-olja az ICV-vel kiegészített üzenettel.

WEP kulcsok



A gyakorlatban legtöbbször csak a default key támogatott, ennek hátulütője, hogy ugyanaz a kulcs van minden állomásnál, így egyik állomás dekódolhatja a másik üzenetét (biztonsági probléma)

Hibák

Hozzáférés védelem:

- egyirányú autentikáció (Man in the Middle attack veszély)
- autentikáció és titkosítás azonos kulcsot használ (ha az egyik feltörhető…)
- nincs session key autentikáció során (ha STA hitelesítése sikeres volt, utána támadó az ő MAC címével küldhet üzeneteket)
- STA megszemélyesíthető

Integritás védelem:

- nincs visszajátszás védelem egyáltalán (IV nem növekszik, vevő nem ellenőrzi frissességét)
- támadó képes módosítani az üzeneteket az ICV ellenére

Titkosítás:

- IV újrafelhasználás: IV mérete túl kicsi → egy 11Mbps sebességű router esetében 7 óra alatt újra használva lesz. Ezen felül általában nullára inicializálják és léptetik, így egymás melletti AP-k gyakran ugyan azt a szekvenciát használják (IV Collision attack)
- RC4 gyengeségeire már letölthető eszközök vannak

WPA

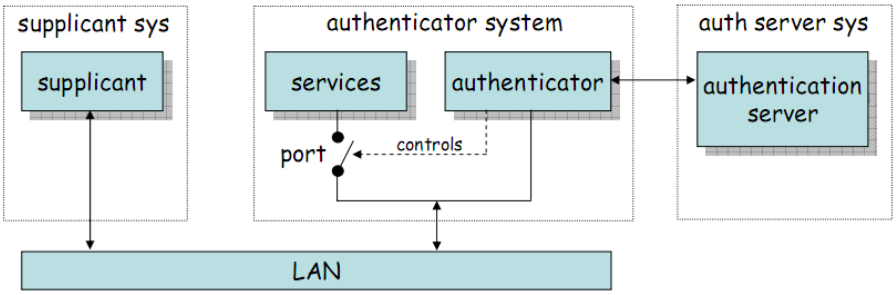
WiFi Protected Access

- integritás védelem: Michael
- IV visszajátszás számláláshoz is használva van
- továbbra is RC4 rejtjelezés, de kiküszöbölve WEP hibáit
- ronda megoldás, de szoftverfrissítés után megy régi hardware-n is

WPA2

- integritás védelem és kódolás AES alapon (CBC-MAC, CTR mód)
- szép megoldás, de új HW kell neki

802.1X autentikációs modell



- supplicant: ő kér hozzáférést a szolgáltatáshoz
- authenticator: felügyeli a portot, azaz ő szabályozza a hozzáférést
- auth server: hitelesíti a hozzáférést (supplicant hitelesíti magát a szervernek, ami ha sikeres, értesíti az authenticator-t, hogy engedélyezze a portot, valamint jelzi supplicant felé a sikerességet)

Ezt képezzük le WiFi-re:

- supplicant → STA
- authenticator → AP
- auth server → AP-n futó alkalmazás
- port → logikai állapot az alkalmazásban implementálva.

Sikeres azonosítás esetén nem csak a port kerül kinyitásra, hanem session key is lesz a mobil eszköz és a szerver között.

EAP: „hordozó” protokoll, az igazi autentikációs protokollok üzeneteinek (pl. TLS) Az AP nem tudja mi van az üzenetben, csak annyit, hogy sikeres-e vagy sem.
EAPOL: EAP on Lan
RADIUS: EAP üzenetek szállítására használják az AP és az auth server között.
EAP-TLS: csak TLS Handshake, titok generálása, autentikáció
EAP-TTLS: Tunneled EAP-TLS

Session key-ek:

- PTK (Pairwaise Transient Keys)
- GTK (Group transient Keys)

Hozzáférés védelem (8. ea.)

Erős fenntartásokkal kezelendő! Hiányos!

Linux biztonsági alapok

A Linux rendszerek DAC-t (Discretionary Access Control) használnak, vagyis az adott objektumhoz való hozzáférést az határozza meg, hogy ki a felhasználó, aki el akarja azt érni, vagy melyik felhasználói csoporthoz tartozik.

Linux alatt (majdnem) minden fájlként érhető el (fájlok, könyvtárak, driverek, perifériák, stb.) ezért nagyon fontos a fájlrendszer védelme. Az aktuális könyvtárban lévő fájlok részletes listázásához használhatjuk az *„ls -l”* parancsot:

```
$ ls -l
-rw-r----- 1 ramesh team-dev 9275204 Jun 13 15:27 mthesaur.txt.g
drwxrw---- 1 ramesh team-dev 2048    May 21 14:11 books
```

A fájlhoz tartozó jogosultságok:

- ’d’ jelzi ha könyvtárról van szó, ’-’ ha fájlról (és más lehetőségek is vannak)
- ezután jön a tulajdonosnak (felhasználó), a hozzá tartozó csoportnak és a többieknek a jogai.
- ’r’: a fájl olvasható/ könyvtár kilistázható.
- ’w’: a fájl írható, könyvtár esetén hozzáadhatunk vagy törölhetünk belőle fájlokat
- ’x’: a fájl végrehajtható (alkalmazás vagy shell program), könyvtár esetén beállítható aktuális könyvtárnak (working directory).

Ezután sorrendben: könyvtárak száma, tulajdonos, csoport, méret, dátum, fájl/könyvtárnév.

Sticky bit: Eredetileg memóriában tartásra használták (gyorsabban indul a program, ha már a memóriában van). Manapság a ragadós bitet inkább arra használják, hogy a könyvtárakban elhelyezkedő állományokat védjék vele. A beállított ragadós bittel rendelkező könyvtárban szereplő fájlokat ugyanis kizárólag a rendszergazda, illetve e könyvtár tulajdonosa törölheti vagy nevezheti át. Beállítása: `chmod +t`

SetUID/SetGID: futtatás olyan privilégiumokkal, mint ha tulajdonos/tulajdonos csoport tagja lenne (Ha egy könyvtáron group „s” van, akkor egy újfájlnak ugyan az lesz a tulajdonos csoportja, mint ami a könyvtáré)

Tehát: az r, w és x karaktereken kívül jogosultság esetén még általában s (S) ill. t (T) betűk láthatók, melyek jelentése attól is függ, mely felhasználói kategóriánál látod az adott karaktert. Az s/S-ek vagy a Set-UserID, vagy a Set-GroupID bitet jelentik, attól függően, hogy a tulaj, vagy a csoport x joga helyén látod, míg a t az un. Sticky ("ragadós") bit. Az, hogy kis- vagy nagybetű, az pedig attól függ, hogy az ugyanazon a helyen ábrázolt futtatási (x) bit létezik, vagy nem

- **-** akkor, ha sem az execute, sem a spéci bit nincs beállítva;
- **x** akkor, ha csak az execute van, a spéci bit nincs
- **s** (vagy t), ha execute **is** és spéci bit is
- **S** (vagy T), ha execute **nincs**, de a spéci bit igen.

Kernel space: Linux kernel és a betöltött modulok által foglalt memória
User space: többi processz által foglalt memória
Fontos ezek izolációja, kernel sosem swappolhat lemezre, és csak root kezelheti a modulokat

SetUID root sebezhetősége

Mindenképp root jogosultsággal fut. Alacsonyabb szintű felhasználóknak ad magasabb jogosultságot, azonban ha egy bug miatt feltörhető, akkor illetéktelenek is jogosultsághoz jutnak.

Rootkit-ek: lehetőséget adnak a támadónak az elrejtőzésre, rendszerfileokba épül. Ha sikeresen telepítették, utána észrevétlenül képes akár rendszerhívásokat is módosítani. Vannak ellene bizonyos eszközök (*chkrootkit*), de a legbiztosabb megoldás a rendszer újrahúzása.

Legalapvetőbb Unix parancsok

cat	fájl megjelenítése/összefűzése
cd	könyvtárváltás
ls	könyvtár tartalmának listázása
chgrp <group> <file>	fájl csoportjának megváltoztatása
chmod <right> <file>	fájl jogosultságok megváltoztatása
cp <source> <target>	másolás
mkdir	könyvtár létrehozása
touch	hozzáférés/módosítás dátumának frissítése
id	UIDvel tér vissza

umask

A létrejövő fájl jogosultságai adhatók meg vele.
Négyjegyű szám, ami beállítja, hogy a kinek milyen joga NINCS (jogosultság elvétele, chmod ellentettje). Első szám a setuid=4, setgid=2 (vagy 4+2=6), utána a szokásos u/g/o sorrendben, read=4, write=2, execute=1, illetve ezek összege.(pl: umask 0777 senkinek nincs semmihez joga (4+2+1=7)). Tipikus beállítás pl 0022, elvéve az írás jogot a csoporttól és másoktól.

Oktális umask: háromjegyű argumentum bitenkénti negálása után végzett AND művelet 666-al fájlok, míg 777-el könyvtárak esetében.

newgrp

Egyszerre csak egy aktív csoportja van a felhasználónak, de ezzel a paranccsal válthat. Az etc/passwd-ben van a default beállítva.

```
boldi@rivest:~$  id
uid=1000(boldi) gid=1000(boldi) groups=4(adm), 1000(boldi)
boldi@rivest:~$ newgrpadm
boldi@rivest:~$ id
uid=1000(boldi) gid=4(adm) groups=4(adm),1000(boldi)
boldi@rivest:/tmp/test$mkdira
boldi@rivest:/tmp/test$ls-la
total340
drwxrwxr-x3 boldiboldi4096 Oct26 13:20 .
drwxrwxrwt3 rootroot4096Oct26 13:20 ..
drwxrwxr-x2 boldi adm 4096 Oct26 13:20 a
```

cp

Másolás, azaz a jogosultságok megmaradnak. Ha törölnénk és újat hoznánk létre, akkor a jogosultságok és a tulajdonos más lesz.
Ha a munkakönyvtáron van setgid bit (s), akkor másoláskor ugyanúgy marad minden, míg törlés után könyvtártól öröklődnek a jogosultságok és a tulajdonos csoport is.

```
drwxrwsr-x2 boldiadm4096 Apr5 00:05 ./
drwxr-xr-x3 boldiusers4096 Apr5 00:04 ../
-rw-rw-rw-1 rootusers6 Apr5 00:11 b
-rw-r--r--1 boldiusers6 Apr5 00:04 c
boldi@shamir:~/access_control/f $ rmb
boldi@shamir:~/access_control/f $ cpc b
boldi@shamir:~/access_control/f $ ls-la
total16
drwxrwsr-x2 boldiadm4096 Apr5 00:11 ./
drwxr-xr-x3 boldiusers4096 Apr5 00:04 ../
-rw-r--r--1 boldiadm6 Apr5 00:11 b
-rw-r--r--1 boldiusers6 Apr5 00:04 c
```

Jogosultságok prioritása

A rendszer csak a tulajdonosi jogokat nézi, ha tulajdonosként hajtunk végre műveletet, így például ha nekünk nincs olvasási jogunk, de a csoportunknak van, akkor sem fogunk tudni olvasni. Ugyanez igaz a csoport és mások viszonyára is.

etc/passwd

Nincs a dián, de ZH-n jól jöhet.

1 Username: felhasználói név, 1-32 karakter hosszú.
2 Password: az x jelzi, hogy az /etc/shadow file-ban van tárolva
3 UID: user ID (UID). 0 a root, 1-99 egyéb előre definiált.
4 GID: az elsődleges group ID (tárolva az /etc/group file-ban)
5 User ID Info: Komment a felhasználóról, pl teljes név, stb..
6 Home directory: bejelentkezés után ez lesz a felhasználónak a working directory
7 Command/shell

etc/goups

Sorrendben:
1 Group name: csoport neve
2 Password: x, titkosítva
3 Group ID (GID): Group ID
4 Group List: a csoport tagjainak felsorolása, vesszővel elválasztva

Hozzáférés védelem

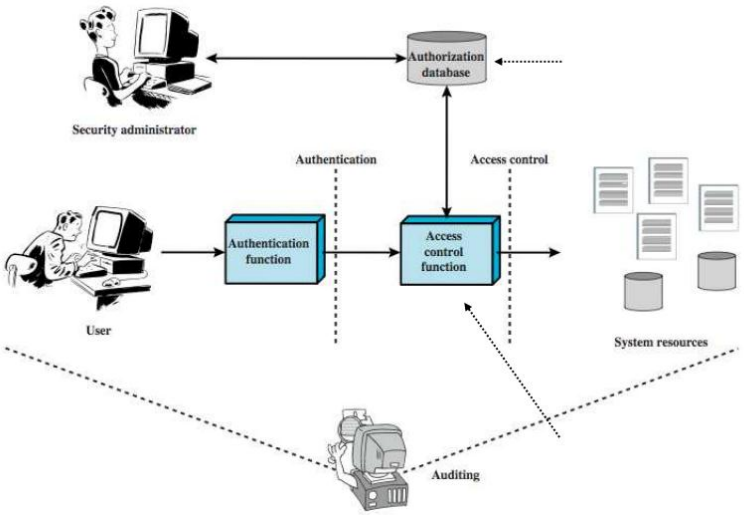
Erőforrások védelme a jogosulatlan felhasználástól.
policy: definiálja ki, mihez, hogyan és milyen körülmények között férhet hozzá
enforcement: olyan összetevők, melyek garantálják a policy-knak megfelelő működést (pl. tűzfal)

Típusok:

Discretionary access control (DAC): a felhasználó privilégiumán alapuló hozzáférés

Mandatory access control (MAC): biztonsági címkék és engedélyek összehasonlításán alapul

Role-based access control (RBAC): szerep alapú hozzáférés



AC elemei:

- subject: entitás, ami object-hez férhet hozzá (pl user, csoport)
- object: hozzáférés védelem alatt álló erőforrás
- access right: jogosultság fentiek között

Discretionary Access Control

Gyakran ábrázolják hozzáférési mátrixszal. Sorok: subject, oszlopok: object
Másik gyakori ábrázolás az autorizációs tábla, ami (subject,object,access right) hármasokból áll.

Modern rendszerek támogatják az ACL-eket (Access Control List). Object-ek alapján rendezett lista az autorizációs táblából.
Amikor hozzáférés ellenőrizendő: leginkább releváns AC kiválasztása, és azon a jogosultság ellenőrzése.
Fontos kérdés: multiple AC esetén az összes rendelkezzen privilégiummal, vagy valamely rendelkezzen?

Role-based access control

Hierarchikusan rendezhető szerepek, öröklődés értelmet nyer. (akár többszörös öröklés is támogatott lehet)

Ezen felül bevezetésre kerültek a kényszerek is. (kölcsönös kizárás, kardinalitás, szerep-előfeltétel)