Kriptográfia

2010. január 3.

Tartalomjegyzék

Előszó

Informatizálódó társadalmunkban egyre fontosabb szerepet kap az adatcsere biztonságának szavatolása. Ezt részben az adatok titkosításával oldhatjuk meg.

A kriptográfia klasszikus értelemben a titkosítás tudománya, titkosítási rendszerek (kriptorendszerek) felépítésével és biztonsági elemzésével (kriptoanalízisével) foglalkozik. Modern értelemben azonban a kriptográfia fogalma magába foglal olyan aktuális témaköröket is, mint a digitális aláírások, hitelesítések, hash függvények stb.

A kriptográfia fontosságát talán az mutatja a leginkább, hogy napjainkban számos vezető informatikai cég újabb és biztonságosabb titkosítási rendszerek tervezésével és ezek lehetséges alkalmazásaival foglalkozik. Ez pedig egyértelműen hozzájárul a modern kriptográfia robbanásszerű fejlődéséhez.

Jelen jegyzet a Babeş–Bolyai Tudományegyetem *Komputacionális matematika* és *Informatikai modellek optimizálása* mesteri képzései *Kriptográfia* előadásának és szemináriumának anyagát tartalmazza. Célja egyrészt a különfajta titkosítási rendszerek felépítésének bemutatása, valamint ezek matematikai hátterének és biztonságának elemzése, másrészt pedig az olyan újabb kriptográfiai témakörök tárgyalása, mint a digitális aláírások, hitelesítések, hash függvények és egyéb kriptográfiai protokollok.

A jegyzet a kriptográfiai alapfogalmak bemutatásával kezdődik, majd a főbb szimmetrikus kulcsú kriptorendszereket tárgyalja a klasszikus rejtjelektől az olyan modern rendszerekig, mint a DES vagy az AES. Ezután az ismertebb aszimmetrikus (nyilvános) kulcsú rendszerek következnek, mint a Merkle–Hellman-rendszer, az RSA és a diszkrét logaritmáláson alapuló rendszerek. A hash függvények fejezetet a digitális aláírások részletezése követi. Végül szó esik az SSL protokollról is. A jegyzetet négy függelék egészíti ki, melyekben a szükséges fontosabb algebrai és bonyolultságelméleti fogalmak mellett egy magyar–angol fogalomtár is bemutatásra kerül.

Hangsúlyozandó, hogy a romániai magyar szakirodalomban ez a könyv egyike az első ilyen témájú egyetemi jegyzeteknek. Ilyen szempontból igazi hiánypótló munka.

Érthetősége és tudományos megalapozottága miatt a könyvet haszonnal forgathatják a felsőbb éves egyetemi hallagtók, a líceumi matematikatanárok és érdeklődő diákok egyaránt.

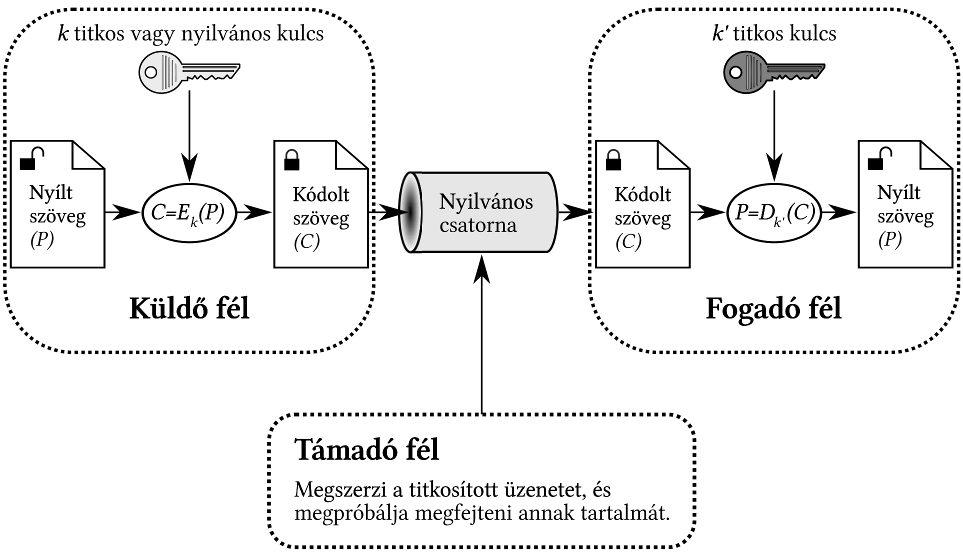
A szerzők köszönetüket fejezik ki Váradi Nagy Pálnak, Ferencz Csaba-Leventének, Nagy László Tibornak valamint a szaklektoroknak a kézirat javításában nyújtott segítségükért, illetve a KMEI-nek az anyagi támogatásért.

1. fejezet

Kriptográfiai alapfogalmak

Ebben a fejezetben általános képet szeretnénk nyújtani a kriptográfiáról, illetve bevezetünk néhány kriptográfiai alapfogalmat.

Egy általános *titkosítási rendszer* az alábbi sémával vázolható:



A séma szerint a küldő fél az *eredeti P üzenet*et a *k kulcs*tól függő E_k *titkosítási eljárás*sal *kódol*ja (*titkosít*ja vagy *rejtjelez*i). Ismert és a magyar szaknyelvben használatos szó a francia eredetű „sifrírozás” is, valamint a származékai (pl. „desrifrírozás”), könyvünkben azonban csak a „titkosít”, „rejtjelez”, „kódol” és „dekódol” kifejezéseket fogjuk használni. A kódolás előtti *eredeti üzenet*et még *nyílt üzenet*nek vagy *nyílt szöveg*nek is nevezzük, a titkosítási eljárást (módszert) pedig *kriptorendszer*nek, *kód*nak vagy néha *rejtjel*nek. A titkosítási eljárással a küldő fél előállítja a $C = E_k(P)$

titkosított (*kódolt*, *rejtjelezett*) *szöveg*et. Ezt egy *nyilvános csatorná*n a fogadó félnek továbbítja, aki a titkosított C üzenetet a k' kulcstól függő $D_{k'}$ *dekódoló eljárás*sal megfejti (*dekódol*ja), visszakapva az eredeti P üzenetet. Nyilvánvalóan az E_k, $D_{k'}$ függvények egymás inverzei kell, hogy legyenek, hiszen $D_{k'}(C) = D_{k'}(E_k(P)) = P$, és fordítva is igaz, $E_k(P) = E_k(D_{k'}(C)) = C$. Ugyanakkor fontos, hogy ezek az eljárások polinomiális (rövid) idő alatt fussanak (lásd az I. függeléket). A képbe bekerül még egy harmadik szereplő is, a támadó fél, aki lehallgatja a nyilvános csatornán továbbított üzenetet, és megpróbálja megfejteni annak tartalmát. Kriptográfiai körökben hagyományosan névvel illetik ezeket a szereplőket: a küldő fél Alice, a címzett Bob, a támadó pedig Eve vagy Marvin.

Auguste Kerckhoffs (1835–1903) holland nyelvész és kriptográfus 1883-ban írt egy értekezést a katonai kriptográfiáról. Ebben megkülönbözteti a katonai titkosítási rendszereket a magánszemélyek által használt módszerektől. Feltételezzük, hogy két személy – Alice és Bob – valamilyen titkosírást használnak levelezésükben. Marvin, aki mondjuk féltékeny Alice-re, mindenképpen meg szeretné fejteni a levelek tartalmát. Az Alice és Bob által használt titkosítási módszer lehet közismert, de lehet egy általuk kitalált titkos eljárás is. Tehát Marvin kódfeltörő dolgát nagy mértékben megnehezíti az is, hogy azt sem tudja, milyen titkosítási rendszerrel áll szemben. Alice és Bob még azt is könnyen megtehetik, hogy váltogatják a használt módszert, lehetőségeiknek csak leleményességük szab határt. Nem ilyen egyszerű a helyzet például a hadseregnél – írja Kerckhoffs. A parancsnokoknak nincs lehetőségük arra, hogy kényükre-kedvükre cserélgethessék a titkosítási módszert, és vigyázniuk kell arra, hogy egyáltalán ne használjanak olyan tárgyat vagy írást, amely ha az ellenség kezére kerül, veszélyeztetheti a titkos kommunikációt, az ellenség tudomására juttatva katonai titkokat. Ezért a hadseregnél használt titkosítási módszerek meg kell, hogy feleljenek a következő hat követelménynek:

1. Ha elméletileg nem is, a rendszernek gyakorlatilag feltörhetetlennek kell lennie.

2. A módszerrel titkosított üzenetek biztonsága ne függjön magának a módszernek a titkosságától: a módszer leírását ugyanis az ellenség is megszerezheti.

3. A kulcs könnyen megjegyezhető, továbbítható és változtatható kell, hogy legyen (anélkül, hogy azt papírra kellene írni).

4. A kódolt üzenetnek olyan formája legyen, hogy azt táviraton továbbítani lehessen.

5. A rendszer által használt segédeszközök hordozhatóak kell, hogy legyenek, és a kódolás/dekódolás műveletét egyetlen személynek is el kell tudnia végezni.

6. A módszer használata legyen egyszerű (ne kelljen túl sok szabályt, lépést megjegyezni), szellemileg ne terhelje túl használóját.

Ez a hat követelmény az úgynevezett Kerckhoffs-elv. Kevés módosítással ugyan (például: a kódolt üzenetnek olyan formája legyen, hogy azt interneten továbbítani lehessen), de ezeket az alapelveket követik a napjainkban használt titkosítási módszereknél is.

Látható tehát, hogy milyen fontos a kulcsok titkossága, hiszen, mivel az E kódolási és D dekódolási eljárások nyilvánosak, a kommunikáció titkossága csupán a sokféleképpen megválasztható k és k' kulcsok titkosságától függ. A titkosítási módszerek nyilvánossága miatt nagy számú felhasználó használhatja ugyanazt a kriptorendszert, csupán a kulcsok változtatásával. Itt fontos rögtön megjegyezni, hogy a kulcsok titkossága alapján a titkosítási rendszereket két családba soroljuk: *titkos kulcsú (vagy szimmetrikus kulcsú) kriptorendszer*ek és *nyilvános kulcsú (vagy aszimmetrikus kulcsú) kriptorendszer*ek.

Egy kriptorendszer titkos kulcsú (szimmetrikus kulcsú) ha k és k' is titkos. Ebben az esetben $k \approx k'$, vagyis a két kulcs egyenlő, vagy ritkábban az egyik kulcsból polinomiális (rövid) idő alatt származtatható a másik kulcs. A következő fejezetben számos példáját látjuk majd szimmetrikus titkosítási módszereknek, a gyors szemléltetés kedvéért legyen azonban az egyik legegyszerűbb szimmetrikus kriptorenszer: a Caesar-kód.

1.0.1. példa. *A Caesar-kód minden egyes betűt az ábécében egy tőle meghatározott távolságra levő betűvel helyettesít. Formálisan (az előbbi jelölésünket felhasználva): P egy betű, $k = k'$ az eltolás mértéke (titkos), $C = E_k(P)$ az a betű, amely P-től számolva (körkörösen, jobbra haladva) pontosan k betű távolságra van az ábécében, a $D_{k'}(C)$ nem más, mint $E_k(P)$ fordítottja, vagyis az eredeti betűt úgy kapjuk vissza, hogy C-től számolva, körkörösen balra haladunk az ábécében, és kiemeljük a k-adik betűt.*

Egy titkosítási módszer nyilvános kulcsú, ha k nyilvános és k' titkos. Ebben az esetben nyilván $k \not\approx k'$, tehát k ismerete nem vezethet könnyen k' ismeretéhez. Nyilvános kulcsú rendszereket az 1970-es évek óta használnak. Az egyik legismertebb és legelterjedtebb ilyen rendszer az RSA.

1.0.2. példa. *Ugyancsak a már bevezetett jelöléseket használva legyen a titkosítandó üzenet (P) egy betű, a nyilvános k kulcs egy telefonkönyv (amely a nevekhez hozzárendeli a telefonszámokat), a titkos k' kulcs pedig a „fordított” telefonkönyv (amely a telefonszámokhoz rendeli hozzá a neveket). A kódolás abból áll, hogy a P betűt helyettesítjük $C = E_k(P)$-vel, egy P-vel kezdődő név telefonszámával a k telefonkönyvből. A dekódolás természetesen ennek a műveletnek a fordítottja, $P = D_{k'}(C)$ a C-számnak megfelelő név kezdőbetűje a k' telefonkönyvből.*

A támadó fél (Marvin) feladata az úgynevezett *kriptoanalízis*, vagyis a titkosítási rendszer feltörése. A Kerckhoffs-elvnek megfelelően feltételezzük, hogy Marvin ismeri az E, D eljárásokat (de nem ismeri a titkos kulcsokat), és a nyilvános továbbítási csatornán mindig képes megszerezni a kódolt üzeneteket (C-ket). Marvin támadási lehetőségeit a következő módon lehet osztályozni:

1. Csak a kódolt üzenet ismeretén alapuló támadás: Marvinnak csak a kódolt üzenetekhez van hozzáférése, semmit sem tud az eredeti üzenetek tartalmáról vagy a titkosításhoz használt kulcsról.

2. Nyílt szöveg ismeretén alapuló támadás: Marvin ismer bizonyos (P,C) párokat, vagyis ismer bizonyos nyílt üzeneteket a megfelelő kódolt üzenetekkel együtt, ezek az információk viszont adottak (például egy kém vagy egy titkos ügynök szerzi be neki). Nem ismeri még a titkosításukhoz használt kulcsot, ezért ő maga nem tud ilyen párokat előállítani.

3. Választható nyílt szövegen alapuló támadás: Marvin akármilyen általa választott nyílt üzenetet tud titkosítani (vagy titkosíttatni) Alice módszerével (tehát gyakorlatilag ismeri és használja E_k-t, így ő maga tudja előállítani a (P,C) párokat). Az ilyen típusú támadásoknak a nyilvános kulcsú rendszereknél van nagy jelentősége, ahol a titkosításhoz használt k kulcs nyilvános, tehát a támadó akármilyen nyílt szöveget kódolhat vele.

4. Választható titkosított üzeneten alapuló támadás: Marvin akármilyen általa választott titkosított üzenetnek megkaphatja a dekódolt változatát (tehát gyakorlatilag ismeri és használja $D_{k'}$-t) anélkül azonban, hogy a k' kulcsot ismerné.

Bármely kriptorendszer titkos kulcsait próbálgatással el lehet elvileg találni. Ehhez azonban (a legrosszabb esetben) szisztematikusan az összes lehetséges kulcsot végig kell próbálni. Az ilyen jellegű támadás a „nyers erő" alkalmazásához hasonlatos (ezért angolul a módszer neve: „brute force attack") – magyarul *kimerítő kulcskeresés*nek nevezzük. A kimerítő kulcskeresésnek a számítógépek megjelenésével óriási mértékben megnőtt a jelentősége. A klasszikus („történelmi") kriptorendszerek kitalálóinak nem kellett még számolniuk azzal, hogy a támadó fél több ezer, több millió vagy még annál is több kulcsot kipróbálhat. Ezért ha a lehetséges kulcsok száma elegendően nagy volt ahhoz, hogy „kézzel" képtelenség legyen minden lehetőséget kipróbálni, a kriptorendszert már biztonságosnak tekintették.

2. fejezet

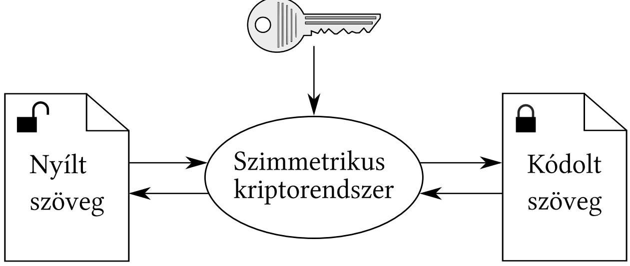
Szimmetrikus kulcsú titkosítás

A *szimmetrikus kulcsú titkosítás*i algoritmusok (vagy *szimmetrikus kulcsú kriptorendszer*ek) közös jellemzője, hogy a kódolásra és dekódolásra gyakorlatilag ugyanazt a titkos kulcsot használják, amit mind a küldő fél, mind a fogadó fél ismer. Ehhez azonban még a kommunikáció megkezdése előtt meg kell állapodniuk egy közös kulcsban, és azt titokban kell tartaniuk. Formálisan: a küldő fél az eredeti P üzenetet a k kulcstól függő E_k titkosítási eljárással kódolja, megkapva a $C = E_k(P)$ titkosított üzenetet. A fogadó fél megkapja a C kódolt üzenetet, és ugyanazt a k kulcsot használva dekódolja a $D_k = E_k^{-1}$ eljárással, így olvashatóvá válik számára a $P = D_k(C) = E_k^{-1}(E_k(P))$ üzenet. Az általános szimmetrikus kulcsú titkosítás sémáját a 2.1 ábrán láthatjuk.

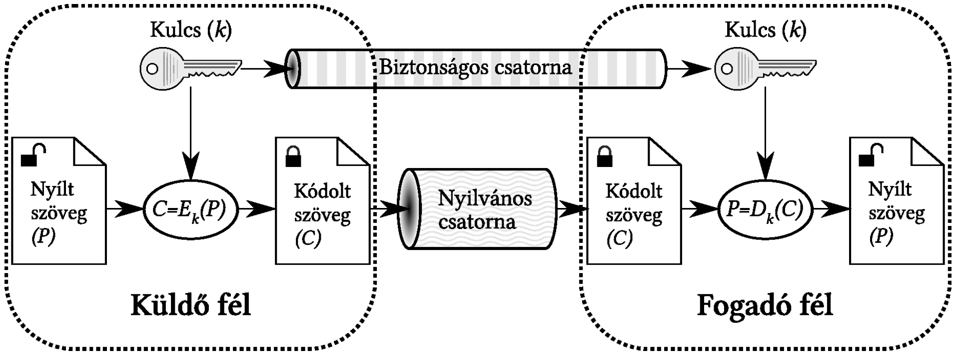
Az 1970-es évekig minden használatban levő titkosítási eljárás szimmetrikus kulcsú volt. A klasszikus kriptorendszerek (Caesar-kód, Vigenère-rejtjel, Playfair-kód stb.) mindegyike szimmetrikus, de szimmetrikus kulcsú még számos modern kriptorendszer is (például a DES és az IDEA), az AES-t – a jelenlegi standard kriptorendszert – beleértve. Ezenkívül a szimmetrikus rendszerek családjába tartozik az egyetlen bizonyítottan biztonságos titkosítási eljárás, a véletlen átkulcsolás is.

A szimmetrikus kulcsú rendszerek előnyei közé sorolható a hatékonyság és egy-

2.1. ábra. Szimmetrikus kulcsú titkosítás



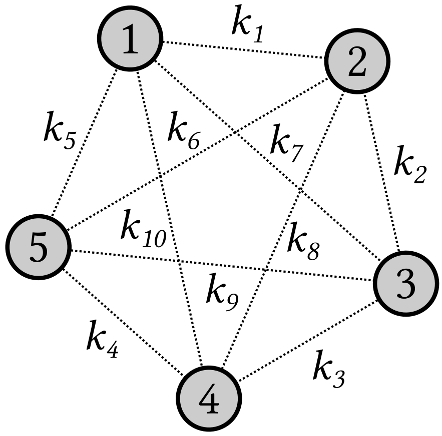
2.2. ábra. (Egyirányú) kulcscsere a szimmetrikus kulcsú rendszerek esetében



szerű kezelhetőség. Általában jóval gyorsabbak, mint a következő fejezetben bemutatott aszimmetrikus kulcsú titkosítási eljárások (a gyakorlatban a DES például körülbelül ezerszer gyorsabb, mint az aszimmetrikus RSA). Sok szimmetrikus kulcsú eljárás (a klasszikusok közül) ugyanakkor számítógép nélkül is használható – ezek kiválóan megfelelnek rövid (és nem életbevágóan titkos) üzenetek kódolására. Nagy hátrányuk viszont a *kulcscsere* problémája. A küldő és a fogadó félnek sokszor nem áll módjában személyesen találkozni, hogy biztonságos keretek között megegyezhessenek a közös kulcsban. A küldő ezért először el kell juttassa kommunikációs partnerének a k kulcsot, vigyázva arra, nehogy illetéktelenek kezébe kerüljön: a kulcs cseréje tehát egy biztonságos csatornán keresztül történik. Ez sokszor nagyon költséges és kockázatos lehet. Miután mindkét fél birtokában van a kulcs, a további titkos üzenetcsere nyilvános csatornán keresztül folyik. Ezt a folyamatot szemlélteti a 2.2 ábra. A biztonságos csatornát azért nem tudják az üzenetek továbbítására használni, mert mondjuk túl költséges, csak kis mennyiségű adat átvitelére képes, vagy túl lassú (biztonságos csatorna lehet például egy heti egy alkalommal útra kelő titkos ügynök, nyilvános csatorna pedig például a távirat vagy a telefon). A kulcs cseréjével kapcsolatosan még egy másik probléma is felmerül: n egyén közti titkosított kommunikáció lebonyolításához összesen $\frac{n(n-1)}{2}$ kulcs szükséges (egy 5 csomópontot tartalmazó hálózatban is már $\frac{5\cdot4}{2} = 10$ kulcsra van szükség – 2.3 ábra). Ha a kulcsok előállítása és célba juttatása költséges, ez nagy többletkiadáshoz vezet, ezenkívül a nagyszámú kulcscsere miatt számottevően megnő annak a valószínűsége is, hogy a támadó fél kezére jut egy vagy több kulcs.

A szimmetrikus kulcsú eljárások egyeduralmának W. Diffie és M. E. Hellman 1976-os cikke vetett véget, amiben bevezetik az aszimmetrikus kulcsú titkosítás fogalmát. Az aszimmetrikus kulcsú rendszerek elegáns megoldást kínálnak a szimmetrikus eljárások hiányosságaira, amint azt a következő fejezetben látni fogjuk. Ez nem jelenti azonban, hogy az aszimmetrikus rendszerek kiszorították volna a szimmetrikusokat a mindennapi használatból. A kétféle rendszer kiegészíti egymást a nagyobb

2.3. ábra. 5 egyén közti kommunikációhoz 10 kulcs szükséges



biztonság és hatékonyság érdekében (aszimmetrikus kulcsú rendszerrel lehet például a szimmetrikushoz használt kulcsokat titkosítani, majd az üzeneteket a szimmetrikus rendszerrel kódolják, a hatékonysága miatt).

A következőkben bemutatjuk a szimmetrikus kulcsú titkosítási módszerek két családját: a napjainkban már csak történelmi és didaktikai szempontból érdekes klasszikus titkosítási rendszereket, illetve a gyakorlatban használt modern titkosítási rendszereket.

2.1. Klasszikus titkosítási rendszerek

*Klasszikus titkosítási rendszer*nek tekintünk minden olyan egyszerű rejtjelező módszert, amit a történelem folyamán használtak, a legrégebbi időktől a XX. század második feléig. Ezek legtöbbjének alkalmazása papírt, ceruzát, figyelmet, türelmet, kitartást és (elég) sok időt igényelt. De ide soroljuk még az első és második világháború idején használt rejtjelező gépeket is (azzal együtt, hogy ezek már sokkal fejlettebb módszerek voltak, hiszen a kódolást/dekódolást már nem kézzel végezték, hanem mechanikus vagy elektro-mechanikus gépekkel).

A klasszikus rendszereknek manapság természetesen már csak elenyésző a gyakorlati haszna, hiszen olyan időkben tervezték őket, amikor még nem kellett számolni a modern informatikai rendszerek által nyújtott számítási kapacitással, és a matematikai háttér is jóval szerényebb volt a mainál. Tanulmányozásuk ennek ellenére elengedhetetlen, hiszen egyrészt egyszerű és hozzáférhető módon lehet szemléltetni általuk a kriptográfiai alapfogalmakat, másrészt struktúrájuknak és gyönge pontjaiknak tanulmányozása révén érthetővé válnak a modern kriptográfiai és biztonsági rendszerek tervezésénél figyelembe vett kritériumok.

Ezeket a rendszereket – annak alapján, hogy milyen módszerrel feleltették meg az eredeti üzenet betűinek a kódolt szöveg betűit – a következő osztályokba sorolhatjuk: *helyettesítő kód*ok, *átrendezéses (keveréses) kód*ok és *helyettesítő-átrendező (hibrid) kód*ok.

A helyettesítő kódok (vagy helyettesítő rejtjelek) az eredeti szövegegységeket egy szabályos rendszer alapján alakítják át rejtjelezett szövegegységekké. Ezek az egységek lehetnek a használt ábécé betűi (a leggyakoribb eset), betűpárok vagy több betűt magukba foglaló csoportok. A lényeg az, hogy egyazon szövegegységnek mindig ugyanaz a kódolt egység felel meg, függetlenül attól, hogy például hol helyezkedik el a szövegben.

Itt megjegyezzük, hogy az eredeti és a titkosított szöveg ábécéi eltérhetnek egymástól. Például, ha a magyar ábécé minden egyes betűjének meghatározott kínai írásjeleket feleltetünk meg, akkor egy monoalfabetikus helyettesítési kódot gyártottunk.

A helyettesítő rejtjelekkel ellentétben az átrendezéses kódok az eredeti szöveg egységeit összekeverik (nem ritkán igen bonyolult módszerrel), de maguk a szövegegységek érintetlenül maradnak, vagyis a kódolt szöveg az eredeti szövegnek egy permutációja lesz.

A helyettesítő és átrendezéses módszereket természetesen ötvözni is lehet, elképzelhető számos olyan módszer, mely például először bizonyos jelekkel helyettesíti az eredeti üzenet betűit, majd ezeket egy szabály szerint permutálja.

A következőkben bemutatunk néhány fontos klasszikus titkosítási rendszert (a teljesség igénye nélkül), az egyszerűbbektől haladva a bonyolultabbak felé.

2.1.1. A Caesar-kód és variációi

A *Caesar-kód* (vagy *Caesar-rejtjel*) egyike a legegyszerűbb és legismertebb titkosítási módszereknek. A helyettesítő rejtjelek családjába tartozik. Julius Caesar (Kr. e. 100 - Kr. e. 44) római császárról kapta nevét, aki előszeretettel és gyakran folyamodott a titkosíráshoz.

2.1.1.1. A Caesar-kód

Legyen egy n-betűs ábécé (angol ábécé esetében $n = 26$). A betűket 0-tól számított ábécébeli sorszámuk alapján azonosíthatjuk $\mathbb{Z}_n$-beli elemekkel.

Kulcs: $k \in \mathbb{Z}_n$.

Kódolás: $C = E_k(P) = (P + k) \bmod n$, ahol $P \in \mathbb{Z}_n$ (egy betű).

Dekódolás: $P = D_k(C) = (C - k) \bmod n$, ahol $C \in \mathbb{Z}_n$ (egy betű).

Látható, hogy a betűket k pozícióval toljuk ciklikusan jobbra az ábécében, ezért ezt a módszert még *Caesar eltolásos ábécéjé*nek is szokták nevezni.

2.1. táblázat. A magyar és angol nyelv öt leggyakoribb betűje

Gyakoriság	**Magyar**	**Angol**
1.	E	E
2.	A	T
3.	T	A
4.	O	O
5.	L/N	I/N

2.1.1. példa. *Tekintsük a* 26 *betűs angol ábécét, melynek sorszámozása a következő:*

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Legyen $k = 13$. *Ekkor:*

$$\begin{aligned}
E_k(\mathbf{T}) = E_k(19) &= (19+13) \bmod 26 = 6 = \mathbf{G}\\
E_k(\mathbf{I}) = E_k(8) &= (8+13) \bmod 26 = 21 = \mathbf{V}\\
E_k(\mathbf{T}) = E_k(19) &= (19+13) \bmod 26 = 6 = \mathbf{G}\\
E_k(\mathbf{O}) = E_k(14) &= (14+13) \bmod 26 = 1 = \mathbf{B}\\
E_k(\mathbf{K}) = E_k(10) &= (10+13) \bmod 26 = 23 = \mathbf{X}
\end{aligned}$$

Tehát a **TITOK** *szó titkosítva* **GVGBX** *lesz.*

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadások)

1. Mivel a lehetséges kulcsok száma kicsi (n), kipróbálhatjuk az összes lehetséges kulcsot (ezt még a *kimerítő kulcskeresés* módszerének is nevezzük). Hogyan ismerjük fel a jó kulcsot, feltéve, hogy az eredeti szöveg értelmes? Elkészítünk az illető nyelv gyakori szavaiból egy listát. Az a kulcs lesz jó, mellyel a titkosított szöveget dekódolva a legtöbb listabeli szó beazonosítható.

2. *Betűgyakoriság-vizsgálat.* Minden nyelvben léteznek úgynevezett betűgyakoriság-táblázatok, amelyek egy-egy adott betű terjedelmes szövegben való előfordulásának százalékos arányát adják meg. A magyar és az angol nyelv öt leggyakoribb betűjét az 2.1 táblázatban tüntettük fel. Mivel a Caesar-kód *monoalfabetikus* – vagyis egy adott betű, bárhol is van a szövegben, ugyanabba a betűbe kódolódik – elég megkeresni a leggyakoribb betűt a kódolt szövegben és az fog megfelelni az adott nyelv leggyakoribb (vagy esetleg második, harmadik leggyakoribb) betűjének. A kulcs így egyszerűen megkapható. Ki kell hangsúlyozni, hogy a betűgyakoriság-vizsgálat csak

2.2. táblázat. Az angol nyelv betűgyakoriság-táblázata

Betű	Gyakoriság (%)	Betű	Gyakoriság (%)
A	8,167	N	6,749
B	1,492	O	7,507
C	2,782	P	1,929
D	4,253	Q	0,095
E	12,702	R	5,987
F	2,228	S	6,327
G	2,015	T	9,056
H	6,094	U	2,758
I	6,966	V	0,978
J	0,153	W	2,360
K	0,772	X	0,150
L	4,025	Y	1,974
M	2,406	Z	0,074

terjedelmes és értelmes szöveg esetén lesz hatékony. Az ugyanarra a nyelvre készített betűgyakoriság-táblázatok között persze lehetnek eltérések, amik abból adódnak, hogy bizonyos típusú szövegeknél (pl. irodalmi, tudományos stb.) más-más szavakat használ az adott nyelv. A 2.2 táblázat tartalmazza az angol nyelv 26 betűjének százalékos gyakoriságát [5]. A 2.4 ábra a gyakoriságtáblázat adatait szemlélteti, itt a betűket csökkenő sorrendbe is helyeztük, előfordulási gyakoriságuk szerint.

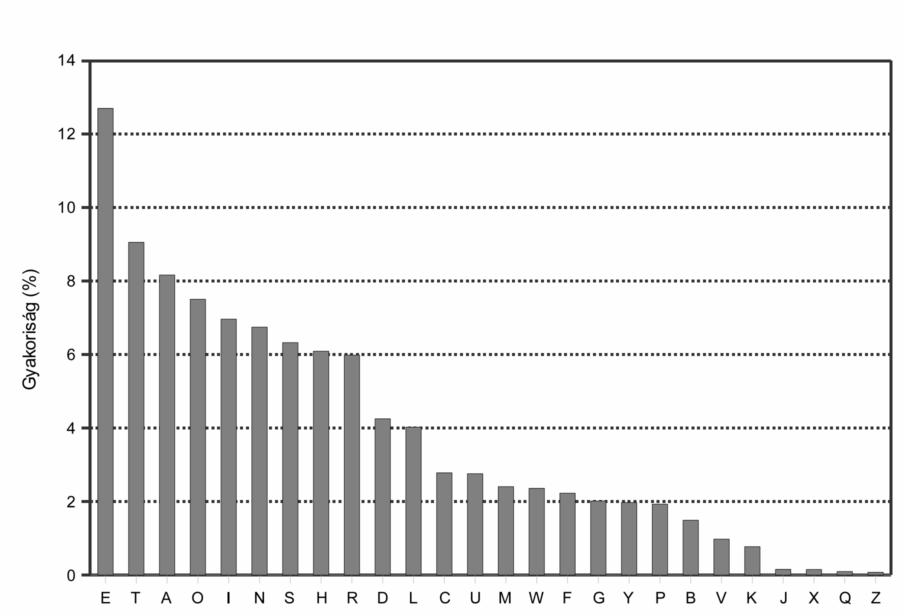
A Caesar-rejtjel használatát praktikussá teszi és felgyorsítja az úgynevezett *Caesar-kerék* (vagy *Caesar-tárcsa*). Ez két egymáshoz viszonyítva elforgatható koncentrikus körlapból áll, a körlapok szélén körkörösen a használt ábécé van feltüntetve. Ha tudjuk az eltolás mértékét (a k-t) akkor elég kellő mértékben elforgatni a körlapokat, máris megvan a könnyen használható megfeleltetési „táblázat”.

A fennmaradt írásos források szerint Caesar a $k = 3$ kulcsot használta, úgy amint azt a 2.5 ábra mutatja. Ezek szerint Caesar híres mondata („Veni, vidi, vici!”) az általa használt kulccsal a kövekező módon titkosítható: Yhql, ylgl, ylfl!

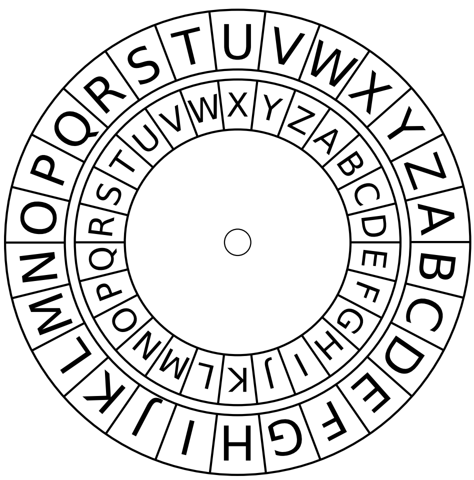
2.1.1.2. A kulcsszavas Caesar-kód

Láthattuk, hogy a Caesar-kód alapja egy betűeltolás, tehát a megfeleltetés függvénye egy partikuláris (és igen egyszerű) betűpermutáció, mely egyetlen paraméter (az eltolás mértéke) ismeretében felírható. A kulcsszavas Caesar-kód alapja egy bonyolultabb betűpermutáció, mely egy $\ell \in \mathbb{Z}_n$-beli elemből és egy szóból álló kulcs alapján szerkeszthető meg a következő módon: a *megfeleltetési táblázat* alsó sorába az ℓ-edik sorszámtól kezdődően beírjuk a szót (az esetleges ismétlődő betűket kihagyva), majd

2.4. ábra. Az angol nyelv betűinek százalékos gyakorisága



2.5. ábra. Caesar-kerék (a kulcs $k = 3$)



az ábécé többi betűjét a kulcsszó után írjuk ciklikusan. A kódolás és a dekódolás betűnként történik a permutációnak megfelelően. Vegyük észre, hogy a kulcsszavas Caesar-rejtjelnél elvileg bármely *betűpermutáció* bejöhet. Egy ilyen megfeleltetési táblázatot (betűpermutációt) láthatunk a következő példában:

2.1.2. példa. *Legyen* $n = 26$, $\ell = 8$ *és a kulcsszó pedig* **CAESAR**. *Ekkor a megfelelő betűpermutáció*

$$\begin{array}{c} \begin{array}{cccccccccccccccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \mathbf{8} & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \end{array} \\ \begin{pmatrix} A & B & C & D & E & F & G & H & \mathbf{I} & J & K & L & M & N & O & P & Q & R & S & T & U & V & W & X & Y & Z \\ Q & T & U & V & W & X & Y & Z & \mathbf{C} & \mathbf{A} & \mathbf{E} & \mathbf{S} & \mathbf{R} & B & D & F & G & H & I & J & K & L & M & N & O & P \end{pmatrix} \end{array}$$

Tehát a **TITOK** *szó rejtjelezett formája* **JCJDE** *lesz.*

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadások)

1. Kimerítő kulcskeresés esetében elvileg minden lehetséges betűpermutációt ki kellene próbálni, tehát összesen $n!$ lehetőséget (hiszen a kulcsszó lehet egy n hosszúságú értelmetlen szó is). Hogyan ismerjük fel a jó kulcsnak megfelelő betűpermutációt? Egyrészt használhatjuk a Caesar-kódnál ismertetett szólistás módszert, de sokkal hatékonyabb és gyorsabb a következő, úgynevezett *legkisebb négyzetösszeg módszer*: legyen C a kódolt szöveg, $D_k(C)$ a k kulccsal dekódolt szöveg, $f(a)$ az a betű százalékos gyakorisága a nyelvben, $f_{D_k(C)}(a)$ pedig az a betű százalékos gyakorisága a k kulccsal dekódolt szövegben. Jelölje továbbá A a használt ábécé betűinek halmazát. A jó kulcs k, ha $\sum_{a\in A}(f(a) - f_{D_k(C)}(a))^2$ minimális. Vegyük észre, hogy ez a módszer minden helyettesítő kriptorendszer esetén működik (ha a-val egy akármilyen szövegegységet jelölünk). Persze ezt is csak értelmes és terjedelmes szövegek esetében célszerű használni.

2. Betűgyakoriság-vizsgálat segítségével be lehet azonosítani a leggyakoribb betűk képeit a permutációban, a többi megfeleltetést próbálgatással kaphatjuk meg.

3. *Betűtávolságok mérése*. Ez a módszer igen hasznos lehet rövid kulcsszavak esetében, amikor a megfeleltetési táblázat alsó sorának kulcsszón kívüli része ciklikusan ábécésorrendben van. A módszert párosítani kell betűgyakoriság-vizsgálattal. A következő példában illusztráljuk:

2.1.3. példa. *Angol ábécét használunk a következő kulcsszavas Caesar-kódnál. Tudjuk, hogy a kulcsszó négy különböző betűből áll és értelmes angol szó. Feltételezzük, hogy betűgyakoriság-vizsgálattal sikerült beazonosítani az* **E**, **T**, **A**, **O**, **I** *betűk megfelelőit, vagyis a permutáció a következő alakú:*

$$
\begin{array}{c}
\begin{array}{cccccccccccccccccccccccccc}
0&1&2&3&4&5&6&7&8&9&10&11&12&13&14&15&16&17&18&19&20&21&22&23&24&25
\end{array}\\
\begin{pmatrix}
A&B&C&D&E&F&G&H&I&J&K&L&M&N&O&P&Q&R&S&T&U&V&W&X&Y&Z\\
U&\cdot&\cdot&\cdot&Y&\cdot&\cdot&\cdot&C&\cdot&\cdot&\cdot&\cdot&\cdot&G&\cdot&\cdot&\cdot&\cdot&M&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot
\end{pmatrix}
\end{array}
$$

Észrevehetjük az alábbiakat:

- *Mivel* **C** *az* **Y** *után van és a kulcsszó négybetűs, a kulcsszó tartalmazza vagy* **Y**-*t vagy* **C**-*t (mindkettőt nem tartalmazhatja).*
- *Az alsó sorban* **G**-*től* **M**-*ig ábécésorrendben van négy kulcsszón kívüli betű, de* **G** *és* **M** *között az ábécében öt betű van:* **H**, **I**, **J**, **K** *és* **L**. *Ez azt jelenti, hogy egyikük a kulcsban van.*
- *Az alsó sorban* **M**-*től* **U**-*ig ábécésorrendben van hat kulcsszón kívüli betű, de* **M** *és* **U** *között az ábécében hét betű van:* **N**, **O**, **P**, **Q**, **R**, **S**, **T**. *Ez azt jelenti, hogy egyikük a kulcsszó része.*
- *Az előbbiek alapján, ha* **Y** *a kulcsszó része, akkor* **V**, **W**, **X**, **Z** *közül csak egy kerülhet a kulcsba. Így azonban a kulcsszó* **Y**-*nal kezdődne és* **C**-*ig terjedne – de ez lehetetlen, mert akkor* **I** *alá* **A** *kellene kerüljön. Tehát* **Y** *nincs a kulcsszóban és így* **C** *a kulcs része, az alsó sor eleje pedig* **UVWXY**.
- *Az alsó sorban* **C**-*től* **G**-*ig van öt betű, ezek csak az* **A**, **B**, **D**, **E** *valamint* **F** *betűk lehetnek,* **ABDEF**, **BADEF**, **DABEF**, **EABDF** *vagy* **FABDE** *sorrendben.*
- *Összegezve megállapítható, hogy a kulcsszó betűi:*
  - **C**
  - *egy betű a* **H**, **I**, **J**, **K**, **L** *közül*
  - *egy betű a* **N**, **O**, **P**, **Q**, **R**, **S**, **T** *közül*
  - **Z** *vagy egy betű* **A**, **B**, **D**, **E**, **F** *közül*

Látható, hogy jóval kevesebb lehetőségünk van, mint hogyha kimerítő kulcskeresést alkalmaztunk volna. Végül bemutatunk két lehetséges megoldást:

$k = 5$, *kulcsszó:* **ZINC**

$$
\begin{array}{c}
\begin{array}{cccccccccccccccccccccccccc}
0&1&2&3&4&\mathbf{5}&6&7&8&9&10&11&12&13&14&15&16&17&18&19&20&21&22&23&24&25
\end{array}\\
\begin{pmatrix}
A&B&C&D&E&\boxed{\mathbf{F}}&G&H&I&J&K&L&M&N&O&P&Q&R&S&T&U&V&W&X&Y&Z\\
U&V&W&X&Y&\mathbf{Z}&\mathbf{I}&\mathbf{N}&\mathbf{C}&A&B&D&E&F&G&H&J&K&L&M&O&P&Q&R&S&T
\end{pmatrix}
\end{array}
$$

$k = 6$, *kulcsszó:* **NICE**

$$
\begin{array}{c}
\begin{array}{cccccccccccccccccccccccccc}
0&1&2&3&4&5&\mathbf{6}&7&8&9&10&11&12&13&14&15&16&17&18&19&20&21&22&23&24&25
\end{array}\\
\begin{pmatrix}
A&B&C&D&E&F&\boxed{\mathbf{G}}&H&I&J&K&L&M&N&O&P&Q&R&S&T&U&V&W&X&Y&Z\\
U&V&W&X&Y&Z&\mathbf{N}&\mathbf{I}&\mathbf{C}&\mathbf{E}&A&B&D&F&G&H&J&K&L&M&O&P&Q&R&S&T
\end{pmatrix}
\end{array}
$$

Mindkettővel dekódoljuk a titkosított szöveget, és amelyikre értelmes üzetenetet találunk, az a jó kulcs.

2.1.1.3. Affin-rejtjel

Az *affin-rejtjel* lényege szintén egy partikuláris betűpermutáció, tehát az affin-rejtjel is egy monoalfabetikus helyettesítő kód. Akárcsak a Caesar-rejtjel esetében, itt is veszünk egy n betűs ábécét, és a betűket azonosítjuk a $\mathbb{Z}_n$-beli elemekkel a 0-tól számított ábécébeli sorszámuk alapján.

Kulcs: $k = (a, b) \in \mathbb{Z}_n^2$ úgy, hogy létezik $a^{-1} \pmod{n}$, vagyis $(a, n) = 1$.

Kódolás: $C = E_k(P) = (aP + b) \bmod n$, ahol $P \in \mathbb{Z}_n$ (egy betű).

Dekódolás: $P = D_k(C) = (a^{-1}C - a^{-1}b) \bmod n$, ahol $C \in \mathbb{Z}_n$ (egy betű).

2.1.4. példa. *Vegyük a* 26 *betűs angol ábécét, a már ismertetett sorszámozással.*
Legyen $k = (5, 2)$. *Látható, hogy* $(5, 26) = 1$. *Ekkor:*

$$
\begin{aligned}
E_k(\mathbf{T}) &= E_k(19) = (5 \cdot 19 + 2) \bmod 26 = 19 = \mathbf{T} \\
E_k(\mathbf{I}) &= E_k(8) = (5 \cdot 8 + 2) \bmod 26 = 16 = \mathbf{Q} \\
E_k(\mathbf{T}) &= E_k(19) = (5 \cdot 19 + 2) \bmod 26 = 19 = \mathbf{T} \\
E_k(\mathbf{O}) &= E_k(14) = (5 \cdot 14 + 2) \bmod 26 = 20 = \mathbf{U} \\
E_k(\mathbf{K}) &= E_k(10) = (5 \cdot 10 + 2) \bmod 26 = 0 = \mathbf{A}
\end{aligned}
$$

Tehát a **TITOK** *szó titkosítva* **TQTUA** *lesz.*

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadások)

1. Kimerítő kulcskereséssel $n\varphi(n)$ kulcsot kell kipróbálni, ahol

   $$\varphi(n) = \{0 \le a < n \,|\, (a, n) = 1\}$$

   az Euler-függvény.

2. Betűgyakoriság-vizsgálattal elegendő két betű kódolt megfelelőjét beazonosítani. Legyenek ezek (P_1, C_1) és (P_2, C_2). Ekkor

   $$\begin{cases} C_1 = (aP_1 + b) \bmod n \\ C_2 = (aP_2 + b) \bmod n \end{cases} \Longrightarrow a(P_1 - P_2) \equiv (C_1 - C_2) \pmod{n}$$

   és ez a kongruencia biztosan megoldható. Legyen $(P_1 - P_2, n) = d$. A következő két eset lehetséges:

   (a) Ha $d = 1$, akkor $a \equiv (P_1 - P_2)^{-1}(C_1 - C_2) \pmod{n}$.

(b) Ha $d > 1$, akkor $a \equiv \left(\frac{P_1-P_2}{d}\right)^{-1}\left(\frac{C_1-C_2}{d}\right) \equiv u \pmod{\frac{n}{d}}$, ahol $u = \left(\frac{P_1-P_2}{d}\right)^{-1}\left(\frac{C_1-C_2}{d}\right)$.

Az a lehetséges értékei tehát:

$$\begin{aligned}
a &\equiv u \pmod{n}\\
a &\equiv u + \frac{n}{d} \pmod{n}\\
a &\equiv u + 2\frac{n}{d} \pmod{n}\\
&\vdots\\
a &\equiv u + (d-1)\frac{n}{d} \pmod{n}
\end{aligned}$$

Van tehát összesen d lehetőségünk a-ra.

2.1.5. példa. *Egy angol ábécét használó affin-rejtjel esetén betűgyakoriság-vizsgálattal megállapítjuk, hogy az* **E** *betűnek* **M** *és az* **A** *betűnek a* **W** *a kódolt megfelelője. Vagyis:*

$$\begin{cases} a \cdot 4 + b \equiv 12 \pmod{26} \\ a \cdot 0 + b \equiv 22 \pmod{26} \end{cases} \Longrightarrow b \equiv 22 \pmod{26}.$$

$a \cdot 4 + 22 \equiv 12 \pmod{26} \Longrightarrow a \cdot 4 \equiv 16 \pmod{26} \Longrightarrow a \cdot 2 \equiv 8 \pmod{13} \Longrightarrow a \equiv 2^{-1} \cdot 8 \equiv 7 \cdot 8 \equiv 4 \pmod{13}$. *Tehát* $a \equiv 4 \pmod{26}$ *vagy pedig* $a \equiv 17 \pmod{26}$.

2.1.1. megjegyzés. *Ha* $(a, n) = 1$, *akkor* $a^{-1} \bmod n = u$, *ahol* $au + bv = 1$ *és* u*-t illetve* v*-t a* kiterjesztett euklidészi algoritmus*sal kapjuk meg.*

Az eddig említett rendszerek mindegyike monoalfabetikus rendszer volt, vagyis egy adott betű bárhol is van a szövegben, ugyanabba a betűbe kódolódik. Az ilyen rendszerek, amint láttuk, könnyen feltörhetőek betűgyakoriság-vizsgálattal. A továbbiakban vizsgált rendszerekben a helyettesített szövegegység nem korlátozódik egyetlen betűre, így nem lesz monoalfabetikus, hanem úgynevezett *polialfabetikus*.

2.1.2. Mátrixos rendszerek

2.1.2.1. A mátrixos affin-rejtjel

Tételezzük fel, hogy egy n betűs ábécét használunk, tehát sorszámuk alapján a betűk $\mathbb{Z}_n$-beli elemeknek tekinthetőek. Legyen $\ell \geq 2$. A szövegünket ℓ hosszúságú tömbökre bontjuk úgy, hogy az első ℓ betű az első tömb, a következő ℓ betű a második tömb stb. Tehát most a titkosítandó eredeti üzenetegység

$$P = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} \in \mathbb{Z}_n^{\ell}.$$

Kulcs: $k = (A, b)$, ahol $A = (a_{ij}) \in M_\ell(\mathbb{Z}_n)$ egy $\ell \times \ell$-es invertálható mátrix $\mathbb{Z}_n$ felett

$$(\text{vagyis } \exists (\det A)^{-1} \mod n \iff (\det A, n) = 1),\ b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_\ell \end{pmatrix} \in \mathbb{Z}_n^{\ell}.$$

Kódolás: $C = E_k(P) = AP + b$. Részletesebben:

$$\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_\ell \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1\ell} \\ a_{12} & a_{22} & \dots & a_{2\ell} \\ \vdots & \vdots & \ddots & \vdots \\ a_{\ell 1} & a_{\ell 2} & \dots & a_{\ell\ell} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_\ell \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_\ell \end{pmatrix}.$$

Dekódolás: $P = A^{-1}C - A^{-1}b$.

2.1.2. megjegyzés. *Megtörténhet, hogy a szöveg hosszúsága nem többszöröse ℓ-nek, ekkor általában vagy levágjuk a szöveg végét, vagy kiegészítjük (angolul* padding*) megfelelő számú betűvel (például* **X***-szel az angol ábécében) úgy, hogy a hossz ℓ többszöröse legyen.*

Kriptoanalízis:

1. Csak a kódolt üzenet ismeretén alapuló támadások:

   (a) Kimerítő kulcskeresés: ha ℓ nagy, nem túl hatékony, hiszen körülbelül $\varphi(n)n^{\ell^2+\ell-1}$ lehetséges kulcsunk van. Ez például $n = 26$ és $\ell = 10$ esetében is már sok.

   (b) A szövegegységek gyakoriságának vizsgálata is akkor működik, ha ℓ kicsi. $\ell = 2$-re betűpárgyakoriság-vizsgálatot lehet végezni. Az angol nyelvben például a **TH** pár a leggyakoribb. $\ell = 3$-ra az **AND** és **THE** szavak ismétlődhetnek gyakran, tehát ezek megfelelőit lehetne megkeresni.

2. Nyílt szöveg ismeretén alapuló támadás. Ha $A = (a_{ij}) \in M_m(\mathbb{Z}_n)$, $b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{Z}_n^m$, akkor ismernünk kell $m + 1$ darab eredeti üzenetet kódolt megfelelőivel

együtt – vagyis $m+1$ darab (P^i, C^i) párt – ahhoz, hogy lineáris kongruencia-rendszerből megkapjuk A-t és b-t. Valóban, ha $P^i = \begin{pmatrix} p_1^i \\ \vdots \\ p_m^i \end{pmatrix}$ és $C^i = \begin{pmatrix} c_1^i \\ \vdots \\ c_m^i \end{pmatrix}$, akkor a rendszer (mátrix alakban):

$$\begin{pmatrix} c_1^1 - b_1 & \dots & c_1^{m+1} - b_1 \\ \vdots & \ddots & \vdots \\ c_m^1 - b_m & \dots & c_m^{m+1} - b_m \end{pmatrix} = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} \begin{pmatrix} p_1^1 & \dots & p_1^{m+1} \\ \vdots & \ddots & \vdots \\ p_m^1 & \dots & p_m^{m+1} \end{pmatrix},$$

ahol az $m^2 + m = m(m+1)$ darab ismeretlen (a_{ij}) és b_j, $i, j = \overline{1, m}$. Látható, hogy van $m(m+1)$ darab egyenlet. Ha $b = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$, akkor elegendő m darab (P^i, C^i) párt ismerni. Megtörténhet, hogy a fenti kongruencia-rendszernek több megoldása van, mint ahogy ez az alábbi példából kiderül [6].

2.1.6. példa. *Feltételezzük, hogy angol ábécét használunk, tehát $n = 26$ és $A \in M_2(\mathbb{Z}_{26})$, $b = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. Tudjuk, hogy a* **WKNCCHSSJH** *kódolt szöveg eleje* **GIVE**. *Találjuk meg az A mátrixot, és olvassuk ki az eredeti szöveget.*

Ebben az esetben két (P^i, C^i) nyílt szöveg–kódolt szöveg párt ismerünk ($i = \overline{1, 2}$), ezek:

$$P^1 = \begin{pmatrix} \mathbf{G} \\ \mathbf{I} \end{pmatrix} = \begin{pmatrix} 6 \\ 8 \end{pmatrix}, \quad C^1 = \begin{pmatrix} \mathbf{W} \\ \mathbf{K} \end{pmatrix} = \begin{pmatrix} 22 \\ 10 \end{pmatrix}$$

$$P^2 = \begin{pmatrix} \mathbf{V} \\ \mathbf{E} \end{pmatrix} = \begin{pmatrix} 21 \\ 4 \end{pmatrix}, \quad C^2 = \begin{pmatrix} \mathbf{N} \\ \mathbf{C} \end{pmatrix} = \begin{pmatrix} 13 \\ 2 \end{pmatrix}.$$

Tehát a rendszerünk (mátrix alakban) $C = AP \bmod 26$, ahol

$$C = \begin{pmatrix} C^1 & C^2 \end{pmatrix} = \begin{pmatrix} 22 & 13 \\ 10 & 2 \end{pmatrix} \text{ és } P = \begin{pmatrix} P^1 & P^2 \end{pmatrix} = \begin{pmatrix} 6 & 21 \\ 8 & 4 \end{pmatrix}.$$

A legegyszerűbb az lenne, ha kifejezhetnénk A^{-1}-et úgy, hogy $A^{-1} = PC^{-1} \bmod 26$, de ez sajnos nem lehetséges, mert $\det(C) = 18$ és 26 nem relatív prímek, tehát $\nexists C^{-1} \bmod 26$.

A következő módszert alkalmazhatjuk: megoldjuk a rendszert modulo 13 (ez lehetséges, mert $\det(C) = 18$ és 13 relatív prímek). Legyen tehát

$$\overline{A} = A \bmod 13,$$

$$\overline{P} = P \bmod 13 = \begin{pmatrix} 6 & 8 \\ 8 & 4 \end{pmatrix}, \quad \overline{C} = C \bmod 13 = \begin{pmatrix} 9 & 0 \\ 10 & 2 \end{pmatrix},$$

$$\overline{A}^{-1} = \overline{P}\,\overline{C}^{-1} = \begin{pmatrix} 6 & 8 \\ 8 & 4 \end{pmatrix} \begin{pmatrix} 9 & 0 \\ 10 & 2 \end{pmatrix}^{-1} \equiv \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix} \pmod{13}.$$

Visszatérve modulo 26*-ra látható, hogy* $A^{-1} = \overline{A}^{-1} + 13 \cdot A_1$*, ahol* A_1 *egy* 2×2*-es bináris mátrix, így elvileg* 2^4 *lehetőség adódik* A^{-1} *megválasztására. Tudjuk viszont, hogy* A^{-1} *invertálható modulo* 26*, vagyis* $(\det(A^{-1}), 26) = 1$*, ezért* $\det(A^{-1})$ *nem lehet páros. Azonnal kizárhatjuk tehát az*

$$A_1 \in \left\{ \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 0 \\ 0 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ 1 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 0 \\ 1 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right) \right\}$$

eseteket. A megmaradt hat A_1 *mátrixot sorra behelyettesítve és ellenőrizve az* $A^{-1}C = P$ *összefüggést, még kizárhatunk négy darab* A_1*-et. Végül két lehetőség marad:* $A_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ *vagy* $A_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$*. Ez azt jelenti, hogy* $A^{-1} = \begin{pmatrix} 15 & 4 \\ 16 & 15 \end{pmatrix}$ *vagy* $A^{-1} = \begin{pmatrix} 15 & 17 \\ 16 & 15 \end{pmatrix}$*. Ezekkel a mátrixokkal dekódolva a szöveget előbb* **GIVEGHEMHP**-*t, majd* **GIVETHEMUP**-*ot kapunk, ami azt mutatja, hogy az* $A^{-1} = \begin{pmatrix} 15 & 17 \\ 16 & 15 \end{pmatrix}$ *a jó kulcs.*

A mátrixos affin-rejtjel egyik egyszerűbb formáját, amikor $b = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{Z}_n^\ell$, még Hill-rejtjelnek is nevezik. Feltalálója, Lester S. Hill, az 1930-as években még egy mechanikus rejtjelezőgépet is szabadalmaztatott, ami a kódolás/dekódolás műveletét automatizálta.

2.1.2.2. A Vigenère-rejtjel

Ennek a rendszernek egy változatát elsőként Giovan Battista Bellaso írta le 1553-ban. Később Blaise de Vigenère is adott egy leírást, ezért róla nevezték el. Eredeti formájában adott volt egy 26×26-os, úgynevezett Vigenère-tábla, melynek első sora az ábécé betűiből áll, ábécé sorrendben, a második sora az első sor ciklikusan balra tolva egy betűvel, a harmadik sora a második sor ciklikusan balra tolva egy betűvel és így tovább. A Vigenère-tábla tehát úgy néz ki, mint ahogyan a 2.3 táblázatban látható (minden sorban és oszlopban egy betű csak egyszer szerepel).

Kulcs: A kulcs egy szó, melynek egymásutáni ismétlésével megkapjuk a kulcsszöveget. A kulcs hosszát *periódus*nak nevezzük.

Kódolás: A kódolt szöveg k-adik betűjét a következő módon kapjuk meg: megkeressük az eredeti szöveg k-adik betűjét (mondjuk ez az ábécé i-edik betűje) és a kulcsszöveg k-adik betűjét (mondjuk ez az ábécé j-edik betűje); ekkor a kódolt

2.3. táblázat. A Vigenère-tábla

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

betű a Vigenère-tábla i-edik sorában és j-edik oszlopában levő betű. Vegyük észre, hogy ez megegyezik a tábla j-edik sorában és i-edik oszlopában levő betűvel, hiszen a tábla szimmetrikus a főátlóra nézve.

Dekódolás: Az eredeti szöveg k-adik betűjét a következő módon kapjuk vissza: ha a kulcsszöveg k-adik betűje az i-edik az ábécében, akkor a tábla i-edik sorában megkeressük a kódolt szöveg k-adik betűjét. Tételezzük fel, hogy ez az i-edik sor j-edik pozíciójában van. Ekkor az eredeti betű az ábécé j-edik betűje.

2.1.7. példa. *Az eredeti szöveg legyen* **GIVETHEMUP**, *a kulcs pedig* **TEA**. *Ekkor a kulcsszöveg* **TEATEATEAT** *(ugyanolyan hosszú, mint az eredeti szöveg). A kódolt szöveg* **ZMVXXHXQUI** *lesz.*

Ha figyelmesebben megnézzük a Vigenère-rejtjelt, láthatjuk, hogy ez egy partikuláris mátrixos affin rendszer, ráadásul ezek közül is az egyik legegyszerűbb. Valóban, ha a kulcs a $b_1 b_2 \dots b_m$ szó, akkor m betűből álló tömbönként kódolva igaz, hogy

$$\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \bmod 26,$$

ahol $P = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{pmatrix}$ a nyílt szöveg és $C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix}$ a kódolt üzenet.

Az előbbi példában ez a következő módon alakul:

$$\begin{pmatrix} \mathbf{G} \\ \mathbf{I} \\ \mathbf{V} \end{pmatrix} + \begin{pmatrix} \mathbf{T} \\ \mathbf{E} \\ \mathbf{A} \end{pmatrix} = \begin{pmatrix} 6 \\ 8 \\ 21 \end{pmatrix} + \begin{pmatrix} 19 \\ 4 \\ 0 \end{pmatrix} = \begin{pmatrix} 25 \\ 12 \\ 21 \end{pmatrix} = \begin{pmatrix} \mathbf{Z} \\ \mathbf{M} \\ \mathbf{V} \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{E} \\ \mathbf{T} \\ \mathbf{H} \end{pmatrix} + \begin{pmatrix} \mathbf{T} \\ \mathbf{E} \\ \mathbf{A} \end{pmatrix} = \begin{pmatrix} 4 \\ 19 \\ 7 \end{pmatrix} + \begin{pmatrix} 19 \\ 4 \\ 0 \end{pmatrix} = \begin{pmatrix} 23 \\ 23 \\ 7 \end{pmatrix} = \begin{pmatrix} \mathbf{X} \\ \mathbf{X} \\ \mathbf{H} \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{E} \\ \mathbf{M} \\ \mathbf{U} \end{pmatrix} + \begin{pmatrix} \mathbf{T} \\ \mathbf{E} \\ \mathbf{A} \end{pmatrix} = \begin{pmatrix} 4 \\ 12 \\ 20 \end{pmatrix} + \begin{pmatrix} 19 \\ 4 \\ 0 \end{pmatrix} = \begin{pmatrix} 23 \\ 16 \\ 20 \end{pmatrix} = \begin{pmatrix} \mathbf{X} \\ \mathbf{Q} \\ \mathbf{U} \end{pmatrix}$$

$$\mathbf{P} + \mathbf{T} = 15 + 19 \quad = 8 \quad = \mathbf{I}$$

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadás)
Ha tudjuk az m periódust, akkor tulajdonképpen m darab Caesar-kódot kell feltörnünk (legegyszerűbben betűgyakoriság-vizsgálattal). Valóban, vegyük észre, hogy $c_i \equiv p_i + b_j \pmod{26}$, $\forall i \equiv j \pmod{m}$, $j = \overline{1, m}$-re, tehát $c_1 c_{m+1} c_{2m+1} \ldots$ a $p_1 p_{m+1} p_{2m+1} \ldots$-ből keletkezett b_1-es eltolással, $c_2 c_{m+2} c_{2m+2} \ldots$ a $p_1 p_{m+2} p_{2m+2} \ldots$-ből keletkezett b_2-es eltolással és így tovább egészen $c_m c_{2m} c_{3m} \ldots$-ig, ami $p_m p_{2m} p_{3m} \ldots$-ből keletkezett b_m-es eltolással. Az egyetlen probléma az m periódus meghatározása. Erre viszont W. Kasiski módszerét lehet alkalmazni, melynek lényege a következő: a kódolt szövegben ismétlődő szövegrészeket keresünk. Legyen például **UCLA** egy ismétlődő szövegrész. Megkeressük, mikor a legkisebb két **UCLA** előfordulás között a távolság, és ekkor a periódus hossza osztja az első előfordulás **U** betűjétől (első betűjétől) a második előfordulás **U** betűjéig (**U**-t kizárva) a betűk számát. Vagyis ha:

U	**C**	**L**	**A**	X	U	S	T	A	V	Z	B	**U**	**C**	**L**	**A**
1	2	3	4	5	6	7	8	9	10	11	12				

akkor m osztója 12-nek.

2.1.2.3. A Playfair-kód

Charles Wheatstone találta fel 1854-ben, de alkalmazását Lord Playfair szorgalmazta – innen az elnevezés.

Kulcs: A kulcs egy 5×5-ös betűtábla, melyben az angol ábécé betűi szerepelnek a **J** kivételével. Ezt a betűtáblát egy kulcsszó segítségével is ki lehet tölteni a kulcsszavas Caesar-kódnál megismert módon: a kulcsszó (betűismétlések nélkül) a legfelső sor bal sarkából indul balról jobbra és fentről lefele (ha 5 betűnél hosszabb); a kimaradt betűk ábécésorrendben követik a kulcsszót balról jobbra és fentről lefele.

2.1.8. példa. *Ha a kulcsszó* **MOON**, *akkor a neki megfelelő Playfair-tábla:*

M	**O**	**N**	A	B
C	D	E	F	G
H	I	K	L	P
Q	R	S	T	U
V	W	X	Y	Z

Kódolás: A kódolás betűpáronként történik, ehhez a szöveget betűpárokra bontjuk, az esetleges **J** betűket **I**-re cserélve. Ha egy betűpárban kétszer szerepel ugyanaz a betű, akkor az első betű után egy **X**-et szúrunk be. Ha a betűk száma páratlan, akkor a szöveget egy **X**-szel egészítjük ki a végén. A betűpárokat három eset szerint a következő módon kódoljuk:

1. Ha a betűpár betűi a tábla azonos sorában vannak, akkor a táblában ciklikusan jobbra toljuk őket egy pozícióval. A 2.1.8 példában: **OB** $\longrightarrow$ **NM**, **IK** $\longrightarrow$ **KL**, **GC** $\longrightarrow$ **CD**.

2. Ha a betűpár betűi a tábla azonos oszlopában vannak, akkor ciklikusan lefele toljuk őket egy pozícióval. A 2.1.8 példában: **MV** $\longrightarrow$ **CM**, **SE** $\longrightarrow$ **XK**.

3. Ha a betűpár betűi nincsenek sem azonos sorban, sem azonos oszlopban, akkor a táblában az általuk (átellenes csúcsokként) meghatározott téglalap másik két átellenes csúcsába kódolódnak úgy, hogy az eredeti betűpár első betűje és a kódolt betűpár első betűje egy sorban legyenek. A 2.1.8 példában: **CP** $\longrightarrow$ **GH**, **RA** $\longrightarrow$ **TO**.

Dekódolás: Ugyanúgy történik, mint a kódolás, csak az első esetben ciklikusan balra tolunk egy pozícióval és a második esetben ciklikusan felfele tolunk egy pozícióval.

A Playfair-kód betűpárokat betűpárokkal helyettesít, ezért ez egy poligrafikus helyettesítő kód.

2.1.9. példa. *A 2.1.8 példabeli tábla alapján a* **JIMMYS** *szót így titkosítjuk:* **JIMMYS** $\longrightarrow$ ***I*****IMMYS** $\longrightarrow$ **I***X* **IM MY S***X* $\longrightarrow$ **KWHOAVXN**.

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadás)

1. A kimerítő kulcskeresés módszerével 25! lehetséges táblát kell végigpróbálni.

2. Betűpárgyakoriság-vizsgálat csökkentheti a lehetséges kulcsok számát (például a **TH** betűpár a leggyakoribb az angol nyelvben).

3. Ha a kulcsszó rövid, akkor a kulcsszavas Caesar-kódnál ismertetett betűtávolság-mérés is hasznos lehet. Ezt a következő példa illusztrálja [10]. Feltételezzük, hogy egy Playfair-kulcsszó három betűs értelmes angol szó, és betűgyakoriság-vizsgálat alapján rájövünk arra, hogy a **TH** betűpár titkosított megfelelője **QN**. Vegyük észre, hogy:

   (a) A Playfair-táblában a **T**, **H**, **Q** és **N** betűk nem kerülhetnek egy sorba. Először lássuk be, miért nem kerülhetnek ezek a betűk az első sorba. Ha az első sorban lennének, mivel a kulcsszó három betűs, a **T**, **H**, **Q** és **N** betűk közül legalább az egyik nem része a kulcsszónak. Tegyük fel, hogy a kulcsszón kívüli betű a **H**, de akkor ez nem lehet az első sorban, mert (a tábla kitötlési módja miatt) a **H**-t meg kell előzze az ábécésorrenben előtte levő hét másik betű (**A**, **B**, **C**, **D**, **E**, **F**, **G**). Ugyanez a gondolatmenet érvényes az **N**, **Q** és **T** betűkre is (esetükben még több betű előzi meg őket). Ezért a **T**, **H**, **Q** és **N** betűk egyike sem lehet az első sorban. Ha nem az első sorban vannak, hanem akármelyik másikban (tehát egyikük sem része a kulcsszónak), akkor mivel **T** $\longrightarrow$ **Q** és **T** a **Q** után van az ábécében, csak a **QHNT** sorrend jöhetne szóba, viszont ez megint lehetetlen, mert **H** a **Q** előtt van az ábécében.

(b) A **T**, **H**, **Q** és **N** betűk nem lehetnek egy oszlopban sem. Mivel **T** $\longrightarrow$ **Q** és **H** $\longrightarrow$ **N**, ha egy oszlopban lennének, a következő két eset állna fenn: vagy **Q** van **T** alatt vagy fordítva, **T** van **Q** alatt. Az első esetben, ha **Q** van **T** alatt, akkor **T**-nek a kulcsszóban kell lennie (az első sor első, második vagy harmadik pozíciójában), **Q**-nak pedig közvetlenül alatta. Ez viszont lehetetlen, mert ehhez **Q** túl hátul van az ábécében. A második esetben, ha **T** van **Q** alatt, akkor a **Q** betű kell szerepeljen a kulcsszóban az első sor első, második vagy harmadik pozíciójában. Ezért ebben az esetben az első három oszlop valamelyike $\begin{array}{c}\mathbf{Q}\\\mathbf{H}\\\mathbf{N}\\\bullet\\\mathbf{T}\end{array}$ vagy $\begin{array}{c}\mathbf{Q}\\\bullet\\\mathbf{H}\\\mathbf{N}\\\mathbf{T}\end{array}$ alakú. Ha feltételezzük, hogy az oszlop $\begin{array}{c}\mathbf{Q}\\\mathbf{H}\\\mathbf{N}\\\bullet\\\mathbf{T}\end{array}$ alakú, akkor a Playfair-táblázat kitöltésének sorrendje szerint **N**-től **T**-ig van 9 üres hely, de a ábécében csak 5 betű, így ez az alak ki van zárva. Ezért csak a $\begin{array}{c}\mathbf{Q}\\\bullet\\\mathbf{H}\\\mathbf{N}\\\mathbf{T}\end{array}$ alakú oszlop jöhet számításba. Ebben az esetben a legalsó sorban marad legfennebb 4 üres hely (ha a **T** betű az első oszlopban van), az ábécében viszont a **T** után van még 6 betű: **U**, **V**, **W**, **X**, **Y**, **Z**. A Playfair-tábla alakja tehát a következő lenne: $\begin{array}{ccccc}\mathbf{Q}&\bullet&\bullet&\mathbf{A}&\mathbf{B}\\\mathbf{C}&\mathbf{D}&\mathbf{E}&\mathbf{F}&\mathbf{G}\\\mathbf{H}&\mathbf{I}&\mathbf{K}&\mathbf{L}&\mathbf{M}\\\mathbf{N}&\mathbf{O}&\mathbf{P}&\mathbf{R}&\mathbf{S}\\\mathbf{T}&\bullet&\bullet&\bullet&\bullet\end{array}$, ahol a kulcs **Q**-val kezdődik és két további betűje az **U**, **V**, **W**, **X**, **Y** és **Z** betűk közül kerül ki. Mivel ezekkel a betűkkel nem lehet értlemes három betűből álló angol szót alkotni, beláthatjuk eredeti állításunkat, miszerint a **T**, **H**, **Q** és **N** betűk nem lehetnek egyazon oszlopban.

(c) Nem lehet $\begin{array}{cc}\mathbf{T}&\mathbf{Q}\\\mathbf{N}&\mathbf{H}\end{array}$ téglalap a táblában, mert így **N** megelőzi **H**-t (nincsenek ábécésorrendben).

(d) Nem lehet $\begin{array}{cc}\mathbf{N}&\mathbf{H}\\\mathbf{T}&\mathbf{Q}\end{array}$ téglalap a táblában, mert így **T** megelőzi **Q**-t (nincsenek ábécésorrendben).

(e) Nem lehet $\begin{array}{cc}\mathbf{Q}&\mathbf{T}\\\mathbf{H}&\mathbf{N}\end{array}$ téglalap a táblában, mert akkor **Q** és **T** a kulcsszóba kerül, így **H** és **N** közé kellene hogy kerüljenek az **I**, **K**, **L** és **M** betűk, ami lehetetlen, mivel **Q** és **T** között csak egy betűnek van hely.

(f) Utolsó lehetőségként marad, hogy a **T**, **H**, **Q** és **N** betűk egy $\begin{smallmatrix}\mathbf{H} & \mathbf{N}\\ \mathbf{Q} & \mathbf{T}\end{smallmatrix}$ alakú téglalapot képeznek. Vizsgáljuk meg most ezt az esetet. A **H** és **N** betűk nem lehetnek az első sorban. Valóban, ha ott lennének, akkor közéjük legtöbb egy betű férne, viszont az ábécében a **Q** és **T** között 2 betű van: **R** és **S**. Még ha egyiküket a kulcsba is tesszük (annak ellenére, hogy sem a **HRN** sem a **HSN** nem értelmes angol szó), akkor sem megyünk sokra, mert lehetetlen lesz felépíteni a Playfair-táblázatot (kimarad a **P** betű): $\begin{smallmatrix}\mathbf{H} & \mathbf{R/S} & \mathbf{N} & \mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D} & \mathbf{E} & \mathbf{F} & \mathbf{G}\\ \mathbf{I} & \mathbf{K} & \mathbf{L} & \mathbf{M} & \mathbf{O}\\ \mathbf{Q} & \mathbf{S/R} & \mathbf{T} & \bullet & \bullet\\ \bullet & \bullet & \bullet & \bullet & \bullet\end{smallmatrix}$. Az eddig végiggondolt eseteket figyelembe véve, a következtetés tehát az, hogy:

(g) A **H** két betűnyi távolságra van **N**-től a harmadik sorban. Valóban, **H** és **N** között a használt ábécében van négy betű (**I**, **K**, **L**, **M**), **Q** és **T** között két betű (**R**, **S**). Ez azt jelenti, hogy **I**, **K**, **L** és **M** közül kettő bekerül a kulcsszóba. **N**-től **Q**-ig a táblázatban van két betű (**O** és **P**), de csak egynek van helye, tehát ezek közül is egy bekerül a kulcsszóba. Tehát a keresett Playfair-tábla:

•	•	•	A	B
C	D	E	F	G
H	•	•	**N**	•
Q	R	S	**T**	U
V	W	X	Y	Z

A kulcsszóban egy betű **O** és **P**, a másik két betű pedig **I**, **K**, **L**, és **M** közül való. Lehetséges kulcsszavak tehát: **OIL** vagy **MOL**.

2.1.3. megjegyzés. *A Playfair-kód általánosítható és alkalmazható más ábécére is, megváltoztatva a tábla dimenzióit.*

2.1.3. A kódkönyv

A *kódkönyv*, mint egy kétirányú szótár, számcsoportokat, esetleg (értelmetlen vagy értelmes) szavakat feleltet meg betűcsoportoknak, szavaknak, szócsoportoknak vagy akár rövid mondatoknak. Nyilván mindkét félnek (küldő és vevő) rendelkeznie kell a kódkönyvvel ahhoz, hogy kommunikálni tudjanak. Előnye a nehéz feltörhetőség, hiszen szinte lehetetlen rájönni egy-egy kódkönyvben szereplő néha 10000-nél is több bejegyzésre. Egyik hátránya a kódolási-dekódolási folyamat automatizálhatóságának nehézsége (számítógép hiányában) és ezáltal a folyamat lassúsága. Szintén hátrány a kódkönyvek biztonságának garantálása, mely igen költséges lehet, valamint a kódkönyvek szintén költséges gyakori frissítése, melynek célja a szóismétlődések vizsgálatán alapuló kriptoanalízis nehezítése.

A kódkönyvek nagy bukása az első világháború idején kódkönyvvel titkosított Zimmermann-távirat megfejtése volt. 1917. január 16-án Arthur Zimmermann német külügyminiszter a 0075-ös német diplomáciai kódkönyvvel kódolt üzenetet küldött Mexikóba, az amerikai német nagykövetségen keresztül. Ebben egyrészt szövetséget ajánlott Mexikónak, másrészt pedig arra ösztönzi, hogy ha a még semleges Amerika brit oldalon belépne a háborúba, akkor támadja meg Amerikát. A szövetség ára Texas, Új Mexikó és Arizona Mexikóhoz való csatolása lett volna. A 0075-ös diplomáciai kóddal titkosított táviratot a 2.6 képen láthatjuk.

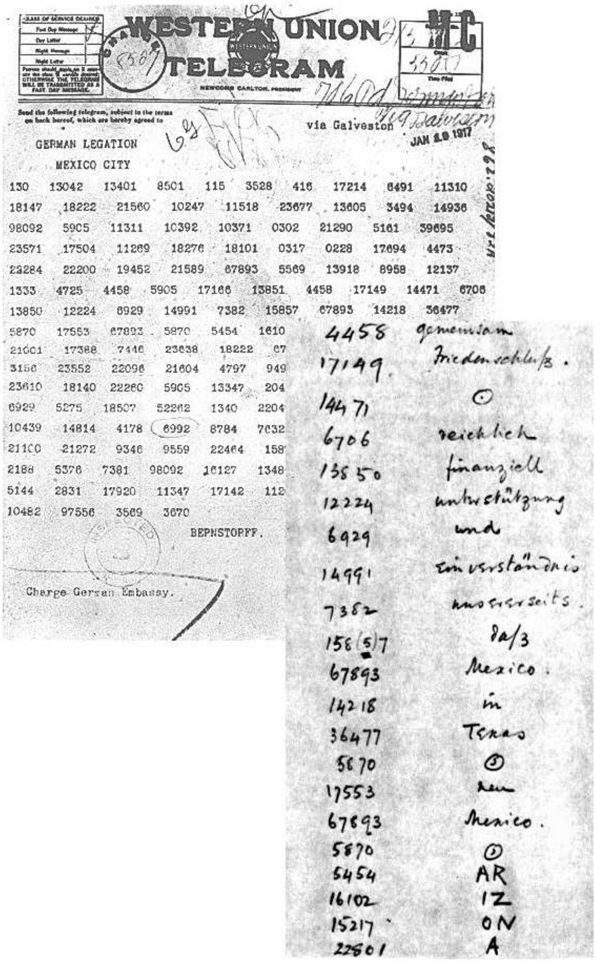
A brit titkosszolgálatnak sikerült megszereznie a kódolt üzenetet, majd részben feltörnie egy kalandos úton megszerzett régebbi német kódkönyv segítségével. A távirat dekódolt szövege a következő[11]:

> *Szándékunkban áll február elsején korlátozás nélküli tengeralattjáró-hadviselést indítani. Tesszük ezt annak dacára, hogy szeretnénk, ha az Egyesült Államok fenntartaná semlegességét. Ha nem így alakulna, szövetséget ajánlunk Mexikónak a következő feltételekkel: közös hadviselés, közös békekötés, bőkezű anyagi támogatás, s részünkről annak tudomásulvétele, hogy Mexikó visszahódítja elvesztett területeit, Texast, Új Mexikót és Arizonát. A megállapodás részleteit önre bízzuk.*
>
> *Amint az Egyesült Államok hadba lépése bizonyossá válik, fentiekről a legnagyobb titokban informálja az elnököt, és javasolja neki, hogy a maga kezdeményezéséből szólítsa fel Japánt az azonnali csatlakozásra, és ugyanakkor közvetítsen közöttünk és a japánok között.*
>
> *Szíveskedjék felhívni az elnök figyelmét arra, hogy tengeralattjáróink korlátozás nélküli bevetésével kilátásunk van arra, hogy Angliát néhány hónapon belül fegyverletételre kényszerűsítsük. Kérem a fentiek tudomásulvételét.*
>
> *Zimmermann*

Az angolok nem akarták nyilvánossá tenni a feltörés tényét (hiszen ekkor a németek rájöttek volna, hogy a kódkönyvük nem biztonságos). Úgy tálalták az esetet, mintha egy mexikói titkosügynökük megszerezte volna a már dekódolt változatot. A táviratot átadták az amerikaiaknak, akik 1917. március 1-én a sajtóban publikálták. Kezdetben a közvélemény nem hitt a távirat valódiságában, de amikor azt maga Zimmermann is beismerte, óriási közfelháborodás támadt a németek ellen. Ez az eset illetve a német tengeralattjárók amerikai hajók elleni támadásai végül oda vezettek, hogy Widrow Wilson elnök április 6-án bejelentette az Egyesült Államok brit oldalon való belépését a háborúba.

Az eddig bemutatott módszerek mindegyike helyettesítő kód volt. Egy szövegegység (legyen az egyetlen betű, vagy több betűből álló tömb) kódolt megfelelője

2.6. ábra. A Zimmermann-távirat



nem függött a szövegegység helyzetétől (csak magától a szövegegységtől). A következő módszereknél pontosan a fordított helyzet áll fenn: a helyettesítés nem függ a szövegegységtől, csupán a szövegen belüli pozíciójától.

2.1.4. Átrendezéses kódok

Mint már említettük, a szigorú értelemeben vett *átrendezéses (keveréses) kód*ok nem cserélik ki az eredeti szöveg betűit. Ezért a kódoló eljárás mindig egy permutáció a betűk pozícióinak halmazán. Ha egy átrendezéses kóddal nem túlságosan hosszú, értelmes szöveget titkosítunk, ez elég könnyen feltörhető lesz az anagrammák keresésének módszerével. (Az anagramma egy adott szöveg valamennyi betűjének átcsoportosításával alkotott új értelmű szöveg.)

Mivel az eredeti szöveg betűit az átrendezéses kódok nem változtatják meg, csak sorrendjüket kuszálják össze, a betűk százalékos eloszlása a kódolt üzenet esetében pontosan ugyanaz, mint a sima szövegnél. Ha tehát egy ismeretlen módszerrel titkosított szöveg betűinek százalékos eloszlását vizsgáljuk, és azt találjuk, hogy nagyon közel áll az adott nyelvre jellemző százalékos eloszláshoz, akkor nagy a valószínűsége annak, hogy átrendezéses kóddal állunk szemben.

Megjegyezzük, hogy az átrendezéses kódokat még *transzpozíció*knak is szokták nevezni.

2.1.4.1. A cikcakk-rejtjel

A *cikcakk-rejtjel* neve elárulja a módszert is, ahogyan az eredeti üzenet betűit keverjük: cikcakkosan írjuk az eredeti üzenetet, majd vízszintesen olvassuk a betűket.

Kulcs: Egy $k \in \mathbb{N}$, $k \geq 2$ természetes szám.

Kódolás: A nyílt szöveg betűit egy szakaszonként k betűből álló cikcakkos tört vonal mentén leírjuk, majd a vízszintes sorok mentén kiolvasott betűket egymás mellé írjuk.

2.1.10. példa. *Legyen az eredeti üzenet a következő „angol” mondat:* Tape at war you one a wash on[1]. *Ha a kulcs $k = 4$, akkor az üzenetet így írjuk le:*

T						**W**						**O**						**S**			
	A				**T**		**A**				**U**		**N**				**A**		**H**		
		P		**A**				**R**		**O**				**E**		**W**				**O**	
			E						**Y**						**A**						**N**

Vízszintesen olvasva a következő kódolt szöveget kapjuk (a könnyebb olvashatóság

[1] Magyarul: *Tépett varjú van a vason.*

kedvéért ötösével csoportosítottuk a betűket): **TWOSA TAUNA HPARO EWOEY AN**.

Dekódolás: A dekódolást legkönnyebb egy négyzetes lapon vagy egy táblázat segítségével végezni. Mivel tudjuk a k kulcsot, azt is tudjuk, hogy milyen alakja lesz cikcakkos vonalnak (és a kódolt szöveg hossza alapján azt is, hogy a vonal milyen hosszú lesz). Megjelöljük azokat a négyzeteket, ahova betű fog kerülni, majd vízszintes soronként haladva kitöltjük a megjelölt négyzeteket a kódolt szöveg egymás utáni betűivel. Így visszakapjuk a kódolásnál használt cikcakkos betűtáblázatot, és olvashatóvá válik a nyílt üzenet.

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadás) Könnyen ki lehet próbálni az összes kulcsot, hiszen maximálisan $\left\lfloor \frac{n}{2} \right\rfloor - 1$ lehetséges kulcs van (ha ennél nagyobb a kulcs, akkor a üzenet eleje változatlan marad).

2.1.4.2. Az útvonal-kód

Az *útvonal-kód* valójában nem egy adott módszer, hanem egy egész rejtjel-család. A küldő fél az üzenet betűit először egy négyzet (de lehet téglalap, rombusz stb.) alakú rácsba írja egy általa válaszott sorrendben (például soronként lefele haladva, soron belül balról jobbra, vagy oszloponként jobbról balra haladva, oszlopon belül fentről lefele) majd egy másik módszer (útvonal) szerint kiolvassa a rácsból és egymás után írja a betűket. Ez jóval nagyobb kulcsteret kínál, mint a cikcakk-rejtjel.

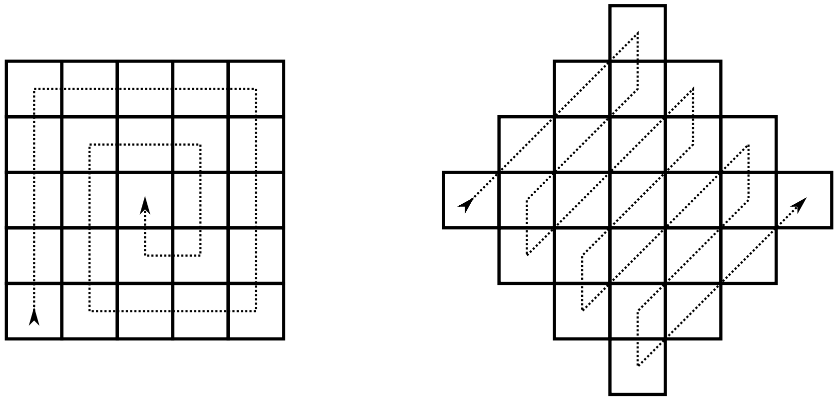
Kulcs: A kulcs gyakorlatilag magának a titkosítási módszernek az ismerete: a fogadó félnek tudnia kell a rács alakját, azt hogy milyen sorrendben volt beírva az eredeti szöveg, és milyen „útvonalon” kell kiolvasni azt a rácsból.

2.1.11. példa. *Egy kulcs lehet a következő információ: a rács négyzet alakú, a nyílt szöveg soronként lefele és soron belül balról jobbra volt beírva, a kódolt szöveget a következő útvonalon olvasd ki: spirálisan befele haladva a rács bal alsó sarkától indulva felfele. Egy másik kulcs: a rács rombusz alakú, nyílt szöveg oszloponként balról jobbra, oszlopon belül lentről felfele, kódolt szöveg a rombusz bal oldali csúcsától indulva felfele, kígyózva. Mindkét esetben a rács méretét ki lehet következtetni az üzenet hosszából. A 2.7 ábrán látható a két kulcs.*

Kódolás: A küldő fél beírja a rácsba az erdeti üzenet betűit a kulcsban foglalt utasításoknak megfelelően.

Dekódolás: A titkos üzenet címzettje felépíti ugyanazt a rácsos betűstruktúrát amit a kódoló is használt, majd kiolvassa a rácsból az üzenetet.

2.7. ábra. Két lehetséges kulcs az útvonal-kódhoz



2.1.12. példa. *Legyen a titkosítandó üzenet a következő szintén „angol” mondat:* If you one door how I mace, I’m egg[2]! *Használjuk a 2.1.11 példa első kulcsát, ahogy az a 2.7 ábrán is látható. A kitöltött rács tehát:*

I	**F**	**Y**	**O**	**U**
O	**N**	**E**	**D**	**O**
O	**R**	**H**	**O**	**W**
I	**M**	**A**	**C**	**E**
I	**M**	**E**	**G**	**G**

A kódolt szöveg (ötösével csoportosítva): **IIOOI FYOUO WEGGE MMRNE DOCAH**.

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadás) A kulcstér itt jóval nagyobb, mint a cikcakk-kód esetében, de (mint a keveréses módszerek általában) ez a rejtjel is anagrammák keresésével támadható. A kódolt szöveget megpróbáljuk különböző szabályos alakú rácsokban felírni (négyzet, rombusz, téglalap stb.) és a rácson belül helyezgetni a betűket úgy, hogy minél több értelmes szót kapjunk. Ha van sejtelmünk az üzenet tartalmáról, ez sokat segíthet.

2.1.4.3. Az oszlopos transzpozíció

Az *oszlopos transzpozíció* nagyban hasonlít egy partikuláris útvonal-kódra, de ebben az esetben a rács és annak kitöltési módja adott, és a kiolvasási sorrend egy jelszótól függ.

[2]Magyarul: *Ifjú vándor, hová mész, állj meg!*

Kulcs: Lehet egy különböző betűkből álló, maximálisan n hosszúságú szó, ahol n a használt ábécé betűinek száma (ha kulcsnak mégis egy olyan értelmes szót választunk, amiben vannak betűismétlések, akkor az ismétlődő betűket kihagyjuk). Ha ennél hosszabb kulcsot szeretnénk használni, akkor a $\{1, 2, \ldots, \ell\}$ halmaz egy permutációja lesz ez, ahol $\ell > n$.

Kódolás: A rács téglalap alakú, a kulcs hossza határozza meg az oszlopok számát, az üzenet hossza pedig a sorokét. Ha a kulcs hossza n, a szövegé pedig s, akkor a szöveg betűit beírjuk egy $\left\lceil \frac{s}{n} \right\rceil \times n$ méretű rácsba (soronként, balról jobbra). Ezután a kulcsszót a rács fölé írjuk úgy, hogy minden egyes betűje egy-egy oszlop fölé kerüljön, majd oszloponként kiolvassuk és egymás mellé írjuk a rácsban levő betűket. A kulcsszó fogja meghatározni az oszlopok kiolvasásának sorrendjét. Ha a kulcs egy szó (tehát az egyes oszlopok fölé betűk vannak írva), akkor betűit ábécésorrendben véve megvan az oszlopokra vonatkozó sorrend is. Ha a kulcs egy számsorozat, akkor analóg módon járunk el, a számok növekvő sorrendje alapján. Ha $n \nmid s$ (tehát a téglalap alakú rács nem telik meg teljesen az eredeti üzenet betűivel) akkor az üres négyzeteket **X** betűkkel tölthetjük ki, de akár üresen is hagyhatjuk őket.

Dekódolás: A kulcs és a rejtjelezett szöveg ismeretében oszloponként fel lehet építeni a rácsot, amiből az eredeti üzenet kiolvasható.

2.1.13. példa. *Legyen az üzenet a következő:* „Reginam occidere nolite timere bonum est si omnes consentiunt ego non contradico[3].” *A jelszó:* **HONORIFICABILITUDINITAS**. *A betűismétlődéseket ki kell hagynunk, tehát a módszer által használt kulcs ez lesz:* **HONRIFCABLTUDS**. *Kulcsszavunk tehát 14 betűs, az üzenet pedig 69, ezért a rács mérete* $\left\lceil \frac{69}{14} \right\rceil \times 14 = 5 \times 14$.

6	10	9	11	7	5	3	1	2	8	13	14	4	12
H	**O**	**N**	**R**	**I**	**F**	**C**	**A**	**B**	**L**	**T**	**U**	**D**	**S**
R	E	G	I	N	A	M	O	C	C	I	D	E	R
E	N	O	L	I	T	E	T	I	M	E	R	E	B
O	N	U	M	E	S	T	S	I	O	M	N	E	S
C	O	N	S	E	N	T	I	U	N	T	E	G	O
N	O	N	C	O	N	T	R	A	D	I	C	O	X

A kulcs fölé írt számok jelzik a kulcsszó betűinek ábécésorrendjét, ezt a sorrendet kell követni, amikor oszloponként kiolvassuk a betűket a rácsból. A kódolt szö-

[3]*Meràniai János esztergomi érsek híres kétértelmű levele (1293-ból), amit az összeesküvést szövő magyar főurakhoz írt. Jelentése:* A királynét megölni nem kell félnetek jó lesz ha mindenki egyetért én nem ellenzem.

veg tehát a következő: **OTSIR CIIUA METTT EEEGO ATSNN REOCN NIEEO CMOND GOUNN ENNOO ILMSC RBSOX IEMTI DRNEC**.

Kriptoanalízis: (csak a kódolt üzenet ismeretén alapuló támadás) Az oszlopos transzpozíció gyenge pontja, hogy próbálgatással megtalálható a rács oszlopainak hossza, és ha a kódolt szöveg értelmes, akkor anagrammák keresésével innen már nem nehéz (a kulcs ismerete nélkül) kitalálni az oszlopok helyes sorrendjét.

2.1.4. megjegyzés. *Egyszerű keveréses rejtjelet alkalmazni rövid, értelmes üzenet titkosítására – mint láttuk – nem biztonságos (az anagrammák keresésének módszere miatt). Ha a kódfeltörő nem is jön rá a teljes kulcsra, és nem sikerül az egész üzenetet megfejtenie, akkor is érthetővé válhatnak az üzenet foszlányai, amikből következtetni lehet eredeti tartalmára.*

Például, a biztonság növelése érdekében, az oszlopos transzpozíció esetében a már kódolt szöveget még egyszer kódolják, ugyanazzal a módszerrel. Ezt a módszert nevezzük dupla oszlopos transzpozíció*nak. Az első világháború elején a németek dupla oszlopos transzpozíciót használtak a titkosításra, de a francia hírszerzés kriptográfusai még így is néhány nap alatt rutinszerűen megfejtették üzeneteiket, ezért a német hadsereg lemondott a módszer használatáról.*

2.1.5. megjegyzés. *Az anagrammák keresése az egyszerű átrendezéses kódok gyenge pontja, de ugyakkor ebben egy érdekes lehetőség is rejlik: „az ellenség megtévesztése". A titkosan kommunikáló felek például előre megyegyezhetnek abban, hogy bármilyen hosszú üzenetet is küldjenek egymásnak, abból csak az első n betűt kell figyelembe venni. Az első n betű után pedig értelmes de megtévesztő szavakat illeszthetnek, amivel tévútra vezethetik az anagrammázással próbálkozú ellenséget. Egy ilyen üzenet lehet például: „**Támadás ma éjjel** elhalasztva, visszavonulunk." Ha előre megyeznek abban, hogy csak az első* $n = 14$ *betűt veszik figyelembe, akkor az üzenet tartalma egészen más lesz. Az anagrammák keresésével próbálkozó ellenség erről a megegyezésről nem tudhat, és annak ellenére, hogy talán az egész üzenetet visszafejti, mégis mást ért belőle.*

2.1.5. Rejtjelező gépek

A rejtjelezés (vagy kódolás) folyamata egyik módszer esetében több, a másikban kevesebb, de minden esetben nagy részben gépies munkát jelent. Ezért a történelem folyamán számtalan rejtjelező eszköz készült. Fejlődésük nyomon követhető a legegyszerűbbektől (például a már bemutatott Caesar-kerék), a XX. században használt mechanikai elven működő gépeken keresztül a számítógépes titkosítási rendszerekig (bizonyos értelemben egy számítógépen implementált titkosító algoritmus is rejtjelező gépnek tekinthető, hiszen a kódolás illetve dekódolás folyamatát automatizálja).

Bár a kriptográfia története szempontjából érdekesek, nem célunk ezeknek az eszközöknek a felsorolása. Meg kell említenünk azonban néhány, nagyrészt mechanikus *rejtjelező gép*et, a XX. század történelmében játszott kulcsfontosságú szerepük miatt.

A mechanikus rejtjelező gépek – bár működésük részleteiben természetesen eltérnek egymástól – lényeges tulajdonságaikban megegyeznek. Általában írógéphez hasonlító, dobozban vagy tokban hordozható készülékek voltak, részletes használati utasítással.

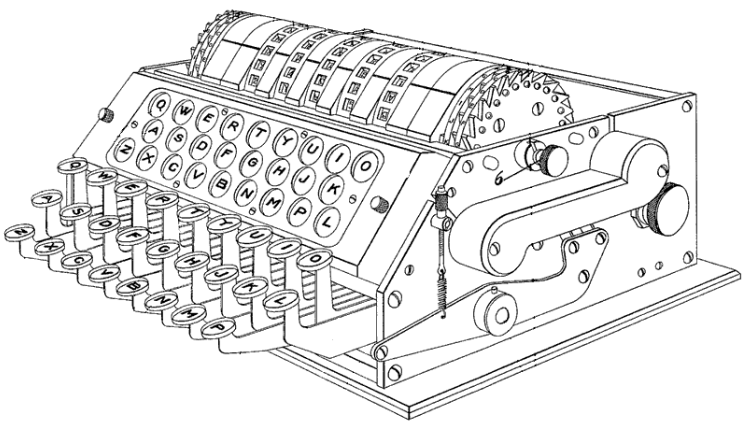
Kulcs: Egy 5-6 betűs szó (de lehet egy szám is). Általában annyi betűből (vagy számjegyből) állt a kulcs, ahány „betűs tárcsa” volt a gépen a kulcs beállítására. A kezelő személy (operátor) a titkos kulcs alapján először beállította a gépet a rajta található betűs tárcsák vagy gombok segítségével egy kezdeti állapotba. Egyes modellek ezenkívül rendelkeztek néhány cserélhető alkatrésszel (pl. különböző módon kilyukasztott lemezekkel vagy tárcsákkal, különböző módon kapcsolható huzalokkal stb.), amivel növelni lehetett a biztonságot. A kulcsszón kívül az is számított, hogy ezek közül hányat, melyeket (és esetleg milyen sorrendben) kellett betenni a gépbe. Tehát ezeknél a modelleknél e cserélhető alkatrészek is a kulcs részét képezték.

Kódolás: Miután az operátor a megfelelő módon kezdeti állapotba hozta a gépet, egy kapcsoló segítségével kódoló üzemmódba helyezte, majd elkezdte beírni a nyílt üzenetet. A betűk beírása modelltől függően betűs tárcsa vagy billentyűzet segítségével történt. A tárcsás modelleknél az üzenetbeíró tárcsát az üzenet egyes betűinek megfelelő pozícióba csavarták, majd megforgattak egy kart (vagy lenyomtak egy gombot), aminek következtében a gép papírtekercsre nyomtatta az adott betűnek megfelelő kódolt betűt. A billentyűs modelleknél már egyszerűbben és gyorsabban ment e művelet, hiszen a rejtjelező gép használata (a kulcs beállítását és a funkció kiválasztását kivéve) olyan volt, mint egy egyszerű írógép esetében: az operátor a billentyűzeten „gépelte be” az üzenetet, a gép pedig közben papírra nyomtatta a kódolt üzenetet.

Dekódolás: Az operátor a kulcsnak megfelelően beállította a gépet majd a dekódoló üzemmódba kapcsolta. Ezután a dekódolás folyamata ugyanúgy ment végbe, mint ahogyan azt a kódolásnál láttuk, persze azzal a különbséggel, hogy az operátor a rejtjelezett üzenetet „gépelte be”, a rejtjelező gép pedig az eredeti üzenetet nyomtatta papírra.

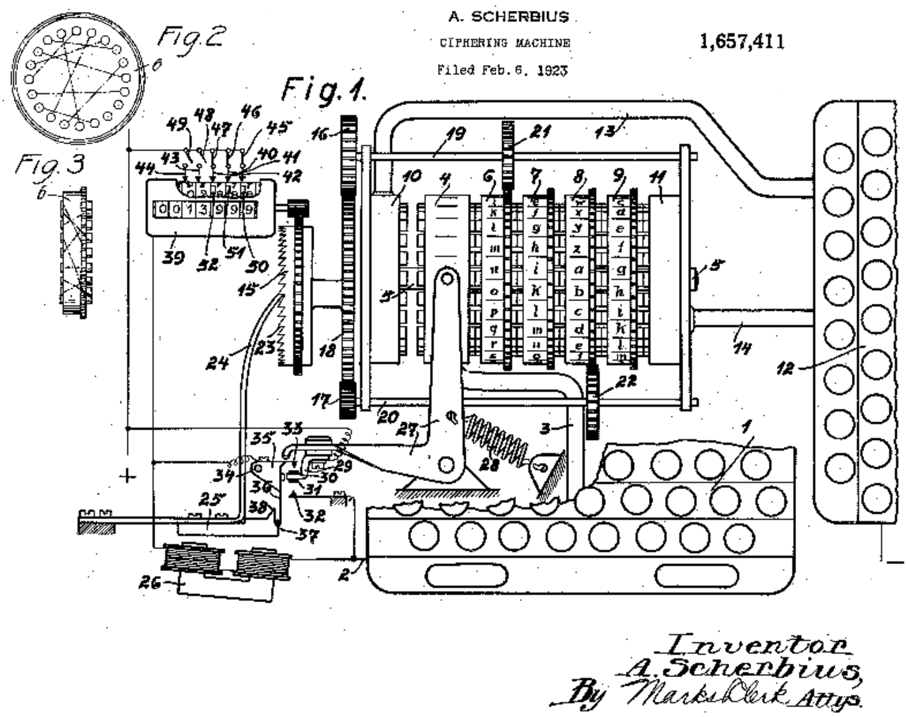
Voltak olyan modellek is, melyek nem nyomtattak papírra, hanem csak kijelezték, hogy mibe kódolódik vagy dekódolódik az aktuális betű: vagy egy kijelző tárcsa fordult el mutatva a betűt, vagy pedig a pedig a gép rendelkezett egy kijelző résszel, amin fel volt tüntetve a használt ábécé összes betűje, és kis elektromos égők gyúltak ki az egyes betűk alatt, jelezve azokat. A 2.8 képen egy ilyen (1928-ban szabadalmaztatott) elektromos-mechanikus hibrid rejtjelező gép vázlatos rajza látható.

2.8. ábra. Rejtjelező gép



Említettük már ezen rejtjelező gépek kiemelkedő történelmi fontosságát. Ezt legjobban talán az Enigma rejtjelező gép (és története) támasztja alá. 1926-ban Németországban megjelent a kereskedelmi forgalomban az Arthur Schrebius által szabadalmaztatott Enigma (a 2.9 képen látható egy korai Enigma modell vázlatos rajza, ahogy az a szabadalomban található). Az Enigmát eredetileg üzleti célokra tervezték, ám a német hadsereg kifejleszetette ennek katonai változatát, amit 1928-tól használtak. Lengyelország veszélyesnek találta Németország külpolitikáját, ezert titkosszolgálata az első világháborút követően folyamatosan lehallgatta és megfejtette a német üzeneteket. Az Enigma bevezetését követően azonban az addig használt módszerek (mint például a betűgyakoriság-vizsgálat) nem vezettek eredményre. A lengyelek külön kutatócsoportot hoztak létre az Enigma feltörése érdekében. A csoportban részt vett három kiváló fiatal matematikus a poznani egyetemről: Marian Rejewski (1905-1980), Jerzy Rúzycki (1907-1942) és Henryk Zygalski (1906-1978). A katonai Enigmák által generált, lehallgatott titkos üzetek beható tanulmányozása és egy franciáknak dolgozó kém által szolgáltatott plusz információk segítségével (fényképek a gépről, használati utasítások, és két hónapos periódust lefedő napi kulcsszavak listája) Rejewskinek sikerült 1932-ben megalkotni az Enigma matematikai modelljét. Ezután Rúzyckival és Zygalskival közösen módszert dolgoztak ki az Enigmák által kódolt német üzenetek rutinszerű feltörésére, még egy mechanikus gépet is építettek, mely a dekódolást automatizálta. A németek nem szereztek tudomást kriptorendszerük feltöréséről, ennek ellenére 1939-ben új biztonsági elemekkel fejlesztették tovább az Enigmát. A működési elv (és így a megtalált matematikai modell) továbbra is érvényes maradt, viszont a lengyelek a „feltörő gépet” már nem tudták használni. Új gép építése túl költséges lett volna, ezért a lengyel titkosszolgálat átadta az Enigmáról szóló információkat a szövetséges Angliának. Az angol titkosszolgálat aztán (az amerikaiakkal együtt)

2.9. ábra. Az Enigma rejtjelező gép vázlatos rajza



továbbfejlesztette és megépítette az új, sokkal erősebb Enigma-feltörő gépet („Bombe”), amivel aztán képesek voltak megfejteni a német hadsereg legtitkosabb üzeneteit is a második világháború alatt. Történészek szerint az Enigma feltörése legalább két (egyesek szerint négy) évvel rövidítette meg a második világháborút, és döntő mértékben járult hozzá a hitlerista Németország legyőzéséhez.

A következőkben bemutatjuk egy másik, egyszerűbb rejtjelező gép matematikai hátterét.

2.1.6. A C-36-os rejtjelező gép matematikai háttere

A *C-36-os rejtjelező gép* megszerkesztője a svéd Boris C. W. Hagelin (1892-1983) volt. Franciaország kérte fel őt egy hordozható és biztonságos (katonai célokra használható) rejtjelező gép szerkesztésére. A C-36 utódja az M-209 Converter, melyet az 1950-es évekig használták az amerikaiak.

A C-36 matematikai modelljének alapja két mátrix:

1. A *cipelő mátrix* (lug matrix) $M \in M_{6,27}(\mathbb{Z}_2)$ egy 6×27-es bináris mátrix azzal a tulajdonsággal, hogy minden oszlopban legfennebb két darab 1-es van.

2. A *lépcsőforma* (step figure). Ennek első sora 17, második sora 19, harmadik sora 21, negyedik sora 23, ötödik sora 25, hatodik sora pedig 26 dimenziós bináris vektor. Ha ezeket a sorvektorokat $s_1, s_2, \dots, s_6$-tal jelöljük, akkor a lépcsőforma $\begin{matrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{matrix}$ lesz.

A lépcsőformából tudunk generálni egy $6\times\infty$-es bináris mátrixot a sorok egymásutáni ismétlésével:

$$N = \begin{pmatrix} s_1 & s_1 & s_1 & \dots \\ s_2 & s_2 & s_2 & \dots \\ \vdots & \vdots & \vdots & \\ s_6 & s_6 & s_6 & \dots \end{pmatrix}.$$

Legyen $v_i = (o_N^i)^t$, vagyis N i-edik oszlopának transzponáltja, ami egy 6 dimenziós bináris sorvektor.

2.1.14. példa. *Ha a lépcsőforma*

1	2	3	4	5	...	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	...
0	1	1	0	1	…	1	0	0	1	1	0	1	…									
1	0	0	1	1	…	0	1	1	0	1	0	0	1	1	…							
0	0	1	0	1	…	1	1	0	0	1	0	0	0	1	0	1	…					
0	1	0	0	0	…	0	1	0	1	0	1	0	1	0	1	0	0	0	…			
0	0	1	0	0	…	1	1	0	1	1	1	1	1	1	0	0	0	1	0	0	…	
0	1	0	1	0	…	1	0	0	1	0	0	1	0	0	0	1	0	1	0	1	0	…

akkor neki megfelelő sorozatok például: $v_1 = (010000)$, $v_{17} = (011110)$, $v_{18} = (010000)$, $v_{19} = (100111)$, $v_{20} = (111010)$.

A lépcsőformánál lehetőleg törekedni kell arra, hogy ugyanannyi egyes legyen, mint nullás, hiszen a cél az, hogy a v_i vektorok hosszabb sorozatai lehetőleg ne ismétlődjenek. Vegyük észre, hogy mivel 17, 19, 21, 23, 25 és 26 páronként relatív prímek, $v_i = v_{i+p}$, ahol $p = 17 \cdot 19 \cdot 21 \cdot 23 \cdot 25 \cdot 26 = 101405850$ viszonylag nagy periódus, tehát a $v_1, v_2, \dots, v_p$ sorozat véletlenszerűnek tűnhet.

Kulcs: $k = (M, N)$ mátrixpár.

Kódolás: Betűnként történik. Ha P_i a nyílt szöveg i-edik betűje, akkor $C_i = h_i - P_i - 1$, ahol C_i a kódolt szöveg i-edik betűje, h_i pedig $v_i M$-ben a nem nulla elemek száma.

Dekódolás: $P_i = h_i - C_i - 1$.

Kriptoanalízis: Vegyük észre, hogy ez a rendszer tulajdonképpen egy ellentétes előjelű, $p = 101405850$ periódusú Vigenère-rejtjel.

2.2. Modern kriptorendszerek

A modern szimmetrikus rendszerek két osztályát különböztetjük meg: beszélünk *folyamtitkosító*król (*folyamrejtjel*ekről), illetve *tömbtitkosító*król (*tömbrejtjel*ekről). A folyamtitkosítók az eredeti szöveg egységeit (betűit, bitjeit vagy bájtjait) egyenként titkosítják, míg a tömbtitkosítók adott méretű adattömböket titkosítanak.

2.2.1. Folyamtitkosítók

2.2.1.1. Véletlen átkulcsolás

G. Vernam és J. Mauborgue alkotta meg 1917-ben. A módszer neve angolul *one-time pad*. Lényege az, hogy az eredeti szöveget szövegegységenként (karakterenként, betűnként, bitenként) összeadjuk modulo n a szöveggel azonos hosszúságú kulccsal. Itt n a használt karakterek száma: angol ábécé esetében $n = 26$, számjegyek esetében $n = 10$, bitek használata esetében $n = 2$ (tehát az összeadás modulo 2 egyszerűen az XOR bináris művelet lesz).

Kulcs: Az eredeti szöveggel azonos méretű, lehetőleg véletlenszerű szövegegység-sorozat, amelyet csak egyszer használunk fel (innen az angol elnevezés). Az ilyen kulcsot még *kulcsfolyam*nak is nevezzük. Jelöljük k_i-vel a k kulcs i-edik egységét.

Kódolás: $E_k(P) = C$, $C_i = (P_i + k_i) \bmod n$, ahol P_i az eredeti szöveg i-edik egysége, C_i pedig a titkosított szöveg i-edik egysége.

Dekódolás: $D_k(C) = P$, $P_i = (C_i - k_i) \bmod n$.

2.2.1. példa. *Angol ábécé esetében (*$n = 26$*):*

$$\begin{array}{rcl} P = \mathbf{T\ I\ T\ K\ O\ S} & = & 19\ \ 8\ \ 19\ 10\ 14\ 18 \\ k = \underline{\mathbf{A\ C\ Z\ D\ P\ Q}} & = & \underline{0\ \ 2\ \ 25\ \ 3\ \ 15\ 16} \\ C = \mathbf{T\ K\ S\ N\ D\ I} & = & 19\ 10\ 18\ 13\ \ 3\ \ 8 \end{array} \quad + \ (\text{mod } 26)$$

2.2.2. példa. *A szövegegységek bitek (*$n = 2$*):*

$$\begin{array}{rl} P = & 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ k = & \underline{1\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0} \\ C = & 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \end{array} \quad + \ (\text{mod } 2)$$

2.2.3. példa. *A szövegegységek a (decimális) számjegyek ($n = 10$):*

$$\begin{array}{rl} P = & 0\ 9\ 5\ 1\ 7\ 8\ 3\ 2\ 1\ 6\ 5\ 2\ 4 \\ k = & \underline{1\ 2\ 9\ 7\ 4\ 8\ 5\ 2\ 3\ 9\ 1\ 0\ 6} \\ C = & 1\ 1\ 4\ 8\ 1\ 6\ 8\ 4\ 4\ 5\ 6\ 2\ 0 \end{array} + \pmod{10}$$

Kriptoanalízis: 1940-ben C. Shannon bebizonyította, hogy a véletlen átkulcsolás véletlenszerű kulcsok egyszeri használatával *tökéletes titkosítás*t valósít meg. Ez nagyjából azt jelenti, hogy a kódolt szöveg nem árul el semmi információt az eredeti szövegről. Mind a mai napig is ez az egyetlen kriptorendszer, amely tökéletes titkosítást nyújt. Ha a kulcsot többször használjuk fel, akkor a rendszer sebezhetővé válik a nyílt szöveg ismeretén alapuló támadással szemben. Valóban, egyetlen (P, C) pár ismerete $k_i = (C_i - P_i) \bmod n$ kiszámítása által (minden i-re) elárulja a kulcsot. Tehát rendkívül fontos, hogy egy bizonyos kulcs felhasználása egyszeri legyen. Ennek, illetve a kulcs véletlenszerűségének fontosságát fedezte fel J. Mauborgue (G. Vernam eredetileg megengedte a kulcsok újrahasznosítását).

Kulcskezelés: Láttuk, hogy ideális esetben a véletlen átkulcsolás a nyílt szöveggel azonos hosszúságú, egyszeri, véletlen kulcsot kellene, hogy használjon. A probléma az, hogy egy ilyen kulcs generálása, kezelése a kommunikáló felek közötti kicserélése és használat utáni megsemmisítése rendkívül költséges folyamat lehet, pénzben és időben egyaránt. Ez a véletlen átkulcsolás nagy hátránya. A történelem folyamán persze különböző ötletek születtek a kulcskezelés könnyítésére. Például a *pad* szó a módszer angol megnevezéséből arra utal, hogy sok esetben az egyszeri kulcsok egy jegyzettömb egy-egy oldalára voltak írva, és használatuk után kitépték a lapot a tömbből, majd elégették (sok esetben a kulcsok szivarpapírra voltak írva). Persze a jegyzettömbre nagyon kellett vigyázni, nehogy ellenséges kezekbe kerüljön. Könnyebb kulcsgenerálásra is születtek különféle ötletek, ilyenek például:

1. Tételezzük fel, hogy a kommunikáló felek rendelkeznek egy adott könyv identikus példányaival (Biblia, valamilyen évkönyv vagy regény stb.). Nevezzük kezdeti kulcsnak a könyv egy oldalszámát, szövegsorszámát és egy betűpozíciót. A $k_0 = (183, 20, 11)$ például azt fogja jelenteni, hogy a 183. oldal 20. sorának 11. betűje. A kulcsfolyam tehát a könyv szövege, az előbb megjelölt betűtől a könyv végéig.
2. *Bruce Schneier Solitaire algoritmusa*. Az algoritmust Neal Stephenson *Cryptonomicon* című regényéhez készítette Bruce Schneier (a regényben a módszer Pontifex néven szerepel). Lényegében egy „kézi” használatra tervezett, véletlenszerű számsorozatot generáló módszerről van szó.

Ahhoz, hogy egy titkosítási rendszert kapjunk, ötvözni kell véletlen átkulcsolással (a Solitaire-rel generált véletlenszerű számsorozatot kulcsfolyamként lehet használni). A Solitaire használatához nem kell semmi egyéb, mint egy 54 lapos francia kártya: a pókerben használatos 52 kártyalap és két dzsóker. Fontos, hogy a két dzsókert megkülönböztessük egymástól, ezért a leírásban úgy fogunk hivatkozni rájuk, mint fehér dzsóker, vagyis ☺, és fekete dzsóker, vagyis ☻. A kártyalapoknak számokat feleltetünk meg, a színeket a bridzs kártyajátéknál használatos sorrendben vehetjük (treff, káró, kőr és pikk), az ugyanolyan színű kártyák közül első az ász, majd egymást követik növekvő sorrendben a kettestől a királyig:

A♣	2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣	Q♣	K♣
1	2	3	4	5	6	7	8	9	10	11	12	13
A♦	2♦	3♦	4♦	5♦	6♦	7♦	8♦	9♦	10♦	J♦	Q♦	K♦
14	15	16	17	18	19	20	21	22	23	24	25	26
A♥	2♥	3♥	4♥	5♥	6♥	7♥	8♥	9♥	10♥	J♥	Q♥	K♥
27	28	29	30	31	32	33	34	35	36	37	38	39
A♠	2♠	3♠	4♠	5♠	6♠	7♠	8♠	9♠	10♠	J♠	Q♠	K♠
40	41	42	43	44	45	46	47	48	49	50	51	52
☺	☻											
53	53											

Mindkét dzsóker értéke 53. Először is az 54 lapot tartalmazó kártyapaklit kezdeti állapotba hozzuk. A kártyapakli kezdeti állapota (a lapok sorrendje) lesz a k_0 kezdeti kulcs. Ahhoz, hogy a felek kommunikálni tudjanak, a pakli kezdeti állapota meg kell, hogy egyezzen mindkettőjüknél. Ezért, meg kell egyezniük egy módszerben, amely szerint a kártyákat kezdeti sorrendbe helyezik. Ezután kezdődhet a k kulcsfolyamot alkotó k_i számok előállítása. Egy ilyen szám generálása a következő módon történik:

(a) Kezünkbe vesszük a megkevert kártyapaklit. Megkeressük benne a fehér dzsókert, és kicseréljük az alatta levő kártyalappal. Ha a fehér dzsóker a pakli legalsó kártyalapja, akkor beszúrjuk felülről az első kártyalap után.

(b) Megkeressük a fekete dzsókert, és két kártyalappal lennebb szúrjuk be a pakliba. Ha a fekete dzsóker a legalsó kártyalap, akkor felülről a második kártyalap után szúrjuk be. Ha a fekete dzsóker a pakli alján az utolsó előtti kártyalap, akkor a legfelső kártyalap alá kerül. Fontos, hogy ezt az első két lépést ebben a sorrendben hajtsuk végre. Ha például az első lépés előtt a kártyapakli így néz ki (a sorban az első kártyalap a legfelsőt jelöli):

akkor a második lépés végrehajtása után a következőképpen módosul:

Ha viszont eredetileg a kártyapakli így néz ki:

akkor a második lépés végrehajtása után a következőképpen módosul:

(c) Cseréljük fel az első dzsóker előtti kártyalapokat a második dzsóker utáni kártyalapokkal. Ennél a lépésnél nem számít a dzsókerek színe, csak az, hogy a kártyapakli tetejétől számítva melyik az első illetve a második dzsóker. A kártyapaklit gyakorlatilag három részre osztjuk, a középső rész első és utolsó kártyája egy-egy dzsóker, az első részt pedig felcseréljük a harmadikkal, anélkül, hogy a kártyák helyzete változna egy-egy részen belül. Tekintsük például a következő kártyapaklit:

ami a harmadik lépés végrehajtása után így módosul:

Ezt a lépést könnyű úgy elvégezni, hogy a kártyapaklit kezünkben tartva, egy-egy ujjunkat beszúrjuk az első dzsóker elé, illetve a második dzsóker után, majd az így leválasztott első és harmadik részt a másik kezünkkel kicseréljük. Ha pakli első és/vagy utolsó kártyája éppen egy dzsóker, akkor is a szabály szerint járunk el (az első és/vagy az utolsó rész nulla kártyalapot tartalmaz). Legyen a kártyapakli a következő:

¦ ☻ 5♠ 8♠ … 46 kártyalap … 7♥ A♣ ☺ ¦ 3♦ 9♣

A lépés elvégzése után ezt kapjuk:

3♦ 9♣ ¦ ☻ 5♠ 8♠ … 46 kártyalap … 7♥ A♣ ☺ ¦

A következő példában az első és utolsó kártya is dzsóker, tehát a pakli ennél a lépésnél nem módosul:

☻ 5♠ 8♠ … 48 kártyalap … 7♥ A♣ ☺

(d) Megnézzük a legalsó kártyalapot, majd felülről annyi kártyalapot számlálunk, amennyi a legalsó kártyalap számértéke (az egyes kártyalapok számértékét a már bemutatott módon állapítjuk meg), majd itt „törjük” a paklit úgy, hogy a legalsó kártya a helyén marad. Ha tehát a legalsó kártyalap számértéke $n \in \{1, \ldots, 53\}$, akkor a felülről az első n kártyát kicseréljük a következő $53 - n$ kártyával, a legalsó kártya a helyén marad. Egy példával szemléltetjük ezt a lépést is. Ha a paklink a következő:

Q♣ 5♠ 8♠ 7♥ A♣ K♥ 3♦ ¦ 9♣ … 43 kártyalap … J♠ 10♣ ¦ 7♣

akkor ennek a lépésnek az elvégzése után ezt fogjuk kapni:

9♣ … 43 kártyalap … J♠ 10♣ ¦ Q♣ 5♠ 8♠ 7♥ A♣ K♥ 3♦ ¦ 7♣

Vegyük észre, hogy az utolsó kártyalap nem mozdul el a helyéről. Ha az utolsó kártyalap dzsóker, akkor a pakli nem módosul ennél a lépésnél. Vigyázni kell arra, hogy a megcserélt részeken belül a kártyalapok sorrendje ne változzon meg. Ezt a lépést legjob úgy elvégezni, hogy mondjuk bal kezünkben tartjuk a kártyapaklit, és jobb hüvelykujjunkkal a jobb kezünkbe átcsúsztatjuk a kellő számú lapot. Ezután bal kezünk kisujjával leválasztjuk a legalsó kártyalapot, és a jobb kezünkben levő részt beszúrjuk a levállasztott utolsó lap és a többi kártyalap közé.

(e) Felülről annyi kártyalapot számlálunk le, amennyi a legfelső kártyalap számértéke, és megnézzük, hogy mi a következő kártyalap (ha mondjuk a legfelső kártyalap értéke $n \in \{1, \ldots, 53\}$, akkor ez az $n + 1$-edik kártyalap). Ennek a kártyalapnak a számértéke lesz a kulcsfolyam eleme. Legyen például a pakli ennél a lépésnél a következő:

J♠ 7♥ 2♠ . . . 44 kártyalap . . . K♣ 9♣ 4♠ $\boxed{\text{A♥}}$ 10♣ 6♦ Q♥

Mivel a legfelső kártyalap a J♠, aminek a számértéke 50, pontosan ennyi kártyalapot számolunk a pakliban (az ötvenedik lap a 4♠), majd a következő lap számértékét kiírjuk egy papírra. Az ötvenedik kártyalap után következő lap a A♥, aminek az értéke 27. Ez a szám lesz tehát a kulcsfolyam eleme. Vegyük észre, hogy ez a lépés nem módosítja a kártyapakliban levő lapok sorrendjét. Ha az első kártyalap éppen egy dzsóker, akkor nem számolunk és nem írunk ki semmit, hanem az első lépéssel folytatjuk.

3. *Álvéletlen számok generálása*. Álvéletlen szám- vagy bitgenerátor alatt egy olyan algoritmust értünk, amely véletlen sorozathoz hasonló számsorozatot vagy bitsorozatot generál. A generált sorozat nem teljesen véletlen, hiszen néhány magnak nevezett kezdeti érték határozza meg egyértelműen. Az egyik legjobb (véletlenhez legközelebb álló) álvéletlen bitgenerátor a Blum–Blum–Shub-algoritmus:

   - Generáljunk két nagy p, q prímet úgy, hogy $p, q \equiv 3 \pmod 4$.
   - $n = pq$.
   - Legyen $s \in \{1, \ldots, n-1\}$ egy véletlen mag .
   - $x_0 := s^2 \bmod n$
   - $x_i := x_{i-1}^2 \bmod n$ és $z_i := x_i \bmod 2$ (vagyis x_i paritása)
   - Az álvéletlen bitsorozat: $z_1, z_2, z_3, \ldots$

2.2.2. Tömbtitkosítók

A *tömbtitkosító*k (vagy *tömbrejtjelező*k) olyan szimmetrikus kriptorendszerek, amelyek rögzített méretű szövegegység-tömböket titkosítanak (legtöbbször bittömböket). Például egy 128 bites tömböt egy 128 bites tömbbe kódolnak. Ha az eredeti üzenet hosszabb, mint a tömb rögzített mérete (például 128 bit), akkor valamilyen működési módot használunk (ezekre később visszatérünk).

Tipikus tömbméretek: 64 bit (DES, IDEA). Tipikus kulcsméretek: 56 bit (DES, IDEA), 64, 80, 128 bit (AES, IDEA), 192, 256 bit (AES). 2006-tól 80 bites az a minimális kulcsméret, amely biztonságot nyújt a kimerítő kulcskeresés ellen.

A legtöbb tömbtitkosító úgynevezett iterált tömbtitkosító. Ezekben többször (általában 4-től 32-ig) ismétlődik ugyanaz az f menetfüggvény, iterált bemenetekkel. Részletesebben: $P_i = f(P_{i-1}, k_i)$, ahol $P_0 = P$ az eredeti üzenet, $P_n = C$, ahol n a menetek száma, k_i pedig a k kulcsból generált i-edik segédkulcs. Sok esetben úgynevezett kulcsfehérítést is használnak, amit azt jelenti, hogy $P_0 = P \oplus k_0$ és $C = P_n \oplus k_{n+1}$, ahol k_0 és k_{n+1} újabb segédkulcsok.

Az első igazi tömbtitkosító H. Feistel 1970-ben alkotott LUCIFER algoritmusa, amely a DES alapja lett.

2.2.2.1. A kitöltés és a tömbtitkosítók működési módjai

Ha az eredeti szöveg hosszabb, mint a rögzített tömbméret, akkor, ahogy már említettük, valamilyen *működési (operációs) mód*ot használunk, amely meghatározza, hogy az egymásutáni tömbök hogyan kódolódnak. Bizonyos működési módoknál szükséges a *kitöltés* (angolul padding), vagyis az eredeti üzenet kiegészítése bizonyos számú bittel úgy, hogy az üzenet bithossza a rögzített tömbméret egész számú többszöröse legyen. Tipikus kitöltési módok:

1. 0 értékű bitekkel pótolunk;

2. (DES módszer) 1 értékű bit, majd 0 értékű bitek;

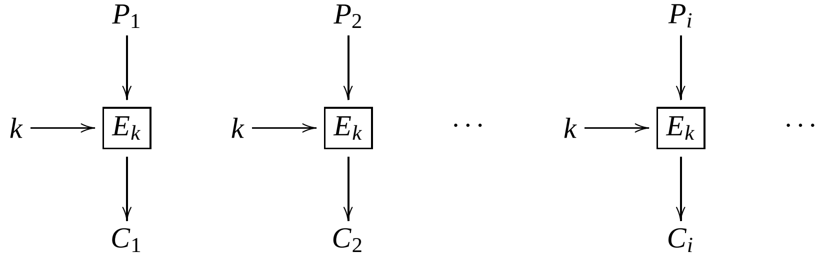
3. (Schneier, Ferguson) n bájttal egészítjük ki az üzenetet, melyek értéke n;

A továbbiakban néhány működési módot vázolunk fel. P_i-vel jelöljük az eredeti üzenet i-edik tömbjét, C_i-vel jelöljük a kódolt üzenet i-edik tömbjét, k a kulcs, E_k a titkosítási eljárás, D_k pedig a dekódoló eljárás. A továbbiakban $\oplus$-szal jelöljük az XOR műveletet.

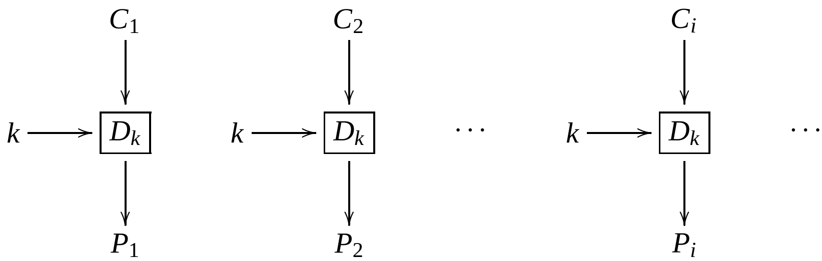
Az ECB mód (Electronic codebook mode)

AZ ECB mód identikus tömböket identikus tömbökbe kódol, (ha $P_i = P_j$, akkor $C_i = C_j$), ami nem takarja el eléggé az eredeti adatstruktúrát. Ezért ezt a működési módot manapság már nem javasolják.

Kódolás: $C_i = E_k(P_i)$, minden i tömbre. Grafikusan:



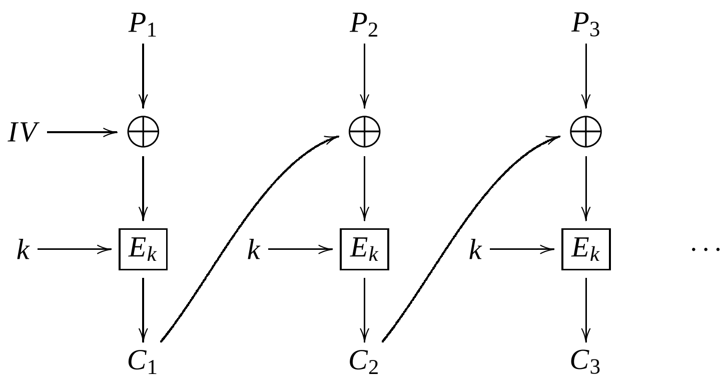
Dekódolás: $P_i = D_k(C_i)$, minden i tömbre. Grafikusan:



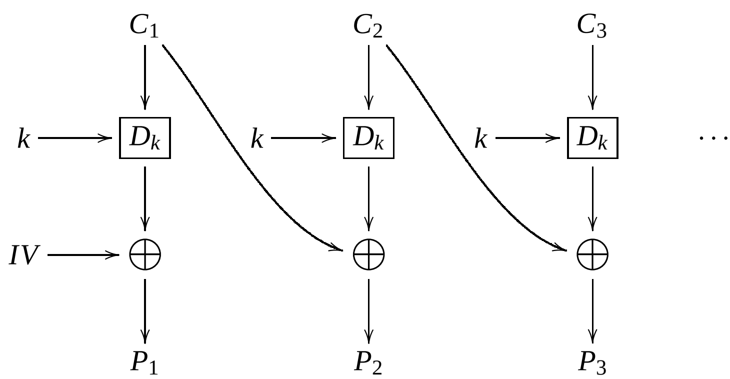
A CBC mód (Cipher-block chaining mode)

Legyen egy IV kezdeti tömbünk.

Kódolás: $C_0 = IV$, minden további C_i tömbre: $C_i = E_k(P_i \oplus C_{i-1})$, $\forall i > 0$. Grafikusan:



Dekódolás: $P_i = D_k(C_i) \oplus C_{i-1}$, $C_0 = IV$. Grafikusan:

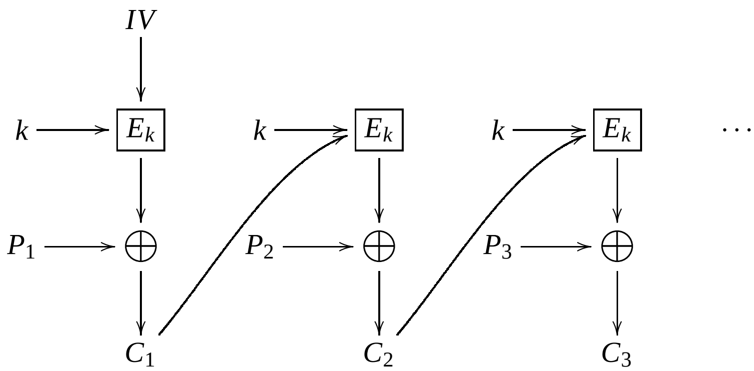


A CBC mód az egyik legelterjedtebb működési mód. Hátránya, hogy a titkosítás szekvenciális (nem tehető párhuzamossá). A dekódolás viszont párhuzamosan is megvalósítható, hiszen P_i csak C_i-től és C_{i-1}-től függ. Egy bit változtatása a P_i tömbben megváltoztatja a C_j-ket, minden $j \geq i$-re, egy bit változtatása C_i-ben megváltoztatja az egész P_i-t és egy darab bitet P_{i+1}-ben.

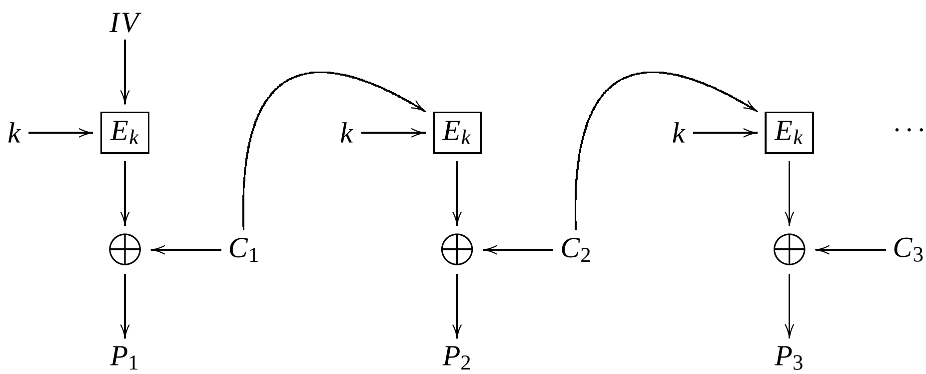
A CFB mód (Cipher feedback mode)

A CFB mód a tömbtitkosítót átalakítja egy folyamtitkosítóvá.

Kódolás: $C_i = E_k(C_{i-1}) \oplus P_i$, $C_0 = IV$. Grafikusan:



Dekódolás: $P_i = E_k(C_{i-1}) \oplus C_i$, $C_0 = IV$. Grafikusan:

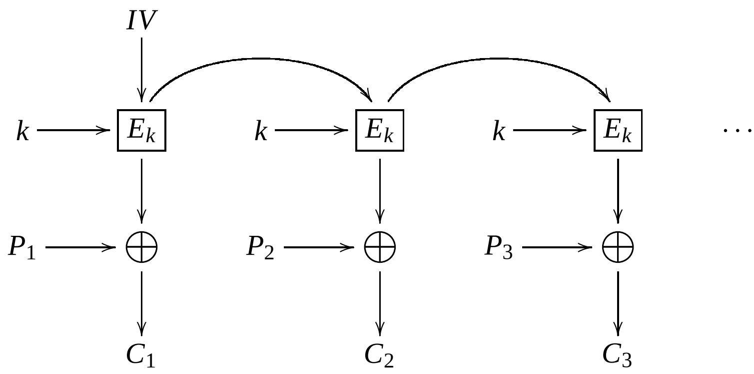


A CFB mód hasonlít a CBC módra abban, hogy kódolása nem, viszont dekódolása párhuzamossá tehető. Ugyanakkor, egy bit változtatása P_i-ben megváltoztat minden C_j-t, bármely $j \geq i$-re. Egy bit változtatása C_i-ben egy bitet módosít P_i-ben, és megváltoztatja az egész P_{i+1}-et.

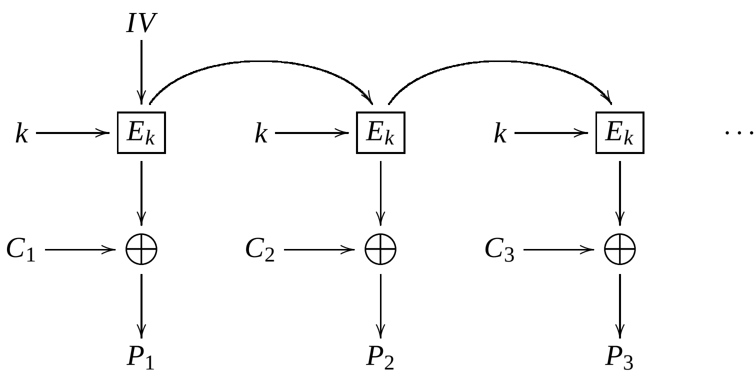
Az OFB mód (Output feedback mode)

Ez a mód szintén folyamtitkosítóvá alakítja a tömbtitkosítót.

Kódolás: $C_i = P_i \oplus O_i$, $O_i = E_k(O_{i-1})$, $O_0 = IV$. Grafikusan:



Dekódolás: $P_i = C_i \oplus O_i$, $O_i = E_k(O_{i-1})$, $O_0 = IV$. Grafikusan:



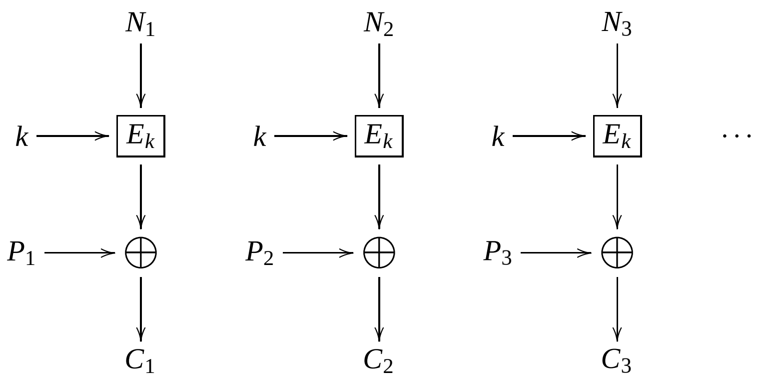
Vegyük észre, hogy mind a kódolás, mind a dekódolás párhuzamosítható, ha az O_i sorozatot előre legeneráljuk. Ez gyorsan elvégezhető, ha CBC módban egy csak nullákból álló szöveget kódolunk IV kezdeti tömbbel.

Az is észrevehető, hogy egy bit változtatása P_i-ben (vagy C_i-ben) pontosan egy bitet változtat C_i-ben (vagy P_i-ben). Az összes többi tömb változatlan marad. Az IV kezdeti tömb (initial vector) újrahasználása ugyanazzal a kulccsal az O_i sorozat (tehát a kulcs) újboli felhasználását jelentené egy folyamtitkosítóban, ami nem megengedhető.

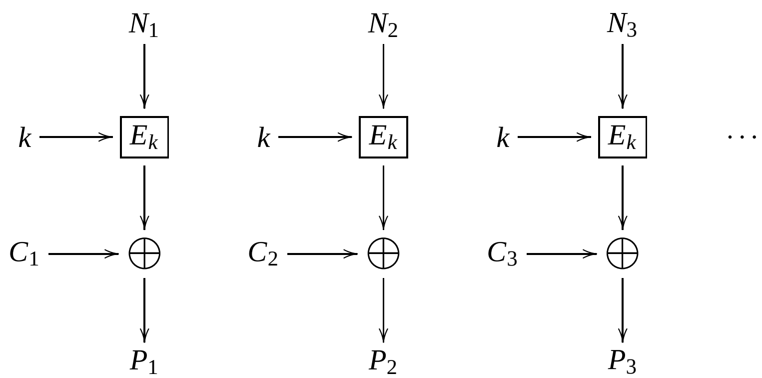
A CTR mód (Counter mode)

Ha $||$-sal jelöljük a bittömbök egymásután való illesztését (konkatenálását), akkor:

Kódolás: $C_i = P_i \oplus E_k(N_i)$, ahol $N_i = IV \parallel \underbrace{00\ldots0i}_{\text{számláló}}$. Grafikusan:



Dekódolás: $P_i = C_i \oplus E_k(N_i)$, ahol $N_i = IV \parallel \underbrace{00\ldots0i}_{\text{számláló}}$. Grafikusan:



A CTR mód nagy előnye, hogy alkalmas párhuzamos kódolásra és dekódolásra, tehát nagyon gyors. Azonban itt is vigyáznunk kell, hogy az *IV* kezdeti vektort ne használjuk újra ugyanazzal a kulccsal.

2.2.2.2. A lavina-effektus

A *lavina-effektus* tömbtitkosítóknál azt jelenti, hogy egy kis változtatás az eredeti üzenetben vagy kulcsban a kimenet drasztikus változását eredményezi. Általában szükséges, hogy a tömbtitkosítók már néhány menet után is megvalósítsák a lavina-effektust.

2.2.2.3. Általános támadások a tömbtitkosítók ellen

1. *A differenciál-kriptoanalízis* módszerét az 1980-as évek végén publikálta E. Biham és A. Shamir. Megállapították azt is, hogy a DES feltűnően ellenálló ezzel

a támadással szemben. Ez nem véletlen, hisz amint ezt 1994-ben D. Coppersmith az IBM DES-csapatának tagja kijelentette, 1974-ben már ismerték ezt a támadást, csak titokban tartották. A differenciál-kriptoanalízis egy választott nyílt szöveg ismeretére alapuló támadás, amely a például a DES-t 2^{47} választott nyílt szöveg segítségével tudja feltörni.
Tekintsünk (P_1, P_2) nyílt szövegpárokat úgy, hogy a $d_P = P_1 \oplus P_2$ úgynevezett különbség állandó legyen. Ezután az összes párra meghatározzuk a $d_C = E_k(P_1) \oplus E_k(P_2) = C_1 \oplus C_2$ lehetséges különbségeket és ezek előfordulási gyakoriságát. A (d_P, d_C) párt *differenciál*nak nevezzük. A támadás alapja az, hogy bizonyos (d_P, d_C) differenciállal különösen nagyszámú (P_1, P_2) pár rendelkezik. Így a tömbtitkosító kimenete már nem tekinthető véletlenszerűnek, és ez alapján bizonyos kulcsok kizárhatóak (a kulcstér szűkíthető). Részletekért lásd [7].

2. *Lineáris kriptoanalízis*. Ez egy nyílt szöveg ismeretén alapuló támadás, amelynek felfedezője M. Matsui (lásd még [7]). A módszert 1993-ban sikeresen alkalmazta a DES-re, 2^{43} nyílt szöveg felhasználásával.
A lineáris kriptoanalízis két fontos részből áll:

   - olyan lineáris összefüggések felírása az eredeti szöveg, a kódolt szöveg és a kulcs bitjei között, amelyek nagy (1-hez közeli) valószínűséggel igazak, vagy nagy valószínűséggel nem teljesülnek.
   - ezen lineáris összefüggések felhasználása arra, hogy nyílt- és a nekik megfelelő kódolt szövegek (szövegpárok) ismerete alapján meghatározzuk a kulcs bitjeit.

   (a) Lineáris összefüggések felírása. Jelölje $B[i]$ a B bittömb i-edik bitjét, és legyen $B[i_1, i_2, \dots i_a] = B[i_1] \oplus \cdots \oplus B[i_a]$, ahol $\oplus$-szal jelöltük az XOR bitműveletet. Akkor lineáris összefüggés alatt a következő egyenletet értjük:
$$P[i_1, i_2, \dots i_a] \oplus C[j_1, j_2, \dots j_b] = K[\ell_1, \ell_2 \dots, \ell_c],$$
ahol K a kulcs, P az eredeti szövegtömb, C pedig a kódolt szövegtömb.
Ideális titkosítási rendszer esetén $0,5$ a valószínűsége annak, hogy egy ilyen összefüggés fennálljon. A cél tehát olyan lineáris egyenleteknek a felírása, melyeknek valószínűsége lehetőleg nagyban eltér $0,5$-től (vagyis közel áll 0-hoz, vagy 1-hez). Ez a tömbtitkosító alapos elemzése alapján tehető meg.

   (b) A kulcs bitjeinek meghatározása. Tételezzük fel, hogy találunk egy $0,5$-től nagyban eltérő valószínűséggel fennálló lineáris egyenletet. Legyen ez
$$P[i_1, i_2, \dots i_a] \oplus C[j_1, j_2, \dots j_b] = K[\ell_1, \ell_2 \dots, \ell_c].$$

Rögzítünk a K kulcsra egy $T = \ell_1\ell_2\ldots\ell_c$ bitrendszert (nevezzük ezt részleges kulcsnak), majd az ismert eredeti–kódolt szövegpárokra és erre a rögzített T részleges kulcsra ellenőrizzük az előbbi egyenletet. Tegyük fel továbbá, hogy az esetek p_T százalékában igaz az egyenlet. Ekkor, ha $\left|\frac{p_T}{100} - \frac{1}{2}\right|$ nagy (vagyis p_T közel áll 0-hoz vagy 1-hez), akkor valószínű, hogy T a valódi kulcs bitjeiből áll. Minél több egyenletre alkalmazzuk ezt a módszert, a kulcs annál több bitjét tudjuk meghatározni.

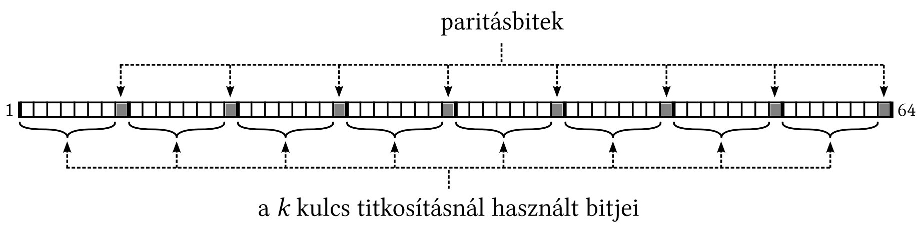
3. *XSL támadás*. 2002-ben publikálta N. Courtois és J. Pieprzyk; aránylag, kevés nyílt szöveg ismeretét igénylő támadás. Lényege, hogy a tömbtitkosító alapján fel lehet írni egy többváltozós kvadratikus egyenletekből álló rendszert, melynek megoldása megadja a kulcsot. A probléma azonban az, hogy ezek az egyenletrendszerek általában hatalmasak (például az AES-128 esetében 8000 egyenletből állnak 1600 ismeretlennel) és általános megoldásuk NP-teljes. Ha azonban sokkal több egyenlet van, mint ismeretlen, akkor linearizálni lehet a rendszert, esélyt teremtve a megoldhatóságra. Courtois és Pieprzyk egy speciális linearizáló megoldási algoritmust fejlesztett (neve XSL = e**x**tended **s**parse **l**inearization), amely állításuk szerint 2^{100} műveletet igényel az AES-128 esetében. Később azonban mások cáfolták ezt, állítva, hogy nincs elég független lineáris egyenlet, amely a megoldáshoz vezetne.

2.2.2.4. A DES

A DES az angol Data Encryption Standard[4] kezdőbetűiből tevődik össze. Az IBM egy kutatócsoportja fejlesztette ki az 1970-es évek elején. Az egyesült államokbeli szabványügyi hivatal először 1976-ban hagyta jóvá, mint az elektronikus kommunikációban az adatok titkosítására használható módszert. A szabványt újravizsgálták és ismételten jóváhagyták 1983-ban, 1988-ban és 1999-ben, míg végül 2002-ben bevezették az új szabványt, az AES-t.

A DES 64 bites adattömböket titkosít.

Kulcs: A k kulcs is egy 64 bites bináris adattömb. Fontos megjegyezni, hogy a lehetséges 2^{64} méretű kulcstér helyett a DES kulcstere csak 2^{56} nagyságú, mivel a kulcs minden egyes bájtjának utolsó bitje hibaellenőrzésre van fenntartva (paritásbit). A DES-kulcs struktúráját láthatjuk az alábbi ábrán:



[4] adattitkosítási szabvány

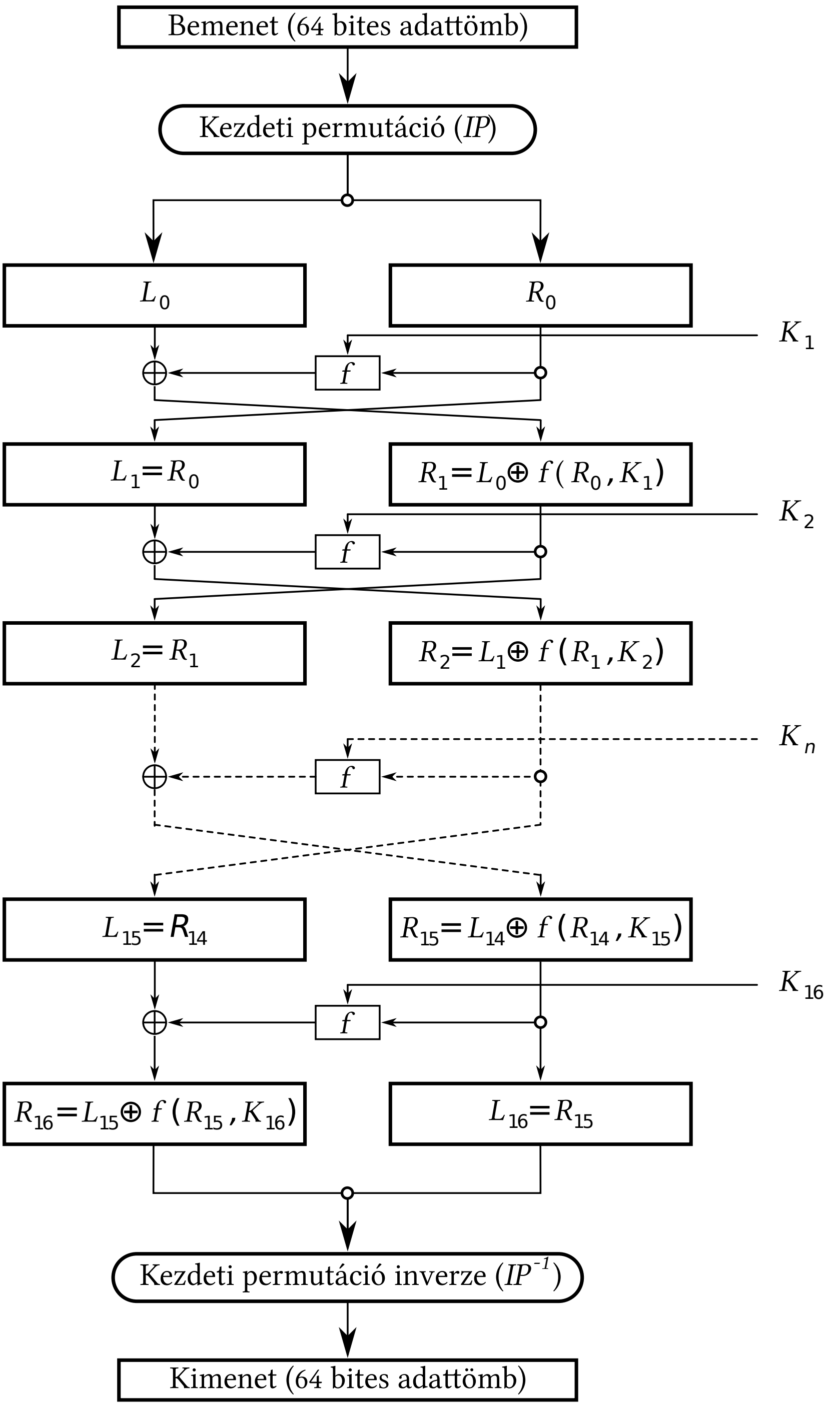
A kulcs bitjeit (és az összes többi adattömb bitjeit, amelyek a DES leírásában szerepelnek) balról jobbra számláljuk, egyestől kezdve. Ennek megfelelően a k kulcs 8., 16., 24., 32., 40., 48., 56. és 64. bitje paritásbit, az összes többi bit a titkosítás paramétere. Egy paritásbit értéke 1, ha az előtte levő hét bit páros számú egyest tartalmaz, és 0 ellenkező esetben.

Kódolás: A kódolás folyamatának vázlata látható a 2.10 ábrán. A titkosító algoritmus bemenete, vagyis a nyílt szöveg, egy 64 bites adattömb, jelöljük ezt B-vel. Első lépésként egy kezdeti permutáció megkeveri a nyílt szöveg bitjeit. A kezdeti permutációt IP-vel jelöljük és táblázatban (illetve grafikusan ábrázolva) adjuk meg (lásd a 2.11 ábrát). A 2.11 ábrának megfelelően a permutált bemenet – $IP(B)$ – első bitje az eredeti szöveg 58-adik bitje lesz, a második bitje az eredeti szöveg 50-edik bitje lesz stb. Ezután a permutált bemenetet egy k kulcstól függő eljárás tovább alakítja. E folyamat eredményét (ami szintén egy 64 bites tömb) az algoritmus – utolsó lépésként – IP^{-1} végső permutációval permutálja. Ez lesz a C kódolt szöveg, vagyis az algoritmus kimenete. A végső permutáció a kezdeti permutációnak az inverze és a 2.12 ábrán láthatjuk (táblázatos és grafikus formában). A kezdeti és végső permutációk rögzítettek, hatásuk nem függ a k kulcstól. A DES titkosítási eljárásának pszeudokódja a következő:

Algorithm DES (E_k):
Input: B, k
Output: C
1: $L_0 \parallel R_0 := IP(B)$ {kulcstól független}
2: **for** $n := 1, \ldots, 16$ **do**
3: $\quad K_n := KS(n, k)$
4: $\quad L_n := R_{n-1}$
5: $\quad R_n := L_{n-1} \oplus f(R_{n-1}, K_n)$
6: **end for**
7: $C' := R_{16} \parallel L_{16}$
8: $C := IP^{-1}(C')$ {kulcstól független}
9: return C
end Algorithm

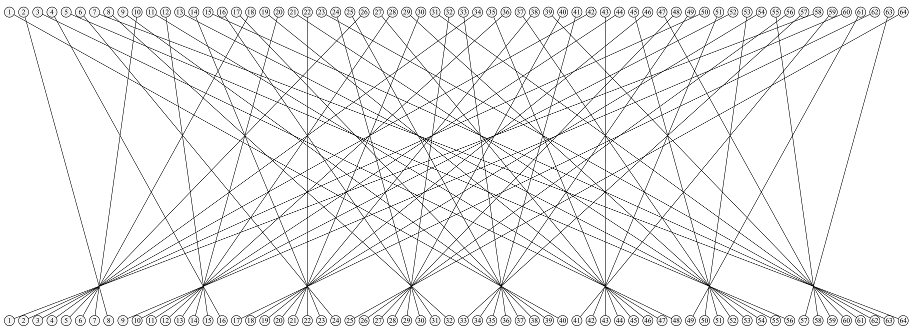
Az algoritmusban második és hatodik sor között található a titkosítási eljárásnak a kulcstól függő része. Az első sor szerint $L_0 \parallel R_0 := IP(B)$, ami azt jelenti, hogy a 64 bites permutált bemenetet két 32 bites tömbre választjuk szét, ezek L_0 és R_0. A $\parallel$ jel bitsorozatok összeillesztését (konkatenálását) jelöli az adott sorrendben, L_n és R_n, $n = \overline{0, 16}$, egyenként 32 bites tömböket jelölnek, a $\oplus$ XOR műveletet jelent. Az algoritmus leírásában szerepel még a 16-szor végrehajtott KS és f függvény. A továbbiakban ezek leírását adjuk meg.

2.10. ábra. A DES kódoló ejárásának vázlata



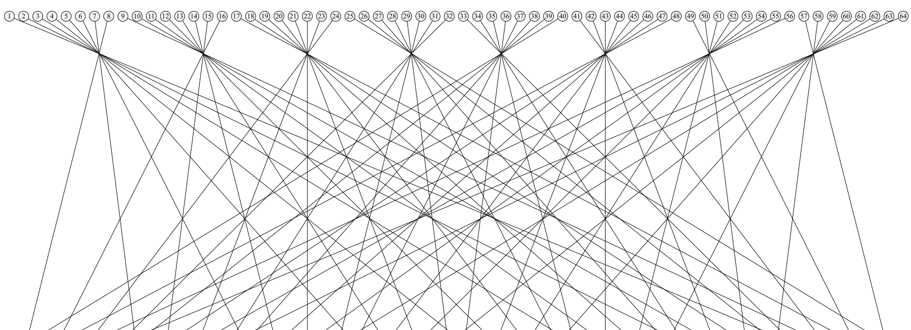
2.11. ábra. A DES kezdeti permutációja (IP) táblázatban és grafikusan

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7



2.12. ábra. A DES végső permutációja (IP^{-1}) táblázatban és grafikusan

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25



KS az úgynevezett *kulcsütemező függvény*[5]. A kulcsütemező függvény szerepe az, hogy az iteráció számától függően kiválasszon 48 bitet a k kulcs titkosításra használt 56 bitje közül (úgy is mondhatjuk, hogy a KS segédkulcsokat generál). Ennek megfelelően, $K_n = KS(n, k)$, vagyis KS egyik bemenete az iteráció száma ($n \in \{1, \ldots, 16\}$), a másik bemenete pedig a kulcs. A KS kimenete egy 48 bites tömb, a K_n. A kulcsütemező működését szemlélteti a 2.13 ábra, leírását pszeudokódban is megadjuk:

Algorithm KS:
Input: n, k
Output: K_n

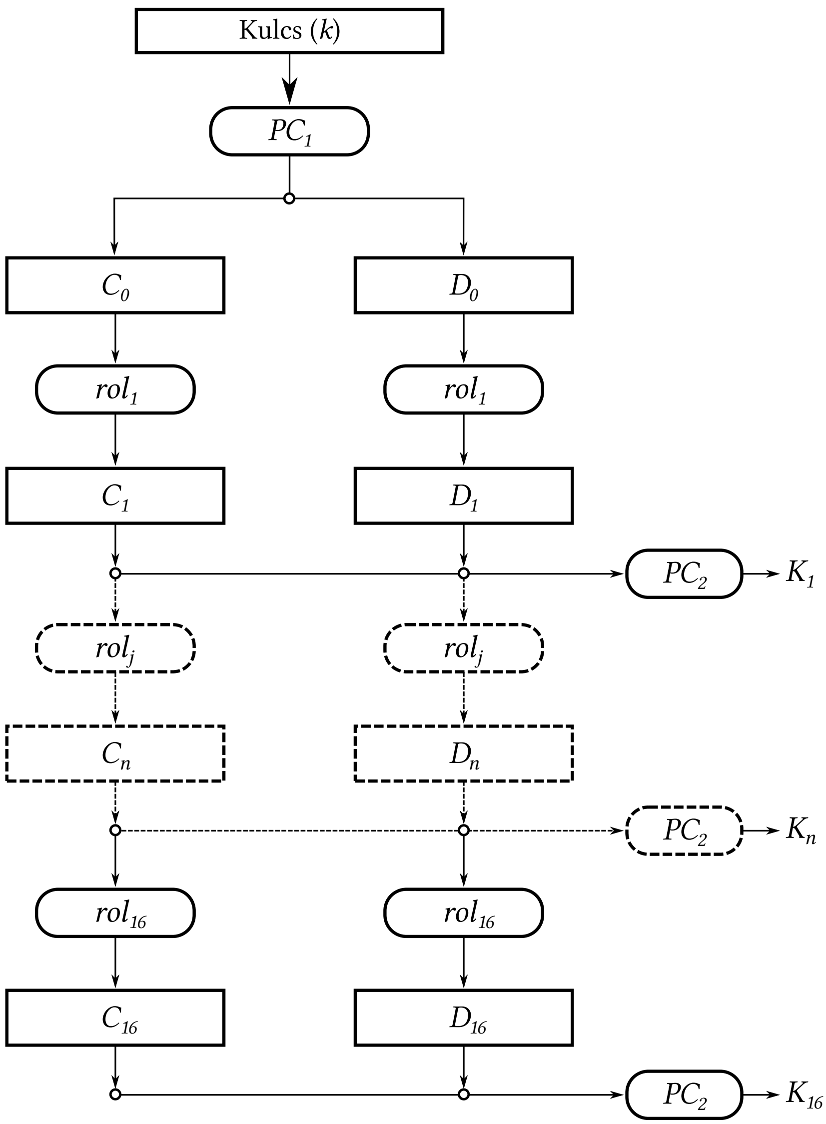
$C_0 \parallel D_0 := PC_1(k)$
for $i := 1, \ldots, n$ **do**
  if $i \in \{1, 2, 9, 16\}$ **then**
    $j := 1$
  else
    $j := 2$
  end if
  $C_i := rol(C_{i-1}, j)$
  $D_i := rol(D_{i-1}, j)$
end for
$K_n := PC_2(C_n \parallel D_n)$
return K_n

end Algorithm

Az algoritmusban megjelenő PC_1 (permuted choice, azaz permultáló-kiválasztó) függvény bemenete a k kulcs, kimenete pedig egy 56 bites tömb (kimaradnak a paritásbitek). A PC_1 pontos meghatározása látható a 2.14 ábrán. A PC_1 56 bites kimenetét két 28 bites tömbre osztjuk, ezek C_0 és D_0. C_m és D_m ($m = \overline{0, 16}$), 28 bites tömböket jelölnek. A 2.14 ábrán látható táblázat két részre van osztva, a felső rész C_0 bitjeinek a meghatározása, a alsó pedig a D_0 bitjeié. Ennek megfelelően a C_0 első bitje a kulcs 57-edik bitje, a második bitje a kulcs 49-edik bitje stb. A D_0 első bitje a kulcs 63-adik bitje, a második bitje a kulcs 55-ödik bitje stb. Az iteráció $n \in \{1, \ldots, 16\}$ számától függően a C_n és D_n tömböket C_{n-1}-ből illetve D_{n-1}-ből kapjuk úgy, hogy az előbbiek bitjeit ciklikusan balra toljuk egyszer vagy kétszer (ezt hajtja végre a $rol(X, s)$ függvény: az X tömb bitjeit ciklikusan balra tolja s pozícióval). A 2.4 táblázatban tüntettük fel azt, hogy az iteráció számának megfelelően hány ciklikus balra tolás történik. Az n-edik iteráció végén az összeillesztett (konkatenált) C_n és D_n tömbök bitjei közül (összesen 56 bit) a PC_2 permutáló-kiválasztó függvény

[5] angolul: *key schedule*

2.13. ábra. A kulcsütemező (KS) vázlata

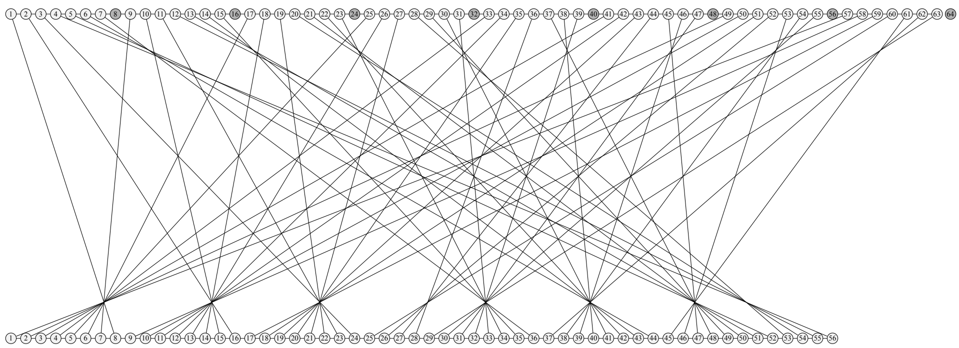


2.4. táblázat. A kulcs bitjeinek ciklikus balra tolása az iteráció számának függvényében

Iteráció száma	Ciklikus balra tolások száma
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

2.14. ábra. A PC_1 permutáló-kiválasztó függvény táblázatban és grafikusan

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4



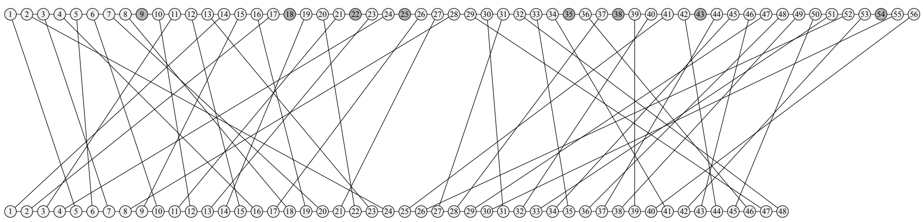
kiválaszt 48 bitet. A PC_2 függvény bemenete tehát egy 56 bites, kimenete pedig egy 48 bites tömb. A PC_2 definíciója a 2.15 ábrán látható.

A DES-t leíró pszeudokódban és a 2.10 ábrán megjelenik még az $f(R,K)$ függvény. Ennek két bemenete van, egy 32 bites tömb (R) és egy 48 bites tömb (K). Az f működési vázlatát a 2.16 ábrán láthatjuk. Legyen E egy olyan függvény, amely a 32 bites bemeneti tömb bitjei közül válogatva épít fel egy 48 bites kimenetet, a 2.17 ábrán látható táblázatnak és grafikus ábrázolásnak megfelelően. Mivel a kimenet több bitből áll, mint a bemenet, E-t még duzzasztó függvénynek[6] is nevezzük. Az E függvény kimenetét összeadjuk a kulcsütemező által szolgáltatott tömbbel (XOR műveletet használva), majd az eredményt ($48 = 8 \times 6$ bites tömb) nyolc különböző függvény, úgynevezett *S-doboz* alakítja tovább. Az $S_1, S_2, \ldots, S_8$ S-dobozok mindegyike 6 bites bemenetből 4 bites kimenetet gyárt. Ahogy az a 2.16 ábrán is látható, a 48 bites tömb első 6 bitje lesz az S_1 bemenete, a második 6 bitje lesz az S_2 bemenete stb. Az ábrák utáni táblázatokban megadjuk a nyolc S-doboz leírását.

[6]Angolul: *expansion function*

2.15. ábra. A PC_2 permutáló-kiválasztó függvény táblázatban és grafikusan

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32



S_1 :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

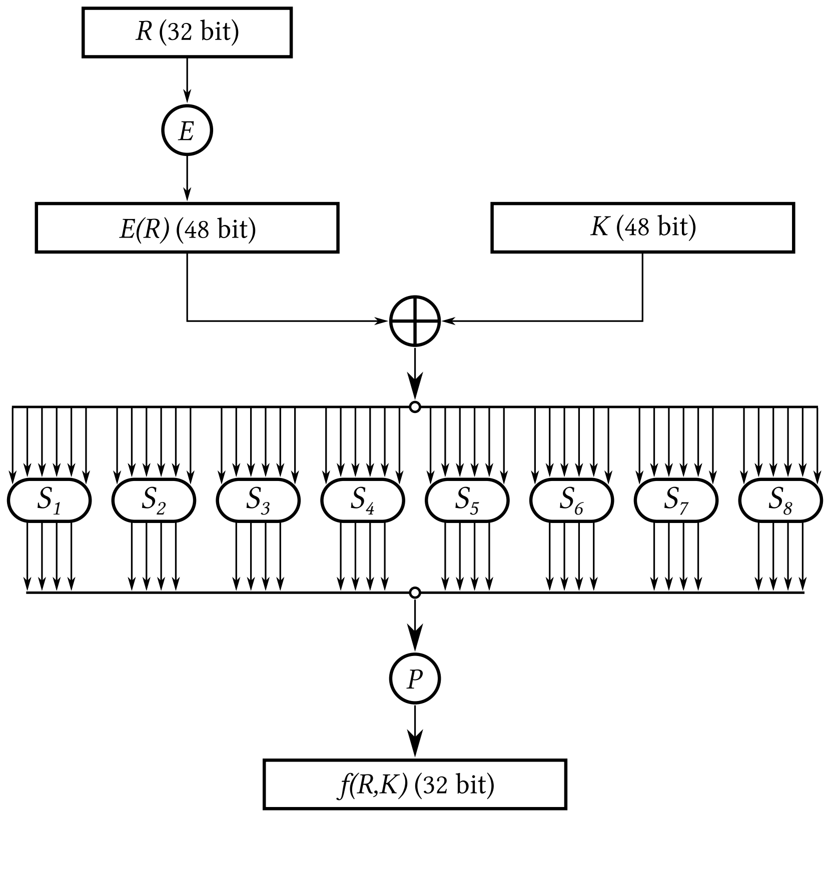
S_2 :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3 :

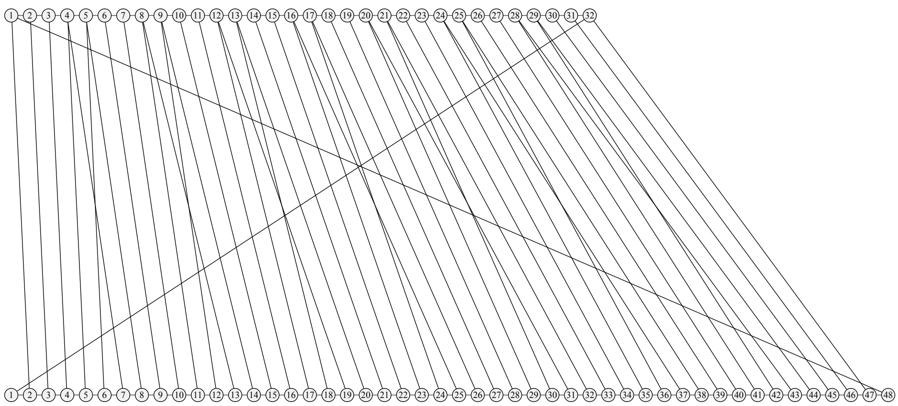
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

2.16. ábra. Az $f(R, K)$ függvény vázlata



2.17. ábra. Az *E* duzzasztó függvény táblázatos és grafikus formában

32	1	2	3	4	5
4	5	6	7	8	9
8	9	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	30	31	32	1



$\mathbf{S_4}$:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

$\mathbf{S_5}$:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

$\mathbf{S_6}$:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

$\mathbf{S_7}$:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

$\mathbf{S_8}$:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Lássuk, hogyan lesz egy S_ℓ, $\ell \in \{1, \ldots, 8\}$ S-doboz 6 bites T bemenetéből 4 bites kimenet. T első és utolsó bitje kettes számrendszerben meghatároz egy $i \in \{0, 1, 2, 3\}$ számot. T középső négy bitje ugyanígy egy $j \in \{0, 1, \ldots, 15\}$ számot határoz meg. Megkeressük az S_ℓ táblázatában az i-edik sorban és j-edik oszlopban levő számot – legyen ez $S_\ell[i, j]$. Az S-dobozok úgy vannak meghatározva, hogy $S_\ell[i, j] \in \{0, 1, \ldots, 15\}$, $\forall \ell \in \{1, \ldots, 8\}$, $\forall i \in \{0, 1, 2, 3\}$, $\forall j \in \{0, 1, \ldots, 15\}$. Így az S_ℓ 4 bites kimenete $(S_\ell[i, j])_2$ lesz, vagyis az $S_\ell[i, j]$ szám kettes számrendszerbeni felírásának számjegyei (ha szükséges, akkor ezt nullákkal egészítjük ki balról, hogy az eredmény négy számjegyű, vagyis négy bit hosszú legyen). Például, az első S-doboz esetén, ha $T = 011011$, akkor $i = (01)_2 = 1$, $j = (1101)_2 = 13$, tehát $S_1(T) = (S_1[1, 13])_2 = 5 = (101)_2 = 0101$. A nyolc S-doboz összesen $8 \times 4 = 32$ bites kimenetét egy P permutáció tovább alakítja, ez lesz az f függvény kimenete. Az f eljárást pszeudokódban is megadjuk:

Algorithm f:
Input: R, K
Output: T
  $T_1 \| T_2 \| T_3 \| T_4 \| T_5 \| T_6 \| T_7 \| T_8 := K \oplus E(R)$
  $T := P(S_1(T_1) \| S_2(T_2) \| \cdots \| S_8(T_8))$
  return T
end Algorithm

A P permutációt a 2.18 ábrán látható módon határozták meg.

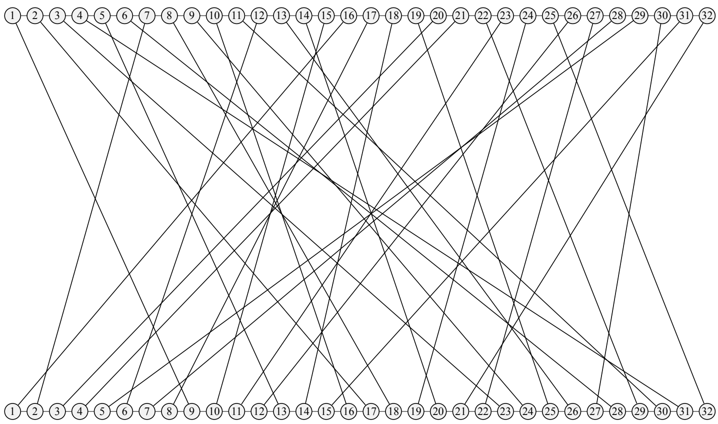
Dekódolás: Mivel a kezdeti IP permutáció a végső permutáció inverze és a titkosítási eljárás kulcstól függő részében megjelenő R és L tömbökre igaz, hogy

$$R_{n-1} = L_n$$
$$L_{n-1} = R_n \oplus f(L_n, K_n)$$

a dekódoló algoritmus ugyanazon lépéseket fogja tartalmazni, mint a titkosító eljárás, melytől annyiban különbözik, hogy most a tömböket fordított sorrendben használjuk: $R_{16} \| L_{16}$ lesz a permutált bemenet, a K_{16} segédkulcsot használjuk az első iteráció alkalmával, A K_{15}-öt a második iteráció alkalmával stb. A dekódoló eljárás pszeudokódja:

2.18. ábra. A *P* permutáció táblázatban és grafikusan

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25



Algorithm DES (D_k):
Input: C, k
Output: B

$R_{16} \parallel L_{16} := IP(C)$ {kulcstól független}
for $n := 16, \ldots, 1$ **do**
  $K_n := KS(n, k)$
  $R_{n-1} := L_n$
  $L_{n-1} := R_n \oplus f(L_n, K_n)$
end for
$B' := L_0 \parallel R_0$
$B := IP^{-1}(B')$ {kulcstól független}
return B

end Algorithm

Kriptoanalízis: A DES az 1990-es évek végétől már nem tekinthető biztonságosnak, az alábbi támadások miatt:

1. Kimerítő kulcskeresés. 1998-ban az EFF (Electronic Frontier Foundation) egy speciálisan a DES feltörésére tervezett 250 ezer dolláros számítógéppel kimerítő kulcskeresést használva 56 óra alatt találta meg a kulcsot. Hat hónappal később ehhez a géphez internet segítségével még 100 ezer személyi számítógépet csatlakoztattak, így 22 óra alatt sikerült a DES feltörése.
2. Differenciál-kriptoanalízis. A DES úgy volt tervezve, hogy ellenálljon a differenciál-kriptoanalízisnek. Legkevesebb 2^{47} választott üzenet szükséges ahhoz, hogy ez a támadástípus sikeres legyen.
3. Lineáris kriptoanalízis. Legkevesebb 2^{43} ismert üzenet szükséges ahhoz, hogy a támadás sikeres legyen.

A biztonság növelése érdekében 1999 októberében az amerikai szabványügyi hivatal szabványosítja az úgynevezett tripla-DESt (TDEA[7]). Ha E_k a DES titkosító eljárása, D_k a dekódoló eljárás, k pedig a kulcs, akkor a tripla-DES a

$$C = E_{k_3}(D_{k_2}(E_{k_1}(P)))$$

módon kódol, és

$$P = D_{k_1}(E_{k_2}(D_{k_3}(C)))$$

módon dekódol, ahol (k_1, k_2, k_3) három DES-kulcs.

[7] TDEA = Triple Data Encryption Algorithm

2.5. táblázat. Bájtok és bitek indexei az AES leírásában

Bináris tömb (bit-index)	0	1	2	3	4	4	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
Bájt-index	0								1								2								...
Bájton belüli bit-index	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	...

2.6. táblázat. Az AES lehetséges kulcshossz–bemenet-hossz–menetszám-kombinációi

	Kulcshossz (Nk szó)	Bemenet hossza (Nb szó)	Menetek száma (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

2.2.2.5. Az AES

1997-ben az amerikai szabványügyi hivaltal (NIST – National Institute of Standards and Technology) pályázatot írt ki a DES-t felváltó, következő titkosítási szabványra. A beérkezett 15 pályázatból 5 jutott a második fordulóba (MARS, RC6, Rijndael, Serpent és Twofish). 2000 októberében bejelentették, hogy az AES szabvány (Andvanced Encryption Standard) a 128, a 192 és a 256 bites kulcsú Rijndael algoritmus lesz. Az algoritmus szerzői Joan Daemen és Vincent Rijmen belga kriptográfusok.

Kulcs: Az AES k kulcsa egy 128, 192 vagy 256 bites bináris adattömb. A kulcs bitjeit balról számláljuk 0-val kezdve, tehát a bitek i indexei $0 \leq i < 128$, $0 \leq i < 256$ illetve $0 \leq i < 256$ határok között lehetnek. Jelöljük Nk-val a kulcs szavakban mért hosszát. Az AES esetében a (számítástechnikai értelemben vett) szóra igaz, hogy: 1 szó = 4 bájt = 32 bit. Az Nk lehetséges értékei tehát: 4, 6 és 8.

Az AES leírásában többször fogunk hivatkozni bizonyos bináris tömbök (bemenet, kulcs stb.) egyes bitjeire vagy bájtjaira. A 2.5 táblázatban tüntettük fel a bitek és bájtok sorrendjét.

Kódolás: Az AES 128 bites tömböket titkosít, jelöljük a 128 bites nyílt szövegblokkot P-vel, szavakban mért hosszát pedig Nb-vel. Nyilván $Nb = 4$. A titkosító algoritmus több menetet (iterációt) hajt végre, a menetek száma a kulcs hosszától függ. Jelöljük Nr-rel az iterációk számát. A 2.6 táblázatban megadjuk a szabványban rögzített, lehetséges kulcshossz–bemenet-hossz–menetszám-kombinációkat.

Először megadjuk az AES titkosító eljárását pszeudokódban:

Algorithm AES (E_k):
Input: P, Nk, k
Output: C

1: $S := P$
2: $K := KeyExpansion(Nk, k)$
3: $S := AddRoundKey(S, K[0 \dots Nb - 1])$
4: **for** $n := 1, \dots, Nr - 1$ **do**
5: $\quad S := SubBytes(S)$
6: $\quad S := ShiftRows(S)$
7: $\quad S := MixColumns(S)$
8: $\quad S := AddRoundKey(S, K[n \cdot Nb \dots (n + 1) \cdot Nb - 1])$
9: **end for**
10: $S := SubBytes(S)$
11: $S := ShiftRows(S)$
12: $S := AddRoundKey(S, K[Nr \cdot Nb \dots (Nr + 1) \cdot Nb - 1])$
13: $C := S$
14: return C

end Algorithm

A következőkben részletesen bemutatjuk az algoritmusban szereplő jelöléseket, változókat és részeljárásokat.

Az AES-titkosítás köztes eredményei az úgynevezett *állapot*ok (state). Az algoritmus pszeudokódjában az állapotnak az S változó felel meg. Az állapotot egy 4 sorból és Nb oszlopból álló táblázatként ábrázolhatjuk, ahol a táblázat minden eleme egy bájt. Az S *állapottáblá*ban minden egyes s bájtnak két index felel meg: az i sorindex és a j oszlopindex ($0 \leq i < 4$ és $0 \leq j < Nb$). Az algoritmus első lépéseként (1. sor) az állapottáblát feltöltjük a bemenet, vagyis a nyílt szöveg bájtjaival, az eljárás végén a kódolt szöveg szintén az állapottáblából kerül ki (13. sor):

a nyílt szöveg (P) bájtjai

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

$\rightarrow$

állapottábla (S)

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$\rightarrow$

a kódolt szöveg (C) bájtai

c_0	c_4	c_8	c_{12}
c_1	c_5	c_9	c_{13}
c_2	c_6	c_{10}	c_{14}
c_3	c_7	c_{11}	c_{15}

A P nyílt szöveg bájtjai $p_0, p_1, \dots, p_{15}$ (tehát $P = p_0 \parallel p_1 \parallel \dots \parallel p_{15}$), az S állapot bájtai $s_{i,j}$, a kódolt szöveg pedig $C = c_0 \parallel c_1 \parallel \dots \parallel c_{15}$. A nyílt szöveg bájtjait az

$$s_{i,j} := p_{i+4j} \qquad 0 \leq i < 4,\ 0 \leq j < Nb$$

módon töltjük be az S állapottáblába, és a kódolt szöveget a

$$c_{i+4j} := s_{i,j} \qquad 0 \le i < 4,\ 0 \le j < Nb$$

módon olvassuk ki ugyanonnan.

A kódoló eljárásban az első $Nr-1$ menet ugyanaz, az utolsó abban különbözik, hogy nem tartalmazza a $MixColumns()$ eljárást. Vegyük sorra a menetekben szereplő eljárásokat.

- A $SubBytes()$[8] egy nemlineáris függvény, amely az állapottábla bájtjait egyenként (egymástól függetlenül) helyettesíti új értékekkel. Legyen $s_{i,j} = (\beta_7\beta_6\beta_5\beta_4\beta_3\beta_2\beta_1\beta_0)_2$ az S állapottábla egy bájtja, $s'_{i,j} = (b'_7b'_6b'_5b'_4b'_3b'_2b'_1b'_0)_2$ pedig az $s_{i,j}$-t felcserélő új érték (egy bájton belül a biteket jobbról balra számláljuk, lásd a 2.5 táblázatot). Az invertálható $SubBytes()$ a következő két transzformáció (T_1 és T_2) összetevéséből áll:

  1. Tekintsük az $s_{i,j}$ bájtot $\mathbb{F}_{2^8} \cong \mathbb{Z}_2[X]/(f)$ véges testbeli elemként (lásd a 3. függeléket). A megfeleltetés a következő módon történik: legyenek az $s_{i,j}$ bitjei egy $g \in \mathbb{Z}_2[X]$, legtöbb hetedfokú polinom együtthatói:

     $$g = \beta_7X^7 + \beta_6X^6 + \beta_5X^5 + \beta_4X^4 + \beta_3X^3 + \beta_2X^2 + \beta_1X + \beta_0 = \sum_{i=0}^{7}\beta_iX^i.$$

     Az AES esetében az irreducíbilis főpolinom $\mathbb{Z}_2[X]$-ben $f = X^8 + X^4 + X^3 + X + 1$. Invertáljuk a nemnulla g polinomot modulo f az $\mathbb{F}_{2^8} \cong \mathbb{Z}_2[X]/(f)$ testben, az inverze legyen $g^{-1} = \sum_{i=0}^{7} b_iX^i$ modulo f. Ha ezt az első transzformációt T_1-gyel jelöljük, akkor

     $$T_1(s_{i,j}) = \begin{cases} (b_7b_6b_5b_4b_3b_2b_1b_0)_2 & \text{ha } s_{i,j} \ne (00000000)_2 \\ (00000000)_2 & \text{ha } s_{i,j} = (00000000)_2 \end{cases}$$

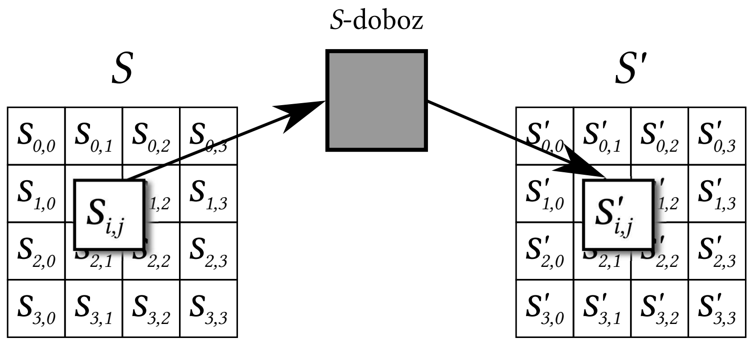
  2. A T_2 a következő affin transzformáció: $T_2((b_7b_6b_5b_4b_3b_2b_1b_0)_2) = (b'_7b'_6b'_5b'_4b'_3b'_2b'_1b'_0)_2$, ahol

     $$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

     bármely $0 \le i < 8$-ra és $c = (c_7c_6c_5c_4c_3c_2c_1c_0)_2 = (01100011)_2$.

[8] Az angol „substitute bytes”, vagyis bájt-helyettesítő rövidítése.

2.19. ábra. A $SubBytes()$ transzformáció



Mátrixos alakban is ki lehet fejezni az előbbi affin transzformációt:

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Az előbbi jelöléseket hasznlálva tehát a $SubBytes()$ az S állapottábla minden egyes bitjére a $T_2 \circ T_1$ transzformációt hajtja végre (lásd a 2.19 ábrát). Az algoritmus hatékonyságát növelni lehet azáltal, hogy előre kiszámítjuk minden egyes bájt helyettesítési értékét. Ezeket az értékeket egy táblázatban (S-dobozban) foglaltuk össze (2.7 táblázat). Ha $s_{i,j} = (h_1h_2)_{16}$ ($s_{i,j}$ felírása a tizenhatos számrendszerben), akkor a helyettesítési értékét a 2.7 táblázat h_1-edik sorában és h_2-edik oszlopában találjuk . Például legyen $s_{1,1} = (5B)_{16}$. Ekkor az $s'_{1,1} = (39)_{16} = (00111001)_2 = 57$ lesz, mert a $(39)_{16}$ hexadecimális érték van a táblázat 5. sorában és $(0B)_{16}$. (vagyis 11.) oszlopában.

- A $ShiftRows()$ eljárás az állapottábla sorait ciklikusan balra tolja. Az első sor változatlan marad, a második sor egy, a harmadik sor kettő a negyedik sor pedig három pozícióval tolódik ciklikusan balra:

$$s'_{i,j} = s_{i,(j+shift(i,Nb)) \bmod Nb} \quad 0 < i < 4,\ 0 \le j < Nb,$$

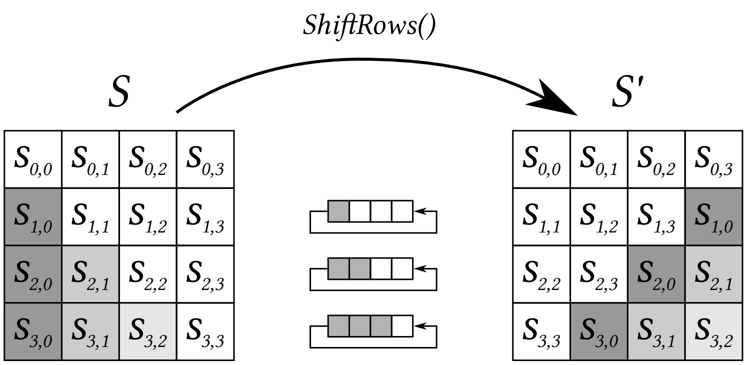
ahol $shift(i, Nb)$ függ az i sorindextől (és $Nb = 4$). A $shift()$ függvény értékei:

$$shift(1,4) = 1; \quad shift(2,4) = 2; \quad shift(3,4) = 3.$$

2.7. táblázat. A $SubBytes()$ eljárás S-doboza

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

2.20. ábra. A $ShiftRows()$ transzformáció



A 2.20 ábra a $ShiftRows()$ transzformációt szemlélteti.

- A $MixColumns()$ eljárás az S állapottábla oszlopaival dolgozik. A tábla minden egyes oszlopa egy (négy bájtból álló) szó. Tekintsük az állapottábla minden oszlopát a $(\mathbb{F}_{2^8}[X]/(f'), +, \otimes)$ véges gyűrű egy-egy elemének. A megfeleltetés – a $SubBytes()$ eljáráshoz hasonlóan – a következő módon történik. Legyen $\begin{matrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{matrix}$ az állapottábla valamelyik oszlopa ($j \in \{0, \ldots 3\}$), ekkor a neki megfelelő polinom az $s_{3,j}X^3 + s_{2,j}X^2 + s_{1,j}X + s_{0,j} \in \mathbb{F}_{2^8}[X]$ lesz. Legyen most $a, b \in (\mathbb{F}_{2^8}[X]/(f'), +, \otimes)$,

$$a = a_3X^3 + a_2X^2 + a_1X + a_0$$

és

$$b = b_3X^3 + b_2X^2 + b_1X + b_0.$$

Ekkor az összegük

$$a + b = (a_3 \oplus b_3)X^3 + (a_2 \oplus b_2)X^2 + (a_1 \oplus b_1)X + (a_0 \oplus b_0)$$

lesz, ahol $\oplus$ az XOR műveletet jelöli. Szorzatukat a következő módon végezzük el. Legyen $a \cdot b = c = \sum_{i=0}^{6} c_iX^i$, ahol

$$\begin{aligned}
c_0 &= a_0 \bullet b_0 \\
c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \\
c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \\
c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \\
c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
c_6 &= a_3 \bullet b_3.
\end{aligned}$$

A $\bullet$ művelet itt az $\mathbb{F}_{2^8} \cong \mathbb{Z}_2[X]/(f)$ testbeli szorzás (vagyis polinomok szorzása modulo $f = X^8 + X^4 + X^3 + X + 1$), az a_i és b_j együtthatókat (bájtokat) a $SubBytes()$ eljárásnál ismertetett módon feleltetjük meg polinomoknak). A c hatodfokú polinom és nem feleltethető meg közvetlenül egy négy bájtos szónak (vagyis az állapottábla egy oszlopának), ezért redukálni kell modulo egy negyedfokú polinommal. Ez a negyedfokú polinom az $f' = X^4 + 1$ (amire igaz, hogy $X^i \bmod (X^4 + 1) = X^{i \bmod 4}$). Legyen tehát

$$a \otimes b = (a \cdot b) \bmod f' = c \bmod f' = d = \sum_{i=0}^{3} d_iX^i,$$

ahol

$$\begin{aligned} d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\ d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\ d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\ d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3) \end{aligned}$$

Ha a egy rögzített polinom, akkor a szorzást mátrixos alakban is meg lehet adni:

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

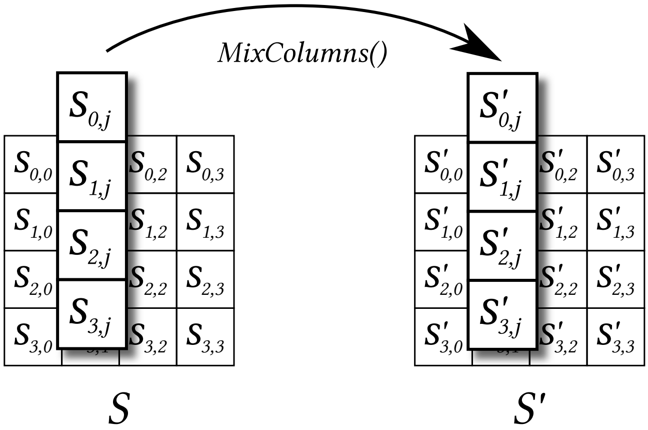
Az f' polinom nem irreducíbilis $\mathbb{F}_{2^8}[X]$-ben, ezért a szorzás általában nem invertálható (és ezért nem test $\mathbb{F}_{2^8}[X]/(f')$, csak gyűrű). Térjünk most vissza a *MixColumns*() eljáráshoz. Ha $s_j = s_{3,j}X^3 + s_{2,j}X^2 + s_{1,j}X + s_{0,j}$ a tábla $\begin{matrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{matrix}$ oszlopának megfelelő polinom a transzformáció végrehajtása előtt, amit a transzformáció az $\begin{matrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{matrix}$ oszloppal helyettesít (az ennek megfelelő polinom $s'_j = s'_{3,j}X^3 + s'_{2,j}X^2 + s'_{1,j}X + s'_{0,j}$), akkor a transzformáció $s' = a \otimes s$, ahol $a = 3X^3 + X^2 + X + 2$. Az a-val való szorzás invertálható, mert az a polinomnak van inverze: $a^{-1} = 11X^3 + 13X^2 + 9X + 14 = (0B)_{16}X^3 + (0D)_{16}X^2 + (09)_{16}X + (0E)_{16}$. A *MixColumns*() transzformáció egy oszlopra tehát:

$$\begin{pmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix},$$

vagyis

$$\begin{aligned} s'_{0,j} &= ((02)_{16} \bullet s_{0,j}) \oplus ((03)_{16} \bullet s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus ((02)_{16} \bullet s_{1,j}) \oplus ((03)_{16} \bullet s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus ((02)_{16} \bullet s_{2,j}) \oplus ((03)_{16} \bullet s_{3,j}) \\ s'_{3,j} &= ((03)_{16} \bullet s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus ((02)_{16} \bullet s_{3,j}) \end{aligned}$$

2.21. ábra. A $MixColumns()$ transzformáció



A 2.21 ábra szemlélteti a $MixColumns()$ transzformációt.

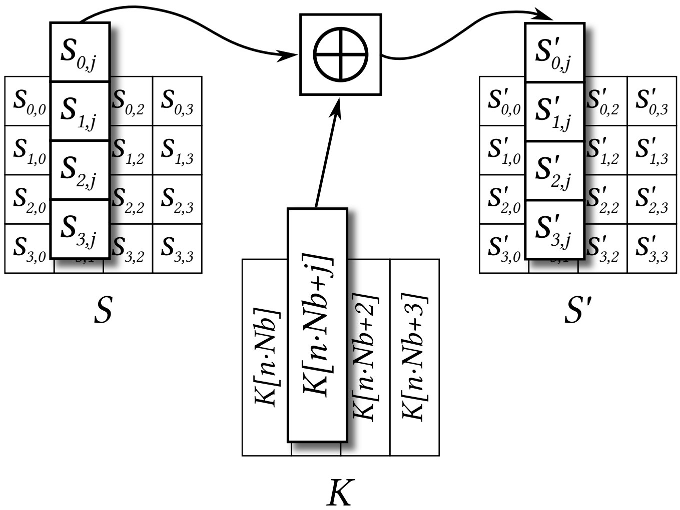
- Az *AddRoundKey*() bemenete az S állapottáblán kívül a $K' = K[n \cdot Nb \ldots (n+1) \cdot Nb - 1]$, $0 \le n \le Nr$ segédkulcs-tömb, amely a $KeyExpansion()$ által generált K tömb Nb szavas része. A $K[\ell \ldots m]$ itt a K tömb azon résztömbjét jelöli, amely ℓ indexű szótól az m indexű szóig terjed ($K[\ell \ldots m]$ mérete tehát $m-\ell+1$ szó $= 4(m-\ell+1)$ bájt $= 32(m-\ell+1)$ bit, $K[m \ldots m]$-et $K[m]$-mel fogjuk jelölni). Az *AddRoundKey*() szerepe annyi, hogy minden egyes menet alkalmával hozzáadja (XOR művelettel) a menetnek megfelelő segédkulcsot az állapottábla bájtjaihoz. Formálisan:

$$\begin{pmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{pmatrix} = \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \oplus \begin{pmatrix} K'[j]_0 \\ K'[j]_1 \\ K'[j]_2 \\ K'[j]_3 \end{pmatrix} = \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix} \oplus \begin{pmatrix} K[n \cdot Nb + j]_0 \\ K[n \cdot Nb + j]_1 \\ K[n \cdot Nb + j]_2 \\ K[n \cdot Nb + j]_3 \end{pmatrix} \qquad 0 \le j < Nb,$$

  ahol $K'[j]_i$ a K' tömb j indexű szavának i indexű bájtját jelenti. Az *AddRoundKey*() eljárást a 2.22 ábra is szemlélteti.

- A $KeyExpansion()$ a k kulcsból legenerálja a K segédkulcs-tömböket tartalmazó tömböt. Az algoritmus futása során összesen $Nr + 1$-szer adódik hozzá a segédkulcs-tömb az állapottáblához, és az állapottábla (valamint a segédkulcs-tömb) mérete $Nb = 4$ szó, ezért a kimeneti K tömb mérete $Nb(Nr + 1)$ szó. A $K[i]$ a K tömb i indexű szavát jelöli, $0 \le i < Nb(Nr + 1)$, k_i pedig a k kulcs i indexű bájtját. A $KeyExpansion()$ eljárás pszeudokódja a következő:

2.22. ábra. Az *AddRoundKey*() transzformáció



Algorithm $KeyExpansion()$:
Input: Nk, k
Output: K
$i := 0$
while $i < Nk$ **do**
$K[i] := k_{4i} \parallel k_{4i+1} \parallel k_{4i+2} \parallel k_{4i+3}$
$i := i + 1$
end while
$i := Nk$
while $i < Nb(Nr + 1)$ **do**
$temp := K[i - 1]$
if $i \bmod Nk = 0$ **then**
$temp := SubWord(RotWord(temp)) \oplus Rcon[i/Nk]$
else
if $(Nk > 6)$ and $(i \bmod Nk = 4)$ **then**
$temp := SubWord(temp)$
end if
end if
$K[i] := K[i - Nk] \oplus temp$
$i := i + 1$
end while
return K
end Algorithm

A $SubWord()$ eljárás bemenete is, kimenete is egy négy bájtos szó. A bemeneti szó bájtait a $SubBytes()$ bemutatásánál ismertetett módon, a 2.7 táblázatban ábrázolt S-doboz segítségével transzformálja, ezek fogják alkotni a kimeneti szó bájtjait. Ha a $RotWord()$ bemenete egy $[a_0, a_1, a_2, a_3]$ szó, akkor a bájtok ciklikus balra tolása után a kimenete $[a_1, a_2, a_3, a_0]$ lesz. Az $Rcon()$ egy menet-konstans szót ad vissza, meghatározása:

$$Rcon(i) = [r_i, 0, 0, 0],$$

ahol r_i az $X^{i-1} \in \mathbb{F}_{2^8} \cong \mathbb{Z}_2[X]/(f)$ polinom együtthatóinak mefelelő bájt (lásd a $SubBytes()$ leírását). Az r_i értékeit (hexadecimális számrendszerben) a következő táblázatban foglaltuk össze:

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
r_i	01	02	04	08	10	20	40	80	1B	36	6C	D8	AB	4D	9A

Dekódolás: A kódolásnál bemutatott eljárások mindegyike invertálható, ezért a dekódoló eljárást könnyen fel lehet építeni. A dekódoló eljárás pszeudokódja az alábbi:

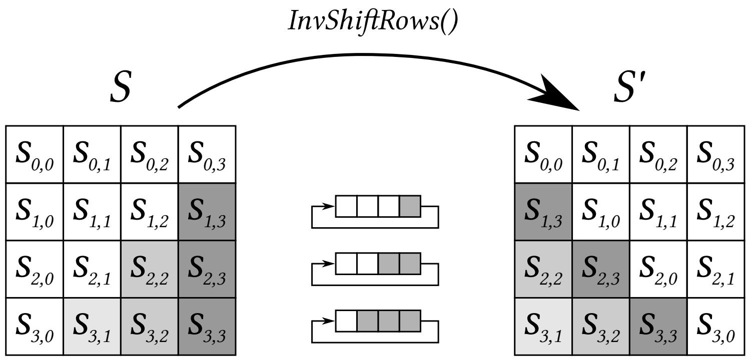
Algorithm AES (D_k):
Input: C, Nk, k
Output: P

$S := C$
$K := KeyExpansion(Nk, k)$
$S := AddRoundKey(S, K[Nr \cdot Nb \ldots (Nr + 1) \cdot Nb - 1])$
for $n := Nr - 1, \ldots, 1$ **do**
  $S := InvShiftRows(S)$
  $S := InvSubBytes(S)$
  $S := AddRoundKey(S, K[n \cdot Nb \ldots (n + 1) \cdot Nb - 1)$
  $S := InvMixColumns(S)$
end for
$S := InvShiftRows(S)$
$S := InvSubBytes(S)$
$S := AddRoundKey(S, K[0 \ldots Nb - 1])$
$P := S$
return P

end Algorithm

A következőkben megadjuk a dekódoló algoritmusban szereplő eljárások leírását. A jelölések és a matematikai háttér ugyanaz, mint a megfelelő eljárásnál a már bemutatott kódoló algoritmusban.

2.23. ábra. Az $InvShiftRows()$ transzformáció



- Az $InvShiftRows()$ a $ShiftRows()$ eljárás inverze. Az első sor változatlan marad, a második sort egy, a harmadik sort kettő a negyedik sort pedig három pozícióval tolja ciklikusan jobbra. Formálisan:

$$s'_{i,(j+shift(i,Nb)) \bmod Nb} = s_{i,j} \quad 0 < i < 4,\ 0 \le j < Nb$$

  A 2.23 ábrán szemléltetjük az $InvShiftRows()$ transzformációt.
- Az $InvSubBytes()$ a $SubBytes()$ inverze, vagyis $(T_2 \circ T_1)^{-1} = T_1^{-1} \circ T_2^{-1}$. A transzformáció S-doboza a 2.8 táblázatban látható.
- Az $InvMixColumns()$ a $MixColumns()$ inverze, az S állapottábla bájtjait oszloponként helyettesíti. Formálisan: $s' = a^{-1} \otimes s$, ahol

$$a^{-1} = 11X^3 + 13X^2 + 9X + 14 = (0B)_{16}X^3 + (0D)_{16}X^2 + (09)_{16}X + (0E)_{16},$$

  a $MixColumns()$ eljárásnál használt $a \in \mathbb{F}_{2^8}[X]/(f')$ polinom inverze.

$$\begin{pmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0d & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{pmatrix},$$

  vagyis

$$s'_{0,j} = ((0E)_{16} \bullet s_{0,j}) \oplus ((0B)_{16} \bullet s_{1,j}) \oplus ((0D)_{16} \bullet s_{2,j}) \oplus ((09)_{16} \bullet s_{3,j})$$
$$s'_{1,j} = ((09)_{16} \bullet s_{0,j}) \oplus ((0E)_{16} \bullet s_{1,j}) \oplus ((0B)_{16} \bullet s_{2,j}) \oplus ((0D)_{16} \bullet s_{3,j})$$
$$s'_{2,j} = ((0D)_{16} \bullet s_{0,j}) \oplus ((09)_{16} \bullet s_{1,j}) \oplus ((0E)_{16} \bullet s_{2,j}) \oplus ((0B)_{16} \bullet s_{3,j})$$
$$s'_{3,j} = ((0B)_{16} \bullet s_{0,j}) \oplus ((0D)_{16} \bullet s_{1,j}) \oplus ((09)_{16} \bullet s_{2,j}) \oplus ((0E)_{16} \bullet s_{3,j})$$

Kriptoanalízis: Az XSL támadás egyelőre az egyetlen, amely – szerzői szerint – a jövőben ígéretesnek bizonyulhat.

2.8. táblázat. Az $InvSubBytes()$ eljárás S-doboza

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

3. fejezet

Aszimmetrikus kulcsú titkosítás

Az eddig tanulmányozott szimmetrikus (titkos) kulcsú rendszereknél az E_k titkosító eljárás k kulcsa illetve a $D_{k'}$ dekódolási eljárás k' kulcsa titkosak voltak és legtöbb esetben megegyeztek (vagy ha nem is, akkor az egyik kiszámítható volt a másikból, polinomiális idő alatt).

Az aszimmetrikus kulcsú kriptorendszerek – vagy más néven nyilvános kulcsú kriptorendszerek – esetében az E_k kódoló eljárás kulcsa is nyilvános, tehát az egyetlen titkos adat a $D_{k'}$ dekódoló eljárás k' kulcsa marad. Ebben az esetben nyilván a k kulcs ismerete nem vezethet el a k' kulcs ismeretéhez, tehát k-ból k'-et csak hosszú, exponenciális idő alatt futó algoritmussal lehet megkapni. A nyilvános kulcsú kriptográfia elindítói W. Diffie és M. Hellman, tudományos cikkük 1976-ban jelent meg.

Ahhoz, hogy szerkesszünk egy E_k, $D_{k'}$ párt k nyilvános és k' titkos kulccsal, szerkesztenünk kell egy $f = E_k$ bijektív függvényt úgy, hogy f nyilvános és értékeit rövid idő alatt könnyen kiszámíthatjuk, $f^{-1} = D_{k'}$ értékeit viszont csak nagyon nehezen (nagyon hosszú idő alatt) tudjuk meghatározni, kivéve ha ismerjük a k' kiskaput (a dekódolás titkos kulcsát). A kiskapu ismerete $f^{-1} = D_{k'}$ számolását hatékonnyá teszi.

Az előbbi tulajdonsággal rendelkező f függvényt *csapóajtó-függvény*nek nevezzük. Az ilyen függvények képezik az alapját valamennyi nyilvános kulcsú rendszernek.

A csapóajtó-függvények előfutárai voltak az úgynevezett *egyirányú függvény*ek. Egy f bijektív függvény egyirányú, ha f könnyen számolható, f^{-1} értékeit viszont minden esetben nehéz meghatározni. 1974-ben G. Purdy szerkesztette meg a következő egyirányú függvényt: $f : \mathbb{Z}_p \to \mathbb{Z}_p$, $p = 2^{64} - 59$ prímszám,

$$f(x) = x^{2^{24}+17} + a_1 x^{2^{24}+3} + a_2 x^3 + a_3 x^2 + a_4 x + a_5,$$

ahol az a_i értékek tetszőleges 19 számjegyű egészek.

Amint az értelmezésükből látható, az egyirányú függvények alkalmatlanok aszimmetrikus kulcsú rendszerekben való felhasználásra, hiszen minden esetben nehezen invertálhatóak (nincsenek kiskapuk, vagy hátsó ajtók). Csapóajtó-függvények esetében f^{-1} kiszámításának nehézsége általában egy NP-feladatra vezethető vissza (amely azonban a kiskapu ismeretében elkerülhető). Pontosabban, ha nem ismerjük a kiskaput jelentő plusz információt, akkor ahhoz, hogy meghatározzuk $f^{-1} = D_{k'}$-et, látszólag meg kell oldanunk egy NP-feladatot. Csak látszólag, ugyanis megtörténhet, hogy akár a kiskapu ismerete nélkül is elkerülhetjük az NP-feladat megoldását (lásd a következő alfejezetben leírt knapsack rendszert). Másfelől arra is gondolni kell, hogy az NP-feladatok idővel átkerülhetnek a P-osztályba – persze ez nem valószínű az NP-teljes (tehát különösen nehéz) feladatok esetében. Az eddig leírtakból látszik, hogy szinte lehetetlen szigorú matematikai értelemben bizonyítani egy függvényről, hogy csapóajtó-függvény-e. Többnyire csak empirikus (tapasztalati) tények támasztják alá az egyes aszimmetrikus kulcsú rendszerek biztonságát.

Egy másik hátránya a nyilvános kulcsú rendszereknek, hogy gyakran lassúbbak a szimmetrikus rendszereknél (például mert sokszor moduláris hatványozást végeznek, ami sokkal több időt igényel, mint néhány XOR).

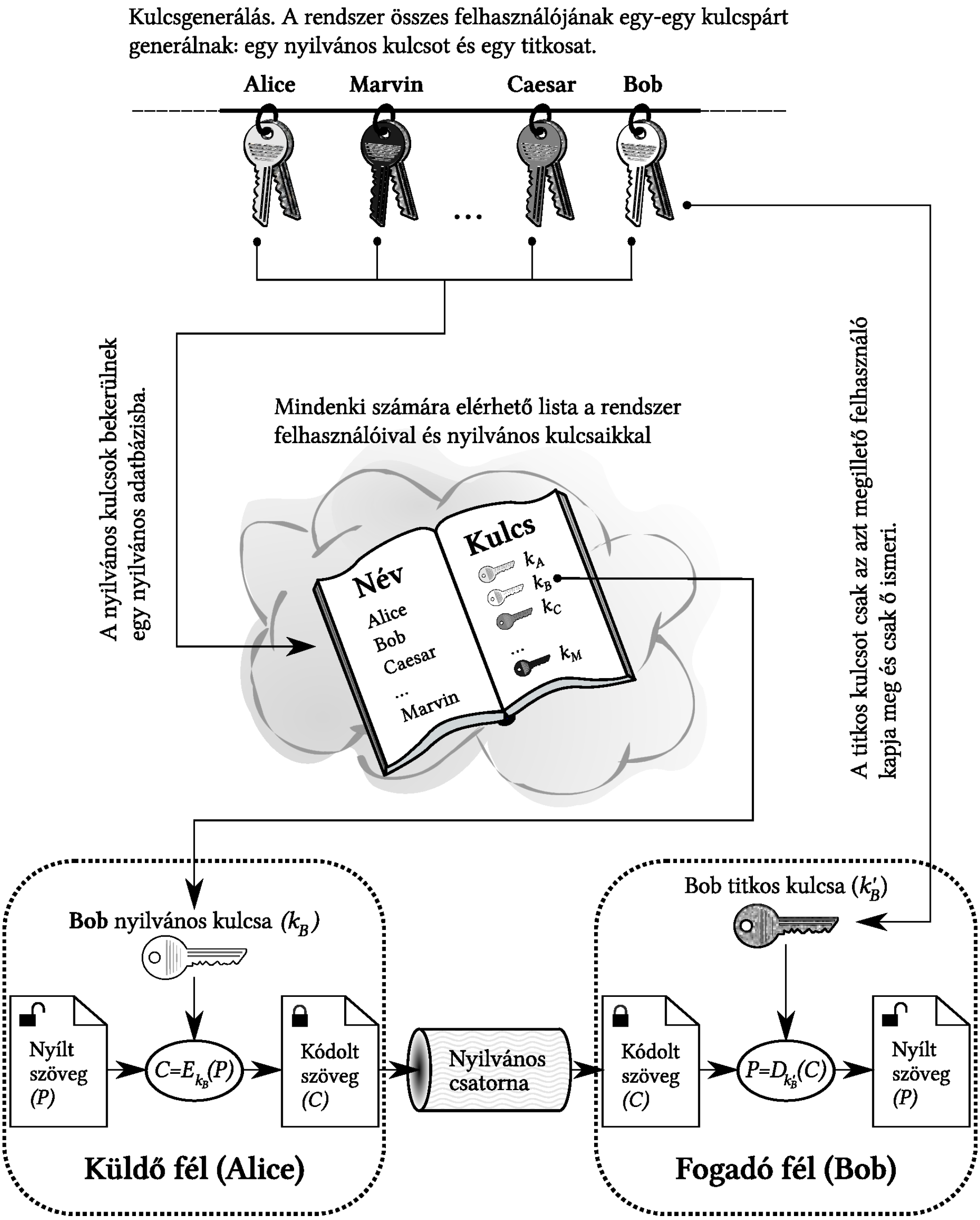
Az aszimmetrikus kulcsú rendszerek előnye azonban a sokkal gazdaságosabb és biztonságosabb kulcskezelés (a szimmetrikus kulcsú rendszerekhez viszonyítva). Valóban, ha egy hálózatban n személy szeretne páronként titkosan kommunikálni, akkor aszimmetrikus kulcsú rendszer esetében csupán n titkos kulcsra van szükség (mindenki kap egy titkos dekódoló kulcsot, a kódoló kulcs nyilvános), szemben a $C_n^2 = \frac{n(n-1)}{2}$ különböző kulccsal, amelyeket szimmetrikus rendszer használata esetében kell szétosztani. A 3.1 ábrán láthatjuk a nyilvános kulcsú kriptorendszerek sémáját. Az ábra olyan rendszert mutat be, ahol a kulcspárokat a rendszer felhasználói egy központi generáló egységtől kapják. A nyilvános kulcsok bekerülnek egy telefonkönyvszerű nyilvános listába, a titkos kulcsokat minden egyes felhasználó titokban tartja. Ha például Alice titkosított üzenetet akar küldeni Bobnak, akkor először megkeresi ebben a listában Bobnak a nyilvános kulcsát, és ezzel titkosítja az üzenetet, a nyilvános csatornán elküldi, Bob pedig a saját titkos kulcsával dekódolja azt. Továbbá fontos, hogy az úgynevezett digitális aláírásokat csak nyilvános kulcsú rendszerekkel lehet megvalósítani (lásd 6. fejezet).

A hátrányokat és előnyöket mérlegelve, a legjobb, ha maga a rendszer szimmetrikus, de a kulcskezelés (kulcscsere, kulcsok kiosztása) nyilvános kulcsú rendszer segítségével valósul meg (lásd a Diffie–Hellman-kulcscserét).

3.1. A Merkle–Hellman (knapsack) kriptorendszer

Ennek a rendszernek a csapóajtó-függvénye (pontosabban ennek invertálása) egy NP-teljes feladatra, a hátizsák (knapsack) feladatra épül. R. Merkle és M. Hellman publikálták a módszert 1978-ban.

A *hátizsák-probléma* (kissé átfogalmazva): adott egy $(v_0, v_1, \ldots, v_{n-1})$ pozitív egészekből álló sorozat és egy V szintén pozitív egész szám. Találjunk egy n-bites egész számot – legyen ez $p = (\varepsilon_{n-1}\varepsilon_{n-2}\ldots\varepsilon_1\varepsilon_0)_2$, ahol $\varepsilon_i \in \{0, 1\}$ a p bináris felírásában szereplő számjegyek úgy, hogy $\varepsilon_0 v_0 + \varepsilon_1 v_1 + \cdots + \varepsilon_{n-1} v_{n-1} = V$. Vegyük észre, 3.1. ábra. Aszimmetrikus (nyilvános) kulcsú titkosítási rendszer



hogy ennek a feladatnak lehet, hogy nincs megoldása, de az is megeshet, hogy több megoldása is van.

3.1.1. példa. $(v_0, v_1, v_2, v_3, v_4) = (2, 3, 1, 4, 5)$, $V = 5$. *Ekkor* $v_0 + v_1 = V$, $v_2 + v_3 = V$ *és* $v_5 = V$ *tehát* $p_1 = (11000)_2 = 40$, $p_2 = (00110)_2 = 6$, *illetve* $p_3 = (00001)_2 = 1$ *mind jó megoldások.*

3.1.2. példa. $(v_0, v_1, v_2) = (10, 20, 30)$, $V = 5$. *Nyilván nincs megoldás.*

Ismert tény, hogy az általános hátizsák-probléma NP-teljes, vagyis ha elfogadjuk a $P \neq NP$ sejtést, akkor nem adható meg egy n-ben és $\log b$-ben polinomiális megoldási algoritmus, ahol b felső korlátja V-nek és $(v_0, v_1, \ldots, v_{n-1})$-nek. Ahhoz azonban, hogy a hátizsák-problémára csapóajtó-függvényt építsünk, szükségünk lesz a feladat egy könnyen megoldható változatára, amely éppen a kiskapu segítségével vezethető le az általános, nehéz feladatból. Ez a könnyen megoldható változat az úgynevezett *szupernövekvő hátizsák feladat*, amikor a $(v_0, v_1, \ldots, v_{n-1})$ pozitív egészekből álló sorozatra igaz, hogy $v_1 > v_0$, $v_2 > v_1 + v_0$, $\ldots$, $v_{n-1} > v_{n-2} + \cdots + v_1 + v_0$, vagyis mindegyik egész nagyobb, mint az előtte levő egészek összege (ebben az esetben azt mondjuk, hogy $v = (v_0, v_1, \ldots, v_{n-1})$ *szupernövekvő sorozat*).

3.1.3. példa. *A* $(3, 5, 9, 20, 40)$ *sorozat segítségével szupernövekvő hátizsák feladatot lehet megadni.*

Ahogy már említettük, a szupernövekvő hátizsák feladat könnyen megoldható (egészen pontosan $O(n \log b)$ idő alatt) a következő módon (az algoritmus leírásában használt vátozók az előbbi jelölést követik).

Algorithm SuperIncreasingKnapsack:
Input: $V, (v_0, v_1, \ldots, v_{n-1})$
Output: p
  $s := V$
  for $i := n-1, \ldots, 0$ **do**
    if $v_i \le s$ **then**
      $\varepsilon_i := 1$
      $s := s - v_i$
    else
      $\varepsilon_i := 0$
    end if
  end for
  if $s = 0$ **then**
    $p := (\varepsilon_{n-1}\varepsilon_{n-2}\ldots\varepsilon_1\varepsilon_0)_2$
    return p
  else
    return „Nincs megoldás”
  end if
end Algorithm

3.1.4. példa. *Adott a* $(v_0, v_1, v_2, v_3, v_4) = (3, 5, 9, 20, 40)$, $V = 32$ *szupernövekvő hátizsák feladat. Látható, hogy*

$$i_0 = 3 \Rightarrow \varepsilon_3 = 1,\ \varepsilon_4 = 0 \textit{ és } V - v_{i_0} = 32 - 20 = 12 > 0$$

$$i_1 = 2 \Rightarrow \varepsilon_2 = 1 \textit{ és } V - v_{i_0} - v_{i_1} = 32 - 20 - 9 = 3 > 0$$

$$i_2 = 0 \Rightarrow \varepsilon_0 = 1,\ \varepsilon_1 = 0 \textit{ és } V - v_{i_0} - v_{i_1} - v_{i_2} = 0,$$

tehát a megoldás $p = (10110)_2 = 42$.

Térjünk rá most a Merkle–Hellman-féle kriptorendszer leírására.

Kulcs: Választunk egy szupernövekvő $v = (v_0, v_1, \ldots, v_{n-1})$ sorozatot, egy $m \in \mathbb{N}$-t úgy, hogy $m > \sum_{i=0}^{n-1} v_i$ és $a \in \mathbb{N}$-t úgy, hogy $(a, m) = 1$ és $0 < a < m$. Ezeket az adatokat véletlenszerűen generálhatjuk a következő módon. Először kiválasztunk egy véletlenszerű pozitív egészekből álló $n+1$ hosszúságú $z_0, \ldots, z_n$ sorozatot. Legyen $v_0 = z_0$, $v_i = z_i + v_{i-1} + v_{i-2} + \cdots + v_0$, $i = \overline{1, n-1}$, $m = z_n + \sum_{i=0}^{n-1} v_i$. Ezután választunk egy szintén véletlenszerű $a_0 < m$ értéket. Legyen a az első pozitív egész úgy, hogy $a \ge a_0$ és $(a, m) = 1$. Ha megvannak az előbbi adatok, meghatározzuk (euklidészi algoritmussal) a $b = a^{-1} \bmod m$-met ($b < m$) és a $w = (w_0, \ldots, w_{n-1})$, $w_i = av_i \bmod m$, $w_i < m$ sorozatot. Ekkor a nyilános kódolási kulcs $k = (w_0, \ldots, w_{n-1})$, a titkos dekódolási kulcs pedig

$k' = (b, m)$, ahonnan azonnal megvan a és k ismeretében $(v_0, v_1, \ldots, v_{n-1})$, hiszen $bw_i = v_i \bmod m$.

Kódolás: A P üzenet most egy n-bites tömb, vagyis $P = (\varepsilon_{n-1}\varepsilon_{n-2}\ldots\varepsilon_1\varepsilon_0)_2$. Ha például angol ábécét használó szövegünk van, akkor minden betű az ábécébeli szorszámán alapulva 5 biten ábrázolható:

$$\begin{array}{c} \mathbf{A} \to 0 = (00000)_2 \\ \mathbf{B} \to 1 = (00001)_2 \\ \vdots \\ \mathbf{Z} \to 25 = (11001)_2 \end{array}$$

Ekkor $C = E_k(P) = \sum_{i=0}^{n-1} \varepsilon_i w_i \in \mathbb{N}^*$.

Dekódolás: Legyen $V = bC \bmod m$, $V < m$. Ekkor

$$\begin{aligned} V &= bC \bmod m \\ &= \sum_{i=0}^{n-1} \varepsilon_i b w_i \bmod m \\ &= \sum_{i=0}^{n-1} \varepsilon_i b a v_i \bmod m \\ &= \sum_{i=0}^{n-1} \varepsilon_i v_i \bmod m, \end{aligned}$$

ahonnan $V = \sum_{i=0}^{n-1} \varepsilon_i v_i$, hiszen $V < m$ és $\sum_{i=0}^{n-1} \varepsilon_i v_i \le \sum_{i=0}^{n-1} v_i < m$. V-re és $(v_0, v_1, \ldots, v_{n-1})$-re megoldva a szupernövekvő hátizsák feladatot, megkapjuk $P = (\varepsilon_{n-1}\varepsilon_{n-2}\ldots\varepsilon_1\varepsilon_0)_2$-t, ami az egyetlen megoldás.

3.1.5. példa. *Az előbbi jelöléseket használva legyen a szupernövekvő sorozat* $v = (3, 5, 9, 20, 40)$, $m = 79$, $a = 17$. *Ekkor* $b = 14$, $w = (3 \cdot 17 \bmod 79, 5 \cdot 17 \bmod 79, 9 \cdot 17 \bmod 79, 20 \cdot 17 \bmod 79, 40 \cdot 17 \bmod 79) = (51, 6, 74, 24, 48)$. *Tehát a nyilvános kulcs* $k = w = (51, 6, 74, 24, 48)$, *a titkos kulcs pedig* $(14, 79)$. *A* **SZIA** *szót a következő módon titkosítjuk:*

$$\begin{array}{lcl} \boldsymbol{S} \to 18 = (10010)_2 = P_1 & \Rightarrow & C_1 = E_k(P_1) = 51 + 24 = 75 \\ \boldsymbol{Z} \to 25 = (11001)_2 = P_2 & \Rightarrow & C_2 = E_k(P_2) = 51 + 6 + 48 = 105 \\ \boldsymbol{I} \to 8 = (01000)_2 = P_3 & \Rightarrow & C_3 = E_k(P_3) = 6 \\ \boldsymbol{A} \to 0 = (00000)_2 = P_4 & \Rightarrow & C_4 = E_k(P_4) = 0. \end{array}$$

A **SZIA** *titkosított változata tehát* $(75, 105, 6, 0)$ *lesz. A dekódolásnál w és b ismeretében megvan* $v = (3, 5, 9, 20, 40)$*, így*

$$\begin{aligned}
V_1 &= bC_1 \bmod m = 23 = 1 \cdot 3 + 1 \cdot 20 && \Rightarrow P_1 = (10010)_2 = 18 \rightarrow \mathbf{S}\\
V_2 &= bC_2 \bmod m = 48 = 1 \cdot 3 + 1 \cdot 5 + 1 \cdot 40 && \Rightarrow P_2 = (11001)_2 = 25 \rightarrow \mathbf{Z}\\
V_3 &= bC_3 \bmod m = 5 = 1 \cdot 5 && \Rightarrow P_3 = (01000)_2 = 8 \ \ \rightarrow \mathbf{I}\\
V_4 &= bC_4 \bmod m = 0 && \Rightarrow P_4 = (00000)_2 = 0 \ \ \rightarrow \mathbf{A}
\end{aligned}$$

3.1.1. megjegyzés. *A fenti példa természetesen nem reális méretű* ($n = 5$)*. Az eredeti rendszerleírásban a javasolt értékek:* $n = 100$*, m 200 bites, a* $(v_0, v_1, \ldots, v_{n-1})$ *sorozat elemei pedig 100 bites értékektől növekednek 199 bites értékek felé.*

Kriptoanalízis: A $k = (w_0, \ldots, w_{n-1})$ nyilvános kulcs nem szupernövekő ugyan, de nagyon speciális, hiszen egy szupernövekvő sorozatból származik, amely meg van szorozva egy a értékkel modulo m. 1982-ben Shamir megadott egy n-ben polinomiális algoritmust, amely megoldja előbbi típusú hátizsák-feladatot. Az algoritmus talál egy (b', m') párt (nem feltétlenül a titkos kulcsot), amellyel $v' = (v'_0, v'_1, \ldots, v'_{n-1})$ szupernövekvő, ahol $v'_i = b'w_i \bmod m'$. A lényeg az, hogy nem csak egy ilyen pár létezik, hanem aránylag elég sok, tehát nem kell feltétlenül eltalálnunk a titkos kulcsot. Shamir támadása kivédhető az úgynevezett iterált hátizsák rendszer segítségével. Legyen $v = (v_0, v_1, \ldots, v_{n-1})$ egy szupernövekvő sorozat és m_1, m_2, a_1, a_2 olyan, hogy $m_1 > \sum_{i=0}^{n-1} v_i$, $m_2 > nm_1$, $(a_1, m_1) = 1$ és $(a_2, m_2) = 1$ – lehetőleg ezek az értékek véletlenszerűek. Megszerkesztjük a $w = (w_0, \ldots, w_{n-1})$, $w_i = a_1 v_i \bmod m_1$ és $u = (u_1, \ldots, u_{n-1})$, $u_i = a_2 w_i \bmod m_2$ sorozatokat. Ekkor a nyilvános kulcs $k = u = (u_1, \ldots, u_{n-1})$, a kódoló függvény pedig $C = E_k(P) = \varepsilon_0 u_0 + \cdots + \varepsilon_{n-1} u_{n-1}$. A titkos kulcs $k' = (b_1, m_1, b_2, m_2)$, ahol $b_1 = a_1^{-1} \bmod m_1$ és $b_2 = a_2^{-1} \bmod m_2$, a dekódoló folyamat pedig: előbb kiszámoljuk $V' = b_2 C \bmod m_2$-t, majd $V = b_1 V' \bmod m_1$-et; mivel $m_2 > nm_1 > \sum_{i=0}^{n-1} w_i \Longrightarrow V' = \sum_{i=0}^{n-1} \varepsilon_i w_i$ és innen $V = b_1 \sum_{i=0}^{n-1} \varepsilon_i w_i \bmod m_1 = \sum_{i=0}^{n-1} \varepsilon_i v_i$, hiszen $m_1 > \sum_{i=0}^{n-1} v_i \geq \sum_{i=0}^{n-1} \varepsilon_i v_i$. Az iterált hátizsák-feladatra nem létezik polinomiális feltörési algoritmus, de kimutatták, hogy számos gyenge pontja van különféle támadási technikákkal szemben.

3.2. Az RSA

Az RSA szerzői R. Rivest, A. Shamir és L. Adleman, 1978-ban publikálták először. Az RSA elnevezés a szerzők neveinek kezdőbetűiből származik. Elvi alapja, hogy könnyebb két nagyon nagy prímszámot generálni és összeszorozni, mint szorzatukat faktorizálni.

Kulcs: Válasszunk véletlenszerűen két, legalább 100 számjegyű prímszámot úgy, hogy az egyik (kettes számrendszerben felírva) néhány bittel hosszabb, mint a másik (lásd a 2. függeléket). Jelölje p és q ezeket a prímeket, legyen $n = pq$ és $\varphi(n) = (p-1)(q-1) = n - p - q + 1$ (az Euler-függvény értéke n-ben). Válasszunk egy e egész számot 1 és $\varphi(n)$ között úgy, hogy $(e, \varphi(n)) = 1$ és e-nek lehetőleg minél kevesebb 1-es bitje legyen. Jó választás e-re például $17 = 2^4+1$ és $65537 = 2^{16}+1$ feltéve, hogy ezek egyike sem osztja $\varphi(n)$-et. A 17 és 65537 is prímszám, tehát ha nem osztják $\varphi(n)$-et, akkor relatív prímek vele. Legyen $d = e^{-1} \bmod \varphi(n)$. Ekkor a nyilvános kulcs $k = (n, e)$, a titkos kulcs pedig $k' = d$.

Kódolás: Tételezzük fel, hogy a szövegünk egy N betűs ábécében íródott. Legyen ℓ olyan, hogy $N^\ell \le n \le N^{\ell+1}$, vagyis $\ell = \lfloor \log_N n \rfloor$. Legyen a P nyílt üzenet egy ℓ betűs tömb. Ez azt jelenti, hogy

$$P = (b_{\ell-1} \dots b_0)_N = b_{\ell-1}N^{\ell-1} + \cdots + b_1 N + b_0 < N^\ell \le n,$$

tehát $P \in \mathbb{Z}_n$. Ekkor $C = E_k(P) = P^e \bmod n$, tehát $C \in \mathbb{Z}_n$, $C < n < N^{\ell+1}$, vagyis C egy $\ell + 1$ betűs tömb.

Dekódolás: $P = D_{k'}(C) = C^d \bmod n$.

Ahhoz, hogy érvényes kriptorendszerünk legyen, be kell látni, hogy $D_{k'} = E_k^{-1}$ mint függvények, vagyis hogy $D_{k'}(E_k(P)) = P$, ($P^{ed} \equiv P \pmod n$), illetve $E_k(D_{k'}(C)) = C$ ($C^{de} \equiv C \pmod n$).

3.2.1. tétel. *Az előbbi jelölésekkel:* $a^{ed} \equiv a \pmod n$, $\forall a \in \mathbb{Z}$.

Bizonyítás. $ed \equiv 1 \mod \varphi(n) \Longrightarrow ed - 1 = \ell\varphi(n) \Longrightarrow ed = 1 + \ell\varphi(n)$.

1. eset: $(a, n) = 1$. Alkalmazzuk Euler tételét: $a^{\varphi(n)} \equiv 1 \pmod n$. Az egyenletet az ℓ-edik hatványra emeljük, $a^{\ell\varphi(n)} \equiv 1 \pmod n$, majd beszorozzuk a-val: $a^{1+\ell\varphi(n)} \equiv a \pmod n \Longrightarrow a^{ed} \equiv a \pmod n$.

2. eset: $(a, n) = p$. Ekkor $(a, q) = 1$. Alkalmazzuk a kis Fermat-tételt, ahonnan $a^{q-1} \equiv 1 \pmod q \Longrightarrow a^{\ell(p-1)(q-1)} \equiv 1 \pmod q \Longrightarrow a^{\ell\varphi(n)} \equiv 1 \pmod q \Longrightarrow a^{1+\ell\varphi(n)} \equiv a \pmod q \Longrightarrow a^{ed} \equiv a \pmod q \Longrightarrow q \mid (a^{ed} - a)$. De $p \mid a \Longrightarrow p \mid (a^{ed} - a) \Longrightarrow n = pq \mid (a^{ed} - a) \Longrightarrow a^{ed} \equiv a \pmod n$.

3. eset: $(a, n) = q$. A 2. esethez hasonlóan járunk el.

4. eset: $(a, n) = n$. Ekkor $a^{ed} \equiv a \equiv 0 \pmod n$. □

A következő példa nem reális, hiszen a benne szereplő prímek túl kicsik, viszont jól szemlélteti az RSA működését.

3.2.1. példa. *Legyen* $p = 19$, $q = 41$, *tehát* $n = 779$. $\varphi(n) = 18 \cdot 40 = 720$. *Legyen továbbá* $e = 17$ *(látható, hogy* $(e, \varphi(n)) = 1$*) és ekkor* $d = 17^{-1} \mod 720 = 593$. *Nyilvánossá tesszük tehát a* $k = (n, e) = (779, 17)$ *kulcsot és titkos marad* $k' = d = 593$.

A szöveg, amelyet titkosítani szeretnénk, angol ábécét használ ($N = 26$)*. Legyen tehát a titkosítandó üzenet a következő:* **SZIASZTOKXBU**. *Mivel* $26^2 < 779 < 26^3$, *kétbetűs tömböket kódolunk. A tömbök a következők lesznek:*

$$\begin{aligned}
P_1 &= \mathbf{SZ} = 18 \cdot 26 + 25 = 493\\
P_2 &= \mathbf{IA} = 8 \cdot 26 + 0 \quad = 208\\
P_3 &= \mathbf{SZ} = 18 \cdot 26 + 25 = 493\\
P_4 &= \mathbf{TO} = 19 \cdot 26 + 14 = 508\\
P_5 &= \mathbf{KX} = 10 \cdot 26 + 23 = 283\\
P_6 &= \mathbf{BU} = 1 \cdot 26 + 20 \quad = 46
\end{aligned}$$

A kódolt tömbök:

$$\begin{aligned}
C_1 &= P_1^e \bmod n = 493^{17} \bmod 779 = 0 \cdot 26^2 + 18 \cdot 26 + 25 = \mathbf{ASZ}\\
C_2 &= P_2^e \bmod n = 208^{17} \bmod 779 = 0 \cdot 26^2 + 8 \cdot 26 + 0 \quad = \mathbf{AIA}\\
C_3 &= P_3^e \bmod n = 493^{17} \bmod 779 = 0 \cdot 26^2 + 18 \cdot 26 + 25 = \mathbf{ASZ}\\
C_4 &= P_4^e \bmod n = 508^{17} \bmod 779 = 0 \cdot 26^2 + 13 \cdot 26 + 0 \quad = \mathbf{ANA}\\
C_5 &= P_5^e \bmod n = 283^{17} \bmod 779 = 0 \cdot 26^2 + 6 \cdot 26 + 24 \quad = \mathbf{AGY}\\
C_6 &= P_6^e \bmod n = 46^{17} \bmod 779 \quad = 1 \cdot 26^2 + 0 \cdot 26 + 1 \quad = \mathbf{BAB}
\end{aligned}$$

A **SZIASZTOKXBU** *kódolt megfelelője tehát* **ASZAIAASZANAAGYBAB**.

Az RSA hatékonyságának lényege, hogy a d titkos kulcs ismerete tulajdonképpen ekvivalens az n szám prímtényezőinek ismeretével (lásd még a következő tételt), mely közismerten nehéz probléma, ha a prímek jól vannak megválasztva. Az eddig ismert legjobb faktorizációs algoritmusok is exponenciális idő alatt futnak.

3.2.2. tétel. *Ha az alábbi adatok közül az egyik ismert, a többi is megtalálható rövid (polinomiális) idő alatt: p, q, $\varphi(n)$, d.*

Bizonyítás. Ha p ismert $\Longrightarrow \varphi(n) = (p-1)(q-1)$ ismert, és $d \equiv e^{-1} \pmod{\varphi(n)}$-t meg lehet kapni euklidészi algoritmussal. Analóg módon járunk el akkor, ha q ismert.

Ha $\varphi(n)$ ismert, akkor $d \equiv e^{-1} \pmod{\varphi(n)}$-t szintén meg lehet kapni euklidészi algoritmussal, p és q pedig az alábbi egyenletrendszer megoldásai (az egyenletrendszer megoldható négyzetes idő alatt):

$$\begin{cases} p + q &= n - \varphi(n) - 1\\ pq &= n \end{cases}$$

Ha d ismert, igazoljuk, hogy p (vagy q) is meghatározható probabilisztikus polinomiális algoritmussal. Tulajdonképpen elég ismerni egy m értéket úgy, hogy $a^m \equiv 1$

$\pmod{n}$, bármely a-ra, ahol $(a, n) = 1$. Jelöljük T-vel m-nek ezt a tulajdonságát. Ha d ismert, akkor $m = ed - 1 = \ell\varphi(n)$ T tulajdonságú. Észrevehető, hogy ha m közös többszöröse $p - 1$ és $q - 1$-nek, akkor szintén T tulajdonságú. Valóban, ha $m = \ell_1(p - 1) = \ell_2(q - 1)$, akkor a kis Fermat-tétel alapján $a^{p-1} \equiv 1 \pmod{p}$, ha $p \nmid a$, illetve $a^{q-1} \equiv 1 \pmod{q}$, ha $q \nmid a$. Az első egyenletet az l_1-edik, a másodikat pedig az l_2-edik hatványra emelve kapjuk, hogy: $a^{\ell_1(p-1)} \equiv 1 \pmod{p}$ és $a^{\ell_2(q-1)} \equiv 1 \pmod{q}$, ha $(a, n) = 1$. Ebből következik, hogy $\left.\begin{array}{c} p \mid (a^m - 1) \\ q \mid (a^m - 1) \end{array}\right\} \Longrightarrow n = pq \mid (a^m - 1)$ $\Longrightarrow a^m \equiv 1 \pmod{n}$, ha $(a, n) = 1$. Tehát feltételezzük a továbbiakban, hogy ismerünk egy m-et, amely T tulajdonságú. Akkor $a = -1$-re $(-1)^m \equiv 1 \pmod{n}$, ahol n páratlan. Innen következik, hogy m páros. A továbbiakban megvizsgáljuk, hogy $\frac{m}{2}$ T tulajdonságú-e vagy sem. Igazolni lehet, hogy ha $\frac{m}{2}$ nem T tulajdonságú, akkor a lehetséges a-k (úgy, hogy $(a, n) = 1$) körülbelül felére $a^{\frac{m}{2}} \not\equiv 1 \pmod{n}$. Tehát, ha néhány a-ra kijön, hogy $a^{\frac{m}{2}} \equiv 1 \pmod{n}$, akkor már nagy valószínűséggel $\frac{m}{2}$ is T tulajdonságú. Tovább felezve és folytatva a gondolatmenetet, végül véges lépés után eljutunk egy m'-hez úgy, hogy m' nem T tuladonságú, vagyis $\exists a$, $(a, n) = 1$ úgy, hogy $a^{m'} \not\equiv 1 \pmod{n}$. Két lehetőségünk van:

1. eset: m' többszöröse $p - 1$-nek, de nem többszöröse $q - 1$-nek (vagy fordítva). Ekkor $a^{m'} \equiv 1 \pmod{p}$ minden a-ra, ahol $(a, n) = 1$, de az a-k körülbelül felére $a^{m'} \equiv -1 \pmod{q}$ (mert ha $a^{2m'} \equiv 1 \pmod{q} \Longrightarrow a^{2m'} - 1 \equiv 0 \pmod{q} \Longrightarrow (a^{m'} - 1)(a^{m'} - 1) \equiv 0 \pmod{q}$, és mivel $\mathbb{Z}_p$ test és nincsenek zérusosztói, következik, hogy $a^m \equiv \pm 1 \pmod{q}$). Tehát néhány a érték kipróbálása után találhatunk egyet, melyre $a^{m'} \not\equiv 1 \pmod{q}$, vagyis $q \nmid (a^{m'} - 1)$. De ekkor $(n, a^{m'} - 1) = p$.

2. eset: m' nem többszöröse sem $p - 1$-nek, sem $q - 1$-nek. Ekkor az a értékek körülbelül négyedére $a^{m'} \equiv 1 \pmod{p}$ és $a^{m'} \equiv 1 \pmod{q}$, szintén az a-k körülbelül negyedére $a^{m'} \equiv -1 \pmod{p}$ és $a^{m'} \equiv -1 \pmod{q}$, végül az a-k körübelül felére $a^{m'} \equiv \pm 1 \pmod{p}$ és $a^{m'} \equiv \mp 1 \pmod{q}$. Tehát néhány a érték kipróbálása után találhatunk egyet, melyre $a^{m'} \equiv 1 \pmod{p}$ és $a^{m'} \equiv -1 \pmod{q}$ vagy $a^{m'} \equiv 1 \pmod{q}$ és $a^{m'} \equiv -1 \pmod{p}$. Ekkor $(n, a^{m'} - 1) = p$ vagy $(n, a^{m'} - 1) = q$. □

Kriptoanalízis: Az alábbiakban felsorolunk pár kritériumot, amelyeknek betartásával elkerülhetünk néhány egyszerűbb támadást.

1. A prímszámokra vonatkozó kritériumok:

   (a) Legyenek véletlenszerűek (a 2. függelékben bemutatjuk, hogyan generálhatunk ilyeneket). Lehetőleg ne legyenek „híres” prímek (pl. Mersenne-prímek, Fermat-prímek stb.).

   (b) p és q ne legyen túl közel egymáshoz. Ha p és q értékek közel vannak egymáshoz, akkor $n = pq$ könnyen faktorizálható az úgynevezett Fermat-faktorizációval. Ennek lényege a következő: mivel $n = pq$ $\Longrightarrow \frac{(p+q)^2}{4} - n = \frac{(p-q)^2}{2}$, tehát ahogy p és q közelítenek egymáshoz

$p-q \to 0 \Longrightarrow \frac{p+q}{2} \to \sqrt{n}$. Ekkor $\sqrt{n}$-hez közeli x értékekkel próbálkozunk, megnézzük, hogy $x^2 - n$ teljes négyzet-e. Ha igen, mondjuk $x^2 - n = y^2$, akkor $\frac{p+q}{2} = x$ és $\frac{p-q}{2} = y \Longrightarrow p = x + y$, $q = x - y$.

2. $\varphi(n)$-re vonatkozó kritériumok. Láttuk, hogy nemcsak $\varphi(n)$ ismerete, hanem $p-1$, $q-1$ közös többszörösének ismerete is már elegendő a rendszer feltöréséhez. Ezért:
   (a) $\varphi(n)$-nek a prímtényezői között legyenek nagyobb prímek is (mert ha nem, $\varphi(n)$ próbálgatással meghatározható).
   (b) $[p-1, q-1]$ legyen lehetőleg nagy, mert ismerete ekvivalens a rendszer feltörésével. Nagyon rossz, ha például $(p-1) \mid (q-1)$ vagy fordítva, mert akkor a legkisebb közös többszörös, $[p-1, q-1]$ kicsi.
3. e-re vonatkozó kritériumok:
   (a) Jó, ha e kisebb, mert akkor $d = e^{-1} \bmod \varphi(n)$ nagyobb lesz.
   (b) Jó, ha e-nek kevés (például csak 1 vagy 2 darab) 1-es bitje van (ezért jó választást jelentenek e értékére a $17 = 2^4 + 1$ vagy $65537 = 2^{16} + 1$ prímek). Ez azért fontos, mert az e-vel való moduláris hatványozás az ismételt négyzetreemeléses módszerrel gyorsabb.
   (c) Lehetőleg $e \neq \frac{\varphi(n)}{2} + 1$, mert ha igen, akkor $e-1$ többszöröse $p-1$-nek és $q-1$-nek is. Ebből következik, hogy $a^{e-1} \equiv a \pmod{n}$, $\forall a \in \mathbb{N}$-re, ahol $(a, n) = 1$, és akkor $a^e \equiv a \pmod{n}$ bármely a-ra, így minden szövegtömb önmagába kódolódna.
4. A szövegtömbökre vonatkozó kritériumok:
   (a) Jó, ha mindegyik tömb végén valamilyen véletlenszerű, értelmetlen szövegrész van. Ezt kitöltésnek nevezzük (angolul padding). Kitöltés alkalmazás nélkül, mivel n és e nyilvánosak, a támadó fél beazonosíthat meghatározott titkosított szövegtömböket a kódolt üzenetünkben.
   (b) Megtörténhet, hogy bizonyos tömbök önmagukba kódolódnak. Például, ha
   $$\begin{cases} a_0 \equiv u \pmod{p} \\ a_0 \equiv v \pmod{q} \end{cases},$$
   ahol $(u, v) \in \{(1, 1), (-1, 1), (1, -1), (-1, -1)\}$, akkor $a_0^e \equiv a_0 \pmod{n}$. Tehát jó, ha $P \neq a_0$.

3.3. Diszkrét logaritmáláson alapuló rendszerek

Ezek a titkosítási rendszerek a diszkrét logaritmálás nehézségére alapoznak (lásd a 3. függeléket). Ha $\mathbb{F}_q$ q elemű véges test és $\mathbb{F}_q^* =< g >$, akkor $y = g^x$ aránylag könnyen meghatározható, de y-ból g ismeretében x-et nehéz meghatározni. A diszkrét logaritmálás körülbelül egy szám faktorizációjával azonos nehézségű feladat. Itt persze feltételezni kell, hogy q nagy (körülbelül 1024 bites) és $q-1$-nek van egy nagy prímosztója.

3.3.1. Shamir háromlépéses protokollja

Ezt a titkosítási módszert Shamir 1980-ban dolgozta ki.

Legyen q egy nagy prímszám. Minden X felhasználó választ magának egy titkos $e_X \in \{1, 2, \ldots, q-2\}$ kitevőt úgy, hogy $(e_X, q-1) = 1$, és meghatározza $d_X = e_X^{-1} \mod (q-1)$-et (euklidészi algoritmussal).

Alice a következő módon küldi el Bobnak a $P \in \{1, \ldots, q-1\}$ üzenetet:

1. lépés: Alice elküldi Bobnak $P^{e_A} \mod q$-t.
2. lépés: Bob elküldi Alice-nek $P^{e_A e_B} \mod q$-t.
3. lépés: Alice elküldi Bobnak $P^{e_A e_B d_A} = P^{e_B} \mod q$-t.

Bob a $P^{e_B} \mod q$-t dekódolni tudja d_B segítségével, hiszen $P^{e_B d_B} \equiv P \pmod q$. Itt felhasználtuk, hogy $P^{e_A d_A} \equiv P^{1+\ell(q-1)} \equiv P \pmod q$, hiszen a kis Fermat-tételből $P^{q-1} \equiv 1 \pmod q \Longrightarrow P^{\ell(q-1)} \equiv 1 \pmod q$. Hasonlóan $P^{e_B d_B} \equiv P \pmod q$.

3.3.2. A Massey–Omura-rendszer

1982-ben fejlesztették ki. Hasonlít az előbb ismertetett rendszerhez. Ami változik az az, hogy nem $\mathbb{Z}_p^*$-ban dolgozunk, hanem $\mathbb{F}_{2^n}^*$-ban. Most tehát $q = 2^n$. Minden X felhasználó választ magának egy titkos $e_X \in \{1, 2, \ldots, 2^n - 2\}$ kitevőt úgy, hogy $(e_X, 2^n - 1) = 1$, és meghatározza $d_X = e_X^{-1} \mod (2^n - 1)$-et.

Alice a következő módon küldi el Bobnak a $P \in \mathbb{F}_{2^n}^*$ üzenetet:

1. lépés: Alice elküldi Bobnak P^{e_A}-t ($\mathbb{F}_{2^n}^*$-ban).
2. lépés: Bob elküldi Alice-nek $P^{e_A e_B}$-t.
3. lépés: Alice elküldi Bobnak $P^{e_A e_B d_A} = P^{e_B}$-t.

Bob a P^{e_B}-t dekódolni tudja d_B segítségével, hiszen $P^{e_B d_B} = P$. Itt felhasználtuk, hogy az $\mathbb{F}_{2^n}^*$ csoportban, mely $2^n - 1$ elemű, $P^{2^n-1} = 1 \Longrightarrow P^{e_A d_A} = P^{1+\ell(2^n-1)} = P$. Hasonlóan $P^{e_B d_B} = P$.

A Massey–Omura-rendszer implementálása hatékonyabb és gyorsabb, mint a prímszám alapú Shamir-féle háromlépéses protokoll. Látható, hogy az előző két rendszerben a diszkrét logaritmálás nehézségére alapozunk. Fontos megemlíteni, hogy mindkét rendszerben a második lépésnél Bobnak egy aláírást is kell küldenie, hogy hitelesítse magát Alice előtt (lásd a 6. fejezetet). Ha ez nem így lenne, akkor egy harmadik fél (Marvin), aki megszerzi P^{e_A}-t a nyilvános csatornán, visszaküldhetné $P^{e_A e_M}$-et (saját exponensével hatványozva). Ha nincs semmilyen aláírás, Alice nincs honnan tudja, hogy az üzenet valójában nem Bobtól, hanem Marvintól jött.

A Diffie–Hellman-hipotézis

Legyen $\mathbb{F}_q$ egy véges test, $q = p^\ell$ elég nagy, $\mathbb{F}_q^* =< g >$ és $a, b \in \{1, \ldots, q-1\}$ véletlenszerűek. Ekkor g, g^a és g^b ismeretéből g^{ab} nem vezethető le, csakis diszkrét logaritmálással.

3.3.3. Az ElGamal titkosítási rendszer

Ezt a titkosítási rendszert 1985-ben szerkesztette T. ElGamal.

Feltételezzük, hogy $\mathbb{F}_q$ véges test rögzített ($q = p^\ell$ elég nagy) és $\mathbb{F}_q^* =< g >$. Ezek az adatok (g is) mind nyilvánosak.

Kulcs: Minden X felhasználó választ magának egy véletlenszerű $a_X \in \{1, \ldots, q-1\}$ titkos kulcsot, és nyilvánossá teszi $g^{a_X} \in \mathbb{F}_q^*$-ot. Tehát a nyilvános kulcs $k = g^{a_X}$, a titkos pedig $k' = a_X$.

Kódolás: Tételezzük fel, hogy Bob Alice-nek akar küldeni egy $P \in \mathbb{F}_q^*$ üzenetet. Véletlenszerűen választ egy $b \in \{1, \ldots, q-1\}$ egész számot, és elküldi a $(C, C') = (g^b, Pg^{a_A b})$ párt. Vegyük észre, hogy ez kiszámolható, hiszen g^{a_A} nyilvános (ez Alice nyilvános kulcsa).

Dekódolás: Alice kiszámolja $s = C^{a_A}$-t, majd $C's^{-1} = P$-t, és megkapja a nyílt szöveget. Valóban, $C's^{-1} = Pg^{a_A b}(g^{ba_A})^{-1} = P$. Itt fontos megemlíteni, hogy $s \in \mathbb{F}_q^*$-ből $s^{-1} \in \mathbb{F}_q^*$-t könnyen meg lehet kapni kiterjesztett euklidészi algoritmussal.
Más lehetőség: Alice kiszámolja $s' = C^{q-1-a_A}$-t és ekkor $C's' = P$. Valóban, $Pg^{a_A b}g^{b(q-1-a_A)} = Pg^{b(q-1)} = P$, mert $g^{q-1} = 1$.

Vegyük észre, hogy az ElGamal titkosítási rendszer biztonsága is a Diffie–Hellman-sejtésen, illetve a diszkrét logaritmálás nehézségén alapul. Valóban, g, g^{a_A} illetve g^b ismerete csak diszkrét logaritmálás útján fedi fel $s = C^{a_A} = g^{a_A b}$-t.

3.3.1. példa. *Legyen* $q = 65537$ *prímszám. Ekkor* $\mathbb{F}_q = \mathbb{Z}_q$, *és ismert, hogy* $\mathbb{Z}_q^* =< 3 >$. *Alice titkos kulcsa* $k'_A = 12907$, *és nyilvánossá teszi* $k_A = 3^{12907} \bmod q = 34625$-*öt. Bob kódolni akarja Alice-nek a* $P =$ **WE_GO!** *üzenetet. Tegyük fel, hogy a következő 31 betűs ábécét használja:*

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**_**	**.**	**?**	**!**	**,**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

A szöveg számokká alakítását (65537-nél kisebb számokká) az RSA-nál ismertetett módon végzi. $31^3 < 65537 < 31^4$, tehát hárombetűs tömböket kódolunk.

$$P_1 = \textbf{WE\_} = 22 \cdot 31^2 + 4 \cdot 31 + 26 = 21292$$

$$P_2 = \textbf{GO!} = 6 \cdot 31^2 + 14 \cdot 31 + 29 = 6229$$

Véletlenszerűen választ $b_1 = 11618$-ot és $b_2 = 9017$-et. Ekkor:

$$(C_1, C_1') = (3^{b_1}, P_1 \cdot 3^{k_A b_1}) = (14155, 42568) = (\textbf{AOWT}, \textbf{BNJF})$$

$$(C_2, C_2') = (3^{b_2}, P_2 \cdot 3^{k_A b_2}) = (20715, 48875) = (\textbf{AVRH}, \textbf{BT\_T})$$

Az átalakítások:

$$\begin{aligned}
14155 &= 0 \cdot 31^3 + 14 \cdot 31^2 + 22 \cdot 31 + 19 = \textbf{AOWT}\\
42568 &= 1 \cdot 31^3 + 13 \cdot 31^2 + 9 \cdot 31 + 5 \quad = \textbf{BNJF}\\
20715 &= 0 \cdot 31^3 + 21 \cdot 31^2 + 17 \cdot 31 + 7 \quad = \textbf{AVRH}\\
48875 &= 1 \cdot 31^3 + 19 \cdot 31^2 + 26 \cdot 31 + 19 = \textbf{BT_T}
\end{aligned}$$

Alice a következőképpen dekódolja az üzenetet. Megkapja a párokat:

$$(C_1, C_1') = (14155, 42568) \qquad (C_2, C_2') = (20715, 48875),$$

elvégzi a

$$P_1 = C_1' \cdot C_1^{q-1-k_A'} \bmod q = 42568 \cdot 14155^{65536-12907} \bmod 65537 = 21292 = \textbf{WE\_}$$

$$P_2 = C_2' \cdot C_2^{q-1-k_A'} \bmod q = 48875 \cdot 20715^{65536-12907} \bmod 65537 = 6229 = \textbf{GO!}$$

műveleteket, és el tudja olvasni a $P = P_1 \parallel P_2 =$ **WE_GO!** üzenetet.

3.4. A Diffie–Hellman-kulcscsere

Ahogy azt már jeleztük – figyelembe véve a nyilvános kulcsú rendszerek előnyeit és hátrányait – a legjobb, ha a tulajdonképpeni titkosítást szimmetrikus rendszerrel végezzük (pl. DES, AES), de a kulcskezelést (kulcscserét) aszimmetrikus kulcsú titkosítási rendszerrel valósítjuk meg.

A *Diffie–Hellman-kulcscsere protokoll*t kifejezetten ehhez szerkesztették 1976-ban. Ebben jelenik meg először a nyilvános kulcs fogalma is. Előnye, hogy a két kommunikáló fél megegyezhet az általuk használt szimmetrikus titkosítási rendszer közös kulcsában anélkül, hogy ezt a kulcsot valamilyen formában el kellene küldeni.

Legyen $\mathbb{F}_q$ egy nyilvános véges test (q elég nagy) és g egy nyilvános generátora $\mathbb{F}_q^*$-nak, vagyis $\mathbb{F}_q^* =< g >$. Minden X felhasználó véletlenszerűen választ egy $a_X \in \{1, \ldots, q-1\}$ titkos elemet (ez lesz a titkos kulcs), és nyilvánossá teszi $g^{a_X} \in \mathbb{F}_q^*$-ot (ez a nyilvános kulcs). Hogyan egyezik meg Alice és Bob az (E_k, D_k) szimmetrikus kulcsú rendszer k közös kulcsában?

Alice titkos kulcsa $k_A' = a$, nyilvános kulcsa $k_A = g^a$. Bob titkos kulcsa $k_B' = b$, nyilvános kulcsa $k_B = g^b$. Ekkor a közös titkos kulcs $k = g^{ab}$. Ezt Alice és Bob is ki tudja számolni (Alice $(g^b)^a$, Bob pedig $(g^a)^b$ módon), de egy harmadik fél – Marvin

– már nem, csak diszkrét logaritmálással, feltéve persze, hogy teljesül a már említett Diffie–Hellman-hipotézis (g, g^a és g^b ismerete csak diszkrét logaritmálással vezet g^{ab} ismeretéhez). Természetesen $k = g^{ab} \in \mathbb{F}_q^*$, viszont ez megfeleltethető egy q-nál kisebb természetes számnak a következő módon: ha $q = p^n$ (lásd a 3. függeléket), akkor $\mathbb{F}_q = \mathbb{Z}_p[X]/(f) = \{a_{n-1}X^{n-1} + \cdots + a_1X + a_0 \bmod f \mid a_i \in \mathbb{Z}_p\}$.

Egy $a_{n-1}X^{n-1} + \cdots + a_1X + a_0 \bmod f$ elemnek egyértelműen megfelel $a_{n-1}p^{n-1} + \cdots + a_1p + a_0 \in \mathbb{Z}$ érték, ahol $a_i \in \{0, \ldots, p-1\}$, mitöbb, $0 \leq a_{n-1}p^{n-1} + \cdots + a_1p + a_0 < p^n = q$. Az így nyert természetes számból bármely szimmetrikus rendszer titkos kulcsa valamilyen módon felépíthető.

4. fejezet

Hash függvények

4.1. Alapfogalmak

Jelölje $\mathcal{M}$ a tetszőleges hosszú bitsorozatok halmazát és $\mathcal{M}_r$ az olyan bitsorozatokét, melyeknek hossza r.

4.1.1. értelmezés. *Egy $h : \mathcal{M} \to \mathcal{M}_r$ függvényt* gyengén ütközésmentes hash függvény*nek nevezünk, ha rendelkezik az alábbi tulajdonságokkal:*

1. h nyilvános és $M \in \mathcal{M}$ ismeretében $h(M)$ könnyen (rövid, polinomiális idő alatt) kiszámolható.

2. Adott $H \in \mathcal{M}_r$-hez tartozó őskép ($M \in \mathcal{M}$ úgy, hogy $h(M) = H$) meghatározása nehéz feladat (gyakorlatilag kivitelezhetetlen számításigényű). Ezt úgy is szoktuk mondani, hogy *h egyirányú függvény*, vagy hogy *h őskép-ellenálló*.

3. Ha $M \in \mathcal{M}$ adott, nehéz egy másik $M' \in \mathcal{M}$-et találni úgy, hogy $h(M') = h(M)$. Másképpen szólva, h gyengén ütközésmentes vagy *h második őskép-ellenálló*.

4.1.2. értelmezés. *Egy hash függvényt* ütközésmentes*nek nevezünk, ha gyengén ütközésmentes, és nehéz olyan M, $M' \in \mathcal{M}$ bitsorozatokat találni, melyekre $h(M) = h(M')$ teljesül.*

A következőkben bevezetünk néhány hash függvényhez kapcsolódó fogalmat és jelölést.

Az $\mathcal{M}$ halmaz elemeit (a tetszőleges hosszú bitsorozatokat) *üzenet*eknek nevezzük, egy üzenetet általában M-mel jelölünk. $L(M)$ fogja jelölni az M üzenet bitekben számolt hosszát. $h(M) \in \mathcal{M}_r$ az M üzenet *lenyomat*a és MD-vel jelöljük (ezért a hash függvényeket még *lenyomatkészítő függvény*eknek is nevezzük). *Ütközés*ről beszélünk akkor, amikor találunk két ugyanolyan lenyomatú de különböző M és M' üzenetet (vagyis $M \neq M'$, ugyanakkor $h(M) = h(M')$).

A lenyomatok bitekben számolt hossza (a továbbiakban r-rel jelöljük) függ az adott hash függvénytől. Például az MD4 és MD5 hash függvények esetében a lenyomat hossza $r = 128$ (ez azt jelenti, hogy a lenyomat 16 bájt hosszú - illetve 32 hexadecimális karakterrel ábrázolható), a SHA-1 esetében a lenyomat hossza viszont $r = 160$ (20 bájt vagy 40 hexadecimális karakter).

Fontos megemlíteni még, hogy a gyakorlatban használt hash függvényeknek rendelkezniük kell az úgynevezett *lavina-hatás*sal. Ez azt jelenti, hogy kismértékű változás az üzenetben – akár 1 bitnyi eltérés – nagymértékű változást idéz elő a lenyomatban.

A fentiek alapján a hash függvények bizonyos értelemben biztonságos, „hűséges" és irreverzibilis módon tömörítenek össze nagy méretük miatt kezelhetetlen adatokat. Ez a tömörített, „hűséges" lenyomat sokkal könnyebben kezelhető a különböző alkalmazásokban, mint például a hitelesítés, a digitális aláírások, üzenetek épségének ellenőrzése.

4.2. Hash függvények szerkesztése

R. Merkle és I. Dåmgard egymástól függetlenül dolgoztak ki általános módszert a hash függvények szerkesztésére. A gyakorlatban használt hash függvények majdnem mindegyike a *Merkle–Dåmgard-konstrukció*t követi.

A módszer lényege a következő: a bemeneti adatot egyenlő méretű blokkokra osztjuk, majd mindegyik blokkot iteratív módon egy tömörítő függvény segítségével redukáljuk. E függvény tehát adott méretű bemenetből adott, kisebb méretű kimenetet hoz létre.

A következőkben felvázoljuk a Merkle–Dåmgard-féle konstrukciót. M jelöli a bemeneti adatot vagy üzenetet és C a tömörítő függvényt. C-nek két argumentuma van: $H \in \mathcal{M}_r$ és $B \in \mathcal{M}_t$, ahol $t \geq r$. A C függvény kimenetének mérete szintén r bit. Az M üzenet hosszát egy s bites változóban tároljuk és $L(M)_s$-el jelöljük. Általában $s = 64$, tehát az üzenet hossza kisebb kell legyen, mint 2^{64} bit.

A Merkle–Dåmgard-konstrukció lépései:

1. Az M üzenetet megtoldjuk egy 1 értékű bittel, majd egy sorozat 0 értékű bittel és végül $L(M)_s$-el úgy, hogy a kiegészített üzenet teljes hossza t-nek legkisebb többszöröse legyen. Jelöljük $\tilde{M}$-mel a kiegészített üzenetet. Ennek alakja tehát $\tilde{M} = (M \parallel 1 \parallel 0 \ldots 0 \parallel L(M)_s)$, ahol $\parallel$ bitsorozatok összeillesztését (konkatenálását) jelöli.

2. $\tilde{M}$-et felosztjuk t bites blokkokra, legyenek ezek: $B_1, \ldots, B_n$.

3. Választunk egy r hosszúságú IV kezdőértéket, és legyen $H_0 := IV$.

4. Minden $i = 1, \ldots, n$-re, legyen $H_i := C(H_{i-1}, B_i)$.

5. Legyen $h(M) = H_n$.

A Merkle–Dåmgard-séma tehát:

$$\begin{tikzcd}
 & B_1 \arrow[d] & B_2 \arrow[d] & \cdots & B_n \arrow[d] & \\
IV \arrow[r] & C \arrow[r] & C \arrow[r] & \cdots \arrow[r] & C \arrow[r] & h(M)
\end{tikzcd}$$

Merkle és Dåmgard tétele biztosítja a konstrukció helyességét:

4.2.1. tétel. *Ha a C tömörítő függvény ütközésmentes, akkor a h hash függvény is ütközésmentes.*

Bizonyítás. Feltételezzük, hogy szerkesztettünk egy $h(M) = h(M')$ ütközést. Az üzenetek kiegészítése és blokkokra való felosztása után kapjuk, hogy $\tilde{M} = B_1 \dots B_n$ és $\tilde{M}' = B'_1 \dots B'_{n'}$. A Merkle–Dåmgard-módszernek megfelelően megszerkesztjük a H_i és H'_jértékeket, és azt kapjuk, hogy

$$C(H_{n-1}, B_n) = H_n = h(M) = h(M') = H'_{n'} = C(H'_{n'-1}, B'_{n'}).$$

Ha ez nem egy ütközés a C függvény esetében, akkor a bemenetek egyenlőek kell hogy legyenek, tehát $B_n = B_{n'}$. De ez akkor azt jelenti, hogy M-nek és M'-nek ugyanaz a hossza (mert egyenlőek az üzenet hosszát tartalmazó utolsó blokkok). Tehát $n = n'$ és akkor

$$C(H_{n-2}, B_{n-1}) = H_{n-1} = H'_{n-1} = C(H'_{n-2}, B'_{n-1}).$$

Mivel $M \neq M'$, az előbbi gondolatmenetet alkalmazva kapunk két olyan blokkot, hogy $B_i \neq B'_i$, vagyis egy ütközést a C függvény esetében. □

A tömörítő függvény lehet olyan, melyet speciálisan lenyomatkészítésre fejlesztettek ki, de sok esetben lehet egy tömbtitkosító (blokkrejtjelező) is. Ilyen például a *Davies–Meyer-séma*: legyen E egy blokkrejtjelező, $E_k : \mathcal{M}_r \to \mathcal{M}_r$ egy titkosító függvény, $E_k(P)$ pedig a k kulccsal kódolt üzenet. Jelöljük D_k-val a megfelelő dekódoló függvényt (tehát $D_k = E_k^{-1}$). Az előbbi jelölést használva, a C tömörítő függvényt a következő módon adjuk meg:

$$C(H_{i-1}, B_i) := E_{B_i}(H_{i-1}) + H_{i-1},$$

ahol „+” bármely csoportművelet $\mathcal{M}_r$-ben (általában XOR). A csoport semleges elemét 0^r-el jelöljük.

4.3. Két híres hash függvény: az MD5 és az SHA-1

Az MD5 (Message Digest 5) egyike a legelterjedtebb hash függvényeknek. Széles körben használják internetes alkalmazásokban, és be van építve számos programozási nyelvbe is (például PHP). Az MD5 hash függvényt az MIT-nél (Massachusetts Institute of Technology) dolgozó Ronald Rivest fejlesztette ki 1991-ben. 1992-ben szabványosították internetes alkalmazásokban való használatra (RFC 1321). Az MD5 biztonságos helyettesítője volt egy korábbi algoritmusnak, az MD4-nek. Napjainkban azonban – mint később látni fogjuk – az MD5 már nem nyújt kielégítő biztonságot.

Az MD5 a Merkle–Dåmgard-konstrukciót követi egy $128 \times 512 \to 128$ típusú tömörítő függvényt használva. Az előző alfejezet jelöléseit használva tehát $r = 128$, $t = 512$ és az üzenet hosszát $s = 64$ biten tároljuk. Az algoritmus leírásánál használni fogjuk a következő elemeket:

- f_i egy bitenkénti Boole-függvény ($\oplus$ jelöli a XOR, $\vee$ az OR, $\wedge$ az AND és $\neg$ a NOT bitműveletet):

$$f_i(X, Y, Z) = \begin{cases} (X \wedge Y) \vee (\neg(X) \wedge Z), & \text{ha } i = 1, ..., 16 \\ (X \wedge Y) \vee (\neg(Z) \wedge Y), & \text{ha } i = 17, ..., 32 \\ X \oplus Y \oplus Z, & \text{ha } i = 33, ..., 48 \\ (\neg(Z) \vee X) \oplus Y, & \text{ha } i = 49, ..., 64 \end{cases}$$

- $t_i = [2^{32}|sin(i)|]$, ahol $i = \overline{1, 64}$ radiánban van megadva

- $r_i = \begin{cases} i - 1, & \text{ha } i = 1, ..., 16 \\ (1 + 5(i - 1)) \bmod 16, & \text{ha } i = 17, ..., 32 \\ (5 + 3(i - 1)) \bmod 16, & \text{ha } i = 33, ..., 48 \\ 7(i - 1) \bmod 16, & \text{ha } i = 49, ..., 64 \end{cases}$

- $$s_i = \begin{cases} 4, & \text{ha } i = 33, 37, 41, 45 \\ 5, & \text{ha } i = 17, 21, 25, 29 \\ 6, & \text{ha } i = 49, 53, 57, 61 \\ 7, & \text{ha } i = 1, 5, 9, 13 \\ 9, & \text{ha } i = 18, 22, 26, 30 \\ 10, & \text{ha } i = 50, 54, 58, 62 \\ 11, & \text{ha } i = 34, 38, 42, 46 \\ 12, & \text{ha } i = 2, 6, 10, 14 \\ 14, & \text{ha } i = 19, 23, 27, 31 \\ 15, & \text{ha } i = 51, 55, 59, 63 \\ 16, & \text{ha } i = 35, 39, 43, 47 \\ 17, & \text{ha } i = 3, 7, 11, 15 \\ 20, & \text{ha } i = 20, 24, 28, 32 \\ 21, & \text{ha } i = 52, 56, 60, 64 \\ 22, & \text{ha } i = 4, 8, 12, 16 \\ 23, & \text{ha } i = 36, 40, 44, 48 \end{cases}$$

- $add(X, Y, ...) = (X + Y + ...) \bmod 2^{32}$

- $rol(X, s) =$ X bitjeit s pozícióval ciklikusan balra toljuk

A Merkle–Dåmgard-konstrukciónak megfelelően az MD5 algoritmus lépései a következők:

1. Az M üzenetet megtoldjuk egy 1 értékű bittel, egy sorozat 0 értékű bittel és végül $L(M)_{64}$-el úgy, hogy a kiegészített üzenet teljes hossza 512-nek legkisebb többszöröse legyen. Jelöljük $\tilde{M}$-mel a kiegészített üzenetet. Ennek alakja tehát $\tilde{M} = (M \parallel 1 \parallel 0 \ldots 0 \parallel L(M)_{64}$, ahol $\parallel$ bitsorozatok összeillesztését jelöli.

2. $\tilde{M}$-et felosztjuk 512 bites blokkokra, legyenek ezek: $B_1, \ldots, B_n$. Továbbá, minden blokkot felosztunk 16 szóra (egy szó 32 bitet tartalmaz). Jelöljük $B_{i,j}$ az i-edik blokk j-edik szavát, ahol $j \in \{1, \ldots, 16\}$ és $i \in \{1, \ldots, n\}$.

3. Választunk egy négy szóból álló (128 bites) $IV = (X_1, X_2, X_3, X_4)$ kezdeti értéket, ahol a négy szó a következő értékeket kapja (hexadecimális számrendszerben):

$$X_1 = 67452301,\ X_2 = efcdab89,\ X_3 = 98badcfe,\ X_4 = 10325476.$$

4. Végrehajtjuk a következőket:

$A := X_1,\ B := X_2,\ C := X_3,\ D := X_4$
for $i := 1, \ldots, n$ **do**
  for $j := 1, \ldots, 64$ **do**
    $T := add(A,\ f_j(B, C, D), B_{i,1+r_j}, t_j)$
    $A := D$
    $D := C$
    $C := B$
    $B := add(B, rol(T, s_j))$
  end for
  $X_1 := add(X_1, A)$
  $X_2 := add(X_2, B)$
  $X_3 := add(X_3, C)$
  $X_4 := add(X_4, D)$
end for

5. Legyen $h(M) = (X_1, X_2, X_3, X_4)$.

4.3.1. példa. *A következő, MD5 hash függvény által készített lenyomatoknál jól megfigyelhető a lavina-effektus:*

- MD5( )=7215ee9c7d9dc229d2921a40e899ec5f.
- MD5(A)=7fc56270e7a70fa81a5935b72eacbe29
- MD5(a)=0cc175b9c0f1b6a831c399e269772661
- MD5(Hello)=8b1a9953c4611296a827abf8c47804d7
- MD5(Hello!)=952d2c56d0485958336747bcdd98590d
- MD5(Message Digest 5 is one of the most popular hash functions.)=
  2ccf2ef3b66683e2d97a21a42da5b8e4
- MD5(Message Digest 5 is one of the most popular hash functions)=
  a39981f6e0d67364dc194b0577a71c1f

Az SHA-1 (Secure Hash Algorithm 1) egy másik széles körben használt hash függvény, a NIST (National Institute of Standards and Technology) szabványa 1995-től. Az MD5-re épül és főleg digitális aláírásra használják.

Az SHA-1 szintén a Merkle–Dåmgard-konstrukciót követi, egy $160 \times 512 \to 160$ típusú tömörítő függvényt használva. Az előző alfejezet szerint tehát $r = 160$, $t = 512$ és $s = 64$. Az algoritmus leírásánál használni fogjuk a következő elemeket:

- f_i egy bitenkénti Boole-függvény, a bemeneti és kimeneti értékek bitek ($\oplus$ jelöli az XOR, $\vee$ az OR, $\wedge$ az AND és $\neg$ a NOT bitműveletet):

$$f_i(X,Y,Z) = \begin{cases} (X \wedge Y) \vee (\neg(X) \wedge Z), & \text{ha } i = 1, ..., 20 \\ X \oplus Y \oplus Z, & \text{ha } i = 21, ..., 40 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & \text{ha } i = 41, ..., 60 \\ X \oplus Y \oplus Z, & \text{ha } i = 61, ..., 80 \end{cases}$$

- $t_i = \begin{cases} 5a827999, & \text{ha } i = 1, ..., 20 \\ 6ed9eba1, & \text{ha } i = 21, ..., 40 \\ 8f1bbcdc, & \text{ha } i = 41, ..., 60 \\ ca62c1d6, & \text{ha } i = 61, ..., 80 \end{cases}$

- $add(X, Y, ...) = (X + Y + ...) \bmod 2^{32}$

- $rol(X, s) = X$ bitjeit s pozicióval ciklikusan balra toljuk

A Merkle–Dåmgard-konstrukciónak megfelelően az SHA-1 algoritmus lépései a következők:

1. Az M üzenetet megtoldjuk egy 1 értékű bittel, egy sorozat 0 értékű bittel és végül $L(M)_{64}$-el úgy, hogy a kiegészített üzenet teljes hossza 512-nek legkisebb többszöröse legyen. Jelöljük $\tilde{M}$-mel a kiegészített üzenetet. Ennek alakja tehát $\tilde{M} = (M \,||\, 1 \,||\, 0 \dots 0 \,||\, L(M)_{64})$, ahol $||$ bitsorozatok összeillesztését jelöli.

2. $\tilde{M}$-et felosztjuk 512 bites blokkokra, legyenek ezek: $B_1, \dots, B_n$. Továbbá, minden blokkot felosztunk 16 szóra (egy szó 32 bitet tartalmaz). Jelöljük $B_{i,j}$ az i-edik blokk j-edik szavát, ahol $j \in \{1, \dots, 16\}$ és $i \in \{1, \dots, n\}$. Induktíve értelmezzük:

$$B_j(i) = \begin{cases} B_{i,j}, & \text{ha } j \in \{1, \dots, 16\} \\ rol(B_{j-3}(i) \oplus B_{j-8}(i) \oplus B_{j-14}(i) \oplus B_{j-16}(i), 1), & \text{ha } j \in \{17, \dots, 80\} \end{cases}$$

3. Választunk egy öt szóból álló (160 bites) $IV = (X_1, X_2, X_3, X_4, X_5)$ kezdeti értéket, ahol az öt szó a következő értékeket kapja (hexadecimális számrendszerben):

$$X_1 = 67452301, \quad X_2 = efcdab89, \quad X_3 = 98badcfe,$$
$$X_4 = 10325476, \quad X_5 = c3d2e1f0.$$

4. Végrehatjuk a következőket:
   $A := X_1$, $B := X_2$, $C := X_3$, $D := X_4$, $E := X_5$
   for $i := 1, \ldots, n$ **do**
   for $j := 1, \ldots, 80$ **do**
   $T := add(rol(A, 5), f_j(B, C, D), E, B_j(i), t_j)$
   $E := D$
   $D := C$
   $C := rol(B, 30)$
   $B := A$
   $A := T$
   end for
   $X_1 := add(X_1, A)$
   $X_2 := add(X_2, B)$
   $X_3 := add(X_3, C)$
   $X_4 := add(X_4, D)$
   $X_5 := add(X_5, E)$
   end for
5. Legyen $h(M) = (X_1, X_2, X_3, X_4, X_5)$.

4.3.2. példa. *A következő, SHA-1 hash függvény által készített lenyomatoknál is jól megfigyelhető a lavina-effektus:*

- SHA-1( )=b858cb282617fb0956d960215c8e84d1ccf909c6
- SHA-1(A)=6dcd4ce23d88e2ee9568ba546c007c63d9131c1b
- SHA-1(a)=86f7e437faa5a7fce15d1ddcb9eaeaea377667b8
- SHA-1(Hello)=f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0
- SHA-1(Hello!)=69342c5c39e5ae5f0077aecc32c0f81811fb8193
- SHA-1(Secure Hash Algorithm 1 is one of the most popular hash functions.)= e72d8eb5282c3717319972db275946244f63ded8
- SHA-1(Secure Hash Algorithm 1 is one of the most popular hash functions)= 49283b6dc3bd9b60b51bb94b640a4ba7b9781254

4.4. A kimerítő kulcskeresés. A születésnap-paradoxon

Általában háromféle támadás indítható egy hash függvény ellen:

1. támadás a hash függvény egyirányúsága ellen (*őskép-támadás*)

2. a hash függvény gyengén ütközésmentességére irányuló támadás (*második ősképtámadás*)

3. a hash függvény ütközésmentességére irányuló támadás (*ütközéses támadás*)

A legelterjedtebb hash függvények többé-kevésbé sérülékenyek az ütközéses támadással szemben, viszont elég jól ellenállnak az őskép- és a második őskép-támadásnak. Megjegyezzük, hogy kismértékű sérülékenység az ütközéses támadással szemben alig befolyásolja egy hash függvény gyarkorlati alkalmazhatóságát. Mint látni fogjuk azonban, az MD5 napjainkra már komolyan sebezhetővé vált az ütközéses támadással szemben, olyannyira, hogy már nem használható biztonságosan rendeltetésének megfelelően.

A hash függvények elleni támadások lehetnek átalánosak (olyanok, amelyek minden hash függvény ellen, vagy legalábbis a hash függvénycsaládok ellen hatékonyak) vagy speciálisan egyetlen hash függvényre kifejlesztettek.

Fontos általános támadásnak számít a kimerítő kulcseresés. A kimerítő kulcskeresés szempontjából a hash függvények fontos jellemzői a lenyomat bithossza és az algoritmus komplexitása vagy futási ideje. Ha egy bizonyos támadáshoz N hash művelet szükséges, akkor azt mondjuk, hogy a támadás $\mathcal{O}(N)$ idő alatt fut, vagy hogy a támadás komplexitása $\mathcal{O}(N)$.

Ha a $h : \mathcal{M} \to \mathcal{M}_r$ hash függvény lenyomatának hossza r, akkor a kimerítő őskép-támadásnak 2^r hash műveletre van szüksége (2^r különböző szövegről kell lenyomatot készíteni). Valóban, ha adott $y \in \mathcal{M}_r$ és egymás után, véletlenszerűen választunk $x \in \mathcal{M}$ üzeneteket, amíg $h(x) = y$, akkor annak a valószínűsége, hogy sikertelen próbálkozások után az i-dik próbálkozás sikerrel jár $(1-2^{-r})^{i-1}2^{-r}$. Tehát a szükséges próbálkozások (hash műveletek) száma

$$\sum_{i=1}^{\infty} i(1-2^{-r})^{i-1}2^{-r} = 2^r.$$

Ugyanilyen módon bizonyíthatjuk azt is, hogy a kimerítő második őskép-támadáshoz szükséges hash műveletek száma 2^r, ha a hash függvény lenyomatának hossza r.

Az úgynevezett születésnap-paradoxon miatt a kimerítő ütközéses támadásokhoz kevesebb hash műveletre van szükség (ezért ezeket még születésnap-támadásoknak is nevezzük). A születésnap-paradoxon a következő megállapításon alapszik: valamivel több, mint 50% (tehát meglepően nagy) az esélye annak, hogy 23 személy közül legalább kettőnek ugyanarra a napra esik a születésnapja. Legalább 60 ember esetében, ugyanennek a valószínűsége több, mint 99%. A bizonyítás egyszerű. Jelöljük $P(n)$-el annak a valószínűségét, hogy egy n tagú csoportból legalább két személynek ugyanarra a napra esik a születésnapja. Akkor annak a valószínűsége, hogy mind az n személynek különböző napokra essen a születésnapja

$$1 - P(n) = \frac{364}{365} \times \frac{363}{365} \times \cdots \times \frac{365-n+1}{365}.$$

Következik tehát, hogy például $P(23) = 0.5$ és $P(50) = 0.96$. Tehát egy 50-60 fős csoportban majdnem biztosan találunk két személyt, akik ugyanazon a napon ünneplik születésnapjukat.

A születésnap-paradoxon pontos és általános megfogalmazását a következő tétel mutatja be:

4.4.1. tétel. *Ha függetlenül és véletlenszerűen (egyenletes eloszlással) választunk $\theta\sqrt{N}$ darab számot az $\{1, 2, ..., N\}$ halmazból, akkor annak a valószínűsége, hogy egy számot kétszer válasszunk*

$$P(\theta, N) = 1 - \frac{N!}{N^{\theta\sqrt{N}}(N - \theta\sqrt{N})!},$$

ami az $1 - e^{-\frac{\theta^2}{2}}$ értékhez tart, amikor $N \to \infty$.

Ezért, amikor $N = 365$ és $n = \theta\sqrt{N}$ kapjuk, hogy $P(n) = P(\frac{n}{\sqrt{365}}, 365) \approx 1 - e^{-\frac{n^2}{730}}$, tehát, mint előbb láttuk, $P(23) = 0,5$ és $P(50) = 0,96$.

Tekintsünk most egy $h : \mathcal{M} \to \mathcal{M}_r$ hash függvényt. A születésnap-támadás abban áll, hogy véletlenszerű üzenetek lenyomatait számoljuk, mindaddig, amíg találunk két egyforma lenyomatú üzenetet. Az előbbi tételt felhasználva (most $N = 2^r$) kapjuk, hogy $\theta 2^{\frac{r}{2}}$ hash műveletet kell végeznünk ahhoz, hogy $1 - e^{-\frac{\theta^2}{2}}$ legyen az ütközés valószínűsége.

Következtetésképpen láthatjuk, hogy ha adott egy r bites lenyomatot generáló hash függvény, a sikeres kimerítő őskép- és második őskép-támadás komplexitása $\mathcal{O}(2^r)$, a kimerítő ütközéses támadás esetében pedig $\mathcal{O}(2^{\frac{r}{2}})$. A hash függvények tervezésekor a cél az, hogy ne lehessen a fentieknél hatékonyabb támadásokat alkalmazni.

4.5. Joux-féle többszörös ütközéses támadás

A Merkle–Dåmgard-konstrukciónak több gyenge pontja is van. Az egyik az, hogy többszörös ütközést (amikor több üzenetnek ugyanaz a lenyomata) generálni nem sokkal költségesebb, mint egyszerű ütközést találni. A következőkben bemutatott általános támadás ezt a sérülékenységet használja ki.

A CRYPTO'04 konferencián A. Joux bemutatott egy módszert, amellyel $\mathcal{O}(k2^{\frac{r}{2}})$ idő alatt lehet 2^k-szoros ütközést generálni a $h : \mathcal{M} \to \mathcal{M}_r$ Merkle–Dåmgard-féle hash függvények esetében [8]. A támadásnak a következő lépései vannak:

- Minden $i = 1, ..., k$-ra kell találni egy $B_i^0 \neq B_i^1$ lokális ütközést úgy, hogy $H_i = C(H_{i-1}, B_i^0) = C(H_{i-1}, B_i^1)$.

- Az összes 2^k darab $(B_1^{b_1}, ..., B_k^{b_k})$ alakú üzenetnek – ahol $b_i \in \{0, 1\}$ – ugyanaz a H_k lenyomata lesz.

Megjegyezzük, hogy minden üzenet kt bit hosszú (t a C tömörítő függvény második argumentumának a hossza).

Ahogy Joux kimutatta, ez a módszer mindig alapját képezheti az iterált hash függvények elleni támadásnak. Legyen $h : \mathcal{M} \to \mathcal{M}_r$, $h(M) = h(h_1(M) \parallel h_2(M))$, ahol h_1 és h_2 két egymástól független r bites lenyomatot készítő hash függvény. Optimális esetben egy ütközés generálása h esetében $\mathcal{O}(2^r)$ időt igényel. Ha azonban a h_1 vagy h_2 bármelyike a Merkle–Dåmgard-sémára épül, akkor ütközést lehet találni a h-ra már $\mathcal{O}((\frac{r}{2})2^{\frac{r}{2}})$ időben is. Feltéve, hogy h_1 a Merkle–Dåmgard-sémára épülő hash függvény, a módszer lépései a következők:

- $2^{\frac{r}{2}}$ darab ütközést keresünk h_1-re, $(\frac{r}{2})2^{\frac{r}{2}}$ idő alatt.
- Statisztikailag, egy ilyen ütközés h_2-ben is érvényes, tehát ütközés lesz h-ban is.

Joux azt is bebizonyította, hogy a többszörös ütközéses támadás módszerét fel lehet használni (második) ősképek hatékony keresésére is. Adott $y \in \mathcal{M}_r$ lenyomatra a támadás lépései:

- Generálunk 2^k ütközést (legyenek ezek $M^1, ..., M^{2^k}$) úgy, hogy $H_k = H(M^1) = ... = H(M^{2^k})$.
- Keresünk egy M_{k+1} üzenettöredéket úgy, hogy $C(H_k, M_{k+1}) = y$.

Így kaptunk 2^k ősképet. Az első lépés $\mathcal{O}(k2^{\frac{r}{2}})$ időt vesz igénybe, ami elhanyagolható a második lépés komplexitása mellett, ami nagyjából egyenlő egy kimerítő őskép-támadás komplexitásával, tehát $\mathcal{O}(2^r)$.

Ha a C tömörítő függvény a Merkle–Dåmgard-konstrukcióból a Davies–Meyer-sémát követi, akkor (véletlenszerű) fixpontokat számolhatunk ki C-re. Valóban, ha választunk egy B_i üzenetet, és kiszámoljuk $H_{i-1} := D_{B_i}(0^r)$-t, akkor ez egy fixpont, mert

$$H_i = C(H_{i-1}, B_i) = E_{B_i}(H_{i-1}) + H_{i-1} = 0^n + H_{i-1}.$$

A $H_{i-1} = H_i$ fixpont függ a B_i megválasztásától, de bármely B_i-re létezik ilyen fixpont.

4.6. Az MD5 és az SHA-1 kriptoanalízise

2004-ig az MD5 ütközésmentes hash függvényként volt számontartva. A helyzet azonban megváltozott 2004 augusztusában, amikor Xiaoyun Wang, Dengguo Feng, Xuejia Lai és Hongbo Yu bejelentették, hogy ütköző üzeneteket generáltak az MD5-re. A következő ütközést adták meg:

- M egy 1024 bites üzenet, a következő hexadecimális ábrázolással:
  d1 31 dd 02 c5 e6 ee c4 69 3d 9a 06 98 af f9 5c 2f ca b5 87 12 46 7e ab 40 04
  58 3e b8 fb 7f 89 55 ad 34 06 09 f4 b3 02 83 e4 88 83 25 71 41 5a 08 51 25 e8
  f7 cd c9 9f d9 1d bd f2 80 37 3c 5b d8 82 3e 31 56 34 8f 5b ae 6d ac d4 36 c9
  19 c6 dd 53 e2 b4 87 da 03 fd 02 39 63 06 d2 48 cd a0 e9 9f 33 42 0f 57 7e e8
  ce 54 b6 70 80 a8 0d 1e c6 98 21 bc b6 a8 83 93 96 f9 65 2b 6f f7 2a 70

- M' is egy 1024 bites üzenet, a következő hexadecimális ábrázolással:
  d1 31 dd 02 c5 e6 ee c4 69 3d 9a 06 98 af f9 5c 2f ca b5 07 12 46 7e ab 40 04
  58 3e b8 fb 7f 89 55 ad 34 06 09 f4 b3 02 83 e4 88 83 25 f1 41 5a 08 51 25 e8
  f7 cd c9 9f d9 1d bd 72 80 37 3c 5b d8 82 3e 31 56 34 8f 5b ae 6d ac d4 36 c9
  19 c6 dd 53 e2 34 87 da 03 fd 02 39 63 06 d2 48 cd a0 e9 9f 33 42 0f 57 7e e8
  ce 54 b6 70 80 28 0d 1e c6 98 21 bc b6 a8 83 93 96 f9 65 ab 6f f7 2a 70

- Ekkor MD5(M)=MD5(M')=79054025255fb1a26e4bc422aef54eb4.

Aláhúztuk azokat a karaktereket, amelyek különböznek M-nél és M'-nél.

Az első ütközést aztán egyre több követte. Újabb és egyre hatékonyabb módszereket fejlesztettek ki. 2006-ban V. Klima közölt egy algoritmust, amivel körülbelül egy perc alatt lehet találni egy ütközést (egy asztali szamítógépen).

Ezek az eredmények érdekesek voltak a kriptográfusok számára, eleinte azonban alábecsülték gyarkorlati hasznukat (vagy veszélyességüket). Főleg azért, mert annak ellenére, hogy hatékonyan lehetett ütköző M és M'üzenetet találni, ezek az üzenetek többé-kevésbé véletlenszerűek és adott hosszúságúak voltak (lásd az előbbi példát).

Az MD5 esetében azonban ez az érvelés hibásnak bizonyult. M. Daum és S. Lucks elő tudtak állítani két értelmes szöveget tartalmazó PostScript fájlt, ugyanazzal az MD5 lenyomattal, éspedig

$$a25f7f0b29ee0b3968c860738533a4b9$$

A PostScript fájlok tartalma a következő oldalakon található (4.1 és 4.2 ábra). A szerzők egy érdekes történettel illusztrálták a támadást.

Alice és főnökének története

Alice már egy jó ideje Rómában dolgozott, Julius Caesar irodai alkalmazottjaként. Egy napon viszont bejelenti a főnökének, hogy állást szeretne változtatni.

Caesar szemszögéből.

Azon a napon, amikor Alice bejelenti felmondását, Caesar egy ajánlólevelet ír számára, papírra. Alice megkéri Caesart, hogy írja alá digitálisan a levelet. Hogy megkönnyítse főnöke dolgát, Alice elő is készíti az eredeti ajánlólevél digitális változatát, egy PostScript fájlt. Caesar megnyitja és leellenőrzi az állományt: pontosan az áll benne, mint az eredeti változatban. Elkészíti a PostScript állomány MD5 lenyomatát, és ezt használva digitálisan aláírja a dokumentumot.

4.1. ábra. Ceasar ajánlólevele

Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

To Whom it May Concern:

Alice Falbala fulfilled all the requirements of the Roman Empire intern position. She was excellent at translating roman into her gaul native language, learned very rapidly, and worked with considerable independence and confidence.

Her basic work habits such as punctuality, interpersonal deportment, communication skills, and completing assigned and self-determined goals were all excellent.

I recommend Alice for challenging positions in which creativity, reliability, and language skills are required.

I highly recommend hiring her. If you'd like to discuss her attributes in more detail, please don't hesitate to contact me.

Sincerely,

Julius Caesar

4.2. ábra. Caesar parancsa

Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

Order:

Alice Falbala is given full access to all confidential and secret information about GAUL.

Sincerely,

Julius Caesar

Hónapokkal később Caesar észreveszi, hogy a gall háborús ügyekkel kapcsolatos, szigorúan titkos dokumentumokhoz fértek hozzá ismeretlen tettesek. Sehogy sem tudja elképzelni, hogyan történhetett mindez. Rájön-e, ki szedte rá és hogyan?

Alice szemszögéből.

Egyszerű irodai alkalmazottként Alice-nek nincs hozzáférése a szigorúan titkos dokumentumokhoz, ezért elhatározza, hogy becsapja Caesart. Mivel Caesar még mindig az MD5 hash függvény használja, Alice támadásának alapja Wang és Yu cikke (amely megad egy módszert, amivel ütközést lehet generálni az MD5 hash függvény esetében). Amikor megkapja Caesar (papírra írt) ajánlólevelét, előkészít két PostScript állományt: az egyikben ugyanaz áll, mint az ajánlólevélben, a másik pedig egy parancs, amiben Caesar arra utasítja a biztonságért felelős alkalmazottat, hogy Alice-nek biztosítson szabad hozzáférést bizonyos dokumentumokhoz. A két állomány tartalma különböző, viszont MD5 lenyomatuk megegyezik. Ezután Alice megkéri főnökét, hogy digitális aláírásával hitelesítse az ajánlólevelet. Caesar, miután ellenőrzi a levél tartalmát, gyanútlanul megteszi ezt. Mivel az ajánlólevél és az írott parancs lenyomata megegyezik, az ajánlólevelet hitelesítő digitális aláírás a parancs esetében is érvényes lesz. Alice a parancsot megtoldja Caesar aláírásával, és az így hitelesített parancs segítségével hozzáfér a szigorúan titkos iratokhoz.

Számos próbálkozál ellenére, sem sikerült még ütközést generálni a SHA-1 esetében. Ezért ez a hash függvény még napjainkban is ütközésmentesnek tekinthető. X. Wang, Y. L. Yin és H. Yu 2005-ben találtak egy módszert amivel $O(2^{69})$ idő alatt lehet ütközést találni, körülbelül kétezerszer gyorsabban, mint a születésnap-támadással (ami $O(2^{80})$ idő alatt talál ütközést). Ezeknek az eredményeknek még nincs túl nagy gyakorlati jelentőségük és nem veszélyeztetik közvetlenül a SHA-1 biztonságát. Becslések szerint, 2^{69} nagyságrendű számítást 56 óra alatt lehetne elvégezni egy olyan szuperszámítógép segítségével, aminek az ára 25-38 millió dollár között van, vagy 4,81 év alatt, ha a számításokat megosztva végezné 2,7 millió asztali számítógép. 2004-ben Schneier és Kelsey bejelentettek egy második őskép-támadást a SHA-1 ellen, ami $O(2^{106})$ idő alatt szolgáltat eredményt, sokkal gyorsabban, mint a $O(2^{160})$-os kimerítő kulcskeresés. Tehát van kilátás a SHA-1 feltörésére. Az amerikai NIST ezért pályázatot írt ki annak érdekében, hogy megtalálják a SHA-1 utódját, a SHA-2-t, ami 2010-ben fog a SHA-1 helyére lépni.

4.7. Alkalmazások

Felsoroljuk a hash függvények néhány fontos gyakorlati alkalmazását. Az egyik legfontosabbnak – a digitális aláírásnak – külön fejezetet szentelünk.

Üzenetek sértetlenségének ellenőrzése

Az üzenetek küldés közben sérülhetnek. Az üzenetek sérülésének több oka van: egy harmadik fél – a támadó – akarattal módosítja őket, vagy a továbbítási csatornán zajok lépnek fel, amik interferálnak az üzenettel (pl. a rádión küldött üzenetek esetében). Egy h hash függvényt hatékonyan fel lehet használni az üzenetek sértetlenségének ellenőrzésére. A felhasználási mód függ a sérülés várt okától.

Ha fennáll az üzenet szándékos (rosszindulatú) módosításának veszélye, akkor a következő módon járhatunk el:

- Az M üzenetet a nyilvános (nem biztonságos) csatornán, a $H = h(M)$ lenyomatot pedig egy biztonságos csatornán továbbítjuk (ebben az esetben H az üzenethitelesítő-kód, vagy angolul Message Authentication Code, MAC).
- Feltételezzük, hogy a címzett megkapja a (valószínűleg módosított) M' üzenetet és a H lenyomatot.
- A címzett leellenőrzi a $H = h(M')$ egyenlőséget. Ha az egyenlőség fennáll, akkor majdnem bizonyosan $M = M'$, tehát a kapott üzenet nem volt módosítva. Ha $H \neq h(M')$, akkor az üzenetet módosította valaki.

Ha csak a csatornán fellépő zajok miatt sérülhet az üzenet (adatátviteli hiba miatt), akkor nincs szükségünk biztonságos csatornára:

- Az M üzenetet és a $H = h(M)$ lenyomatot ugyanazon a csatornán küldjük.
- Feltételezzük, hogy a címzett megkapja a (valószínűleg sérült) M' üzenetet és a H lenyomatot.
- A címzett leellenőrzi a $H = h(M')$ egyenlőséget. Ha az egyenlőség fennáll, akkor majdnem bizonyosan $M = M'$, tehát a kapott üzenet nem sérült. Ha $H \neq h(M')$, akkor az üzenet adatátviteli hiba miatt sérült.

Jelszavas azonosítás

Egy számítógépes rendszer a felhasználókat jelszó alapján azonosítja. A jelszavak természetesen nem tárolhatók sima szöveges állományban, biztonsági okok miatt. Az egyik legkézenfekvőbb megoldás egy h hash függvény használata: a jelszavak helyett ezeknek a h függvénnyel számolt lenyomatát tárolják. A rendszerbe való belépéskor a felhasználó által beírt jelszó lenyomata kerül összehasonlításra a tárolt lenyomattal. A hash függvények használatából egy másik előny is származik: mivel csak a lenyomatok vannak tárolva, a jelszavakhoz még a rendszergazda sem tud hozzáférni, ami azért jó, mert egy felhasználó ugyanazt a jelszót több helyen használhatja (pl. a személyes e-mailjénél).

Kötelezettségvállalási sémák

A következő két példán keresztül szemléltetjük a kötelezettségvállalási séma fogalmát.

Az egyik forgatóköny szerint Alice és Bob azért versenyeznek, hogy melyikük tud hamarabb bizonyítani egy nehéz matematikai sejtést. Alice bebizonyít egy fontos tételt, de a sejtés bizonyítása még nem teljes. Alice szeretné, ha mindenki tudná, ő volt az első, aki a tételt bizonyította, de ugyanakkor nem akarja publikálni eredényét, mert ezzel Bob munkáját segítené. Ezért Alice egy h hash függvény segítségével kiszámolja M tudományos cikkének $H = h(M)$ lenyomatát. Ezután e-mailben értesíti a szakértőket és Bobot arról, hogy áttörést ért el a sejtés bizonyításában (nem közöl részleteket). Csatolja a H lenyomatot is, bizonyítva a részeredményt tartalmazó tudományos cikk létezését. Feltételezzük most, hogy bizonyos idő után Bobnak szintén sikerül bebizonyítania a tételt (Alice-től függetlenül) és szaklapban megjelenteti eredményét. Ekkor Alice újra küld egy e-mailt a szakértőknek, csatolva most az M cikket, bizonyítva, hogy $h(M) = H$ (ahol H a már elküldött lenyomat), tehát ő volt az első, aki megkapta az eredményt.

Egy másik forgatókönyv szerint Alice és Bob telefonon beszélgetve egymással elhatározzák, hogy együtt mennek autózni. A baj viszont az, hogy egyikük sem szeret vezetni, ezért úgy döntenek, hogy „telefonos pénzfeldobással” döntik el, ki lesz a sofőr. Mivel nem bíznak egymásban, a következő módon járnak el:

- Választanak egy h hash függvényt.
- Alice kiválaszt találomra egy M természetes számot, kiszámolja a $H = h(M)$ lenyomatát és ezt megmondja Bobnak. Ezután arra kéri, hogy próbálja meg kitalálni azt, hogy az M páros volt-e vagy páratlan: ha Bob eltalálja azt, hogy M páros-e vagy páratlan, akkor Alice vezeti az autót, ha nem akkor Bob.
- Bob leírja a H lenyomatot és tippel (50% esélye van arra, hogy eltalálja azt, hogy M páros-e vagy páratlan). Megmondja Alice-nek telefonon, hogy mire tippelt.
- Alice megmondja telefonon, hogy kettőjük közül ki vezeti majd az autót.
- Ha Alice azt mondja, hogy Bob lesz a sofőr és Bob meg akar bizonyosodni arról, hogy Alice nem csalt, megkérheti Alice-t, hogy monjda meg, melyik számra gondolt eredetileg. Alice megmondja neki M-et így Bob kiszámolhatja $h(M)$-et és összehasonlíthatja H-val.

Látható, hogy Alice mindig tudna nyerni, ha olyan ütközéseket tudna előállítani, ahol $h(M) = h(M')$ és M illetve M' közül egyik páros, másik páratlan.

Biztonságos kulcscsere

A Diffie–Hellman-féle kulcscserét modellezni lehet egy hash függvény és a születésnap-paradoxon segítségével is. Tegyük fel, hogy Alice és Bob meg szeretnének egyezni egy közös és titkos AES kulcsban. Először kiválasztanak egy r bit hosszúságú lenyomatokat készítő h hash függvényt. Alice véletlenszerűen választ $2^{\frac{r}{2}}$ kulcsot, amiket titokban tart. Ezután kiszámolja a kulcsok lenyomatait, és ezeket nyilvánossá teszi. Bob hasonlóan jár el. A születésnap-paradoxon eredményeként, nagyon valószínű, hogy a nyilvánossá tett lenyomatlistákban van közös elem. Az lesz a közös AES kulcs, aminek a lenyomata mindkét listában benne van. Mivel megtörténhet, hogy két különböző kulcsnak ugyanaz legyen a lenyomata, ajánlott egy második hash függvényt használni, és az ezzel készített lenyomatokat is összehasonlítani.

5. fejezet

A digitális aláírás

A digitális aláírás az üzenethez csatolt, korlátozott hosszúságú adat, amelynek segítségével az üzenet címzettje ellenőrizheti a küldő fél vagy az aláíró identitását, valamint meggyőződhet a dokumentum eredetiségéről és épségéről (felismerheti az esetleges hamisítást). Előállításához általában nyilvános kulcsú kriptorendszert és hash függvényt használnak. A hash függvény felel azért, hogy az aláírás az üzenet (az állomány) tartalmától függjön.

5.1. Digitális aláírás általános nyilvános kulcsú kriptorendszer esetében

Először bemutatjuk, hogyan lehet digitális aláírást szerkeszteni egy általános nyilvános kulcsú rendszer segítségével.

Jelöljük E_U-val azt a nyilvános kulcsú titkosítási eljárást, amellyel a rendszer bármely felhasználója titkosított üzenetet tud küldeni az U felhasználónak, és D_U-val a titkos dekódolási eljárást (ezt csak U ismeri). Természetesen $D_U = E_U^{-1}$. Rögzítünk egy h hash függvényt is, amit minden felhasználó ismer.

Tegyük fel most, hogy Alice (A felhasználó) alá akar írni egy Bobnak (B felhasználónak) küldött P üzenetet. Ekkor a következő adatot küldheti Bobnak:

$$C = E_B(P \parallel D_A(E_B(h(P)))),$$

ahol E_B Bob nyilvános titkosítási eljárása, D_A Alice titkos dekódoló eljárása, $\parallel$ pedig a bitsorozatok összeillesztését jelöli. $D_A(E_B(h(P)))$ lesz Alice digitális aláírása, amit – mint látható – a P üzenet után csatol.

Vegyük észre, hogy Alice aláírását rajta kívül senki más nem állíthatja elő, mert az aláírás függ D_A-tól (amit csak Alice ismer). Az aláírás az üzenet tartalamától is függ, és mivel egy hash függvény lenyomatának kódolt változata, a hossza sem túl nagy. Senki sem hamisíthatja meg a P üzenetet, és lehetetlen ezt az aláírást egy másik üzenet esetében is használni. Nagyon fontos dolog még, hogy az aláírás személyre van szabva (ebben az esetben csak Bobnak szól): E_B használata miatt Bob nem tudja elküldeni a P üzenetet egy harmadik félnek, Alice aláírásával. Alice csak Bobnak írta alá a dokumentumot.

Hogyan tudja Bob leellenőrizni Alice digitális aláírásának hitelességét? Bob megkapja C-t, és kiszámolja titkos eljárásával $(P' \parallel S) = D_B(C)$-t, ami egy értelmes szöveg (Alice üzenete, P'), végén egy pár bájtnyi „értelmetlen” adattal (Alice aláírása, S). Ezután leellenőrzi a $h(P') = D_B(E_A(S))$ egyenlőséget. Ha az egyenlőség fennáll, akkor az aláírás valódi, és az üzenet is az eredeti ($P = P'$, vagyis nem sérült és nem volt hamisítva). Ha az egyenlőség nem áll fenn, akkor az üzenet és/vagy az aláírás sérült (vagy akarattal módosították).

5.2. A digitális aláíró algoritmus (DSA)

Tekintsünk most egy gyakorlatban használt, szabványosított aláírási módszert, a DSA-t (Digital Signature Algorithm). A szerzője D. Kravitz, a NIST 1991-ben ajánlotta szabványosításra, végül 1993-ban fogadták el a DSS (Digital Signature Standard) részeként. Apró módosításokat hajtottak végre rajta 1996-ban, majd a szabványt 2000-ben kiterjesztették. A DSS-nek az AES-hez hasonló szerepe van: egy biztonságos szabvány digitális aláírások generálására, amit állami hivalatoktól a magánszemélyekig mindenki használhat.

A DSS a h = SHA-1 hash függvényt használja és egy nyilvános kulcsú titkosítási rendszert, aminek alapja a diszkrét logaritmus problémája a véges testekben. A következőkben megadjuk az algoritmus fő lépéseit:

1. **A kulcsok generálása** (ezt a lépést a rendszer összes felhasználója elvégzi)

    - Válassz egy 160 bites prímszámot, legyen ez q.
    - Válassz egy L bites p prímszámot úgy, hogy $p = 1 \bmod q$, $512 \le L \le 1024$ és L osztható 64-gyel.
    - Válassz egy $h \in \mathbb{N}$ értéket úgy, hogy $1 < h < p-1$ és $1 < g = h^{\frac{p-1}{q}} \bmod p$.
    - Válassz véletlenszerűen egy $x \in \mathbb{N}$ értéket, ahol $0 < x < q$.
    - Számítsd ki $y = g^x \bmod p$-t.
    - A nyilvános kulcs (p, q, g, y), a titkos kulcs pedig x. Vegyük észre, hogy a (p, q, g) számhármast a rendszer több felhasználója is használhatja.

2. **Az üzenetek aláírása**

    - Minden egyes üzenet esetében válassz egy egyszer használatos véletlenszerű k számot, ahol $0 < k < q$.

- Számítsd ki $r = (g^k \bmod p) \bmod q$-t.
- Számítsd ki $s = (k^{-1}(h(M) + xr)) \bmod q$-t, ahol $h(M)$ az M üzenet SHA-1 lenyomata.
- Generálj másik k értéket, és végezd el újra az előbbi két műveletet abban az esetben, ha $r = 0$ vagy $s = 0$ (ennek a valószínűsége elég kicsi).
- Az aláírás az (r, s) pár lesz.

3. **Az aláírás ellenőrzése**

- Az aláírás hiteltelen, ha a $0 < r < q$ valamint $0 < s < q$ egyenlőtlenségek nem teljesülnek.
- Számítsd ki $w = s^{-1} \bmod q$-t.
- Számítsd ki $u_1 = (h(M)w) \bmod q$-t.
- Számítsd ki $u_2 = rw \bmod q$-t.
- Számítsd ki $v = ((g^{u_1} y^{u_2}) \bmod p) \bmod q$-t.
- Az aláírás akkor valódi, ha $v = r$.

5.2.1. tétel. *A DSS helyes.*

Bizonyítás. Az előbbi jelöléseket használva a kis Fermat-tétel alapján $g = h^{\frac{p-1}{q}} \bmod p$-ből következik, hogy $g^q \equiv h^{p-1} \equiv 1 \pmod{p}$. Mivel $g > 1$ és q prím, g-nek a rendje q a $\mathbb{Z}_p^*$ csoportban. Az aláíró kiszámítja s-t, ahol

$$s = (k^{-1}(h(M) + xr)) \bmod q,$$

tehát következik, hogy

$$k = h(M)s^{-1} + xrs^{-1} \equiv h(M)w + xrw \pmod{q}.$$

Tekintsünk most egy z egész számot. $\mathbb{Z}_p^*$-ben a következő egyenlőségünk van:

$$g^k = g^{h(M)w+xrw+qz} = g^{h(M)w} y^{rw} \equiv g^{u_1} y^{u_2} \pmod{p}.$$

Tehát

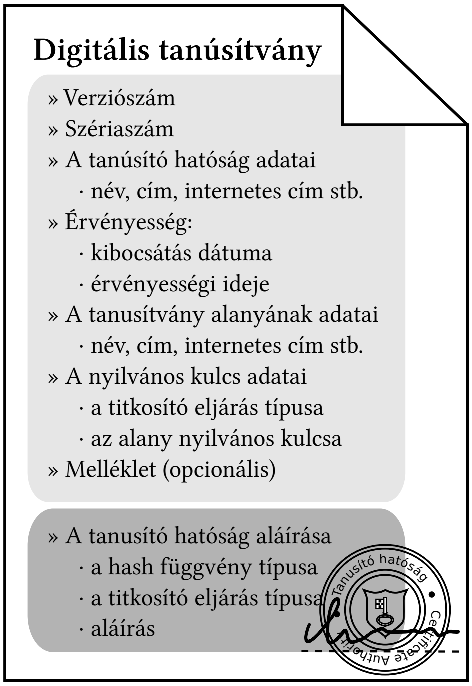
$$r = (g^k \bmod p) \bmod q = (g^{u_1} y^{u_2} \bmod p) \bmod q = v,$$

ami bizonyítja az algoritmus helyességét. □

5.3. A digitális tanúsítvány

A digitális tanúsítvány olyan, mint az aláíró elektronikus személyi igazolványa, mely az aláíróról hiteles és pontos adatokat szolgáltat. A digitális tanúsítványt egy tanúsító hatóság (angolul *Certificate Authority*, rövidítve CA) adja ki, akiben a rendszer összes felhasználója feltétlenül megbízik. A tanúsító hatóság által kibocsátott elektronikus dokumentum minden kétséget kizáróan bizonyítja az alany aláíró (titkos) és ellenőrző (nyilvános) kulcsának összetartozását, és olyan adatokat tartlamaz, ami alapján a tanúsítvány alanyát egyértelműen azonosítani lehet.

5.1. ábra. Digitális tanúsítvány



A digitális tanúsítványoknak kiemelkedően fontos szerepük van az internetes alkalmazások bizonságossá tételében, ugyanis ezek garantálják bizonyos internetes szolgáltató vagy szerver azonosságának valódiságát. Egy online banki szolgáltatásokat nyújtó alkalmazás esetében például minden kétséget kizáróan meg kell bizonyosodjunk arról, hogy az adott alkalmazás tényleg a bank szerverén fut-e vagy sem. Ellenkező esetben megtörténhet, hogy bankkátyánk adatait egy másik (például internetes bűnözők által üzemeltetett) szerveren futó alkalmazás keretén belül adjuk meg. Egy interneten használt digitális tanúsítványnak két fő része van: egy adatokat tartalmazó rész és a tanúsító hatóság digitális aláírása. Az első, adatokat tartalmazó részben a következő elemek vannak: a digitális tanúsítvány verziószáma (amely meghatározza, hogy formátuma melyik szabánynak felel meg), szériaszám (ez minden tanúsítvány esetében egyedi és a tanúsító hatóság bocsájtja ki), a tanúsítvány alanyát azonosító adatok (a szerver internetes címe, tulajdonosának elérhetőségei stb.), az alany nyilvános kulcsa és a titkosító eljárás típusa, a kibocsátó hatóságot azonosító adatok, a tanúsítvány érvényességi ideje (kibocsátásának dátuma és az a dátum, ameddig érvényes) és egy fakultatív, egyéb adatokat tartalmazó melléklet. Ezeket követi a második rész, ami tartalmazza a digitális aláíráshoz használt hash függvény és titkosítási eljárás típusát és magát a tanúsító hatóság digitális aláírását (ami nem más, mint a digitális tanúsítvány első részének lenyomata, amit a hatóság a saját titkos kulcsával kódolt).

6. fejezet

Az SSL protokoll

Az SSL (Secure Socket Layer – biztonsági alréteg) egy protokoll réteg, amely a szállítási rétegbeli protokoll (pl. TCP/IP) és valamely alkalmazási rétegbeli protokoll között helyezkedik el. Mint neve is sugallja, az SSL mindenféle forgalom titkosítására használható - LDAP, POP, IMAP és legfőképp HTTP. Webböngészésnél például az SSL biztosítja a biztonságos kommunikációt a kliens (böngésző) és a szerver (webszerver) között.

1994-ben tervezte a Netscape Corporation annak érdekében, hogy biztonságos módon lehessen adatokat közvetíteni a böngésző és a webszerver között (például bankkártyák adatait). Az SSL használatát a Netscape nem korlátozta a HTTP protokollra, hanem használhatóvá tette bármilyen TCP/IP kapcsolat titkosítására. Egy évvel később az SSL fejlesztésének felügyeletét átvette az IETF (Internet Engineering Task Force) és nevét TLS-re változtatta (Transport Layer Security).

Az SSL protokoll két részből áll. Első része az úgynevezett SSL kézfogás (SSL handshake) aminek célja az kommunikáló felek azonosítása és a titkosított csatorna felállítása. A második rész a titkosított csatornán zajló adatcsere. Az SSL kézfogás lépései a következők:

1. A kliens elküldi a szervernek az SSL verziószámát, a titkosító eljárásokkal kapcsolatos beállításait és egyéb a kapcsolat létrejöttéhez szükséges adatokat.

2. A szerver elküldi a kliensnek az SSL verziószámát, a titkosító eljárásokkal kapcsolatos beállításait és egyéb a kapcsolat létrejöttéhez szükséges adatokat. A szerver elküldi a digitális tanúsítványát is (lásd 5.3 alfejezet) és ha a kliens olyan szolgáltatásokhoz akar hozzáférni, aminek eléréséhez a kliens azonosítása szükséges, akkor kéri a kliens digitális tanusítványát.

3. A szerver által küldött adatok alapján a kliens megpróbálja azonosítani a szervert. Ha a szerver azonosítása nem sikeres, akkor a kliens egy hibaüzenettel jelzi a felhasználónak, hogy a kapcsolat nem jöhet létre. Ha a kliens azonosította a szervert, az SSL kézfogás a 4. lépéssel folytatódik.

4. A kapcsolatban eddig generált adatok alapján a kliens (a szerver hozzájárulásával és a használt titkosítási algoritmusnak megfelelően) létrehozza a titkosított kapcsolat elsődleges kulcsát (pre-master secret), a szerver nyilvános kulcsával titkosítja, majd elküldi a szervernek (a szerver nyilvános kulcsa a digitális tanúsítványában található).

5. Ha szükséges a kliens azonosítása (ez a kézfogás opcionális lépése) akkor a titkosított elsődleges kulcs mellett a kliens elküldi a digitális tanúsítványát is a szervernek, aminek alapján a szerver megpróbálja azonosítani a klienst. Ha a kliens azonosítása nem sikerült, a kapcsolat lezárul.

6. A szerver titkos kulcsával dekódolja a klienstől kapott elsődleges kulcsot, majd ebből kiszámolja a szimmetrikus titkosításhoz használt kulcsot. A kliens is ugyanazzal a módszerrel kiszámolja az elsődleges kulcsból a kulcsot.

7. A kliens küld egy üzenetet a szervernek, amiben értesíti, hogy minden további adatcsere titkosítva lesz. Ezután küld még egy (titkosított) üzenetet, amivel jelzi, hogy a kézfogás lezárult.

8. A szerver küld egy üzenetet a kliensnek, amiben értesíti, hogy minden további adatcsere titkosítva lesz. Ezután küld még egy (titkosított) üzenetet, amivel jelzi, hogy a kézfogás lezárult.

Az SSL kézfogás ezennel mindkét fél részéről lezárult. A kliens és a szerver a továbbiakban a (szimmetrikus eljárással) titkosított csatornán kommunikálnak. Az SSL egyik legbiztonságosabb felállása RSA-t használ a kulcscserére, SHA-1 hash függvényt hitelesítésre és 168 bites tripla DES-t a titkos kommunikációra.

I. függelék

Néhány bonyolultságelméleti alapfogalom

Az alábbiakban leegyszerűsített formában (a teljesség és pontosság igénye nélkül) tisztázunk néhány bonyolultságelméleti fogalmat. További részletekért lásd például [6].

Egy adott problémára különféle megoldási algoritmusok létezhetnek. A cél ezen algoritmusok futásidejének matematikai becslése, a bemeneti adatmennyiség függvényében. Így egy hiteles választ kapunk arra, hogy az adott problémára melyik a leggyorsabb megoldási algoritmus.

A futásidő egyik lehetséges matematikai értelmezéséhez az algoritmust lebontjuk elemi lépésekre, úgynevezett bitműveletekre. A mi esetünkben legyen egy *bitművelet* két bit összeadása a műveletet kísérő esetleges 1-es átvitelével (ez az átvitel akkor történik meg, ha két 1-est adunk össze; ekkor az eredmény 0 és egy 1-es továbbmegy). Egy algoritmus *elméleti futásideje* legyen az algoritmus által elvégzett bitműveletek száma. Ez nyilván a bemeneti adatok függvénye lesz.

Vegyük észre, hogy az így definiált futásidő független a használt számítógép műszaki adottságaitól (memória nagysága, processzor kapacitása). Nyilvánvaló, hogy egy adott számítógépen az adott algoritmus valódi futásideje az előbb értelmezett futásidő szorozva,egy a számítógép kapacitását jellemző állandóval (mely kisebb jobb gépeknél és nagyobb a gyengébb gépeknél).

Ha B az A algoritmus bemeneti adatait jelöli (és $n(B)$ ezen adatok össznagyságát bitekben), akkor legyen $f_A(B)$ a megfelelő futásidő. Nyilvánvaló, hogy $f_A(B) < f_A(B')$, ha $n(B) < n(B')$.

A fenti leegyszerűsített megközelítéssel egy baj van. Általában nehéz pontosan meghatározni a bitműveletek számát. Ezekre legtöbbször csak felső becslést lehet adni. Továbbá jó lenne, hogyha ez a felső becslés már csak a bemeneti adatmennyiség méretétől függne, vagyis kapnánk egy $g_A(n)$ függvényt úgy, hogy $f_A(B) \leq g_A(n)$ minden olyan B bemenetre, melyre $n(B) = n$. Ez azt jelenti, hogy $g_A(n)$ nagyobb az adott n méretű „legrosszabb bemenetnek” megfelelő futásidőnél is. Nyilvánvalóan a legszorosabb becslés lenne a leghasznosabb, de ezt sokszor nehéz megtalálni. Legtöbbször egy adott becslés idővel szorosabb becslésre cserélődik, de azt, hogy ez utóbbi-e a legszorosabb, nem tudjuk igazolni.

Becsléseknél igen hasznos az úgynevezett „nagy O jelölés” bevezetése.

Értelmezés. *Legyenek $f, g : \mathbb{N} \to \mathbb{R}$ fügvények úgy, hogy $f(n), g(n) \geq 0$, $\forall n \geq n_0$ (vagyis elég nagy természetes számokra pozitívak). Azt mondjuk, hogy $f(n) = O(g(n))$ ha $\exists c \in \mathbb{R}_+$ és $\exists n_1 \in \mathbb{N}$ úgy, hogy $0 \leq f(n) \leq cg(n)$, $\forall n \geq n_1$ (vagyis elég nagy értékekre a g egy konstans erejéig nagyobb f-nél, vagy másképp fogalmazva a g konstans erejéig gyorsabban növekszik f-nél).*

Az alábbiakban felsorolunk néhány alaptulajdonságot.

Tulajdonság. *Legyenek $f, g, h, l : \mathbb{N} \to \mathbb{R}$ nagy értékekre pozitív függvények. Ekkor:*

1. $f(n) = O(f(n))$.
2. Ha $f(n) = O(g(n))$ és $g(n) = O(h(n))$, akkor $f(n) = O(h(n))$.
3. Ha $f(n) = O(g(n))$ és $h(n) = O(l(n))$, akkor $(f \cdot h)(n) = O(h(n) \cdot l(n))$.
4. Ha $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ véges, akkor $f(n) = O(g(n))$, mitöbb, ha véges és nem nulla, akkor $f(n) = O(g(n))$ és $g(n) = O(f(n))$.
5. Ha $f(n) = a_0 + a_1 n + \cdots + a_k n^k$, $a_k > 0$ egy k-ad fokú polinomfüggvény, akkor $f(n) = O(n^k)$.
6. Legyen $0 < \delta < 1 < \alpha$. Akkor elég nagy n értékekre, akár egy konstanssal való szorzás erejéig igaz, hogy

$$\log_2 \log_2 n < \log_2 n < e^{\sqrt{\log_2 n \log_2 \log_2 n}} < n^\delta < n^\alpha < n^{\log_2 n} < \alpha^n < n^n < \alpha^{\alpha^n}.$$

A fenti O jelölést használva egy reálisabb és gyakorlatiasabb értelmezést adhatunk a futásidőre.

Értelmezés. *Azt mondjuk, hogy egy A algoritmus futásideje (bonyolultsága) $O(g(n))$, ha létezik $n_0 \in \mathbb{N}$ és $c \in \mathbb{R}_+$ úgy, hogy $f_A(B) \leq cg(n)$ minden B bemenetre, melyre $n(B) = n \geq n_0$ (ahol $f_A(B)$ a B bemenetnek megfelelő elméleti futásidő).*

Megjegyzés. *Nyilván a fenti értelmezésben szereplő $g(n)$ függvény a legtöbb esetben kisebb függvényre cserélődhet (szorosabb becslés megtalálása esetén).*

Példa. *Legyenek $a, b \in \mathbb{N}$ úgy, hogy $a, b \leq N$. Ekkor n bitjeinek száma $\lfloor \log_2 a \rfloor + 1$, ami közelíthető $n = \log_2 N$-el. A fenti tulajdonságok alapján könnyen látható, hogy a és b összeadásának (és kivonásának) futásideje $O(\log_2 N) = O(n)$, vagy pontosabban $O(\log_2 a + \log_2 b)$. Hasonlóan látható, hogy a és b szorzásának (és osztásának) futásideje $O(\log_2^2 N) = O(n^2)$, vagy pontosabban $O(\log_2 a \cdot \log_2 b)$.*

Példa. *Ha $N!$ értékét szeretnénk kiszámítani (a faktoriális értelmezése alapján) és $n = log_2 N$ az N bitjeinek száma, akkor az algoritmus futásideje (durva becsléssel) $\mathcal{O}(N^2 log_2^2 N) = \mathcal{O}(2^{2n} n^2)$. Valóban, mivel $N! = N(N-1)!$ tulajdonképpen $N-2$ darab szorzást kell elvégeznünk úgy, hogy az egyik tényező kisebb vagy egyenlő, mint N (tehát legtöbb n bites), míg a másik kisebb vagy egyenlő, mint $N!$ (tehát legtöbb Nn bites). A futásidő durva becsléssel $(N-2)n(Nn) = \mathcal{O}(N^2 n^2)$ lesz.*

Értelmezés. *Egy n bites bemenetű algoritmus* polinomiális, *ha a bonyolultsága $\mathcal{O}(n^k)$. Partikulárisan az algoritmus konstans, lineáris, bináris illetve ternáris, ha a bonyolultsága rendre $\mathcal{O}(1)$, $\mathcal{O}(n)$, $\mathcal{O}(n^2)$ illetve $\mathcal{O}(n^3)$. Egy algoritmust exponenciálisnak nevezünk, ha nem polinomiális.*

Az alábbi táblázat összehasonlítja a különböző algoritmusok effektív futásidejét [2]:

Algoritmus	Komplexitás	Műveletek száma $N = 10^6$-ra	A műveletek elvégéséhez szükséges idő (10^6 művelet/s)
konstans	$\mathcal{O}(1)$	1	1 μs
lineáris	$\mathcal{O}(N)$	10^6	1 s
négyzetes	$\mathcal{O}(N^2)$	10^{12}	1,6 nap
köbös	$\mathcal{O}(N^3)$	10^{18}	32000 év
exponenciális	$\mathcal{O}(2^N)$	10^{301030}	$10^{301006}\times$ a világegyetem becsült kora

A fő kérdés, hogy egy feladatnak van-e polinomiális megoldási algoritmusa. A válasz a feladatokat úgynevezett *bonyolultsági osztályokba* osztja.

Értelmezés. *Egy feladat P* osztályú, *ha van rá polinomiális megoldási algoritmus. Az ilyen feladatokat még* könnyű*nek (rövid idő alatt megoldhatónak) mondjuk. Egy feladat* nehéz *(hosszú idő alatt megoldható vagy számításilag kivitelezhetetlen) ha nincs P-ben.*

Értelmezés. *Egy feladat NP* osztályú, *ha egy általunk megadott megoldási tipp polinomiális idő alatt leellenőrizhető.*

Például ha a feladat $N \in \mathbb{N}$ faktorizálása, akkor egy felbontási tipp egyszerű összeszorzással leelenőrizhető.

Világos, hogy $P \subseteq NP$ (a tipp P-ben üres). Megtörténhet, hogy egy feladat idővel átkerül NP-ből P-be (például annak eldöntése, hogy egy szám prím-e [1]) azonban a sejtés az, hogy $P \neq NP$.

Értelmezés. *Egy feladat NP*-teljes, *ha NP osztályú és minden NP-feladat megoldása polinomiális idő alatt visszavezethető ezen feladat megoldására.*

Ha igazolni tudnánk, hogy egy NP-teljes feladat P-ben van, akkor $P = NP$; mivel azonban éppen az ellenkezőjét sejtik, az NP-teljes feladatok a legesélyesebbek arra, hogy számításilag kivitelezhetetlenek legyenek.

Fontos példák NP-teljes feladatokra:

- az *utazóügynök-probléma*, amelyben adva van n város, illetve az útiköltség bármely két város között, és meg kell keresnünk a legolcsóbb utat egy adott városból indulva, amely minden várost pontosan egyszer érint, majd a kiindulási városba ér vissza;
- a *hátizsák-probléma* szerint, ha adottak a $v_1, ..., v_n$ térfogatú tárgyak és egy V zsáktérfogat, válasszunk ki tárgyakat úgy, hogy össztérfogatuk V legyen.

Kriptográfiai szempontból sokszor az átlagos bonyolultság fontosabb a legrosszabb eset bonyolultságánál. Ilyen értelemben hasznosak az úgynevezett probabilisztikus polinomiális algoritmusok (az ilyen algoritmussal megoldható feladatok osztályát BPP-vel jelöljük). Lényegük, hogy polinomiális idő alatt futnak, azonban csak egy bizonyos (általában elég nagy) valószínűséggel oldják meg a problémát, a maradék esetben vagy rossz megoldást szolgáltatnak (Monte Carlo-algoritmusok), vagy egyáltalán nem adnak meg megoldást (Las Vegas-algoritmusok).

II. függelék

Nagy prímszámok véletlenszerű generálása

Az alábbiakban bemutatunk egy módszert, amellyel véletlenszerű, nagy prímeket lehet generálni. Feltételezzük, hogy 100 számjegyes véletlenszerű prímeket akarunk kapni. Először is szükségünk lesz a következő prímszámtételre:

Tétel. *(Hadamard, de la Vallé-Poussin, Erdős, Selberg)*

$$\lim_{n\to\infty} \frac{|\{p \le n | p \text{ prím}\}|}{\frac{n}{\ln n}} = 1.$$

A fenti tétel szerint, ha n elég nagy, akkor 1 és n között körülbelül $\frac{n}{\ln n}$ prímszám van. Tehát 100 számjegyű prímből van körülbelül $\frac{10^{100}}{\ln 10^{100}} - \frac{10^{99}}{\ln 10^{99}}$, ami a 100 számjegyű páratlan számoknak nagyjából a $0,9\%$-a. Ha a prímek eloszlása egyenletes lenne, akkor körülbelül 100-150 véletlenszerű páratlan szám közül legalább egy prímszám kellene, hogy legyen. A tapasztalat azt mutatja, hogy egy véletlenszerű 100 számjegyű n páratlan szám után következő 200 páratlan számból ($n, n+2, \ldots, n+400$) legalább egy prímszám. Persze ez azt jeleni, hogy a 200 páratlan számra le kell futtatni egy lehetőleg gyors prímtesztet.

Egy ilyen gyors prímteszt a *Miller–Rabin-teszt*. A Miller–Rabin-teszt a következő tulajdonságra épül:

Tulajdonság. *Ha n prím és* $n-1 = 2^s t$, *ahol s páratlan, akkor bármely b-re úgy, hogy* $(b, n) = 1$ *fennáll a következő két összefüggés valamelyike:*

1. $b^t \equiv 1 \pmod n$

2. $\exists r \in \mathbb{N}, 0 \le r \le s-1$ *úgy, hogy* $b^{2^r t} \equiv -1 \pmod n$.

Értelmezés. *Legyen n egy páratlan összetett szám,* $n-1 = 2^s t$, *t páratlan és legyen b olyan, hogy* $(b, n) = 1$. *Ekkor azt mondjuk, hogy n erős pszeudoprím a b alapra nézve, ha* $b^t \equiv 1 \pmod n$ *vagy* $\exists r \in \mathbb{N}, 0 \le r \le s-1$ *úgy, hogy* $b^{2^r t} \equiv -1 \pmod n$.

Tétel. *Ha n páratlan összetett szám, akkor n erős pszedoprím a b alapra nézve a lehetséges* $0 < b < n$, $(b, n) = 1$*-ek legfennebb* 25%*-ára.*

A Miller–Rabin-teszt. Legyen n egy nagy (100 számjegyű) páratlan szám. El akarjuk dönteni, hogy n prímszám-e vagy sem. Legyen $n - 1 = 2^s t$ úgy, hogy t páratlan. Választunk egy véletlenszerű b természetes számot úgy, hogy $0 < b < n$ és $(b, n) = 1$. Ezután kiszámoljuk b^t mod n-et. Ha ± 1-et kapunk, akkor n átment a teszt első lépésén, és választunk egy másik véletlenszerű b alapot. Ha b^t mod $n \neq \pm 1$, akkor kiszámoljuk sorra b^{2t} mod n, $b^{2^2 t}$ mod n, $\ldots$, $b^{2^{s-1} t}$ mod n-et. Ha valamelyik ezek közül -1, akkor megállunk, és n átment a teszten. Ha viszont egyik sem -1, akkor n elbukta a b alapra a tesztet, következésképpen n biztosan összetett.

Ha n átmegy a teszten k darab különböző alapra, akkor a fenti tétel alapján annak az esélye, hogy n mégis összetett legyen $\left(\frac{1}{4}\right)^k$, tehát n $1 - \left(\frac{1}{4}\right)^k$ valószínűséggel prímszám.

A Miller–Rabin-teszt nagyon gyors, körülbelül harmadfokú polinomiális. Ha biztosak akarunk lenni, hogy egy szám, mely a Miller–Rabin-teszt szerint nagy valószínűséggel prím, valóban prím-e, lefuttathatjuk rá a sokkal lassúbb (körülbelül kilencedfokú polinomiális) AKS prímtesztet [1], amely teljes bizonyossággal eldönti, hogy az adott szám valóban prímszám-e vagy sem.

III. függelék

Néhány algebrai alapfogalom

Az alábbiakban összefoglalunk néhány, a könyvben szereplő algebrai alapfogalmat, illetve ezekhez kötődő alaptulajdonságokat.

Csoportok

Értelmezés. *A* $(G, \cdot)$ *csoport ciklikus, ha* $\exists x \in G$ *úgy, hogy* $G = < x > = \{x^k | k \in \mathbb{Z}\}$.

Tulajdonság. *Ha* $(G, \cdot)$ *véges csoport és* $|G| = n$*, akkor* $x^n = 1$, $\forall x \in G$*-re.*

Maradékosztályok gyűrűje modulo $n \geq 2$

Tulajdonság. *A* $(\mathbb{Z}_n, +, \cdot)$ *egységelemes kommutatív gyűrűnek van zérusosztója* $\iff$ n *összetett szám.*

Tulajdonság. $\exists \hat{a}^{-1} \in \mathbb{Z}_n \iff (a, n) = 1$. $(U(\mathbb{Z}_n), \cdot)$ *csoport, ahol* $U(\mathbb{Z}_n) = \{\hat{a} \in \mathbb{Z}_n \,|\, \exists \hat{a}^{-1} \in \mathbb{Z}_n\}$. $|U(\mathbb{Z}_n)| = \varphi(n) = |\{0 \leq a < n \,|\, (a, n) = 1\}|$*, ahol* $\varphi(n)$ *az Euler-függvény.*

Tulajdonság. *Ha* $n = p_1^{\alpha_1} \cdot \cdots \cdot p_\ell^{\alpha_\ell}$*, akkor* $\varphi(n) = n\left(1 - \frac{1}{p_1}\right) \cdot \cdots \cdot \left(1 - \frac{1}{p_\ell}\right)$*. Sajátos esetben, ha* p *prím, akkor* $\varphi(p) = p - 1$.

Tulajdonság. $(\mathbb{Z}_p, +, \cdot)$ *test* $\iff$ p *prím.*

Tétel. *(Euler) Ha* $(a, n) = 1$*, akkor* $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Bizonyítás. Elég annyit belátni, hogy $(a, n) = 1 \Longrightarrow \hat{a} \in U(\mathbb{Z}_n)$, ahol $U(\mathbb{Z}_n)$ csoport, és $|U(\mathbb{Z}_n)| = \varphi(n)$. □

Következmény. *(Fermat) Ha* $p \nmid a$*, akkor* $a^{p-1} \equiv 1 \pmod{p}$.

Bizonyítás. Euler-tétel alkalmazása $n = p$-re. □

Véges testek

Tétel. *(Wedderburn) Minden véges test kommutatív.*

Tulajdonság. *Ha $(K,+,\cdot)$ véges test, akkor $|K| = p^n$, ahol p prímszám.*

Tétel. *Izomorfizmus erejéig egyetlen $q = p^n$ elemszámú test létezik. A q elemszámú test egyik alakja $\mathbb{F}_q = \mathbb{Z}_p[X]/(f)$, ahol f egy n-ed fokú irreducibilis főpolinom $\mathbb{Z}_p[X]$-ben. $\mathbb{F}_q = \mathbb{Z}_p[X]/(f) = \{a_{n-1}X^{n-1} + \cdots + a_1X + a_0 \bmod f \,|\, a_i \in \mathbb{Z}_p\}$.*

Az $\mathbb{F}_q$ testbeli műveletek:

- Összeadás: polinomok összeadása.
- Szorzás: polinomok szorzása, majd redukálása modulo f (vagyis f-fel való osztási maradék).

Az $\mathbb{F}_q$ testben egy $g = a_{n-1}X^{n-1}+\cdots+a_1X+a_0 \neq 0$ elemet a következő módon invertálunk (megszerkesztjük $g^{-1} \bmod f$-et). Mivel f irreducíbilis n-ed fokú $\Longrightarrow (g, f) = 1$ $\Longrightarrow$ kiterjesztett euklidészi algoritmussal lehet találni olyan $u, v \in \mathbb{Z}_p[X]$ elemeket, hogy $gu + fv = 1 \Longrightarrow gu \equiv 1 \pmod f \Longrightarrow g^{-1} \bmod f = u$.

Példa. *Legyen a véges test $\mathbb{F}_q = \mathbb{Z}_3[X]/(X^2 - X - 1)$, ahol $q = 3^2$ és $f = X^2 - X - 1$ irreducíbilis $\mathbb{Z}_3[X]$-ben. Nyilván $X^2 - X - 1 \equiv 0 \pmod f \Longrightarrow X^2 \equiv X + 1 \pmod f$.*

$X(X + 1) \equiv X^2 + X \equiv 2X + 1 \pmod f$.

$$\begin{array}{r|l}
X^2 + X & X^2 - X - 1 \\ \cline{2-2}
-X^2 + X + 1 & \hfill 1 \\ \cline{1-1}
2X + 1 &
\end{array}$$

$(X, f) = 1$, tehát X invertálható. Határozzuk meg inverzét, $X^{-1} \bmod f$-et. A kiterjesztett euklidészi algoritmust fogjuk használni: $X^2 - X - 1 = X(X - 1) - 1 \Longrightarrow 1 = X(X - 1) - (X^2 - X - 1) \Longrightarrow X(X - 1) \equiv 1 \pmod f \Longrightarrow X^{-1} \equiv X - 1 \pmod f$.

Valóban, $X(X - 1) = X^2 - X = \underbrace{X^2 - X - 1}_{=0} + 1 \equiv 1 \pmod f$.

Tétel. *Ha $(K,+,\cdot)$ véges test, akkor $(K^*,\cdot)$ csoport ciklikus.*

Diszkrét logaritmus

Értelmezés. *Legyen $(G,\cdot)$ egy véges csoport és $g \in G$ úgy, hogy $\mathrm{ord}(g) = n$ (vagyis $n > 0$ a legkisebb természetes szám, amelyre teljesül, hogy $g^n = 1$). Tételezzük fel, hogy $y \in G$ a g-nek valamilyen hatványa. Ekkor y g-alapú diszkrét logaritmusa $\log_g y = x \in \{0, \ldots, n - 1\}$, ha $g^x = y$.*

Példa. *Legyen* $G = (\mathbb{F}_9^*, \cdot)$ *úgy, hogy* $\mathbb{F}_9 = \mathbb{Z}_3[X]/(f)$*, ahol* $f = X^2 - X - 1$*. Ekkor tudjuk, hogy* $(\mathbb{F}_9^*, \cdot)$ *ciklikus, és belátható, hogy* $\mathbb{F}_9^* =< X >$*. Valóban:*

$$
\begin{aligned}
&X^0 \equiv 1 \pmod{f}\\
&X^1 \equiv X \pmod{f}\\
&X^2 \equiv X + 1 \pmod{f}\\
&X^3 \equiv X^2 + X \equiv 2X + 1 \pmod{f}\\
&X^4 \equiv X^2 + 2X + 1 \equiv 3X + 2 \equiv 2 \pmod{f}\\
&X^5 \equiv 2X \pmod{f}\\
&X^6 \equiv 2X^2 \equiv 2X + 2 \pmod{f}\\
&X^7 \equiv 2X^2 + 2X \equiv X + 2 \pmod{f}
\end{aligned}
$$

Ekkor $\log_X(2X+1) = 3$, $\log_X(X+1) = 2$, $\log_X X = 1$, $\log_X(2X) = 5$, $\log_X(2X+2) = 6$ *és* $\log_X(X + 2) = 7$.

IV. függelék

Magyar-angol kriptográfiai fogalomtár

MAGYAR	ANGOL
affin-rejtjel	affine cipher
aszimmetrikus kulcsú kriptorendszer	asymmetric key cryptosystem
álvéletlen szám	pseudorandom number
átrendezéses kód	transposition cipher
betűgyakoriság-vizsgálat	frequency analysis
bonyolultság	complexity
bonyolultsági osztály	complexity class
Caesar-kerék, Caesar-tárcsa	Caesar wheel
Caesar-kód, Caesar-rejtjel	Caesar cipher
cikcakk-rejtjel	rail rence cipher, zigzag cipher
ciklikus csoport	cyclic group
csapóajtó-függvény	trapdoor function
dekódoló eljárás	decoding algorithm
differenciál-kriptoanalízis	differential cryptanalysis
digitális aláírás	digital signature
digitális tanúsítvány	digital certificate
diszkrét logaritmus	discrete logarithm
dupla oszlopos transzpozíció	double columnar transposition cipher
egyirányú függvény	one-way function
eredeti üzenet	plaintext, cleartext
folyamtitkosító	stream cipher
futásidő	run-time
hátizsák-probléma	knapsack problem
hash függvény	hash function
helyettesítő kód	substitution cipher

jelszavas azonosítás	password authentication
kötelezettségvállalási séma	commitment scheme
kódkönyv	codebook
kódolt szöveg	ciphertext
Kerckhoffs-elv	Kerckhoffs' principle
keveréses kód	transposition cipher
kimerítő kulcskeresés	brute force attack
kitöltés	padding
kriptoanalízis	cryptanalysis
kulcscsere	key exchange, key establishment
kulcsfolyam	keystream
kulcsszavas Caesar-kód	keyword Caesar cipher
lavina-effektus	avalanche effect
lineáris kriptoanalízis	linear cryptanalysis
második őskép-támadás	second preimage attack
nyílt üzenet, nyílt szöveg	plaintext, cleartext
nyilvános kulcsú kriptográfia	public-key cryptography
oszlopos transzpozíció	columnar transposition cipher
őskép-támadás	preimage attack
S-doboz	*S*-box
Shamir háromlépéses protokollja	Shamir three-pass protocol
születésnap-paradoxon	birthday paradox
szimmetrikus kulcsú kriptorendszer	symmetric-key cryptosystem
szupernövekvő hátizsák feladat	superincreasing knapsack problem
szupernövekvő sorozat	superincreasing sequence
tömörítő függvény	compression function
tömbtitkosító	block cipher
tömbtitkosító működési módja	block cipher mode of operation
titkos kulcsú kriptorendszer	secret key cryptosystem
titkosított szöveg	ciphertext
utazóügynök-probléma	travelling salesman problem
ütközés	collision
ütközéses támadás	collision attack
üzenetek sértetlenségének ellenőrzése	message integrity check
üzenethitelesítő-kód	message authentication code (MAC)
véletlen átkulcsolás	one-time pad

Irodalomjegyzék

[1] M. Agrawal, N. Kayal, N. Saxena, *Primes is in P*, Ann. Math. 160, 781-793, 2004.

[2] S. Crivei, A. Mărcuş, C. Săcărea, Cs. Szántó, *Computational Algebra with Applications to Coding Theory and Cryptography*, EFES Cluj-Napoca, 2006.

[3] N. Ferguson, B. Schneier, *Practical Cryptography*, Wiley Publishing, 2003.

[4] A. Kerckhoffs, *La cryptographie militaire*, Journal des sciences militaires, vol. IX, pp. 5–83, pp. 161–191, 1883.

[5] R. E. Klima, N. Sigmon, E. Stitzinger, *Applications of Abstract Algebra with Maple and MATLAB, 2nd edition*, CRC Press, 2006.

[6] N. Koblitz, *A Course in Number Theory and Cryptography*, Graduate Texts in Mathematics, Vol. 114, Springer, 1994.

[7] S. Landau, *Standing the Test of Time: the Data Encryption Standard*, Notices of the American Mathematical Society, March 2000, pp. 341-349.

[8] S. Lucks, *Design principles for iterated hash functions*, e-print, 2004.

[9] A. Marcus, *Komputeralgebra*, Presa Universitară Cluj-Napoca, 2005.

[10] A. Salomaa, *Public-Key Cryptography*, Springer, 1990.

[11] S. Singh, *Kódkönyv*, Park kiadó, 2007.

Tárgymutató