1. **A kriptográfia és az RSA – algoritmus:**
   1. **A kriptográfia**

A kriptográfia, másnéven titkosítás vagy rejtjelezés egy több ezer éves múltra visszatekintő tudomány. Célja nem más, minthogy valamilyen „írott” szöveget úgy alakítson át, hogy az illetéktelen személyek számára visszafejthetetlen, vagy kvázi – visszafejthetetlen legyen. A kriptográfiai algoritmus az az eljárás, mellyel során a titkosítás történik. Egy titkosítás akkor tekinthető jónak, ha a „visszafejthetetlensége” mellett rendelkezik azzal a tulajdonsággal is, hogy a jogosult személy(ek) valamilyen kulcs birtokában képesek a rejtjelezett üzenetet hatékonyan és egyértelműen visszakapni. A kriptográfiával körülbelül egyidejű tudomány a kriptoanalízis, mely azzal foglalkozik, hogy a titkosított üzeneteket bármilyen kulcs, vagy jogosultság nélkül visszafejtse vagy feltörje.

* + 1. **A szimmetrikus kulcsú titkosítás**

A szimmetrikus kulcsú titkosítás legfőbb jellemzője, hogy ebben az esetben a titkosításhoz és a visszafejtéshez használt kulcs megegyezik. Ennek eredményeképpen ezt a kulcsot feltétlenül titokban kell tartani. Ez a titkosító eljárások legegyszerűbb módja. A régebbi titkosító eljárások ezen az alapon működtek. Ilyen például a betűeltolásos titkosítás, a DES (1976) (vagy későbbi változata a 3DES), az AES (1997) vagy az IDEA (1991). Ennek a fajta rejtjelezésnek nagy előnye az egyszerűsége, hátránya azonban az, hogy mivel a titkosító és visszafejtő kulcs közös, ezért azt a kommunikáció előtt meg kell osztani a kommunikáló felek között, ami egy nagyfokú biztonsági kockázatot jelent. További nehézség az, hogy a kommunikáció összes résztvevőjének különböző kulcsot kell biztosítani, hiszen amennyiben ezek megegyeznének, kölcsönösen el tudnák olvasni egymás üzeneteit.További érdekesség, hogy a második világháborúban a Harmadik Birodalom által használt Enigma nevet viselő titkosító berendezés is ilyen elven működött. Mint azt ez az eset is mutatja, a számítógépek megjelenésével, majd ezek teljesítményének növekedésével ezek a fajta kódok viszonylag egyszerűen feltörhetővé váltak.

* + 1. **Az asszimmetrikus (vagy nyílt) kulcsú titkosítás**

Mivel a szimmetrikus kulcsú titkosítási eljárások nem nyújtottak elegendő biztonságot, szükség volt újfajta eljárások kidolgozására. 1976-ban Diffie és Hellman publikált egy új titkosítási eljárást, melynek lényege az volt, hogy a kommunikáció minden résztvevője két, nem azonos kulccsal rendelkezik. A két kulcs (jellemzően e-vel és d-vel jelöljük őket) a nyilvános (vagy publikus) kulcs (az e-vel jelölt), míg a d-vel jelölt kulcs a titkos (vagy privát) kulcs. Fontos megjegyezni, hogy ez az titkosítási eljárás úgy van kidolgozva, hogy a privát kulcs ismeretében a visszafejtés gyors, azonban a publikus kulcsból a privát kulcs meghatározása „nagyon nehéz”, ideális esetben lehetetlen. De mit jelent az, hogy nagyon nehéz? Ideális esetben ez azt jelentené, hogy bizonyítottan nem létezik olyan algoritmus, mely képes a problémát (jelen esetben a titkos kulcs visszafejtését a nyilvános kulcsból) polinomiális időben megoldani. Sokszor előfordul azonban, hogy bár nem ismerünk olyan algoritmust amire ez igaz lenne, nem bizonyított a tény, hogy nem is létezik ilyen algoritmus. De miért baj az, hogy a titkos kulcs nem visszafejthető polinomiális időben? Ez azt jelenti, hogy a kulcs ugyan visszafejthető, de megfelelően bonyolult kulcsok és titkosítási eljárás választása esetén ez észszerűtlenül sok időt venne igénybe. Valamint, hogy a kulcsok hosszának növelésével a számítási idő exponenciális jelleggel növekszik. Ilyen elven működik napjaink egyik leggyakrabban használt titkosító algoritmusa, az RSA algoritmus is.

* 1. **Az RSA algoritmus**

Az RSA – algoritmus napjaink leggyakrabban használt titkosítási eljárásaink egyike. Használják többek között internetes kommunikáció titkosítására, digitális aláírások létrehozására, illetve az elektronikus tranzakciók során is. Legfőbb jellemzője, hogy nyílt kulcsú, más néven aszimmetrikus.

* + 1. **Története**

Az eljárást 1977-ben publikálta Ron Rivest, Adi Shamir és Len Adleman. Az RSA a nevük kezdőbetűiből áll össze.

* + 1. **Alapja**

Az eljárás a nagy számok prímfelbontásának nehézségére alapszik. A számelmélet alaptétele alapján minden 1-nél nagyobb pozitív egész szám felbontható prímszámok szorzatára, és ez a felbontás a sorrendtől eltekintve egyértelmű. A problémát az adja, hogy nagy számok esetén jelenleg nem ismerünk minden esetben hatékony algoritmust a prímfelbontásra. Azonban adott hosszúságú számok közül is vannak melyek könnyebben és vannak melyek nehezebben bonthatók fel prímszámok szorzatára. A tudomány jelenlegi állása szerint két, közel azonos nagyságú prímszám szorzataként előálló számok felbontása a legnehezebb, nem véletlen, hogy az RSA algoritmusban is igyekszünk így választani számokat.

* + 1. **Működése**

A használt RSA kód egyik legfontosabb tulajdonsága a használni kívánt modulus hossza (N). Ez a hossz határozza meg a kódoláshoz, és ami még fontosabb: a feltöréshez szükséges számítások idejét. Ez a kezdetben 128 bit volt, azonban ez a technika fejlődésével és a számítógépek teljesítményének növekedésével már nem bizonyult biztonságosnak. A kódolás biztonságát ennek a számnak a növelésével igyekeztek fenntartani, hiszen ahogy korábban említettük, jelen tudásunk szerint nincs olyan algoritmus, mely a prímtényezőkre bontást polinomiális időben végezni tudja. Ennek következményeképpen a modulus hosszának növelése exponenciálisan növeli a feltöréshez szükséges időt. A későbbiekben a modulus hossza volt 256 bit, majd 512 bit is, napjainkban azonban már az 1024 bites modulus hossz az elterjedt, ami 309 decimális számjegynek felel meg. A modulus hosszának növelése olyan szempontból hátrányos, hogy ezáltal a titkosítási és (a kulcs birtokában) dekódolási fázis számításigénye is megnő. A titkosítás első lépése az, hogy választunk két, közel azonos nagyságú (N/2 bit hosszúságú) prímszámot, általában p-vel és q-val jelöljük őket. Ezek után előállítjuk a φ(N) számot, mely φ(N)=(p-1) \* (q-1) módon áll elő. Ezután választunk két egész számot (jellemzően e-t és d-t), melyekre igaz, hogy e\*d 1 mod φ(N). Azaz, hogy az e\*d szorzat kongruens legyen 1-gyel modulo φ(N). Ez azt jelenti, hogy az e\*d szorzat és az 1 φ(N) ugyanazon maradékosztályába tartoznak. Az így kapott e-t nyilvános exponensnek, míg a d-t privát exponensnek nevezzük. Az (N,e) páros a publikus kulcs, míg az (N,d) páros a privát kulcs. A privát kulcsot kihirdetjük, ennek a segítségével tudnak számunkra üzenetet küldeni, míg a privát kulcsot titokban tartjuk, ezzel tudjuk a nekünk küldött kódolt üzeneteket visszafejteni. Az üzenetküldés úgy zajlik, hogy fogjuk az M üzenetet, mely leképezhető valamilyen egész számra, majd elvégezzük a számítást, mellyel előáll a titkosított C üzenet. Ezek után a C üzenetet elküldik nekünk valamilyen csatornán. A hozzánk beérkezett kódolt üzenetből mi pedig az számítás segítségével visszakapjuk az eredeti (M) üzenetet. A kódolás és dekódolás közötti kapcsolat matematikai hátterében az Euler – Fermat tétel áll. De mi történik, amikor valaki elfogja a nekünk küldött titkosított üzenetet?

A nyilvános kulcsunkat mindenki ismerheti, ezért a N és az e számok ismertek az illetéktelenek számára is. Azonban a visszafejtéshez szükség van a d úgynevezett privát exponensre is. A d számításhoz azonban szükséges a φ(N) ismerete, ami, mint p és q szorzata áll elő. Ezeket a számokat az illetéktelen lehallgatók azonban csak úgy kaphatják meg, amennyiben képesek N-t faktorizálni. Ez pedig az a probléma, amelyről korábban megállapítottuk, hogy korántsem egyszerű, sőt „nagyon nehéz”. Már-már annyira nehéz, hogy a gyakorlatban legtöbbször megpróbálni sem érdemes. Azonban van néhány körülmény, melyet, ha figyelmen kívül hagyunk, jelentősen csökkenthetjük a kódolásunk biztonságát, és megkönnyíthetjük támadóink dolgát.

* + 1. **Az RSA támadások**

A támadásokat alapvetően két nagy csoportba oszthatjuk, léteznek az úgynevezett implementáció függő támadások, melyek csak adott helyzetben alkalmazhatóak, illetve vannak a helytelen alkalmazáson alapuló támadások is, ahol a kódoló fél nem kellő körültekintése vezet a kódolás biztonságának csökkenéséhez. Nézzünk ezek közül néhányat, a teljesség igénye nélkül.

* + - 1. **Implementáció függő támadások**

Ezen típusú támadások közös jellemzője, hogy az elméleti megfontolásokon kívül, komoly technikai feltételekkel is rendelkeznek.

A **kulcskereséses támadás** alapja, hogy amennyiben egy, a privát kulcsot használó szervert támadunk. A szerver memóriájában nyilvánvalóan jelen kell lennie a privát a kulcsnak, de ez az idő nagy részében kódoltan tárolódik. Azonban amikor a szerver a kulcsot használja, akkor kell lennie olyan időszakoknak, amikor a kulcs kódolatlanul a memóriában van. Amennyiben a támadó el tudja érni, hogy a rendszer a megfelelő pillanatban összeomoljon, akkor nem maradt más feladata, mint hogy megkeresse a kulcsot a memóriában (vagy, amennyiben ahhoz nincs hozzáférése, akkor a memória dump-ban). Ez azonban egy nagy méretű adathalmaz, ami elméletben nagyon megnehezítené a keresést. Azonban, mivel a kulcsokat általában álvéletlenszám generátorral generálják, az elkészült kulcs olyan adat, melynek entrópiája igen nagy. Ennek az információnak a birtokában már jóval egyszerűbb megtalálni a memóriában a megfelelő, 1024 bites bitsorozatot.

A **számítási idő mérésén alapuló támadások** abban az esetben kerül előtérben, amikor az RSA műveletek végzése közben a kulcs nem kerül ki a rendszerből. Ilyenek például a smarcardokon alapuló alkalmazások. Itt nyilvánvalóan nem férünk hozzá a rendszer memóriájához sem. Ilyenkor egy lehetőség annak a mérése, hogy a kártyának mennyi időbe telik egy RSA művelet. Ehhez hasonló eljárás, mikor nem az időt mérjük, hanem a rendszer áramfelvételét, majd ebből következtetünk, az elvégzett műveletekre.

* + - 1. **Helytelen alkalmazáson alapuló támadások**

Ezek a támadások arra alapulnak, hogy a kódolást végző személy, vagy rendszer nem kellő körültekintéssel jár el, a kezdeti prímszámok megválasztásakor. Ennek egyik példája az az eset, amikor p és q számok ikerprímek (azaz q = p + 2), ebben az esetben a feltörés rendkívül egyszerűvé válik. Helytelen alkalmazások esetén, bár az előálló modulus mérete ezt nem indokolná, mégis könnyen feltörhetővé válik a kódolásunk. Nézzünk néhány példát a helytelen alkalmazásokra:

**Közös modulus:** Előfordulhat, hogy a rendszer úgy szeretne erőforrásokat spórolni, hogy nem generál minden résztvevőnek saját modulust, hanem ugyanahhoz a modulushoz generál e és d értékeket. Ez azért veszélyes, mert matematikailag belátható, hogy N faktorizációja, és a privát kulcs ismerete egyenértékű. Ezáltal a kommunikáció minden résztvevője kiszámíthatja más résztvevők privát kulcsát is, csökkentve a rendszer biztonságát.

**Kis privát exponens használata:** A számítási igény csökkentése érdekében néhány alkalmazásban (jellemzően a gyengébb hardware-eken futókban) kis méretű privát exponenst (d) használnak. Igaz, hogy ezáltal a számítási idő jelentősen csökkenthető, viszont matematikailag megmutatható, hogy amennyiben d értéke eléggé kicsi, abban az esetben a kód feltörhetővé válik. Jelen állás szerint a biztonsági határ.

**A modulus faktorizációján alapuló támadások:** A támadások ezen alfaja a publikus kulcs ismeretében N faktorizációjával igyekszik megfejteni a privát exponenst. A ma ismert legjobb algoritmus a Number Field Sieve algoritmus melyet 1993-ban publikáltak. A jelenlegi nagyteljesítményű célgépek segítségével már az 512 bites RSA kulcsok is biztonsági kockázatnak vannak kitéve.

* + - 1. **Kvantumalgoritmusok használata**

Ahogy azt korábban láthattuk, megfelelő körültekintéssel, és a modulus hosszának folyamatos növelésével az RSA kódolás bár nem bizonyítottan, de védve van a klasszikus támadásokkal szemben. Nincs kizárva, hogy a jövőben találnak olyan klasszikus algoritmusokat, amelyek ismeretében ez már nem lesz igaz, azonban ez egyelőre nincs kilátásban. Éppen ezért a RSA kódolásra jelenleg a legnagyobb fenyegetést a kvantumszámítógépek és a kvantumalgoritmusok jelentik. Ezen algoritmusok a Lov Grover indiai-amerikai informatikus 1996-ban publikált Grover algoritmus és a Peter Shor által 1994-ben publikált Shor algoritmus.

1. **A kvantumkeresés**

Az emberiség történelme során mindig is nagy szerepet játszott a rendezetlen adatbázisokban való keresés. Elegendő csak arra gondoljuk, hogy a bennünket körülvevő környezetre lehet, mint erőforrások rendezetlen adatbázisa tekinteni, melyben a megtalálni kívánt erőforrás a keresett adat. A rendezetlen adatbázisokban történő keresés azonban meglehetősen nehézkes. A két alkalmazható keresési algoritmus a véletlen keresés, illetve a kimerítő keresés. Egy N elemű adatbázis esetén azonban mindkettő módszerben legrosszabb esetben N lépés kell, hogy a keresett elemet (adatot) megtaláljuk (amennyiben véletlen keresés esetén nem választunk olyan véletlen elemet, amelyet már választottunk és a keresett elem csak egyszer szerepel az adatbázisban). A számítógépek megjelenésével tovább nőtt az adatbázisokban való keresés hatékonyságát növelő módszerek iránti igény. A született klasszikus megoldások azonban feltételeznek valamilyen rendet az adatbázisunkban, ezért ezen keresőalgoritmusok hatékony működéséhez megfelelő rendező algoritmusokra is szükség van. A rendezetlen keresés hatékonyságának növelésére egészen a kvantumos algoritmusok megjelenéséig kellett várni.

* 1. **A Grover algoritmus**

A Grover algoritmus 1996-os publikálása nagyon csábító eredményeket vetített előre. Az addigi maximális N lépés helyett csupán maximális lépés szükségességét vetítette előre rendezetlen adatbázisokban való keresés esetén.

* + 1. **A Grover algoritmus működése - alapok**

Az algoritmus működéséhez 2 kvantumregiszter szükséges. A „felső” (később az ábrán) n bites. A regiszter n bitszámát úgy választjuk ki, hogy az adatbázis N elemszáma előálljon alakban. Amennyiben N elemszáma nem kettő hatványa, kiegészítjük érdektelen értékekkel. Ezt a regisztert egy n bites Hadamard kapura vezetjük, míg az alsót egy ismeretlen T kapura. A kapuk után előálló állapot, azaz a valószínűségi amplitúdók eloszlása az ábrán látható. Az x értékek az adatbázis elemei, x0 a keresett érték. Gyakori jelölés, hogy az adatbázisban azokat az x elemet/elemeket, melyet/melyeket keresünk jelzett állapotnak/állapotoknak, míg a többi állapotot jelzetlen állapotnak nevezzük. Az amplitúdók értékei egységesen . Ez beleillik a korábban tanultakba, miszerint a valószínűségi amplitúdók négyzetösszegei az 1 értéket kell kiadják, ami ebben az esetben teljesül is. A **G,** azaz a Grover operátor első állomása az úgynevezett Orákulum. Az Orákulum feladata az, hogy felismerje az adatbázisban a jelzett elemet, majd ennek valószínűségi amplitúdóját megszorozza -1-gyel, míg a jelzetlen állapotokét változatlanul hagyja.

A **G** következő lépése, a jelzett állpapot valószínűségi amplitúdójának növelése, míg a jelzetlenek amplitúdóinak elnyomása. Ennek matematikai eszköze az átlag körüli tükrözés. Ennek egy precízebb megfogalmazása az, hogy minden értéket kivonunk az átlag kétszereséből. Az átlagunk valamivel kisebb mint , hiszen van egy értékünk is. Ezt az átlagot jelöljük -val. Látható, hogy ennek hatására a jelzetlen állapotok valószínűségi amplitúdóinak értéke az eredeti átlag alá csökken. Jelzett állapoté azonban 2 + lesz. Ehhez a második lépéshez szükség van két Hadamard kapura és egy vezérelt fáziseltoló kapura (amit P-vel jelölünk), melyet 180o-os forgatásra állítunk be a megfelelő valószínűségi amplitúdóra. Ezek után a **G** operátort a **G** = HPHO alakban írhatjuk fel. Az algoritmust úgy használjuk, hogy a jelzett érték fázistolását, majd az átlag körüli tükrözést újra és újra elvégezzük, ezáltal jobban kiemelve a valószínűségi amplitúdóját, és jobban elnyomva a jelzetlen állapotokét. Fontos kérdés azonban, hogy hányszor éri meg iterálni.

* + 1. **A Grover algoritmus működése – az iterációk száma**

Az iterációk számának eldöntéséhez egy geometriai interpretációt használunk. Ehhez egy olyan bázist írunk fel, melyben a kezdeti |γ1⟩ állapotot felírhatjuk |α⟩ és |β⟩ segítségével. Itt |α⟩ tartalmazza jelzetlen x értékeket míg |β⟩ a jelzetteket. |α⟩ és |β⟩ egy ortogonálist bázist alkotnak, melyben |γ1⟩ |α⟩-val egy fokos szöget zár be. Megmutatható, hogy minden iterációban Ωγ szöget forgatunk |β⟩ felé a |γ1⟩ pozíciójából kezdve. Ωγ-t N és M segítségével számíthatjuk, ahol N a korábbiakhoz hasonlóan az adatbázis elemszáma, M pedig a jelzett elemek száma az adatbázisban. Célunk az, hogy |γ1⟩-et beleforgassuk a |β⟩ tengelybe, majd ezután elvégezzük a mérést. Ez azonban nem mindig tehető meg. Sok esetben meg kell elégednünk azzal, hogy |γ1⟩-et a lehető legközelebb forgatjuk |β⟩-hez. Ez a gyakorlatban azt jelenti, hogy a mérés valamekkora valószínűséggel hibás lesz. Mivel a lehető legkevesebb iterációt szeretnénk végezni, ezért belátható, hogy az optimális iterációszám: . Mint azt a bevezető részben is említettük a Grover algoritmus nagy előnye az, hogy N elemszámú adatbázis esetén a jelzett elem megtalálásához szükséges lépések száma N négyzetgyökével skálázódik.

* + 1. **A Grover algoritmus használata az RSA kód feltörésér**

Ahogy abban az RSA-ról szóló fejezetben is szó volt, a kódolás alapját azt képezi, hogy két nagy prímszám szorzatát kiszámolni egyszerű, azonban egy nagy számot visszafejteni prímtényezőkre „nagyon nehéz”. A kérdés az, hogy tudunk-e ezen a nehézségen könnyíteni a Grover algoritmus segítségével. A válasz az, hogy igen. A feladatunk nem más, mint megkeresni N osztóját egy természetes számokat tartalmazó adatbázisban. Az adatbázis mérete a 2 és közötti számok száma, ugyanis elég eddig keresni, hogy megtaláljuk N osztói közül a kisebbet, ahonnan a nagyobb már triviálisan adódik. Az is egyértelműen látszik, hogy M = 1. Ezekből az iterációk számára a következő kifejezés adódik: , ami azt jelenti, hogy Grover algoritmust használva a számítási idő N negyedik gyökével skálázódik. Létezik a Grover algoritmusnak más felhasználási lehetősége is az RSA feltörésére, azonban annak hatékonysága nem éri az imént vázoltét, ezért csak említést teszünk róla.

* 1. **A Shor algoritmus**

Ahogy azt a korábbiakban tárgyaltuk, jelenleg nem ismerünk olyan klasszikus algoritmust, mely a számok prímtényezőkre bontását polinomiális időben el tudná végezni. Peter Shor amerikai matematikus 1994-ben feltalált egy olyan kvantumalgoritmust, mely rendelkezik ezzel a kívánatos tulajdonsággal. Ez az algoritmus az ő nevét viselő Shor algoritmus, mely egy klasszikus és egy kvantumos részből áll.

* + 1. **A Shor algoritmus – működés**

A Shor algoritmus működése egy klasszikus lépéssel kezdődik, mégpedig azzal, hogy véletlenszerűen kiválasztunk egy **a** számot, mely kisebb, mint a faktorizálandó N számunk. Ebben a lépésben azt vizsgáljuk meg, hogy **a** és N relatív prímek-e. Amennyiben ezek nem relatív prímek, úgy **a** N egyik faktora, és az algoritmus véget ér. Ha ez nem így van, azaz **a** és N legnagyobb közös osztója 1 (azaz relatív prímek), akkor az előkészítési fázis véget ért, áttérhetünk az algoritmus kvantumos részére. A Shor algoritmus lényegi részének alapját a rend keresése adja. Ebben a kontextusban a rend egy számelméleti fogalom, melyet úgy definiálhatunk, hogy az a legkisebb pozitív természetes **r** szám, melyre teljesül az egyenlet (x és N pozitív egész és x < N). Ekkor azt mondjuk, hogy x rendje **r** modulo N értelemben. A Shor algoritmus ennek a rendnek a megtalálásához a Kvantum Fourier transzformációt hívja segítségül.

* 1. **A Shor algoritmus használata az RSA kód feltörésére**

A Shor algoritmus még a Grover algoritmusénál is hatékonyabb az RSA kód feltörésének tekintetében.