NYILATKOZAT

Név: Führinger Dóra
ELTE Természettudományi Kar, szak: Matematika Bsc
NEPTUN azonosító: H07PLB
Szakdolgozat címe: Az RSA kriptográfiai algoritmus kulcsgenerálási hibái gyakorlati alkalmazásokban

A **szakdolgozat** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2022.05.30.

Führ Dóra

a hallgató aláírása

Eötvös Loránd Tudományegyetem
Természettudományi Kar

Az RSA kriptográfiai algoritmus kulcsgenerálási hibái gyakorlati alkalmazásokban

Führinger Dóra

Szakdolgozat
Matematika Bsc, Alkalmazott matematikus szakirány

Témavezető: Szabó István
Eötvös Loránd Tudományegyetem, Természettudományi Kar
Valószínűségelméleti és Statisztika Tanszék

Budapest, 2022

Köszönetnyilvánítás

Szeretném megköszönni a témavezetőmnek, Szabó Istvánnak, aki segített számomra megfelelő témát találni, és tanácsaival segített a szagdolgozatom megírásában.

Valamint köszönet mindazoknak, akik mellettem álltak az egyetemi éveim alatt.

Tartalomjegyzék

1. fejezet

Bevezetés

Az RSA algoritmus, amelyet 1977-ben publikált Ron Rivest, Adi Shamir és Leonard Adleman, a mai napig az egyik leggyakrabban és legszélesebb körben alkalmazott nyilvános kulcsú titkosítási algoritmus. Biztonságossága azon alapul, hogy a faktorizációs problémára (egy egész szám prímtényezőkre bontására) nem ismert polinomiális idejű algoritmus. Vagyis ha két kellően nagy és jól generált prímszámot összeszorzunk, egy másik személy a kapott eredmény ismeretében sem tudja könnyen kiszámolni a két eredeti prímszámot.

Gyakran használják többek közt hitelesítési eljárásokhoz, digitális aláírásokhoz vagy biztonságos internetes kommunikációt lehetővé tevő protokollokhoz. Ezekben az alkalmazásokban rendkívül fontos a megfelően biztonságos titkosítás, ugyanis ennek hinyában például bárki, aki le tudja hallgatni a vonalon a titkosított forgalmat, hozzáférhetne egy internetes banki átutalásnál használt adatainkhoz, vagy eljárhatna a nevünkben az e-személyi igazolványunk segítségével.

Az RSA algoritmus egyik fontos eleme a kulcsgenerálás. Ennek alapja általában véletlen prímek generálása, azonban a kapott számoknak bizonyos követelményeket teljesíteniük kell. Az RSA használatához szükséges paraméterekre vonatkozó követelményeket, és a megfelelően biztonságos véletlenszám generálásra vonatkozó szabályokat részletes szabványok rögzítik.

Dolgozatom célja, hogy bemutassam, milyen biztonsági kockázatokhoz vezethet, ha a gyakorlati alkalmazásokban nem megfelelően végzik a kulcsgenerálást. Számos kutatás készült ebben a témában, főleg az utóbbi évtizedben. Dolgozatomban igyekszem áttekinteni az ezek által feltárt legfontosabb kulcsgenerálási hibák matematikai hátterét, illetve azt, hogy ezek milyen problémákhoz vezethetnek az egyes gyakorlati alkalmazásokban, miért teszik törhetővé az algoritmust. Röviden kitérek arra is, hogy az említett alkalmazások miben tértek el a szabványokban rögzített ajánlásoktól.

2. fejezet

Az RSA algoritmus

2.1. Kulcsgenerálás

Definíció. a, b egész számok relatív prímek, ha a legnagyobb közös osztójuk 1, vagyis $(a, b) = 1$.

Definíció. az Euler-féle $\varphi(n)$ függvény egy pozitív egész n-re az n-nél kisebb, n-hez relatív prím pozitív egész számok darabszáma.

Algoritmus. Legyen p, q két megefelelően nagy és jól generált prím.
Számoljuk ki $N = p * q$-t
Számoljuk ki $\varphi(n) = (p-1)(q-1)$-t
Legyen e olyan, hogy $1 < e < \varphi(N)$, valamint $\varphi(N)$ és e relatív prím
Számoljuk ki $d \equiv e^{-1} mod(\varphi(N))$-t

Definíció. A nyilvános kulcs az (N, e) számpár, a titkos kulcs pedig a d szám. A fogadó természetesen ismeri p-t és q-t is, és ezeket is titokban tartja.

2.2. Titkosítás és dekódolás

2.2.1. Titkosítás

Tegyük fel, hogy Bob az m üzenetet ($m \in \mathbb{Z}, 0 \leq m < N$) szeretné küldeni Alice-nak. Ehhez Bobnak ismernie kell Alice nyilvános kulcsát: (N, e).
Bob kiszámolja a titkosított üzenetet: $c \equiv m^e (mod N)$, majd elküldi Alice-nek.

2.2.2. Megjegyzések:

Ha az üzenet N-nél hosszabb, először fel kell darabolni kisebb részekre, és azokat külön titkosítani.

A gyakorlati alkalmazásokban általában elvégeznek az üzenetre a titkosítás előtt egy töltés (padding) nevű műveletet. Ez randomizált biteket ad hozzá az üzenethez, ezáltal nehezebben feltörhetővé teszi azt.

2.2.3. Dekódolás

Alice a titkos kulcsát használva dekódolhatja a titkosított üzenetet: $c^d \equiv (m^e)^d \equiv m \pmod{N}$

3. fejezet

A Coppersmith-támadás visszatérése

Ebben a fejezetben a szakirodalomban 'ROCA' (a 'Return of Coppersmit's attack' kezdőbetűiből) néven elhíresült kulcsgenerálási hibát szeretném bemutatni. A problémát, és a módszert, amellyel az érintett kulcsok feltörhetőek, Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, Vashek Matyas: The Return of Coppersmith's Attack[3] című, 2017-es cikkükben írták le először. A fejezet nagyrészt ez alapján íródott. A probléma oka, hogy az érintett szoftverkönyvtár (RSALib) által konstruált prímek egy felismerhető séma szerint készültek, ezáltal jelentősen csökken az entrópájuk a teljesen véletlenszerű generáláshoz képest.

3.1. Prímek ujjlenyomata

Definíció. $P_n\# = \prod_{i=1}^{n} P_i$, vagyis a prímek számának szorzata az n-edik prímig.

A vizsgált prímek szerkezete: $p \equiv k * M + 65537^a \pmod{M}$, ahol $k, a \in \mathbb{Z}$ és $M = P_n\#$. Ha p, q ilyen szerkezetű, akkor $N = p * q = (k * M + 655333^a \pmod{M})(l * M + 65537^b \pmod{M})$, ahol $a, b, k, l \in \mathbb{Z}$, vagyis $N \equiv 65537^{a+b} \equiv 65537^c \pmod{M}$.

Definíció. $\mathbb{Z}_M^*$: az M-nél kisebb, M-hez relatív prím számok multíplikatív csoportja.

Definíció. Egy csoportelem rendje (ord) az a legkisebb, nem nulla kitevő, amire emelve az elemet 1-et kapunk.

Állítás. Létezik c, amire $c = log_{65537}N \pmod{M}$.

Ez a c szám, amit a Pohlig-Hellman algoritmus segítségével tudunk hatákonyan kiszámolni, fog a fent leírt szerkezetű RSA kulcsok ujjlenyomatául szolgálni, amely könnyen azonosíthatóvá teszi őket. A gyakorlati alkalmazások ismertetésénél ezeket "ujjlenyomattal ellátott kulcs" néven említem majd. Elméletileg lehetséges olyan N, amelyet nem ilyen módon állítottak elő, mégis létezik a hozzá megfelelő c, azonban ennek valószínűsége rendkívül kicsi, és a gyakorlatban tesztelt
kulcsok közt ilyen egyáltalán nem fordult elő.

3.2. A faktorizálás módszere

Definíció. Egy $L = \{\sum_{i=0}^{n} a_i \cdot v_i : a_i \in \mathbb{Z}\}$ $\mathbb{R}^n$-beli halmazt, ahol $(b_1, ..., b_n)$ az $\mathbb{R}^n$ egy bázisa, rácsnak nevezünk.

3.2.1. Coppersmith algoritmusa:[3]

Coppersmith algoritmusát általában olyan esetekben használjuk N faktorizálására, ahol a nyilvános kulcs (vagy az üzenet) első bitjeit ismerjük. Elég $log_2(\frac{N}{4})$ bitet ismerni. Ehhez először a problémát egy $f(x) \equiv 0 \pmod{p}$ alakú moduláris egyenletként írjuk fel, ahol $p \mid N$, p ismeretlen (de N-et ismerjük). Ennek szeretnénk megkapni egy x_0 megoldását, amelyre $|x_0| < X$ egy megfelelő konstansra. A moduláris egyenletet egy $g(x) = 0$ alakú, egészek feletti egyenletté alakítjuk, melynek ugyanazok a gyökei, mint az eredetinek, ez teljesül, ha $|g(x_0)| < p$. Legyen $g(x) = \sum_i a_i * f_i(x), a_i \in \mathbb{Z}$. Az f_i-ket úgy választjuk, hogy mod p ugynazok legyenek a gyökeik, mint $f(x)$-nek. A $g(x)$-et a Lenstra-Lenstra-Lovász (LLL) algoritmus segítségével állítjuk elő. Ez az algoritmus egy rács $(b_1, ..., b_n)$ bázisából állít elő egy kisebb vektorokból álló $(b'_1, ..., b'_n)$ bázist. Legyen ezek közül a legkisebb $b'_0 = [b'_{0,0}, b'_{0,1}, ..., b'_{0,n-1}]$, ekkor $g(x) = \sum_{i=0}^{n-1} b'_{0,i} x_0^i$. Ezután már csak $g(x)$ egészek feletti polinom gyökeit kell megkeresnünk.

3.2.2. Naív algoritmus:[3]

Ha meg akarjuk találni a fent leírt formájú N kulcs egyik prímtényezőjét (legyen ez p), akkor meg kell találnuk a-t és k-t, amelyek segítségével előaállítottuk p-t. Naív megközelítéssel kiszámolnánk 65537^a mod M-t az összes lehetséges a-ra, majd a Coppersmith-algoritmus segítségével megkeresnénk k-t. Ekkor az algoritmus futási idejét elsősorban az határozza meg, hogy hány lehetséges a-értéket kell kipróbálnunk, bár természetesen a Coppersmith-algoritmus futási idejétől is függ. A potenciális a értékek száma épp a 65537 rendje $\mathbb{Z}_M^*$-ben $(ord_M(65537))$. Ez sajnos már kis méretű kulcsokra is túl sok időt vesz igénybe, így nem elég hatékony a gyakorlatban.

3.2.3. Hatékonyabb módszer:[3]

A módszer ötletét az adja, hogy a prímek előállításánál használt M mérete megegyezik a Coppersmith-algoritmusban ismert bitek számával. M mérete azonban nagyobb, mint az ismert bitek számának minimuma. Konstruáljunk ezért egy új, kisebb M'-t, amelyre a Coppresmith-algoritmus még mindig működik, vagyis $log_2(M') > log_2(\frac{N}{4})$. Valamint legyen M' osztója M-nek. Ekkor $ord'_M(65537)$ is kisebb lesz, de a p prím továbbra is a fentebb leírt alakú (a megfelelő a'-vel és k'-vel). Ilyen alakú p, q prímekre természetesen N is megfelel a fentebb leírt szerkezetnek, a megfelelő a', b', c', k', l' értékekre. A cél, hogy a faktorizálást a lehető legrövidebb idő alatt tudjuk elvégezni, ehhez az M' és a Coppersmith-algoritmus paramétereinek egy optimális kombinációját kell megtalálni.

3.2.4. Howgrave-Graham módszer:[3]

A Howgarve-Graham módszer annyiban tér el az eredeti Coppersmith-algoritmustól, hogy p helyett p^m lesz a modulus. Tehát x_0 gyöke $f_i(x)$-nek modp^m, és olyan $g(x)$ egészek feletti polinomot keresünk, amelyre $|g(x_0)| < p^m$. Legyen $f_i(x) = x^j \cdot N^i \cdot f^{m-i}(i = 0, ..., m-1; j = 0, ..., \delta-1)$, ahol $\delta = deg(f(x))$, ami az itt tárgyalt esetben 1. Ezt fogjuk használni, hogy megtaláljuk egy $p \equiv k \cdot M + 65537^a \pmod{M}$ alakú prímhez a megfelelő k'-t. Ehhez használhatjuk az $f(x) = x \cdot M' + (65537^{a'} \bmod M')$ függvényt.

3.3. A faktorizálás algoritmusa:

3.3.1. Algoritmus (Faktorizálás):[3]

Input: N, M', m, t
Output: p (N egyik prímfaktora)
  $c' := log_{65537} N mod M$
  $ord' := ord_{M'}(65537)$
  for all $a' \in [\frac{c'}{2}, \frac{c'+ord'}{2}]$ **do**
    $f(x) := x + (M'^{-1} \bmod N)(65537^{a'} \bmod M')(\bmod N)$
    $(\beta, X) := (0.5, 2 * \frac{N^\beta}{M'})$
    $k' := Coppersmith(f(x), N, \beta, m, t, X)$
    $p := k' \cdot M' + (65537^{a'} \bmod M')$
    if $N \equiv 0 \pmod{p}$ **then**
      return p
    end if
  end for

3.3.2. Algoritmus (M' előállítása):[3]

Input: $M = P_n\#$, ord' ($ord' \mid ord_M(65537)$)
Output: $M'(M$ legnagyobb osztója, amire $ord'_M(65537 \mid ord'))$
  $M' := M$
  for all $P_i \mid M$, P_i prím **do**
    $ord_{Pi} := ord_{Pi}(65537)$
    $M' := \frac{M'}{P_i}$
  end if
end forM'

3.3.3. Paraméterek:[3]

I: az algoritmus során egyszerre próbáljuk p-t vagy q-t (illetve a hozzájuk tartozó a'-t vagy $b' - t$) megtalálni. Mivel $c' \equiv a' + b' \pmod{ord'}$, ezért az algoritmusban megadott I-ben valamelyik biztosan előfordul.
β: $p < N^\beta$, mivel p, q mérete kisebb, mint N méretének a fele, $\beta := 0.5$
X: felső korlát az x_0 megoldásra.
$ord' = [ord_{P1}, ord_{P2}, ...]$, minden $P_i \mid M$-re $= ord_{M'}(65537)$

Optimalizálás:

1. Megkeressük a lehetséges M'-ket, amelyekre ord' elég kicsi.
2. Minden M'-höz meghatározzuk az optimális m és t paramétereket. Ezekre alkalmazzuk a Howgrave-Graham módszert. A helyes a' paraméterekre alkalmazva kiszámíthatjuk a sikerességi arányt, az összes a'-re alaklmazva pedig a próbálkozások átlagos futási idejét.
3. Így megadhatjuk az optimális M'-t, m-et, t-t és a minimális futási időt.

Az alábbi táblázatban[3] összefoglaljuk, hogy különböző paraméterekre (az eredeti is az optimalizált $M - re$ és M'-re) milyen futási időket kapunk különböző méretű RSA kulcsok esetén. A naív módszer esetén az eredeti M-et, a javított módszernél az új M'-t használjuk, a fentebb leírtaknak megfelelően. Jelöljük a naív módon elvégzett kísérletek számát $\#N$-nel, a javított módszerrel elvégzett kísérletek számát $\#J$-vel. A faktorizálás várható ideje a fele a legrosszabb eset idejének.

Kulcs mérete	M	M mérete	M' mérete	$\#N$	$\#J$	Egy kísérlet ideje	Legrosszabb eset
512b	$P_{39}\# = 167$	$219,19$b	$140,77$b	$2^{61,09}$	$2^{19,20}$	$11,6$ms	$1,93$CPU óra
1024b	$P_{71}\# = 353$	$474,92$b	$285,19$b	$2^{133,73}$	$2^{29,04}$	$15,2$ms	$97,1$CPU óra
2048b	$P_{126}\# = 701$	$970,96$b	$552,50$b	$2^{254,78}$	$2^{34,29}$	212ms	$140,8$ CPU év
3072b	$P_{126}\# = 701$	$970,96$b	$783,62$b	$2^{254,78}$	$2^{99,29}$	1159 sec	$2,84 * 10^{25}$ év
4096b	$P_{225}\# = 1427$	$1962,42$b	$1098,42$b	$2^{433,69}$	$2^{55,05}$	1086ms	$1,28 * 10^{9}$ év

3.4. Alkalmazások:

3.4.1. Hatás felmérése:

Több tényezőtől is függ, hogy egy potenciális támadó mennyi befektetéssel, és milyen típusú kárt tud okozni az egyes alkalmazásoknál. Ezek a következő szempontok alapján osztályozhatók[3]:

1. Mennyire könnyen hozzáférhetőek a nyilvános kulcsok? Egyes alkaklmazásoknál ezek könnyen és nagy számban elérhetőek, itt a fentebb leírt "ujjlenyomatkészítés" segítségével könnyen beazonosíthatóak az érintett kulcsok. Más esetekben már az is nehéz, hogy a támadó hozzáférjen az esetlegesen faktorizálható kulcsokhoz.
2. Hány faktorizálható kulcsot sikerül találni? Néhány alkalmazás esetében csak egy minimális százaléka érintett, még más esetekben ez a szám jóval nagyobb.
3. A faktorizáláshoz szükséges futási idő és költség. Ez függ a kulcsok hosszától, azonabn nem minden esetben igaz, hogy minnél hosszabb a kulcs, annál nehezebben törhető.

4. A sikeres faktorizációval milyen károkat tud okozni a támadó?

3.4.2. Észt személyi igazolványok[3][5]:

Bár az elektronikus igazolványok kulcsai általában nem hozzáférhetőek a nyilvánosság számára, az elérhető adatok alapján az Észtország által kibocsátott személyazonosító okmányok több típusa is jelentősen érintett volt a 'ROCA' problémában. A vizsgált igazolványok közül az úgynevezett e-residency igazolványok voltak a legnagyobb mértékben érintettek. Észtország volt az első ország, amely kibocsátott ilyen típusú iratokat, melyek lehetővé teszik, hogy bárki a világ bármely pontjáról alapíthatson és működtethessen vállalkozást az EU határain belül. Az ehhez szükséges digitális ID kártyák kulcsai közül az összes vizsgált példány sérülékenynek bizonyult. Az észt állampolgároknak kiadott elektronikus személyi igazolványokat is vizsgálták, itt a minta több, mint fele volt érintett a problémában. Az igazolványoknál 2048 bites RSA kulcsokat használtak, ami azt jelenti, hogy az ujjlenyomattal elátott kulcsok faktorizálhatóak a gyakorlatban. Az észt személyazonosító okmányokkal kapcsolatban más biztonsági problémák is felmerültek, ezekről még később lesz szó.

3.4.3. Kódaláírás (GitHub)[3]:

A kódaláírás célja, hogy biztosítsa egy program, alkalmazás vagy egyéb kód hitelességét. Verziókezelő rendszerek esetén (pl. GitHub) használható a verziók és a címkék aláírására. A GitHub esetén használható SSH (Secure Shell Protocoll) a felhasználónévvel és jelszóval történő azonosítás helyett. A fejlesztők által használt SSH kulcsok elérhetőek, és a 4,7 millió kulcsból 447 bizonyult ujjlenyomattal ellátottnak. Sőt, ezek közül 237 db 2048 bites RSA kulcs, amelyek így faktorizálhatóak.

3.4.4. TPM[3]:

A TPM (Trusted Platform Module, platformmegbízhatósági modul) jellemzően egy különálló chip az alaplapon, amely kriptográfiai funkciókat lát el. A TPM nyilvános kulccsal szólítható meg, ami ezután a védett privát kulccsal ellenőriz, és ha megfelelő a megszólítás, adja vissza a kért információt. Például a Bitlocker hazsnálata biztonságosabbá tehető, ha a szükséges kulcsokat a TPM-ben tároljuk. Nagyobb rendszerek esetén az is különösen fontos alkalmazás lehet, hogy csak jogosultan telepített alkalmazások futtatását engedélyezzük (támadó programokét ne). A kutatásban 41 különböző típusú laptopot vizsgálatak, ezek közül 10 bizonyult érintettnek a problémában. Az érintett kulcsok 2048 bitesek, vagyis törhetőek a fent leírt módszerrel.

3.4.5. PGP[3]:

A PGP (Pretty Good Privacy) titkosítási és hitelesítési célokra egyaránt használt program. Gyakran használják például e-mailen történő kommunikáció titkosítására. A program által használt nyilvános kulcsok nagy része az interneten hozzáférhető, így könnyen beazonosíthatók az érintett kulcsok. A vizsgált kulcsoknak csak egy rendkívül alacsony százaléka, 2892 db, bizonyult ujjlenyomattal ellátottnak. Ezek között összesen 956 db 1024 és 2048 bites kulcs fordult elő, ezek a gyakorlatban

faktorizálhatóak a fent leírt módszerrel. A kulcsok feltörése hitelesítésnél hamisításra ad lehetőséget, titkosított kommunikációnál pedig illetéktelenek férhetnek hozzá titkosnak szánt üzenetváltásokhoz.

4. fejezet

Lehetséges üzenetek visszafejtése

Az RSA kulcsgenerálás egyik legalapvetőbb feltétele, hogy a publikus kitevő (e) relatív prím legyen $(p-1)\cdot(q-1)$-hez, vagyis $\varphi(n)$-hez. De vajon mi történik, ha ez a feltétel nem teljesül? Ekkor a titkos kitevő (d) nem lesz egyértelműen meghatározható, így a dekódolási algoritmust nem tudjuk elvégezni. Azonban a titkosítási algoritmushoz nincs szükség a titkos kulcsunkra, tehát azt végre tudjuk hajtani. Felmerülhet a kérdés, hogy mi történik, ha egy ilyen módon hibásan generált nyilvános kulcsot használunk az üzenet titkosításához? A probléma látszólag teljesen elméleti jellegű, azonban látni fogjuk, hogy gyakorlati implementációban is előfordult.

4.1. Hibás titkosítás [4]

Tétel. (Lagrange tétele) Ha G véges csoprt, akkor minden részcsoportjának rendje (vagyis a részcsoport elemszáma) osztója G rendjének.

Vizsgáljuk meg, mi történik a titkosítás során, ha e nem relatív prím $\varphi(n)$-hez. Felthetejük, hogy e prím (a legtöbb gyakorlati alkalmazásban így van), illetve $\varphi(n)$ nem osztható e-nek semmilyen 1-nél nagyobb kitevőjű hatványával. Legyen E azon elemek részcsoportja $\mathbb{Z}_M^*$-ben, amelyek rendje e. Ekkor $\mathbb{Z}_M^* \cong G * E$, mivel $\forall x \in \mathbb{Z}_M^* : x = g * \ell, g \in G, \ell \in E$. Lagrange-tétele miatt $|E| = e$ és $|G| = (p-1)\cdot(q-1)/e$. Legyen x az eredeti, y a titkosított üzenetünk, ekkor y:-t következőképp számolhatjuk:

$$y = x^e \equiv (g\cdot\ell)^e \equiv g^e \cdot \ell^e \equiv g^e \bmod N$$

Látható, hogy ℓ értéke "elveszik" a titkosítás során, így a lehetséges ősképek halmaza: $P = \{g * \ell | \ell \in E\}$, ahol P rendje e. Természetesen ha nem ismerjük e értékét, akkor innen még nem tudjuk meghatározni a P halmazt. Azonban e értéke a legtöbb gyakorlati alkalmazásban 65537, így van remény a P halmaz meghatározására.

Mekkora az esély, hogy ez előfordul? Tehát mekkora a valószínűsége, hogy e osztója $(p-1)$-nek, vagy $(q-1)$-nek, de nem mindkettőnek. Feltéve, hogy a számokat teljesen véletlenszerűen generáljuk, annak az esélye, hogy egy szám osztható e-vel $\frac{1}{e}$, annak a valószínűsége, hogy nem osztható e-vel $(1-\frac{1}{e})$. Tehát annak a valószínűsége, hogy $(p-1)$ osztható e-vel, de $(q-1)$ nem, $\frac{1}{e}\cdot(1-\frac{1}{e})$. Ez ugyanígy igaz $(q-1)$-re is, tehát összeségében $\frac{2}{e}\cdot(1-\frac{1}{e})$ a valószínűsége, hogy $(p-1)$ és $(q-1)$ közül pontosan az egyik osztható e-vel. Ez $e = 65537$ esetén körülbelül $1:32000$-hez. Annak a valószínűsége, hogy $(p-1)$ és $(q-1)$ közül pontosan az egyik osztható e^2-tel akkor lehetséges, ha $(p-1)$ és $(q-1)$ is osztható e-vel, vagy pontosan az egyik osztható

e^2-tel. Ez hasonló számolás alapján: $\frac{1}{e^2}(1-\frac{1}{e})^2 + \frac{2}{e^2}\cdot(1-\frac{1}{e})$. Ennek valószínűsége $e = 65537$ esetén nagyjából $1 : 1,43 \cdot 10^9$-hez.

4.2. Algoritmus a lehetséges üzenetek visszafejtésére[4]

Tétel. (Kínai maradék tétel) Legyenek $n_1, ..., n_k$ 1-nél nagyobb egészek, és jelölje N az n_i-k szorzatát ($i \in \{1, 2, ..., k\}$) Ha n_i páronként relatív prímek, az $x \bmod N \mapsto (x \bmod n_1, x \bmod n_k)$ leképezés egy gyűrűizomorfizmus a modulo N maradékosztályok gyűrűje és a modulo n_i marasékosztályok gyűrűinek direkt szorzata közt:

$$\mathbb{Z}/N\mathbb{Z} \cong \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$$

A feldatunk tehát az, hogy megtaláljuk E-t, amely a lehetséges eredeti üzenetek halmaza lesz. Mivel e prím és $e^2 \nmid \varphi(n)$, ezért E elemszáma e, és E ciklikus lesz. Feltehető (szimmetria miatt), hogy $e \mid p-1$, ekkor a $\mathbb{Z}_M^*$ generátorához először egy olyan g-t keresünk, amire $g^{\frac{p-1}{e}} \not\equiv 1 \pmod p$. Ez a g primitív gyök lesz $\bmod p$ és $\tilde{g} = g^{\frac{p-1}{e}} \pmod p$ generálja $\mathbb{Z}_M^*$ e rendű részcsoprtját. Ezután a kínai maradéktételben szereplő izomorfizmus segítségével megtalálhatjukn g_E-t, ami az E részcsoport generátora lesz.

A gyakorlatban ennél egyszerűbb eljárást alkalmazunk: kiválasztjuk a lehetséges g értékeket, és teszteljük, hogy melyikre lesz $g_E = g^{\frac{p-1}{e}} \not\equiv 1 \pmod N$. Erre szolgál az alábbi algoritmus[4]:

Algoritmus. $\tilde{\varphi} := (p-1)(q-1)/e$
  $g := 1$
  repeat
    $g := g + 1$
    $g_E := g^{\tilde{\varphi}} \bmod N$
  until $g_E \neq 1$
  return g_E

Ezután végignézzük a lehetséges $\ell \in E$ értékeket[4]:

Algoritmus. $\tilde{\varphi} := (p-1)(q-1/e)$
  $d := e^{-1} \bmod \tilde{\varphi}$
  $a := c^d \bmod N$
  Algoritmus1-gyel megkeressük g_E-t
  $P :=$
  $\ell := 1 \bmod N$
  for all $i = 0 \ldots e-1$ **do**
    $x := a \cdot \ell \bmod N$
    if x egy helyesen paddingelt üzenet **then**
      $P := P \cup \{x\}$
    end if
    $\ell := \ell \cdot g_E \bmod N$
  end for
  return a lehetséges üzenetek P halmaza

Könnyen látható, hogy ez a az algoritmus valóban a lehetséges üzeneteket találja meg. Mint korábban láttuk, ha x a titkosítatlan üzenet, akkor $x = a \cdot \ell$, ahol $a \in G$, illetve $\ell \in E$. Ekkor a titkosított üzenet:

$$c = x^e \equiv (a \cdot \ell)^e \equiv a^e \cdot \ell^e \equiv a^e \pmod{N}$$

. Mivel $ed \equiv 1 \pmod{\frac{(p-1)(q-1)}{e}}$, ebből következik, hogy

$$c^d \equiv (a^e)^d \equiv a^{ed} \equiv a \pmod{N}$$

. Itt felhasználtuk, hogy $|G| = \tilde{\varphi} = \frac{(p-1)(q-1)}{e}$. Látható, hogy az algoritmus megtalálja a-t. A ciklus i-edik iterációjában $\ell = g_E^i$, és mivel g_E az E részcsoport generátora, ez a ciklus minden $\ell \in E$ elemet érinteni fog, így meghatározza x értékét.

4.3. Padding eljárások[4]

4.3.1. RSA OAEP

Az RSA OAEP (Optimal Asymmetric Encription Padding) egy modern, és gyakran használt padding algoritmus.

Legyen M az üzenetünk, m ennek a hossza, L egy címke (string, lehet az üres string is), és H egy hash-függvény. Ezen kívül jelöljük n-nel az N RSA algoritmusbeli modulus méretét (bájtban), és h-val a H függvény kimenetének méretét. Továbbá definiálnunk kell egy $MGF(s, k)$ (mask generation function) függvényt, amely az s-ből a H hash-függvény segítségével előállít egy k hosszú stringet. Ahhoz, hogy ez megfelelően működjön, $m \leq n - 2h - 2$ kell hogy legyen. Ezután a következő műveleteket végezzük el az üzeneten:

1. Kitöltjük $n - h - 1$ bájt hosszúra:

   $$D = H(L)||PS||0x01||M$$

   ahol PS egy $n - m - 2h$ hosszú csupa 0 string.

2. Majd egy véletlenszerű h bájt méretű S-et generálunk, majd az MGF függvény és D segítségével előállítjuk D'-t a következőképpen:

   $$D' = MGF(S, n - h - 1) \oplus D$$

   ahol $\oplus$

3. Ezután D' és S segítségével előállítjuk S'-t:

   $$S' = MGF(D', h) \oplus S$$

   A kész titkosítandó üzenet a következő konnkatenáció lesz:

   $$x = 0x00||S'||D'.$$

Ezt csökkenő bájtsorrendű egészként kell értelmezni, és ezután már titkosíthatjuk az RSA algoritmus segítségével.

Mekkora a valószínűsége, hogy az előző részben leírt algoritmusban egy véletlenszerűen előállított n bájt méretű x megfelel ennek az előállításnak? Először is ellenőrizzük, hogy x legnagyobb helyiértékű bájtja $0x00$, ennek a valószínűsége $\frac{1}{256}$. Az MGF függvény visszatérési értéke tekintheő egy véletlenszerű stringnek. Ezután az előbbi lépéseket visszafelé elvégezve kiszámítjuk D-t:

$$D = MGF(MGF(D', h) \oplus S', n - h - 1) \oplus D'.$$

Annak a valószínűsége, hogy ez valóban a padding eljárásnak megfelelő formátumú, megegyezik annak a valószínűsgével, hogy a D első h bájtja azonos $H(L)$-lel. Ez 256^{-h}. D többi része akkor megfelelő, ha $0x00$ bájtokból áll, kivéve az utolsó bájtot, ami $0x01$. Mivel D-nek ez a része $n-2h-1$ bájt hosszú, és 256^{n-2h-1}-féle különböző olyan lehetséges string van, amely k db $0x00$ bájtból majd egy $0x01$ bájtból áll. Ezt összegezve minden $1 \leq k < n-2h-1$ k-ra, azt kapjuk, hogy $\frac{256^{n-2h-1}-1}{256-1}$ lehetséges, a padding eljárásnak megfelelő üzenet lehetséges, az összes, 256^{n-2h-1} lehetőség közül. Így tehát összegezve, annak a valószínűsége, hogy egy véletlenszerű string az OAEP padding eljárásnak megfelelő:

$$\frac{1}{256} \cdot \frac{1}{256^h} \frac{256^{n-2h-1} - 1}{255 \cdot 256^{n-2h-1n}} = \frac{256^{n-2h-1} - 1}{256^{h+1} \cdot 255 \cdot 256^{n-2h-1}} \approx \frac{1}{256^{h+1} \cdot 255}$$

Ezáltal egy e elemű halmazból annak a valószínűsége, hogy fals pozitív megoldást találjuk (tehát olyat, amely ez alapján az ellenőrzés alapján megfelel a paddineg eljárásnak, pedig valójában nem): $\frac{e}{256^{h+2}}$. Ez $e = 65537$ esetén nagyjából 256^{-h}, ami elhanyagolhatóan kicsi. Tehát a lehetséges üzeneteket kereső algoritmusról feltehetjük, hogy pontosan egy helyes megoldást fog találni.

4.3.2. PKCSI v1.5

A PKCSI v1.5 egy régebb óta használt, az OAEP-nél egyszerűbb, de bizonyos támadásokkal szemben sérülékenyebb padding eljárás.

Legyen ismét M egy m bájt méretű üzenet, n bájt a nyilvános RSA kulcs mérete, és legyen $m \leq n-11$. Ezután generáljunk egy $n-m-3$ hosszú stringet, véletlenszerű nemnulla bájtokból, jelölje ezt PS. Látható, hogy ekkor PS mérete legakább 8 bájt. Ezután a következőképp állítjuk elő a titkosítandó üzenetet:

$$x = 0x00||0x02||PS||0x00||M.$$

Ezután ezt az OAEP-nél látottakhoz hasonlóan csökkenő bájtsorrendű egészként értelmezve titkosíthatjuk. Hogyan szűrhetjük ki, hogy egy string ezzel az eljárással lett-e előállítva? Először is ellenőrizzük, hogy a két legnagyobb helyiértéken álló bájt $0x00||0x02$, annak a valószínűsége, hogy egy véletlenszerű string nem ilyen, 256^{-2}. A másik tulajdonság, amit ellenőrizhetünk, az, hogy a első két bájtot követő rész (PS) első 8 bájtja közül valamelyik nulla-e? 256^7 olyan lehetőség van, ahol az első 8 bájt közül egy adott értéke 0, így összesen $8 \cdot 256^7 = 2^59$ lehetőség van az első 8 bájtra, ami nem felel meg a padding eljárásnak. Tehát $2^{64} - 2^{59} = 2^{59}(2^5 - 1)$ helyes lehetőség van az első 8 bájtra. Vagyis annak a valószínűsége, hogy PS-nek megfelelő rész első 8 bájtja helyes:

$$\frac{2^{59}(2^5 - 1)}{2^{64}} = \frac{2^5 - 1}{2^5} = \frac{31}{32}$$

A két feltétel valószínűségeit összeszorozva:

$$\frac{1}{2^16} \cdot \frac{31}{32} = \frac{31}{2^{21}}$$

Tehát várhatóan $\frac{31e}{2^21}$ db lehetséges üzenet lesz megfelelő, ez $e = 65537$ esetén nagyjából $0,968$. Tehát a lehetséges üzenetek keresésekor általában lesz legalább egy fals pozitív, azonban a kevés eset miatt ezek már egyenként is ellenőrizhetők.

4.4. Windows 10 [4]

A hiba vizsgálatának eredeti oka, hogy a Windows 10 egy előzetesen kiadott verziójában a CNG (Cripthography API:Next Generation) ilyen, hibás RSA kulcsokat generált. A hiba nyilvánosságra kerülése után egyértelművé vált, hogy ez komoly problémákat okozhat a felhasználóknak. A fent leírt algoritmus segítségével azonban visszaszerezhetőek a hibás titkosítás miatt elveszett adatok. A Windows 10 érintett, 1803-as számú verziója 2021 májusáig volt támogatott, és a következő, 1903-as verzióban már nincs jelen a probléma.

5. fejezet

Modulusok közös prímtényezővel

Az eddig vizsgált problémáknál közös volt, hogy a hibás kulcsok önmagukban is törhetőek voltak a leírt módszerekkel, nem volt szükséges egyszerre több kulcsot vizsgálni. Ebben a fejezetben azt fogjuk megvizsgálni, milyen problémákhoz vezet, ha az RSA kulcsok generálásakor két különböző modulusnak (N-nek) van egy közös prímtényezője. Ha két modulusról tudnánk, hogy az egyik prímtényezőjük közös, akkor természetesen könnyen törhetővé válnának, hiszen ekkor az RSA biztonságát garantáló faktorizációs probléma leegyszerűsödik a legnagyobb közös osztó megtalálására. Ez az euklideszi algoritmus segítségével hatékonyan számolható nagy számokra is. Az első, naív ötletünk lehetne, hogy ha meg akarjuk vizsgálni, hogy az RSA kulcsaink közt vannak-e olyanok, amelyek rendelkeznek közös prímtényezővel, egyszerűen számoljuk ki minden lehetséges kulcspár legnagyobb közös osztóját. Ha n db kulcsunk van, az $\binom{n}{2}$ lehetséges kulcspárt jelent. Az összes lehetséges pár legnagyobb közös osztójának kiszámítása már nem lesz gyorsan elvégezhető az euklideszi algoritmussal. Az alábbiakban megnézzük, hogy oldható meg ez a probléma hatékonyabban, illetve hogy a gyakorlatban milyen alkalmazásokat érint ez a hiba.

5.1. Közös legnagyobb közös osztó[2]

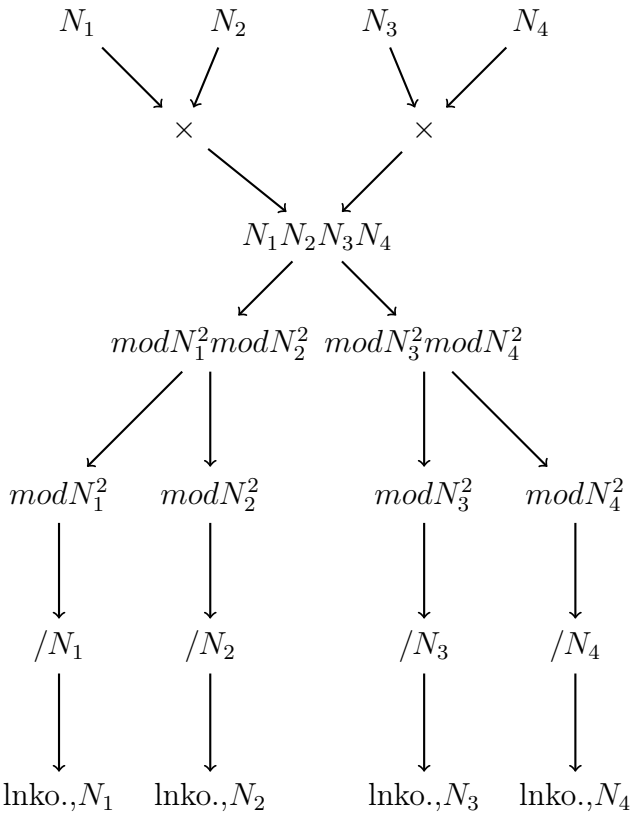
Az következő algoritmus segítségével hatékonyan megvizsgálható, hogy adott számok (esetünkben RSA modulusok) között vannak-e olyanok, amelyek nem relatív prímek, vagyis a legnagyobb közös osztójuk nem 1:

Input: $N_1, ..., N_m$ RSA modulusok

1. $P = \prod N_i$ kiszámítása
2. $z_i = (PmodN_i^2)$ minden i-re

Output: $lnko(N_i, \frac{z_i}{N_i})$

Az algoritmus részletesebben áttekinthető, ha a lépéseket egy fán ábrázoljuk, $m = 4$ estén.



Nézzük meg egy példán keresztül, hogyan működik az algoritmus a gyakorlatban:

Legyen $N_1 = 65$, $N_2 = 14$, $N_3 = 51$, és $N_4 = 22$. Ekkor $N_1N_2N_3N_4 = 1021020$. Ezután a fenti fán ábrázolt módon kiszámoljuk az algoritmusban leírt z_i értékeket. A fának ezt a részét maradékfának nevezzük. A maradékfa első szintje:

$$1021020 \equiv 192920 \pmod{828100}$$

$$1021020 \equiv 1021020 \pmod{128884}$$

Második szintje:

$$192920 \equiv 2795 \pmod{4225}$$

$$192920 \equiv 56 \pmod{196}$$

$$1021020 \equiv 1428 \pmod{2601}$$

$$1021020 \equiv 264 \pmod{484}$$

Ezután a kapott eredményeket rendre leosztjuk a megfelelő N_i értékkel, és vesszük az így kapott szám és N_i legnagyobb közös osztóját:

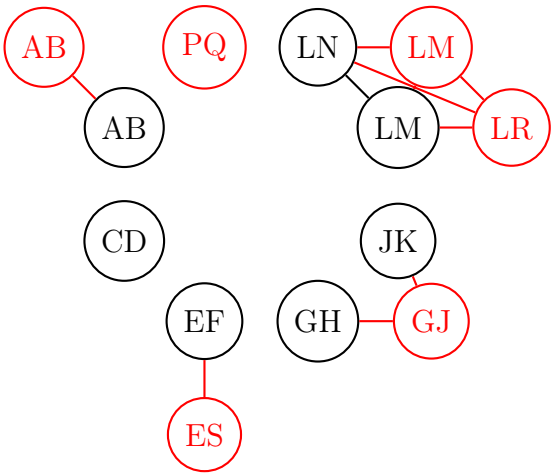
$lnko(43, 65) = 1$, $lnko(4, 14) = 2$, $lnko(28, 57) = 1$, $lnko(12, 22) = 2$.

Így megkaptuk, hogy N_2 és N_4 nem relatív prímek, hiszen a legnagyobb közös osztójuk 2.

5.2. A modulusok ábrázolása gráfon

Ábrázoljuk az RSA modulusokat egy gráfon, amelynek csúcsai a modulusok, és két csúcs pontosan akkor szomszédos, ha a két modulusnak van legalább egy közös prímtényezője. Legyen a kulcsaink halmaza $\{AB, CD, EF, GH, JK, LM, LN\}$,

ahol a különböző betűk különböző prímtényezőket jelölnek, a betűpárok pedig a két prím szorzataként előálló modulust. Ezek közül az utolsó kettő kivételével egyik sem törhető (legalábbis ezzel a módszerrel nem), ezek azonban törhetőek a közös L prímtényezőjük miatt. Tegyük fel, hogy most a következő új kulcsokat generáljuk (az ábrán piros színnel jelölve): $\{PQ, AB, LM, LR, ES, GJ\}$. A PQ kulcs biztonságos lesz. Az AB kulcs két példányban szerepel, így mindkettő törhetővé válik (erről a problémáról részletesebben a következő fejezetben lesz szó). Az új LM is egy meglévő, és törhető kulcs második példánya lesz, így természetesen az is törhető. Az LR kulcs is ugyanebbe a törhető halmazba fog tartozni. Mivel az új ES kulcsnak van közös prímtényezője az EF kulccsal, mindketten törhetővé válnak. Az új GJ kulcs miatt az eddig biztonságosnak számító GH és JK kulcsok is törhetővé válnak.



Egy 2012-es kutatás[1], amely az X.509 szabványnak megfelelő RSA kulcsok biztonságosságosságával foglalkozott, a vizsgált modulusok között számos olyat talált, amelyek egyik prímtényezőjükben megegyeztek. Az eredményeket egy, az előzőekben leírt módon készült gráfon ábrázolták. Ideális esetben c db modulusból egy c komponensből álló gráfot kapnánk, és minden komponens egy-egy független élből állna, vagyis összesen $2c$ db csúcsunk lenne. A valóságban a kapott gráf 1995 db legalább 3 csúcsból álló komponenst tartalmazott, ezek összesen 14901 csúcsból és 12934 levélből álltak. Ezek közül 1988 db egy mélységű fa volt, amelyekben egy gyökércsúcshoz több levél csatlakozott. 1200 ilyen fának 2 db levele volt, 345-nek pedig 3, de előfordult egy 4627 levelet tartalmazó, egy mélységű fa is. Hét komponens ezeknél is összetettebb volt. Hat ezek közül 4 csúcsból és 3 élből állt. A gráf legösszetettebb komponense egy kilenc csúcsú teljes gráf (K_9) volt. Ez azt jelenti, hogy a vizsgált RSA modulusok közül $\binom{9}{2} = 36$ ennek a kilenc prímnek az összes lehetséges páronkénti szorzatából állt elő.

5.3. Alkalmazások

5.3.1. SSH

Az SSH (Secure Shell Protocol) kriptográfiai protokoll egy számítógép távolról való biztonságos elérését biztosítja. Két fő verziója van: az SSH-1 esetében a kliens a szerver publikus kulcsának segítségével készít egy munkafolyamat azonosító kul-

csot; az SSH-2 a munkafolyamat-azonosító kulcs létrehozásához a Diffie-Hellmann kulcs-scere protokollt használja. Egy, már korábban is idézett kutatásban a vizsgált 10216363 SSH alkalmazás közül 2459-nél (vagyis nagyjából a $0,03\%$-nál) fordult elő ez a probléma.

5.3.2. TLS

A TLS (Transport Layer Security), illetve az elődjének tekinthető SSL (Socket Layer Security) kriptogrfiai protokollok, amelyek az internetes böngészés biztonságossá tételét szolgálják. A TLS egyik legismertebb és legszéleskörűbb alkalmazása a HTTPS protokoll, de az internetes kommunikáció számos területén (e-mail, azonnali üzenetküldés,VoIP) is használják. A TLS kézfogás során a kliens a TLS-sel rendelkező szerverhez fordul, és egy listát ad át neki az általa támogatott titkosítási módszerekről és hash-függvényekről. Ezután a szerever kiválasztja ezek közül a legerősebbet, majd elküld a kliensnek egy digitális aláírást, amely a szerver nevéből, egy tanúsítványból, és a szerver nyilvános kulcsából áll. Majd a kliens generál egy véletelen számot, és ezt a szerver nyilvános kulcsával titkosítja, ez lesz a munkafolyamat-azonosító kulcs. Az így létrehozott biztonságos kapcsolaton keresztül ezután a munkafolyamat-azonosító kulcs segítségével,szimmetrikus titkosítással történik a kommunikáció. A kuatásban [2] vizsgált 12828613 TLS alkalmazás közül 64081-nél (az esetek $0,5\%$-ában) találtak azonos prímtényezővel rendelkező kulcsokat.

6. fejezet

Közös kulcsok

Talán az összes eddiginél nyilvánvalóbb hiba az, ha kiderül, hogy két különböző felhasználó ugyanazt az RSA kulcsot használja. Ekkor ezek a kulcsok természetesen nem tekinthetőek biztonságosnak. Ebben a fejezetben azt fogjuk megvizsgálni, hogy ez milyen okokból fordulhat elő, illetve a gyakorlatban milyen veszélyei lehetnek, ha ez a probléma előfordul.

6.1. Lehetséges okok

Annak, hogy két RSA alkalmazás ugynazt a kulcsot használja, több lehetséges oka is lehet. Ezek között előfordul olyan is, amely véleményem szerint könnyen kivédhető lenne. Például számos esetben kiderült, hogy a kulcsok azért egyeznek meg, mert a felhasználók nem változatatták meg a gyári beállítás kulcsát. Az egyik, korábban már idézett kutatás[2] a vizsgált TLS alkalmazások $5,23\%$-át (670391 darabot) azért nem talált biztonságosnak, mert gyári TLS tanúsítványt vagy kulcsot használtak. Szintén gyakori, hogy az egyező kulcsok ugyanannak a szervezetnek a különböző TLS tanúsítványaihoz tartoztak. Egy másik lehetséges oka az egyező kulcsoknak, ha a kulcsgenerálás során nem volt megfelelő az entrópia mértéke. Később még lesz szó a kulcsgeneráláshoz szükséges véletlenszámok előállításának lehetséges módszereiről, és hogy ezek milyen feltételek mellett biztonságosak.

6.2. Észt személyi igazolványok[5]

Ahogy arról már a 3. fejezetben is szó volt, Észtországban többféle elektronikus személyazonosító okmány van használatban. Az e-személyi igazolványok kiadása 2002-ben kezdődött, és máig az egyik legsikeresebb, az állami ügyintézés digitalizációjára irányuló programnak számít. Számos szolgáltatás elérhető ezeknek az igazolványoknak a segítségével online formában, sőt digitiális aláírásokhoz, és választásokon való szavazáshoz is használhatóak.
A személyi igazolványok kétféle aszimmetrikus kulccsal rendelkeznek:

- Azonosító kulcs: az elektronikus ügyintéző rendszerbe történő belépéskor ez használható azonosítóként a TLS protokollhoz. Illetve ezzel lehet a kártya tulajdonosának szánt, titkosított dokumentumokat megnyitni. Mindkét művelethez szükség van még ezen kívül a kártyához tartozó négyjegyű PIN1 kód megadásához is.

- Digitális aláíró kulcs: digitális aláírásokhoz használható, amelyhez az ötjegyű PIN2 kód megadása is szükséges.

A kártya használatához szükséges műveletekhez (például új PIN kódok beállítása, új kulcsok generálása), a kártya szimmetrikus kulcsokkal is rendelkezik, ezeket a gyártó tudja használni.
Egy, az észt e-ID kártyák biztonságosságát vizsgáló kutatás számos egyező RSA kulcsot talált az igazolványok kulcsai közt. Ezek nagy részéről kiderült, hogy ugyanannak a kárytának az azonosító és a digitális aláíró kulcsáról van szó, de előfordultak olyan esetek is, mikor az egyező kulcsok különböző személyekhez tartoztak. A kutatást végző szakembereknek sikerült kapcsolatba lépniük valakivel, akinek a azonosító kulcsa megegyezett egy másik személy digitális aláíró kulcsával. A gyakorlatban kipróbálva kiderült, hogy az ő azonosító kulcsát használva valóban sikerült a másik érintett személy nevében digitális aláírást létrehozni.

6.3. TLS és SSH

Az előző fejezetben is említett, a TLS és az SSH kulcsok biztonságára irányuló átfogó kutatás[2] 12828613 TLS alkalmazást vizsgált meg, ezek közül 7770232 használt legalább egy másik alkalmazással közös kulcsot, amely a vizsgált kulcsok $60,5\%$-a.

Az SSH alkaklmazások közül a 10216363 esetből 6642222-nél derült fény erre a problémára, ez a vizsgált kulcsok 65%-a.

7. fejezet

RSA szabványok

Már az RSA algoritmus eredeti publikációjában is megfogalmaztak a szerzők ajánlásokat a kulcsok megválasztására vonatkozóan. Az RSA használatának elterjedésével pedig egyre részletesebb szabványok jelentek meg, amelyek leírják az algoritmus paramétereire vonatkozó aktuális követelményeket. Ebben a fejezetben néhány ilyen szabványt szeretnék röviden ismertetni, illetve rámutatni, hogy az eddig bemutatott hibás alkalmazások miben tértek el ezektől.

7.1. X.509 [7]

Az 1988-ban, az ITU (International Telecommunication Union, korábbi nevén Consultative Committee for International Telephony and Telegraphy, röviden CCITT) által kiadott szabványban a következő követleményeket fogalmazzák meg p és q választásával kapcsolatban:

- véletlenszerűen kell őket választani
- prímszámokanknek kell lenniük
- $|p - q|$-nek nagynak kell lennie
- $(p + 1)$-nek kell, hogy legyen egy nagy prímtényezője
- $(q + 1)$-nek kell, hogy legyen egy nagy prímtényezője
- $(p - 1)$-nek kell, hogy legyen egy nagy prímtényezője, nevezzük ezt r–nek
- $(q - 1)$-nek kell, hogy legyen egy nagy prímtényezője, nevezzük ezt s-nek
- $(r - 1)$-nek kell, hogy legyen egy nagy prímtényezője
- $(s - 1)$-nek kell, hogy legyen egy nagy prímtényezője

A publikus kitevőre vonatkozó követelmény, hogy $e > log_2(n)$. Ha e értéke állandó, akkor ajánlott az $e = 2^{16} + 1 = 65537$ értéknek választani. Ekkor, mivel e prím, így ahhoz, hogy e és $\varphi(n)$ relatív prím legyen, elég azt ellenőrizni, hogy se $(p - 1)$, se $(q - 1)$ ne legyen osztható e-vel. A 4. fejezetben megvizsgáltuk, milyen következményekkel jár, ha ez a feltétel nem teljesül. Egy, az X.509 szabvány szerinti kulcsokat külön is vizsgáló kutatás[1] talált közöttük néhány, az ajánlottnál kisebb e értéket használó alkalmazást.

7.2. ETSI szabványok

Az ETSI (European Telecommunications Standard Institution) az Európai Bizottság javaslatára létrehozott független, nonprofit szervezet, amely infokommunikációs szabványokat készít. Egyike az Európai Unió által hivatalosan is elismert szabványügyi szervezeteknek.

7.2.1. Követelmények p-re és q-ra

Az első, az RSA kulcsgenerálására vonatkozó részletes szabvány, amit ettől a szervezettől találtam, a 2003-as[8]. Az alábbi követelmények szerepelnek benne p és q váalasztására: p és q legyenek véletlenszerűen és egymástól függetlenül generált prímszámok, amelyek teljesítsék a következő feltételeket:

- $n = p \cdot q$ bithossza ≥ 1020
- p és q közel azonos méretűek: $0,1 < |log_2 p - log_2 q| < 30$
- a prímek halmaza, amelyből p-t és q-t genereáljuk, kellően nagy és egyenletes eloszlású

Egy k bites RSA modulus generálására egy elfogadott algoritmus:

1. $p \in [2^{\frac{k}{2}-15}, 2^{\frac{k}{2}+15}]$
2. $q \in [2^{\frac{k}{2}-15}, 2^{\frac{k}{2}+15}]$
3. ha a $0,5 < |log_2 p - log_2 q| < 30$ feltétel nem teljesül, akkor újra kezdjük az első lépéstől
4. $n = p \cdot q$

A 2014-es ETSI szabványban[10] változás a korábbiakhoz képest, hogy a p-re és q-ra vonatkozó korábbi feltétel helyett elég, ha: $\sqrt{2} \cdot 2^{\frac{nBits}{2}-1} \leq p, q \leq 2^{\frac{nBits}{2}} - 1$ és $|p - q| < 2^{\frac{nBits}{2}} - 100$

7.2.2. A publikus kitevőre vonatkozó feltételek

A 2003-as szabvány[8] szerint válasszunk $e \in \{3, ..., n-1\}$ egész számot, amelyre $lnko(e, lkkt(p-1, q-1)) = 1$.

A 2011-es szabványban[9] változás a fentebb leírtakhoz képest, hogy az e nyilvános kitevő választásakor ajánjott, hogy $e \geq 2^{16} + 1$ legyen, de ha különösen fontos a gyorsaság, akkor kisebb (pl. $e = 3$) is választahtó.

A 2014-es szabvány[10] szerint e értéke: $2^{16} < e < 2^{256}$ közötti páratlan szám kell, hogy legyen.

Egy 2012-ben készült kutatásban[1] az általuk vizsgált RSA alkaklmazások körülbelül $4,5\%$-ában e értéke kisebb volt, mint $2^{16} + 1$, vagyis az ETSI szabányainak nem felelnének meg.

7.2.3. Kulcsok mérete

A szabványok n méretére is tesznek ajánlásokat. ezeket az alábbi tábláazatokban foglaljuk össze. A MinModLen n bithosszát jelöli, az érvényesség pedig azt, hogy a szabvány kiadásától számítva melyik évben mekkora a minimálisan használható kulcsméret.

A 2011-es ETSI szabvány[9] követelményei:

érvényesség	1 év	3 év	6 év
MinModLen	1536	2048	2048

A 2014-es ETSI szabvány[10] követelményei:

érvényesség	1 év	3 év	6 év
MinModLen	1536	2048	3072

A 2019-es ETSI szabvány[11] követelményei:

érvényesség	1 év	3 év	6 év
MinModLen	≥ 1900	≥ 1900	≥ 3000

Az előbb is említett, 2012-es kutatás[1] szerint a vizsgált kulcsok $1,6\%$-a 512 bites modulust használt, $0,8\%$-a pedig 768 bitest, ezek nem nyújtanak megfelelő biztonságot.

7.3. NIST szabványok

Egy másik fontos szabványügyi szervezet az amerikai NIST (National Institution of Standards and Technology), amely számos különböző témában állít ki szabványokat.

7.3.1. Követelmények p-re és q-ra

A NIST 2019-es publikációja szerint p-nek és q-nak a következő követelményeknek kell megfelelnie ($nBits = n = p \cdot q$ bithossza):

- $2^{\frac{nBits-1}{2}} < p < 2^{\frac{nBits}{2}}$
- $2^{\frac{nBits-1}{2}} < q < 2^{\frac{nBits}{2}}$
- $|p - q| > 2^{\frac{nBits}{2}-100}$

7.3.2. Kulcs mérete, biztonsági erősség

A NIST szabványok egy nBits bithosszúságú modulus biztonsági erősségét a következő módon határozzák meg:

$$E(nBits) = \frac{1,923 \cdot \sqrt[3]{nBits \cdot ln(2)} \cdot \sqrt[3]{(ln(nBits \cdot ln(2)))^2} - 4,69}{ln(2)}$$

Mivel ez általában nem egész, a becsült biztonsági erősséget ($ES(nBits)$) hazsnáljuk helyette, amely az $E(nBits)$-hez legközelebb eső nyolccal osztható szám. Az

alábbi táblázat néhány gyakran használt modulushosszt, és a hozzájuk tartozó becsült biztonsági erősséget tartalmazza:

nBits	ES(nBits)
2048	112
3072	128
4096	152
6144	176
8192	200

A NIST 2019-es publikációja[12] legalább 112-es erősségű modulus használatát javasolja.

7.4. Véletlen számok generálása

7.4.1. Trueran[8]

Az egyik lehetséges megközelítése a kulcsgeneráláshoz szükséges véletlen számok előállításának, ha valamilyen fizikai kísérletből származó, valódi véltelen adatokkal dolgozunk. Ezt nevezzük valódi véletlenszám generálásnak (True Random Number Generator, TRNG). Ez lehet egy kvantumfolyamat, például egy radioakítv anyag bomlsának megfigyelése Geiger-Müller-számláló segítségével. Egy másik gyakori megoldás a számítógép használatának megfigyeléséből (például billentyűk leütése, egérkattintás) való véletlenszám sorozat előáálítás. Az így kapott értéken a biztonság növelése érdekében általában még további műveleteket végeznek. Statisztikai próbák segítségével ellenőrizhető, hogy a használt forrás megfelelően véletlenszerű-e. Azonban az önmagában nem elég, ha ezeknek a teszteknek megfelel, hiszen például a π tizedesjegyei például hiába tűnnek véletlenszerűnek, titkosnak nem tekintehtőek.

7.4.2. Pseuran[8]

A másik lehetséges megközlítés, hogy egy valódi véletlen magot (seed) felhasználó determinisztikus algoritmussal (Pseudorandom Number Generator, PRNG) állítunk elő egy pszeudovéletlen bitsorozatot. Egy PRNG akkor tekinthető biztonságosnak, ha:

- Semmilyen információt nem tudunk előre az generált bitekről
- A generált bitek egy részhalmazából nem tudjuk meghatározni a többit, a teljesen véletlenszerű tippelésnél lényegesen nagyobb valószínűséggel
- Nincs módszer, amellyel a generált bitek egy részéből visszafejtehtőek a korábban vagy később generált bitek, illetve nem tudunk ezekből következtetni az algoritmus belső működésére, vagy a magra

A szabványok részletesen leírják, hogy pontosan milyen TRNG és PRNG módszerek használhatóak az RSA kulcsgenrálásához. A nem megfelelő véletlenszám generálás egy gyakori oka a hibás kulcsoknak. Többek között ez vezethet ahhoz,

hogy RSA alkalmazások közös modulust, vagy közös prímtényezővel rendelkező modulusokat használnak. A ROCA hiba esetében pedig láthattuk, hogy ha a prímek egy felismerhető konstrukció alapján készülnek, az törhetővé teheti az algoritmust.

8. fejezet

Összefoglalás

Dolgozatom célja az volt, hogy bemutassam, az RSA algoritmus gyakorlati alkalmazásaiban milyen hibák fordulhatnak elő a kulcsgenerálás során, illetve ezek milyen biztonsági kockázatokhoz vezethetnek. A bemuatatott példákon keresztül látható, hogy hiába tartunk egy algoritmust elméletileg biztonságosnak, a gyakorlati alkalmazásokban elkövetett hibák mégis törhetővé tehetik azt. Természetesen ezeknek a hibáknak csak egy része kapcsolódik a kulcsgeneráláshoz, az RSA számos más módszerrel is támadható, ezek azonban ennek a dolgozatnak a témáján kívül esnek.

Az ismertetett hibás alkalmazások különböző biztonsági kockázatokat rejtenek magukban. Az azonos kulcsokat, vagy közös prímtényezővel rendelkező kulcsokat használó alkalmazások esetében láthattuk, hogy bár önmagában kezelve egy ilyen alkalmazás nem törhető, nyilvánosan hozzáférhető adatokból is elég nagy számú kulcsot lehet összegyűjteni, amelyek közül már törhetővé válnak ezek az esetek. Gyakran ezeket a problémákat a nem megfelelően generált véletlenszámok okozzák. Ha pedig a generált prímek egy felismerhető sémát követnek, akkor ez alapján is támadhatóvá vállhat az alkalmazás, ezt láttuk a ROCA probléma esetében. Egy ehhez hasonló probléma merült fel taiwani e-szemlyi igazolványokkal kapcsolatban is [7].

Véleményem szerint a kulcsgenerálási hibák elkerülésében rendkívül fontos szerepet játszik az előző fejezetben említett szabványok betartása az RSA alkalmazása során. Ezen kívül szinte minden, a dolgozatomban idézett cikk szerzői kiemelték, hogy a cikk megjelenése előtt tájékozatták az érintett szervezeteket az eredményeikről. Úgy vélem, ez a kommunikáció a témával foglalkozó kutatók és az RSA-t a gyakorlatban használók között kiemelten fontos a hibás alkalmazások kiszűrésében, és az ezekből adódó esetleges biztonsági kockázatok elkerülésében.

Források

[1] Arjen K. Lenstra , James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, Christophe Wachter: Ron was wrong, Whit is right (https://eprint.iacr.org/2012/064.pdf)

[2] Nadia Heninger, Zakir Durumeric, Eric Wustrow, J. Alex Halderman: Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices (https://factorable.net/weakkeys12.conference.pdf)

[3] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas: The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli
(https://crocs.fi.muni.cz/_media/public/papers/nemec_roca_ccs17_preprint.pdf)

[4] Daniel Shumow: Incorretly generated RSA keys (https://www.google.com/url?sa=t&source=web&rct=j&url=https://eprint.iacr.org/2020/1059.pdf&ved=2ahUKEwicupjP9PLzAhXIl4sKHVEpDpQQFnoECAMQAQ&usg=AOvVaw1MFzsmNPIowz-UBVUxHlGD)

[5] Arnis Parsovs: Estonian Electronic Identity Card: Security Flaws in Key Management (https://www.usenix.org/system/files/sec20-parsovs.pdf)

[6] Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, Nicko van Someren: Coppersmith in the wild (https://www.google.com/url?sa=t&source=web&rct=j&url=https://www.researchgate.net/publication/265720345_Factoring_RSA_Keys_from_Certified_Smart_Cards_Coppersmith_in_the_Wild&ved=2ahUKEwi38tGrksP1AhVx8LsIHU5RDYQQFnoECAwQAQ&usg=AOvVaw3QMOntcbsGACPMo2WmJAFU)

[7] CCITT X.509-es szabvány:
https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.509-198811-S!!PDF-E&type=items

[8] ETSI 2003-03 szabvány:
https://www.etsi.org/deliver/etsi_sr/002100_002199/002176/01.01.01_60/sr_002176v010101p.pdf

[9] ETSI 2011-07 szabvány:
https://www.etsi.org/deliver/etsi_ts/102100_102199/10217601/02.01.01_60/ts_10217601v020101p.pdf

[10] ETSI 2014-11 szabvány:
https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.01.01_60/ts_119312v010101p.pdf

[11] ETSI 2019-02 szabvány:
https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.03.01_60/ts_119312v010301p.pdf

[12] NIST Special Pulication 800-56B:
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf