Eötvös Loránd Tudományegyetem

Természettudományi Kar

Kenyeres Zoltán

Az elektronikus aláírások kriptográfiai algoritmusai

Szakdolgozat

Matematika BSc, matematikai elemző szakirány

Témavezető:

Szabó István

Valószínűségelméleti és Statisztika Tanszék

Budapest, 2019

Köszönet nyilvánítás

Szeretnék köszönetet nyilvánítani a családomnak és barátaimnak, hogy támogattak a szakdolgozatom megírásában. Köszönettel tartozom a témavezetőmnek, Szabó Istvánnak a sok türelemért, a segítségéért, és a gyors válaszaiért.

Tartalomjegyzék

1. fejezet

Bevezetés

Az aláírások mindennapi életünkben fontos szerepet töltenek be. Az aláírásunkkal nyilatkozatot teszünk arról, hogy például elfogadjuk az adott dokumentumban foglaltakat, vagy igazoljuk, hogy átvettük és elolvastuk a dokumentumot. A kézzel írott aláírást egyedi grafika jellemzi, és ez személyenként eltér, és más személy által nehezen másolható.

Napjainkban egyre elterjedtebbek a papír alapú aláírások mellett az elektronikus aláírások használata. A hitelességében nagyon hasonló, az aláíróra egyedileg jellemző, és elválaszthatatlanul hozzákapcsolódik az aláírt dokumentumhoz, azonban ezeket a követelményeket más eszközökkel valósítja meg. Ha elektronikusan írunk alá egy dokumentumot, akkor nem a beszkennelt aláírásra kell gondolnunk, hanem egy speciális kódolásra, melyet számítógép segítségével hajtunk végre.

Az elektronikus aláírások hitelességét a jogszabályok is elismerik, és okirati bizonyítékként is felhasználnálható. Mindezek alapján a kézzel írott aláíráshoz hasonlóan érvényes a bíróság előtt is, minősített tanúsítványon alapuló fokozott biztonságú aláírással vagy minősített aláírással. A minősített aláírás pedig a kézzel írott aláírással egyenértékű (az EU teljes területén). [1]

A minősített elektronikus aláírás egy olyan, fokozott biztonságú elektronikus aláírás, amelyet minősített elektronikus aláírást létrehozó eszközzel állítottak elő, és amely elektronikus aláírás minősített tanúsítványán alapul. A minősített elektronikus aláírást létrehozó eszközöknek megfelelő technikai és eljárási megoldásoknak kell megfelelnie (pl. adatok bizalmas kezelése, megbízható védelem). Az elektronikus aláírás minősített tanúsítványa pedig olyan, elektronikus igazolás, amely az elektronikus aláírást érvényesítő adatokat egy természetes személyhez kapcsolja, és igazolja legalább az érintett személy nevét vagy álnevét, valamint amelyet minősített bizalmi szolgáltató bocsát ki, és amely tartalmazza a beazonosításhoz szükséges adatokat (pl. aláíró neve vagy álneve, az elektronikus aláírás érvényesítéséhez használt adat, a tanúsítvány érvényességi ideje).

Az elektronikus aláírás joghatása és bírósági eljárásokban bizonyítékként való elfogadhatósága nem tagadható meg kizárólag amiatt, hogy az elektronikus formátumú, illetve nem felel meg a minősített elektronikus aláírásra vonatkozó követelményeknek. A minősített elektronikus aláírás a saját kezű aláírással azonos joghatású. A valamely tagállamban kibocsátott minősített tanúsítványon alapuló minősített elektronikus aláírást az összes többi tagállamban el kell ismerni minősített elektronikus aláírásként.

A fokozott biztonságú elektronikus aláírásnak az alábbi követelményeknek kell megfelelnie:

- kizárólag az aláíróhoz köthető;
- alkalmas az aláíró azonosítására;
- olyan, elektronikus aláírás létrehozásához használt adatok felhasználásával hozzák létre, amelyeket az aláíró nagy megbízhatósággal kizárólag saját maga használhat;
- olyan módon kapcsolódik azokhoz az adatokhoz, amelyeket aláírtak vele, hogy az adatok minden későbbi változása nyomon követhető.

[7]

Az elektronikus aláírás az írott aláírás digitális analógiája. Mint már említettem, a digitális aláírás bizonyítékul szolgál az információ elfogadására, továbbá használható arra, hogy lássuk, hogy a dokumentum volt-e módosítva az aláírás óta. A dolgozatomban leírt algoritmusok megfelelnek ezeknek a követelményeknek.[2] Szakdolgozatom során azoknak az algoritmusoknak a működését írom le, amelyeket a NIST (National Institute of Standards and Technology) által kiadott FIPS 186-4 es titkosítási szabványában biztonságosnak ítéltek. A dolgozatom fő elemeként a DSA algoritmust mutatom be, továbbá rövidebb kitekintést teszek az RSA valamint az Elliptikus görbe algoritmusok működésére terjedelmi korlátok miatt nem térek ki.

2. fejezet

Az aláírás folyamata [2]

Ebben a fejezetben az elektronikus aláírások folyamatát mutatom be. Első lépésként definiálok néhány fontos fogalmat:

Tulajdonos: A kulcspár tulajdonosa az a személy, aki jogosult a kulcspár titkos kulcsának a használatára.

Kulcspár: A nyilvános és a hozzá tartozó titkos kulcs.

Kulcs: Egy paraméter, együtt használatos egy kriptográfiai algoritmussal (ami meghatározza funkcióját). Ebben a dolgozatban a funkciók vagy egy digitális aláírás létrehozása egy üzenethez (titkos kulcs), vagy egy digitális aláírás ellenőrzése (nyilvános kulcs).

Tanúsítvány: Egy adathalmaz, ami egyértelműen azonosítja egy kulcspárt és annak tulajdonosát. A tanúsítvány tartalmazza a tulajdon titkos kulcsát, és esetlegesen más információkat. Egy harmadik személy által van aláírva (Certification Authority, trusted party), így köti össze a titkos kulcsot a tulajdonossal.

Biztonsági erősség: Egy szám, amely azt jelzi mennyi számolás szükséges ahhoz, hogy feltörjünk egy kriptográfiai algoritmust, vagy rendszert.

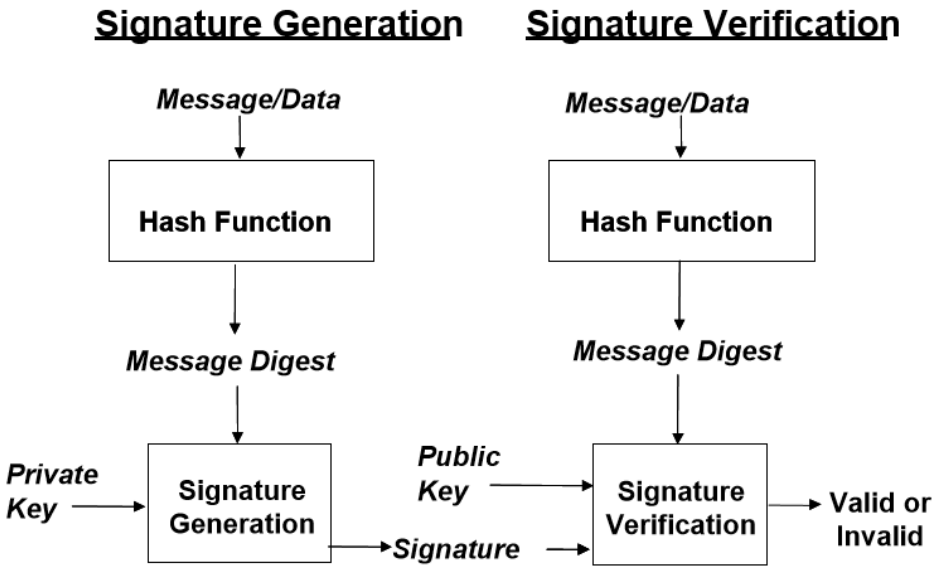
Message digest, Hash: Az eredmény, ha egy üzeneten alkalmazunk egy hash függvényt.

TTP: (Trusted third party) Egy az aláírótól és az ellenőrzőtől különböző személy, akiben legalább az egyikük megbízik.

A diszkrét logaritmikus kriptográfia (DLC) felbontható a véges testek feletti kriptográfiára (FFC) és az elliptikus görbe kriptográfiára (ECC). A két csoport közötti különbség, hogy milyen matematikát használunk. A DSA algoritmus a FFC-re egy példa, az elliptikus görbe algoritmus pedig az ECC-re. Az egész faktorizációs kriptográfiára egy példa az RSA algoritmus.

A digitális aláírás algoritmusok két részből állnak, az aláírás generálásból és az aláírás vizsgá-

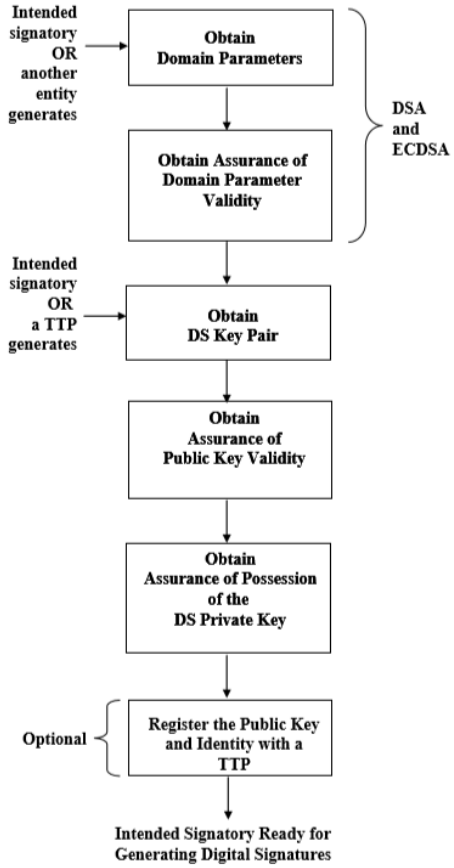
latből. Az aláíró használja a generációt egy adott adathalmazon (dokumentumon), a vizsgáló használja az ellenőrzést, hogy megállapítsa az aláírás hitelességét. Minden aláírónak van egy nyilvános (public key) és egy titkos kulcsa (private key), és ő a tulajdonosa ennek a kulcspárnak. Ezt nevezzük nyilvános kulcsú titkosításnak. Az alábbi ábráról leolvasható, hogy az aláíró az aláírás generálásánál használja a titkos kulcsát. A titkos kulcs használatára az aláíró az egyetlen jogosult személy, ő készíthet elektronikus aláírást. Annak elkerülése végett, hogy más személy is a kulcs tulajdonosának vallja magát, és hamis aláírásokat készítsen, a titkos kulcsnak titkosnak kell maradnia. Az elfogadott algoritmusok úgy lettek kitalálva, hogy megakadályozzon egy másik személyt, aki nem tudja az aláíró titkos kulcsát, hogy ugyanazt azt az aláírást készítse, mint az aláíró egy másik dokumentumon. Más szavakkal, az aláírások nem hamisíthatóak.



1. ábra: A digitális aláírás folyamata

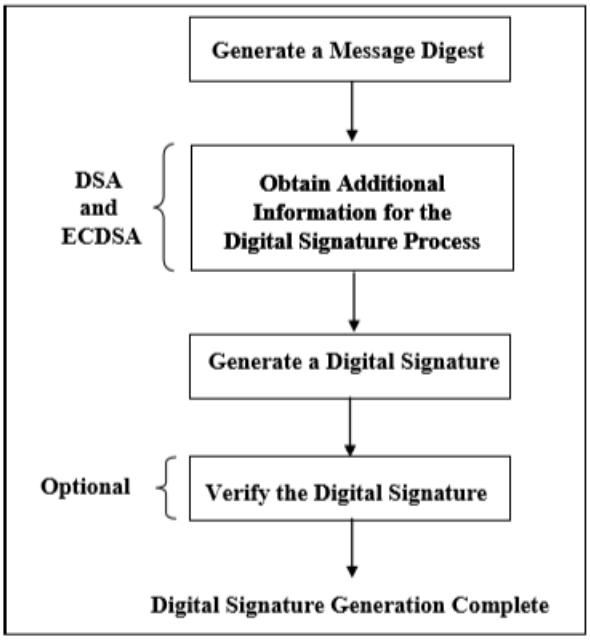
A nyilvános kulcs az aláírás hitelességének ellenőrzésekor játszik szerepet. Nem kell titokban tartani, azonban a hitelességét meg kell őrizni. Bárki ellenőrizni tud egy helyesen titkosított üzenetet a nyilvános kulcs segítségével.

Az üzenet átalakulásra kerül egy fix hosszúságú message digest-é, egy kriptográfiai hash függvény segítségével, mind a generáláshoz, mind az ellenőrzéshez. Az ellenőrző személy számára a hash és az eredeti szöveg is biztosítva van. Az ellenőrző személynek továbbá bizonyosságot kell szereznie arról, hogy a nyilvános kulcs valóban az aláírótól származik, a titkos kulcs a birtokában van, és hogy a két kulcs egymáshoz tartozik. Ezek után, ha az ellenőrzés sikeres, akkor az ellenőrző személy biztos lehet abban, hogy az aláírás valódi.



2. ábra: Az aláíró első lépései

A fenti ábrán láthatóak azok a lépések, amiket meg kell tenni mielőtt valaki elektronikus aláírást hozhat létre. Ezekről a lépésekről az 5. fejezetben írok részletesebben.



3. ábra: Az aláírás generálásának lépései

A használt algoritmustól függően lehetséges, hogy további információt is generálni kell. Például DSA és ECDSA esetén szükség van egy titkos, véletlen számra, amit minden alkalommal újra kell generálni, ha aláírunk.

Ezek után a következő fejezetekben leírtak alapján kell generálni az aláírásunkat, majd az aláírást magunk is ellenőrizhetjük a leírt módokon (opcionális, azonban itt észrevehetünk esetleges hibákat, amelyek a generálás folyamán keletkeztek).

Ahhoz, hogy az aláírás ellenőrzését megkezdhessük, a birtokunkban kell lennie:

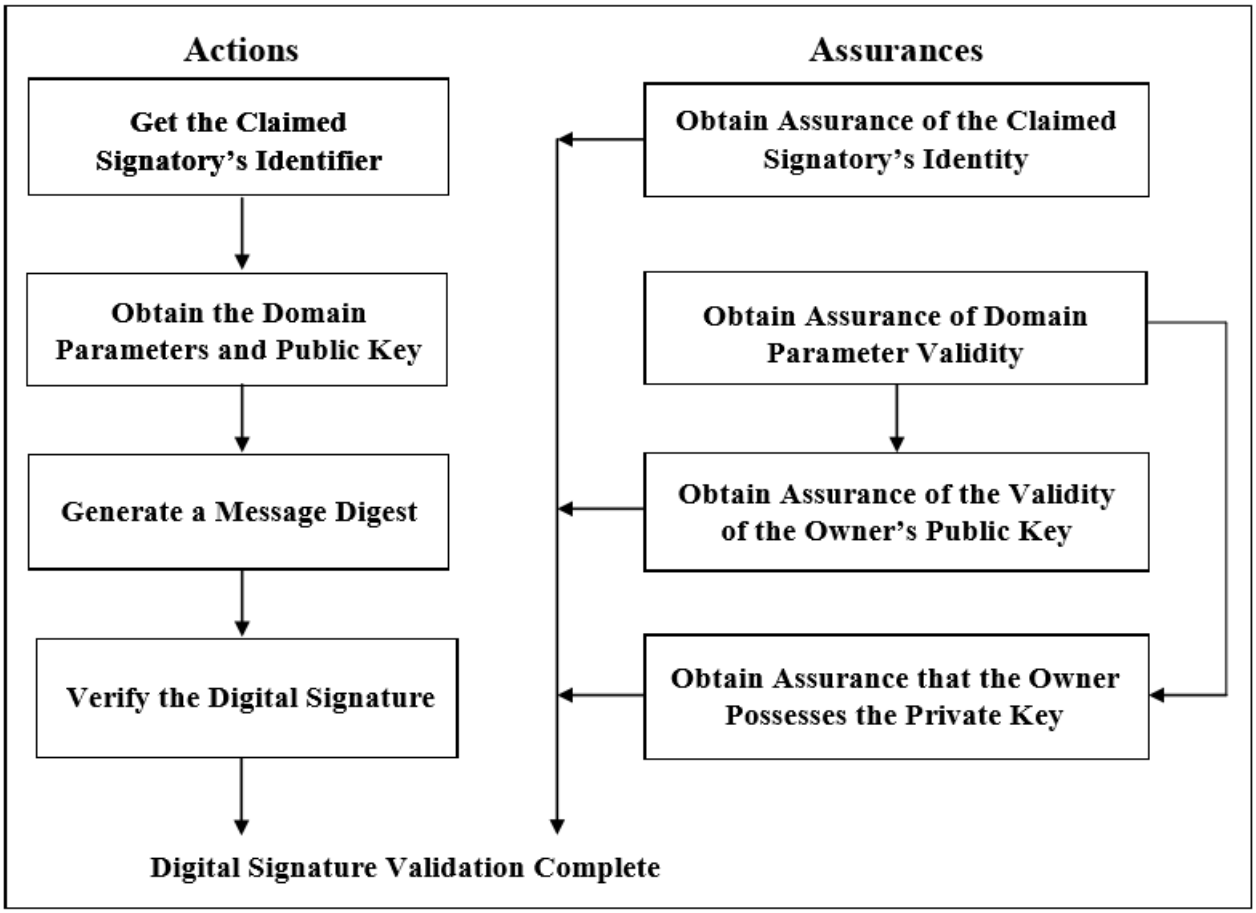
- az aláíró nyilvános kulcsának,
- a domain paramétereknek (ha DSA vagy ECDSA alapú az aláírás),
- egy message digest arról az adatról, amit ellenőrzünk (azzal a hash függvénnyel, amivel az aláírás készült),
- és a vizsgálandó elektromos aláírásnak.

Ha az ellenőrzés hibával végződik, nem tudunk semmit mondani az adat helyességéről, csak azt tudjuk mondani, hogy ez a specifikus nyilvános kulcs, ezzel az aláírási formával az aláírás nem fogadható el.

Mielőtt elfogadja az aláírást, meg kell bizonyosodnunk:

- az aláíró személyazonosságáról,
- a domain paraméterek hitelességéről,
- a nyilvános kulcs hitelességéről,
- és arról, hogy az aláíró személy a titkos kulcs birtokában volt az aláírás pillanatában.

Ha a fenti követelmények teljesülnek, akkor az aláírást hitelesnek ítéljük.



4. ábra: Az aláírás ellenőrzésének lépései

3. fejezet

Hashelés [4][5]

Egy kriptográfiai hash függvény egy olyan hash függvény, ami alkalmas kriptográfiai használatra. Egy matematikai algoritmus, ami feltérképez egy tetszőleges méretű adatot (gyakran az üzenet,"message"), egy adott méretű bitsorozatban (hash, "message digest"). Ez egy egyirányú függvény, ami praktikusan nem visszafejthető. Ideálisan az egyetlen módszer arra, hogy megtaláljunk egy üzenetet, ami az adott hash-t adja vissza, az nem más, mint hogy "brute force" végig nézzünk lehetséges inputokat, és megnézzük, hogy azok megegyeznek-e a hashünkel. A kriptográfiai hash függvény a modern kriptográfia alapvető eszköze.

Az ideális hash függvény rendelkezik a következő tulajdonságokkal:

- determinisztikus, azaz ugyanaz az üzenet mindig ugyanazt a hasht adja eredményül
- gyorsan ki lehet számolni a message digest-et minden üzenetre.
- nem lehetséges olyan üzenetek készíteni, ami az adott hash-t generálja
- nem lehet találni két különböző üzenetek, ami ugyanazt a hasht adja vissza
- egy kis változtatás az üzenetben olyan mértékben megváltoztatja a message digestet, hogy az új hash korreláltalannak tűnik a régi hashhez képest

A leírás alapján a következő hash függvények számítanak megbízhatónak:

- SHA-224,
- SHA-256,
- SHA-384,
- SHA-512,

- SHA-512/224
- és SHA-512/256.

A SHA-1 az Egyesült államok kormányának Capstone projektjének keretein belül fejlesztették ki. Az algoritmus első specifikációját (SHA-0) 1993-ban publikálták, Secure Hash Standard címmel. Nem sokkal később visszahívták és egy újabb verziót adtak ki, SHA-1 néven. 2017 Február 23.-án a CWI (Centrum Wiskunde & Informatica) és a Google publikált egy támadást, ami során generáltak két eltérő PDF fájlt ugyanazzal a SHA-1 hash-el, mindössze $2^{63.1}$ számítás alatt. Ez 100,000-szer gyorsabb, mint ha brute force próbálnánk feltörni a "Birthday Attack" módszerrel, ami 2^{80} számítás alatt tudná ezt megtenni. Ezt a támadást nevezik "SHAtterd attack"-nak és az algoritmus többé nem számít elég erősnek kriptográfiai célokra.

4. fejezet

RSA algoritmus [3][2]

Terjedelmi korlátok miatt, részletes leírás helyett egy kisebb példán keresztül írom le az RSA algoritmus működését.

Az RSA algoritmus paraméterei:

Nyilvános:

- n: A titkos p és q prímek szorzata. A szám hossza ($nlen$) lehet 2048 vagy 3072 bit.
- e: Egy egész szám amire $lnko(\varphi(n), e) = 1$, ahol $\varphi()$ az Euler függvény.

Titkos:

- p, q: RSA számok, 1024 vagy 1536 bit hosszú prímek (k).
- d: e inverze $mod\ lnko(p-1, q-1)$

Az Euler függvény az n-nél kisebb és tőle relatív prím természetes számok darabszámát jelöli. Abban az esetben, ha n prím akkor $\varphi(n) = n-1$. Ebből könnyedén következik, hogy $\varphi(p*q) = (p-1)(q-1)$, ha p és q prím.

A kis Fermat tétel alapján, ha $0 < a$ és p prím, akkor $a^{p-1} \equiv 1\ mod\ p$

Az Euler-Fermat tétel szerint, ha $lnko(n, a) = 1$, akkor $a^{\varphi(n)} \equiv 1\ mod\ n$

4.1. A kulcspár generálás folyamata

Első lépésként, meg kell választanunk e értékét p,q és d megválasztása előtt, úgy hogy $2^{16} < e < 2^{256}$ és e páratlan.

Ezután egy véletlen prím generátorral elő kell állítanunk p és q prímeket, úgy hogy a következők teljesüljenek:

- $(p-1)$ és $(q-1)$ relatív prím e-től
- p-nek ki kell elégítenie a $(\sqrt{2})(2^{\frac{nlen}{2}-1}) \leq p \leq (2^{\frac{nlen}{2}-1})$ feltételt
- q-nak ki kell elégítenie a $(\sqrt{2})(2^{\frac{nlen}{2}-1}) \leq q \leq (2^{\frac{nlen}{2}-1})$ feltételt
- $\|p-q\| > 2^{\frac{nlen}{2}-100}$

Számítsuk ki d-t úgy, hogy $2^{\frac{nlen}{2}} < d < lnko(p-1, q-1)$, valamint $d = e^{-1}\ mod\ lnko(p-1, q-1)$.

4.2. Példa [3]

Legyen $e = 7$, $p = 3$, $q = 11$. Ekkor $n = 33$, $\varphi(n) = (3-1)(11-1) = 20$ és e és $\varphi(n)$ relatív prímek. $7^{-1}\ mod\ 20 = 3$ így ez lesz a titkos kulcs.

Tegyük fel, hogy az üzenet, amit elküldünk a hashelés után a "9" szám.

Ekkor az elektromos aláírást úgy készítjük el, hogy az üzenetet felemeljük a titkos kulcsunk hatványára, majd ennek vegyem az n-nel osztva lévő maradékát, azaz

$m^d = 9^3 \equiv 3\ mod\ 33$.

Ha le akarom ellenőrizni, hogy minden rendben történt, vagy valaki validálni akarja az aláírást, akkor ezt a számot kell felemelnie a nyilvános kulcs hatványára $mod\ n$.

$3^7 \equiv 9\ mod\ 33$.

Ezt összehasonlítva a hash-el, bizonyítottam, hogy valóban én írtam alá a dokumentumot.

5. fejezet

DSA algoritmus [2]

5.1. Az algoritmus felépítése

A DSA elektronikus aláírás létrehozásához szükség van domain paraméterek egy halmazára (p, q, g), x titkos kulcsra, k üzenetenkénti titkos számra, az aláírandó üzenetre és egy hash függvényre.

A megerősítéshez szükség van ugyanazokra a domain paraméterekre (p, q, g), y nyilvános kulcsra, ami matematikailag összefügg x titkos kulccsal, amit a generálás során használtunk, a megerősíteni kívánt adatra, és ugyanarra a hash függvényre, amit a generálás során használtunk. Ezek a paraméterek definíció szerint:

- p: a prím modulus, ahol $2^{L-1} < p < 2^L$, és L p bitjeinek a száma. L lehetséges értékei az 1. táblázatban találhatók.
- q: $(p-1)$ egy prím osztója, ahol $2^{N-1} < q <^N$, és N q bit hossza. N lehetséges értékei az 1. táblázatban találhatók.
- g: $GF(p)$ multiplikatív csoport, egy q-ad rendű részcsoportjának generátora, ahol $1 < g < p$.
- x: A titkos kulcs (titokban kell maradnia). x egy véletlen vagy egy pszeudo véletlen egész, úgy hogy $0 < x < q$, azaz x a $[1, q-1]$ intervallumban van.
- y: a nyilvános kulcs, ahol $y = g^x \ mod \ p$.
- k: egy titkos szám, amit minden üzenethez újra kell generálni. k egy véletlen vagy pszeudo véletlen egész, úgy hogy $0 < k < q$, azaz k a $[1, q-1]$ intervallumban van.

p **bithossza** (L)	q **bithossza** (N)
1024	160
2048	224
2048	256
3072	256

1. táblázat: A lehetséges bithossz párok

A generálás során csak jóváhagyott hash függvényeket lehet használni. A használt hash függvény ereje nem lehet kisebb a használt (L, N) pár erejénél.

A DSA algoritmus elvárja, hogy a kulcspárunk, amit a generálása és megerősítés folyamán használunk, bizonyos domain paraméterek segítségével legyenek előállítva. Ezek a paraméterek elérhetőek lehetnek egy csoportnak, vagy akár nyilvánosak is lehetnek, azonban használatuk előtt meg kell győződni érvényességükről. Minden kulcspárnak kell lennie egy domain paraméter halmaza, amihez tartozik (pl. egy tanúsítvánnyal, ami összekapcsolja a paramétereket a nyilvános kulccsal).

Minden aláírónak birtokában van egy kulcspár, x, y , amelyek matematikailag összekapcsolódnak. A kulcspárok csak egy bizonyos ideig használhatóak, utána új kulcspárt kell generálni. A nyilvános kulcs használható az intervallum letelte után is, amíg vannak aláírások, amiknek hitelességét ezzel a kulccsal lehet igazolni.

Néhány előírás a kulcspárok védelmére:

- A domain paraméterek érvényességéről meg kell győződni a kulcspár generálás előtt és az aláírás érvényességének ellenőrzése előtt.
- Minden kulcspárhoz hozzá kell lennie rendelve annak a paraméter halmaznak, ami generálta.
- Minden kulcspár csak akkor alkalmas aláírások generálására és ellenőrzésére, ha a hozzá tartozó paramétereket használjuk.
- A titkos kulcs csak az itt leírtak alapján használható aláírás generálásra, és titokban kell tartani. A nyilvános kulcs csak megerősítésre használható, akár publikus is lehet.
- Az aláírónak tudnia kell bizonyítani, hogy a birtokában van a titkos kulcs az aláírás generálása előtt.
- A titkos kulcshoz illetéktelen személyek nem férhetnek hozzá.
- A nyilvános kulcshoz illetéktelen személyek nem módosíthatják, vagy cserélhetik. Egy nyilvános kulcs tanúsítvány, amit egy Tanúsító bizottság (CA) aláírt megfelelő védelmet jelent.

- Az ellenőrző személynek meg kell arról bizonyosodnia, hogy a nyilvános kulcs, a paraméterek és a tulajdonos összeköthető.
- Az ellenőrző személynek biztos forrásból kell megszereznie a nyilvános kulcsot.
- Az ellenőrző személynek ellenőriznie kell, hogy az aláíró a kulcspár tulajdonosa, és biztokában van a titkos kulcs, amivel az aláírás létre lett hozva.
- Az aláírónak és az ellenőrző személynek biztosnak kell lennie a nyilvános kulcs érvényességében.

Egy új titkos számot (k) kell generálni minden egyes aláírás előtt. k^{-1} a k szám inverze $mod\ q$. k értékét előre is kiszámíthatjuk, az üzenet ismerete nem szükséges az előállításhoz.

Ha a fent elhangzott lépések megtörténtek, akkor elkezdhetjük az elektronikus aláírás generálását.

Legyen N q bithossza. $min(N, outlen)$ jelölje N és $outlen$ minimumát, ahol $outlen$ a hash függvény kimenetének a bithossza. Egy M üzenet aláírása r és s számpárból áll, amiket a következőképpen számíthatunk ki:

$r = (g^k\ mod\ p)\ mod\ q.$

$z = Hash(M)$ első $min(N, outlen)$ bitje.

$s = (k^{-1}(z + xr))\ mod\ q.$

s számításakor, z-t, amit $Hash(M)$-ből képzünk, át kell váltani egésszé. (az átváltás algoritmusa az 5.7. fejezetben). r kiszámítható amint k, p, q és g számok a birtokunkba kerülnek. r és s értékét ellenőriznünk kell, hogy $r = 0$ vagy $s = 0$. Ha $r = 0$ vagy $s = 0$, új k értéket kell generálni, és (r, s)-t újra kell számolni. Ha a folyamat jól lett végrehajtva, akkor az előbbi eset nagyon valószínűtlen. Az aláírást, (r, s)-t elküldhetjük az ellenőrző személynek az üzenettel együtt.

A validálás folyamatát bárki el tudja végezni a nyilvános kulcs segítségével. Az aláíró ellenőrizheti, hogy nem keletkezett hiba a generálás során, mielőtt elküldené a címzettnek. A címzett pedig bizonyítékot szerezhet arról, hogy a dokumentumot valóban az írta alá, aki állítja. Ellenőrzés előtt, a domain paramétereket, az aláíró nyilvános kulcsát és kilétét hiteles módon meg kell osztani az ellenőrző személlyel (TTP vagy CA által, esetleg személyes találkozón).

Legyenek $M^{'}$, $r^{'}$, és $s^{'}$ az ellenőrző személy által kapott adatok, és legyen y a nyilvános kulcs. Legyen N q bithossza, valamint legyen $outlen$ a hash függvény kimenetének a bithossza.

A validáció folyamata a következőképpen zajlik:

- Ellenőrizzük, hogy $0 < r^{'} < q$ és $0 < s^{'} < q$. Ha bármelyik eset nem igaz, akkor az aláírást el kell utasítani.

- Majd számítsuk ki a következőket:

  $w = (s^{'})^{-1}\ mod\ q.$

  $z = Hash(M^{'})$ első $min(N, outlen)$ bit-je.

  $u1 = (zw)\ mod\ q.$

  $u2 = ((r^{'})w)\ mod\ q.$

  $v = (((g)^{u1}(y)^{u2})\ mod\ p)\ mod\ q.$

  z-t át kell alakítani egész számmá.(5.7. fejezet)

- Ha $v = r^{'}$, akkor az aláírást elfogadjuk. Bizonyítás a 5.1. szakaszban.

- Ha $v \neq r^{'}$, akkor az üzenet módosítva volt, hiba történt az aláírás generálás során, vagy egy csaló megpróbálta hamisítani az aláírást. Az aláírás invalid. Nem tudunk következtetni arra, hogy az adat helyes, csak azt tudjuk mondani, hogy amikor a nyilvános kulccsal ellenőrizzük az üzenetet, az aláírás hibás hozzá.

- Mielőtt az aláírást elfogadjuk, a validátornak rendelkeznie kell a felsorolt biztosítékokkal.

5.1.1. Bizonyítás

Ebben a fejezetben bebizonyítom, hogy ha $M^{'} = M$, $r^{'} = r$ és $s^{'} = s$ az aláírás ellenőrzés folyamán, akkor $v = r^{'}$.

Legyen $Hash$ a használt Hash függvény. Használjuk a következő Lemmát .

Lemma: Legyenek p és q prímek, úgy hogy q ossza $(p-1)$-et, legyen h egy p-nél kisebb egész szám, valamint $g = (h^{\frac{(p-1)}{q}}\ mod\ p)$. Ekkor $(g^q\ mod\ p) = 1$, és ha $(m\ mod\ q) = (n\ mod\ q)$, akkor $(g^m\ mod\ p) = (g^n\ mod\ p)$.

Bizonyítás:

$$\begin{aligned} g^q\ mod\ p &= (h^{\frac{p-1}{q}}\ mod\ p)^q\ mod\ p \\ &= h^{(p-1)}\ mod\ p \\ &= 1 \end{aligned}$$

a Kis Fermat's Tétel alapján. Legyen $(m\ mod\ q) = (n\ mod\ q)$, tehát, $m = (n + kq)$ valamilyen k egészre. Ekkor

$$\begin{aligned} g^m\ mod\ p &= g^{n+kq}\ mod\ p \\ &= (g^n g^{kq})\ mod\ p \\ &= ((g^n\ mod\ p)(g^q\ mod\ p)^k)\ mod\ p \end{aligned}$$

$$= g^n \ mod \ p$$

, mivel $(g^q \ mod \ p) = 1$.

Tétel: Ha $M^{'} = M$, $r^{'} = r$, and $s^{'} = s$ az aláírás ellenőrzése során, akkor $v = r^{'}$.

Bizonyítás:

$$w = (s^{'})^{-1} \ mod \ q = s^{-1} \ mod \ q,$$

$$u_1 = ((Hash(M^{'}))w) \ mod \ q = ((Hash(M))w) \ mod \ q,$$

$$u_2 = ((r^{'})w) \ mod \ q = (rw) \ mod \ q.$$

Jelenleg $y = (g^x \ mod \ p)$, így a Lemma alapján,

$$v = ((g^{u_1} y^{u_2}) \ mod \ p) \ mod \ q$$

$$= ((g^{Hash(M)w} y^{rw}) \ mod \ p) \ mod \ q$$

$$= ((g^{Hash(M)w} g^{xrw}) \ mod \ p) \ mod \ q$$

$$= ((g^{(Hash(M)+xr)w)} \ mod \ p) \ mod \ q.$$

Valamint:

$$s = (k^{-1}(Hash(M) + xr)) \ mod \ q.$$

Tehát:

$$w = (k(Hash(M) + xr) - 1) mod q$$

$$(Hash(M) + xr) w mod q = k mod q.$$

Ezért a Lemma miatt:

$$v = (gkmodp) modq = r.$$

5.2. Domain paraméterek generálása és validálása

Ebben az alfejezetben a DSA algoritmushoz szükséges domain paraméterek előállítására és validálására szolgáló algoritmusokat mutatom be. A domain paraméterek a $(p, q, g\{domain\_parameter\_seed, counter\},)$ halmaz elemeit jelölik.
A *domain_parameter_seed* és a *counter* két opcionális paraméter.
A *domain_parameter_seed* határozza meg, hogy milyen sorrendben ellenőrizzük le a számokat p és q generálásakor.

5.2.1. P és q generálása

A leírás két módszert ír a p és q prímek generálására, az egyik módszer véletlen elemeket választ ki egy intervallumból, és egy valószínűségi prím teszttel eldönti, hogy a szám összetett, vagy nagy valószínűséggel prím-e. A másik módszer kisebb egészekből rak össze nagyobb egészeket, úgy hogy azok garantáltan prímek legyenek. Dolgozatom során az első módszert dolgozom fel.

Az algoritmus eredményként visszaadja a keresett p és q prímeket, valamint a *domain_parameter_seed* és *counter* értékét, amelyek a prímek generálásakor keletkeztek, hogy az ellenőrzés folyamán a validáló személy ezeket fel tudja használni.
A *domain_parameter_seed* és a *counter* értékét nem kell titokban tartani. Legyen $Hash()$ a használt hash függvény, és legyen *outlen* a hash bithossza. *outlen* legyen legalább akkora, mint N.

Input:

1. L: p prím hossza (bit-ben).
2. N: q prím hossza (bit-ben).
3. *seedlen*: a domain paraméter seed kívánt hossza. $seedlen \geq N$ szükséges.

Output:

1. *status*: Az algoritmus eredménye. VALID vagy INVALID. INVALID esetén a többi output 0 ként jelenjen meg.
2. p, q: A prímek p és q.
3. *domain_parameter_seed* (Opcionális): A seed amit p és q generálásakor használtunk.
4. *counter* (Opcionális): A generálás során létrejövő számláló.

Folyamat:

1. Ellenőrizzük, hogy (L, N) pár megfelelő-e. Ha nem, akkor INVALID-ot adjunk vissza.
2. Ha $(seedlen < N)$, akkor INVALID.
3. $n = \lceil \frac{L}{outlen} \rceil - 1$.
4. $b = L - 1 - (noutlen)$.
5. Adjunk meg egy önkényes *seedlen* bitláncot *domain_parameter_seed*-ként.

6. $U = Hash(domain\_parameter\_seed)\ mod\ 2^{N-1}$.

7. $q = 2^{N-1} + U + 1 - (U\ mod\ 2)$.

8. Egy valószínűségi prím teszttel ellenőrizzük, hogy q prím-e.

9. Ha nem, akkor menjünk vissza az 5. lépésre.

10. $offset = 1$.

11. For $counter = 0$ to $(4L - 1)$ do

    (a) For $j = 0$ to n do $V_j = Hash((domain\_parameter\_seed + offset + j)\ mod\ 2^{seedlen})$.

    (b) $W = V_0 + (V_1 * 2^{outlen}) + \cdots + (V_{n-1} * 2^{(n-1)*outlen}) + ((V_n\ mod\ 2^b) * 2^{n*outlen})$.

    (c) $X = W + 2^{L-1}$. Megjegyzés: $0 \leq W < 2^{L-1}$; tehát,$2^{L-1} \leq X < 2^L$.

    (d) $c = X\ mod\ 2q$.

    (e) $p = X - (c - 1)$. Megjegyzés: $p \equiv 1\ (mod\ 2q)$.

    (f) Ha $(p < 2^{L-1})$, akkor menjünk a 11.(i). lépésre.

    (g) Ellenőrizzük, hogy p prím-e.

    (h) Ha p prím, akkor VALID. Adjuk vissza p és q értékét.
    (opcionálisan) a *domain_parameter_seed* és *counter* értékét is.

    (i) $offset = offset + n + 1$.

12. Menjünk vissza az 5. lépésre.

5.3. P és q validálása

Ha a prímek az előző módszerrel lettek generálva, akkor ezzel az algoritmussal lehet őket validálni. Az algoritmusnak szüksége van a p, q, *domain_parameter_seed* és a *counter* értékeire. Legyen $Hash()$ a generáció során használt hash függvény.

Input:

1. p, q: A generált prímek p és q.

2. *domain_parameter_seed*: A használt domain parameter seed.

3. *counter*: A count érték.

Output:

1. *status*: A validáció eredménye.VALID vagy INVALID.

Folyamat:

1. $L = len(p)$.
2. $N = len(q)$.
3. Ellenőrizzük, hogy (L, N) pár benne van-e az elfogadott párok között. Ha nem, akkor INVALID.
4. Ha $(counter > (4L - 1))$, akkor INVALID.
5. $seedlen = len(domain\_parameter\_seed)$.
6. Ha $(seedlen < N)$, akkor INVALID.
7. $U = Hash(domain\_parameter\_seed)\ mod\ 2^{N-1}$.
8. $computed\_q = 2^{N-1} + U + 1 - (U\ mod\ 2)$.
9. Ellenőrizzük, hogy $computed\_q$ prím-e egy valószínűségi prím teszttel.
   Ha $(computed\_q \neq q)$ vagy ($computed\_q$ nem prím), akkor INVALID.
10. $n = \lceil \frac{L}{outlen} \rceil - 1$.
11. $b = L - 1 - (n * outlen)$.
12. $offset = 1$.
13. For $i = 0$ to $counter$ do
    (a) For $j = 0$ to n do
    $V_j = Hash((domain\_parameter\_seed + offset + j)\ mod\ 2^{seedlen})$.
    (b) $W = V_0 + (V_1 * 2^{outlen}) + \cdots + (V_{n-1} * 2^{(n-1)*outlen}) + ((V_n\ mod\ 2^b) * 2^{n*outlen})$.
    (c) $X = W + 2^{L-1}$.
    (d) $c = X\ mod\ 2q$.
    (e) $computed\_p = X - (c - 1)$.
    (f) Ha $(computed\_p < 2^{L-1})$, akkor menjünk a 13.(i) lépésre.
    (g) Ellenőrizzük, hogy $computed\_p$ prím-e.
    (h) Ha $computed\_p$ prím, akkor menjünk a 14. lépésre.
    (i) $offset = offset + n + 1$.

14. Ha $((i \neq counter)$ vagy $(computed\_p \neq p)$ vagy $(computed\_p$ nem prím), akkor INVALID.

15. Különben VALID.

5.3.1. Generátor létrehozása

g értéke p-től és q-tól függ. A leírásban két algoritmus szerepel a generátor kiszámítására. Az első akkor használandó, amikor g validálása nem szükséges. A második leírásban olyan generátor készítenek, ami részlegesen validálható. Dolgozatomban a második algoritmust írom le.

5.3.2. A generátor (validálható) létrehozása

g generálása a p, q és $domain\_parameter\_seed$ értékeket használja fel. Ha a prímek generálására az előző alfejezetben szereplő algoritmust használtuk, akkor kapnunk kellett egy $domain\_parameter\_seed$-et (a másik algoritmus során több seed-ből kellene összeraknunk ezt). Egy $index$ nevű változó segítségével több generátort is létre hozhatunk a p és q értékekhez. Legyen $Hash()$ a p és q generálásához használt hash függvény.

Input:

1. p, q: A prímek.

2. $domain\_parameter\_seed$: A seed, amit p és q generálása során használtunk.

3. $index$: Egy 8 hosszúságú bitlánc, ami egy egész számot jelöl.

Output:

1. $status$: Az algoritmus eredménye. VALID vagy INVALID.

2. g: g értéke.

Folyamat: Megjegyzés: $count$ egy 16-bites egész. Megjegyzés: Ellenőrizzük, hogy az $index$ valid-e.

1. Ha $index$ hibás, akkor INVALID.

2. $N = len(q)$.

3. $e = \frac{(p-1)}{q}$.

4. $count = 0$.
5. $count = count + 1$. Megjegyzés: Ellenőrizzük, hogy *count* nem megy-e körbe 0-ba.
6. If ($count = 0$), akkor INVALID. Megjegyzés: "ggen" a következő bitlánc 0x6767656E (hexadecimális).
7. $U = domain\_parameter\_seed||''ggen''||index||count$.
8. $W = Hash(U)$.
9. $g = W^e \ mod \ p$.
10. If ($g < 2$), Menjünk vissza az 5. lépéshez. Megjegyzés: Nem találtuk meg a generátort.
11. Különben VALID. Adjuk vissza g értékét.

5.3.3. A generátor validálása

A következő algoritmust használjuk g validálására, ha az előállításához az előző alfejezetben leírt algoritmust használtuk. Felhasználjuk p, q, $domain\_parameter\_seed$ és $index$ értékét, valamint feltesszük, hogy p-t és q-t már validáltuk az előzőekben leírtak szerint. Legyen $Hash()$ a g generátor előállításához használt hash függvény (tehát ugyanaz a függvény, amit p és q generálására használtunk), $index$ pedig az ott használt szám.

Input:

1. p, q: A prímek.
2. $domain\_parameter\_seed$: p és q generálására használt seed.
3. $index$: Egy 8-bit hosszú sorozat, ami egy számot jelöl.
4. g: A validálandó g.

Output:

1. *status*: A generációs eljárás státusza, az értéke vagy VALID vagy INVALID.

Folyamat:

(*count* egy 16-bites egész.) Megjegyzés: Ellenőrizzük, hogy az index helyesen lett-e megadva.

1. Ha az index hibás, akkor INVALID.

2. Ellenőrizzük, hogy $2 \leq g \leq (p-1)$. Ha nem, akkor INVALID.
3. Ha $(g^q \neq 1 \ mod \ p)$, akkor INVALID.
4. $N = len(q)$.
5. $e = \frac{(p-1)}{q}$.
6. $count = 0$.
7. $count = count + 1$. Megjegyzés: Elenőrizzük, hogy *count* nem megy-e körbe 0-ig.
8. Ha $(count = 0)$, INVALID-ot adunk vissza. Megjegyzés: "ggen" a következő bitlánc 0x6767656E (hexadecimális).
9. $U = domain\_parameter\_seed||''ggen''||index||count$.
10. $W = Hash(U)$.
11. $computed\_g = W^e \ mod \ p$.
12. Ha $(computed\_g < 2)$, Menjünk vissza a 7. lépésre. Megjegyzés: Nem találtunk generátort.
13. Ha $(computed\_g = g)$, VALID-ot, különben INVALID-ot adunk vissza.

5.4. Kulcspár generálás

Az (x, y) kulcspárt a $(p, \ q, \ g \ \{, domain\_parameter\_seed, \ counter\})$ domain paraméterek halmazából állítjuk elő. Dolgozatomban két algoritmust írok le a titkos és nyilvános kulcsok generálására. A kulcspár generálása előtt meg kell róla győződni, hogy a domain paraméterek ervényességét.

5.4.1. Extra Random Bit-ek használatával

Ezzel a módszerrel, 64-gyel több bit-et kérünk egy RBG (Random Bit Generator)-ből, mint amennyi x-hez kellene, így a 6.lépésben lévő *mod* függvény által okozott eltérés elhanyagolható.

Input:

1. (p, q, g): Az előzőekben generált domain paraméterek. Az értékeket egészként adjuk meg.

Output:

1. *status*: A titkos szám generálásának státusa. Az eredmény vagy OK, vagy HIBA.

2. (x, y): A generált titkor és nyilvános kulcs. Hiba esetén hibás értéket visszaadása szükséges x-hez és y-hoz, ezek $Invalid\_x$ és $Invalid\_y$-ként szerepelnek. x-t és y-z egészként kapjuk vissza. A titkos kulcs x a $[1, q-1]$ intervallumból, a nyilvános kulcs y pedig a $[1, p-1]$ intervallumból való.

Folyamat:

1. $N = len(q)$; $L = len(p)$. Megjegyzés: Ellenőrizzük, hogy az (L, N) számpár megfelelő-e.

2. Ha az (L, N) számpár hibás, akkor adjunk vissza egy HIBA jelzést, $Invalid\_x$-t, és $Invalid\_y$-t.

3. $requested\_security\_strength$ = Az (L, N) pár titkosítási ereje; (melléklet).

4. Generáljunk egy $N + 64$ hosszú véletlen bitsorozatot ($returned\_bits$) egy RBG segítségével, aminek az ereje meghaladja a $requested\_security\_strength$-et. Ha HIBA-t kapunk, akkor adjunk vissza egy HIBA jelzést, $Invalid\_x$-t, és $Invalid\_y$-t.

5. Alakítsuk át $returned\_bits$-t c nemnegatív egésszé (5.7. fejezet).

6. $x = (c \ mod \ (q-1)) + 1$. Megjegyzés: $0 \leq c \ mod \ (q-1) \leq q-2$, és ebből következik, hogy $1 \leq x \leq q-1$.

7. $y = g^x \ mod \ p$.

8. Adjuk vissza OK-t, x-t, és y-t.

5.4.2. "Testing Candidate"-ek használatával

Ezzel a módszerrel, egy véletlen szám lesz generálva és ellenőrizve, hogy az ebből képzett x a megfelelő intervallumban van-e . Ha x az intervallumon kívül esik, akkor egy új véletlen számot generálunk. Ezt addig ismételjük, amíg egy megfelelő x-et nem kapunk.

Input:

1. (p, q, g): Az előzőekben generált domain paraméterek. Az értékeket egészként adjuk meg.

Output:

1. *status*: A titkos szám generálásának státusa. Az eredmény vagy OK, vagy HIBA.

2. (x, y): A generált titkor és nyilvános kulcs. Hiba esetén hibás értéket visszaadása szükséges x-hez és y-hoz, ezek $Invalid\_x$ és $Invalid\_y$-ként szerepelnek. x-t és y-z egészként kapjuk vissza. A titkos kulcs x a $[1, q-1]$ intervallumból, a nyilvános kulcs y pedig a $[1, p-1]$ intervallumból való.

Folyamat:

1. $N = len(q)$; $L = len(p)$. Megjegyzés: Ellenőrizzük, hogy az (L, N) számpár megfelelő-e.
2. Ha az (L, N) számpár hibás, akkor adjunk vissza egy HIBA jelzést, $Invalid\_x$-t, és $Invalid\_y$-t.
3. $requested\_security\_strength$ = Az (L, N) pár titkosítási ereje; (melléklet).
4. Generáljunk egy N hosszú véletlen bitsorozatot ($returned\_bits$) egy RBG segítségével, aminek az ereje meghaladja a $requested\_security\_strength$-et. Ha HIBA-t kapunk, akkor adjunk vissza egy HIBA jelzést, $Invalid\_x$-t, és $Invalid\_y$-t.
5. Alakítsuk át $returned\_bits$-t c nemnegatív egésszé (5.7. fejezet).
6. Ha $(c > q - 2)$, akkor térjünk vissza a 4. lépésre.
7. $x = c + 1$.
8. $y = g^x \ mod \ p$.
9. Adjuk vissza OK-t, x-t, és y-t.

5.5. Üzenetenkénti titkos szám genenárása

A DSA-ban szükség van egy új titkos véletlen szám (k) generálására, minden aláírandó üzenethez. Ebben a dolgozatban két módszert írok le k generálására. Legyen $inverse(k, q)$ egy olyan függvény, amely kiszámítja egy nem negatív egész (k) inverzét modulo q, ahol q prím. A függvénynek egy lehetséges módszere az 5.6. fejezetben van leírva.

5.5.1. Extra Random Bit-ek használatával

Ezzel a módszerrel, 64-gyel több bit-et kérünk egy RBG-től, mint amennyi k-hoz kellene, így a 6. lépésben lévő *mod* függvény által okozott eltérés elhanyagolható. Ez a folyamat (vagy ezzel megegyező) használható az üzenetenkénti különböző titkos szám generálására.

Input:

1. (p, q, g): Az előzőekben generált domain paraméterek.

Output:

1. $status$: A titkos szám generálásának státusa. Az eredmény vagy OK, vagy HIBA.
2. (k, k^{-1}): A üzenetenként változó, titkos szám, k és inverze, k^{-1} . Hiba esetén is vissza kell adnunk értékeket, ebben a leírásban ezek $Invalid\_k$ és $Invalid\_k\_inverse$-ként szerepelnek. k és k^{-1} a $[1, q-1]$ intervallumból kerül ki.

Folyamat:

1. $N = len(q)$; $L = len(p)$. Megjegyzés: Ellenőrizni kell, hogy a (L, N) számpár a megfelelő értékek közül való-e.
2. Ha az (L, N) számpár hibás, adjunk vissza egy HIBA jelzést, $Invalid\_k$-t, és $Invalid\_k\_inverse$-t.
3. $requested\_security\_strength$ = a (L, N) pár titkosítási ereje; (melléklet).
4. Generáljunk egy $N+64$ hosszú véletlen számot ($returned\_bits$) egy olyan RBG segítségével, aminek az ereje meghaladja a $requested\_security\_strength$-et. Ha HIBA jelzést kapunk vissza, akkor adjunk vissza HIBA-t, $Invalid\_k$-t, és $Invalid\_k\_inverse$-t.
5. Alakítsuk át $returned\_bits$-t egy nemnegatív egésszé (c) (5.7. fejezet).
6. $k = (c \ mod(q-1)) + 1$.
7. $(status, k^{-1}) = inverse(k, q)$.
8. Adjuk vissza a $status$-t, k-t, és k^{-1}.

5.5.2. "Testing Candidate"-ek használatával

Ezzel a módszerrel, egy véletlen szám lesz generálva és ellenőrizve, hogy az ebből képzett k a megfelelő intervallumban van-e . Ha k az intervallumon kívül esik, akkor egy új véletlen számot generálunk és ezt ismételjük, amíg egy megfelelő k-t nem kapunk.

Input:

1. (p, q, g): Az előzőekben generált domain paraméterek.

Output:

1. $status$: A titkos szám generálásának státusa. Az eredmény vagy OK, vagy HIBA.
2. (k, k^{-1}): A üzenetenként változó, titkos szám, k és inverze, k^{-1} . Hiba esetén is vissza kell adnunk értékeket, ebben a leírásban ezek $Invalid\_k$ és $Invalid\_k\_inverse$-ként szerepelnek. k és k^{-1} a $[1, q-1]$ intervallumból kerül ki.

Folyamat:

1. $N = len(q)$; $L = len(p)$. Megjegyzés: Ellenőrizni kell, hogy a (L, N) számpár a megfelelő értékek közül való-e.
2. Ha az (L, N) számpár hibás, adjunk vissza egy HIBA jelzést, $Invalid\_k$-t, és $Invalid\_k\_inverse$-t.
3. $requested\_security\_strength$ = a (L, N) pár titkosítási ereje; (melléklet).
4. Generáljunk egy N hosszú véletlen számot ($returned\_bits$) egy olyan RBG segítségével, aminek az ereje meghaladja a $requested\_security\_strength$-et. Ha HIBA jelzést kapunk vissza, akkor adjunk vissza HIBA-t, $Invalid\_k$-t, és $Invalid\_k\_inverse$-t.
5. Alakítsuk át $returned\_bits$-t egy nemnegatív egésszé (c) (5.7. fejezet).
6. Ha $(c > q - 2)$, akkor menjünk vissza a 4. lépésre.
7. $k = c + 1$.
8. $(status, k^{-1}) = inverse(k, q)$.
9. Adjuk vissza a $status$-t, k-t, és k^{-1}.

5.6. Inverz számítása

Az alábbi (vagy egy ezzel megegyező eredményű) algoritmus használatát írják elő a multiplikatív inverz, z^{-1} mod a kiszámítására, ahol $0 < z < a$, $0 < z^{-1} < a$, és a egy prím. Ebben a dolgozatban, z mindig k vagy s, valamint a mindig q vagy n.

Input:

1. z: Az invertálandó érték mod a (k vagy s).
2. a: A domain paraméter és (prím) modulus (q vagy n).

Output:

1. *status*: A függvény eredményének a státusza, vagy OK, vagy HIBA.
2. z^{-1}: A multiplikatív inverze z-nek $mod\ a$, ha létezik.

Folyamat:

1. Ellenőrizzük, hogy a és z pozitív egész számok és $z < a$. Ha nem, akkor HIBA-t adunk vissza.
2. Legyen $i = a$, $j = z$, $y_2 = 0$, és $y_1 = 1$.
3. $quotient = \lfloor \frac{i}{j} \rfloor$.
4. $remainder = i - (j * quotient)$.
5. $y = y2 - (y1 * quotient)$.
6. Legyen $i = j$, $j = remainder$, $y_2 = y_1$, és $y_1 = y$.
7. Ha $(j > 0)$, akkor térjünk vissza a 3. lépésre.
8. Ha $(i \neq 1)$, akkor adjuk vissza a HIBA jelzést.
9. Adjuk vissza a OK jelzést és $y_2\ mod\ a$-t.

5.7. Bitek és egészek közti átváltás

Egy n hosszú bitsorozat $\{x_1, \ldots, x_n\}$ a következő szabály szerint alakítjuk egész számmá $\{x_1, \ldots, x_n\} \rightarrow (x_1 * 2^{n-1}) + (x_2 * 2^{n-2}) + \cdots + (x_{n-1} * 2) + x_n$.

Input:

1. $b_1, b_2, \ldots, b_n$: A konvertálni kívánt bitsorozat.

Output:

1. C: Az egész szám, ami a bitsorozatot reprezentálja.

Folyamat:

1. Legyenek $(b_1, b_2, \ldots, b_n)$ b bitjei balról jobbra olvasva.
2. $C = \sum_{i=1}^{n} 2^{n-i} * b_i$
3. Adjuk vissza C-t.

C bináris hossza a legkisebb egész n, amelyre $C < 2^n$.

6. fejezet

Mellékletek [6]

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA[21]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Az elektromos aláírás algoritmusok biztonsági ereje

Security Strength	Digital Signatures and hash-only applications	HMAC[22], Key Derivation Functions[23], Random Number Generation[24]
≤ 80	SHA-1[25]	
112	SHA-224, SHA-512/224, SHA3-224	
128	SHA-256, SHA-512/256, SHA3-256	SHA-1
192	SHA-384, SHA3-384	SHA-224, SHA-512/224
≥ 256	SHA-512, SHA3-512	SHA-256, SHA-512/256, SHA-384, SHA-512, SHA3-512

A hash függvények biztonsági ereje

Irodalomjegyzék

[1] https://e-szigno.hu/tudasbazis/elektronikus-alairas.html, letöltés ideje: 2019.12.11.

[2] National Institute of Standards and Technology: *Digital Signature Standard (DSS)*, 2013, letöltés helye: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, letöltés ideje: 2019.11.12.

[3] Joan Gómez: *Matematikusok, kémek, hekkerek*, 2019., London

[4] National Institute of Standards and Technology: *Secure Hash Standard (SHS)*, 2015, letöltés helye: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf, letöltés ideje: 2019.11.12.

[5] Stevens, Bursztein, Karpman, Albertini, Markov: *The first collision for full SHA-1*, 2017, letöltés helye: https://shattered.io/static/shattered.pdf, letöltés ideje: 2019.12.11.

[6] National Institute of Standards and Technology: *Recommendation for Key Management*, 2016,
letöltés helye: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-57pt1r4.pdf, letöltés ideje: 2019.12.11.

[7] Az Európai Parlament és az Európai Unió Tanácsa: *AZ EURÓPAI PARLAMENT ÉS A TANÁCS 910/2014/EU RENDELETE*, 2014, letöltés helye:
https://eur-lex.europa.eu/legal-content/HU/TXT/?uri=CELEX%3A32014R0910, letöltés ideje: 2019.12.31.