Titokmegosztások

Veres Diána

Matematika BSc

Szakdolgozat

Témavezető:

Villányi Viktória Ildikó

Adjunktus

Operációkutatási Tanszék

Eötvös Loránd Tudományegyetem
Természettudományi Kar

Budapest,
2019

Tartalomjegyzék

1. Bevezetés **2**

2. A titokmegosztás alapjai **3**

2.1. Definíciók 3

2.2. Optikai titokmegosztás 5

2.3. Shamir-féle titokmegosztás 8

2.3.1. Az egyszerű (k, n) Küszöbséma: 10

2.3.2. Néhány hasznos tulajdonsága a (k, n) küszöbértékű sémának 10

2.3.3. Használat 11

2.4. Blakley titokmegosztás sémája 13

2.4.1. Osztó fázis 13

2.4.2. Részeket egyesítő fázis 13

3. Proaktív titokmegosztás, avagy hogyan küzdjünk meg a folyamatos kiszivárogtatásokkal **16**

3.1. Bevezetés 16

3.2. A modell és feltevései 18

3.2.1. Szerverek és a kommunikációs modell 18

3.2.2. Időszakok és frissítési fázisok 18

3.2.3. A mobil ellenfél modell 18

3.2.4. A proaktív titokmegosztás biztonsága 19

3.3. Kriptográfiai eszközök 21

3.3.1. SHAMIR-féle titokmegosztás 21

3.3.2. Ellenőrizhető titokmegosztás(VSS-verifiable secret sharing) 21

3.3.3. FELDMAN-féle VSS 21

3.3.4. Nyilvános-kulcs titkosítások és aláírások 22

3.4. Időszakos frissítések 23

3.4.1. Kezdeti beállítás: Fekete-doboz nyilvános kulcs feltételezések 23

3.4.2. A titokmegosztás inicializálása 23
3.4.3. A részek frissítése 24
3.4.4. A részek frissítési protokollja aktív támadók jelenlétében . 25
3.4.5. A vádak megoldása 26
3.4.6. A fentebb említett RSA bemutatása 28
3.4.7. Biztonsági tulajdonságok 30
3.5. Rész helyreállítási rendszer 31
3.5.1. Eltulajdonított részek érzékelése 31
3.5.2. Alapvető részhelyreállítás protokollja 31
3.5.3. A teljes részhelyreállítás protokollja 32
3.6. Proaktív titokmegosztás: ötvözött protokoll 33
3.7. Köszönetnyilvánítás 34

1. fejezet

Bevezetés

Sokat gondolkodtam azon, hogy milyen téma álljon a szakdolgozatom középpontjában, a témát, amely már az első pillanattól kezdve szimpatikus volt számomra, a témavezetőm, Villányi Viktória ajánlotta nekem. A kriptográfia egyik fontos részterületére a Titokmegosztásra esett a választás, amelynek a lényege a következő: van egy titkunk, amit csak úgy tudunk rekonstruálni, ha több ember kölcsönösen együttműködik hozzá. Egy örök klasszikust szeretnék példaként felhozni, ami a következő: kétféle zár is van egy páncélszekrényhez, és ezekhez a zárakhoz két embernél van a hozzájuk tartozó két különböző kulcs. Egyedül egyikük sem tudja a páncélszekrényből a pénzt kivenni, viszont együtt természetesen hozzáférnek. Adi Shamir és George Blakley a titokmegosztást egymástól függetlenül 1979-ben találták fel. Az általuk használt módszerben egy úgynevezett osztó az n játékosnak (résztvevőknek) ad egy-egy darabot a titokból, amikkel a játékosok csak bizonyos feltételek teljesülése mellett tudják rekonstruálni a titkot. Ez úgy érhető el, hogy az n résztvevő közül bármely k vagy több játékos közösen meg tudja határozni a titkot, viszont bármely $k-1$ vagy annál kevesebb ne rendelkezzen semmilyen információval se a titokról. Ezt nevezzük (k, n) küszöbértékű rendszernek. Szakdolgozatomban ezeket a módszereket szeretném szemléltetni. Az első fejezetekben ismertetem magát a problémakört, az alapfogalmakat, valamint az optikai, a Shamir-féle és a Blakley-féle titokmegosztási sémák működését. Ezután megvizsgálom az elvárásokat a titokmegosztással szemben, valamint azt, hogy az előbbiekben megismert módszerek teljesítik-e ezt. Végezetül bemutatom a proaktív titokmegosztást, melyhez a [8] cikket használtam.

2. fejezet

A titokmegosztás alapjai

2.1. Definíciók

Legyen α a résztvevők véges halmaza: $P = \{v_1, v_2, v_3, \cdots, v_n\}$ A titokmegosztás $n+1$ darab (közös eloszlással rendelkező) valószínűségi változó. Ξ jelölje a titok értékét, az n darab ξ_i pedig a titokrészeket, ahol az egyes ξ_i értékeket egyedül v_i résztvevő ismeri.

2.1.1. Definíció. Ekkor $(\xi_1, \xi_2, \ldots, \xi_n)$ az α-t megvalósító tökéletes titokmegosztás, ha

- $\forall A \in \alpha$ esetén $\{\xi_a : a \in A\}$ meghatározza ξ-t (ekkor A-t kvalifikált részhalmaznak nevezzük),
- $\forall B \in \alpha$ esetén $\{\xi_b : b \notin B\}$ független ξ-től (ekkor B nem kvalifikált részhalmaz).

[4]

2.1.2. Definíció. A résztvevők részhalmazainak egy α családja elérési struktúra, ha

- $A \in \alpha$ és$A \subseteq B$, akkor $B \in \alpha$

[5]

2.1.3. Tétel. *Minden α-elérési struktúrához létezik titokmegosztási rendszer.*

[5] Ezt a tételt Ito, Saito és Nishizeki 1987-ben, illetve Banaloh és Leichter 1988-ban fogalmazta meg.

2.1.4. Definíció ((n, k) küszöbséma)**.** Tegyük fel, hogy az osztó a titkot n részre osztja, és teljesülnek a következő feltételek:

- bármely k vagy több titokrész elegendő a titok kiszámításához
- bármely $k - 1$ vagy kevesebb titokrész független a titok értékétől.

[1]

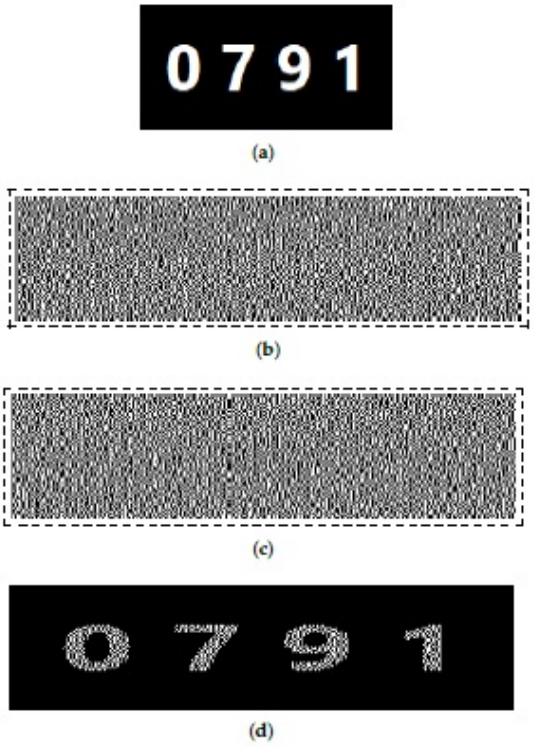
2.1.5. Definíció. Ívnek nevezzük a $PG(n, q)$ olyan legalább $n + 1$ elemű ponthalmazát, amelynek nincs $n + 1$ pontja a hipersíkban.

[3]

2.2. Optikai titokmegosztás

Jelöljük ω-val a titkot, melyből ω_i részeket, vagy más néven árnyékokat készítünk. Ezen árnyékok egy $W = \{\omega_1, \omega_2, \ldots, \omega_k\}$ halmazát hasonlóan az alap titokmegosztási esethez, k-küszöb sémának nevezzük, ha ω az W bármely k eleméből visszaállítható, de semelyik $k-1$-ből még nem. Az optikai titokmegosztás (Visual Cryptography) különösebben hangsúlyos 1987-ben lett Kafri és Keren által. Egy olyan egyszerű technikát írtak le a $2D$ képek titkosítására, amelyben véletlen rácshálókat használtak. A dekódoláshoz a két ábra szuperpozícióját használták. Ezeket az ábrákat gyakran szokás részeknek nevezni. 1994-ben, Naor and Shamir népszerűsített egy analóg optikai titokmegosztási módszert. A munkáik leírnak egy pixel-alapú titkosítási technikát, amely közvetlenül dekódolható az emberi képrendszer által. Egyetlen kép kerül az érdeklődés középpontjába. (úgy, mint bemeneti titkos kép). Ezután két kulcsábrát hozunk létre a titkos képből. A létrehozott kulcsábrákat elküldik a címzettnek két különböző kommunikációs csatornát használva a biztonság érdekében.(pl, MMS-ben és email-ben, külön-külön). Kommunikációs szempontból az egyik visszautasíthatja. Miután a két kulcsábrát a címzettek összegyűjtik, az eredeti titkos kép dekódolhatóvá válik a kulcsábrák szuperpozíciójának vizuális olvasatából. Figyelembe kell vennünk azt, hogy két kulcsábrát kell létrehozni. Vegyünk egy üres fóliát, és sűrítsük a rácsozást: minden oszlopot osszunk fel két részre. Ezzel az eredeti rácsozás minden cellája két cellára bomlott: egy bal- és egy jobboldalira. Majd ezek után válasszuk ki ezek közül véletlenszerűen az egyiket, és $\frac{1}{2}$ valószínűséggel színezzük a bal oldalát, illetve $\frac{1}{2}$ valószínűséggel a jobb oldalát feketére! Tegyük ezt meg minden cellára: ez lesz a ω_1 árnyék (2.1 b).Vegyünk még egy üres fóliát, amin szintén a sűrűbb rácsozás van. Ezen is töltsük ki minden cella bal vagy a jobb felét a következő szerint: ha ω-ben egy mező ki volt töltve, és ω_1-ben a bal felét satíroztuk be, akkor most a jobb felét fessük feketére (vagy fordítva!)Ha ω-ban egy mező üres volt, akkor a neki megfelelő mezőt ugyanúgy töltsük ki, mint ahogy azt ω_1-ben tettük! Az így nyert fólia lesz a ω_2 árnyék (2.1 c). Az így kapott árnyékok semmilyen információt nem tartalmaznak magáról ω-ról, mert ω_1 egy véletlenszerűen satírozott fólia, ω_2-re pedig nyugodtan tekinthetünk úgy, mint az ω és ω_1 $(mod 2)$ vett összegére. Így tehát lehetetlen belőlük visszaállítani ω-t, de a kettő együtt éppen ω-vá egészíti ki egymást. Ha a két fóliát egymásra helyezzük, és úgy hunyorítunk, hogy a ritkább rácsozást érzékeljük, akkor kétféle mezőt látunk majd: lesznek szürke mezők (amelyeknek csak az egyik fele van kitöltve),

és lesznek fekete mezők (amelyek teljesen ki vannak töltve). Így tehát visszanyerhető az eredeti üzenet (2.1 d). Mutatok egy példát az optikai titokmegosztás használatára: Az optikai titokmegosztás másik formája animációkat von be vagy

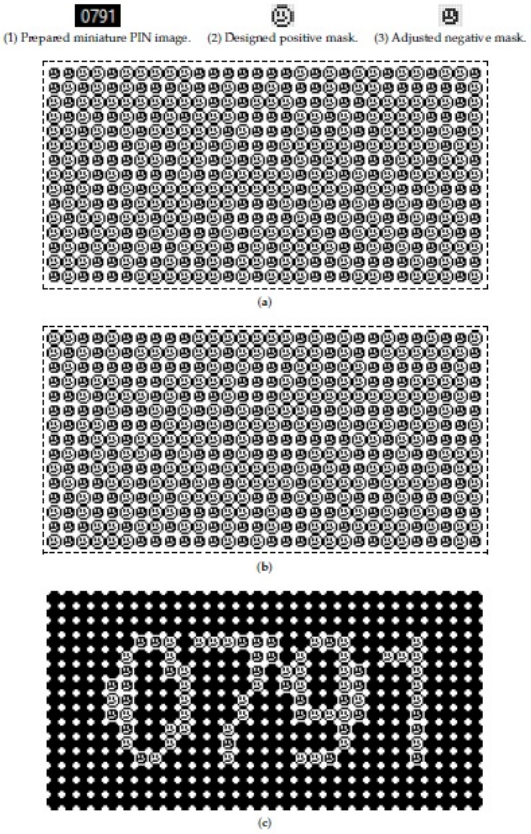


2.1. ábra.

animált GIF ábrákat. Több, rokonságban áll az optikai titokmegosztás ábrázoló típusával, Borchert javasolta a felosztás-alapú optikai titkosítást a digitális ábrák ismertetésére. A pixel alapú VC alapelve az volt, hogy kiterjesztette az álcát vízszintes és függőleges részekre osztással, ezzel létrehozva a 7-rész-kijelzést (Seven.Segment-Display, röviden SSD). Lehetséges előnyei a rész alapú VC-nek a klasszikus pixel alapúval összehasonlítva:

- egy egyszerűbb elrendezése a szuperpozíciós kulcsábrának, a részálarcok képesek megkönnyíteni a titkosított kép kézi beállítását;
- az emberi szem által egyszerűbb felismerni a szimbólumokat.

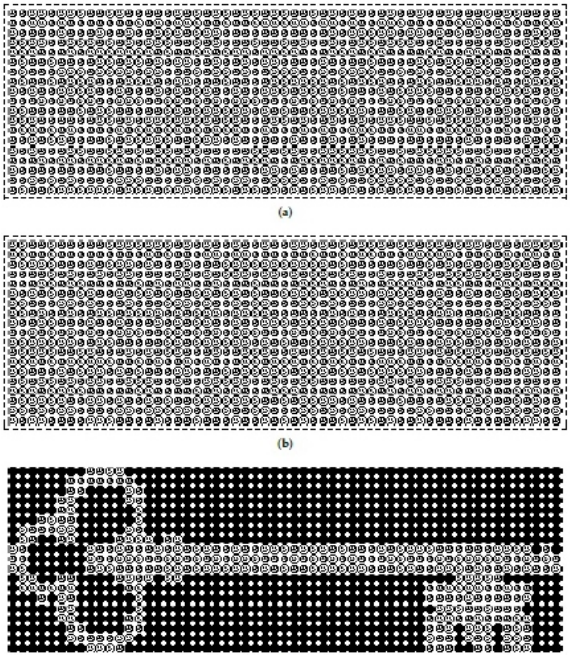
A legjobb tudásunk szerint, az optikai titkosítási módszerek nem javasolják használni azt a technikát, ahol a részeket közvetlenül a kommunikáció középpontjába helyezik (beágyazott rejtett információ). Valóban, a részek többsége úgy néz



2.2. ábra.

ki, mint egy rácshálója a véletlenszerű fekete és fehér pixeleknek, nincs közvetlen jelentésük.

Összefoglalva, azok a részek, amik a két piktogram alapú fedésből lettek létrehozva, esztétikusabbak a hagyományosnál. Egyfelől, egy vizuális reklámüzenetet közöl, másrészt optikailag kellemes megjelenést nyújt. A javasolt testreszabása a részek megjelenítésének a logók, ikonok vagy képírásjelek választékát használja papíron vagy digitálisan. Erre mutatok 2 példát: [7]



2.3. ábra.

2.3. Shamir-féle titokmegosztás

Ebben a szakaszban [1]-re támaszkodva meg fogom mutatni, hogy hogyan lehet egy D titkot n részre osztani úgy, hogy D-t könnyen rekonstruálni tudjuk bármely k darabból, de ha csak $k-1$ darabot ismerünk, akkor semmilyen információval ne rendelkezzünk az eredeti D-ről. Ez a technika teszi lehetővé a robusztus kulcs szerkezetét olyan kriptográfiai rendszerekre, amelyek biztosan és megbízhatóan működnek még akkor is, amikor különféle balszerencsék semmisítik meg a titokrészek felét, és a biztonsági megszegések felfedik a megmaradó darabok közül egy kivételével az összeset. [9]-ben, Liu a következő problémát boncolgatja: Tizenegy tudós dolgozik egy titkos projekten. Szeretnék a dokumentumokat egy széfben úgy tárolni, hogy csak akkor lehessen kinyitni azt, ha a tudósok közül legalább hat jelen van. Mi a legkisebb szükséges zárak száma? Mennyi a legkevesebb kulcs a zárakhoz, amelyet minden tudósnak magával kell vinnie? Általánosítva a problémát: legyen D a titok. A célunk az, hogy D-t n részre osztjuk $D_1, D_2, \ldots, D_n$ oly módon, hogy:

- amennyiben legalább k-t ismerünk a D_i részek közül, könnyen kiszámíthatóvá válik D,
- de ha csak $k-1$ vagy kevesebb részről van információnk, az teljesen isme-

retlenül hagyja D-t.

Egy ilyen sémát (k, n) küszöbsémának nevezünk. A hatékony küszöbsémák nagyon segítőkészek lehetnek a titkosító kulcsok kezelésében. Azért, hogy meg tudjuk védeni az adatainkat, titkosíthatjuk őket, azonban ahhoz, hogy megvédhessük a titkosítási kulcsokat, szükségünk van különböző módszerekre (további titkosítások inkább megváltoztatják a problémát mintsem megoldják azt). A legbiztonságosabb kulcskezelő módszer a kulcsot egy egyedülálló, jól őrzött helyen tartja (ilyen egy számítógép, az emberi agy, vagy egy széf). Ez a módszer azonban eléggé megbízhatatlan, hiszen a számítógép meghibásodása, hirtelen halál vagy egy szabotázs az információt hozzáférhetetlenné képes tenni. Erre egy nyilvánvaló megoldás, hogy a kulcsokról különböző másolatokat kell tárolni különféle helyeken, de ez növeli a biztonsági rések kialakulásának esélyét. Egy (k, n) küszöbsémát használva, ahol $n = 2 \cdot k - 1$ egy nagyon robusztus kulcsirányító tervet eredményez. Visszakaphatjuk az eredeti kulcsot akár $\lfloor \frac{n}{2} \rfloor = k - 1$ megsemmisített rész esetén is, de az ellenfeleink nem tudják visszaállítani a kulcsot még akkor sem, ha a biztonsági rések $\lfloor \frac{n}{2} \rfloor = k - 1$ részt fednek fel a megmaradó k-ból. Más alkalmazásokban a csere nem a megbízhatóság és a titoktartás között, hanem a biztonság és kényelmes használhatóság között van. Tekintsünk például egy vállalatot, amely digitálisan írja alá a csekkjeit. Ha minden végrehajtónak adunk egy másolatot a cég titkos aláírási kulcsáról, a rendszer kényelmes, de könnyen vissza lehet vele élni. Ha a vállalat vezetői közül mindegyiknek az együttműködése szükséges ahhoz, hogy aláírják a csekkeket, a rendszer biztonságos, de kényelmetlen. Az általános megoldás legalább három aláírást igényel csekkenként, ami könnyen megvalósítható egy $(3, n)$ küszöbsémával. Minden végrehajtó kap egy kis mágneskártyát, amely tartalmaz egy D_i részt, és a vállalat aláírást generáló eszköze elfogadja ezek közülük bármelyik hármat ahhoz, hogy létrehozzon (és később megsemmisítsen) egy ideiglenes másolatot a D aktuális aláírási kulcsról. Az eszköz nem tartalmaz semmilyen titkos információt és így azt nem kell megvédeni. Egy áruló vezetőnek legalább két bűntársra van szüksége ahhoz, hogy hamisítani tudja a cég aláírását ebben a rendszerben. A küszöbsémák ideálisak azokhoz az alkalmazásokhoz, amelyekben a gyanús, ellentétes érdekekkel rendelkező egyedek csoportjának együtt kell, hogy működjenek. Ideális esetben azt szeretnénk, hogy az együttműködés kölcsönön beleegyezésen alapuljon, de a vétójog, amit a szerkezet a tagoknak ad, megbéníthatja a csoport tevékenységeit.

2.3.1. Az egyszerű (k, n) Küszöbséma:

A módszerünk a polinom interpoláción alapul: Adott k pont a 2-dimenziós síkban $(x_1, y_1), \ldots, (x_k, y_k)$ különböző x_i-kel, adott továbbá pontosan egy $k-1$-ed fokú $q(x)$ polinom, ahol $q(x) = y_i$ minden i-re. Feltehetjük, hogy D egy szám (vagy azzá tudjuk alakítani). Ahhoz, hogy D_i részekre bontsuk, felveszünk véletlenszerűen egy $k-1$-ed fokú $q(x) = a_0 + a_1 \cdot x + \cdots + a_{k-1} \cdot x^{k-1}$ polinomot, ahol $a_o = D$, és megoldjuk azt: $D_1 = q(1), \ldots, D_i = q(i), \ldots, D_n = q(n)$. D_i bármely k részhalmazára meg tudjuk találni $q(x)$ együtthatóit interpoláció útján, ezután pedig meg tudjuk adni $D = q(0)$-t is. Másrészt, ha csak $k-1$ értéket ismerünk, nem elegendő ahhoz, hogy kiszámolhassuk D-t.

2.3.2. Néhány hasznos tulajdonsága a (k, n) küszöbértékű sémának

- Egyetlen egy darab mérete sem lehet nagyobb az eredeti titok méreténél,
- Rögzített k mellett a D_i titokrészek dinamikusan hozzáadhatók vagy törölhetőek anélkül, hogy annak hatása lenne a többi D_i darabra.(Egy darab csak akkor törlődik, amikor az teljesen hozzáférhetetlenné válik.)
- Egyszerűen lehet változtatni ezeket a D_i darabokat anélkül, hogy az eredeti D értéke megváltozna - ehhez mindösszesen annyira van szükségünk, hogy egy új $q(x)$ polinomot választunk, méghozzá olyat, melynek foka megegyezik az eredeti polinomunk fokával. Ez a típusú gyakori változtatás jelentősen javíthatja a biztonságot, a titokrészek a biztonsági réseken csak akkor tudnak kiszivárogni, ha a megszerzett adatok ugyanabból a $q(x)$ polinomból származnak.
- A polinom értékeinek rendezett, véges n-es listáját a D_i részekként használva kaphatunk egy hierarchikus rendezést, amiben a részek számának szükséges meghatároznia D-t, fontosságuktól függően. Például ha adunk egy cég elnökének $q(x)$ értékei közül hármat, az alelnökének kettőt és minden felső vezetőnek egyet, ezután a $(3, n)$ küszöbérték rendszer lehetővé teszi az ellenőrzését annak, hogy szerződhet bármely 3 felső vezető, vagy bármelyik 2 felső vezető, ha az egyikőjük az alelnök, illetve az elnök egyedül.

2.3.3. Használat

Az alábbi példa illusztrálja az alapötletet. Megjegyzendő, hogy a számítások integer számítások. Ezért ez a példa nem ad tökéletes biztonságot és nem a teljes valós példája a Shamir-féle sémának. Legyen a titkunk 2345 ($D = 2345$). Nézzük meg a szétosztást: Fel szeretnénk bontani 6 részre ($n = 6$), ahol 3 rész ($k = 3$) elég a titok helyre állításához. (Az ($n = 6$) meghatározza, hogy 6 pontot kell kiszámolnunk és szétosztanunk, míg a ($k = 3$) meghatározza, hogy az egyenlet ($k - 1 = 2$) fokú lesz, és hogy ($k - 1 = 2$) tetszőlegesen választott számra van szükségünk.) A példában véletlennek a következő 2 számot válasszuk: $180, 86$. ($a_1 = 180; a_2 = 86$) A polinomunk, ami a titokrészeket létrehozza (a pontokat) a következő:

$$f(x) = 2345 + 180x + 86x^2 \tag{2.1}$$

A létrehozott 6 pont a következő:

$$(1, 2611); (2, 3049); (3, 3659); (4, 4441); (5, 5395); (6, 6521)$$

Minden birtokos kap egy pontot (az x és $f(x)$ értékeket is). Ezután következhet az összeállítás: Az összeállításhoz 3 pont már elég. Legyenek ezek a pontok:

$$(x_0, y_0) = (1, 2611); (x_1, y_1) = (3, 3659); (x_2, y_2) = (5, 5395).$$

Határozzuk meg a Lagrange polinomokat: A Lagrange polinomok meghatározása a következő:

$$lj(x) := \prod_{0 \le i \le k, i \ne j} \frac{(x - x_i)}{(x_j - x_i)} =$$
$$\frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \cdot \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)},$$

azaz a produktumot kell számítani (össze kell szorozni) az $\frac{(x-x_i)}{(x_j-x_i)}$ tagokat egymással, ahol azt a tagot, ahol $i = j$ ki kell hagyni, mert az osztás egyébként értelmetlen lenne. Behelyettesítve $j = 0, 1, 2$ esetekben a következőt kapjuk:

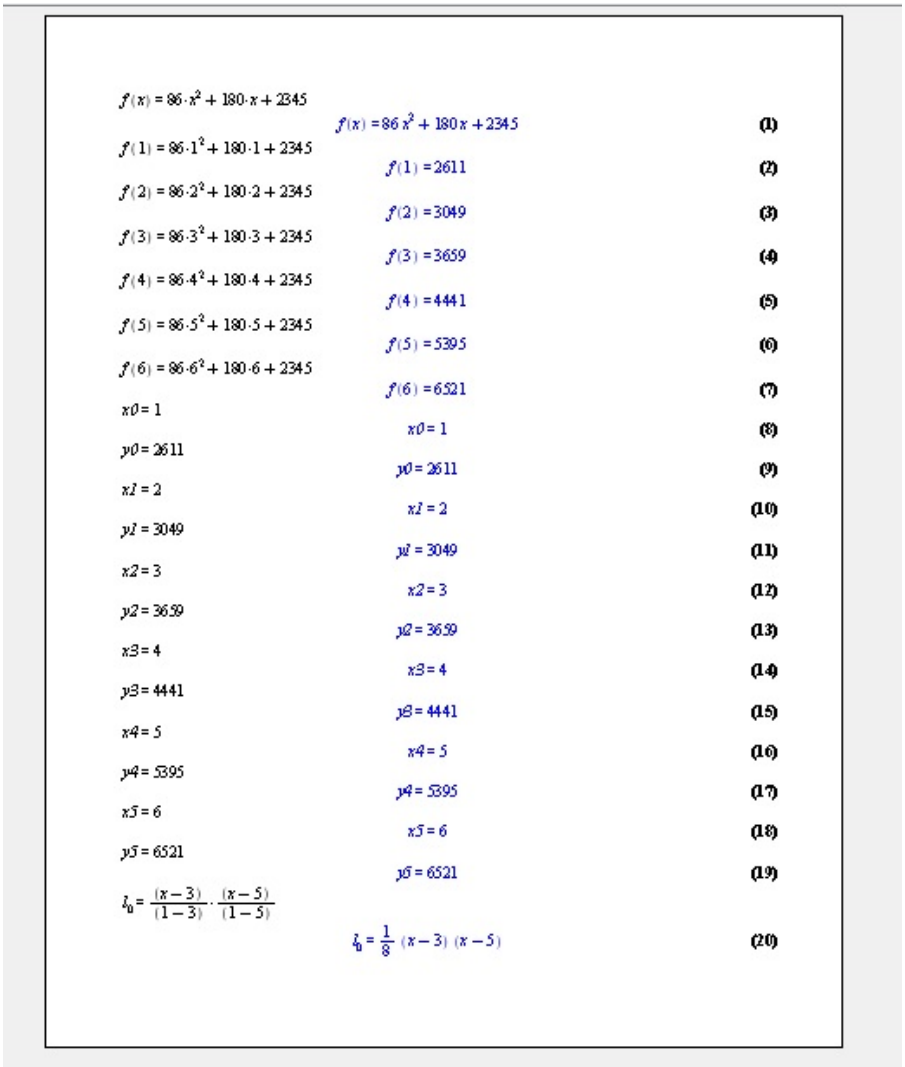
$$l_0(x) = \frac{(x - x_1)}{(x_0 - x_1)} \cdot \frac{(x - x_2)}{(x_0 - x_2)} = \frac{x - 3}{1 - 3} \cdot \frac{x - 5}{1 - 5} = \frac{1}{8} \cdot x^2 - x + \frac{15}{8}\cdot$$
$$l_1(x) = \frac{(x - x_0)}{(x_1 - x_0)} \cdot \frac{(x - x_2)}{(x_1 - x_2)} = \frac{x - 1}{3 - 1} \cdot \frac{x - 5}{3 - 5} = -\frac{1}{4} \cdot x^2 + \frac{3}{2} \cdot x - \frac{5}{4}$$
$$l_2(x) = \frac{(x - x_0)}{(x_2 - x_0)} \cdot \frac{(x - x_1)}{(x_2 - x_1)} = \frac{x - 1}{5 - 1} \cdot \frac{x - 3}{5 - 3} = \frac{1}{8} \cdot x^2 - \frac{1}{2} \cdot x + \frac{3}{8}$$

Mivel az interpolációs polinom a következő:

$$f(x) = \sum_{j=0}^{2} y_j \cdot l_j(x) = 2611 \cdot (\frac{1}{8}x^2 - x + \frac{15}{8}) + 3659 \cdot$$
$$(-\frac{1}{4}x^2 + \frac{3}{2} \cdot x - \frac{5}{4}) + 5395 \cdot (\frac{1}{8}x^2 - \frac{1}{2} \cdot x + \frac{3}{8}) =$$
$$= 2345 + 180 \cdot x + 86 \cdot x^2$$

Látható, hogy a titok a szabad együttható, azaz $D = 2345$, és a visszaállítás megtörtént.

2.3.1. *Megjegyzés.* A fenti számításokat MAPLE programmal a következőkben bemutatva ellenőriztem le.



$f(x) = 86 \cdot x^2 + 180 \cdot x + 2345$

$f(1) = 86 \cdot 1^2 + 180 \cdot 1 + 2345$

$f(2) = 86 \cdot 2^2 + 180 \cdot 2 + 2345$

$f(3) = 86 \cdot 3^2 + 180 \cdot 3 + 2345$

$f(4) = 86 \cdot 4^2 + 180 \cdot 4 + 2345$

$f(5) = 86 \cdot 5^2 + 180 \cdot 5 + 2345$

$f(6) = 86 \cdot 6^2 + 180 \cdot 6 + 2345$

$x0 = 1$

$y0 = 2611$

$x1 = 2$

$y1 = 3049$

$x2 = 3$

$y2 = 3659$

$x3 = 4$

$y3 = 4441$

$x4 = 5$

$y4 = 5395$

$x5 = 6$

$y5 = 6521$

$l_0 = \frac{(x-3)}{(1-3)} \cdot \frac{(x-5)}{(1-5)}$

$f(x) = 86\,x^2 + 180\,x + 2345$ (1)

$f(1) = 2611$ (2)

$f(2) = 3049$ (3)

$f(3) = 3659$ (4)

$f(4) = 4441$ (5)

$f(5) = 5395$ (6)

$f(6) = 6521$ (7)

$x0 = 1$ (8)

$y0 = 2611$ (9)

$x1 = 2$ (10)

$y1 = 3049$ (11)

$x2 = 3$ (12)

$y2 = 3659$ (13)

$x3 = 4$ (14)

$y3 = 4441$ (15)

$x4 = 5$ (16)

$y4 = 5395$ (17)

$x5 = 6$ (18)

$y5 = 6521$ (19)

$l_0 = \frac{1}{8}\,(x-3)\,(x-5)$ (20)

2.4. ábra.

2.4. Blakley titokmegosztás sémája

[11] A Blakley-féle titokmegosztás [3] a hipersík geometriát használja annak érdekében, hogy megoldja a problémát. A titok egy pont a t-dimenziós térben és az n részek hasonlóak azokhoz a hipersíkokhoz, amelyek átmennek ezen a ponton. Egy hasonló hipersík a t-dimenzionális térben, amely az F testbeli koordinátákkal rendelkezik, le tud írni egy lineáris egyenletet, amely a következőképpen néz ki:

$$a_1x_1 + a_2x_2 + \cdots + a_tx_t = b.$$

Metszéspontot találhatunk bármely t hipersík metszetében. A titok lehet akármelyik koordinátája a metszéspontnak vagy a koordináták bármely függvénye. Tekintsük a titkot a metszéspont első koordinátájának.

2.4.1. Osztó fázis

Legyen m egy prímszám és legyen $\mathbf{F} = \mathbf{Z}_m$ az a test, amelyben dolgozunk. Az osztó generál egy titkos x pontot $\mathbf{F}^t$-ben, ahol az első koordináta $x[1]$-et állítjuk be a titok értékeként, és állítsuk be a többi koordináta értékét véletlenszerűen az $\mathbf{F}$ testből. Az i. felhasználó kap egy hipersík egyenletet az $\mathbf{F}$ felett,

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{it}x_t = y_i. \tag{2.2}$$

A (t, n) küszöbrendszert tekinthetjük n hipersíkbeli egyenletként, és ennélfogva lesz egy $n \times t$-s lineáris rendszerünk.

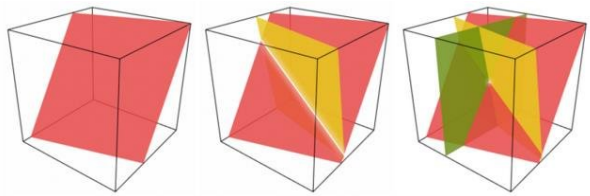
$$Ax = y. \tag{2.3}$$

Ezután az osztó tovább küldi az y_i titok értékét az $a_{i1}, \cdots, a_{it}$ az i. felhasználónak. Az a_{ij} együttható nem érzékeny és nyilvánossá tehető, ha szükséges.

2.4.2. Részeket egyesítő fázis

A részeket egyesítő lépés egyszerűen megtalálja a megoldást a lineáris rendszer egyenleteire. Feltételezve hogy az $S = \{i_1, \cdots, i_t\}$ felhasználók koalíciója összejön. Egy A_S mátrix alakot alkotnak használva a hipersík egyenleteket és azok megoldásait:

$$A_Sx = y_S, \tag{2.4}$$



2.5. ábra.

ahol y_S egy vektora a titokrészeknek, amiket a felhasználók birtokolnak. A titok megtalálható a megoldás első koordinátájaként.

3. fejezet

Proaktív titokmegosztás, avagy hogyan küzdjünk meg a folyamatos kiszivárogtatásokkal

3.1. Bevezetés

Az érzékeny adatok számára a titokmegosztási rendszerek alapvető védelmi eszközt jelentenek, amely arra kényszeríti az ellenséget, hogy egyszerre több helyet támadjon meg annak érdekében, hogy az információhoz hozzáférjen vagy megsemmisítse azokat. Kiváltképpen igaz ez, egy $(k+1, n)$ küszöbértékrendszerben. Az ellenfélnek ebben az esetben több mint k helyet kell megtámadnia ahhoz, hogy megtanulja a titkot, és el kell rontani legalább $n-k$ részt belőle ahhoz, hogy az információ megsemmisüljön. Azonban az ellenfél a titok teljes élettartama alatt indíthat támadásokat. Fokozatos és pillanatnyi behatolások is lehetségesek egy hosszú időtartam alatt az ellenség számára. Ezért a hosszú élettartamú és érzékeny titkok esetében lehetséges, hogy a hagyományos titokmegosztás által nyújtott védelem nem elegendő. Egy természetes védekezés, hogy rendszeresen frissítsük a titokrészeket, habár ez nem mindig lehetséges. Az eredendően hosszú élettartamú információ esetére ez vonatkozik, ilyenek a titkosítási kulcsok (pl. aláírás / tanúsítási kulcsok), adatfájlok (pl. Orvosi feljegyzések), jogi dokumentumok (pl. végrendelet vagy szerződés), tulajdonosi kereskedelmi titok (pl. a Coca-Cola recept), és így tovább. Így ami valójában szükséges ahhoz, hogy megvédjük az információ titkosságát, az, hogy rendszeresen frissítsük a részeket anélkül, hogy a titok megváltozna. Ezt megtehetjük oly módon, hogy az ellenfél által

az egyes részekről megismert bármely információk elavulnak a frissítések után. Hasonlóképpen, hogy elkerüljék az információk fokozatos megsemmisítését a részek sérülése révén, szükséges az elveszett vagy megsérült részeket rendszeresen visszaszerezni anélkül, hogy a már behajtott részek titkossága veszélybe kerülne. A proaktív titokmegosztás legfontosabb tulajdonságai:

- Ebben a megközelítésben a titok élettartama időtartamra oszlik (például egy napra, egy hétre stb.).
- Minden egyes időszak elején a résztvevők egy interaktív frissítési protokollt folytatnak, amely után egy teljesen új részt kapnak ugyanahhoz a titokhoz.
- Az előző részek elavultak, és biztonságosan törölhetőek.

Következésképpen egy $(k+1, n)$ proaktív küszöbértékrendszer esetén, annak az ellenfélnek, aki megpróbál hozzájutni a titok szükséges $k+1$ részéhez, meg kell támadnia azt az egyes időszakok alatt. Azonban fokozatosan kompromitál $k+1$ helyet a teljes titok élettartam alatt. Hasonló módon, a titok megsemmisítéséhez szükséges az ellenfélnek elrontani $n-k$ részt egyetlen időszakban. Jegyezzük meg, hogy ebben a beállításban az ellenfél mobil, és többször is betörhet minden kiszolgáló esetén. Mindazonáltal nem egyezhet a titkot, ha bármelyik időintervallumban kevesebb mint k helyen jut csak be. A megoldásunk a proaktív titokmegosztási problémára akár $k = \frac{n}{2} - 1$ sérült részeket is támogathat bármikor. Feltételezett a biztonságos titkosítási és aláírási funkciók létezése, épp úgy, ahogy a homomorf funkciókon alapuló ellenőrizhető titokmegosztási rendszer (Verifiable Secret Sharing - VSS) biztonsága. A rendszer szintjén feltételezünk egy közvetítő csatornát és szinkronitást (mint a VSS-ben). A pontos modellt és feltevéseket a későbbiekben részletezem majd. A mobil ellenség beállítást eredetileg a biztonságos rendszerek kontextusában mutatta be Ostrovsky és Yung, melyben az "általánosan elosztott funkcióértékelés" elméleti beállítására összpontosítottak. Ez a megoldás engedélyezte a nagy (polinom) redundanciát az adott rendszerben (a redundancia az összes n kiszolgáló aránya a k küszöbbel szemben, amely az egyszerre hibás szerverek száma). Ezt a modellt később sokkal praktikusabban beállítással használta Canetti és Herzberg. Alkalmazások: A proaktív titokmegosztás számos alkalmazással rendelkezik, elsősorban olyan adatok karbantartásával, amelyek hosszú élettartamúak olyan esetekben, ahol a rendelkezésre állás és a titoktartás kulcsfontosságú.

3.2. A modell és feltevései

Tekintsük egy n szerverből álló rendszert: $A = \{P_1, P_2, \ldots, P_n\}$, amely (proaktívan) egy titkos x értéket oszt meg a $(k+1, n)$-küszöbrendszer által (azaz k rész nem ad semmilyen információt a titokról, ugyanakkor $k+1$ elegendő a titok rekonstruálásához). Feltételezzük, hogy a rendszer beállításai megfelelőek és biztonságosak. A módszer célja, hogy megakadályozza az ellenfelet abban, hogy megismerje vagy megsemmisítse a titkos x-et.

3.2.1. Szerverek és a kommunikációs modell

Minden A-beli szerver egy közös C adathordozóhoz van csatlakoztatva, melyet kommunikációs csatornának nevezünk. Ezen a C-n keresztül az összes beérkező üzenet azonnal elérhető a csatlakozó tagokhoz. Feltételezzük, hogy a rendszer szinkronizált.

3.2.2. Időszakok és frissítési fázisok

Az idő időszakokra van bontva, amelyeket a közös globális óra határoz meg (pl. egy nap, egy hét, stb ...). Egy-egy időszak elején a szerverek interaktív frissítési protokollt használnak (úgy nevezett frissítési fázist). Egy frissítési fázis végén a szerverek új részeket tartanak meg az x titokból.

3.2.3. A mobil ellenfél modell

Az ellenfél meg tudja támadni a szervereket egy időintervallum bármely pillanatában. Ha a rendszert megtámadják egy frissítési fázisban, a kiszolgálót sérültnek tekintjük mindkét szomszédos időszakban is. Feltesszük, hogy az ellenség kevesebb mint k szerver felett tudta átvenni az irányítást n-ből minden egyes időszakban, ahol k-nak kisebbnek kell lennie $\frac{n}{2}$ (ez garantálja a $k+1$ helyesen működő kiszolgáló létezését minden időintervallumban). Egy kiszolgáló megtámadása a szerveren történő titoktanulás megakadályozásának bármely kombinációját jelenti, az adatai módosítását, a kiszolgáló magatartásának megváltozását, a lekapcsolást, és így tovább. Az egyszerűség kedvéért nem teszünk különbséget

a rosszindulatú meghibásodás és a szerverhibák között (pl. összeomlás, áramkimaradások, stb ...). Azt is megjegyezhetjük, hogy az ellenfél csatlakozik a C kommunikációs csatornához, ami azt jelenti, hogy ismerheti az összes üzenetet és sajátjaként használhatja azokat. Azonban nem módosíthatja a C-nek küldött azon üzeneteket, amelyek olyan kiszolgálón keresztül érkeznek, amit ő nem ellenőriz. Továbbá nem akadályozza meg, hogy a nem sérült szerver a C-nek címzett üzenetet megkapja. Emellett az ellenfél mindig ismeri a nem titkos adatokat és az összes gép által végrehajtott algoritmust.

3.2.4. A proaktív titokmegosztás biztonsága

A korábbiakban meghatározott ellenfelet feltételezve most megadom a proaktív titokmegosztási algoritmus biztonsági tulajdonságait. A formális definícióban az ellenfelet úgy modellezzük, mint egy számításhoz kötött interaktív Turing gép, amit betápláltak a titokról a nyilvánosság számára összes elérhető információval (mint pl: annak hossza, az adott altér, ahonnan a titkot megválasztják, a titok értékét egy függvényben, és így tovább), és az ellenfél által megismert információkkal (ez magába foglalja az összes nyilvános kommunikációt a kiszolgálók között, titkos információt az ezekben az időszakokban megtámadott kiszolgálókról, stb ...). Legyen K egy függvény az x titok helyére vonatkozóan. Legyen $p_0^{(K)}$ annak a valószínűsége, hogy az ellenfél helyesen kiszámolja az értéket, $K(x)$, ha be vannak táplálva a nyilvános információk a titokról, és legyen $p_1^{(K)}$ az analóg valószínűség, ha az ellenfél összegyűjtötte a további információkat mialatt lefutott a protokoll. Intuitívan a $K(x)$ függvény modellez némi ismeretet x-ről, amíg a $p_1^{(K)} - p_0^{(K)}$ különbség számolja ki ennek az ellenség által "megismert" információk a mennyiségét figyelve a protokoll végrehajtását és az aktívan behatoló kiszolgálókat.

3.2.1. Definíció. Azt mondhatjuk, hogy a proaktív titokmegosztás módszere szemantikusan biztonságos, ha a protokol futtatása során megszerzett információk mennyisége nem lehet lényegesen több, mint a nyilvános paraméterekből kinyerhető információké.

A lényegesen több pontos fogalma a fenti definícióban függ az ellenfél konkrét modelljétől. A hagyományos értelemben véve, úgy véljük, hogy annak a valószínűsége, hogy a titkot ki tudom számolni a program futtatása során kinyert

információkkal, ne legyen lényegesen nagyobb, mint az a valószínűség, amit a nyilvános paraméterekből kiszámolhatunk. Egy gondosabb modell megköti a különbséget $p_0^{(K)}$ és $p_1^{(K)}$ között mint (kis) valószínűségek explicit függvénye, amivel az ellenfél meg tudja szakítani az alapul szolgáló kriptográfiai primitíveket. Néhány esetben azért, hogy hangsúlyozzák az apriori nyilvános információ $\Pi(x)$ létezését az x titokra nézve, azt fogjuk mondani, hogy a proaktív titokmegosztási rendszer szemantikailag biztonságos a $\Pi(x)$-re. Nem csak az érdekel minket, hogy megőrizze az x titkosságát, hanem szintén garantálnia kell az elérhetőségét és a helyrehozhatóságát is.

3.2.2. Definíció. Ha a proaktív titokmegosztás módszere garantálja a titok tökéletes helyreállíthatóságát, akkor robusztusnak nevezhető.

Jegyezzük meg a proaktív titokmegosztás rendszere robusztus, feltéve, hogy bármely időszakban a megfelelően működő kiszolgálók (amelyek lehettek korábbi időintervallumokban megtámadottak is) megfelelő részeket birtokolnak (azaz azok, amelyek egyesítik a helyes x titkot), és hogy ezt a helyességet a többi kiszolgáló ellenőrizheti. Ez megköveteli a jól működő kiszolgálóktól, hogy képesek legyenek ellenőrizni azt, hogy mindegyikük helyes részeket tárol-e. Továbbá, azok, akik helyes részeket tárolnak együtt kell működniük azért, hogy visszaszerezzék azokat a részeket, amiket egyesek elvesztettek (anélkül, hogy feltárnák a visszaszerzett részeket bárki számára is, kivéve azt, aki birtokába szánják).

3.3. Kriptográfiai eszközök

3.3.1. SHAMIR-féle titokmegosztás

A titokmegosztási módszerünk a Shamir-féle titokmegosztáson alapszik. Legyen q egy prímszám, $x \in Z_q$ a megosztott titok, n a résztvevők száma (vagy részeket megtartó szervereké), és $k+1$ a rekonstruálható küszöb szám. A titok D osztója választ véletlenszerűen egy f polinomot Z_q-ból melynek foka legyen k és teljesítse az $f(0) = x$ feltételt. Ezután minden x_i-re D kiszámítja $f(i)$-t és titokban továbbítja a P_i résztvevőnek. A titok helyreállítása végrehajtható $k+1$ résztvevő rendelkezésre álló részeiből készített interpolációs polinommal, amely kiszámítja x-et.

3.3.2. Ellenőrizhető titokmegosztás(VSS-verifiable secret sharing)

A Shamir-féle módszerben egy rosszul viselkedő osztó képes következetlenül kiosztani a részeket a résztvevők között, ami miatt nem fogják tudni a titkot visszaállítani. Hogy megakadályozzuk az ilyen rosszindulatú osztói viselkedést, egy eljárást vagy protokollt kell végrehajtani, amit egy következetes osztás során a részeken keresztül ellenőrizni lehet. Az ilyen módszert nevezzük ellenőrizhető titokmegosztásnak (VSS). A megoldás Feldman és Pedersen módszerek.

3.3.3. FELDMAN-féle VSS

A teljesség érdekében röviden ismertetem Feldman tervét. Legyen p egy prím, $p = m \cdot q + 1$, ahol q is egy prím és m egy egész szám (lehetőleg egy kicsi, mint $2, 3$ vagy 4). Legyen $g \in Z_q$. Az osztó választ egy f polinomot Z_q -ból $f_0, f_1, \ldots, f_k$ együtthatókkal és $g^{f_0}, g^{f_1}, \ldots, g^{f_k}$ hozzájuk tartozó értékekkel. Ezután titokban továbbítja az $x_i = f(i) \pmod q$ értékeket P_i-nek. Minden P_i szolgáltató ellenőrzi a saját részét az alábbi egyenlet szerint:

$$g^{x_i} \stackrel{?}{=} (g^{f_0})(g^{f_1})^i (g^{f_2})^{i^2} \ldots (g^{f_k})^{i^k} \pmod p \tag{3.1}$$

Ha az egyenlet teljesül, P_i úgy közvetíti az üzenetet, hogy az a rész megfelelő. Ha minden szolgáltató helyesnek találja a részét, akkor ezután a kiosztás rész

sikeresen befejeződik. Valójában, az exponenciális függvény homomorf tulajdonságai által (azaz $g^{(a+b)} = g^a \cdot g^b$) a fenti egyenlet igaz minden $i \in \{1, \ldots, n\}$ akkor és csak akkor, ha a részek helyesen lettek kiosztva. Érdemes megjegyezni, hogy azon kívül, hogy lehetőség van ellenőrizni a részek helyes kiosztását, a nyilvános g^{x_i} értékek használhatóak a titok visszaállítása idején annak ellenőrzésére, hogy a résztvevő részek helyesek-e. Vegyük észre, hogy a Feldman-féle módszer feltárja $y = g^x \pmod p$ értéket , ahol $x = f_0$ a titok, ami meg lett osztva. Bár y-ból nem számolhatjuk ki x értékét, a segítségével részleges információkat megtudhatunk róla. Ezért a megoldásunk szemantikai biztonsága (ami a Feldman-féle sémán alapul) csak viszonylagosan kijelenthető a $g^x \pmod p$ ismeretéhez képest. Ez elfogadható azokban az esetekben, amikor g^x bárhonnan ismerhető, általában azonban szeretnénk megakadályozni, hogy kiszivárogjon a részleges információ. Ebben az esetben valaki kódolja az éppen aktuális titkot, mondjuk s-t , egy hosszabb x string-be és végrehajtjuk x-re a proaktív titokmegosztást. A kódolásnak rendelkeznie kell azzal a tulajdonsággal, hogy adott x esetén s értékéből könnyű a visszaállítás, de adott $g^x \pmod p$-ből nem származik semmilyen információ ahhoz, hogy könnyen kiszámítható legyen s. Az ilyen rendszerek létezése az exponenciális függvény bitjein alapul. Ez a probléma teljesen elkerülhető ha a Feldman-féle VSS-t felcseréljük a Pederson-módszerrel.

3.3.4. Nyilvános-kulcs titkosítások és aláírások

A megoldásunk szemantikailag biztonságos titkosítást és CMA-biztonságos aláírásokat igényel. Nem határozzuk meg vagy feltételezzük egyetlen konkrét megvalósítást sem ezek közül. Egy küldő S és fogadó R páros számára jelöljük meg ENCR (adatok) alapján az R nyilvános kulcsának a titkosítását, és a SIG (adatok) által az S titkos kulcsok aláírását.

3.4. Időszakos frissítések

Most bemutatom az időszakosan frissülő részek protokollját, ami megőrzi a titkot és ugyanakkor a múltbéli ismereteket elavulttá teszi az ellenfél számára. A megosztott titkok titkosságának garantálása mellett a rendszerünk robusztus az integritás biztosításában és a titok rendelkezésre állásában, ha legfeljebb k hibás szerver van jelen!

3.4.1. Kezdeti beállítás: Fekete-doboz nyilvános kulcs feltételezések

A kriptográfiai megoldások elosztott környezetében jellemzően megkövetelt, hogy képes legyen fenntartani magán- , és hitelesített kommunikációt a szerverek között. Ez megvalósul azon szerverek által amelyek rendelkeznek titkos-, és nyilvános kulcs párokkal (pl. RSA, EIGamal, stb ...). Azonban egy ellenfél, aki betört a kiszolgálóhoz azért, hogy hozzáférjen annak titkos kulcsaihoz, ezután képes megszemélyesíteni ezt a szervert ezeknek a titkos kulcsoknak az egész élettartama alatt. Ezért hogy a proaktív biztonság megmaradhasson, szükséges, hogy frissítsük őket időnként. Meg fogom mutatni, hogy ez hogyan valósítható meg. A bemutatás egyértelműségéhez azzal az erős feltevéssel kezdem, hogy a kiszolgálók fel vannak szerelve egy nyilvános- és egy titkos kulcs párossal oly módon, hogy a titkos kulcs az ellenfél által ne legyen elérhető vagy módosítható, még akkor sem, ha ennek az ellenségnek sikerült betörnie a szerverbe. Amíg egy ilyen betolakodó képes legális aláírásokat létrehozni, és megfejti az üzeneteket a titkos kulcsok használatával, (mint pl a fekete doboz), addig nem lesz képes elérni vagy módosítani magát a kulcsot.

3.4.2. A titokmegosztás inicializálása

Feltehetjük, hogy a kezdeti szakaszban, ahol a titok $x \in Z_q$ (q prímre) n részre van kódolva, $x_1, x_2, \ldots, x_n \in Z_q$ egy k-küszöbértékű Shamir-féle titokmegosztás: Minden $P_i, i \in \{1, \ldots, n\}$ tartja a saját x_i részét, ahol $x_i = f(i)$ néhány k-ad fokú $f(.)$ polinomra Z_q felett , $x = f(0)$. Az inicializáció után, minden időszak elején, minden helyesen működő kiszolgáló olyan frissítési fázist indít, amiben a

kiszolgálók a részekre frissítési protokollt végeznek. A t időszakban kiszámított részeket jelöljük (t) felső indexszel, azaz $x_i^{(t)}$, ahol $t = 0, 1, \ldots$; a polinomban a megfelelő részeket jelölje $f^{(t)}(.)$.

3.4.3. A részek frissítése

Ahhoz, hogy a részeket frissítsük a t. időszakban, ahol $t = 1, 2, \ldots$, alkalmazkodunk az Ostrovsky és Yung által bemutatott frissítési protokoll egyszerűsített változatához. Ahol az x titok tárolva van (elosztva) $f^{(t-1)}(0) = x$ értékeként, ahol $f^{(t-1)}(.)$ k-ad fokú polinom Z_q-ban. Frissíthetjük ezt a polinomot az által, hogy hozzáadjuk egy random k-ad fokú $\delta(.)$ polinomhoz, ahol $\delta(0) = 0$, tehát $f^{(t)}(0) = f^{(t-1)}(0) + \delta(0) = x + 0 = x$. A polinomértékelési művelet linearitása által az $x_i^{(t)} = f^{(t)}(i)$ részek frissítését kapjuk a következők szerint:

$$f^{(t)}(.) \leftarrow f^{(t-1)}(.) + \delta(.) \pmod q \tag{3.2}$$

$$\Leftrightarrow \forall i f^{(t)}(i) = f^{(t-1)}(i) + \delta(i) \pmod q \tag{3.3}$$

A rendszerünkben lesz egy $\delta(.) = (\delta_{1(.)} + \delta_{2(.)} + \cdots + \delta_{n(.)}) \pmod q$, ahol minden $\delta_{i(.)}$ polinom, $i \in \{1, \ldots, n\}, k$ fokkal és egymástól függetlenül véletlenszerűen válogatottak, valamint $\delta_i(0) = 0$ az ith szerver általi feltétel. Minden P_i kiszolgálóra a részek frissítésének protokollja, ahol $i \in \{1, \ldots, n\}$, egy t időszakban a következő:

1. P_i felvesz véletlen k számokat úgy, hogy $\delta_{\{im\}m\in\{1,\ldots,k\}}, Z_q$-ból. Ezek a számok egy $\delta_i(x) = \delta_{i1}x^1 + \delta_{i2}x^2 + \cdots + \delta_{ik}x^k \in Z_q$ polinomot definiálnak, melynek szabad együtthatója 0 és így $\delta_i(0) = 0$.

2. Minden más P_j kiszolgálóra, P_i titkosan elküldi a $u_{ij} = \delta_i(j) \pmod q$-t P_j-nek.

3. u_{ij} megfejtése után,$\forall j \in \{1, \ldots, n\}$, P_i kiszámítja annak új részét

   $$x_i^{(t)} \leftarrow x_i^{(t-1)} + (u_{1i} + u_{2i} + \cdots + u_{ni}) \pmod q$$

   és törli az összes változót, amit használt, kivéve az aktuális $x_i^{(t)}$ titkos kulcsot.

Ez a protokoll megoldja a részek frissítésének problémáját az (passzív) ellenféllel szemben, akik hozzáférnek a titkos információhoz, elérhetővé téve azt a megtámadott szerverek számára de itt minden kiszolgálónak követnie kell az előre

meghatározott protokollt. Ezt a következő tétel fogja bizonyítani. Vegyük észre, azt feltételezzük, hogy a részek tökéletes titoktartással vannak továbbítva a megfelelő titoktartóknak. Ez teszi lehetővé számunkra, hogy igazoljuk ennek a sémának az információs elméletének titkosságát. A továbbiakban explicit titkosítást fogunk használni ezeknek a részeknek az átvitelére és ezután a rendszer titkosságának erőssége csökkenti a titkosításhoz szükséges munkát.

3.4.1. Tétel. *Ha minden kiszolgáló követi a fenti frissítési protokollt, akkor: A robusztusság: az új részek a frissítési fázis végén kiszámítottak, megfelelnek a titkos x-nek. (azaz, bármely részhalmaza a $k+1$ új résznek képes visszaállítani az x-et.)*
A titkosság: Egy ellenfél, aki bármely időszakban nem tud többet a titokról mint k részlet (esetleg mindig különböző részek) semmit nem tud a titokról.

3.4.4. A részek frissítési protokollja aktív támadók jelenlétében

A fenti alapvető frissítési protokollban ha egy aktív ellenfél irányítja a szervert az a titok megsemmisülését okozhatja, hiszen vagy ellentmondásos részfrissítésekkel foglalkozik, vagy csak úgy választja meg a δ_i polinomot, hogy $\delta_i(0) \neq 0$. Annak érdekében, hogy érzékelje a rosszul kiosztott részeket, adjunk hozzá a fenti alapvető protokollhoz egy ellenőrizhetőségi jellemzőt. Nevezetesen alkalmazzuk a korábban bemutatott Feldman-féle ellenőrzött titokmegosztási rendszert. Az ellenőrizhető titokmegosztás hagyományos alkalmazása során, a titok helyességének bizonyítéka, hogy a saját részét következetesnek találja a használat során. Esetünkben ez a bizonyítás most arra szolgál, hogy a kiszolgálók által a frissített részek helyesen lettek kiosztva. A kiszámítható frissítési protokoll az összes P_i szerverre egy t időpillanatban a következő:

1. P_i felvesz k véletlen számot $\delta_{\{im\}m\in\{1,\ldots,k\}}, Z_q$-ból, hogy meghatározzon egy
$$\delta_i(z) = \delta_{i1}z^1 + \delta_{i2}z^2 + \cdots + \delta_{ik}z^k$$
polinomot. Szintén kiszámolja az $\epsilon_{im} = g^{\delta_{im}} \pmod q$ értékeket, ahol $m \in \{1, \ldots, k\}$.

2. P_i kiszámítja $u_{ij} = \delta_i(j) \pmod q, j \in \{1, \ldots, n\}$, és kiszámítja
$$e_{ij} = ENC_j(u_{ij}), \forall j \neq i.$$

3. P_i közvetíti a $VSS_i^{(t)} = (i, t, \{\epsilon_{im}\}_{m\in\{1,\dots,k\}}, \{e_{ij}\}_{j\in\{1,\dots,k\}\setminus\{i\}})$ üzenetet, és az $SIG_i(VSS_i^{(t)})$ aláírást.

4. Minden közvetített üzenetért az előző lépésben a többi kiszolgáló által, P_i dekódolja a P_i -nek szánt részt (azaz kiszámítja u_{ji} -t az e_{ji} -ből $\forall j \neq i$ esetén), és ellenőrzi a részek pontosságát az (3.1)-es egyenletet használva az ellenőrzésre minden $j \neq i$-re:

$$g^{u_{ii}} \stackrel{?}{=} (\epsilon_{j1})^i (\epsilon_{j2})^{i^2} \dots (\epsilon_{jk})^{i^k} \pmod p \tag{3.4}$$

   (Megjegyezzük, hogy ez az egyenlet beszámol a $\delta_i(0) = 0$ feltételről.)

5. Ha P_i minden a többi szerver által elküldött üzenetet korrektnek ítéli meg az előző lépésben (pl. mind helyesen volt aláírva, mindnek helyes idő intervallum száma volt, stb.) és mindre teljesül a fenti egyenlet, akkor közvetíti az aláírt elfogadási üzenetet, melyben bejelenti, hogy az összes ellenőrzés sikeres volt.

6. Ha minden kiszolgáló küld egy elfogadó üzenetet, akkor ezután P_i folytatja a saját részének frissítését az $x_i^{(t)} \leftarrow x_i^{(t-1)} + (u_{1i} + u_{2i} + \dots + u_{ni}) \pmod q$ végrehajtásával és törli az összes használt változót, kivéve az éppen használt $x_i^{(t)}$ részt.

7. Ha az 5. lépésben, P_i talál bármi szabálytalanságot a többi kiszolgáló viselkedésében a 4-es lépés közben, akkor közzétesz egy aláírt jelentést a rosszul viselkedő szerver(ek) ellen.

Hogy mikor küldenek ilyen jelentést és hogy hogyan oldják meg azokat, a következő alfejezetben részletesebben megtalálható.

3.4.5. A vádak megoldása

A fenti protokoll 5. lépésében, minden kiszolgáló ellenőrzi a helyes viselkedést és hogy a többiekkel hogyan foglalkoznak. Ha egy rosszul viselkedő kiszolgálót találnak, akkor kétféle intézkedés lehetséges. Az egyik hogy nem használja a $\delta_i(.)$ polinomkiosztást a szerver a részek frissítésére a 6. lépésben. A második, hogy riasztja a rendszerirányítást, így az intézkedhet arról, hogy kijavítsa a rosszul működő kiszolgálókat.(pl. Előfordulhat, hogy újra kell indítani a szervert, azért hogy kizárja az ellenfelet). Azonban az egyik kiszolgáló által a másik kiszolgálóval szembeni vádak megkövetelik az ellenőrzést, mivel egy rosszul működő szerver

hamisan vádolhatja a többit. A részek konzisztens frissítése érdekében a jól működő kiszolgálóknak szükséges beleegyezniük abba, kik a rossz kiszolgálók. A későbbiekben megmagyarázom majd, hogyan döntik el az egyes P_i kiszolgálók a B_i rossz processzorok listáját. Azt mondjuk, hogy egy P_i -től származó t-ik időszakból való üzenet helyes, ha az megfelel a fenti protokoll előírásainak, beleértve az összes meghatározott mezőket és információkat (pl. a helyes időszak számát) valamint a helyes aláírásokat. Megkülönböztetünk 3 szabálytalansági osztályt a protokollban:

- Formálisan nem megfelelő üzenet: rossz időszak számmal rendelkezik, olyan számokkal, amik a határokon kívül vannak, stb.
- Kettő vagy több helyes üzenet származik ugyanattól a kiszolgálótól, amelyek érvényes aláírással ellátottak, vagy egyáltalán nincs üzenet néhány szervertől, és
- eltér a (3.4)-es egyenlettől

Jegyezzük meg, hogy a szabálytalanságok első két változata csak a nyilvános információk használatakor fedezhető fel, és ezután megtalálja az összes (megfelelően működő) kiszolgáló. Minden ily módon rosszul viselkedő processzort a többi rossznak jelöl. A hibák harmadik fajtája sokkal problémásabb, mivel az csak helyileg felfedezhető fel egy olyan szerver által, amelyik megkapja azt a részt, ami az eltérést okozza a (3.4.) egyenlettől. Amikor egy P_i kiszolgáló a (3.4)-es egyenletet megfelelőnek találja, a P_j a küldött információt nem tartja meg, akkor szükség van P_j ellen vádat jelenteni. Ezután a kiszolgálók eldöntik majd, hogy vajon P_i vagy P_j csalt-e. Ennek az a módja, hogy P_j megvédi saját magát. Ha P_j helyes u_{ij}-t küldött, nevezetesen olyat, ami megfelelt a (3.4)-nek, akkor ezután feltárhatja ezt az értéket és bizonyíthatja hogy ez megfelel a nyilvánosan elérhető e_{ji} titkosítási értéknek, amit P_j küldött P_i-nek. Hogy ezt P_j bebizonyítsa szüksége lehet további információk felfedésére, amiket a titkosítás kiszámításához használtak. P_j-nek azonban nem kell titkos információt feltárnia. Ezután mindenki le tudja ellenőrizni, hogy vajon u_{ji} és a P_j által közzétett kiegészítő információk titkosítva lettek-e P_i nyilvános kulcsával e_{ji}-be ahogy a 2. lépésen P_j közvetítette. Másrészt, mindenki leellenőrizheti azt is, vajon ez az u_{ji} összeillik-e a (3.4.) egyenlettel. Ha P_j megfelelően védi magát, az összes kiszolgáló rossznak jelöli P_i-t, egyébként P_j jelöljük rossznak. Jegyezzük meg, hogy néhány titkosítási módszerben (mint pl az RSA-ban), azok az információk, amely a vádló által

lettek közzétéve, elegendőek a nyilvános ellenőrzéshez, és ez leegyszerűsíti a fenti általános protokollt. Ha egyszer minden vádat leellenőriztünk, minden helyesen működő P_i kiszolgáló ugyanazt a listát tekinti a B_i rossz kiszolgálókról. Ezután az új részek kiszámítása elvégezhető az által, hogy a részek frissítési protokolljának 6-os lépését kicseréljük a következővel: $x_i^{(t)} \leftarrow x_i^{(t-1)} + \sum_{j \notin B_i} u_{ji} \pmod q$.

3.4.6. A fentebb említett RSA bemutatása

[10] szerint: Az RSA-eljárás nyílt kulcsú (vagyis asszimetrikus) titkosító algoritmus, melyet 1976-ban Ron Rivest, Adi Shamir és Len Adleman fejlesztettek ki (és az elnevezést nevük kezdőbetűiből kapta). Ez napjaink egyik leggyakrabban használt titkosítási eljárása. A nyilvános kulcsú kriptográfia alapja, hogy a kódoló és a dekódoló transzformáció, illetve az azokat paraméterező nyilvános és titkos kulcsok nem azonosak, sőt egyiket a másikból kiszámítani nagy komplexitású feladat, azaz praktikusan lehetetlen. Ezért a két kulcs közül az egyiket, általában a kódoló kulcsot, nyilvánosságra lehet hozni. Ez azért hasznos, mert ezáltal leegyszerűsödni látszik a kulcscsere probléma, ugyanis a titkos kommunikáció megvalósításához nincsen szükség egy titkos (szimmetrikus) kapcsolatkulcs létrehozásra a küldő és a fogadó között; elegendő a rejteni kívánt üzenetet a fogadó nyilvános kódoló kulcsával rejtjelezni. Megjegyezzük azonban, hogy gondoskodni kell a nyilvános kulcsok hitelességéről. Az RSA algoritmus három algoritmusból, a kulcsválasztás, a kódolás és a dekódolás algoritmusából áll. A kulcsválasztást a rendszer minden felhasználója elvégzi:

1. Véletlenszerűen választunk két nagy, p és q prímszámot ($p \neq q$). Jelenleg "nagynak" a legalább 1000 bit bináris méretű számokat tekintik.

2. Kiszámítjuk az $N = p \cdot q$ modulust és a $\psi(N) = (p-1) \cdot (q-1)$ szorzatot. Választunk továbbá egy e számot, amelyik mind $(p-1)$-hez, mind $(q-1)$-hez, tehát $\psi(N)$-hez relatív prím.

3. Kiszámítjuk az e inverzét modulo $\psi(N)$, azaz keresünk egy d számot, amelyre $1 < d < \psi(N)$ és $e \cdot d = 1 \bmod \psi(N)$

A kulcsválasztás a 3. lépéssel be is fejeződik. A rendszerszerű üzemeltetés során azonban egy központi, minden felhasználó által hozzáférhető nyilvántartásra van szükség. Ennek érdekében a kulcs nyilvános részét, esetünkben az (N, e) számpárt nyilvánosságra hozzuk, a kulcs titkos részét, a (d, p, q) számhármast s

ezzel együtt a $\psi(N)$ számot titokban tartjuk. Ha A felhasználó B felhasználónak akar üzenni, akkor A felhasználó kikeresi B nyilvános kulcsát, az $e = e_B$ és $N = N_B$ számokat. A felhasználó előkódolja az üzenetet. A nyílt szöveg valamilyen karakterkészlet elemeiből áll. A rejtjelezéshez ezt a karaktersorozatot olyan nemnegatív egészek sorozatává kell átalakítani, amelyek mindegyike kisebb, mint N_B. Ez egy kölcsönösen egyértelmű transzformáció, amelyet minden felhasználó ismer. A rejtjelezést az előkódolt üzenet egymás utáni számain A felhasználó egyenként hajtja végre. Ha az előkódolt szöveg következő száma x, akkor a megfelelő rejtjeles szám:

$$y = E_B(x) = X^{e_B} \bmod (N_B).$$

A B felhasználó megkap egy rejtjeles üzenetet. Ez az üzenet szükségszerűen a 0 és $N_B - 1$ közti egész számok egy $y_1, y_2, \cdots$ sorozata. A dekódolást a számsorozat elemein külön-külön hajtja végre. Legyen a rejtjeles szöveg a következő szám: y. Ekkor az "előkódolt" üzenet megfelelő száma

$$x = D_B(y) = y^{d_B} \pmod{N}_B.$$

A visszaállított, "előkódolt" $x_1, x_2, \cdots$ sorozatból az előkódolás inverzét alkalmazva adódik a valódi nyílt szöveg.

3.4.2. Tétel. *Bármely x egész számra fennáll, hogy*

$$(x^e)^d = x \pmod{N}$$

Bizonyítás. Mivel d az e mod $\psi(N)$ inverze, ezért valamely v egész számra $e \cdot d = v \cdot \psi(N) + 1$, s így a tétel bizonyításához az

$$x = x^{v\psi(N)+1} \pmod{N} \tag{3.5}$$

modulo egyenlőséget kell belátni. Először tetszőleges u prímre igazoljuk, hogy tetszőleges x és s esetén

$$x = x^{s(u-1)+1} \pmod{u} \tag{3.6}$$

fennáll: ha u prím nem osztója x-nek, akkor a Fermat-tétel szerint

$$(x^{u-1})^s = 1^s = 1 \pmod{u},$$

tehát (3.6) fennáll, ha pedig u osztója x-nek, akkor nyilván

$$x = 0 = x^{s(u-1)+1} \pmod{u}.$$

Visszatérve az általános esetre, mivel $p-1|\psi(N)$ és $v-1|\psi(N)$, ezért (3.6) felhasználásával

$$x = x^{v\psi(N)+1} \pmod p$$
$$x = x^{v\psi(N)+1} \pmod q$$

fennállnak. Innen

$$p|x^{v\psi(N)+1} - x$$

és

$$q|x^{v\psi(N)+1} - x$$

teljesülnek, amelyből $N|x^{v\psi(N)+1} - x$ is következik. □

A kulcsválasztásnak elfogadható időn belül kivitelezhetőnek kell lennie. Ezért meg kell vizsgálni, hogy milyen nagy sebességgel tudunk "nagy" prímszámokat generálni. Továbbá könnyű eldönteni, hogy $\psi(N)$ és az e kitevőjelölt relatív prímek-e, amely vizsgálatához az euklideszi algoritmus használható. Az e szám inverzét is gyorsan meg lehet határozni.

3.4.7. Biztonsági tulajdonságok

Foglalkoztunk a részek frissítésével, feltételezve hogy a szerverek kíváncsiak, de jól működnek. Most ismertetek egy megoldást arra az esetre, amikor akár k szerver is önkényesen rosszul viselkedik a frissítési protokoll során.

3.4.3. Tétel. *Ha az ellenfél a protokoll alatt akár k kiszolgálót is ellenőriz, akkor:*
Robusztusság: Az új részek a frissítési szakasz végén a helyesen működő kiszolgálók által lesznek kiszámolva, úgy hogy megfeleljenek a titkos x-nek. (pl., bármely $k+1$ részhalmaza a jól működő kiszolgálók által interpolált x titok új részeinek).
Titkosság: A fenti titokmegosztási séma szemantikailag biztonságos.

3.5. Rész helyreállítási rendszer

Egy proaktív titokmegosztási rendszernek szükséges hogy le tudja ellenőrizni, vajon egy része a résztvevő kiszolgálóknak sérült-e (elveszett), és visszaállítja a helyes részeket, ha szükséges. Másrészt, az ellenfél okozhatja a titok elvesztését az által, hogy fokozatosan megsemmisíti az $n-k$ részeket. A későbbiekben bemutatom a szükséges mechanizmusokat és a sérült részek helyreállítását. A P_i birtokában lévő $x_i^{(t)}$ részt, amely t időintervallumbeli, helyesnek nevezzük, ha $x_i(t) = f^{(t)}(i) \pmod q$, ahol $f^{(t)}$ az aktuális titokmegosztó polinom. Egyébként azt mondjuk, hogy a rész hibás. A kiszolgáló képes hibás megosztást nyújtani, azért, mert az ellenfél által ellenőrizve voltak a részek a frissítési protokoll alatt (és ezáltal megakadályozta a részek helyes frissítését), vagy azért, mert az ellenség megtámadta a szervert a frissítési fázis után és módosította a szerver titokmegosztását. Egy titokmegosztás el tud tűnni oly módon, hogy a kiszolgáló újra lett indítva vagy helyettesítve lett egy új szerver által.

3.5.1. Eltulajdonított részek érzékelése

Hogyan lehet az eltulajdonított részeket észlelni? Néhány esetben egyszerű észrevenni, hogy egy kiszolgáló igényli a helyes részek helyreállítását. Ez a helyzet azokkal a szerverekkel, amelyek nem vesznek részt a frissítési fázisban (pl. egy összeomlás miatt), vagy amelyek rosszul viselkednek ezalatt az időszak alatt. Azonban ha néhány szerver titokrésze módosul az ellenfél által (pl. egy frissítési fázis után), ezután ez a módosítás lehet hogy nem lesz észlelhető. Ennélfogva, a proaktivitás szellemében, a rendszernek szükséges periodikusan ellenőrizni a résztvevő szerverek helyi állomásait és azok helyességét, észlelve a részek eltűnését vagy módosulását. Hogy megvalósítsuk az elosztott részek ellenőrizhetőségét, adjunk hozzá egy invariánst minden t időszakban. Minden P_i kiszolgáló tárol egy $y_{j^{(t)}}, j \in \{1, \ldots, n\}$ készletet az $y_{j^{(t)}} = g^{x_{j^{(t)}}} \pmod p$ exponensek közül az aktuális részek az összes A-beli szerverben.

3.5.2. Alapvető részhelyreállítás protokollja

Egyszerűen rekonstruáljuk a részeket, $x_r = f^{(t)}(r)$, $r \in B$, oly módon, hogy engedjük minden kiszolgálónak $D = A \setminus B$-ben, hogy a saját részét küldje el P_r-nek.

Azonban ez felfedné az x titkot a P_r számára. Ahelyett, hogy minden $r \in B$-re, a D-beli kiszolgálók együttesen generáljanak egy véletlen titokmegosztását x-nek, hasonló módon járjunk el mint amit a részek frissítési protokolljában használtunk arra, hogy újra véletlenszerűsítjük az x titokmegosztását: Minden D-beli P_i kiszolgáló egy véletlenszerű k-ad fokú $\delta_{(.)}$ polinommal foglalkozik, oly módon, hogy $\delta_{(.)}n = 0 \pmod q$. Ezen a módon, egy új, véletlen titokmegosztását kapjuk x_r-nek: x_i, $i \in D'$. A kiszolgálók most már el tudják küldeni ezeket az új részeket P_r-nek, így lehetővé válik x_r számítása anélkül, hogy megengedjük P_r-nek, hogy bármit is megismerjen az eredeti x_i, $i \in D$ részről. Ugyanígy minden k vagy kevesebb szerver koalíciója, kivéve P_r-t, semmit sem fog tudni x_r értékéről.
Először megmutatjuk a részek helyreállító protokolljának letagadhatatlan ellenőrizhetőségét. Ez csak olyan ellenfél ellen biztonságos, aki lehallgat k vagy kevesebb kiszolgálót, de nem változtatja meg azok viselkedését. Minden P_r számára, amely megköveteli a részek helyreállítását, a következő protokoll teljesített:

1. Minden $P_i, i \in D$, felvesz egy véletlen $\delta_{(.)}k$-adfokú polinomot Z_q felett,mint $\delta_i(r)0$, pl. véletlen együtthatókat vesz fel$\{\delta_{i,j}\}_{j\in\{1,\ldots,k\}} \subset Z_q$, és ezután kiszámítja $\delta_{i0} = -\sum_{j\in\{1,\ldots,k\}} \delta_{ij} r^j \pmod q$.

2. Minden $P_i, i \in D$, közvetíti $\{ENC_j(\delta_i(j))\}_{j\in D}$.

3. Minden $P_i, i \in D$, létrehozza az új x_r részt, $x_i' = x_i + \sum_{j\in D} \delta_j(i)$ és elküldi ezt P_r-nek közvetítve $ENC_r(x_i')$-t.

4. P_r dekódolja ezeket a részeket és interpolálja őket x_r helyreállításához.

3.5.3. A teljes részhelyreállítás protokollja

3.5.1. Tétel. *Ha az ellenfél nem befolyásol több mint k kiszolgálót akármelyik időszakban, akkor a teljes frissítési protokoll a következő két tulajdonsággal rendelkezik:*
Robusztusság: Minden helyreállító szerver, amelyik követi a protokollt helyreállítja a helyes állapotot: $x_r = f^{(t)} \pmod q$.
Titkosság: A szemantikus titkossága az x titoknak megmarad.

3.6. Proaktív titokmegosztás: ötvözött protokoll

Ötvözve az összes fenti részt, azt kapjuk, hogy a teljes protokollunk a proaktív titokmegosztásra: Minden időintervallum elején, a frissítési fázis aktiválódik, ami a következő 3 szakaszból áll:

- a titkos kulcs frissítési protokollja,
- a részek helyreállítási protokollja (az eltűnt darabok észlelését is tartalmazza),
- és a részek frissítési protokollja.

A következő tétel összegzi a fő eredményünket:

3.6.1. Tétel. *Ha nincs k-nál több hibás kiszolgáló az egyes t időszakokban (ahol a kiszolgálók megtámadottak a frissítési fázisban, annak tekinthetők mind a két szomszédos periódusban is), akkor a fenti protokoll alapít egy biztonságos és robusztus proaktív titokmegosztási sémát.*

3.7. Köszönetnyilvánítás

A dolgozatom végén szeretném megköszönni témavezetőmnek, Villányi Viktóriának a türelmét és segítségét, illetve szeretnék köszönetet mondani a témáért is. Köszönettel tartozom még családomnak és barátaimnak a tanulmányaim alatti folyamatos támogatásért, biztatásért.

Irodalomjegyzék

[1] A. Shamir. How to share a secret. Communications of the ACM, 1979

[2] A. C. Yao. Protocols for secure computations. 23rd IEEE Symposium on the Foundations of Computer Science, 1982

[3] G.R. Blakley: Safeguarding cryptographic keys, Proceedings of the National Computer Conference 48: 313-317 (1979

[4] L. Csirmaz: Titokmegosztás gráfokon, Nyíregyházi kriptográfiai és diofantikus nap,http://www.renyi.hu/ csirmaz/ (2005).

[5] Csirmaz, László (2010) Online secret sharing. In: Combinatorics in Secret Sharing, March 26, 2010, Amsterdam, RISC@CWI.

[6] Csirmaz, László (2012) Gruppen secret sharing, or how to share several secrets if you must? In: Tatracrypt 2012, July 2-4, 2012, Smolenice, Slovakia.

[7] Hammoudi, K.; Melkemi, M. Personalized Shares in Visual Cryptography. J. Imaging 2018, 4, 126

[8] Amir Herzberg,Stanisław Jarecki,Hugo Krawczyk,Moti Yung. Proactive Secret Sharing Or: How to Cope With Perpetual Leakage Conference paper 2001

[9] Liu, C.L. Introduction to Combinatorial Mathematics. McGraw- Hill, New York, 1968.

[10] Buttyán Levente, Vajda István Kriptográfia és alkalmazásai, Typotex, Budapest,2005

[11] Threshold Cryptography Based on Blakley Secret Sharing Ilker Nadi Bozkurt, Kamer Kaya, Ali Ayd?n Selcuk,Ahmet M. Güloglu