Eötvös Loránd Tudományegyetem

Informatikai Kar

Komputeralgebra Tanszék

Hitelesíthető késleltetett függvények és alkalmazásaik

Témavezető:

Burcsi Péter

Docens

Szerző:

Hosszejni Darjus

Programtervező informatikus MSc

Budapest, 2019.

Szeretnék köszönetet mondani Burcsi Péternek a támogatásáért és a segítségéért, továbbá Manz Ingridnek, aki nélkül a diplomamunka szintén nem jött volna létre.

Tartalomjegyzék

1. fejezet

Bevezetés

Mi sem bizonyítja jobban, hogy a véletlen számok az emberiség fontos eszközei közé tartoznak, mint hogy dobókockákat már 4-5000 évvel ezelőtt, az írott számok kifejlődése előtt használtak (L'Ecuyer, 2017). A véletlen számok nemcsak a szerencsejátékokhoz szükségesek, de modellezhető segítségükkel a természetbeli radioaktív bomlás is, továbbá kellenek a numerikus módszerek körében a Monte Carlo szimulációkhoz, és a kriptográfiában is hasznosak mint előre meg nem jósolható kimenetelű számítások eredménye.

A mai szükségletekhez a dobókockához hasonló tárgyak már túl lassúak, és a „valódi véletlen számok" nem is minden alkalmazáshoz létfontosságúak. Ennek apropóján fejlődött ki a (statisztikailag) véletlennek tűnő ún. pszeudovéletlen számok fogalma (lásd pl. Knuth, 1987, 3. fejezet). Ez alatt olyan véletlen szám- vagy bitsorozatot értünk, amely néhány fontosnak vélt elméleti tulajdonsággal bír (Kendall és Babington-Smith, 1938). Most röviden áttekintünk egy alkalmazást, ami érzékelteti a dolgozatban vizsgált problémákat.

1.1. Determinisztikus lottó

A mai számítógépes rendszerek eredően determinisztikus módon működnek, alkalmatlanok valódi véletlenszerűség előállítására. Mégis felmerül az igény adott eloszlású, megjósolhatatlan és megbízható véletlen számok generálására, gyakori példa egy lottó implementálása.

Egy lottóban adott egy kettős kimenetelű jövőbeli esemény, aminek az eredményére fogadva két fél játszik. Mindkét fél feltesz valamennyi pénzt, majd az esemény kimenetelétől függően az egyik fél nyereménye az összes feltett pénz, míg a másik fél nem kap semmit. Természetes módon merülnek fel többek közt a következő kérdések. Mi legyen az esemény? Milyen valószínűségek társulnak az esemény lehetséges eredményeihez? Tudja-e bármelyik fél befolyásolni az eseményt?

Az első probléma, hogy nincs véletlen, azaz senki által sem megjósolható kimenetelű esemény egy determinisztikus rendszerben. Emiatt szükséges egy rendszeren kívüli forrás, egy harmadik fél, amelyben mindkét játékos megbízik (Rabin, 1983). Ha létezik ilyen, akkor a probléma meg is van oldva: a megbízható harmadik fél mondja meg, hogy ki győzött.

Egy ilyen protokollról ír Clark és Hengartner (2010), amiben a harmadik fél a tőzsdepiac. Ebben a protokollban egy pénzügyi statisztikai modell segítségével vonnak ki véletlen biteket részvények árfolyammozgásából. A tőzsdepiac mozgását a passzív megfigyelő szemszögéből nehezen megjósolhatónak tartják, ennek köszönhetően lesz a piac alkalmas harmadik fél. A probléma azonban az, hogy nem tudhatjuk, hogy a lottóbeli ellenfelünknek van-e elég hatalma a piacot aktívan befolyásolni.

Képzeljünk el egyszerű esetet, amikor a lottó nyertese a napi záróárfolyamnak egy nehezen invertálható függvénye, és passzív megfigyelőként 50-50% eséllyel nyer bármelyik fél. Elég az ellenfelünknek közepes erőforrásokkal rendelkeznie, az is elég lehet ahhoz, hogy néhány perccel a lottó határidejének lejárta előtt kiszámolja az összes még valószínű forgatókönyvet. Ez alapján néhány másodperccel a határidő lejárta előtt úgy tud beavatkozni a piacba – részvényt adni vagy venni –, hogy minimálisan a maga érdekének irányába változtassa a részvényárat. Ha ellenfelünk kellően sok lottószelvénnyel kellően nagy értékben játszik, egy ilyen „csalás” megérheti neki.

Az ilyen jellegű támadásokat megelőzendő hasznos módosítás, ha a lottó nyertese a napi záróárfolyamnak nemcsak nehezen invertálható, de nehezen kiértékelhető függvénye is. Ekkor ugyanis nem éri meg az összes valószínű forgatókönyvet kiszámolni, ez a lottó érvényessége alatt végig túl sok erőforrást venne igénybe. Ez még mindig nem a végső megoldás: ha mindenkinek el kell végeznie a költséges függvény kiértékelését, akkor az esetleg túl sokat csökkent a lottó nyereségességén, és ez a lottót a gyakorlatban használhatatlanná teszi.

Mindezen problémák megoldására alkotta meg Boneh et al. (2018a) hosszú előzmények után a hitelesíthető késleletett függvények (HKF) fogalmát. Informálisan egy HKF egy olyan függvény, amit költséges kiértékelni, önmagában nem lehet invertálni és aminek az eredménye nem megjósolható, viszont az eredmény és egy bizonyítás birtokában a függvény invertálható, így mindenki hatékonyan meggyőzhető, hogy az eredmény helyes.

1.2. Diplomamunka célja

Jelen diplomamunka célja áttekinteni a szakirodalmat és empirikus összehasonlítást végezni a létező, gyakorlatban is biztató hitelesíthető késleltetett függvények között.

1.3. Előzmények a szakirodalomban

Több, mint fél évszázada foglalkoztatják a matematikusokat az ún. pszeudovéletlenszám-generátorok, vagyis az olyan determinisztikus függvények, amik adott bemenethez bizonyos szempontból véletlennek tűnő számsorozatot rendelnek. Elsőként említhető Kolmogorov (1965), aki véges bitsorozatra bevezeti a véletlenség egy fogalmát, mely az algoritmikus tömöríthetetlenséghez fűződik. Másik irányt képvisel Collet és Eckmann (1980) és Brudno (1982), ahol a nemlineáris, kaotikus dinamika jelenti a megjósolhatatlanságot. Lagarias (1993) ad áttekintést, és ki is emeli a legfontosabb problémát, a Megjósolhatatlanság Paradoxonját. Eszerint ha egy determinisztikus függvény megjósolhatatlan, akkor nehéz bármit is bebizonyítani róla – többek közt azt is, hogy megjósolhatatlan.

A nehezen invertálható függvények egy a kriptográfiában elengedhetetlen osztályát alkotják a hasítófüggvények, mely kifejezés nyomtatásban először Hellerman (1967) könyvében jelent meg. A hasítófüggvények a tetszőleges hosszúságú bemeneti bitsorozatot fix hosszúságú bitsorozattá alakítják, és az adatbázis-kezelés mellett alkalmazhatók titkosításhoz, hitelesítéshez és veszteséges tömörítéshez is. Megfelelő minőségű hasítófüggvények iterációjával könnyen készíthető nehezen invertálható, költségesen kiértékelhető függvény. Sőt s darab processzor használatával s darab

számítási részeredmény elmentése után a számítási időnél akár s-szer gyorsabban hitelesíthető a számítás eredménye. De a kiértékelés és a hitelesítés között ez a fajta konstansszoros gyorsítás a gyakorlatban kevés.

Egy biztatóbb konstrukció fűződik Dwork és Naor (1993) és Jerschow és Mauve (2011) nevéhez, akik a moduláris gyökvonás nehézségére támaszkodnak. A számítás eredménye a gyökvonás bonyolultságához képest logaritmikus időben hitelesíthető, ami már megfelelő lehet. Problémás viszont, hogy nagyon nagy prímre van szükség ahhoz, hogy a függvény kiértékelése tényleg költséges legyen. Ezen javít Lenstra és Wesolowski (2015) úgy, hogy egy iteratív protokollt javasol, viszont ezzel egyidőben elveszíti a hitelesítés logaritmikus sebességét.

1.4. Diplomamunka felépítése

A diplomamunkát három fő részre bontjuk. A 2. fejezetben megalapozzuk a HKF fogalmát, majd áttekintést adunk a jelenlegi két legismertebb előterjesztésről, melyek a gyakorlatban is HKF-ként szolgálhatnak. A 3. fejezetben bemutatjuk, hogy az irodalomban leírt specifikációkat hogyan valósítottuk meg. A 4. fejezetben ismertetjük a kutatásunk kimenetét, az általunk fejlesztett szoftver segítségével végzett empirikus összehasonlítás eredményét. A három fő részt a zárszó követi az 5. fejezetben.

2. fejezet

Elméleti háttér

A modern kriptológia matematikai alapokon nyugszik, a számítástudományban és a kombinatorikában gyökerezik, azok fogalomrendszerét erősen használja. Ebben a fejezetben is szükség van erre a fogalomrendszerre, ezért először ismertetjük a hitelesíthető késleltetett függvények tárgyalásához szükséges matematikai alapdefiníciókat, majd azokra támaszkodva a legfontosabb eredményeket.

Először a „nagy ordo" és a „teta" jelölést definiáljuk.

1. Definíció. *Legyen $f(n)$ és $g(n)$ két, a természetes számok halmazán értelmezett, pozitív értékű függvény. Az $f = O(g)$ jelölés azt jelenti, hogy léteznek olyan c és N konstansok, hogy minden $n > N$-re $f(n) \leq cg(n)$. Ha $f = O(g)$ és $g = O(f)$, akkor $f = \Theta(g)$.*

Ezután rátérhetünk az itt használt algoritmikus bonyolultságelméleti fogalmakra. Egy párhuzamos algoritmus *futási ideje* alatt ebben a fejezetben a párhuzamos lépésszámot értjük, vagyis a legtöbb lépést igénylő programszál szekvenciális lépésszámát. Ez definiálja a degenerált esetet is, a szekvenciális algoritmusét, amikor csak egy programszál van. A futási idő fogalmán belül kiemelünk két osztályt.

2. Definíció. *Egy szekvenciális algoritmust rendre* polinom*, illetve* polilog idejű algoritmus*nak hívunk, ha létezik egy c pozitív egész szám, hogy a legfeljebb n bites egészekkel végzett bitműveletek száma $\Theta(n^c)$, illetve $\Theta((\log(n))^c)$. Ezeket rendre* poli(n)*-nel, illetve* polilog(n)*-nel jelöljük.*

Következőnek formalizálunk egy konvenciót, hogy mit tartunk elhanyagolhatónak.

3. Definíció. *Egy $f : \mathbb{N} \to \mathbb{R}^+$ függvény λ elhanyagolható függvénye, ha minden $d > 0$-ra $\lim_{\lambda\to\infty} f(\lambda)\lambda^d = 0$, vagyis f reciproka minden polinomnál gyorsabban nő.*

A fejezet hátralévő részében az elhanyagolható valószínűség azt jelenti, hogy a valószínűség az egyik – általában a biztonsági – bemeneti érték elhanyagolható függvénye.

A kriptográfiai bizonyítások során gyakran támaszkodunk a redukció módszerére, amikor egy matematikai feltétel fennállására vagy egy absztrakt objektum létezésére vezetjük vissza az adott tétel állítását. Egy ilyen absztrakt objektum a véletlen orákulum is, amivel a bevezetőt zárjuk. Az orákulum definíciója tartalmában megegyezik Buttyán és Vajda (2012) 13.6-os definíciójával, majd Katz és Lindell (2014) 13.1.1-es alfejezete alapján bevezetjük a *véletlen orákulum* fogalmát.

4. Definíció. *Amikor azt mondjuk, hogy egy $\mathcal{A}$ algoritmus a J* orákulum*ot hívja (orákulum-lekérdezés), akkor ezen azt értjük, hogy $\mathcal{A}$ előállítja J egy bemenetét, amelyre J „azonnal" előállítja a kimenetét, amelyet $\mathcal{A}$ aztán felhasznál. Az „azonnal" azt jelenti, hogy $\mathcal{A}$ futási idejébe J futási ideje nem számít bele. Az orákulum egy függvény, vagyis adott bemenetre mindig ugyanazt a kimenetet adja.*

Véletlen orákulum*ról akkor beszélünk, ha adott $n, m \in \mathbb{N}$-re a $\{0,1\}^n$-értékű bemenetet fogadó orákulum egy korábban nem használt bemenetre $\mathcal{A}$ szempontjából úgy viselkedik, mint egy $\{0,1\}^m$-en értelmezett, egyenletes eloszlású valószínűségi változó. Ekvivalens átfogalmazásban: ha jön egy eddig nem látott b bemenet, akkor az orákulum „sorsol" egy k kimenetet, és megjegyzi k-t annak érdekében, hogy b-re később is k-val tudjon válaszolni.*

2.1. Hitelesíthető késleltetett függvények

Intuitíve, egy f függvényt akkor nevezünk hitelesíthető késleltetett függvénynek (HKF), ha f-et csak lassan lehet kiértékelni, ellenben az $f(x) = y$ eredmény egy bizonyíték birtokában gyorsan hitelesíthető. Végleges, széles körben elterjedt, formális, matematikai definícióról a szakirodalomban még nem beszélhetünk. Ezek a HKF-től megkövetelt tulajdonságokban térnek el, példákért lsd. a Boneh et al. (2018a), Wesolowski (2018), Pietrzak (2019) és Boneh et al. (2018b) cikkeket. A dolgozatban

használt formális definíciót – mely Boneh et al. (2018a) legmegengedőbb verziója – alább közöljük.

5. Definíció. *Egy hitelesíthető késleltetett függvény három algoritmusból áll, amelyek egy* $f : \mathcal{X} \ni x \mapsto y \in \mathcal{Y}$ *függvény kiértékelését és hitelesítését végzik.* $\mathsf{Előkészít}(\lambda, T) \to P$ *egy véletlen algoritmus, mely előkészíti a HKF-et két bemeneti érték alapján:* λ *a HKF biztonságát,* T *pedig a késleltetését befolyásolja. Az algoritmus* P*-vel, a HKF nyilvános paramétereivel, tér vissza. Ezután* $\mathsf{Kiértékel}(P, x) \to (y, \pi)$ *kiértékeli és* y*-ba menti* $f(x)$*-et, illetve opcionálisan egy bizonyítást is megad* π*-ben. Végül* $\mathsf{Hitelesít}(P, x, y, \pi)$ *hitelesíti az eredményt, és „elfogad" értékkel tér vissza, ha* $f(x) = y$*, ellenkező esetben „elutasít" a visszatérési értéke.*

Továbbá, a HKF a következő feltételeknek is megfelel:

- $(1+\varepsilon)T$ *futási idő: létezik olyan* $\varepsilon > 0$*, amire* $\mathsf{Kiértékel}(P, x)$ *futási ideje minden lehetséges* P *és* x *értékre legfeljebb* $(1 + \varepsilon)T$*.*

- *Szekvencialitás: egy legfeljebb* $\mathrm{poli}(\lambda)$ *számú processzort párhuzamosan használó* $\mathcal{A}_1$ *algoritmus, amelynek kiértékelése* T *lépésnél kevesebb időt vesz igénybe, véletlen* $x \in \mathcal{X}$ *és* $P \leftarrow \mathsf{Előkészít}(\lambda, T)$ *értékekre elhanyagolható valószínűséggel találja meg a helyes* $y = f(x)$ *eredményt.*

- *Egyértelműség: ha egy* $\mathcal{A}_2$ *hatékony algoritmus a* P *nyilvános paraméterhez a* $(x, \tilde{y}, \pi)$ *értéket rendeli, amire* $\tilde{y}$ *különbözik a* $\mathsf{Kiértékel}(P, x)$ *eredményében lévő* y*-tól, akkor* $\mathsf{Hitelesít}(P, x, \tilde{y}, \pi)$ *elhanyagolható valószínűséggel jár sikerrel.*

- *Hatékony hitelesítés:* $\mathsf{Hitelesít}(P, x, y, \pi)$ *futási ideje* $O(\mathrm{polilog}(T)\,\mathrm{poli}(\lambda))$*.*

A kriptográfiai tulajdonságok bizonyításához szükséges, hogy ne közvetlenül $\mathbb{G}$ legyen megadva, hanem az Előkészít metódus, ami egy véletlen $\mathbb{G}$-t állít elő. Mint később látni fogjuk, a biztonság feltételei mind Előkészít függvényében vannak megfogalmazva.

A késleltetés lényeges eleme a HKF-nek, és pontosan szeretnénk tudni befolyásolni azt. Ezt formalizálja a definíció első két feltétele. A T paraméter az, amelyik a késleltetést beállítja. Az első két feltétel alapján a feladat a gyakorlatban nem megoldható T lépésnél kevesebből, viszont a HKF-et becsületesen ki lehet értékelni

T-nél nem sokkal több, lineáris számú lépésben. Az ε a bizonyítás kiszámításához szükséges lépéseknek van fenntartva.

Boneh et al. (2018a) megemlít még két kívánatos tulajdonságot, amik egy HKF-et még előnyösebbé tehetnek: a *dekódolhatós*ágot és a *növelhető*séget. Dekódolható egy HKF, ha – a jelölésben π elhagyásával – minden lehetséges P-re $\mathsf{Kiértékel}(P, .) : x \mapsto y$ injektív, más néven $\mathsf{Kiértékel}$ veszteségmentes tömörítés. Ekkor létezik egy $\mathsf{Dekódol}(P, .) : y \mapsto x$ függvény, amire $\mathsf{Dekódol}(P, \mathsf{Kiértékel}(P, x)) = x$ minden $x \in \mathbb{G}$-re. Ha a HKF *hatékonyan dekódolható*, tehát $\mathsf{Dekódol}$ $\mathsf{Hitelesít}$-hez hasonlóan hatékony, akkor a bizonyítást, π-t, valóban el lehet hagyni, ugyanis $\mathsf{Dekódol}$ közvetlenül y-ra alkalmazható.

Növelhető HKF-ről beszélünk, ha P több T-vel is működik, és π számítása közben határozhatjuk meg T értékét. Ennek az előnye az egymás után fűzött HKF-ekhez képest, hogy π mérete továbbra is pici, legfeljebb a hatékony hitelesítéshez meghatározott $O(\text{polilog}(T)\,\text{poli}(\lambda))$. Ha π nagyobb lenne, $\mathsf{Hitelesít}$-nek nem lenne ideje még beolvasni se π-t.

2.2. Egy általános konstrukció

Egy HKF alapja egy lassan kiértékelhető függvény, amit hatékonyan párhuzamosítani sem lehet. A T darab egymásutáni négyzetre emelést mint leképezést széles körben eredendően szekvenciálisnak tartják, amit már Rivest et al. (1996) is kihasznált. Erre alapozva az utóbbi időben két, a gyakorlatban is alkalmazható konstrukció jelent meg a szakirodalomban. Először Wesolowski (2018), majd nem sokkal később Pietrzak (2019) konstruált egy jelentősen eltérő HKF-et, amelyet Boneh et al. (2018b) általánosított. Boneh et al. (2018b) nyomán az általuk feljavított HKF-re, és nem az eredeti Pietrzak-féle HKF-re fogunk Pietrzak-protokollként hivatkozni. A Wesolowski-protokoll és a Pietrzak-protokoll egy közös keretrendszerben leírhatók (Boneh et al., 2018b):

- $\mathsf{Előkészít}(\lambda, T)$ kimenete $P = (\mathbb{G}, H, T)$, ahol $\mathbb{G}$ egy ismeretlen, de végesrendű Abel-csoport, és $H : \mathcal{X} \to \mathbb{G}$ egy hatékonyan kiértékelhető hasítófüggvény, amit a bizonyítások során véletlen orákulumként kezelünk.

- $\mathsf{Kiértékel}(P, x)$ először kiértékeli a $H(x)^{2^T} \in \mathbb{G}$ kifejezést, és az eredményét y-ba menti, majd y alapján kiszámolja a π bizonyítást. Hatványozás alatt a $\mathbb{G}$-n belül elvégzett ismételt szorzást kell érteni.

- $\mathsf{Hitelesít}(P, x, y, \pi)$ végül egy interaktív bizonyítási módszerrel ellenőrzi, hogy $(\mathbb{G}, H(x), y, T)$ része-e az $\mathcal{L}_{\mathrm{HKF}}$ nyelvnek, ahol

$$\mathcal{L}_{\mathrm{HKF}} = \left\{ (\mathbb{G}, g, h, T) : g \in \mathbb{G}, h \in \mathbb{G}, h = g^{2^T} \right\}. \tag{2.1}$$

A Wesolowski- és a Pietrzak-féle HKF a $\mathbb{G}$ csoportra vonatkozó feltételekben és a bizonyítás konstrukciójában és menetében térnek el, melyeket a következő alfejezetekben ismertetünk.

2.3. Wesolowski-féle protokoll

Wesolowski (2018) volt az első, aki egy kifejezetten az 5. definíciónak megfelelő módszerrel állt elő. Nagy előny, hogy a protokollban π összesen egy elem $\mathbb{G}$-ben, tehát a bizonyítás során kevés kommunikációra van szükség.

2.3.1. Bizonyítás menete

Az interaktív bizonyítás menetéhez szükséges az első 2^λ darab prímszám halmaza, jelölje ezt $\mathsf{Prímek}(\lambda)$. Jelölje továbbá A és B a két félt, ahol A szeretné B-nek bebizonyítani, hogy adott $(P, x) = ((\mathbb{G}, H, T), x)$ bemenetre az y eredmény birtokában van. Ekkor a bizonyítás menete:

1. B ellenőrzi, hogy y eleme-e $\mathbb{G}$-nek. Ha nem, akkor B megáll, és elutasítja y-t, különben továbbmegy.

2. B küld A-nak egy p prímet, melyet – A szemszögéből – egyenletes eloszlással választ $\mathsf{Prímek}(\lambda)$-ból.

3. A kiszámolja a $2^T/p$ egészosztás q hányadosát és r maradékát, és elküldi a $\pi = H(x)^q$ értéket B-nek.

4. B is kiszámolja r-t, és a bizonyítást sikeresnek tekinti, ha $y = \pi^p H(x)^r$ teljesül, különben elutasítja azt.

A protokollban a 2^T érték helyettesíthető egy tetszőleges e egésszel is. Fontos viszont, hogy B hatékonyan ki tudja számolni az $r \equiv e \pmod{p}$ maradékot.

A bizonyítás a Fiat-Shamir-protokoll (Buttyán és Vajda, 2012, 9.3-as fejezet) segítségével neminteraktívvá is változtatható. Ehhez szükség van egy nyilvános $J : (\mathbb{G}, x, y, T) \to \mathsf{Prímek}(\lambda)$ hasítófüggvényre, amit véletlen orákulumként kezelhetünk. A második lépés helyett J segítségével A kiszámolja p-t, majd a harmadik lépésnek megfelelően p-ből π-t, és az (y, π) pár közzététele után leáll. B ezután A-tól függetlenül tudja ellenőrizni A állítását. A-hoz hasonlóan B is kiszámolja p-t, majd elvégzi a negyedik lépést, ezzel elfogadva vagy elutasítva A eredményét.

2.3.2. Bizonyítás bonyolultsága

A két kis $\mathbb{G}$-beli hatvány kiszámításán kívül B-nek csak r-t kell kiszámolnia, ami összesen $O(\log_2(T))$ lépés $\mathbb{Z}/p$-ben (lásd pl. Buttyán és Vajda, 2012, 3.1-es példa algoritmusa). Ezzel szemben B legköltségesebb feladata a bizonyítás kiszámítása során a $\pi = H(x)^q$ hatvány kiértékelése, ami elvégezhető $2T$ csoportművelettel konstans tárhelyen (lásd pl. Boneh et al., 2018b, 3. oldal alja).

2.3.3. Biztonság

Egy a Wesolowski-protokollhoz szükséges és elégséges feltételt tárgyalunk az alfejezet hátralévő részében (Boneh et al., 2018b).

6. Definíció. *Legyen* Gen *olyan valószínűségi változó, amely egy $\lambda \in \mathbb{N}$ biztonsági paraméterhez egy $\mathbb{G}$ véletlen, végesrendű Abel-csoportot rendel. Azt mondjuk, hogy* Gen*re teljesül az adaptív gyök feltétel, ha tetszőleges $(\mathcal{A}_1, \mathcal{A}_2)$ hatékony algoritmuspárra a következő támadás λ függvényében elhanyagolható valószínűséggel jár sikerrel. Először* Gen *generál adott nyilvános λ értékre egy $\mathbb{G}$ csoportot. Ezt követően $\mathcal{A}_1$ kiválaszt egy tetszőleges $w \in \mathbb{G}$ elemet, és azt tetszőleges egyéb információval együtt megosztja $\mathcal{A}_2$-vel. Ezután $\mathcal{A}_2$ egy a támadók szemszögéből véletlen $p \in \mathsf{Prímek}(\lambda)$ prímre az $u \in \mathbb{G}$ értékkel tér vissza. A támadás sikeres, ha $u^p = w$, ellenkező esetben sikertelen.*

Egy sikeres támadás valószínűsége minden esetben minimum $1/|\mathsf{Prímek}(\lambda)|$, hiszen elég hozzá előre „eltalálni” a véletlen p értéket. Annak ismeretében ugyanis $\mathcal{A}_1$

tetszőleges u-ra már az elején a $w = u^p$ értéket tenné közzé és küldené el u-val együtt $\mathcal{A}_2$-nek, majd $\mathcal{A}_2$ egyszerűen az u-val térne vissza. Emiatt muszáj $\mathsf{Prímek}(\lambda)$ számosságát nagynak választani. Másrészt viszont azért van szükség arra, hogy p prím legyen, mert egy másik alaphalmaz, pl. $p \in \{1, \ldots, 2^\lambda\}$ esetén túl nagy valószínűséggel lenne p ún. sima szám, vagyis egy kis prímek szorzataként előálló egész szám. Ebben az esetben a következő támadás járhat könnyen sikerrel. $\mathcal{A}_1$ választ egy a sima számot, ami sok kicsi, mondjuk k-nál kisebb prímek szorzata, és a $w = u^a$ értéket u-val és a-val együtt elküldi $\mathcal{A}_2$-nek. $\mathcal{A}_2$ erre a véletlen p tudatában a könnyen kiszámolható $u^{a/p}$-nel válaszol, ha p osztja a-t, egyébként pedig valami mással, ami valószínűleg sikertelen lesz. Ez a támadás tehát működik, ha p osztja a-t, amire nagy az esély általános $p \in \mathbb{Z}$ esetén, és elhanyagolható az esély nagy p prím esetén.

A Wesolowki-protokoll zárásaként bemutatjuk a protokoll biztonságáról szóló tételt. A tétel és a bizonyítás megtalálhatók Wesolowski (2018) és Boneh et al. (2018b) cikkeiben is.

1. Tétel. *A Wesolowski-féle HKF biztonságához szükséges és elégséges feltétel, hogy a $\mathbb{G}$ csoportot is generáló $\mathsf{Előkészít}(\lambda, T)$ mint λ függvénye megfelel az adaptív gyök feltételnek. A feltétel teljesülése esetén egy sikeres támadás valószínűsége λ függvényében elhanyagolható.*

Bizonyítás. Szükséges: tegyük fel, hogy nem teljesül a feltétel. Legyen az A támadó birtokában egy olyan $(\mathcal{A}_1, \mathcal{A}_2)$ algoritmuspár, amelyek hatékonyan végrehajtják a 6. definícióban leírt támadást, és legyen $(\mathbb{G}, H, T) \leftarrow \mathsf{Előkészít}(\lambda, T)$ a korábbi jelöléseket használva. A tetszőleges x bemenetre $\mathcal{A}_1$ elküld $\mathcal{A}_2$-nek egy tetszőleges $w \in \mathbb{G}$ értéket, és a A elküldi wy-t a hitelesítőnek, B-nek. A el fogja hitetni B-vel, hogy $(\mathbb{G}, H(x), wy, T) \in \mathcal{L}_{\mathrm{HKF}}$, pedig $wy \neq H(x)^{2^T}$. B a protokollnak megfelelően elküldi A-nak a $p \in \mathsf{Prímek}(\lambda)$ értéket, amire A-nak egy olyan π-vel kell reagálnia, amire $wy = \pi^p H(x)^r$, ahol $2^T = qp + r$, és $0 \geq r < p$. Ehhez A futtatja $\mathcal{A}_2$-t, hogy egy olyan $u \in \mathbb{G}$-t kapjon, amire $u^p = w$. Végül $\pi = uH(x)^q$ bizonyítja a hamis megoldást, hiszen

$$\pi^p H(x)^r = u^p H(x)^p q H(x)^r = u^p H(x)^{2^T} = wy. \tag{2.2}$$

Elégséges: legyen A birtokában egy olyan $\mathcal{A}$ egy algoritmus, mely ε valószínűséggel talál egy $\mathsf{Előkészít}(\lambda, T)$ által generált $\mathbb{G}$ csoportban olyan $(\mathbb{G}, H(x), y, T) \notin \mathcal{L}_{\mathrm{HKF}}$

négyest, amelyet B helytelenül elfogad. Ekkor konstruálható $(\mathcal{A}_1, \mathcal{A}_2)$ algoritmuspár, amely a 6. definícióban leírt adaptív gyök támadásban ε valószínűséggel sikerrel jár. $(\mathcal{A}_1, \mathcal{A}_2)$ futásideje egy $\mathcal{A}$ futásidejéhez mérhető részből és T egymásutáni $\mathbb{G}$-beli négyzetre emelés idejéből áll.

Első lépésben $\mathcal{A}$ segítségével $\mathcal{A}_1$ generál egy „hamis" $(\mathbb{G}, H(x), y, T)$ négyest, majd ezt $w = y/H(x)^{2^T}$-nel kiegészítve elküldi $\mathcal{A}_2$-nek. Vegyük észre, hogy $w \neq 1$, mint ahogy a támadáshoz szükséges is. Következőnek $\mathcal{A}_2$, miután megkapta a $p \in \mathsf{Prímek}(\lambda)$ számot, $\mathcal{A}$ segítségével generál egy $\pi \in \mathbb{G}$ bizonyítást. Az utolsó lépésben $\mathcal{A}_2$ közzéteszi az $u = \pi/H(x)^q$ értéket, ahol ismét $q, r \in \mathbb{Z}$ és $0 \leq r < p$ úgy, hogy $2^T = qp + r$. Ha π valóban egy hihető bizonyítás, vagyis $y = \pi^p H(x)^r$, akkor könnyen látható, hogy $u^p = w$. □

2.4. Pietrzak-féle protokoll

Egy újabb HKF jelent meg nem sokkal Wesolowski (2018) után Pietrzak (2019) cikkében. Ebben nemcsak költséges az interaktív bizonyítás, de Pietrzak nehezen ellenőrizhető feltételek mellett mutatta meg a protokoll biztonságát. Később Boneh et al. (2018b) általánosította, és egyszerűsítette a protokollt. E fejezet a feljavított verziót közli.

2.4.1. Bizonyítás menete

Ismét jelölje a két felet A és B, ahol A győzi meg B-t arról, hogy adott $(P, x) = ((\mathbb{G}, H, T), x)$ bemenetre az y eredményt ismeri, más szóval azt, hogy $(\mathbb{G}, H(x), y, T) \in \mathcal{L}_{\text{HKF}}$. Az egyszerűbb jelölés végett az általánosság elvesztése nélkül tegyük fel, hogy T egy kettőhatvány. A bizonyítás lépései könnyen általánosíthatók általános $T \in \mathbb{Z}$-re. Ekkor az interaktív bizonyítás az alábbi rekurzív lépéseket követi:

1. B ellenőrzi, hogy y eleme-e $\mathbb{G}$-nek. Ha nem, akkor B megáll, és elutasítja y-t, különben továbbmegy.

   A rekurzió leírásához bevezetjük a g és h változókat, amiket rendre $H(x)$ és y értékkel inicializálunk. A bizonyítás során g, h és T is minden körben változik, és a cél végig B meggyőzése arról, hogy $(\mathbb{G}, g, h, T) \in \mathcal{L}_{\text{HKF}}$.

2. Ha $T = 1$, akkor $h = g^2$ teljesülése esetén B „elfogad” értékkel tér vissza, különben „elutasít” értékkel. Ha $T > 1$, akkor a felek továbbmennek.

3. A kiszámolja a $v = g^{2^{T/2}}$ értéket, majd elküldi B-nek, aki $v \notin \mathbb{G}$ esetén „elutasít”-tal leáll.

   A felek a helyes h ismeretét a helyes v ismeretére vezetik vissza. Ezen v birtokában ugyanis nemcsak $v = g^{2^{T/2}}$ teljesül, hanem $h = v^{2^{T/2}}$ is, sőt, bármely $r \in \mathbb{Z}$ értékre $v^r h = (g^r v)^{2^{T/2}}$ is.

4. B küld A-nak egy tetszőleges $r \in \left\{1, \ldots, 2^\lambda\right\}$ kitevőt.

5. A és B visszatérnek a második lépéshez, ahol ezúttal $(\mathbb{G}, g^r v, v^r h, T/2) \in \mathcal{L}_{\mathrm{HKF}}$ bizonyítása az új feladat.

Ahogy Wesolowski esetében, úgy most is neminteraktívvá tehető a bizonyítási folyamat a Fiat-Shamir-protokoll (Buttyán és Vajda, 2012, 9.3-as fejezet) segítségével. Ehhez feltételezzük egy $J : (\mathbb{G}, g, h, v) \mapsto r \in \left\{1, \ldots, 2^\lambda\right\}$ hasítófüggvény létezését, ahol $g, h, v \in \mathbb{G}$, és amit véletlen orákulumként kezelhetünk. A módosítás az egyetlen olyan lépést érinti, amelyben B küld üzenetet A-nak. A negyedik lépésben ugyanis ha A is és B is J segítségével számítja ki az $r = J(\mathbb{G}, g, h, v)$ kitevőt, akkor A előre le tudja játszani a rekurziót, és közzé tudja tenni a (y, π) párt, ahol $y = H(x)^{2^T}$, illetve π a rekurzió során kapott $\log_2(T)$ darab v értéket tárolja az algoritmus szerinti időrendben. Ezután π birtokában B bármikor vissza tudja játszani a rekurziót, és a protokollnak megfelelően elfogadni vagy elutasítani A eredményét.

2.4.2. Bizonyítás bonyolultsága

A hitelesítő a rekurzió mind a $\log_2 T$ körében kiszámolja $g^r v$-t és $v^r h$-t $\mathbb{G}$-ben, ahol r egy viszonylag kis kitevő. Tehát összesen $2 \log_2 T$ kis hatvány és ugyanannyi szorzat kiértékelése $\mathbb{G}$-ben a hitelesítő feladata.

Jelölje a rekurzió i-edik körében kapott r és v értéket rendre r_i és v_i, ahol $i \in \{1, \ldots, \log_2 T\}$. A bizonyítónak, A-nak minden i-re ki kell számolnia v_i-t, ami a naiv megközelítésben $T/2^i$ négyzetre emelés, összesen $T - 1$ négyzetre emelés, lényegében y kiszámításával megegyező költség. Egy hatékonyabb megközelítés végett írjuk ki

az első néhány i-re v_i-t:

$$\begin{aligned} v_1 &= w^{2^{T/2}}, \\ v_2 &= \left(w^{2^{T/4}}\right)^{r_1} w^{2^{3T/4}}, \\ v_3 &= \left(w^{2^{T/8}}\right)^{r_1 r_2} \left(w^{2^{3T/8}}\right)^{r_1} \left(w^{2^{5T/8}}\right)^{r_2} w^{2^{7T/8}}, \\ v_4 &= \ldots, \end{aligned}$$

ahol $w = H(x)$. Észrevehetjük, hogy v_i előáll $H(x)$-nek a $2^{(2k+1)T/2^i}$ kitevőjű hatványai és $r_1, \ldots, r_{i-1}$ egyszerű függvényeként, ahol $k \in \{0, \ldots, 2^{i-1}-1\}$. A hatványokat mind már y kiértékelése során egyszer ki kell számolni, tehát akár el is lehet őket menteni. Ekkor v_i-hez csak 2^{i-1} kis hatvány és $i-1$ szorzat kiértékelése a költség $\mathbb{G}$-ben. Csakhogy ha minden i-re elmentünk 2^{i-1} értéket, az összesen $T-1$ darab $\mathbb{G}$-beli elem, ami nagy T esetén túl sok lehet. Mivel a naiv megközelítésben a rekurzió első lépései exponenciálisan több műveletet igényelnek, mint a későbbiek, ezért egy köztes út lehet egy választott $d \in \{1, \ldots, \log_2 T\}$ egészre elmenteni a $v_1, \ldots, v_d$-hez szükdéges hatványokat. Ezáltal jelentősen felgyorsíthatjuk a rekurzió első lépéseit, és a maradékot tényleges négyzetre emelések által értékelhetjük ki. Ez végül – a költséges műveleteket megtartva – 2^d kis hatványozás és $T/2^d$ négyzetre emelés $\mathbb{G}$-ben és 2^d méretű tárhely A számára. Ha feltesszük, hogy a kis hatványozás és a négyzetre emelés hasonló bonyolultságú műveletek, akkor $d = \frac{1}{2}\log_2 T$ esetén érjük el az optimum $O(\sqrt{T})$ lépésszámot.

2.4.3. Biztonság

A Pietrzak-protokoll biztonságához is létezik szükséges és elégséges feltétel, melyet Boneh et al. (2018b) tárgyal, ezt mutatjuk be az alfejezet hátralévő részében.

7. Definíció. *Legyen* Gen *olyan valószínűségi változó, amely egy $\lambda \in \mathbb{N}$ biztonsági paraméterhez egy $\mathbb{G}$ véletlen, végesrendű Abel-csoportot rendel. Azt mondjuk, hogy* Gen*re teljesül az alacsony rend feltétel, ha tetszőleges $\mathcal{A}$ algoritmusra a következő támadás λ függvényében elhagyagolható valószínűséggel jár sikerrel. Először* Gen *generál adott nyilvános λ értékre egy $\mathbb{G}$ csoportot. Ezt követően $\mathcal{A}$ talál egy ún. alacsony rendű $m \in \mathbb{G}$ elemet, más szóval egy $(m, d) \in \mathbb{G} \times \mathbb{N}$ párt, amire $m \neq 1$, $m^d = 1$ és $1 < d < 2^\lambda$.*

Egy egyszerű ellenpélda, amire az alacsony rend feltétel nem teljesül, ha $-1 \in \mathbb{G}$, ekkor ugyanis $(-1, 2)$ egy alacsony rendű elem, amit $\mathcal{A}$ azonnal meg is talál.

A Pietrzak-protokoll zárásaként kimondjuk a protokoll biztonságáról szóló tételt, majd a tétel egyik irányát egy intuitív példán keresztül bebizonyítjuk. A tétel másik irányának bizonyítása meghaladja e dolgozat kereteit. A tétel és a bizonyítás megtalálhatók a Boneh et al. (2018b) cikkben is.

2. Tétel. *A Pietrzak-féle HKF biztonságához szükséges és elégséges feltétel, hogy a* $\mathbb{G}$ *csoportot is generáló* $\mathsf{Előkészít}(\lambda, T)$ *mint* λ *függvénye megfelel az alacsony rend feltételnek. A feltétel teljesülése esetén egy sikeres támadás valószínűsége* λ *függvényében elhanyagolható.*

Bizonyítás. Szükséges: tegyük fel, hogy nem teljesül a feltétel, és legyen ismét $(\mathbb{G}, H, T) \leftarrow \mathsf{Előkészít}(\lambda, T)$, ahol $\mathbb{G}$-ben van egy alacsony rendű m elem $1 < d < 2^\lambda$ renddel. Legyen továbbá az A támadó birtokában egy a 7. definícióban leírt $\mathcal{A}$ algoritmus, ami hatékonyan meg is találja m-et. Ekkor tetszőleges helyes $(\mathbb{G}, H(x), y, T) \in \mathcal{L}_{\text{HKF}}$ bemenetre a $(\mathbb{G}, H(x), ym, T) \notin \mathcal{L}_{\text{HKF}}$ négyest $1/d$ valószínűséggel hibásan fogadja el a hitelesítő, B. A ehhez $v = mH(x)^{2^{T/2}}$-t küldi B-nek, amit B helytelenül elfogad, ha olyan r-t választ, amire $r \equiv 2^{T/2} - 1 \pmod{d}$. Ennek valószínűsége $1/d$. Vegyük észre, hogy B azért fogadja el a $(\mathbb{G}, H(x), ym, T)$ négyest, mert $(\mathbb{G}, vH(x)^r, v^r ym, T) \in \mathcal{L}_{\text{HKF}}$. □

2.5. A két protokoll összehasonlítása

Ebben a részben röviden bemutatjuk, hogy a két ismertetett protokoll közül egyik sem jobb objektíven a másiknál, mindkettőnek megvan az erőssége és a gyengesége a másikhoz képest. Két fontos szempontot vizsgálunk: az őszinte bizonyító és a hitelesítő algoritmikus bonyolultságát, illetve a biztonság feltételeinek viszonyát.

2.5.1. Bonyolultság

Szembeötlő különbség, hogy a π bizonyíték Wesolowski esetében mindössze egy $\mathbb{G}$-beli elem, míg Pietrzak esetében $\log_2 T$ darab $\mathbb{G}$-beli elem. Így a hitelesítő az első esetben csupán két hatványozást végez, míg a második esetben $2 \log_2 T$ hatványozást.

Ezzel ellentétben a bizonyító a Wesolowski-protokoll során $O(T)$ lépést igényel π kiszámításához, míg a Pietrzak-protokoll során már $O(\sqrt{T})$ lépésben végez.

2.5.2. Biztonság

A biztonság szempontjából a Pietrzak-protokoll erősebb, ugyanis gyengébb feltételre van szüksége, amit az alábbi állításban formalizálunk.

3. Tétel. *Az adaptív gyök feltételből következik az alacsony rend feltétel.*

Bizonyítás. Legyen Gen – a 6. és a 7. definícióhoz hasonlóan – egy olyan valószínűségi változó, amely egy $\lambda \in \mathbb{N}$ biztonsági paraméterhez egy $\mathbb{G}$ véletlen, végesrendű Abel-csoportot rendel. Tegyük fel, hogy Gen-re nem teljesül az alacsony rend feltétel, és megmutatjuk, hogy ekkor az adaptív gyök feltétel se teljesül rá. Először Gen generál adott nyilvános λ értékre egy $\mathbb{G}$ csoportot, amiben van egy $m \neq 1$ alacsony rendű elem $d > 1$ renddel. Legyen továbbá $\mathcal{A}$ az algoritmus, ami az (m, d) párt hatékonyan meg is találja. $\mathcal{A}$-t felhasználva $\mathcal{A}_1$ elküldi $\mathcal{A}_2$-nek $w = m$-et, aki a kapott $p \in \mathsf{Prímek}(\lambda)$ prímre $u = m^s$-t teszi közzé, ahol $s \equiv p^{-1} \pmod{d}$, ha p nem osztja d-t, különben egy véletlen $u \in \mathbb{G}$-beli értéket tesz közzé. Ez utóbbi eset elhanyagolható valószínűséggel történik meg.

Ha p nem osztja s-t, akkor ez a támadás működik, ugyanis $m^d = 1$-ből és $ps = 1 + kd$-ből következik, hogy

$$u^p = m^{sp} = m^{1-kd} = m(m^d)^k = m.$$

A támadás hatékony, ugyanis s-t és m^s-t hatékonyabb kiszámolni, mint $m^{1/p}$-t. □

2.6. Konkrét Abel-csoportok

Wesolowski (2018) és Boneh et al. (2018b) két csoportot említenek jelöltként: az RSA csoportot (Rivest et al., 1978) és egy a képzetes kvadratikus testekből képzett csoportot (bevezetésnek lásd pl. Kiss, 2014, 5. és 6. fejezetét). Boneh et al. (2018b) kételkedik az utóbbi alkalmasságában, és felvázol egy lehetséges támadást alacsony rendű elem megtalálására. Az RSA csoportról viszont nemcsak ők, de Pietrzak (2019) is megmutatja, hogy megfelel a feltételeknek. Ezért ezt most formálisan is definiáljuk, és az empirikus méréseket is ezzel a csoporttal végezzük.

8. Definíció. *A $\mathbb{G}_{RSA}$ csoportot RSA csoportnak nevezzük, ha p, q prímekre és $N = pq$ egészre $\mathbb{G}_{RSA}$ elemei olyan $(x, -x)$ párok, amikre $x \in (\mathbb{Z}/N)^*$, vagyis x a modulo N szorzáscsoport eleme. A csoportművelet megtartja a két előjelet:* $(x, -x) * (y, -y) = (xy, -xy)$.

Ezt a csoportot választva tehát $\mathsf{Előkészít}(\lambda, T)$ két prímet generál, majd azok szorzatát, N-t, teszi közzé. A $P = (\mathbb{G}, H, T)$ nyilvános paraméterekben pedig $\mathbb{G}$ egy RSA csoport, amit a $(\mathbb{Z}/N)^*$ csoportból képeztünk. Az ebben a csoportban végzett ismételt négyzetre emelést Pietrzak (2019) nyomán RSW feladatnak hívjuk a későbbiekben.

3. fejezet

Implementáció

A diplomamunka célja az elméleti áttekintésen kívül a két részletezett HKF empirikus összehasonlítása. Ennek érdekében szoftvert készítettünk, hogy időmérést tudjunk végezni. Ebben a fejezetben áttekintjük a felmerülő problémákat, az azokra adott megoldásokat, majd a szoftver szerkezetét.

3.1. Használt technológiák

Áttekintjük, hogy mely problémákhoz sikerült hasznos technológiát találni, és hogy hogyan épülnek ezek be a szoftverbe. A 3.1. táblázat foglalja össze az alkalmazott szoftvereket.

3.1.1. C++

Mivel időmérést szeretnénk végezni, mindenféleképp fordított nyelvre kell támaszkodnunk. Fontos még, hogy a nyelv könnyen használható legyen népszerű, hatékony könyvtárakkal, platformfüggetlen legyen, és jó esetben támogasson modern funkcionális és objektumorientált irányokat is. Így esett a választás a C++ nyelvre, hiszen az nemcsak fordított, de az új specifikációk kényelmessé teszik, és a C++ természetes módon kombinálható C és C++ nyelvben írt könyvtárakkal is. A szoftver használatához a C++ legalább 2014-es verziójára van szükség.

	Verzió	Licenc
G++	8.1.0	GNU GPLv3
OpenSSL	1.1.1b	OpenSSL + SSLeay
GNU MP	6.1.2	GNU LPGLv3 + GNU GPLv2
GNU MPFR	4.0.2	GNU LPGLv3 + GNU GPLv3

3.1. táblázat. Az elkészített szoftverhez használt technológiák verziószáma és licence.

3.1.2. OpenSSL és GMP

Az egyik legfontosabb megoldandó problémát a nagy számokkal való műveletek jelentették. Egy RSA csoportban gyakran több ezer bites számokat kell kezelni ahhoz, hogy kriptográfiailag kellően biztonságos legyen, és ezt a mai számítógépek hardveresen nem, csak külön könyvtár használatával tudják megtenni. Ez utóbbira pedig a C++-nak nincs beépített szolgáltatása.

Két népszerű, nagy egészeket (is) kezelő, ingyenesen hozzáférhető könyvtár a GNU Multiple Precision Library (GNU MP vagy GMP Granlund, 2019) a rá épülő GNU MPFR-rel (Fousse et al., 2005) és az OpenSSL (Project). Mind a kettő könyvtárat C nyelven lehet használni, de céljuk más: előbbi kizárólag a sebességre törekszik, utóbbi pedig a kriptográfiai biztonság érdekében kompromisszumokat hoz, pl. a felszabadított memóriaterületet felülírja, nehogy más szoftver később ugyanazon a területen a titkos kulcsokat ki tudja olvasni. Ez utóbbi is hozzátartozik a gyakorlatban egy HKF-hez, ezért választottuk az OpenSSL 1.1.1b verzióját a szoftverünk alapvető funkcióihoz. A 3.2. táblázat ad összehasonlítást a számunkra fontos tulajdonságok alapján.

A Wesolowski-protokoll működéséhez szükség van egy véletlen orákulumra, mely az első 2^λ darab prím egyikét választja egyenletes eloszlással. Ennek pontos megvalósítását a következő szakaszban részletezzük, de itt megjegyezzük, hogy egy rugalmas véletlenszám-generátorra és tetszőleges pontosságú lebegőpontos számokra volt hozzá szükségünk. Ezeket a szoftverünkben a GMP 6.1.2-es verziója és az arra épülő, összetettebb numerikus módszereket tartalmazó GNU MPFR könyvtár 4.0.2-es verziója szolgáltatja.

	OpenSSL	GNU MP + MPFR
Felhasználói felület nyelve	C	C, kis részben C++
Hardveres gyorsítás (hardver nyelvén írt kód)	van	van
Tetszőleges pontosságú lebegőpontos számok kezelése	nincs	van
Véletlenszám-generátor kezelhetősége	körülményes	rugalmas
Prímszámgenerátor	van	nincs
Kitűzött cél	kriptográfiai alkalmazások	hatékonyság minden eszközön

3.2. táblázat. Összehasonlítás az OpenSSL és a GNU MP és MPFR funkciói közt.

3.2. Kézi megoldások

Három olyan algoritmust részletezünk, amit kézzel kellett implementálnunk. Az első a 2.3.2. fejezetben ismertetett egészosztás, majd csoporton belüli hatványozás. A másik kettő a neminteraktív protokollokhoz szükséges két véletlen orákulum.

3.2.1. Egészosztás után hatványozás

A Boneh et al. (2018b) 2.1-es szakaszában részletezett algoritmus hatékony megoldást ad az alábbi feladatra: emelj egy tetszőleges $g \in \mathbb{G}$ elemet q-adik hatványra, ahol $q = \lfloor 2^T/p \rfloor$. A 3.1. algoritmus az általunk használt algoritmust specifikálja, amely a Boneh et al. (2018b) által közzétettnek egy minimálisan hatékonyabb megvalósítása, ugyanis egy elágazás segítségével megkerüli az osztást és a moduláris számítást is, ezeket pusztán egy összehasonlítással helyettesítve.

3.1. algoritmus. Pszeudokód a $\pi = g^q \in \mathbb{G}$ érték kiszámítására, ahol $g \in \mathbb{G}$ tetszőleges, és $q = \lfloor 2^T/p \rfloor$.

Bemenet: $p, T \in \mathbb{N}$, $g \in \mathbb{G}$

$\pi \leftarrow 1 \in \mathbb{G}$

$r \leftarrow 1 \in \mathbb{Z}$

for 1 **to** T **do**

$r \leftarrow 2r$

$\pi \leftarrow \pi^2$

if $r > p$ **then**

$\pi \leftarrow \pi g$

end if

end for

Kimenet: π

3.2.2. Prímszám értékű hasítás

Ebben a szakaszban a diplomamunka többi részétől eltérően π-vel nem a HKF bizonyítását jelöljük, hanem a prímszámláló függvényt. Formálisan legyen $\pi : \mathbb{N} \to \mathbb{N}$, $\pi(n) = |\{0 \leq p \leq n \mid p \text{ prímszám}\}|$.

A 2.3.1. fejezetben, a Wesolowski-protokoll neminteraktív változatához szükség van egy hasítófüggvényre, amely egy 2^λ elemszámú, prímszámokat tartalmazó halmazba képezi a bemenetét. Kézenfekvő lenne az első 2^λ prím, de sajnos nem ismerjük ezeket a halmazokat általános λ-ra, még a $\pi(n) = 2^\lambda$ egyenletet se tudjuk megoldani, hogy a halmazban lévő legnagyobb prímet megismerjük.

Felső korlát a prímek halmazára

Rosser és Schoenfeld (1962) alapján tudjuk, hogy $n \geq 17$-re, ami a gyakorlatban érdekes eseteket számunkra teljesen lefedi,

$$\frac{n}{\log n} < \pi(n) < 1.26\frac{n}{\log n}. \tag{3.1}$$

Szerencsére a Wesolowski-protokoll biztonságosságához csak arra van szükség, hogy *legalább* 2^λ méretű legyen a képhalmaz. Tehát jó nekünk, ha találunk n-t, amire

$\pi(n) \geq 2^\lambda$. Ebből és a (3.1) egyenletből következik, hogy olyan n-t keresünk, amire

$$\frac{n}{\log n} \geq 2^\lambda. \tag{3.2}$$

Egy ilyen n birtokában használhatjuk a legfeljebb n méretű prímszámokat képhalmaznak, ezzel nem lövünk nagyon a 2^λ képméret fölé.

Az egyenlőtlenség megoldásához segítségünkre siet a Lambert-féle W-függvény, mely jelölést először Pólya és Szegő (2010) használta.

9. Definíció. *A Lambert-féle W-függvény a tetszőleges komplex z-re értelmezett $f(z) = ze^z$ függvény inverze.*

Mivel a definícióban használt f függvény nem injektív, ezért W nem egyértelmű. Sőt, W képének függvényében megszámlalhatóan végtelen sok jelölt létezik W-re, melyek közül a -1-gyel indexelt W_{-1}, aminek valós képe $[-\infty, -1]$, a számunkra hasznos W.

W_{-1} segítségével (3.2)-re a legkisebb megoldás $n = \lceil x \rceil$, ahol $x = e^{-W_{-1}(-2^{-\lambda})}$. Ennek bizonyításához tekintsük (3.2)-t egyenletnek, ahol a természetes n-t egy pozitív, valós x-szel helyettesítjük.

$$\begin{aligned}
\frac{x}{\log x} &= 2^\lambda, \\
-\frac{1}{x}\log x &= -2^{-\lambda}, \\
\frac{1}{x}\log\frac{1}{x} &= -2^{-\lambda}, \\
W_{-1}^{-1}\left(\log\frac{1}{x}\right) &= -2^{-\lambda}, \\
\log\frac{1}{x} &= W_{-1}(-2^{-\lambda}),
\end{aligned}$$

amiből következik az eredmény.

Véletlen orákulum pszeudovéletlenszám-generátorból

Most, hogy tudjuk, hogy melyik természetes számig kell prímeket keresnünk, egy prímteszt és az elutasító mintavételezés (lásd pl. Antal et al., 2014, 4.1. fejezetét) együttesével generálhatunk egyenletes eloszlású prímszámokat. Ezután, ha van már egy prímeket generáló pszeudovéletlenszám-generátorunk (PVSZG), akkor azt könnyen determinisztikus függvénnyé, végül közelítőleg véletlen orákulummá alakíthatjuk. A végső függvényünk bemenetét arra használjuk, hogy (újra)inicializáljuk

vele a PVSZG-t, majd meghívjuk egyszer a PVSZG-t, ami generál nekünk egy megfelelő prímszámot, és ez lesz a végső függvényünk kimenete.

Egy elterjedt prímteszt a Miller-Rabin-teszt (Artjuhov, 1966), ami véletlen mintavételezésen alapszik, és nagyon hatékonyan, magas valószínűséggel mondja meg egy természetes számról, hogy prímszám-e. A tesztet a megnövekedett hatékonyság végett mi kiegészítettük azzal, hogy a 2 és 3 prímeken kívül mindegyik $6k \pm 1$ alakú. A gyakorlatban releváns λ értékek mellett a két legkisebb prím elhagyása elhanyagolható különbséget eredményez, cserébe megháromszorozza a véletlen orákulum sebességét.

Az elutasító mintavételezést pedig akkor használhatjuk, ha egy nagy halmazon meglévő eloszlást meg szeretnénk szorítani egy részhalmazra, és abból szeretnénk értékeket generálni. Ez pont a mi esetünk is, ugyanis valamilyen $M \in \mathbb{N}$-re az $\{1, \ldots, M\}$ halmazból tudunk egyenletes eloszlással húzni, de mi ennek egy részhalmazából, az M-nél nem nagyobb prímszámok halmazából szeretnénk húzni. Az elutasító mintavételezés úgy működik, hogy addig húzunk újra és újra a nagy halmazból, amíg véletlen bele nem találtunk a kis halmazba, és akkor megállunk, és csak a legutolsó értéket használjuk.

Összefoglalás

A W_{-1} függvény implementációja során szükséges, tetszőleges pontosságú lebegőpontos számokat többek közt a GNU MP könyvtár valósítja meg, és a saját implementációnkat a GNU Scientific Library (Galassi et al., 2018) `gsl_sf_lambert_Wm1()` függvényének forráskódjára alapoztuk. Ebben a Halley-iterációt (Ortega és Rheinboldt, 1970) használják a függvény kiértékeléséhez. A GNU MP könyvtár szolgáltat továbbá egy hatékony véletlen prímtesztet, és mindemellett egyszerűen (újra)inicializálható a véletlenszám-generátora.

A 3.2. algoritmus foglalja össze a prímszámokat egyenletesen generáló hasítófüggvény működését.

3.2.3. Adott bit hosszú értékű hasítás

A 2.4.1. fejezetben, a Pietrzak-protokoll neminteraktív változatához szükség van egy hasítófüggvényre, amely egy 2^λ elemű halmazba képzi a bemenetét. A legegy-

3.2. algoritmus. Pszeudokód a Wesolowski-prototkoll prímszámkba képező véletlen orákulumához, ami az s bemenethez rendeli a p prímszámot. A W_{-1} függvény a 9. definícióban lévő Lambert-féle W-függvény. A $Miller\_Rabin\_teszt$ „igaz" értékkel tér vissza, ha a bemenete átmegy a Miller-Rabin-teszten, különben „hamis" értékkel. A $VSZG\_inicializal$ inicializálja a véletlenszám-generátort. Az $egyenletes(H)$ valószínűségi változó a véges H halmazból egyenletes eloszlással húz egy véletlen elemet, és azzal tér vissza.

Bemenet: $s, \lambda \in \mathbb{N}$

$x \leftarrow W_{-1}(-2^{-\lambda})$

$n \leftarrow \lceil e^{-x} \rceil$

$max\_int \leftarrow \lceil \frac{n+1}{6} \rceil$

$p \leftarrow 0$

$VSZG\_inicializal(s)$

while not $Miller\_Rabin\_teszt(p)$ **do**

$r \leftarrow egyenletes(\{1, \ldots, k\})$

$j \leftarrow egyenletes(\{-1, 1\})$

$p \leftarrow 6r + j$

end while

Kimenet: p

szerűbb választás a $\{0, \ldots, 2^\lambda - 1\}$ képhalmaz, amivel ekvivalens $\{0,1\}^\lambda$, a λ hosszúságú bitsorozatok halmaza. Ehhez egy megoldás, ha CBC módban (az angol „cipher block chaining" kifejezésből, lásd pl. Buttyán és Vajda, 2012, 5.2. fejezetét) futtatunk egy népszerű blokkrejtjelezőt, mint pl. az AES-t 128, 192 vagy 256 bites kulcshosszal (lásd pl. Buttyán és Vajda, 2012, 2.5. fejezetét).

Az AES 128 bit hosszú bemenetet rejtjelez 128 bit hosszú kimenetbe. A CBC mód pedig 128 bites blokkokra bontja a λ hosszúságú bemenetet – esetlegesen az utolsó blokkot kiegészítve –, és a blokkrejtjelezőt iteratívan használva az i-edik rejtjeles blokkot az $(i-1)$-edik rejtjeles blokk és az i-edik nyílt blokk XOR-olásából kapott üzenet rejtjelezéséből kapjuk. A CBC mód segítségével tehát tetszőleges n-re n hosszú bemenetet tudunk n hosszú kimenetbe kódolni.

A Pietrzak-protokollhoz elég a CBC mód kimenetének λ utolsó bitjét ereménynek venni. Ez biztonságos marad, ugyanis az eddigi kriptoanalízis eredményei alapján az AES kimenetének része hasonlóan nehezen megjósolható, mint a teljes kimenete (Buttyán és Vajda, 2012), illetve a λ utolsó bitre a teljes bemenet hatással van. Az OpenSSL könyvtár implementálja az AES-t és a CBC módot is, és könnyű az eredmény λ utolsó bitjét is venni.

3.3. Könyvtárszerkezet

A diplomamunka mellékletében található forráskód könyvtárszerkezetét az eredmény szempontjából fontos fájlokkal a 3.1. ábra mutatja. Az include mappában vannak a header fájlok típusdefiníciókkal, egy template osztállyal, függvény- és osztálydeklarációkkal, melyeket a src mappában lévő függvény- és metódusdefiníciók használnak. A Makefile fájl a GNU Make program nyelvezetén keresztül specifikálja a fordítás menetét:

1. A src mappában lévő .cpp kiterjesztésű forráskódfájlokból először objektumfáljok készülnek, amelyek az obj mappába kerülnek.

2. Az obj mappában lévő objektumfáljokat összelinkeljük egy dinamikus könyvtárrá, ami a lib mappába kerül.

3. A dinamikus könyvtár ezután már használatra kész. Mi a test mappában lévő tesztfáljokhoz linkeltük a dinamikus könyvtárat, majd a kapott futtatható fáljok segítségével teszteltük a szoftvert.

Az eredmények mérését a `Gyökér` könyvtárban lévő `timing.sh` Bash szkript végezte. Ez a szkript meghívja a `timing_pietrzak.cpp` és a `timing_wesolowski.cpp` fájlok fordításának eredményét, amiket szintén a dinamikus könyvtárhoz kell linkelni. A mérések eredményét pedig az `evaluate.R` R fájl elemzi.

3.4. Osztályszerkezet

A 3.2. ábra mutatja a mérésekhez használt szoftver osztályszerkezetét. Ennek megértését segíti a típusdefiníciókat mutató a 3.1. forráskód, ami az include/types.h fájl tartalma.

```
#include <openssl/evp.h>
#include <memory>
#include <vector>
#include <utility>
#include "zallocator.hpp"

using byte = unsigned char;
using bytevec = std::vector<byte>;
using SecureString = std::basic_string<byte, std::char_traits<byte
    >, zallocator<byte> >;
using EVP_CIPHER_CTX_free_ptr = std::unique_ptr<EVP_CIPHER_CTX,
    decltype(&::EVP_CIPHER_CTX_free)>;
using BN_CTX_free_ptr = std::unique_ptr<BN_CTX, decltype(&::
    BN_CTX_free)>;
using ProofWesolowski = bytevec;
using ProofPietrzak = std::vector<bytevec>;

template<typename proof>
using Solution = std::pair<proof, bytevec>;

using SolutionWesolowski = Solution<ProofWesolowski>;
```

```
19 using SolutionPietrzak = Solution<ProofPietrzak>;
```

3.1. forráskód. Használt típusdefiníciók az include/types.h fejlécben.

Mivel több könyvtárat is használunk nagy egészek kezelésére, egy közös protokoll elősegíti a hibamentes kommunikációt. Ez lett az általános és könyvtárfüggetlen `bytevec` típus, ami pozitív egészeket tárol $2^{\text{CHAR\_BIT}}$-es – C++-implementációfüggő, de általában 256-os – számrendszerben. A `SecureString`, `EVP_CIPHER_CTX_free_ptr` és a `BN_CTX_free_ptr` az OpenSSL-ben használt struktúrák biztonságos memóriakezelését segítik elő. Végül a `SolutionWesolowski` és a `SolutionPietrzak` típusok adják meg a Kiértékel visszatérési típusát a két részletezett protokollban.

További fontos típusok a 3.1. forráskódból és a 3.2. ábráról:

- `BIGNUM`: az OpenSSL nagy egészeket tároló C struktúrája.
- `BN_CTX`: az OpenSSL natív `BIGNUM` tárolója. A neve az angol bignum contextből ered.
- `EVP_CIPHER`: a szimmetrikus rejtjelező metódusok C típusa az OpenSSL-ben.
- `EVP_CIPHER_CTX`: az OpenSSL natív `EVP_CIPHER` tárolója.
- `gmp_randclass`: a GNU MP véletlenszám-generátorának C++ osztálya.
- `mpz_class`: a GNU MP nagy egészeket tároló C++ osztálya.
- `solution`: a `VerifierPietrzak` osztályban a `SolutionPietrzak` aliasa, míg a `VerifierWesolowski` osztályban a `SolutionWesolowski` aliasa.

Most röviden áttekintjük a 3.2. ábrán található osztályok szerepét.

RSWPuzzle

Az 5. definíció jelölése szerint a $P = (\mathbb{G}, H, T)$ nyilvános paramétereket és a λ biztonsági paramétert tárolja. Az RSW az osztály nevében Pietrzak (2019) nyomán az RSA-csoportban végrehajtandó ismételt négyzetre emelés feladatát takarja, és ennek a feladatnak a szerzőiről lett elnevezve (Rivest et al., 1996). Ebben az osztályban

tehát $\mathbb{G}$ egy RSA-csoport (8. definíció), amit a rendjén, N-en keresztül tárolunk, és az egyszerűség kedvéért H az identitás.

A 3.2. ábrán másodikként felsorolt `RSWPuzzle(_lambda, _t, _x, _lambdaRSW)` konstruktor az Előkészít(λ, T) függvényt kiegészítve implementálja, amiben `_lambda` $= \lambda$, `_t` $= \log_2 T$, `_x` a bemenet, és `_lambdaRSW` határozza meg, hogy hány bit hosszú legyen N. Az elsőként felsorolt `RSWPuzzle(_lambda, _t, _x, _N)` konstruktor pedig az előzőhöz hasonló, de inkább tesztelésre alkalmas, ugyanis N értékét itt közvetlenül megadhatjuk. A konstruktorok egyszerűen beállítják az `x`, `N`, `T`, `t` és `lambda` adattagokat.

Hash

A Pietrzak-protokoll neminteraktív változatához szükséges véletlen orákulum osztálya, mely funktor is egyben. Működését a 3.2.3. fejezetben tárgyaltuk.

A lényeges adattagjai közül `lambda` a hasítás/rejtjelezés eredményének bithossza, `cipher` a rejtjelező függvény, `key` és `iv` pedig a rejtjelezés kulcsa és inicializáló vektora. A többi adattagja csak kényelmi vagy hatékonysági célt szolgál.

A 3.2. ábrán elsőként felsorolt `Hash(_lambda, _key_size, _block_size)` konstruktor ajánlott gyakorlati használatra, ugyanis az generálja a `_key_size` méretű kulcsot és a `_block_size` méretű inicializációs vektort. A másodikként felsorolt konstruktor inkább tesztelésre alkalmas, ugyanis ott kézzel adható meg a kulcs és az inicializációs vektor is, ami nem biztonságos.

Az osztály funktorként használható: kerek zárójelek segítségével függvényként használhatók az osztály példányai, ekkor az elsőként kapott `BIGNUM`-ot a másodikként kapott `BIGNUM`-ba hasítja. A függvény meghívása után a másodikként megadott `BIGNUM` módosul, és legfeljebb `lambda` bit hosszúságú lesz.

Hash2Prime

A Wesolowski-protokoll neminteraktív változatához szükséges véletlen orákulum osztálya, mely funktor is egyben. Működését a 3.2.2. fejezetben tárgyaltuk.

A lényeges adattagjai közül `max_int` a 3.2. algoritmusban max_int-tel jelölt változóval egyezik meg, és `primeGen` a GNU MP könyvtár véletlenszám-generátora. A többi adattagja csak kényelmi vagy hatékonysági célt szolgál.

Az osztálynak egy konstruktora van, mely a bemenő `lambda` értékből kiszámolja `max_int`-et. Ezután osztály funktorként használható: kerek zárójelek segítségével függvényként használhatók az osztály példányai, ekkor az elsőként kapott `BIGNUM`-ot a másodikként kapott `BIGNUM`-ba hasítja. A függvény meghívása után a másodikként megadott `BIGNUM` módosul, és nagy valószínűséggel prímszám lesz.

ProverPietrzak

A Pietrzak-protokollban (2.4. fejezet) a Kiértékel függvény funkcióit megvalósító funktor. Egyetlen adattagja csupán hatékonysági célt szolgál. Az osztály egy példányát kerek zárójelekkel meghívva megkapjuk a megoldást és a neminteraktív bizonyítást a bemenetként megadott `VerifierPietrzak` hitelesítő által tárolt RSW fejtörőre az általa tárolt véletlen orákulum felhasználásával. Második paraméterként megadható a 2.4.2. fejezetben tárgyalt d paraméter is, mely a bizonyítás kiszámolásának hatékonyságát befolyásolja.

ProverWesolowski

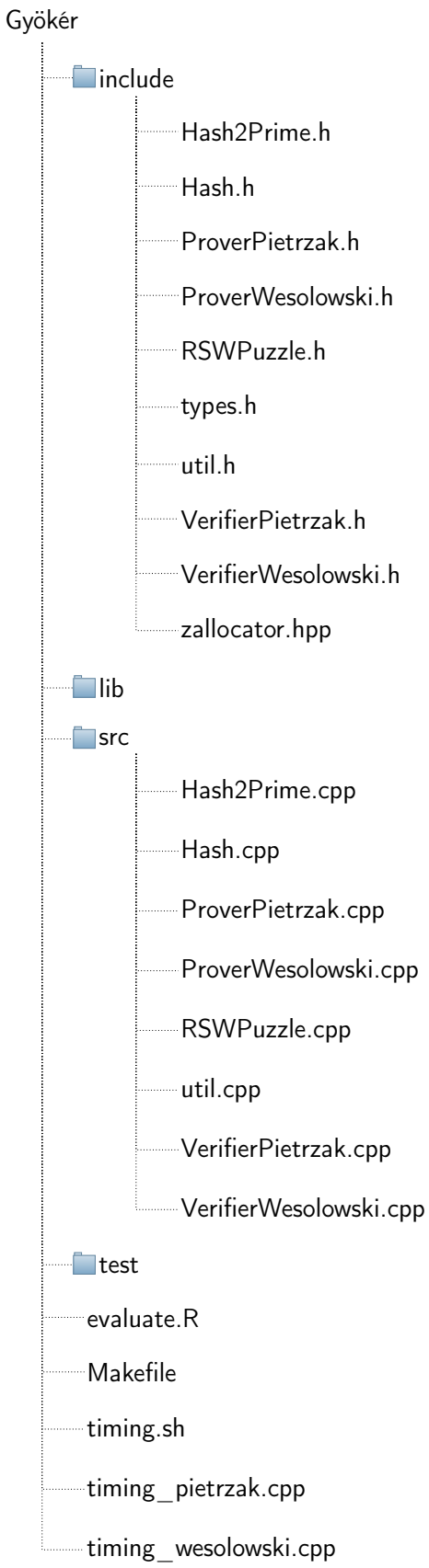
A Wesolowski-protokollban (2.3. fejezet) a Kiértékel függvény funkcióit megvalósító funktor. Egyetlen adattagja csupán hatékonysági célt szolgál. Az osztály egy példányát kerek zárójelekkel meghívva megkapjuk a megoldást és a neminteraktív bizonyítást a bemenetként megadott `VerifierWesolowski` hitelesítő által tárolt RSW fejtörőre az általa tárolt véletlen orákulum felhasználásával.

VerifierPietrzak

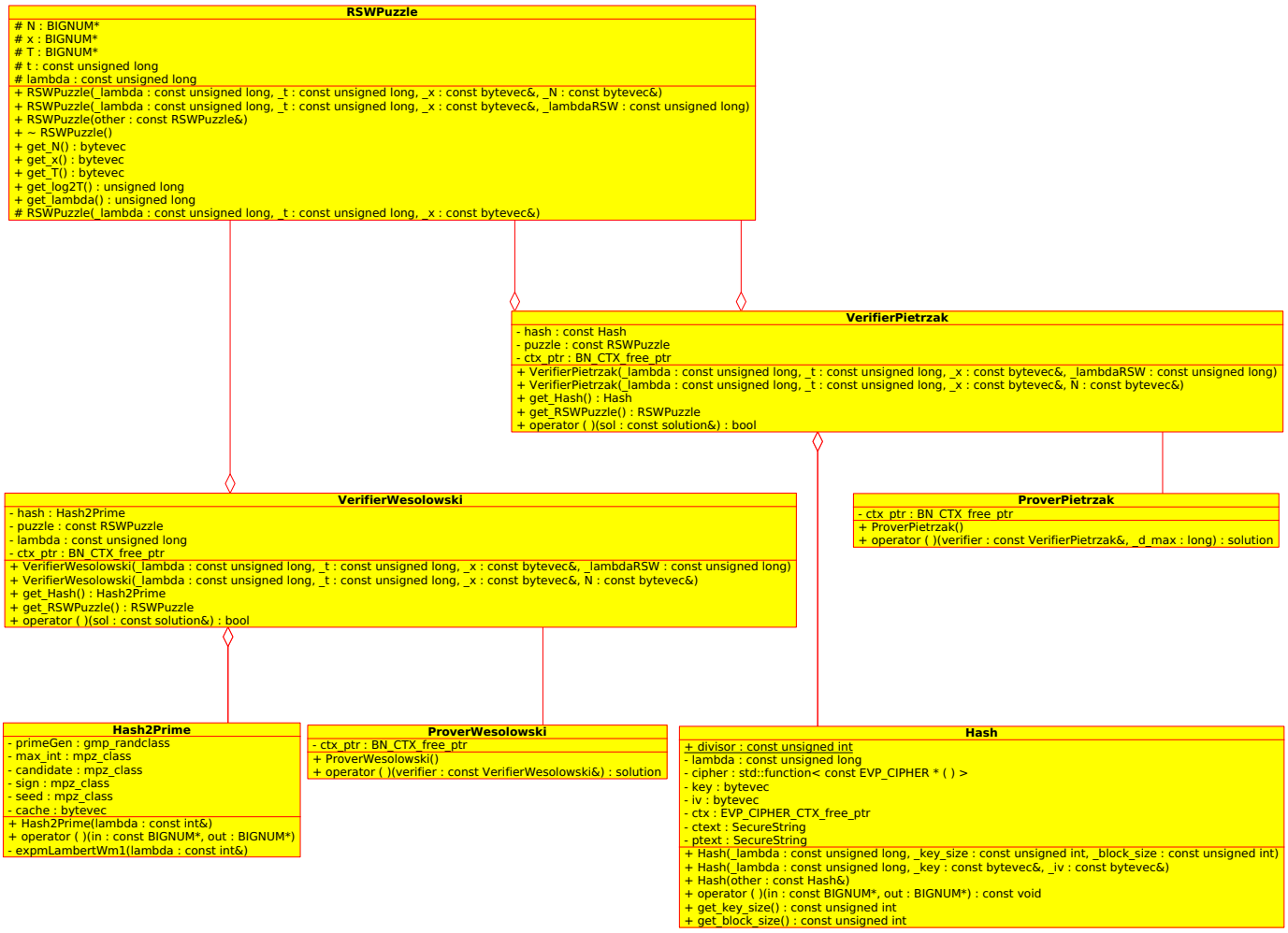
A Pietrzak-protokollhoz (2.4. fejezet) használatos kétfunkciós osztály. Egyrészt tárolja az RSW fejtörőt és a neminteraktív bizonyításhoz használt véletlen orákulumot, másrészt funktorként használva a Hitelesít függvény funkcióit is ellátja.

VerifierWesolowski

A Wesolowski-protokollhoz (2.3. fejezet) használatos kétfunkciós osztály. Egyrészt tárolja az RSW fejtörőt és a neminteraktív bizonyításhoz használt véletlen orákulumot, másrészt funktorként használva a Hitelesít függvény funkcióit is ellátja.



3.1. ábra. A diplomamunka mellékletében található szoftver könyvtárszerkezete a forrásfájlokkal.



3.2. ábra. UML diagram a szoftver osztályszerkezetéről.

4. fejezet

Eredmények

Ebben a fejezeteben összehasonlítjuk a Wesolowski-protokollt és a Pietrzak-protokollt a mérési eredményeink alapján. Az előző fejezetekkel ellentétben, ebben a fejezetben futási idő alatt nem algoritmikus bonyolultságot, hanem órával mért eltelt időt értünk. Itt meg kell említenünk, hogy a célunk természetéből adódóan igazságtalan (Kriegel et al., 2016). Bár mi törekszünk a lehető leggyorsabb könyvtárak és technológiák használatára, a mi programozási készségünk elkerülhetetlen tényezője e munkának, és ezt az eredmények értelmezésénél is figyelembe kell venni.

Az összehasonlítás alapja: azonos késleltetés mellett melyik protokollban hatékonyabb a bizonyító és melyikben hatékonyabb a hitelesítő? Ahogy az elméleti bevezetőben is tárgyaltuk, a véletlen orákulum futási idejét nem feltétlenül számoljuk hozzá a végső futási időhöz. Ezen felül még lényegesek lehetnek a futási idő során az elvégzett memóriaműveletek.

Mindezeket figyelembe véve, a két HKF minden általunk vizsgált paraméterkombinációjára a bizonyító négy-négy, míg a hitelesítő három-három műveletének mérjük meg az összidejét:

- Bizonyító:
  - A műveletekhez szükséges memóriaterület lefoglalásának időtartama, ezután
  - y kiszámításának időtartama az esetlegesen π számításához előre elmentett értékek idejét is beleszámítva, ezután

  - π kiszámításának időtartama, a véletlen orákulummal töltött időt nem beleszámítva, végül
  - a véletlen orákulummal töltött idő.

- Hitelesítő:
  - A műveletekhez szükséges memóriaterület lefoglalásának időtartama, ezután
  - a hitelesítésre fordított idő a véletlen orákulum használatának kivételével, végül
  - a véletlen orákulummal töltött idő.

Az összehasonlítás alapjául szolgáló mérésekhez a következő paraméterértékek minden kombinációját vettük:

$$t \in \{1, 2, 4, 8, 16\},$$
$$\lambda \in \{16, 32, 64, 128\},$$
$$\lambda_{\mathrm{RSW}} \in \{256, 512, 1024, 2048\}.$$

Ezen felül a Pietrzak-protokoll bizonyítójánál még a $d_{\max} \in \{0, 1, 2, 3, t/2, t\}$ értékeket is vettük. A mérést az összes paraméterkombinációval hatszor megismételtük, hogy megbízhatóbb összképet kaphassunk.

Ezekkel a beállításokkal Pietrzak (2019) szerint a gyakorlatban használatos értékek szintjét elérjük – t kivételével, ami inkább 40-ig is felmegy. Továbbá ez a paraméterrács már alkalmas arra, hogy összefüggéseket vizsgáljunk a mért idő és a paraméterek változása között. A 3.2.3. fejezetben említett AES hasítófüggvény kulcsmérete, ami Pietrzak protokolljában jön elő, tapasztalatunk alapján nem befolyásolja a futási időket, ezért azt ebből a fejezetből ki is hagyjuk.

Az összehasonlításhoz egy Linux Ubuntu 16.04 verziójú operációs rendszert futtató, 2,4 GHz-es Intel Core i5-6300U CPU-val és 24 GB RAM-mal rendelkező számítógépet használatunk. A grafikonok elkészítéséhez az R (R Core Team, 2018) nyelv 3.4.4-es verziójában futtatott `tidyverse` (Wickham, 2017) adatelemző és vizualizációs programcsomagot használtuk.

4.1. Eredmény becsületes kiértékelése

A 4.1. ábra a Wesolowski- és a Pietrzak-protokollban a bizonyító számára y kiértékelésének idejét mutatja a paraméterek függvényében. A két grafikon alapján a kiértékelés időtartama λ-tól független, λ_{RSW} – vagyis $\mathbb{G}$ méretének logaritmusa – függvényében nő, és t függvényében szemmértékre exponenciálisan nő. Ez utóbbi egyben azt is jelenti, hogy a kiértékelés ideje $T = 2^t$ függvényében lineárisan nő, ami egy elvárás a HKF-fel szemben. Az eredmények logikusak, ugyanis λ csak a véletlen orákulumot befolyásolja, amit y kiértékelésénél nem használunk, és a négyzetre emelés költsége $\mathbb{G}$ méretének növelésével emelkedik $\mathbb{G}$-ben.

Továbbá ez az idő a Pietrzak-protokollban nem csak y kiértékelésének idejét tartalmazza, hanem a bizonyításhoz, π-hez, felhasznált értékek eltárolásának idejét is. Ezt befolyásolja $d \leq t$, melyet a 2.4.2. fejezetben tárgyalunk. A kapcsolat a 4.1. ábrán is jól látszik: d értékének növelésével a kiértékelési idő is enyhén növekszik. λ és λ_{RSW} növelésével azonban d hatása elhanyagolható lesz, és a gyakorlatban is releváns értékeknél már lényegtelen.

4.2. Bizonyítás becsületes kiértékelése

A 4.2. ábra a Wesolowski- és a Pietrzak-protokollban a bizonyító számára π kiértékelésének idejét mutatja a paraméterek függvényében. A felső grafikon megtévesztésig hasonlít az eredmény becsületes kiértékelését mutató ábra, a 4.1, felső grafikonjára. Ez alátámasztja a 2.5.1. fejezetben tárgyaltakat, miszerint π kiértékelése T-ben lineáris idő alatt történik.

A 4.2. ábra alsó grafikonja viszont más képet mutat, mint a 4.1. ábrabeli megfelelője. Két fontos különbség d és λ hatása az időtartamra. Jól látható, hogy minden t-re van d-nek egy optimális értéke. Viszont az is látható, hogy ez az érték a 2.4.2. fejezetben tárgyaltakkal ellentétben nem $d = t/2$, hanem kisebb. Következésképpen a gyakorlatban a kis hatványozás és az ismételt négyzetre emelés $\mathbb{G}$-ben nem ugyanolyan költséggel hatjhatók végre. λ erősebb hatása az időtartamra is magától értetődő: a kis hatványozás nagyobb nagyobb λ esetén.

Vegyük még a 4.2. ábrán észre, hogy minden panelben $t = 16$-ra az optimális d értékkel a Pietrzak-protokoll bizonyítója a Wesolowski-protokoll bizonyítójánál

gyorsabban számolja ki π-t.

4.3. Hitelesítés

A 4.3. ábra a Wesolowski- és a Pietrzak-protokollban a hitelesítő számára az eredmény és a bizonyítás hitelesítésének időtartamát mutatja a paraméterek függvényében. Először is a felső grafikon több panelében is egy szembetűnő ugrás van $t = 4$-ről $t = 8$-ra. Az ugrás minden bizonnyal az $r \equiv 2^T \pmod{p}$ számításból ered, amit a Wesolowski-protokoll esetén a hitelesítőnek el kell végeznie. Ezt a számítást a `BN_mod_exp` beépített OpenSSL függvénnyel végezzük, és a jelenségre két magyarázatot tudunk elképzelni. Az első magyarázat szerint a függvény a bemenete alapján valószínűleg több különböző numerikus módszer közül választ, és $t = 4$-re más módszert használ, mint $t = 8$-ra. A második magyarázat szerint pedig az OpenSSL által implementált numerikus módszer bizonyos méretű bemenetig hatékonyan tudja kihasználni a processzorhoz tartozó hierarchikus memóriaegységeket, míg egy szint felett ezt már nem tudja. Ekkor ez a szint $t = 4$ és $t = 8$ között lenne.

Az ugrástól eltekintve a Wesolowski-protokoll eredményei nem meglepőek: λ, λ_{RSW} és t függvényében is nő a hitelesítés időtartama, és ezek valóban mind részt vesznek a számításokban.

A 4.3. ábra alapján a Pietrzak-protokoll hitelesítője stabilan jelentősen lassabb a Wesolowski-protokoll hitelesítőjénél. Továbbá Pietrzak esetében t-ben lineárisan nő a hitelesítés időtartama. Mindkét eredmény egybecseng a 2.5.1. fejezetben tárgyaltakkal.

4.4. Memóriaallokáció

A memóriaallokáció a négy esetből egy kivételével mindegyikben elhanyagolható, nagyságrendekkel kisebb a többi műveletnél. A kivételes eset a Pietrzak-protokoll bizonyítója, amelyhez tartozó mérési eredményeket a 4.4. ábra mutatja. Az ábrán láthatjuk, hogy d nagyban befolyásolja az allokációval töltött időt. Ez önmagában várható eredmény, hiszen 2^d számnak foglal helyet a Pietrzak-féle bizonyító. Az ábra

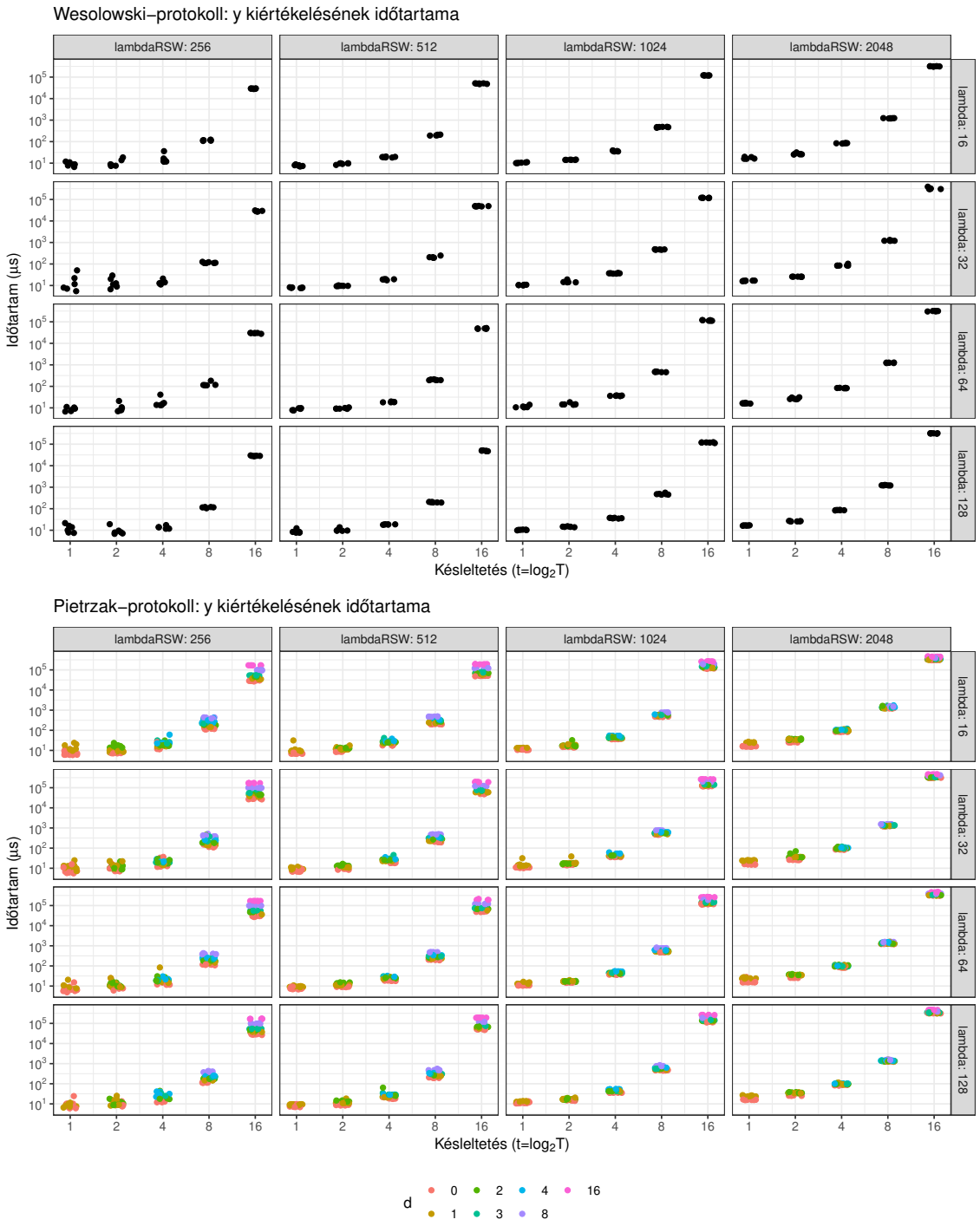
alapján hihető, hogy valóban exponenciálisan növekszik az allokációval töltött idő d függvényében.

4.5. Teljes időtartam

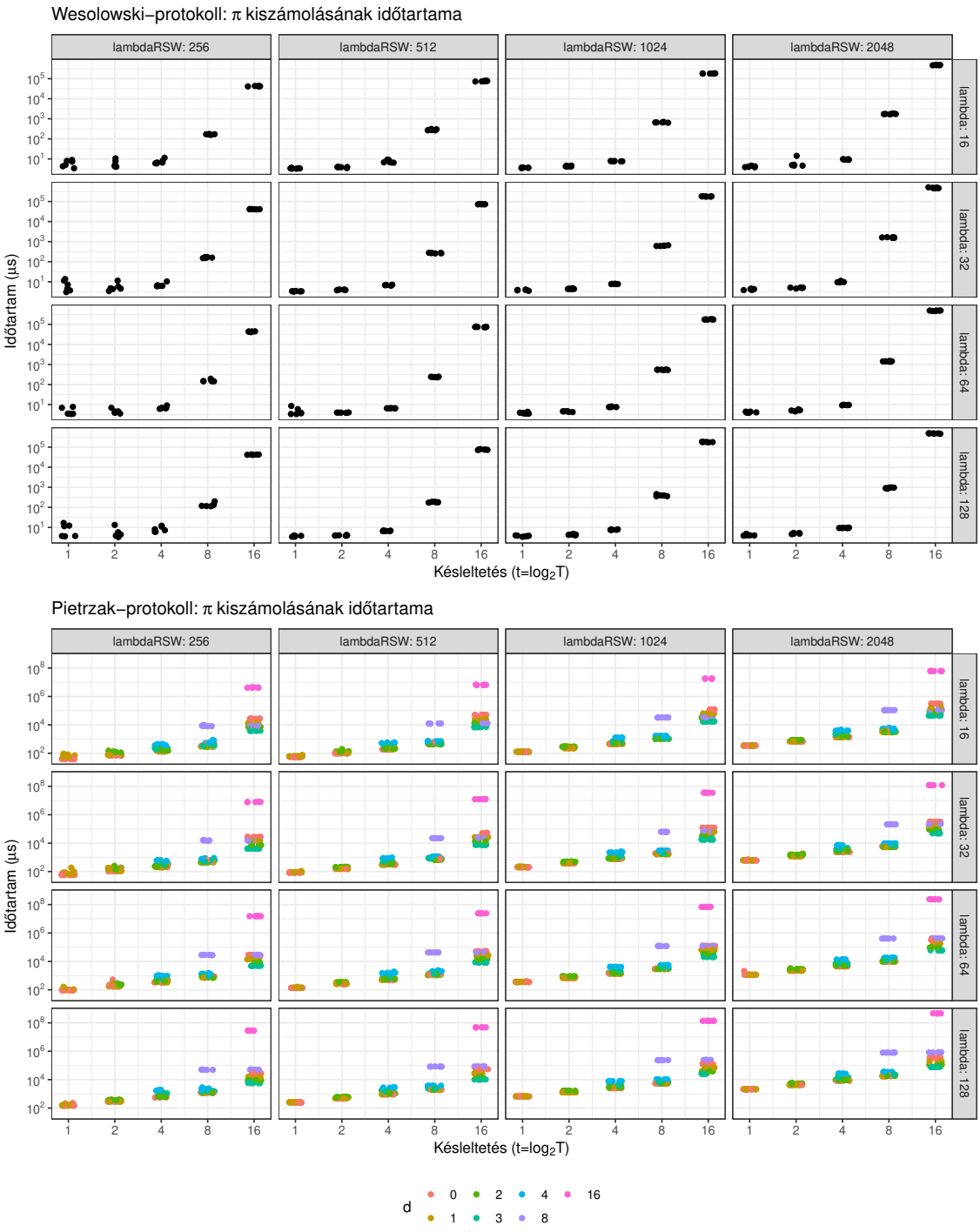
A 4.5. ábrán és a 4.6. ábrán rendre a protokollok bizonyítóinak és hitelesítőinek a teljes futásidejét látjuk. Ebben a véletlen orákulummal töltött idő is benne van. Az ábrát a korábbi ábrákkal összevetve látjuk, hogy a bizonyítók összidejére mindkét protokoll esetén igaz, hogy y kiértékelése a futási időt dominánó művelet.

A hitelesítők esetében kettős képet kapunk: a Pietrzak-protokoll hitelesítője az ideje nagy részét a hitelesítéssel tölti, a Wesolowski-protokoll hitelesítője viszont a véletlen orákulummal. Ez a magyarázata a nagy kilengéseknek is, melyeket a 4.6. ábra felső grafikonján látunk. Ha visszaemlékszünk a 3.2.2. fejezetben ismertetett véletlen orákulum implementációnkra, ami elutasító mintavételezést használ, a kiugró értékek máris érthetőbbé válnak. Az elutasító mintavételezés ugyanis adott bemenetre és véletlenszámgenerátor-állapotra könnyen lehet „szerencsétlen”, és utasíthat el sok értéket az első elfogadásig.

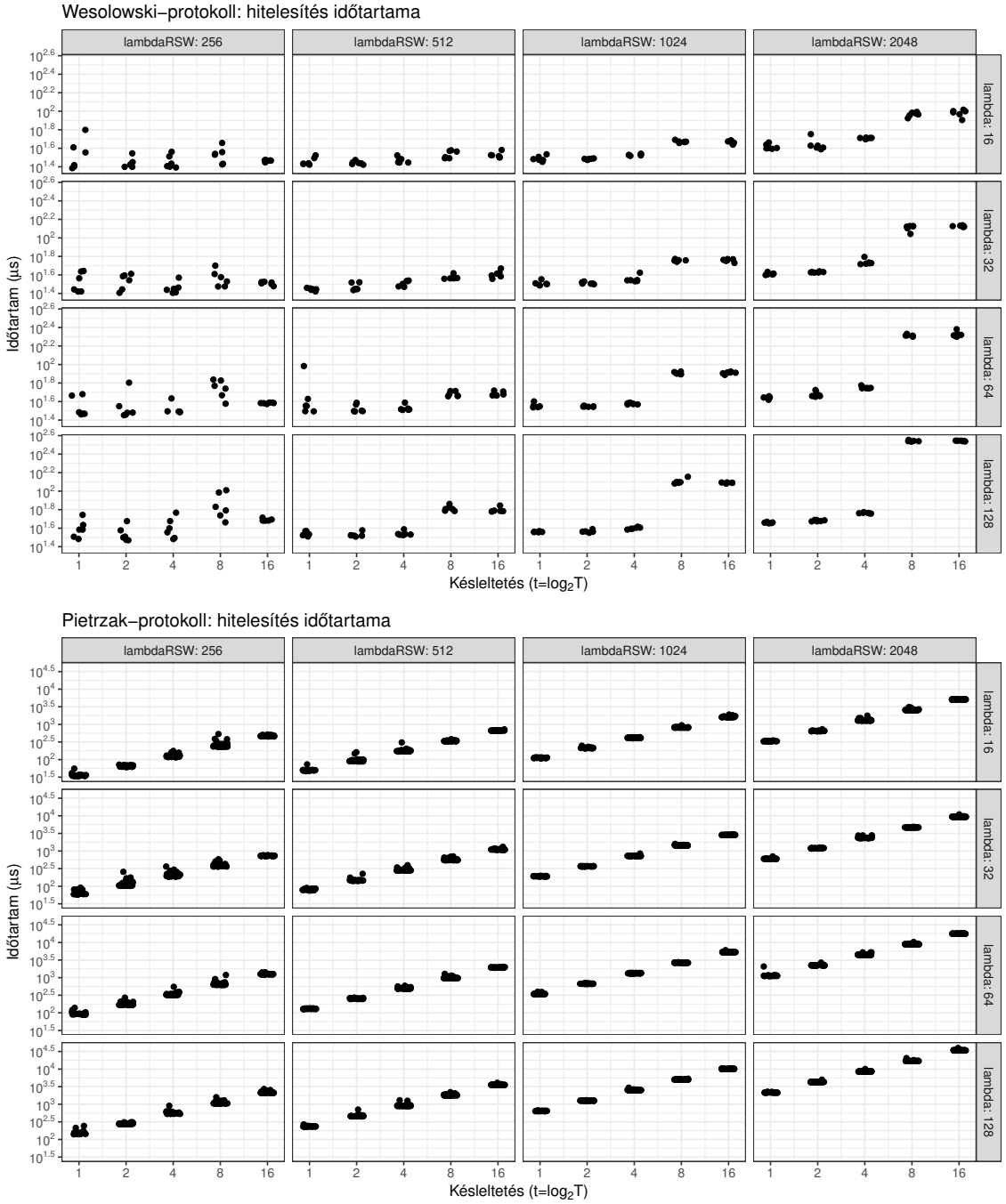
Vegyük észre azonban, hogy Wesolowski hitelesítője még így is nagyságrendekkel hatékonyabb Pietrzak hitelesítőjénél a gyakorlati szempontból érdekes $(\lambda, \lambda_{\mathrm{RSW}}) = (128, 2048)$ esetben.



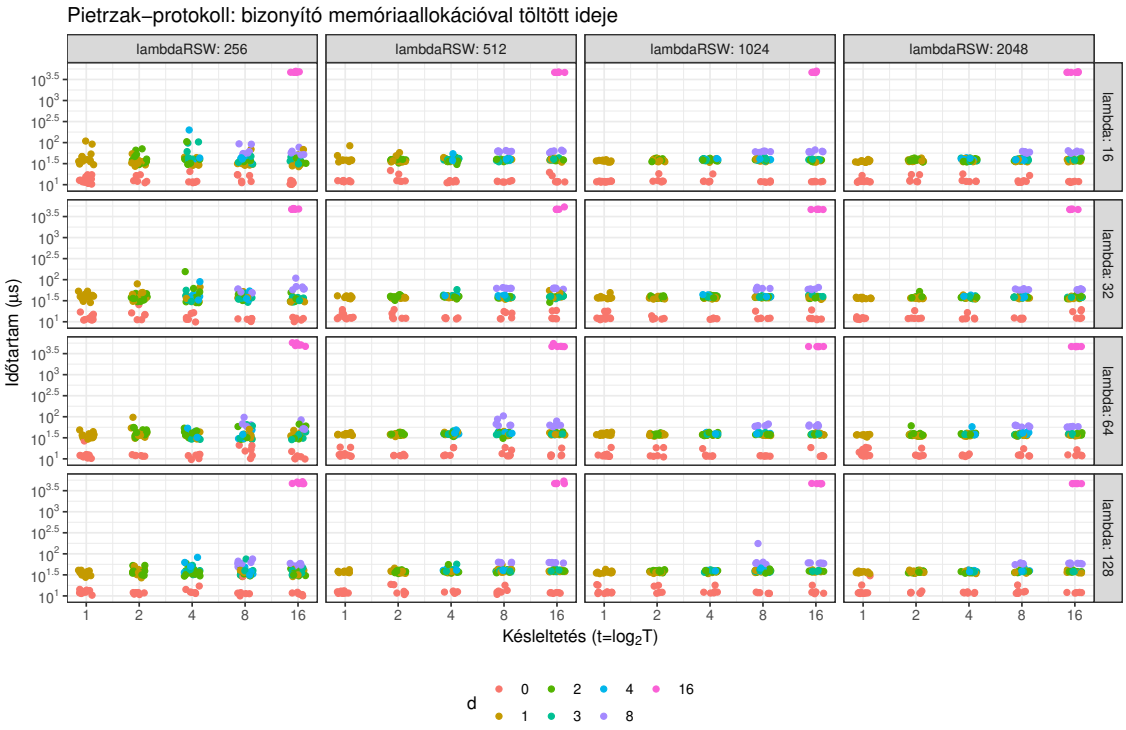
4.1. ábra. A Wesolowski-protokoll (felül), illetve a Pietrzak-protokoll (alul) bizonyítójának az y kiértékelésével töltött ideje a teljes paraméterrácson. Mindkét grafikon 4×4 pontdiagramot tartalmaz, vízszintesen λ_{RSW}, míg függőlegesen λ függvényeként. Pietrzak esetében színek jelölik d értékét. A vízszintes tengelyen a késleltetés mint $\log_2(T)$ paramétere látható, a függőleges tengelyen a kiértékelés időtartama. Mindkét tengelyen logaritmikus skálát alkalmaztunk. A pontok x értékéhez véletlen zajt adtunk a grafikon olvashatóságának növelése érdekében.



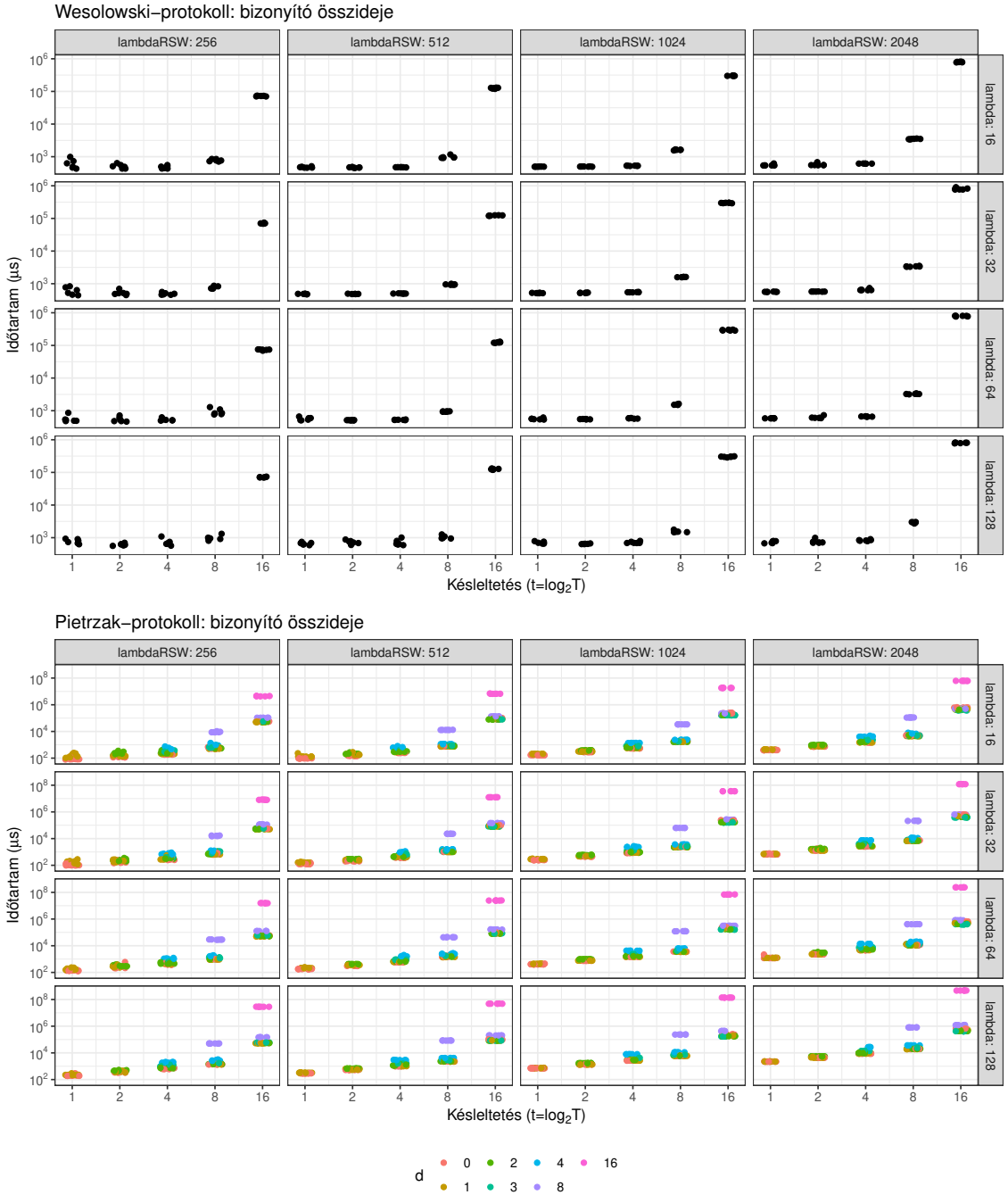
4.2. ábra. A Wesolowski-protokoll (felül), illetve a Pietrzak-protokoll (alul) bizonyítójának a π kiértékelésével töltött ideje a teljes paraméterrácson. Mindkét grafikon 4×4 pontdiagramot tartalmaz, vízszintesen λ_{RSW}, míg függőlegesen λ függvényeként. Pietrzak esetében színek jelölik d értékét. A vízszintes tengelyen a késleltetés mint $\log_2(T)$ paramétere látható, a függőleges tengelyen a kiértékelés időtartama. Mindkét tengelyen logaritmikus skálát alkalmaztunk. A pontok x értékéhez véletlen zajt adtunk a grafikon olvashatóságának növelése érdekében.



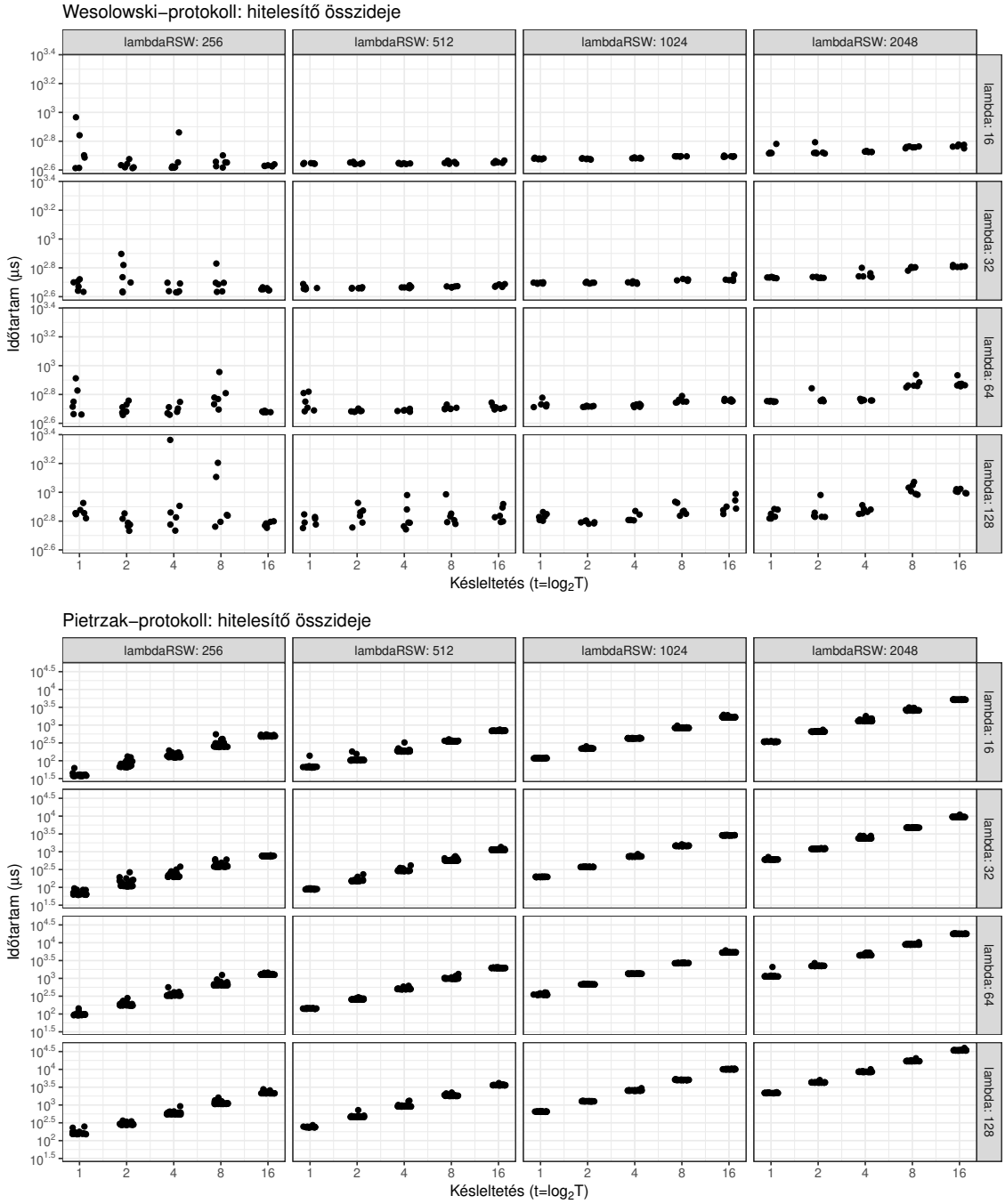
4.3. ábra. A Wesolowski-protokoll (felül), illetve a Pietrzak-protokoll (alul) hitelesítőjének az eredmény hitelesítésével töltött ideje a teljes paraméterrácson. Mindkét grafikon 4×4 pontdiagramot tartalmaz, vízszintesen λ_{RSW}, míg függőlegesen λ függvényeként. A vízszintes tengelyen a késleltetés mint $\log_2(T)$ paramétere látható, a függőleges tengelyen a kiértékelés időtartama. Mindkét tengelyen logaritmikus skálát alkalmaztunk. A pontok x értékéhez véletlen zajt adtunk a grafikon olvashatóságának növelése érdekében.



4.4. ábra. A Pietrzak-protokoll bizonyítójának a memóriafoglalással töltött ideje a teljes paraméterrácson. A grafikon 4×4 pontdiagramot tartalmaz, vízszintesen λ_{RSW}, míg függőlegesen λ függvényeként. A vízszintes tengelyen a késleltetés mint $\log_2(T)$ paramétere látható, a függőleges tengelyen a kiértékelés időtartama. Mindkét tengelyen logaritmikus skálát alkalmaztunk. A pontok x értékéhez véletlen zajt adtunk a grafikon olvashatóságának növelése érdekében.



4.5. ábra. A Wesolowski-protokoll (felül), illetve a Pietrzak-protokoll (alul) bizonyítójának a teljes futásideje a teljes paraméterrácson. Mindkét grafikon 4×4 pontdiagramot tartalmaz, vízszintesen λ_{RSW}, míg függőlegesen λ függvényeként. A vízszintes tengelyen a késleltetés mint $\log_2(T)$ paramétere látható, a függőleges tengelyen a kiértékelés időtartama. Mindkét tengelyen logaritmikus skálát alkalmaztunk. A pontok x értékéhez véletlen zajt adtunk a grafikon olvashatóságának növelése érdekében.



4.6. ábra. A Wesolowski-protokoll (felül), illetve a Pietrzak-protokoll (alul) hitelesítőjének a teljes futásideje a teljes paraméterrácson. Mindkét grafikon 4×4 pontdiagramot tartalmaz, vízszintesen λ_{RSW}, míg függőlegesen λ függvényeként. A vízszintes tengelyen a késleltetés mint $\log_2(T)$ paramétere látható, a függőleges tengelyen a kiértékelés időtartama. Mindkét tengelyen logaritmikus skálát alkalmaztunk. A pontok x értékéhez véletlen zajt adtunk a grafikon olvashatóságának növelése érdekében.

5. fejezet

Összefoglalás

Jelen diplomamunka során rendszereztük és összefoglaltuk a hitelesíthető késleltetett függvények (HKF) megértéséhez és használatához szükséges irodalmat, majd ez alapján szoftveresen megvalósítottunk két előterjesztett HKF-et. Az implementáció során modern technológiákra támaszkodtunk, miközben a hatékonyságra fókuszáltunk. A szoftverünk segítségével szimulációkat végeztünk, aminek eredményeit grafikusan elemeztük, és a részletes következtetéseket az előző fejezetben ismertettük.

A kutatásunk eredményeként egyértelmű kép bontakozik ki arról, hogy a paraméterek hogyan befolyásolják a HKF-ek hatékonyságát, illetve hogy a HKF mely része mennyi erőforrást vesz igénybe. Megmértünk öt különböző résszel töltött időt, melyek az eredmény kiértékelése, a bizonyítás kiértékelése, a hitelesítés, a memóriaműveletek és a véletlen orákulum. Elmondható, hogy az eredményeink egybecsengenek az elmélettel: a bizonyító a Pietrzak-protokoll esetében hatékonyabb, és a különbség a bizonyítás kiszámításának idejéből fakad, míg a hitelesítő a Wesolowski-protokoll esetében hatékonyabb. Megemlítendő még, hogy a Wesolowski-protokoll hitelesítőjének hatékonyságán lényegesen lehet javítani egy hatékonyabb véletlen orákulummal.

A diplomamunka eredménye a segíthet a Wesolowski-protokoll és a Pietrzak-protokoll közti döntésben, illetve a protokollok gyengeségeinek felderítésében, javításában. Az eredményeinkhez fejlesztett szoftver pedig jó kiindulópont lehet egy a bevezetőben ismertetett determinisztikus lottó megvalósításához.

Irodalomjegyzék

P. Antal, G. Hullám, A. Millinghoffer, I. Kocsis, G. Horváth, A. Salánki, A. Pataricza és A. Antos. *Intelligens Adatelemzés*. Typotex, 2014.

M. M. Artjuhov. Certain criteria for primality of numbers connected with the little Fermat theorem. *Acta Arithmetica*, 12:355–364, 1966.

D. Boneh, J. Bonneau, B. Bünz és B. Fisch. Verifiable delay functions. Megjelent: *Advances in Cryptology – CRYPTO 2018*, 10991. kötet, *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018a. doi: 10.1007/978-3-319-96884-1_25.

D. Boneh, B. Bünz és B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018b. `https://eprint.iacr.org/2018/712`.

A. A. Brudno. Entropy and the complexity of the trajectories of a dynamical system. *Moscow Mathematical Society*, 44:127–151, 1982.

L. Buttyán és I. Vajda. *Kriptográfia és Alkalmazásai*. Typotex, 3. kiadás, 2012.

J. Clark és U. Hengartner. On the use of financial data as a random beacon. Megjelent: *EVT/WOTE'10: Proceedings of the 2010 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, 2010.

P. Collet és J.-P. Eckmann. Iterated maps on the interval as dynamical systems. Technical report, Birkhäuser, Boston, 1980.

C. Dwork és M. Naor. Pricing via processing or combatting junk mail. Megjelent: *Advances in Cryptology – CRYPTO '92*, 740. kötet, *Lecture Notes in Computer Science*, pages 139–147. Springer, 1993.

L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier és P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. Rr-5753, INRIA, 2005. inria-00070266.

M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi és R. Ulerich. *GSL: GNU Scientific Library*, 2.5-es verzió, 2018. `https://www.gnu.org/software/gsl/doc/latex/gsl-ref.pdf`.

T. Granlund. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 6.1.2-es verzió, 2019. `https://gmplib.org/`.

H. Hellerman. *Digital Computer System Principles*. Computer Science. McGraw-Hill, 1967.

Y. I. Jerschow és M. Mauve. Non-parallelizable and non-interactive client puzzles from modular square roots. Megjelent: *2011 Sixth International Conference on Availability, Reliability and Security*, pages 135–142. IEEE, 2011.

Y. Katz és J. Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2. kiadás, 2014.

M. G. Kendall és B. Babington-Smith. Randomness and other random sampling numbers. *Journal of the Royal Statistical Society*, 101:147–166, 1938.

E. Kiss. *Bevezetés az Algebrába*. Typotex, 2014.

D. E. Knuth. *Szeminumerikus Algoritumusok*, 2. kötet, *A Számítógép-Programozás Művészete*. Műszaki Kiadó, 1987. Fordította: Freud Róbert, Fiala Tibor, Gerlits János, Hanák Gábor, Nemetz Tibor.

A. N. Kolmogorov. Three approaches to the concept of „the amount of information". *Problems of Information Transmission*, 1:3–13, 1965.

H.-P. Kriegel, E. Schubert és A. Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowledge and Information Systems*, 52:341–378, 2016.

J. C. Lagarias. Pseudorandom numbers. *Statistical Science*, 8(1):31–39, 1993.

P. L'Ecuyer. History of uniform random number generation. Megjelent: *Proceedings of the 2017 Winter Simulation Conference*, pages 202–230, 2017.

A. K. Lenstra és B. Wesolowski. A random zoo: Sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015. `https://eprint.iacr.org/2015/366`.

J. M. Ortega és W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, Inc., 1970.

K. Pietrzak. Simple verifiable delay functions. Megjelent: *10th Innovations in Theoretical Computer Science Conference*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

G. Pólya és G. Szegő. *Feladatok és Tételek az Analízis Köréből I–II.* Typotex, 2010.

T. O. Project. OpenSSL—cryptography and SSL/TLS toolkit. `https://www.openssl.org/`.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Bécs, Ausztria, 2018. `https://www.R-project.org/`.

M. O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27:256–267, 1983.

R. Rivest, A. Shamir és L. Adleman. A method for obtaining digitalsignatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

R. L. Rivest, A. Shamir és D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996.

J. B. Rosser és L. Schoenfeld. Approximate formulas for some functions of prime numbers. 6(1):64–94, 1962.

B. Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. `https://eprint.iacr.org/2018/623`.

H. Wickham. *tidyverse: Easily Install and Load the 'Tidyverse'*, 2017. `https://CRAN.R-project.org/package=tidyverse`. R csomag, 1.2.1-es verzió.