Szakdolgozat

Szerzők: Mezei Botond, Szürti Szilárd Dávid

Témavezetők: Bagossy Attila, Vécsi Ádám

Debreceni Egyetem

Informatikai Kar

Számítógéptudományi Tanszék

Attribute-based Encryption WASI-alapú platformfüggetlen implementációja

Lezárva: 2020. 04. 27.

Tartalomjegyzék

Köszönetnyilvánítás

Köszönjük témavezetőinknek, Bagossy Attila és Vécsi Ádám doktoranduszoknak a dolgozatunk megírásában nyújtott segítségüket. Dolgozatunk iránymutatásuk, javaslataik és számtalan kérdésünk megválaszolásának köszönhetően készülhetett el.

A kutatást részben a „Integrált kutatói utánpótlás-képzési program az informatika és számítástudomány diszciplináris területein” (EFOP-3.6.3-VEKOP-16-2017-00002), részben az „IoT rendszerek biztonságát növelő technológiák” projekt (2018-1.2.1-NKP-2018-00004) című projekt támogatta. Utóbbi projekt a Nemzeti Kutatási és Innovációs Alapból biztosított támogatással, a "Nemzeti Kiválósági Program: 2018-1.2.1-NKP" pályázati program finanszírozásában valósult meg.

NEMZETI KUTATÁSI, FEJLESZTÉSI ÉS INNOVÁCIÓS HIVATAL
AZ INNOVÁCIÓ LENDÜLETE
AZ NKFI ALAPBÓL MEGVALÓSULÓ PROJEKT

SZÉCHENYI 2020
MAGYARORSZÁG KORMÁNYA
Európai Unió
Európai Szociális Alap
BEFEKTETÉS A JÖVŐBE

1. fejezet

Bevezetés

A dolgozatunkban bemutatjuk a CryptID (2020) nyílt forráskódú, platformfüggetlen kriptográfiai könyvtár általunk készített bővítését. A könyvtár WebAssemblyre épülő platformfüggetlenségét az eszközök még szélesebb körében alkalmazható, WebAssembly System Interface (WASI) alapú platformfüggetlenségre fejlesztettük tovább. A dolgozatunkat megelőzően a könyvtár Identity-based Encryption és Signature algoritmusokat (IBE és IBS) tartalmazott, melyeket kiegészítettünk egy elliptikus görbéken alapuló Attribute-based Encryption (ABE) implementációval. Ezáltal megalkottuk az általunk fellelhető első platformfüggetlen, elliptikus görbéken alapuló, szabadon hozzáférhető implementációt.

1.1. Attribute-based Encryption

Az Attribute-based Encryption egy olyan nyilvános kulcsú kriptográfiai eljárás, melynek újdonságtartalma a titkos kulcshoz társított attribútumlistában és a titkosításhoz használt, konjunkciót és diszjunkciót lehetővé tevő, attribútumszabályokat megadó hozzáférési fában *(access tree)* rejlik. Ennek köszönhetően a titkosítás személyre szabott módon, a visszafejtésre jogosultak identitása helyett attribútumaik ismeretével történhet. Az első koncepciót Sahai és Waters írta le (Sahai & Waters, 2004), majd 2007-ben egy, a gyakorlati megvalósítást részletező, Ciphertext-Policy ABE publikációt Bethencourt, Sahai és Waters tettek közzé (Bethencourt, Sahai & Waters, 2007).

1.2. Motiváció

A CryptID korábbi platformfüggetlensége is lehetőséget biztosított a széles körben (asztali számítógép, okostelefon, stb.) való használhatóságra, azonban ez az esetek többségében

böngészőt is igényelt. Amennyiben mégsem igényelt böngészőt, akkor sem állt rendelkezésre kiforrott támogatás a böngészőn kívüli működésre. Azonban a WebAssembly System Interface megjelenésével (Clark, 2019) ez a helyzet megváltozott, így már elérhető egy stabil, egységes megoldás, amit alapul véve a webböngészőkön túl is működőképessé tehető a könyvtár. Ennek következtében láttuk úgy, hogy a platformfüggetlenség kibővítése és annak stabillá tétele a programkönyvtárat felhasználó fejlesztők előnyét szolgálná.

A CryptID által ilyen módon megalapozott platformfüggetlenség és egyszerű használhatóság olyan előnyöket mutattak meg, amellyel egyik általunk fellelhető, nyilvánosan elérhető Attribute-based Encryption könyvtár sem rendelkezik. Ebből következett számunkra, hogy az egyébként is nagyon kevés, az említett előnyökkel nem rendelkező ABE implementáció (*OpenABE*, 2020; *BSW CPABE Toolkit*, 2006) helyett a CryptID ilyen téren való bővítése számos alkalmazási lehetőséget nyitna meg.

1.3. A dolgozat felépítése

A dolgozat első három fejezete alkotja a szükséges alapismeretek bevezetését. Vázoljuk az elliptikus görbe kriptográfia alapjait, majd az ABE-t érintő matematikai hátteret. Ezután a technológiai hátteret biztosító WASI-t mutatjuk be. Az ezt követő fejezetben pedig a dolgozat eredményét jelentő implementációt, valamint az általunk végzett teljesítménytesztek és a platformfüggetlenség demonstrációját ismertetjük. Zárásként implementációnk lehetséges felhasználási módját mutatjuk be.

2. fejezet

Elliptikus görbe kriptográfia

A fejezet célja bemutatni az elliptikus görbére alapuló kriptográfia dolgozatunkat érintő részleteit, az elliptikus görbék alapvető tulajdonságait. Továbbá ismertetjük, hogy az Attribute-based Encryption implementálása során miért választottuk az elliptikus görbe kriptográfiát.

Az elliptikus görbék múltja régre nyúlik vissza, használatuk a kriptográfiában is évtizedek óta kutatások tárgya. Ennek oka egyszerű: bár az elliptikus görbék matematikai háttere bonyolultabb, azonban kisebb kulcsmérettel teszik lehetővé ugyanazt a biztonsági szintet, mint egyéb, napjainkban használt titkosítási eljárások, amelyek egyéb matematikai problémákon alapulnak, mint például a prímfaktorizáció (RSA). Az elliptikus görbék kriptográfiára való alkalmassága az *elliptikus diszkrét logaritmus problémából* ered. A kisebb kulcsméret miatt hatékonyabb implementációt nyújt mind a tárterület, mind a sebesség szempontjából (Bos és mtsai., 2014).

Az elliptikus görbék előnyeikből adódóan mára a gyakorlatban is alkalmazásra találtak. Példaként említhetjük a közismert *Bitcoin* kriptovalutát, valamint az *SSH* és a *TLS* protokollokat, melyek mind támogatják az elliptikus görbe kriptográfiát (Bos és mtsai., 2014). Kiemelendő, hogy a TLS legújabb, 1.3 verziószámú protokolljából a statikus RSA és Diffie-Hellman kulcscsere módok el lettek távolítva. A támogatás ebben a PSK-n (*pre-shared key*) kívül az elliptikus görbék vagy véges testek feletti DHE-re (*Diffie-Hellman Ephemeral*) szűkült (Rescorla, 2018).

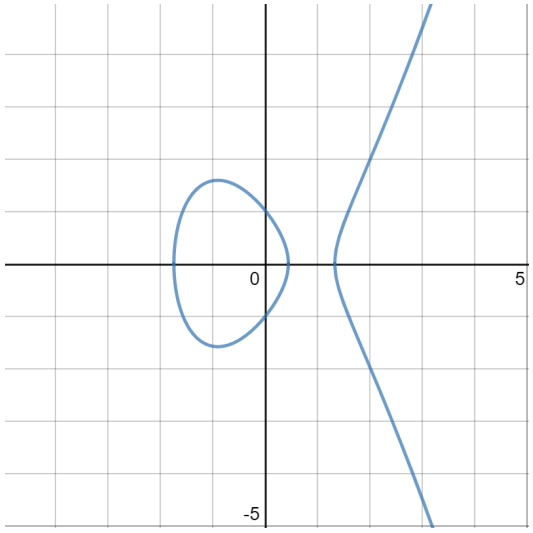
2.1. Elliptikus görbék elmélete

Az alfejezetben az elliptikus görbék általános áttekintése helyett a kriptográfiában nagy jelentőséggel bíró Weierstrass elliptikus görbékre szűkítjük ismertetőnket. Az alábbi képlet írja le az $\mathbb{F}$ test felett értelmezett elliptikus görbét, melyet $(x, y) \in \mathbb{F}^2$ pontok alkotnak

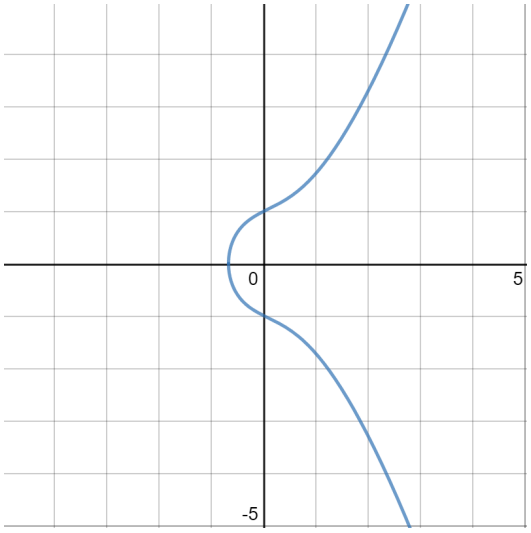
(*feltételezve, hogy a görbe karakterisztikája nem 2 vagy 3*):

$$y^2 = x^3 + ax + b \text{ (Koblitz, 1987).}$$

Két ilyen görbét ábrázolnak az alább látható, 2.1a és 2.1b ábrák.



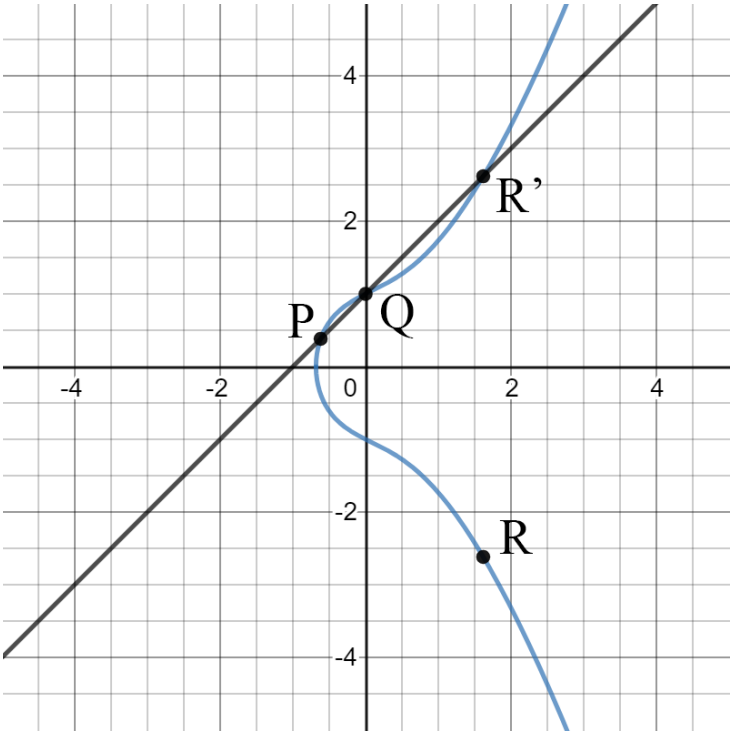
(a) $y^2 = x^3 - 2.5x + 1$ görbe egyenlet



(b) $y^2 = x^3 + x + 1$ görbe egyenlet

A görbéhez tartozik továbbá egy absztrakt O pont, ami a *végtelenben* elhelyezkedő pontot jelenti. Ezzel együtt a görbe pontjai megfelelően definiált összeadás művelettel Abel-csoportot alkotnak, melynek neutrális eleme az O pont.

Két pont összeadása. Adott az E elliptikus görbe az $\mathbb{F}$ test felett ($E(\mathbb{F})$). Adottak $P = (x_1, y_1)$, $Q = (x_2, y_2)$, az elliptikus görbe két pontja. A P és Q pontokon átmenő egyenes a görbét egy harmadik, R' pontban metszi. Az R' pontot az x tengelyre tükrözve megkapjuk az $R = P + Q$ pontot a 2.2 ábrán látható módon. Amennyiben $P = Q$, az R' pont a P pontot érintő egyenes metszéspontja (Miller, 1985).



2.2. ábra. A P és Q pontok összeadása elliptikus görbén.

Pont skalárral szorzása. Egy P pont k skalárral való szorzására kézenfekvő megoldás az önmagával $k-1$ alkalommal végzett összeadás, azonban ennél léteznek jóval hatékonyabb módszerek is.

Az egyik ilyen hatékonynak tekinthető algoritmus például a *wNAF*. Ez a *non-adjacent form (NAF)* számábrázolási módszerre épülő elliptikus görbe skalárszorzási algoritmus. A NAF az aláírt bitek egy reprezentációs módja, melynek a lényege, hogy a nem nulla értékek nem lehetnek szomszédosak (Hankerson, Menezes & Vanstone, 2006). A *wNAF* skalárszorzó algoritmus hatékonyságát annak köszönheti, hogy egy kis mennyiségű elő-feldolgozást használ, valamint a tényt, hogy az elliptikus görbe két pontjának összeadása és kivonása ugyanolyan költséggel jár (Benger, Van de Pol, Smart & Yarom, 2014).

3. fejezet

Attribute-based Encryption

Az Attribute-based Encryption egy nyilvános kulcsú titkosítási eljárás, amely esetén a titkos kulcs egy előre meghatározott attribútumlistával rendelkezik, míg a titkosítandó szöveg rejtjelezése egy hozzáférési fával *(access tree)* történik. Ezáltal a visszafejtés csak abban az esetben lehetséges, ha az attribútumlista az említett hozzáférési fát kielégíti. A fejezet célja az ABE lényegi tulajdonságainak és az ehhez nélkülözhetetlen kriptográfiai előfeltételeknek a tárgyalása, melyhez a Bethencourt, Sahai és Waters publikációját vettük alapul (2007).

3.1. Párosítás-alapú kriptográfia

A párosítás-alapú kriptográfia lényege két kriptográfiai csoport felett értelmezett egyes problémák párosítása, melynek eredményeképpen egy újabb csoport felett értelmezett probléma áll elő. Az így keletkezett harmadik csoportbeli elem különböző matematikai azonosságok révén használható fel a titkosítási eljárásokhoz, potenciálisan lehetővé téve két nehéz probléma párosításából egy újabb, könnyebb létrejöttét.

G_0 és G_1 legyenek p rendű multiplikatív ciklikus csoportok, g a G_0 csoport generátora, $e : G_0 \times G_0 \to G_1$ egy bilineáris leképezés a következő tulajdonságokkal:

- **Bilineáris.** Minden $u, v \in G_0$ és $a, b \in \mathbb{Z}_p$ esetén $e(u^a, v^b) = e(u, v)^{ab}$.
- **Nem elfajuló.** $e(g, g) \neq 1$.
- **Hatékonyan kiszámítható.**

Megjegyzés. A G_0 csoportban végzett műveleteknek is hatékonyan kiszámíthatóknak kell lenniük (Lynn, 2007).

3.2. Személyre szabott titkosítás

A korábban említettek szerint az attribútum-alapú titkosítás lényege, hogy a titkosítás egy ÉS/VAGY hozzáférési fa alapján történik, például:

(*Programozó* **VAGY** *Projektmenedzser*) **ÉS** *Szervezeti tag*.

Ez alapján a titkosított szöveg visszafejtése annak lesz lehetséges, aki *Programozó* vagy *Projektmenedzser*, emellett pedig *Szervezeti tag* attribútumokkal rendelkezik. Rendelkezhet további attribútumokkal is, azonban e feltételeknek mindenképpen teljesülnie kell a sikeres visszafejtéshez. Azaz a titkosítónak nem kell ismernie a visszafejtésre jogosultak identitását, elég azoknak a releváns attribútumait. Fontos továbbá, hogy két különböző, a feltételeket külön-külön ki nem elégítő kulcs kombinálásából ne jöhessen létre a feltételeket kielégítő kulcs, például a fenti hozzáférési fa esetén a *Programozó* és a *Szervezeti tag* kulcsokat tartalmazó kulcsok kombinálásából.

3.2.1. BSW Ciphertext-policy Attribute-based Encryption

Az Attribute-based Encryption első koncepcióját 2004-ben Sahai és Waters írták le (Sahai & Waters, 2004). Ezt az Identity-based Encryption (IBE) (Shamir, 1985) egy új sémájaként mutatták be, úgynevezett Fuzzy IBE néven, melyben az identitást leíróattribútumok listája jelentette. 2007-ben Bethencourt, Sahai és Waters adták ki *Ciphertext-Policy Attribute-Based Encryption* című publikációjukat, melynek egyik legnagyobb előnye a korábbi ABE eljárásokkal szemben az volt, hogy a titkosított adat akkor is biztonságosnak tekinthető, ha a központi tárolószerver veszélybe kerül (Bethencourt és mtsai., 2007).

A titkosítási eljárás arra a problémára ad kriptográfiai megoldást, amikor egy adatnál meg kell határozni, kik férhessenek majd hozzá. A titkosító ezt egy hozzáférési fa segítségével teszi meg, míg a visszafejtő a privát kulcs generátortól (*private key generator*, PKG) beszerzett privát kulcsával visszafejti azt (Kate & Goldberg, 2010), miután személyes attribútumait autentikálta. A PKG egy *megbízható fél*, melynek feladata, hogy a mesterkulcs segítségével privát kulcsot hozzon létre egy megfelelő autentikációs folyamat után. Így biztosítható, hogy csak a hozzáférési fát kielégítő attribútumokkal rendelkező személyek férhessenek hozzá a kívánt privát kulcsokhoz.

Az eljárások használata előtt szükség van egy úgynevezett inicializáló lépésre. Ez hozza létre a publikus- és mesterkulcsokat. A publikus kulcs tartalmazza a publikus paramétereket, melyekre szükség van a titkosító eljárásokhoz. A mesterkulcsot csak a privát kulcs generátor ismerheti a megelőző bekezdésben leírtak miatt. A privát kulcs segítségével visszafejthető a titkosított adat, amennyiben kielégíti az ahhoz tartozó hozzáférési fát.

A 3.1 táblázat tartalmazza a továbbiakban használt legfontosabb jelöléseket és azok magyarázatát.

Paraméter neve	**Típusa**	**Megjegyzés**
$E(\mathbb{F}_p)$	elliptikus görbe	-
G_0	az $E(\mathbb{F}_p)$ ciklikus részcsoportja	generátora P
$G_{\mathbb{T}}$	az $E(\mathbb{F}_p)$ ciklikus részcsoportja	generátora $e(P,P)$
e	párosítás	$e : G_0 \times G_0 \to G_{\mathbb{T}}$
P	elliptikus görbe pontja	$P \in E(\mathbb{F}_p)$
αP	elliptikus görbe pontja	$\alpha P \in E(\mathbb{F}_p)$
βP	elliptikus görbe pontja	$\beta P \in E(\mathbb{F}_p)$
$threshold(T)$	T elem *threshold* értéke	-
$Polynom(d, s, PublicKey)$	d fokú x polinomot ad vissza	$q_x(0) = s$
$attribute(x)$	x levélelem attribútuma	-
$parent(x)$	x faelem szülője	-
$index(x)$	x faelem indexe	-
$H(x)$	kriptográfiai *hash* függvény	$H : \{0,1\}^* \to G_0$
$leaf(x)$	*true*, ha x levélelem	-
$leaves(x)$	x fa összes levéleleme	-
$children(x)$	x gyerekei	-

3.1. táblázat. A pszeudókódok kiegészítőtáblázata.

Setup. Létrehozza a publikus kulcsot és a mesterkulcsot, bemenete a k biztonsági szintet megadó paraméter.

A *Setup* pszeudokódját az Algoritmus 1 adja meg.

Algoritmus 1 Setup

procedure SETUP(k)

p prím inicializálása ▷ k paraméternek megfelelően

$E(\mathbb{F}_p)$ elliptikus görbe inicializálása

$P = randomPoint(E(\mathbb{F}_p))$ ▷ $P \in E(\mathbb{F}_p)$

G_0 csoport generátora legyen P

$e : G_0 \times G_0 \to G_{\mathbb{T}}$ párosítás kiválasztása

$G_{\mathbb{T}}$ csoport generátora legyen $e(P, P)$

$\alpha = randomInteger(p)$ ▷ $\alpha \in \mathbb{Z}_p$

$\beta = randomInteger(p)$ ▷ $\beta \in \mathbb{Z}_p$

$PublicKey = (G_0, P, h = \beta P, f = (1/\beta)P, e(P, P)^{\alpha})$

$MasterKey = (\beta, \alpha P)$

return *PublicKey, MasterKey*

end procedure

Encrypt. Az adat titkosításáért felel, bemenete a *PublicKey* publikus kulcs, az M adat és a T hozzáférési fa.

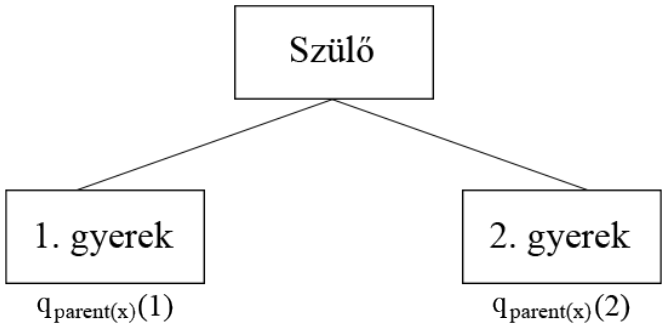
A titkosítási folyamat része a bemeneti hozzáférési fa polinomjainak megalkotása, melyért a `ComputeTree` függvény felel.

Egy fa minden gyerekkel rendelkező eleme rendelkezik egy *threshold* értékkel, mely **VAGY** fák esetén 1, **ÉS** fák esetén pedig az elem gyerekeinek száma.

A polinomok esetén az együtthatók a következőképpen vannak meghatározva:

- a nulladik fok esetén a polinomot létrehozó függvény paramétere (s),
- minden további fokszám esetén egy véletlen $n \in p$ szám.

A 3.1. ábra szemlélteti a faelemekhez tartozó polinomok szülőktől való öröklését. A $q_x(0)$ polinom értéke minden esetben megegyezik a $q_{parent(x)}(index(x))$ polinom értékével.



3.1. ábra. Fa elemeinek $q_x(index)$ értéke.

A $q_{polynom}(index)$ függény *polynom* értékét az *index* paraméterrel a következőképpen számolja ki, ahol *degrees* a fokszám, *coeff*(d) pedig a d fokszámú tényezőhöz tartozó együttható: $\sum_{d=0}^{degrees} coeff(d) \cdot index^d$

Algoritmus 2 ComputeTree

procedure COMPUTETREE(T, s, $PublicKey$)
  if not $leaf(T)$ **then**
    $d = threshold(T) - 1$ ▷ d a polinom fokszáma
    $polynom = Polynom(d, s, PublicKey)$
    for all $y \in children(T)$ **do** ▷ T gyerekei
      $sum = q_{polynom}(index(y))$
      $ComputeTree(y, sum)$
    end for
  else
    $T_C = P^{q_T(0)}$
    $T_{C'} = H(attribute(y))^{q_T(0)}$
  end if
end procedure

Algoritmus 3 Encrypt

procedure ENCRYPT($PublicKey$, M, T)
  $s = randomInteger(p)$ ▷ $s \in \mathbb{Z}_p$
  $\widetilde{C} = M \cdot e(P,P)^{\alpha s}$
  $C = h^s$
  $ComputeTree(T, randomInteger(p), PublicKey)$
  $CT = (T, \widetilde{C}, C, \forall y \in leaves(T) : y_C, y_{C'})$
  return CT
end procedure

KeyGen. Két bemenetből, a mesterkulcsból ($MasterKey$) és egy attribútumlistából (S) generál egy privát kulcsot, mely a titkosított adat visszafejtésére lesz használható. Kimenete a privát kulcs (SK).

Algoritmus 4 KeyGen

procedure KEYGEN($MasterKey$, S)
  $r = randomInteger(p)$ ▷ $r \in \mathbb{Z}_p$
  $D = (1/\beta)(rP + \alpha P)$
  for all $j \in S$ **do**
    $r_j = randomInteger(p)$ ▷ $r_j \in \mathbb{Z}_p$
    $D_j = rP \cdot H(j) \cdot r_j$
    $D'_j = g \cdot r_j$
  end for
  $SK = (D, \forall j \in S : D_j, D'_j)$
  return SK
end procedure

Delegate. Az S attribútumlistával generált SK privát kulcs és egy $\widetilde{S} \subseteq S$ attribútumlista bemenetet felhasználva generál egy új privát kulcsot. A kimenet az $\widetilde{SK}$ új privát kulcs, melyhez az $\widetilde{S}$ attribútumlista társul.

Algoritmus 5 Delegate

procedure DELEGATE(SK, $\widetilde{S}$)
  $\tilde{r} = randomInteger(p)$ ▷ $\tilde{r} \in \mathbb{Z}_p$
  $\widetilde{D} = D \cdot f \cdot \tilde{r}$
  for all $k \in \widetilde{S}$ **do**
    $\tilde{r}_k = randomInteger(p)$ ▷ $\tilde{r}_k \in \mathbb{Z}_p$
    $\widetilde{D}_k = D_k \cdot \tilde{r} P \cdot H(k)^{\tilde{r}_k}$
    $\widetilde{D}'_k = {D'}_k \cdot \tilde{r}_k \cdot P$
  end for
  $SK = (D, \forall j \in S : D_j, {D'}_j)$
  return SK
end procedure

Decrypt. A CT titkos adat és az SK privát kulcs bemenetből meghatározza M adat értékét.

A *Decrypt* eljárás része az Algoritmus 6 által ismertetett *DecryptNode* metódus, bemenete a CT titkos adat, az SK privát kulcs és az x faelem.

A pszeudokódbeli *LagrangeCoefficient*(i, S, x) a $\triangle_{i,S}(x) = \prod_{j \in S}^{j \neq i} \frac{x-i}{i-j}$ képlettel felírható Lagrange-együtthatót adja meg.

Algoritmus 6 DecryptNode

procedure DECRYPTNODE(CT, SK, x)
  if *leaf*(x) **then**
    $i = attribute(x)$
    if $i \in S$ **then**
      return $e(D_i, x_C)/e(D'_i, x_{C'})$
    else
      return *false*
    end if
  else
    $A = 1, S_x = empty\ array$
    for all $z \in children(x)$ **do** ▷ x gyerekei
      $F_z = DecryptNode(CT, SK, z)$
      if F_z **then**
        $S_x.insert(index(z))$ ▷ beszúrás az S_x tömbbe
      end if
    end for
    for all $z \in children(x)$ **do** ▷ x gyerekei
      if F_z **then**
        $A = A \cdot F_z^{LagrangeCoefficient(index(z), S_x, 0)}$
      end if
    end for
    return A
  end if
end procedure

A *satisfy*(T, *attributes*) függvény visszatérési értéke akkor igaz, ha az *attributes* attribútumlista kielégíti a T hozzáférési fát.

A *DecryptNode* metódusnak a hozzáférési fa gyökérelemét paraméterül adva kiszámolható az $A = e(P, P)^{rs}$ érték. Az $e(C, D)$ érték a párosítás alapvető azonosságait figyelembe véve felírható $e(P, P)^{s(\alpha+r)}$ alakban. Előbbi azonosságot felhasználva $e(C, D)/A = e(P, P)^{\alpha s}$, amivel ha elosztjuk a $\widetilde{C} = M \cdot e(P, P)^{\alpha s}$ értéket, megkapjuk M-et.

Algoritmus 7 Decrypt

procedure DECRYPT(CT, SK)
  if not *satisfy*(T, *attributes*) **then**
    return error
  end if
  $A = DecryptNode(CT, SK, T)$
  $M = \widetilde{C}/(e(C, D)/A)$
  return M
end procedure

3.2.2. Az ABE és IBE összevetése

A CryptID programkönyvtár tartalmaz Identity-Based Encryption személyre szabott titkosítást nyújtó eljárást is. Emiatt a két titkosító eljárás közötti hasonlóságok és különbségek bemutatását fontosnak éreztük.

Az Identity-Based Encryption lényege a publikus kulcsot képező, entitást egyértelműen azonosító karaktersorozat. A különbség tehát leegyszerűsítve az, hogy míg az IBE esetén az entitás rendelkezik egy egyedi azonosítóval (például egyedi név), addig az ABE esetén egy-egy entitás egy teljes attribútumlistával rendelkezhet, és ez akár az egyedi azonosítóját is magában foglalhatja, de lehetőséget ad jóval bonyolultabb elérési szabályokra is (az attribútumok közti ÉS/VAGY operátorok használatával). Míg IBE esetében $1 : 1$, addig ABE esetében $1 : N$ titkosításról beszélhetünk, hiszen egy rejtjelezéssel több címzett számára is titkosíthatjuk az adatot. Az ABE emellett attribútumok használatával anonim titkosítást is lehetővé tehet (Wang, Xu, Shi & Lin, 2014), ami egyértelmű előnye az IBE-vel szemben.

4. fejezet

WebAssembly System Interface

A CryptID kriptográfiai programkönyvtár alapvető jellemzője a platformfüggetlenség. A dolgozatunkban készített ABE implementáció nemcsak kihasználja a CryptID és a WebAssembly által eddig is nyújtott platformfüggetlenséget, hanem a WebAssembly System Interface segítségével egy új szintre emeli azt. Ezt megalapozandó, ebben a fejezetben először rövid betekintést adunk a WebAssembly System Interface (továbbiakban WASI) alapjául szolgáló WebAssemblybe, majd ezt követően kitérünk a WASI-ra, és példákon keresztül szemléltetjük annak gyakorlati alkalmazását.

4.1. A WebAssemblyről röviden

A következőkben (Mills, 2017) alapján fogjuk ismertetni a WebAssembly főbb jellemzőit, különös figyelmet fordítva a platformfüggetlenségre és annak hiányosságaira.

A WebAssembly a W3C WebAssembly Community Group által fejlesztett kompakt, bináris formátummal rendelkező, alacsony szintű nyelv. A felhasználók 90%-a használ olyan böngészőprogramot, mely rendelkezik WebAssembly támogatással (*Can I use WebAssembly?*, 2020). A WebAssembly legfontosabb jellemzői a következők:

Gyors, hatékony, hordozható. A WebAssembly kód végrehajtási sebessége közel akkora, mint a natív kódé. Ezen felül hardver- és platformfüggetlen, csupán egy megfelelő környezet szükséges a futtatáshoz, akár mobil és IoT eszközökön is.

Jól olvasható és hibakereshető. A WebAssembly egy alacsony szintű assembly nyelv, azonban nem csak bináris reprezentációval rendelkezik. Létezik hozzá egy ember számára is fogyasztható szöveges formátum. Ez egy olyan assembly-jellegű nyelv, amely lehetővé teszi a kód írását, vizsgálatát és hibakeresését. E formátum jelenleg is fejlesztés alatt áll, még nem végleges.

Biztonságos. Futása egy zárt, úgynevezett *sandbox* környezetben történik. Ennek egyik előnye, hogy a WebAssembly modulok csak a megfelelő API hívásokkal tudnak kilépni a környezetükből, ezzel megelőzve az esetleges rosszindulatú viselkedést. Másik előnye, hogy ebben a környezetben, néhány kivételtől eltekintve, az alkalmazások futása determinisztikus.

Illeszkedik a Webbe. A WebAssembly kialakításának már a kezdetektől is fontos szempontja volt, hogy együtt tudjon működni a Weben már jelen lévő technológiákkal, melynek egyik alapvető hozzávalója a visszafelé kompatibilitás.

Számos alkalmazási területen jelent komoly előrelépést a böngészőben, a korábban felsorolt tulajdonságoknak köszönhetően. Elsősorban az erőforrás-igényes alkalmazások számára nyit eddig nem látott lehetőségeket. Ezen alkalmazási területek, a teljesség igénye nélkül (*WebAssembly – Use Cases*, 2020) :

- kép- és videószerkesztés,
- nagy sebességű alkalmi és magas erőforrásigényű felső kategóriás játékok,
- zenei és videós streaming szolgáltatások,
- virtuális és kiterjesztett valóság,
- titkosítás, virtuális magánhálózatok, távoli asztali elérés.

A WebAssembly egy olyan célplatform, mely gyors, hatékony, zárt környezetben való kódvégrehajtást biztosít.

Egyik fontos tulajdonsága e célplatformnak a nyelvfüggetlenség, azaz számos programnyelvből lehetséges WebAssembly kódra fordítani. A már szinte a kezdetektől támogatott C/C++ és a Rust mellett már a C#, a Go és a TypeScript is stabil támogatással rendelkezik, de más népszerű nyelveknél is folyamatban van annak kiépítése, mint például a Java esetében (*Awesome WebAssembly Languages*, 2020).

Annak ellenére, hogy a platform megoldást jelentett egy böngészők által felvetett problémára előnyös tulajdonságainak köszönhetően, a szabvány nem kizárólag a Webes használatot szem előtt tartva került kialakításra. Megfelelő beágyazó környezetek segítségével bárhol lehetséges futtatni a WebAssembly kódokat.

A WebAssembly binárisokat moduloknak nevezzük. A böngésző és a beágyazó környezetek is ezeket töltik be, és hajtják végre. A legegyszerűbb módja egy ilyen modul létrehozásának az *Emscripten* használata.

Az *Emscripten* egyszerű használhatóságának bemutatására vegyünk alapul egy OpenGL grafikus alkalmazást. Az *Emscripten* ezen alkalmazás kódjából elkészíti a megfelelő WebAssembly modult, sőt a böngészőbe való beágyazáshoz szükséges JavaScript és HTML kódot is biztosítja (*About Emscripten*, 2015). Mindezt csupán egyetlen parancs segítségével el is tudjuk érni. Feltéve, hogy a C nyelven írt példa alkalmazásunk neve `demo.c`, a parancs a következő:

```
emcc demo.c -o demo.html
```

4.1. Kódrészlet. Az *Emscripten* használata.

Felmerülhet bennünk a kérdés az imént említett egyszerű grafikus alkalmazás kapcsán, hogy az *Emscripten* hogyan képes például a grafikára. Az *Emscripten* kiegészíti a kapott kódot egy olyan, futtatókörnyezethez hasonló egyedi környezettel, melyben minden megtalálható, amire a kódnak szüksége van. Ez lehet olyan egyszerű, mint például egy *malloc* implementáció, de akár olyan összetett is, mint egy virtuális fájlrendszer. Vagy akár az esetünkben látott OpenGL megvalósítása WebGL segítségével (*Compiling WebAssembly with Emscripten*, 2015).

4.2. WASI

A WebAssembly böngészőn kívüli futtatására korábban is léteztek megoldások, ám ezek távolról sem voltak kiforrottnak, stabilnak nevezhetők. Az *Emscripten* hiába volt működőképes, az elkészült modulok inkább csak a böngészőben voltak használhatók. Bár általa az elvi megoldás létezett a felmerült problémára, a standardizáció hiánya mégis meggátolta a gyakorlati alkalmazást. A WebAssembly System Interface ezt az űrt hivatott betölteni. A WASI egy rendszer interfész egy elméleti operációs rendszerre a WebAssembly számára.

4.2.1. Mi az a rendszer interfész?

A rendszer interfész biztosítja a programok számára a számítógép erőforrásainak elérését. Már az informatika korai szakaszában ismert volt a *kernel* fogalma, mely egyfajta védőburokként szolgál a számítógép erőforrásai köré (*Kernel Space Definition*, 2005). Az adott programok az elérni kívánt fájlokat, memóriát vagy hálózati kapcsolatokat rendszerhívások segítségével érik el. Így elkerülve az olyan eseteket, hogy egy adott program a működése során úgy módosítson egy fájlt, hogy az más program számára már használhatatlan legyen, ezzel esetleges összeomlást, vagy adatszivárgást eredményezve. Ugyanis

amikor egy felhasználó olyan programot futtat, ami rendszerhívást intéz, akkor a *kernel* a felhasználó jogosultságainak figyelembevételével engedélyezi vagy utasítja el az adott kérést.

Ezeket a rendszerhívásokat az operációs rendszer teszi lehetővé. Azt a problémát, hogy minden operációs rendszer saját rendszerhívásokkal rendelkezik az absztrakció oldja meg. A nyelvek többsége egy standard könyvtárat biztosít a fejlesztők számára. Ezt az interfészt kiaknázva csak a kódot magát kell megírni, az adott operációs rendszerhez alkalmas rendszerhívás kiválasztása már a választott nyelvhez tartozó eszközkészlet feladata. Ezek nyilvánvalóan rendszerspecifikusak lesznek, hiszen az adott rendszer API hívásait alkalmazzák (Clark, 2019).

4.2.2. A WASI céljai

Általános esetben ezt jelenti a rendszer interfész. Azonban a WebAssembly esetében ez nem ilyen egyszerű. Egyik fő elv volt, hogy hordozható legyen. Ennek következtében az adott program fordításakor nem fogjuk tudni, hogy mi a cél operációs rendszer, így nyilván nem is lehet alkalmazni egy nem ismert rendszer hívásait. Ezért szükséges egy olyan interfész kialakítása a WebAssembly számára, amely egy elméleti operációs rendszert céloz meg. Hasonlóan, ahogy a WebAssembly kialakításánál a cél egy elméleti architektúra volt.

Az interfész megalkotásakor két – korábban a WebAssembly kialakításakor is szem előtt tartott – elvet fogalmaztak meg (Clark, 2019).

Az egyik cél a **hordozhatóság**. A korábbi probléma az volt, amit az *Emscripten* esetében is láthattunk, hogy minden fordító vagy beágyazó környezet egyedi megoldásokkal próbálkozott, így ami az egyikben működött, az a másikban már nem, lehetetlenné téve a modulok hordozását a különböző környezetek között. A cél az volt, hogy egy egységes futtatókörnyezet alapjait teremtsék meg. Ezzel a fejlesztők rengeteg konfigurációs beállítást megtakaríthatnak, felgyorsítva ezzel a fejlesztés folyamatát.

A másik cél pedig a **biztonság.** Egy operációs rendszerben, a már korábban is említett eljárás az, hogy a programot futtató felhasználó jogkörével rendelkezik a program is, legyen ez tulajdonosi vagy csoport jogosultság. Ez azonban egy olyan környezetben működik jól, ahol több felhasználó van jelen. Egy olyan rendszerben, ahol viszont csak egy felhasználó van jelen – manapság ez a leggyakoribb – ott az esetleges probléma a harmadik féltől származó programok esetében jelentkezhet, a megbízhatatlanság miatt. Így a legnagyobb ellenségünk a saját programunk lehet. Ezért a WASI modulok egy zárt, *sandbox* környezetben futnak. Azt, hogy az adott program milyen erőforrásokat képes elérni, a felhasználó dönti el. Ez persze nem nyújt teljes körű biztonságot, de adott a lehetőség annak megteremtésére. A WASI által biztosított izoláltság azonban többet jelent, mint korábban a WebAssembly esetében. Erre a következő alfejezetben láthatunk majd

példát.

4.2.3. A WASI megvalósítása

Ebben az alfejezetben (Gohman, 2019) alapján fogjuk ismertetni a WASI alapjait. Az interfész a kialakítását tekintve moduláris. A standardizálási folyamat részeként bizonyos funkciók külön-külön modulként lesznek hozzáadva az interfészhez. Az első és legfontosabb modul a *wasi-core*, mely alapvető funkcionalitásokat tartalmaz, egy POSIX-hoz hasonló alapot, mint a fájlkezelés, a hálózati kapcsolatok vagy a véletlen számok (*WASI – wasi-core*, 2020).

Ez további hordozhatóságbeli lehetőségeket rejt magában. Minden futtató környezet saját maga implementálhatja a *wasi-core*-t, a saját igényeire szabva. Ráadásul a biztonság is teljesül, ugyanis a környezet döntheti el, hogy milyen *wasi-core* függvényeket, ezek által pedig milyen rendszerhívásokat engedélyez a benne futó program számára.

Ezen felül további, a biztonságot növelő eszközök állnak rendelkezésünkre a WASI-nak köszönhetően. Egy olyan függvényhívásnál, mely egy fájlt hivatott elérni, meg kell adni egy fájlleírót, amihez jogosultságok tartoznak. Ezek vonatkozhatnak fájlokra, de akár az őket tartalmazó mappákra is, csak a megadott mappákhoz biztosítva hozzáférést. Elég csak a felső szintű modulnak biztosítani a fájlleírókat, és azok szükség alapján osztják tovább más modulokban használt függvényeknek. Így a modulok pontosan annyi erőforráshoz kapnak hozzáférést, amennyire a működéshez szükségük van.

Összefoglalásképpen elmondható, hogy a WASI egy olyan API, mely egy elméleti operációs rendszer funkcióit hivatott biztosítani a WebAssembly számára, függetlenül bármilyen böngészőtől, de még a korai böngészőn kívüli megoldásokban jelen lévő JavaScripttől is.

4.3. A WASI működés közben

A fordításhoz használt eszközt az LLVM fordító infrastruktúra biztosítja (*Getting started with the LLVM System*, 2020). Az LLVM egy olyan eszköz, amely egy *Intermediate Representationt* (*IR*) definiál. Ez az *IR* választja el a *front-* és a *backendeket*. Számos nyelvhez létezik LLVM *frontend*, ami *IR*-t generál a nyelv forráskódjából, például ilyen a C nyelv. Ebből a *bitcode*-ból pedig a megfelelő *backend* segítségével hoz létre például WASM modulokat. Az LLVM a 8.0 verzió kiadása óta stabil WebAssembly támogatással rendelkezik (*Changes to the WebAssembly Target*, 2019). Az általunk használt verzió a *10.0.0*. Ezen infrastruktúra számos eszköze közül, egy C/C++ nyelven íródott kód WebAssembly binárissá fordításához, a *clang* eszközre van szükségünk.

Azonban az LLVM WebAssembly támogatása ellenére nem tartalmazza a legtöbb C/C++ nyelvű kód fordításához szükséges standard C könyvtárat, a *libc*-t. Ezért a C és C++ számára elkészült a *wasi-sysroot/wasi-libc*, mely a standard C könyvtár egy implementációja (*WASI libc implementation for WebAssembly*, 2019). Ezen implementáció különlegessége, hogy teljes egészében a *wasi-core* függvényekre épül. Az LLVM mellett tehát még erre van szükségünk. A modulok a megfelelő *wasi-core* függvényeket a futtató környezet segítségével importálva kapják meg. Az importot követően a modulon belül felhasználhatók lesznek az adott függvények (*WebAssembly – Imports*, 2016).

Így e két eszköz segítségével rendelkezésünkre áll egy olyan fordítási környezet, mely WASI-kompatibilis WebAssembly binárist képes előállítani. Az elkészült modul futtatásához egy WASI futtató környezetre van szükségünk. Ilyen környezet például a *wasmtime* (*Standalone JIT-style runtime for WebAssembly*, 2019).

A következőkben bemutatjuk a fordítási és futtatási folyamatot. Először vegyünk egy C nyelven írt, egyszerű kódot:

```
1   #include <stdio.h>
2
3   int add (int first, int second)
4   {
5           return first + second;
6   }
7
8   int main(int argc, char **argv)
9   {
10          printf("Result: %d\n", add(1,2));
11
12          return 0;
13  }
```

4.2. Kódrészlet. A fordítandó `demo.c` fájl.

A 4.2 kódrészletben két szám összeadását végző függvény megadása után, annyi történik, hogy két konstans egész érték – jelen esetben az 1 és a 2 – összegét kiírjuk a standard kimenetre egy rövid szöveg kíséretében. Tehát a várt eredmény a *„Result: 3”*. Fontos jellemzője a kódnak az, hogy nincs egyetlen olyan részlete sem, amely arra utalna, hogy ezt később WebAssembly binárisra szeretnénk fordítani.

A fordítást a korábban említett LLVM infrastruktúra *clang* eszközével végezzük, mégpedig a következő módon:

```
clang --target=wasm32-unknown-wasi \
    --sysroot <wasi-sysroot-elérési-útvonala> \
```

```
demo.c -o demo.wasm
```

4.3. Kódrészlet. A `demo.c` fordítása *clanggel*.

A 4.3 kódrészlet parancsában szereplő első kapcsoló a `target`, mely a fordítási célunkat adja meg a fordító számára. Ez esetünkben a **wasm32-unknown-wasi** ami egy *target triplet*. Egy ilyen *triplet* egy célplatformot ad meg a fordító számára (*General Cross-Compilation Options in Clang*, 2014). A hármas első tagja, a **wasm32** azt adja meg, hogy az általunk használni kívánt architektúra a 32 bites pointereket alkalmazó WebAssembly. A második paraméter a gyártót jelöli, azonban esetünkben ez nem releváns, ezért az **unknown** értéket használjuk. Az utolsó paraméter pedig a választott operációs rendszert adja meg. Ez esetünkben a **wasi**, ami megegyezik azzal a korábbi megállapításunkkal, hogy a WASI célja egy elméleti operációs rendszer a WebAssembly platform számára. A második kapcsoló a `sysroot`. Ez az előre telepített, és korábban taglalt *wasi-sysroot* elérési útvonala. Amennyiben ezt a tároló gyökerében helyeztük el, úgy e kapcsoló alkalmazása nem szükséges, a *clang* automatikusan ezt a könyvtárat fogja alkalmazni a fordításkor. Az eredmény a parancsban is meghatározott `demo.wasm`, mely a kívánt WebAssembly binárist tartalmazza.

Az elkészült binárist futtathatjuk tetszőleges WASI-kompatibilis beágyazó környezetben, például a korábban említett *wasmtime*-ban:

```
wasmtime demo.wasm
```

4.4. Kódrészlet. A `demo.wasm` futtatása *wasmtime* segítségével.

A 4.4 kódrészletben látható parancs kiadása után megjelenik a konzolon a futtatás eredménye, vagyis *„Result: 3”*, ami megegyezik korábban felállított elvárásunkkal.

5. fejezet

Implementáció

A korábbi fejezetekben ismertettük elkészült és működő implementációnk matematikai, kriptográfiai és platformfüggetlenséget illető alapjait. Ennek a fejezetnek a célja bemutatni az elkészült munkánkat.

Az Attribute-based Encryption (ABE) implementálását célszerűnek láttuk egy teljesen új kriptográfiai könyvtár megalkotása helyett egy meglévőbe beépíteni. Ahogy a bevezetőben is említettük, a választás a CryptID (Vécsi, Bagossy & Pethő, 2019) platformfüggetlen könyvtárra esett, mely már tartalmazott egy *Identity-based Encryption* (IBE) és *Identity-based Signature* (IBS) implementációt. Az egyik oka ennek az volt, hogy így egy ismeretlen könyvtár elkészítése helyett egy már használatban lévőt bővíthettünk. Egy másik ok az volt, hogy már kialakult előírásokkal, elvekkel rendelkező könyvtárba beépítve a megfelelő minőségű kódot készíthettük el, illeszkedve annak struktúrájához és minden elvárásához.

5.1. Platformfüggetlenség

A CryptID korábbi implementációja is a WebAssemblyre épült, azonban akkor még nem állt rendelkezésre egy olyan, a böngészőn kívüli futtatást lehetővé tevő ajánlás, mint a WASI. Az eredeti implementáció is kínált egy bizonyos szintű platformfüggetlenséget; működőképes volt bármely böngészővel rendelkező eszközön, legyen az Google Chrome, Mozilla Firefox vagy Microsoft Edge. A korábbi implementáció felépítése szerint a könyvtár egy C nyelven írt alappal rendelkezett, efölött pedig egy interoperabilitási réteg közbeékelésével helyezkedett el a *CryptID.js*. Ennek nem csupán az előbb említett böngésző-alapú platformfüggetlenség volt az előnye, hanem a *Node.js*-ben való alkalmazhatóság is. Azonban ez még mindig nem nyújtott teljes körű platformfüggetlenséget. Például az eredeti cél, hogy a könyvtár működése IoT eszközökön is biztosítva legyen, ez által nem valósult meg kielégítően.

Ez motivált minket arra, hogy a könyvtár platformfüggetlenségét tovább bővítsük a WASI segítségével, annak stabil alapot biztosítva. Ugyanis egy WASI kompatibilis modul platformfüggetlensége sokkal kiterjedtebb, mint a korábbi böngésző- és *Node.js*-alapú függetlenség. Ennek köszönhetően bárhol, ahol létezik WASI beágyazó környezet, lehetséges a modulok futtatása, hiszen már számos WASI-t implementáló beágyazó környezet létezik, az eszközök széles skáláján biztosítva a WASI előnyeit. Ilyen környezetet biztosít például a WAMR vagy a WASM3. E környezeteken keresztül olyan eszközök is tudnak élni a WebAssembly adta lehetőségekkel, mint a mikrokontrollerek és bizonyos routerek is (*WASM3 – Supported platforms*, 2020), vagy olyan architektúrák, mint a MIPS vagy az XTENSA (*WAMR – Supported architectures and platforms*, 2020).

Egy kriptográfiai könyvtár esetében a nagy egész számok kezelésének képessége alapvető. A korábbi implementáció erre a célra a GMP (GNU Multiple Precision Arithmetic Library) programkönyvtárat használta (*Introduction to GMP*, 2008). A célunk az volt, hogy egy WASI-ban is hatékony működésre képes, nagy számokat kezelő könyvtárat helyezzünk a CryptID alá, anélkül, hogy bármilyen változtatást kellene végrehajtani a CryptID felépítésén, ide értve a struktúrákat és a függvény szignatúrákat.

E probléma kiküszöbölésére négy lehetőség állt rendelkezésünkre:

Új, nagy számokat kezelő könyvtár implementációja. Egy ilyen könyvtár elkészítése hatalmas feladat, így ez számunkra nem volt megfelelő megoldás.

A GMP helyett más könyvtárat alkalmazunk. Ilyen lehetőség az MPIR, vagyis a Multiple Precision Integers and Rationals (B. Gladman & J. Moxham, 2015). Azonban e könyvtár sem a WASI-t célozza meg, így a GMP lecserélésével nem jutnánk semmilyen előnyhöz.

A MiniGMP-t használjuk. A MiniGMP a GMP egy olyan részimplementációja, mely csak az egész számokat kezeli. Azonban, ahogy azt a dokumentációja is említi, nem alkalmas nagyon nagy számok kezelésére és teljesítményben is alulmarad, akár 10-szer lassabb működést eredményezve, a normál GMP-hez képest (Möller, 2013). Ez pedig egy kriptográfiai programkönyvtár számára, ahol a teljesítmény sem utolsó szempont, kifejezetten hátrányos.

Megtartjuk, de módosítjuk a GMP-t. A GMP megtartása önmagában nem elegendő, mert számos olyan funkcionalitással rendelkezik, amelyre nincs szükség az implementációnkban.

Így a választásunk a GMP megtartására esett. Adott volt tehát a feladat, hogy azt olyanná alakítsuk át, hogy az nekünk megfelelő legyen.

5.1.1. A GMP átalakítása

A GMP számos olyan funkcionalitást tartalmazott, amely számunkra felesleges volt. Ilyenek voltak a formázott I/O rutinok, a véletlenszám-generáláshoz kapcsolódó függvények, valamint a lebegőpontos és a racionális számokat kezelő függvények. Ezeket eltávolítottuk a könyvtárból, ügyelve arra, hogy ezen elemek eltávolítása ne változtasson a könyvtár számunkra fontos működésén.

A GMP könyvtár alapértelmezésben számos architektúrát és platformot támogat. Számunkra azonban csak két architektúra releváns. Az egyik az x86-64, melyre a fejlesztés szempontjából van szükség, a másik pedig természetesen a WASM. Ezek az architektúra- és platform-specifikus betétek azt a célt szolgálták, hogy a GMP könyvtárat a célnak megfelelően, minél jobban konfigurálni és optimalizálni lehessen. Azonban az imént említett két architektúra támogatásán kívül másra nem volt szükségünk, így azokat a betéteket eltávolítottuk a könyvtárból.

Az említett, assmebly nyelven írt optimalizáló betétek jelenléte megnehezítette a GMP könyvtár buildelését WebAssemblyben való alkalmazhatóság érdekében. Ahhoz, hogy a fordítási folyamat gördülékenyen menjen, csupán C kódokra lett volna szükségünk, mert a korábban említett platformspecifikus assembly betétek WebAssemblyre történő fordítása nem megoldható. Annak ellenére, hogy a MiniGMP pont egy ilyen tulajdonságokkal rendelkező GMP változat, mégsem ezt alkalmaztuk, a már fentebb is említett teljesítménybeli elmaradottsága miatt. Fontos kiemelni egyúttal, hogy a könyvtárat egyszerűsítés nélkül is van lehetőség WebAssemblyben alkalmazni. A korábbi fejezetben említett *Emscripten* segítségével kivitelezhető, de általában Webes felhasználási céllal, modulok és hozzátartozó JavaScript kódok előállítására (Zakai, 2013).

A mi céljaink azonban ennel ambiciózusabbak voltak:

- egyfelől, hogy a könyvtárból lehessen önálló, importálható WASI modult előállítani,
- másfelől, hogy a kódot bármely, WASI-t célzó szoftver módosítások nélkül beemelhesse, mint függőséget.

Az eredeti könyvtár működése a gyakorlatban a következő módon történik. A platform- és architektúra-specifikus betétekkel való konfigurációt követően lehet buildelni a könyvtárat. Ekkor készül el egy, az adott platformra specializált `gmp.h` fejlécállomány. Ahhoz, hogy ez számunkra megfelelően konfigurált és WASI-kompatibilis legyen, eltávolítottuk a már említett betétek mellett a GMP általunk nem használt függvényeit, különös tekintettel az olyanokat, melyek rendszerhívásokra épülnek. Csupán olyan elemeit hagytuk meg a könyvtárnak, melyek a nagy egész számok kezelését végzik, vagy a memóriaműveletekért felelnek. Utóbbiak esetében lehetőség adódik arra, hogy saját kódot alkalmazzunk. A

rendszerhívással kapcsolatos fel nem használt függvények eltávolítására azért volt szükség, hogy jól körül tudjuk határolni, hogy a WASI mely funkcionalitásait vesszük igénybe.

A törlések következménye volt, hogy a könyvtár eredeti build szkriptjét már nem tudtuk alkalmazni. Ezért saját szkripttel biztosítottuk azt, hogy az elkészülő `gmp.h` fejlécállomány az igényeinkre szabott legyen és a WASI-ra való fordítás triviális legyen. A szkript teljes mértékben testreszabható, ugyanis paraméterként lehet megadni a C fordítót, a szükséges kapcsolókkal együtt. Ebbe beletartozik a `sysroot` kapcsoló is, aminek értéke esetünkben a *wasi-libc/sysroot* elérési útvonala, ezzel biztosítva a WASI-kompatibilis standard C könyvtár használatát. Utolsó paraméterként az archívum-készítőt kell megadni (például: *llvm-ar*), amivel egy statikusan linkelhető könyvtárat is létrehozunk a `gmp.h` mellett.

Így elkészült a CryptID WASI implementációjához szükséges saját GMP forkunk, mely a *WebAssembly Arbitrary Precision Maths*, azaz röviden a *WARM* nevet kapta (*WARM*, 2020).

A *WARM* egy olyan, kizárólag C nyelven írt könyvtár, amely nagy egész számok kezelésére alkalmas. Ebből több előny is származik. Egyfelől, csak a számunkra releváns részeit tartalmazza az eredeti GMP könyvtárnak, így nem kell a könyvtár számos célplatformjával és egyéb függvényeivel foglalkoznunk. Másfelől, könnyen tudjuk a szükséges WebAssembly-specifikus optimalizációkat végrehajtani. Erre abban az esetben lehet szükség, amennyiben a későbbiek folyamán a WebAssembly jobban fogja támogatni a kriptográfiai alkalmazásokat.

Annak érdekében, hogy a könyvtár esetleges bővítése esetén is gyorsan lehessen ellenőrizni annak megfelelő buildelését, elkészítettünk egy CI (*Continous Integration*) *pipeline*-t, a GitHub Actions[1] segítségével. A *WARM* könyvtár függőségei az LLVM infrastruktúra és a *wasi-libc*. A *pipeline* ezek telepítése után automatikusan buildeli a könyvtárat. Így az egyes változások után mindig láthatjuk, hogy valóban sikeresen buildelhető maradt-e a kód.

A *WARM* tehát úgy lett kialakítva, hogy az eredeti CryptID implementációban használt GMP könyvtár helyett legyen alkalmazható (azaz a CryptID szempontjából *drop-in replacementnek* tekinthető). Ennek segítségével már zökkenőmentesen állítható elő a CryptID WASI-kompatibilis verziója. E célból is készült egy *pipeline*, ami a megfelelő függőségek telepítését követően egy WASI-kompatibilis WebAssembly binárist fordít a CryptID-ből. Ezáltal bármilyen bevezetett újdonságot követően, az automatikus futásnak köszönhetően rövid időn belül ellenőrizhetjük a módosítások WASI-kompatibilitását.

[1] `https://github.com/features/actions`

5.2. Attribute-based Encryption

Az ABE megvalósítása során számos kriptográfiai és aritmetikai implementációt nyújtott a CryptID, amelyek egy megfelelő alapot biztosítottak számunkra. Az ABE implementációjának bemutatása során ismertetjük a felhasznált műveletek részleteit is.

5.2.1. Megvalósítási részletek

A 3. fejezetünkben bemutattuk az ABE öt alapvető eljárását, melyeket azoknak megfelelően implementáltunk.

A `setup` függvény esetében a biztonsági szintet meghatározó k bemeneti paraméter közvetetten a ciklikus csoportok méretét adja meg. Ugyanebben a függvényben kerül meghatározásra a biztonsági szintnek megfelelő $H(x)$ kriptográfiai *hash* függvény is. Ezekhez a kriptográfiai könyvtárban már használt, RFC 5091-ben részletezett (2007), p bithosszúságára ajánlásokat tartalmazó táblázatot és a $H(x)$ *hash* függvényre vonatkozó javaslatokat használtuk fel. Az 5.1. táblázatban ismertetjük a kriptográfiai könyvtár erre használt jelöléseit.

Jelölése a könyvtárban	p **bithosszúsága**	**Hash függvény**
LOWEST	512	*SHA1*
LOW	1024	*SHA224*
MEDIUM	1536	*SHA256*
HIGH	3840	*SHA384*
HIGHEST	7680	*SHA512*

5.1. táblázat. Az RFC 5091 javaslata p bithosszára és a kriptográfiai *hash* függvényre.

A $H(x)$ függvény 5.1 táblázatban meghatározott, biztonsági szintnek megfelelő változatát a CryptID `hashToPoint` függvényén alkalmazva egy $\{0,1\}* \rightarrow G_0$ leképezést kapunk, ami így használható a `ComputeTree`, `keygen` és `delegate` eljárásokban, többek között az Algoritmus 2 által ismertetett módon. A használt *hash* függvények biztonságát az adja, hogy a CryptID-be való implementálás a megbízhatónak ítélhető RFC 6234 ajánlásai alapján készült (Hansen & Eastlake, 2011).

A könyvtár az RFC 5091-ben leírt *Type-1*, a szuperszinguláris görbék egy részcsoportját képező elliptikus görbéket használja. Ezek alakja a következő:

$$p \equiv 11 \pmod{12} \textit{ prím, } E(\mathbb{F}_p) : y^2 = x^3 + 1.$$

Az $E(\mathbb{F}_p)$ görbe pontjainak száma $p + 1$, melyek esetében igaz, hogy

$$\forall (x, y) \in E(\mathbb{F}_p) \backslash \{0\} \textit{ esetén } x = (y^2 - 1)^{(1/3)} \pmod{p}.$$

Köszönhetően annak, hogy Tate párosítás és az elliptikus görbe implementáció ajánlások alapján készült, biztonságosnak tekinthető. Emellett eredetileg is olyan célkitűzést követve lettek elkészítve, hogy maximálisan kielégítsék a platformfüggetlenség feltételét is, ezáltal kisméretű, csak az adott igényekhez tervezett implementációt jelentenek (Vécsi és mtsai., 2019). A könyvtár más részeihez hasonlóan a prímgenerálást $2^a \pm 2^b \pm 1$ alakú, NIST ajánlásaiban is szereplő (*Digital Signature Standard (DSS)*, 2013) Solinas prím generálásával oldjuk meg. Ezután a $p = 12 \cdot r \cdot q - 1$ alakú (Boyen & Martin, 2007), a táblázatnak megfelelő hosszúságú prím keresését hajtjuk végre.

Az `encrypt` eljárás az Algoritmus 3 pszeudokódban leírttal szemben módosításra szorult abból adódóan, hogy a titkosítandó szöveg számként való reprezentációja átlépheti a ciklikus csoport méretét. Az eljárásba beépítettük az üzenet több részre való feldarabolását, így tetszőleges hosszúságú adaton használható. A gyakorlatban n_i karakter hosszúságú részletekre kerül felbontásra a szöveg, ahol i a szövegrészlet indexe. n_i az a lehető legnagyobb szám, amivel a szövegrészlet számként való reprezentációja kisebb, mint a ciklikus csoport mérete. Ez szövegrészletenként egy $\widetilde{C}$ érték kiszámítását jelenti egy titkosított üzenetben. Ennek megfelelően a `decrypt` algoritmus a különböző $\widetilde{C}$ értékekkel külön kiszámolja az ahhoz tartozó M értékeket, majd szöveggé alakítva összefűzi őket.

A könyvtár igényeinek megfelelve implementációnk során szem előtt kellett tartanunk az API réteg egyszerű kezelhetőségét, ezáltal többek között a nyilvános struktúrákban csak elemi változótípusok használatát. Ez megkönnyíti az adat későbbi sorosítását. Elkészítettük a megfelelő egységteszteket, amik magukban foglalnak véletlenszerű és irányított tesztet is, továbbá sikeres és sikertelen visszafejtést is. Továbbá, teszteltük a könyvtár általunk fejlesztett részének esetleges memóriaszivárgásait, hogy megbizonyosodhassunk ezek hiányáról.

5.3. Az implementáció használata

Az alábbiakban bemutatjuk az implementáció egyszerű használatát, az áttekinthetőség kedvéért C nyelven írva. Ez a 4. fejezetben leírtak szerint könnyen fordítható WebAssemblyre, azonban natív módon a C kód is lefordítható és futtatható. A példakódból az olvashatóság kedvéért hibakezeléseket és memóriafelszabadításokat mellőztük. A sorszámok a függelékben látható 7.2 kódban elfoglalt helyet jelölik. Az olvashatóság elősegítése miatt a példakód elején a hosszú függvénynevek miatt különböző definíciókat adtunk hozzá, valamint mellőztük a beágyazások részletezését.

```
17 publicKey* publickey = malloc(sizeof(publicKey));
18 masterKey* masterkey = malloc(sizeof(masterKey));
19 CryptidStatus status = cryptid_abe_bsw_setup(publickey, masterkey, LOWEST);
```

Először a `setup` eljárás futtatására van szükség, mely létrehozza a `publickey` publikus kulcsot és a `masterkey` mesterkulcsot. Harmadik paramétere a 5.1 táblázatban ismertetett biztonsági szint, ami a példakód esetében **LOWEST**.

```
21 accessTree* accessTree = accessTree_init(2, NULL, 0, 2);
22
23 accessTree* child0 = accessTree_init(1, NULL, 0, 2);
24 accessTree->children[0] = child0;
25
26 char* attribute0 = "Programozó";
27 accessTree* child0_0 = accessTree_init(1, attribute0, strlen(attribute0), 0);
28 child0->children[0] = child0_0;
29
30 char* attribute1 = "Projektmenedzser";
31 accessTree* child0_1 = accessTree_init(1, attribute1, strlen(attribute1), 0);
32 child0->children[1] = child0_1;
33
34 char* attribute2 = "Szervezeti tag";
35 accessTree* child1 = accessTree_init(1, attribute2, strlen(attribute2), 0);
36 accessTree->children[1] = child1;
```

A következő lépés a megfelelő hozzáférési fa *(access tree)* létrehozása. A kódban látható hozzáférési fa a következően leírttal ekvivalens:

(*Programozó* **VAGY** *Projektmenedzser*) **ÉS** *Szervezeti tag.*

Az `accessTree_init(threshold, attribútum, attribútumHossz, gyerekSzám)` alakú függvény a megfelelő, publikus API-ban használható hozzáférési fát hozza létre. Ebből a `threshold` az ABE fejezetben kifejtett értéke a fának, mely **ÉS** fa esetén megegyezik a gyerekek számát megadó `gyerekSzám` paraméterrel, míg **VAGY** fa esetén 1. Az `attribútum` a levélelemek esetén annak attribútuma, míg az `attribútumHossz` ennek a hossza. Nem levélelemek esetén előbbi `NULL`, utóbbi 0. A kódban látható módon egy fa elemének a gyerekeit a `children` tömbben kell elhelyezni.

```
38 char* message = "Teszt üzenet.";
39 encryptedMessage* encrypted = malloc(sizeof(encryptedMessage));
40 status =
41 cryptid_abe_bsw_encrypt(encrypted, accessTree, message, strlen(message), publickey);
```

A megfelelő publikus kulccsal (`publickey`) és a titkosított üzenet esetében a visszafejtésre jogosultak körét meghatározó hozzáférési fával (`accessTree`) titkosítható a `message` üzenet az `encrypt` eljárásban, paraméterül adva még annak hosszát.

```
43 int numAttributes = 2;
44 char** attributes = malloc(sizeof(char*) * numAttributes);
```

```
45 attributes[0] = attribute0;
46 attributes[1] = attribute2;
47 secretKey* secretkey = malloc(sizeof (secretKey));
48 status = cryptid_abe_bsw_keygen(secretkey, masterkey, attributes, numAttributes);
```

A `keygen` függvény használatához először szükség van egy attribútumlista (*tömb*) létrehozására. Jelen esetben az `attributes` tömböt hozzuk létre a korábban definiált *Programozó* és *Szervezeti tag* attribútumokkal. Ez az attribútumlista egy generált példaentitás attribútumainak listáját jelenti, aki vissza szeretné fejteni az előző lépésben titkosított adatot. A függvényt a létrehozott attribútumlistával, az attribútumok számával, a mesterkulccsal és a létrehozandó titkos kulccsal meghívva egy olyan kulcsot hozunk létre, amely kielégíti a korábban létrehozott hozzáférési fát. Így használható lesz az azzal titkosított üzenetek visszafejtésére.

```
50 secretKey* secretkeyNew = malloc(sizeof (secretKey));
51 int numAttributes2 = 1;
52 char** attributes2 = malloc(sizeof(char*) * numAttributes2);
53 attributes2[0] = attribute2;
54 status =
55 cryptid_abe_bsw_delegate(secretkeyNew, secretkey, attributes2, numAttributes2);
```

Opcionálisan lehetőség van egy privát kulcsból egy teljesen új privát kulcsot alkotni, melynek attribútumlistája részhalmaza az eredetinek. A példakódban látható `delegate` eljárás a `secretkeyNew` új privát kulcsot hozza létre a `keygen` függvényhez hasonlóan, a mester kulcs helyett a privát kulcsot adva bemenetként.

```
57 char* result;
58 status = cryptid_abe_bsw_decrypt(&result, encrypted, secretkey);
59 printf("%s\n", result);
```

A `decrypt` függvényt a titkosított üzenettel (`encrypted`), valamint annak a visszafejtésére képes privát kulccsal meghívva megkapjuk az eredeti üzenetet (`result`).

5.4. Teljesítménytesztek

Munkánk különböző környezetekben megfigyelhető teljesítményének, használhatóságának bemutatására teszteket végeztünk. Ezáltal szeretnénk összehasonlítást adni a különböző környezeteken való futási teljesítményről.

5.4.1. Tesztkörnyezetek

A dolgozatban feltüntetett tesztkörnyezetek az alábbiakat foglalják magukban:

- Linux operációs rendszeren, felhő alapú virtuális gépen futtatott *wasmtime*, *Node.js* és C kódból fordított natív bináris. A környezet pontos adatait a 7.1 táblázat tartalmazza.
- Windows operációs rendszeren, laptopon futtatott Firefox böngészőt, melynek pontos adatait a 7.2 táblázat tartalmazza. A továbbiakban: *Desktop FF*.
- Android operációs rendszeren, mobilon futtatott Firefox böngészőt, melynek pontos adatait a 7.4 táblázat tartalmazza. A továbbiakban: *Mobil FF*.

5.4.2. Mérési módszer

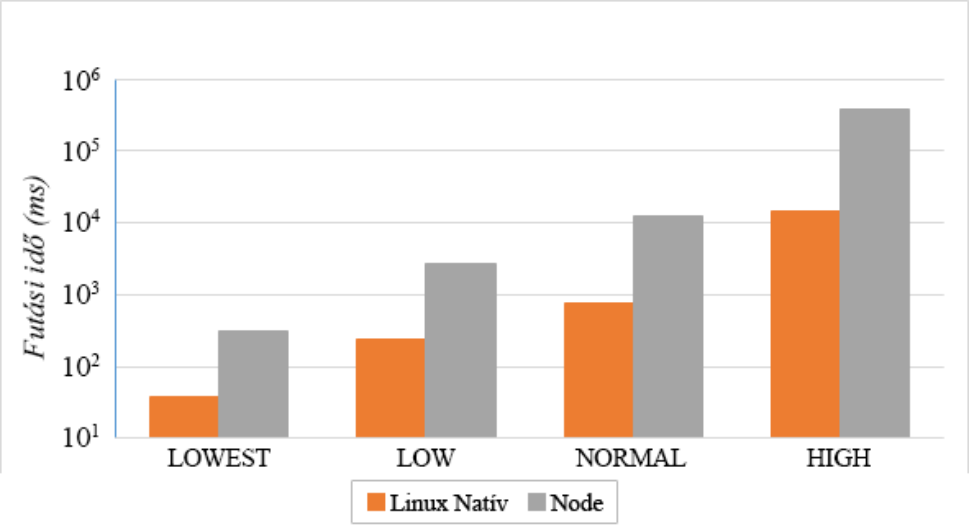
A méréseket minden esetben függvényenként és biztonsági szintenként legalább 20 alkalommal elvégeztük. Minden tesztelt függvényt először 5 bemelegítő iterációban futtattunk, amit 10 mért iteráció követett. A függvényeken kívül lévő memóriaműveletek nem képezték a mérés részét. A mérési módszer pontos leírását a 7.2 alfejezetben ismertetjük.

5.4.3. Mérési eredmények

A teszteket az ABE fő algoritmusain végeztük el az 5.4.1 alfejezetben ismertetett tesztkörnyezetekben.

Setup

A `setup` eljárásra jellemzően kevés alkalommal van szükség, ezekben az esetekben is többnyire nagyteljesítményű szervereken fut. Ebből adódóan úgy éreztük, hogy ennél a függvénynél a tesztet elegendő *Node.js (Node)* és natív C *(Linux Natív)* környezetekben elvégezni. A CryptID könyvtár IBE implementációnak teljesítménytesztjei (Vécsi és mtsai., 2019) alapján úgy gondoltuk, hogy egyértelműen a Linux Natív lesz a gyorsabb, nagyjából háromszor-négyszer.

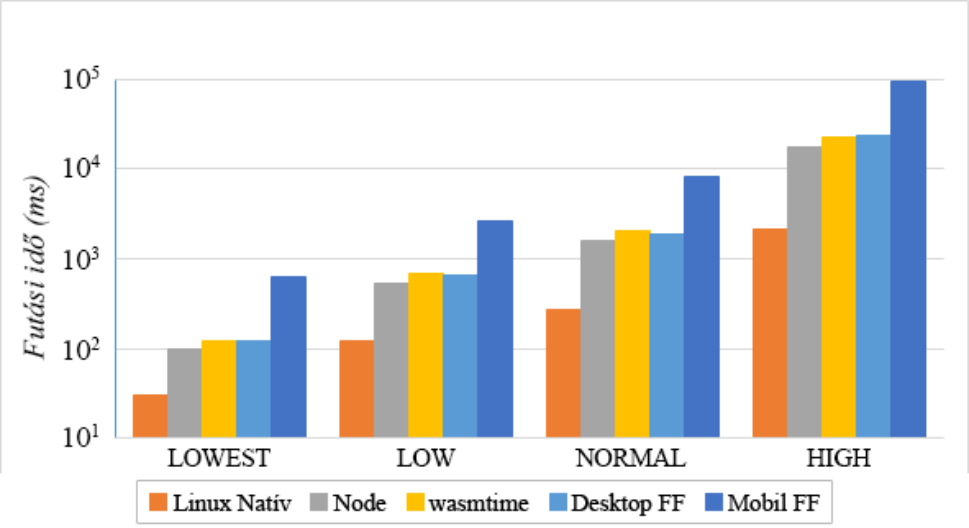


5.1. ábra. A `setup` függvény futási ideje.

Az eredmények tudatában megállapítható, hogy valóban a Linux Natív környezetben végzett tesztek voltak jóval gyorsabbak. Az arány azonban feltételezéseinkhez közel sem volt, a natív környezetben végzett teszt az összesített átlag alapján közel 26-szor hamarabb végzett ugyanazzal a feladattal. Ennek oka a 7.11 táblázat alapján a `setup` eljárásban található prímszám-keresés, amely WASM környezetben jóval több időt vesz igénybe. Amíg *Node.js* környezetben a p prím keresése az idő 84%-át veszi el, addig natív C esetén mindössze 52%-ot. Az eredményt ettől függetlenül a platformfüggetlenség szempontjából nem gondoljuk rendkívül rossznak, hiszen a Setup jellemzően nem WASM-alapon fut, ez inkább a többi függvény esetében fontos.

Keygen

Habár a `keygen` eljárás is többnyire nagyteljesítményű szervereken futhat, ennek ellenére az összehasonlítás érdekében ebben az esetben öt tesztkörnyezetet is ábrázoltunk. A `setup` esetén végzett tesztek eredményét is figyelembe véve kialakított előzetes várakozásaink alapján a natív Linux tesztet vártuk a leggyorsabbnak. Úgy gondoltuk, hogy ezt a különböző WASM futtatókörnyezetek nagyjából hasonló időkkel követik majd, ahol a legnagyobb futási idő a mobil platformon lesz megfigyelhető.

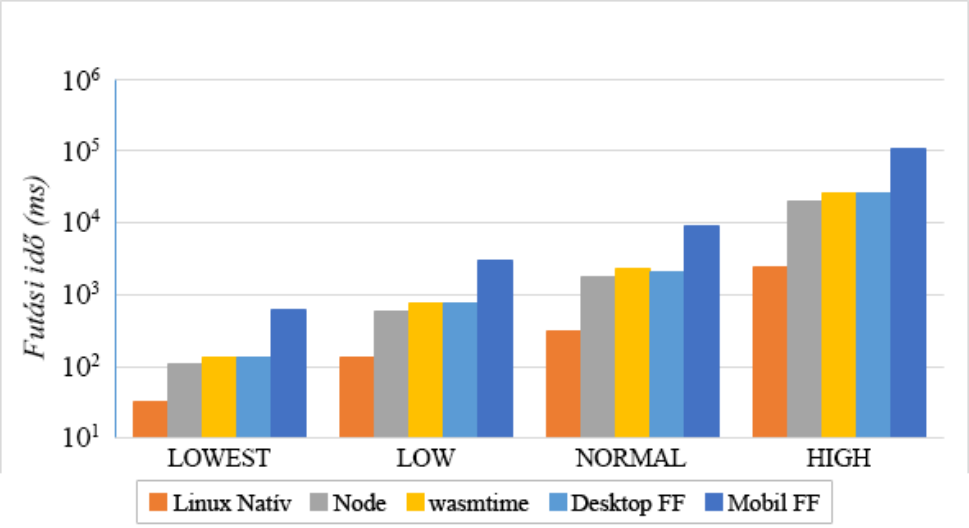


5.2. ábra. A `keygen` függvény futási ideje.

A tesztek a várakozásainkat megerősítették. Az átlagos eltérés a natív Linux és a *Node.js*-ben futtatott WASM között közel nyolcszoros volt, ami határozottan jobb, mint a `setup` esetében. Előbbi körülbelül $0,03$ másodpercet igényelt **LOWEST** biztonsági szinten. A 7.11 táblázat `setup` függvényre vonatkozó részeiből azt a következtetést vontuk le, hogy a jobb eredmény ebben az esetben a prímkeresések hiányának köszönhető. A *wasmtime* átlagosan körülbelül 30%-kal igényelt több időt, mint a *Node.js*, mely várakozásainkhoz képest meglepő volt. A Desktop Firefox mindössze $0,5$%-kal igényelt több időt, mint a *wasmtime*. A mobilon futtatott Firefoxban nagyjából négyszer annyi idő alatt futott le az eljárás, mint a *Desktop* platform esetében. Tehát a leghosszabb futási időket minden esetben mobilon tapasztaltuk. A **LOWEST** biztonsági szint esetén ez itt $0,62$ másodpercet jelentett, amit egy nagyon jó eredménynek tartunk.

Delegate

A `delegate` eljárás esetében is minden, az 5.4.1 alfejezetben említett környezeten elvégeztük a tesztelés ábrázolását. A különböző tesztkörnyezetekben mért futási idő arányánál a várakozásunk hasonló volt, mint a `keygen` esetében, hiszen ez matematikai logikáját tekintve egy nagyon hasonló eljárás.

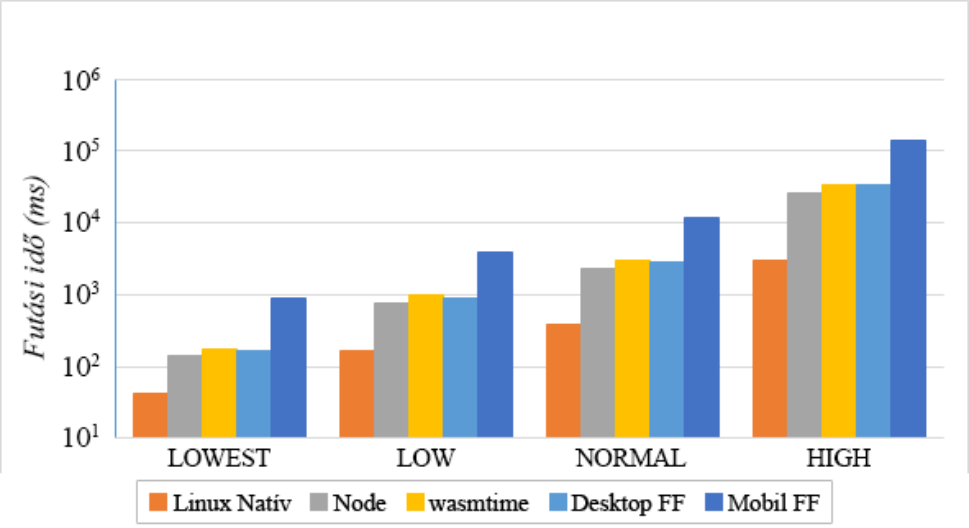


5.3. ábra. A `delegate` függvény futási ideje.

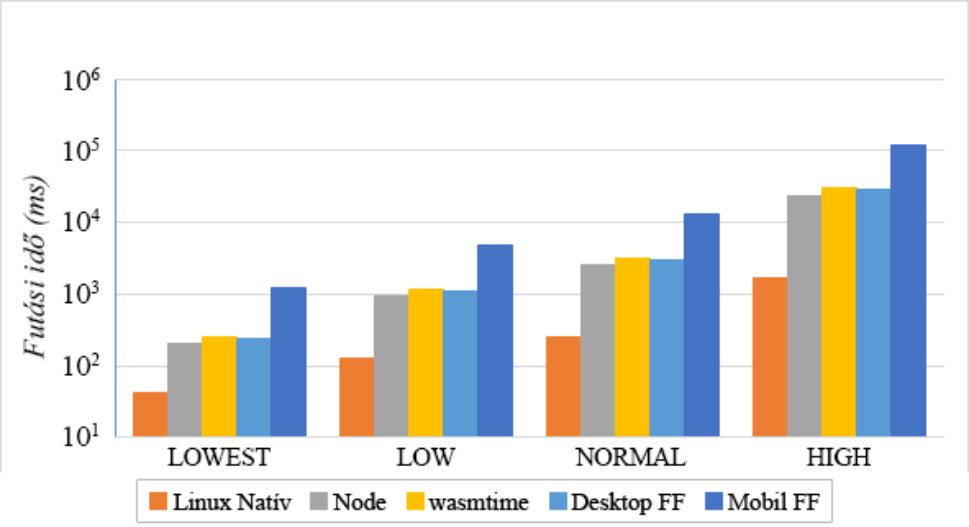
Az adatok valóban az előző függvény méréseihez hasonló mintát mutatnak, amik így a várakozásainkat megerősítették. Az egyetlen eltérés, hogy míg a `keygen` esetében a Windowson lévő böngésző csak a mobilos böngészőt előzte meg, a `delegate` esetében körülbelül 1.7%-kal előzte a Windows Firefox a *wasmtime*-ot. Azonban ezek a kis eltérések a gyakorlatban inkább jelentenek közel azonos eredményt utóbbi két WASM futtatókörnyezetek esetén. Ami erről és az előző diagramról is leolvasható, hogy a *Node.js* minden más WASM környezetet megelőz. Ez az érdekes eredmény további tesztek elvégzésére motivált minket, amelynek eredményeiről a fejezet végén számoltunk be.

Encrypt, Decrypt

Az `encrypt` és a `decrypt` esetében különösen fontos volt az összes korábbi környezeten tesztelni, hiszen ezek a leggyakrabban használt függvények. A tesztek során a két eljárás esetében a teszteket 4 és 280 karakter hosszú titkosítandó szöveggel is lefuttattuk. Tekintve, hogy különböző hosszúságú bemenetek közötti futási idő nem volt szignifikáns, ezért csak a 7.5 táblázatban tüntettük fel az adataikat. A diagramon látható adatok a különböző hosszúságú bemenetekkel mért idők átlagát mutatják. Az arányok tekintetében a `keygen` eredményei alapján azt vártuk, hogy a *Node.js* körülbelül nyolcszor lesz lassabb, mint a natív C platform. Feltételezésünk abból eredt, hogy ezek a függvények a `setup` eljárással szemben szintén nem tartalmaznak olyan időigényes, nehéz feladatokat, mint a prímkeresés.



5.4. ábra. Az `encrypt` függvény futási ideje.



5.5. ábra. A `decrypt` függvény futási ideje.

Az `encrypt` esetében a *Node.js* kicsivel több mint nyolcszor lassabban futott, mint natív C esetén. Ez a várakozásainknak megfelelt. A `decrypt` tizenháromszor volt lassabb ugyanebben a tekintetben. Ez alulmúlta a várakozásainkat. Ennek oka a Tate párosításban keresendő, valamint a többszöri hatványozásban a `DecryptNode` eljárás esetében. A mobileszköz esetében a két eljárás összesített átlaga **LOWEST** biztonsági szint esetén 1 másodperc. Úgy gondoljuk, hogy ez egy jó eredmény, ahogy a **LOW** esetén vett 4,3, és a **NORMAL** biztonsági szint esetén vett 12,6 másodperc is megfelelő a biztonsági szinthez képest. A **HIGH** eljárás azonban közel 130 másodpercig futott, de megjegyzendő, hogy ez asztali környezetben is közel 31 másodpercet igényelt.

5.4.4. Összegzés

Összességében úgy gondoljuk, hogy könyvtárunk megfelel egy platformfüggetlen módon használható megoldásnak. Habár implementációnk a WASM platformon lassabb, mint az egyszerű C bináris esetén, úgy gondoljuk, hogy ez magától értetődő. Ezzel szemben bizonyos környezetekben a WASM így is a legjobb választás lehet.

Az eredményeket látva meglepő volt számunkra a *Node.js* jó eredménye, összesítve nagyjából 29 százalékkal teljesített jobban, mint a *wasmtime*, és nagyjából 26 százalékkal jobban, mint a *Desktop Firefox*. Ennek az oka feltehetőleg a *Node.js* legújabb, `v13.3.0` verziójától elérhető kísérleti natív WASI támogatásában keresendő (*Node.JS WASI API*, 2020). Míg a böngészőben való futtatáshoz egy JavaScriptben fejlesztett futtatókörnyezetet (*Wasmer-JS*, 2020) használtunk, addig a *Node.js* natív módon támogatja a WASI rendszerhívásait. Ennek igazolására Google Chrome-ban is teszteket végeztünk, ami a *Node.js*-hez hasonlóan a V8 motort használja (*V8*, 2020). Ennél tesztjeink eredménye szerint a *Node.js* kicsivel több, mint 30 százalékkal volt gyorsabb. Tekintve, hogy dolgozatunknak erre nincs ráhatása, ezért nem tartottuk annyira fontosnak, hogy diagramokon részletesebben elemezzük ezt az eredményt.

5.5. Platformfüggetlenségi demonstráció

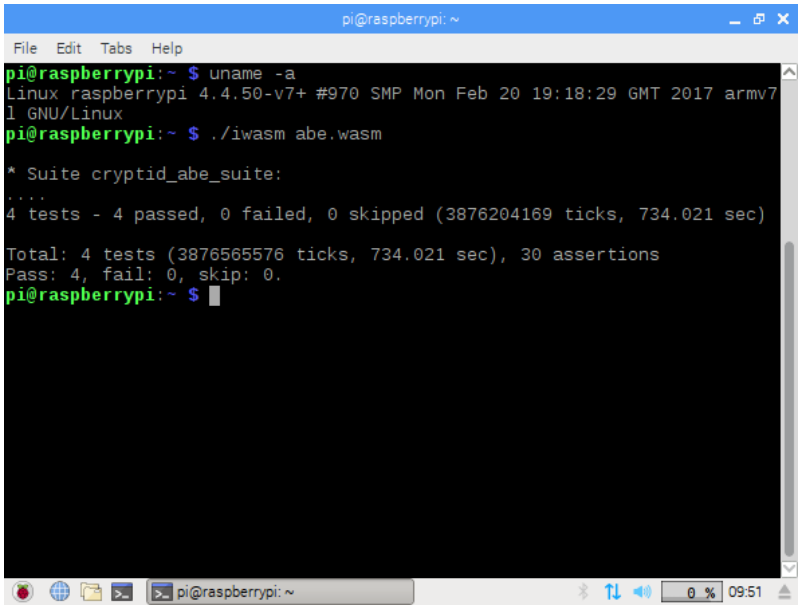
Annak érdekében, hogy a kibővített, WASI által biztosított platformfüggetlenséget alátámasszuk, egy Raspberry Pi eszközön is futtattuk a könyvtár egy tesztjének WebAssembly binárisát. Azért ezen az eszközön végeztük el a demonstrációt, mert ez állt rendelkezésünkre. Fontos azonban megjegyezni, hogy az ezen eszközön történő futtatás nem fedi le az összes lehetőséget, amit a platformfüggetlenség kínál számunkra. Csupán azt szerettük volna bemutatni, hogy az eddig elméleti szinten tárgyalt platformfüggetlenség gyakorlati alkalmazása is lehetséges. A további eszközök listáját az 5.1 alfejezetben is említett WAMR és WASM3 környezetekhez tartozó hivatkozások címén lehet elérni.

A Raspberry Pi általunk használt kiadása a 3-as volt, ennek architektúrája pedig armv7l[2]. Mivel még a WASI fejlesztésének kezdeti fázisában vagyunk, így nem meglepő, hogy az ismertebb beágyazó környezetek, mint például a *wasmtime* még nem rendelkeznek az ARM architektúrák támogatásával. Azonban léteznek már olyan beágyazó környezetek is, amelyek támogatják ezt. Ilyen eszköz például a WebAssembly Micro Runtime **iwasm** virtuális gépe is, amely egy jól konfigurálható eszköz, ami képes WASI modulok futtására (*WebAssembly Micro Runtime - iwasm VM core*, 2019).

Így már minden a rendelkezésünkre áll ahhoz, hogy le tudjuk futtatni a könyvtár egyik

[2] `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`

tesztjéből[3] készített WASI modulunkat.



5.6. ábra. A CryptID ABE tesztjének futása Raspberry Pi 3 eszközön.

Az 5.6 ábrán látható a sikeres futás eredménye, mely igazolja, hogy a CryptID könyvtár általunk létrehozott új verziója valóban képes olyan platformon is futni, melyen az előző verzió nem volt végrehajtható.

[3] `https://github.com/cryptid-org/cryptid-native/blob/master/test/src/BSWCiphertextPolicyAttributeBasedEncryption.test.c`

6. fejezet

Felhasználás

Ebben a fejezetben szeretnénk bemutatni platformfüggetlen Attribute-based Encryption (ABE) implementációnk egy lehetséges felhasználási módját. Ezzel a lehetséges felhasználási móddal az ABE gyakorlati használatát is szemléltetjük.

6.1. Modernizált SSH autentikációs mód

Az SSH protokollban (Lonvick & Ylonen, 2006a) leírt egyik lehetséges autentikálási mód a publikus kulcsok használata. Nagyvállalati környezetekben megoldandó probléma a felhasználónkénti publikus kulcsok karbantartása. Ennek egy modern kivitelezése lehet az általunk elképzelt, ABE-t alkalmazó megoldás. Az ötlet biztonsági és gyakorlati megvalósíthatósági vizsgálata egy jövőbeli kutatás tárgyát képezi.

6.1.1. Bevezető

Habár léteznek a felvetett problémára megoldások (Migeon, 2008), általunk nem volt fellelhető olyan, amely kizárólag kriptográfiai alapú jogosultságkezelést alkalmazna. Úgy gondoljuk, hogy az SSH Transport Layer Protocol-ban (Lonvick & Ylonen, 2006b) leírt publikus kulcs algoritmusok ABE-val való bővítése új lehetőségeket nyitna. A motiváció abból eredt, hogy az ABE használata az egyszerű titkosításhoz hasonlóan az autentikálás esetén is biztosítaná, hogy hozzáférési fákkal *(access tree)* határozzuk meg, kik jogosultak a belépésre. Ez lehetővé tenné annak a megadását, hogy egy adott szerverre való belépésre kik jogosultak, a 3. fejezetben említett példához hasonlóan a következő módon:

(*Programozó* **VAGY** *Projektmenedzser*) **ÉS** *Szervezeti tag.*

Ez azt jelenti, hogy olyan entitások férnek hozzá, akik attribútumlistája tartalmaz *Programozó* vagy *Projektmenedzser*, emellett pedig *Szervezeti tag* attribútumot.

6.1.2. Megvalósítás

Jelen esetben a publikus kulcs autentikációt alkalmazó SSH szerverre csatlakozó, hozzáféréssel rendelkező kliens privát kulcsának publikus párja a szerveren van tárolva. A publikus kulccsal való véletlenszerű adat titkosításának a kliens számára elküldése, majd a kliens által küldött visszafejtett adat megerősítése jelenti a sikeres autentikációt (Lonvick & Ylonen, 2006b; Jonsson & Kaliski, 2003). Amennyiben a kliens és a szerver is támogatja, ez az ABE esetén is megvalósítható elméletben. Az ABE esetén azonban a jelenleg használt titkosítóeljárásokkal szemben a publikus kulcs a rendszer publikus paramétereit tartalmazza, nem egy privát kulcs párja, így ez jóval kevesebb tárolandó és menedzselendő adatot jelent.. A titkosításhoz az eljárás során kell meghatározni az elérési szabályokat hozzáférési fa formájában. Tehát az SSH szervernél nem elegendő egyszerűen az ABE publikus kulcs támogatását lehetővé tenni, a hozzáférési fát is nyilván kell tartani. A gyakorlatban ez véleményünk szerint leghatékonyabban a hozzáférési fa publikus kulcshoz való csatolását jelentené, ami így már a többi kulcshoz hasonlóan lenne tárolható. Ez lehetővé tenné, hogy bonyolult elérési fák határozzák meg, kik jogosultak egy adott szerver elérésére.

Módszerünk lehetőséget nyújt arra, hogy a felhasználók csupán egy privát kulccsal rendelkezzenek, melyek tartalmazzák az összes rájuk jellemző attribútumot, a szükséges mélységeknek megfelelően. Ezt használhatnák az összes, általuk elérni kívánt szervernél való autentikációhoz, hiszen attól függetlenül, hogy az entitás egy privát kulccsal rendelkezik, a különböző szerverek rendelkezhetnek különböző hozzáférési szabályokkal. Ahogy a 3. fejezetben említettük, a mesterkulcsot egy megbízható harmadik fél állítja ki, ennek a segítségével lehet privát kulcsokat létrehozni. Jelen esetben a megbízható harmadik fél a szervert működtető szervezet, aki megfelelően tudja hitelesíteni a hozzáférésre jogosultak attribútumait.

További lehetőségek

Lehetőség lenne arra is, hogy minden privát kulcsot a kiállítás idejét tartalmazó dátumokkal ellássunk a tetszőleges mélységig (például `"év:2020"`, `"hónap:04"`). A számokat összehasonlító szabályok hozzáférési fákká alakítását Bethencourt, Sahai és Waters szemléltették CP-ABE publikációjukban (2007). Ez lehetővé tenné, hogy a publikus kulcsok hozzáférési fájában meghatározásra kerüljön a privát kulcs kiállítási ideje (például csak egy adott hónap után létrejövő kulcsokat engedélyezve), ezáltal akár időkorlátozott kulcsokat generálva.

Utóbbit felhasználva egy adott másodpercben kiállított kulcs is érvényteleníthető, ha olyan szabályt adunk meg, amely alapján a csatlakozás csak akkor engedélyezett, ha az

adott másodperc előtt vagy után állították ki a kulcsot. Ez lehetővé teszi kulcsok érvénytelenítését, úgy, hogy nem kell az összes hozzáférésre jogosult számára új kulcsot generálni. Ennek egy hátránya, hogy tömeges érvénytelenítés esetén komplex fák alkalmazását igényli, melynek kezelése bonyolulttá válhat.

Amennyiben egy adott szerverhez hozzáféréssel rendelkező személyek csoportjaiért több különböző szervezet felel, azok birtokában külön mesterkulcs lehet. Ez lehetőséget adhat arra, hogy minden, a szerver kezelésére jogosult szervezet saját publikus kulcsot és hozzáférési fát határozzon meg. Az általunk vázolt autentikációs móddal ez a lehetőség továbbra is fennáll. A gyakorlatban annak köszönhetően, hogy az ABE egy $1 : N$ titkosítás, N számú privát kulcs jogosultságkezelését is kielégítheti egy hozzáférési fa. Ezzel egy fa és egy publikus kulcs tárolásával a teljes jogosultságkezelés megvalósíthatóvá válik, aminek köszönhetően sok hozzáférésre jogosult entitás esetében rendkívül csökkenhet a tárolandó adat mennyisége.

6.1.3. Összegzés

Elméleti megvalósításunk kriptográfiai alapokon nyugvó megoldást nyújthat az egyik legkritikusabb informatikai protokoll modernizált autentikációjára. SSH kliensekre és szerverekre szinte minden környezetben igény van, így nagy előny az implementációnk által nyújtott platformfüggetlenség, mely akár ehhez a felhasználáshoz is alapot nyújthat.

7. fejezet

Összefoglalás

Dolgozatunk elején bemutattuk az elliptikus görbe kriptográfia, az Attribute-based Encryption (ABE) és a WebAssembly System Inferface (WASI) alapvető tulajdonságait. Ezután áttekintettük a platformfüggetlen ABE implementációnkat, beleértve a teljesítménymérések és a platformfüggetlenség bizonyítása céljából elvégzett tesztünk bemutatását. Munkánk gyakorlatban használható voltát egy elméleti példaalkalmazás kifejtésével is szemléltettük.

7.1. Eredmények

A példa felhasználási lehetőséget részletező fejezetben ismertetettek alapján is elmondható, hogy az ABE rendkívül hasznos alkalmazási lehetőségeknek nyit kaput. Ennek ellenére habár kutatások sorai foglalkoznak vele, viszonylag kevés fellelhető implementáció létezik és ezek nem törekednek platformfüggetlenségre. Dolgozatunk eredményeképpen implementáltuk az ABE-t egy meglévő kriptográfia könyvtár lehetőségeit bővítve. Emellett annak platformfüggetlenségét is továbbfejlesztettük. A WASI-nak köszönhetően már nem csak böngészőkből és *Node.js*-ből használható fel a könyvtár, hanem bármely olyan eszközön, amely rendelkezik WASI beágyazó környezettel. A WASI fiatal volta ellenére is számos IoT eszköz rendelkezik működő beágyazó környezettel. Ezzel a kriptográfiai könyvár korábbi implementációból származó részei is előreléptek, az ABE-hez hasonlóan az IBC platformfüggetlenségét is egy új szintre emelte munkánk.

Számos teljesítménytesztte mutattuk be egyrészről az általunk továbbfejlesztett könyvtár hordozhatóságát és platformfüggetlenségét, különböző szoftverkörnyezetekben és hardvereken való futtathatóságát. Másrészről összevetettük és konzekvenciákat vontunk le teljesítményméréseink eredményeiből. Az 5.5. alfejezetben külön demonstráltuk a platformfüggetlenséget az ARM architektúrájú Raspberry Pi használatával.

7.2. További lehetőségek

Elkészült implementációnk továbbfejlesztésére sok lehetőséget el tudunk képzelni. Ezek között van a különböző, az ABE-t támogató segédfüggvények létrehozása. Erre egy példa az egyszerűbb, negációt nem tartalmazó szám összehasonlító operátorokkal megadott szabályok hozzáférési fákká *(access tree)* való alakítása. Emellett lehetővé szeretnénk tenni az elérési fák egyszerűbb megadását (például szövegből való feldolgozását), ezzel az alkalmazhatóságot egyszerűbbé téve. További komoly továbbfejlődési lehetőség az 5.4 alfejezetben kifejtett teljesítménytesztek eredményeinek javítása. Ehhez elsősorban el kell végeznünk a teljesítménytesztek további, még részletesebb elkészítését és elemzését, majd megfelelő optimalizációs lehetőségek kutatását.

Irodalomjegyzék

About emscripten. (2015). Letöltve, 2020. 04. 23.: https://emscripten.org/docs/introducing_emscripten/about_emscripten.html

Awesome webassembly languages. (2020). Letöltve, 2020. 04. 23.: https://github.com/appcypher/awesome-wasm-langs

Benger, N., Van de Pol, J., Smart, N. P. & Yarom, Y. (2014). "ooh aah... just a little bit": A small amount of side channel can go a long way. *International workshop on cryptographic hardware and embedded systems* (pp. 75–92).

Bethencourt, J., Sahai, A. & Waters, B. (2007, 06). Ciphertext-policy attribute-based encryption. (p. 321-334). doi: 10.1109/SP.2007.11

B. Gladman, W. H. & J. Moxham, e. a. (2015). *MPIR: Multiple Precision Integers and Rationals.* (Version 2.7.0, http://mpir.org)

Bos, J., Halderman, J., Heninger, N., Moore, J., Naehrig, M. & Wustrow, E. (2014, 03). Elliptic curve cryptography in practice. (Vol. 8437). doi: 10.1007/978-3-662-45472-5_11

Boyen, X. & Martin, L. (2007). Identity-based cryptography standard (IBCS) #1: Supersingular curve implementations of the BF and BB1 cryptosystems. *RFC*, *5091*, 1–63. Letöltve: https://doi.org/10.17487/RFC5091 doi: 10.17487/RFC5091

BSW CPABE Toolkit. (2006). Letöltve, 2020. 04. 17.: http://acsc.cs.utexas.edu/cpabe/

Can i use webassembly? (2020). Letöltve, 2020. 04. 23.: https://caniuse.com/#feat=wasm

Changes to the webassembly target. (2019). Letöltve, 2020. 04. 24.: https://releases.llvm.org/8.0.0/docs/ReleaseNotes.html#changes-to-the-webassembly-target

Clark, L. (Szerk.). (2019). *Standardizing wasi.* Letöltve, 2020. 04. 23.: https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/

Compiling webassembly with emscripten. (2015). Letöltve, 2020. 04. 26.: https://emscripten.org/docs/compiling/WebAssembly.html#compiler-output

CryptID. (2020). Letöltve, 2020. 04. 17.: https://github.com/cryptid-org/cryptid-native

Digital Signature Standard (DSS) (Tech. Rep.). (2013, jul). Letöltve: https://doi.org/10.6028/nist.fips.186-4 doi: 10.6028/nist.fips.186-4

General cross-compilation options in clang. (2014). Letöltve, 2020. 04. 23.: https://clang.llvm.org/docs/CrossCompilation.html#target-triple

Getting started with the llvm system. (2020). Letöltve, 2020. 04. 23.: https://llvm.org/docs/GettingStarted.html

Gohman, D. (Szerk.). (2019). *Wasi: Webassembly system interface.* Letöltve, 2020. 04. 23.: https://github.com/bytecodealliance/wasmtime/blob/master/docs/WASI-overview.md

Hankerson, D., Menezes, A. J. & Vanstone, S. (2006). *Guide to elliptic curve cryptography.* Springer Science & Business Media.

Hansen, T. & Eastlake, D. E. (2011, May). *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)* (No. 6234). RFC 6234. RFC Editor. Letöltve: https://rfc-editor.org/rfc/rfc6234.txt doi: 10.17487/RFC6234

Introduction to gmp. (2008). Letöltve, 2020. 04. 25.: https://gmplib.org/gmp-man-6.2.0.pdf#Introduction%20to%20GMP

Jonsson, J. & Kaliski, B. (2003, February). *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1* (No. 3447). RFC 3447. RFC Editor. Letöltve: https://rfc-editor.org/rfc/rfc3447.txt doi: 10.17487/RFC3447

Kate, A. & Goldberg, I. (2010). Distributed private-key generators for identity-based cryptography. *International conference on security and cryptography for networks* (pp. 436–453).

Kernel space definition. (2005). Letöltve, 2020. 04. 24.: http://www.linfo.org/kernel_space.html

Koblitz, N. (1987). Elliptic curve cryptosystems..

Lonvick, C. M. & Ylonen, T. (2006a, January). *The Secure Shell (SSH) Authentication Protocol* (No. 4252). RFC 4252. RFC Editor. Letöltve: https://rfc-editor.org/rfc/rfc4252.txt doi: 10.17487/RFC4252

Lonvick, C. M. & Ylonen, T. (2006b, January). *The Secure Shell (SSH) Transport Layer Protocol* (No. 4253). RFC 4253. RFC Editor. Letöltve: https://rfc-editor.org/rfc/rfc4253.txt doi: 10.17487/RFC4253

Lynn, B. (2007). *On the implementation of pairing-based cryptosystems* (Unpublished doctoral dissertation). Stanford University Stanford, California.

Migeon, J.-Y. (2008). The mit kerberos administrator's how-to guide. *Kerveros constor-*

tium, 6.

Miller, V. (1985, 01). Use of elliptic curves in cryptography. (p. 417-426).

Mills, C. (Szerk.). (2017). *Webassembly concepts.* Letöltve, 2020. 04. 23.: https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts

Möller, N. (Szerk.). (2013). *Mini-gmp.* Letöltve, 2020. 04. 23.: https://github.com/Cl3Kener/gmp/tree/master/mini-gmp

Node.js wasi api. (2020). Letöltve, 2020. 04. 25.: https://nodejs.org/api/wasi.html

OpenABE. (2020). Letöltve, 2020. 04. 17.: https://github.com/zeutro/openabe

Rescorla, E. (2018, August). *The Transport Layer Security (TLS) Protocol Version 1.3* (No. 8446). RFC 8446. RFC Editor. Letöltve: https://rfc-editor.org/rfc/rfc8446.txt doi: 10.17487/RFC8446

Sahai, A. & Waters, B. (2004, 05). Fuzzy identity based encryption. (Vol. 3494). doi: 10.1007/11426639_27

Shamir, A. (1985). Identity-based Cryptosystems and Signature Schemes. *Proceedings of CRYPTO 84 on Advances in Cryptology* (pp. 47–53). New York, NY, USA: Springer-Verlag New York, Inc. Letöltve: http://dl.acm.org/citation.cfm?id=19478.19483

Standalone jit-style runtime for webassembly. (2019). Letöltve, 2020. 04. 23.: https://bytecodealliance.github.io/wasmtime/

V8. (2020). Letöltve, 2020. 04. 25.: https://v8.dev/

Vécsi, Á., Bagossy, A. & Pethő, A. (2019). Cross-platform identity-based cryptography using webassembly. *Infocommunications*, 31.

Wamr – supported architectures and platforms. (2020). Letöltve, 2020. 04. 25.: https://github.com/bytecodealliance/wasm-micro-runtime#supported-architectures-and-platforms

Wang, C.-J., Xu, X.-L., Shi, D.-Y. & Lin, W.-L. (2014). An efficient cloud-based personal health records system using attribute-based encryption and anonymous multi-receiver identity-based encryption. *2014 ninth international conference on p2p, parallel, grid, cloud and internet computing* (pp. 74–81).

Warm. (2020). Letöltve, 2020. 04. 25: https://github.com/cryptid-org/warm#warm---webassembly-arbitrary-precision-math

Wasi libc implementation for webassembly. (2019). Letöltve, 2020. 04. 24.: https://github.com/WebAssembly/wasi-libc#wasi-libc

Wasi – wasi-core. (2020). Letöltve, 2020. 04. 24.: https://github.com/WebAssembly/WASI/blob/master/phases/snapshot/docs.md

Wasm3 – supported platforms. (2020). Letöltve, 2020. 04. 25.: https://github.com/wasm3/wasm3#status

Wasmer-js. (2020). Letöltve, 2020. 04. 25.: https://github.com/wasmerio/wasmer-js

Webassembly – imports. (2016). Letöltve, 2020. 04. 24.: https://webassembly.org/docs/modules/#imports

Webassembly micro runtime - iwasm vm core. (2019). Letöltve, 2020. 04. 23.: https://github.com/bytecodealliance/wasm-micro-runtime#iwasm-vm-core

Webassembly – use cases. (2020). Letöltve, 2020. 04. 23.: https://webassembly.org/docs/use-cases/

Zakai, A. (Szerk.). (2013). *Gmp, a library for arbitrary precision arithmetic, to javascript using emscripten.* Letöltve, 2020. 04. 23.: https://github.com/kripken/gmp.js

Függelék

ABE C példaprogram

```
#include "attribute-based/BSWABE.h"
#define accessTree \
        bswCiphertextPolicyAttributeBasedEncryptionAccessTreeAsBinary
#define accessTree_init \
        bswCiphertextPolicyAttributeBasedEncryptionAccessTreeAsBinary_init
#define publicKey \
        bswCiphertextPolicyAttributeBasedEncryptionPublicKeyAsBinary
#define masterKey \
        bswCiphertextPolicyAttributeBasedEncryptionMasterKeyAsBinary
#define secretKey \
        bswCiphertextPolicyAttributeBasedEncryptionSecretKeyAsBinary
#define encryptedMessage \
        bswCiphertextPolicyAttributeBasedEncryptionEncryptedMessageAsBinary

int main(int argc, char *argv[])
{
publicKey* publickey = malloc(sizeof (publicKey));
masterKey* masterkey = malloc(sizeof (masterKey));
CryptidStatus status = cryptid_abe_bsw_setup(publickey, masterkey, LOWEST);

accessTree* accessTree = accessTree_init(2, NULL, 0, 2);

accessTree* child0 = accessTree_init(1, NULL, 0, 2);
accessTree->children[0] = child0;

char* attribute0 = "Programozó";
accessTree* child0_0 = accessTree_init(1, attribute0, strlen(attribute0), 0);
child0->children[0] = child0_0;

char* attribute1 = "Projektmenedzser";
accessTree* child0_1 = accessTree_init(1, attribute1, strlen(attribute1), 0);
child0->children[1] = child0_1;

char* attribute2 = "Szervezeti tag";
accessTree* child1 = accessTree_init(1, attribute2, strlen(attribute2), 0);
accessTree->children[1] = child1;

char* message = "Teszt üzenet.";
encryptedMessage* encrypted = malloc(sizeof (encryptedMessage));
```

```
40 status =
41 cryptid_abe_bsw_encrypt(encrypted, accessTree, message, strlen(message), publickey);
42
43 int numAttributes = 2;
44 char** attributes = malloc(sizeof(char*) * numAttributes);
45 attributes[0] = attribute0;
46 attributes[1] = attribute2;
47 secretKey* secretkey = malloc(sizeof (secretKey));
48 status = cryptid_abe_bsw_keygen(secretkey, masterkey, attributes, numAttributes);
49
50 secretKey* secretkeyNew = malloc(sizeof (secretKey));
51 int numAttributes2 = 1;
52 char** attributes2 = malloc(sizeof(char*) * numAttributes2);
53 attributes2[0] = attribute2;
54 status =
55 cryptid_abe_bsw_delegate(secretkeyNew, secretkey, attributes2, numAttributes2);
56
57 char* result;
58 status = cryptid_abe_bsw_decrypt(&result, encrypted, secretkey);
59 printf("%s\n", result);
60 }
```

Teljesítménytesztek

Tesztkörnyezetek

Paraméter	Érték
VM típus	Azure Standard F2s_v2
Processzor	Xeon Platinum 8168, 2.7-3.7 GHz
Memória	4 GB
Operációs rendszer	Ubuntu 18.04.4 LTS
gcc verzió	7.5.0
Node.js verzió	v14.0.0
wasmtime verzió	0.15.0
clang verzió	10.0.0

7.1. táblázat. Natív, Node.js és wasmtime tesztekhez használt környezet.

A Node.js tesztek paramétere:

--experimental-wasi-unstable-preview1

--experimental-wasm-bigint --max-old-space-size=4000

Paraméter	Érték
Típus	Dell G3 3779
Processzor	i7-8750H, 2.2-4.1 Ghz
Memória	16 GB, DDR4, 2666 Mhz
Operációs rendszer	Windows 10 Enterprise
Windows verzió	Version 1909 (OS Build 18363.778)
Firefox verzió	75.0 (64 bit)
Chrome verzió	81.0.4044.122 (64 bit)
Chrome V8 verzió	8.1.307.31

7.2. táblázat. Desktop Firefox, Chrome teszthez használt környezet.

Paraméter	Érték
Processzor	Xeon E-2136, 3.3-4.5 Ghz
Memória	12 GB, DDR4, 2666 Mhz
Operációs rendszer	Windows Server 2016 Standard
Windows verzió	Version 1607 (OS Build 14393.447)
Firefox verzió	75.0 (64 bit)
Wasmer-JS (FF) verzió	v0.10.2

7.3. táblázat. Server Firefox teszthez használt környezet.

Paraméter	Érték
Típus	Huawei Mate 20 Lite
Processzor	Hisilicon Kirin 710
Memória	4 GB
Operációs rendszer	8.1.0 - Kernel 4.4.103+
Firefox verzió	68.7.0
Wasmer-JS (FF) verzió	v0.10.2

7.4. táblázat. Mobil teszthez használt környezet.

Mérési módszer

A teljesítménymérésekhez a C-ben elérhető *clock_ gettime* függvényt használtuk. A tesztelés a következőképpen zajlott minden esetben:

- minden tesztelt függvényt először 5 bemelegítő iterációban futtattunk, amit 10 mért iteráció követett;
- az időmérés csak a konkrét `cryptid_abe_bsw_*` függvényre vonatkozott, az azon kívül lévő memóriaműveletekre nem;
- minden biztonsági szinten lefuttattuk a 15 iterációt tartalmazó ciklust kétszer (`setup` esetén háromszor), ami így 20 (`setup` esetén 30) mérést jelent biztonsági szintenként és függvényenként;
- `setup` esetén további 20 mérést végeztünk **LOWEST-NORMAL** biztonsági fokozatok között a Solinas prím generáláson és az 5.2.1 alfejezetben leírt prímkeresésen, melyet a 7.11 táblázat tartalmaz;
- az `encrypt` és a `decrypt` eljárások esetén 4 és 280 karakter hosszú statikus inputokkal is lefuttattuk a tesztet. Ez 4 karakteres inputnál *"helo"*, 280 karakteres inputnál

*"helo"*70* volt;

- a *WebAssembly* tesztek esetén minden esetben ugyanaz a *wasm* bináris volt tesztelve;
- a tesztkód a függelékben is látható példakód volt a többi feltételnek megfelelően átalakítva. A hozzáférési fa a következő volt:

(developer ***VAGY*** *reviewer)* ***ÉS*** *CryptID.*

A *Server Firefox* környezet nem tekinthető jellemző felhasználásnak (hiszen szervereken nem böngészőben szokás titkosítóeljárásokat futtatni), ezért a diagramokból szintén kihagytuk, azonban az eredményt elég érdekesnek találtuk a függelékben való feltüntetéshez. Az adatok leírására a tesztesetekből az átlagot találtuk a legmegfelelőbbnek, az alábbi táblázatok ezt tartalmazzák.

Mérési eredmények

	4 karakteres input	280 karakteres input
Encrypt	12 149,144 ms	12 172,6079 ms
Decrypt	10 677,36 ms	10 684,5005 ms

7.5. táblázat. Titkosítandó szöveg hossza alapján való összehasonlítás, összesített átlag.

	LOWEST	LOW	NORMAL	HIGH
Natív Linux	37,2074 ms	244,9309 ms	770,5689 ms	14 538,9265 ms
Linux Node	315,6845 ms	2759,8146 ms	12 595,2503 ms	386 829,6897 ms

7.6. táblázat. `setup` futási ideje.

	LOWEST	LOW	NORMAL	HIGH
Natív Linux	30,4793 ms	119,4048 ms	274,8311 ms	2094,2105 ms
Server Firefox	83,65 ms	471,15 ms	1387,9 ms	15 263,15 ms
Linux Node	96,9005 ms	538,3327 ms	1601,3804 ms	17 609,4899 ms
Linux wasmtime	120,753 ms	676,1259 ms	2015,5972 ms	22 889,0606 ms
Desktop Firefox	122,25 ms	649,35 ms	1895,85 ms	23 181,85 ms
Desktop Chrome	147,7292 ms	693,5642 ms	2178,5402 ms	22 934,6012 ms
Mobil Firefox	620,65 ms	2658,85 ms	8111,5501 ms	94 043,6 ms

7.7. táblázat. `keygen` futási ideje.

	LOWEST	LOW	NORMAL	HIGH
Natív Linux	33,1311 ms	132,3497 ms	308,0031 ms	2364,1865 ms
Server Firefox	92 ms	515,25 ms	1542,5 ms	17 245,2 ms
Linux Node	106,704 ms	597,1042 ms	1792,0333 ms	19 872,1736 ms
Linux wasmtime	132,2827 ms	750,2649 ms	2252,0739 ms	25 842,2136 ms
Desktop Firefox	131,85 ms	767,55 ms	2090,05 ms	25 500,85 ms
Desktop Chrome	147,5472 ms	710,2937 ms	2459,489 ms	28 481,5722 ms
Mobil Firefox	615,25 ms	2947,3 ms	9060,25 ms	106 210,9 ms

7.8. táblázat. `delegate` futási ideje.

	LOWEST	LOW	NORMAL	HIGH
Natív Linux	41,6549 ms	166,7631 ms	389,6181 ms	3018,342 ms
Server Firefox	120,325 ms	673,1 ms	2015,45 ms	22 813,45 ms
Linux Node	137,6686 ms	779,111 ms	2349,2927 ms	26 304,9978 ms
Linux wasmtime	170,9549 ms	976,706 ms	2950,2219 ms	34 172,792 ms
Desktop Firefox	161,275 ms	913,45 ms	2843,5 ms	33 617,4 ms
Desktop Chrome	205,843 ms	1037,5163 ms	2776,1961 ms	32 951,3261 ms
Mobil Firefox	888,125 ms	3956,725 ms	11 935,15 ms	140 464,95 ms

7.9. táblázat. `encrypt` futási ideje.

	LOWEST	LOW	NORMAL	HIGH
Natív Linux	41,7135 ms	129,5324 ms	256,5249 ms	1635,0507 ms
Server Firefox	170,375 ms	796,875 ms	2133,825 ms	19 381,6 ms
Linux Node	202,0908 ms	945,4424 ms	2539,3419 ms	23 124,5679 ms
Linux wasmtime	251,061 ms	1181,8381 ms	3165,6261 ms	29 651,1864 ms
Desktop Firefox	233,05 ms	1080,725 ms	3048,95 ms	28 346,725 ms
Desktop Chrome	247,8529 ms	1217,4115 ms	2977,5061 ms	29 765,7664 ms
Mobil Firefox	1197,75 ms	4815,55 ms	13 171,725 ms	118 841,2 ms

7.10. táblázat. `decrypt` futási ideje.

	Natív Linux	Linux Node
Teljes idő	350,9024 ms	5223,5831 ms
q generálása	1,9905 ms	10,3604 ms
p keresése	182,7803 ms	4391,6568 ms

7.11. táblázat. `setup` lebontott, átlagolt eredmények.

Saját munka leírása

Mezei Botond

- Elkészítettem a WAMR-t, a hozzá tartozó build szkripttel együtt.
- CI pipelineokat készítettem a WAMR-hez és a CryptID WASI buildjéhez.
- Elvégeztem a platformfüggetlenségi demonstrációt egy Raspberry Pi eszközön.

Szürti Szilárd Dávid

- Implementáltam az Attribute-based Encryption szükséges részeit.
- Teljesítménymméréseket készítettem mobil, asztali (Windows) és szerver (Linux) környezetekben egyaránt a futási idő összehasonlíthatóságának érdekében.
- Kifejtettem az ABE egy lehetséges felhasználási módját.