A NIGHTFALL ESZKÖZ

zk-SNARK ERC 721 és ERC 20-as tokeneken

ABSTRACT
Titkosítási próbálkozás Ethereum hálózaton nyilvános tokenek titkosított küldésére.

Burian Sándor, AWXYHE
Információ és kódelmélet -
NMXIK1HMNE/IK1_EA_MS_N

2020-2021, második félév

Óbudai Egyetem, Neumann János Informatikai kar

Tartalom

Alap probléma

Az etherum egy nyílt blokklánc hálózat ami tokenek küldésére alkalmas. Ezek a tokenek lehetnek bármik gyakorlatilag, így lehet egy privát kulcs megosztása, egy üzenet, fájl[1], vagy akár egy program is. Ugyanakkor a hálózaton minden küldött token, minden tranzakció szabadon megtekinthető és látható, pl az https://etherscan.io/ segítségével. Épp ezért érdekes kérdés lehet, hogy küldhetünk úgy tokeneket, hogy azok valamilyen módon elrejtsék a nagyközönség elől a privát információkat.

Ezért hozta létre az EY a Nightfall eljárást[2]. Ehhez a hálózaton kétféle tokent haszálhatunk:

- ERC20: ezek reprodukálható, megismételhető tokenek, ilyen értelemben fel is oszthatóak akár.
- ERC721: Ezek egyedi, megismételhetetlen tokenek, amik valamilyen sjaát, különleges tulajdonsággal rendelkeznek, mint olyanok akár fizikai, valós megtestesülésük is lehet!

Sajnos aezeknél a módszereknél alapesetben szabadon elérhetőek a publikus kulcsok, de ez a megoldás egy olyan rendszt tesz lehetővé ami közben ezek (is) rejtve maradnak, ugyanakkor fontos, hogy a módszer maga ne ablokklánc hálózaton történjen kiszámításra amennyiben az lehetséges, mert úgy a tranzakció mérete és így a költsége is jelentősen megnő., Épp ezért a felek küön a saját gépükön végeznek egy zk-SNARK eljárást, majd az abból kapott publikus kulcsokat küldik ki a nyilvános blokklánc hálózatra. Ehhez azonban egy bizonyítékot is el kell küldeniük ami azt igazolja, hogy az eljárást megfelelően használták. Ezek után már csak annyi látszik majd a hálózaton, hogy valaki küldött valakinek valamilyne rejtett üzenetet. A kérdés az, hogy küldhetünk úgy rejtett üzenetet, hogy ne legyen visszafefjthető, ha az publikus kulcs és a számítást igazoló értéket is mindneki előtt megosztjuk. Például hasznos lenne, ha a folyamatból kiiktatnánk a fogadó fél azonosságát igazoló nyilvános kulcs publikusságát.

Kiegészítés 1 – az ethereum blokkláncról röviden

Ahhoz, hogy a fenti problémát értsük és kezeljük nem kell mélyen megértenünk az ethereum működését, de pár alapvető dologgal[3] fontos tisztában lenni. A rendszer úgy van felépítve, hogy van egy küldő és egy fogadó fél, ami között a csatorna (a blokklánc maga) teljesen nyílt, minden ott történő kommunikáció bárki számára elérhető és olvasható. A küldő, és a fogadó egy-egy nyilvános kulccsal azonosítja magát, tehát az ő címük is ismert. A csatornán ráadásul minden korábban küldött információ megmarad, tehát, ha valaki tavaly küldött egy tokent azt jövő ilyenkor is látni lehet majd, mivel minden egyes üzenetküldés, (blokkláncechológiákkal foglalkozók körében elfogadott nevén tranzakció) visszakereshető, és bármikor lekérdezhető. A csatorna használata azonban nincs ingyen, ezzel is védve a rendszert a spam jellegű üzenetektől, ezért számít, hogy mekkorák az üzenetek amiket küldeni szeretnénk.

Kiegészítés 2 – az ERC20 és ERC 721-as tokenekről röviden

A kommunikáció tokenekkel történik, tehát minden üzenet ami a csatornán átmegy egy-egy token. Ugyanakkor a tokenek úgy nevezett smart contractoknak is megfeleltethetőek. Ezek két féle tokenek lehetnek. Ahogy fentebb említettük az ERC20 egy reprodukálható tokentípus, ami azt jelenti, hogy kvázi egy karakterláncnak megfeleltethető, a karakterláncot is felvághatjuk,

darabolhatjuk, ez a funkcionalitása miatt van így, hiszen kifejezetetn piaci tranzakciók modelleként találták ki[4]. Szemben az ERC 721-as tokenekkel melyek teljesen egyediek, így alkalmasak akár fizikai tárgyak, valós eszközök virtualizálására is. Ez a token nem feloszthető, nem módosítható, csak és kizárólag átadható.[5] Ilyen megoldás ideális lehet nem csak a fzikai valóság digitalizálására, de teljesen digitális értékek egyediség azonosítására is, például domain nevek, közösségi média profilok, hasonlók.

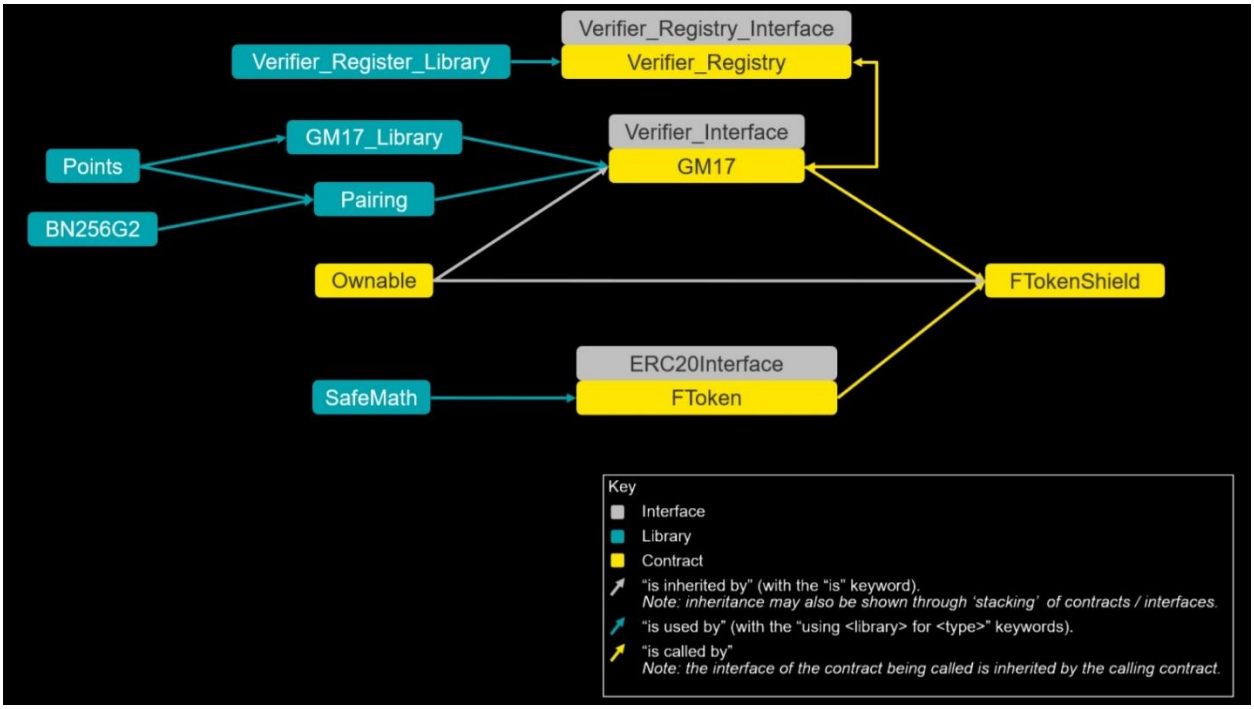
Összefoglalva az elgondolás röviden

Megoldásként a küldő fél a saját lokális gépén alkalmazza az zk-SNARK eljárást, ami egyszerre több kimeneti eredményt is ad, mind titkosított, és így a hálózatra kiküldhető, ez idegen szemlélő számára értelmetlen, de a küldő és a fogadó össze tudja rakni a részeket. Azonban azért, hogy a szemlélő harmadik fél felé se legyen kezlehetetlen a probléma ezért a küldő egy igazolást is megoszt a hálózaton, mely igazolja, hogy ő bizony elvégezte, és helyesen végezte el a számításokat. Ezen igazolás és a részek alapján visszaszámolható maga a számítás, és igazolható a helyessége, de az üzenet nem fedhető fel!

Nightfall

ERC20-as tokenek esetében

A felbontható tokenek esetében az alábbi ábbra mutatja be mi történik nagy vonalakban:



ábra 1 - Nightfall ERC20-as tokenekkel, forrás: nightfall whitepaper

Hitelesítőkulcs ZoKrates használatával

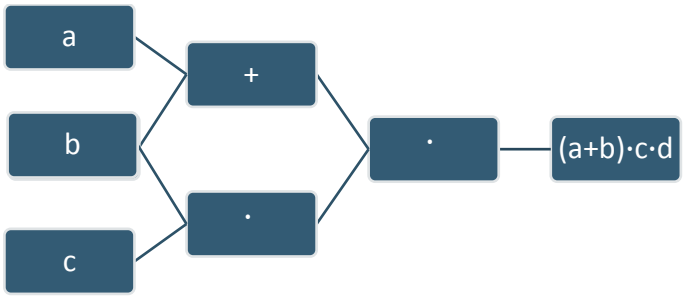
A hitelesítést a Nightfall eljárás ZoKrates segítségével végzi ami miatt az, hogy mennyi idő alatt sikerül ezt hitelesítést előállítani az rendkívűl változó, sajnos sok mindentől függ, ezért a folyamat nem igazán becsülhető meg konstans időben.

A zk-SNARK eljárás

A ZKP protokollok, azaz a *zero-knowledge proof* protokollakat még 1985-ben publikálták[6]. Azóta számos változtatáson átestek, ez a *tömör, nem interaktív tudásalapú* verziója[7] az amiről most beszélünk. Ennek az eljárásnak az a lényege, hogy bizonyítja egy információ meglétét anélkül, hogy magát az információt elárulná a hitelesítő felé. Például egy véletlen szám létét igazolhatja, sőt azt is akár, hogy tudjuk mi az a szám, anélkül, hogy elmondanánk a számot magát. Jelenleg ennek a technológiának a legleterjedtebb felhasználása a Zcashnél figylehető meg. A Zcash nem egy Ethereum alapú blokklánc hálózat, sőt még csak nem is ERC20 szerű tokeneket használ, ez az eljárás szempontjából nem lényeges, de fontos megemlítenünk.

Jelenleg használt algoritmusunk úgy kezdődik, hogy létrehozunk egy közös karakterláncot ami az ellenörző és a bizonyító számára is ismert. Nevezzük ezt *bizonyító karakterlánc*nak[1]. Ez a rendszer nyilvános eleme! Ez egy lehetséges veszélyforrás is, hiszen, ha valaki ismeri a véletlenszám algoritmust amivel ezt a karakterláncot gyártottuk akkor képes lehet egy hamis bizonyító karakterláncot[2] alkotni!

Ezután létre szeretnénk hozni egy validáló értéket a tranzakciónkból. A tranzakciót validáló függvénnyé alakító folyamat első lépése, hogy alapvető matematikai műveletekre bontjuk szét: összeadás, kivonás, szorzás, osztás[3].



folyamatábra 1 – „(a+b)×b×c" felbontása, forrás: z.cash[8] alapján.

Erre a fentebbi példa alapján gondolhatunk úgy, hogy *a,b,c* értékek „átutaznak" balról jobbra. Úgyhogy akkor a mi feladatunk ellenőrizni, hogy helyesen utaznak át. Ez az R1CS ~ *Rank 1 Constraint system*, tehát a fenti példa alapján *b* és *c*-ből lesz-e b×c?

Ehhez, hogy ezt elég nagy valószínűséggel leellenőrizzük polinomokként kezeljük a folyamat pontjait és ha egy művelet végére az eredmény nem egy polinomiális érték akkor helytelen a művelet. Ezt az ellenőrzést igazából egy általunk véletlenszerűen választott pontján is elvégezhetjük ennek a folyamatnak. Fontos megjegyezni, hogy ezt a pontot szitén titokban kell tartani, hiszen ha tudjuk, hogy hol ellenőrzünk akkor előállíthatunk egy azon a ponton hiteles, végeredményben viszont nem hiteles folyamatot! Ahhoz tehát, hogy a polinomokat „vakon" érétkeljük nem triviális milyen technikát haszálunk. Erre homomorf titkosítást, használhatunk ami lehetővé teszi az adatok használatát dekódolás nélkül is[8]; vagy elliptikus görbéket akár, ami szintén, más algoritmusoknál, digitálsi aláírásoknál is bevett eljárás. Ezeket alkalmazva igazából sem az ellenörző, sem a bizonyítékot létrehozó nem tudja előre hol van pontosan az a pont a műveletfolyamban ahol ellenőrzünk, ugyanakkor pontosan meghatározható mindkét fél számára a

pont, így mindketten ugyanott ellenőrizhetnek. Ezzel a bizonyíték részét le is tudtuk a folyamatnak.

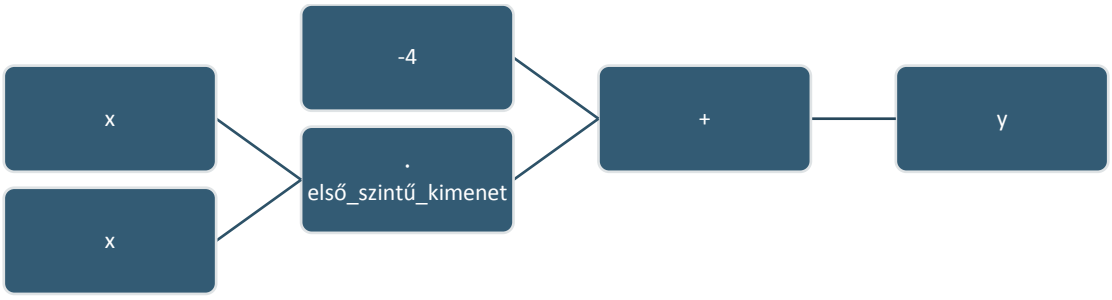
Következik a bemenet titkosságának megőrzése, és annak ellenőrzése, a „zk”-része a protokollnak. Ehhez alapvetően akár az is elegendő lenne, hogy a polinomokat eltoljuk az eredetihez képest, véletlenszerű pontokon. Ez matematikailag nem változtatna a kimeneten, ez könnyen belátható.

A zk-SNARK eljárás hiányosságai és példavégigszámítás

Eddig említettük, hogy a folyamat sebezhető, ha bizonyos adatok véletlenül kiszivárognak, ilyen a véletlenszám generátor algoritmusunk, vagy a folyamatot ellenörző polinom művelet folyamat beli pontja. De a fentebb már taglalt hiányosságokon kívül van más problémánk is. Ahhoz, hogy az eljárást használhassuk teljesen át kell alakítanunk a bemenetet. Például[9], ha azt szeretnénk igazolni, hogy tudunk egy megoldást $x^2 - 4 = 0$ egyenletre[4], anélkül, hogy elmondanánk mi maga a megoldás, akkor átfogalmazva az egyenletünk így néz ki: $x^2 - 4 = y$, ahol x egy titkos érték ami csak az eljárást használó ismer ami y-hoz vezet. Ez két problámhoz vezet. Annak akinek bizonyítunk, meg kell engednünk, hogy ismerje y-t de nem ismerheti a folyamatot ahogy azt kaptuk, hiszen, ha csak annyit mondanánk, hogy y = 0 a fenti egyenletre igaz volna és máris értelmét vesztené az egész. Tehát az alábbi folyamaton megyünk végig:

1. Ehhez az egyenletett felbontjuk a már ismert, tárgyalt módon, de két részre:
   a. x=y | y ∈R; és
   b. x = y m*ű*v. z | m*ű*v.∈{+,−,·,/}

   Azaz:



folyamatábra 2 - A Korábban bemutatott felbontás alkalmazása a konkrét példánkon

2. Követező lépés R1CS alakítás. Az R1CS tulajdonképpen egy vektorhármas $(\overrightarrow{a_i}, \overrightarrow{b_i}, \overrightarrow{c_i})$ és az eredmény vektoruk $\vec{s}$, alább $\langle \cdot, \cdot \rangle$ skalár szorzata a vektoroknak, és keressük $\vec{s}$-t amire igaz az állítás,hogy: $\langle \overrightarrow{a_i}, \vec{s} \rangle * \langle \overrightarrow{b_i}, \vec{s} \rangle - \langle \overrightarrow{c_i}, \vec{s} \rangle = 0 \mid \forall$ i∈N. Ugyanezt felfoghatjuk úgy is, mintha olyan vektort alkotnánk ami a bemeneti értékekkel van feltöltve:
   $\vec{s} = \begin{pmatrix} 1 \\ x \\ első\_szintű\_kimenet \\ y \end{pmatrix}$. Ehhez hasonlóan létrehozzuk $\overrightarrow{a_i}, \overrightarrow{b_i}, \overrightarrow{c_i}$ vektorokat is.
   Tehát ahhoz, hogy az *első_szintű_kimenet*-et megkapjuk $\overrightarrow{c_i}$ képében az alábbiak lesznek a vektoraink:
   $\overrightarrow{a_1} = (0\ 1\ 0\ 0)$; $\overrightarrow{b_1} = (0\ 1\ 0\ 0)$; $\overrightarrow{c_1} = (0\ 0\ 1\ 0)$;
   Ahhoz, hogy $y = x^2 - 4$ -et megkapjuk $\overrightarrow{c_2}$ képében a következő vektorokat használjuk:

$\overrightarrow{a_2} = (-4\ 0\ 1\ 0); \overrightarrow{b_2} = (1\ 0\ 0\ 0); \overrightarrow{c_2} = (0\ 0\ 0\ 1)$

Tehát az első sor, az $első\_szintű\_kimenet$ -ig, tehát $x^2 = 0$-ig a következő polinom formában lesz:

$\langle \overrightarrow{a_l}, \vec{s} \rangle * \langle \overrightarrow{b_l}, \vec{s} \rangle - \langle \overrightarrow{c_l}, \vec{s} \rangle = \sum_{i=1}^{n} a_{1,i} s_i * \sum_{i=1}^{n} b_{1,i} s_i - \sum_{i=1}^{n} c_{1,i} s_i = x \cdot x - első\_szintű\_kimenet = 0$

A második sor, az $x^2 - 4 - y = 0$-ig a következő :

$\langle \overrightarrow{a_2}, \vec{s} \rangle * \langle \overrightarrow{b_2}, \vec{s} \rangle - \langle \overrightarrow{c_2}, \vec{s} \rangle = \sum_{i=1}^{n} a_{2,i} s_i * \sum_{i=1}^{n} b_{2,i} s_i - \sum_{i=1}^{n} c_{2,i} s_i = első\_szintű\_kimenet - 4 - y = 0$

Ez egyenlő azzal mintha a következő Python kódrészletet használnánk:

```
def f(x):
    out_1 = x * x
    y = out_1 - 4
```

Ezzel megkaptuk az R1CS alakítást és megkaptuk a vektorhármast.

3. Végezetül egy másodfokú hozzárendelési problémát kell megoldanunk, hogy a vektorokat polinomokká alakítsuk. A vektorainkat sorra megfeleltetjük egy-egy polinomnak, például $A_i(n) = \overrightarrow{a_{n,l}}$ és így tovább $B_i(n)$, $C_i(n)$ esetében is, hogy $i \epsilon [1, n] \mid n\ a\ vektor\ elemek\ száma$. Így konkrétan:

$A_1(x) = -4x + 4$
$A_2(x) = -x + 2$
$A_3(x) = x - 1$
$A_4(x) = 0$

$B_1(x) = x - 1$
$B_2(x) = -x + 2$
$B_3(x) = B_4(x) = 0$

$C_1(x) = C_2(x) = 0$
$C_3(x) = -x + 2$
$C_4(x) = x - 1$

Ezek alapján az egyenletünk: A(x)*B(x)-C(x) = H(x)*Z(x) $\mid \forall\ x \in R$

Ahol:

$\overrightarrow{A(x)} = (A_1(x)\ A_2(x)\ A_3(x)\ A_4(x))$
$\overrightarrow{B(x)} = (B_1(x)\ B_2(x)\ B_3(x)\ B_4(x))$
$\overrightarrow{C(x)} = (C_1(x) C_2(x) C_3(x) C_4(x))$
$A(x) = \langle \vec{A}, \vec{s} \rangle ; B(x) = \langle \vec{B}, \vec{s} \rangle ; C(x) = \langle \vec{C}, \vec{s} \rangle$; Z(x) = (x-1)(x-2)

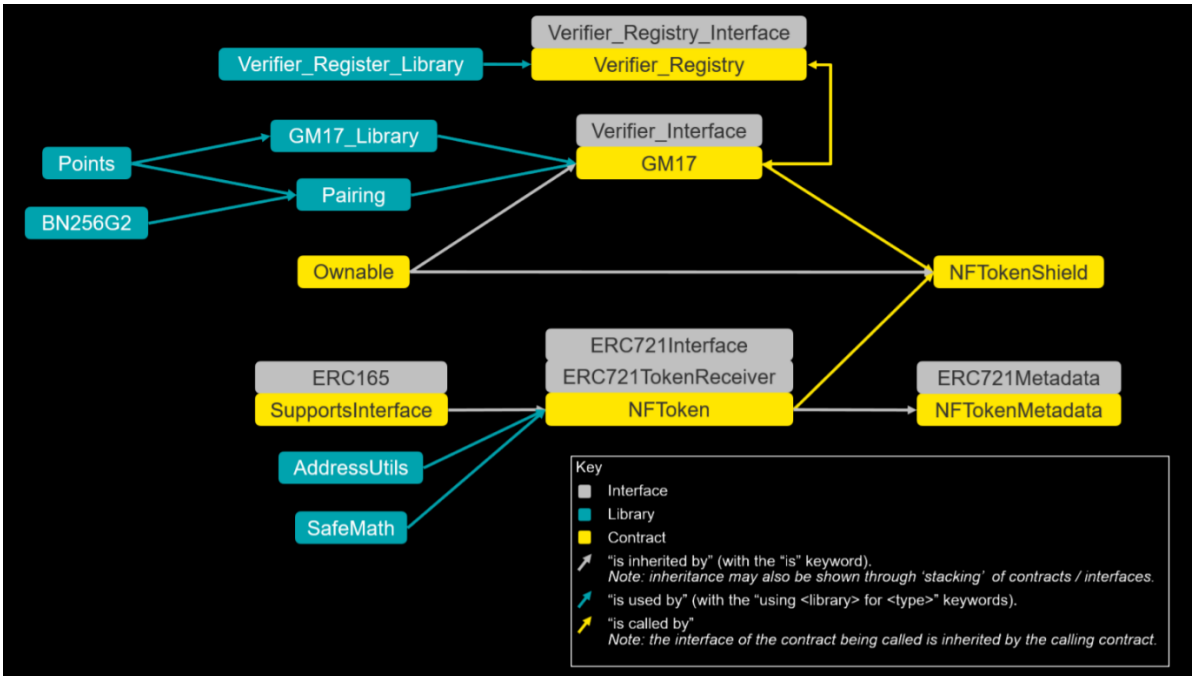
Tehát A(x)*B(x)-C(x) = 0 | x∈ {1,2} → A(x)*B(x)-C(x) maradék nélkül osztható (x-1)(x-2)-vel. Tehát $\exists H(x)$: $\mathrm{A(x)} * \mathrm{B(x)} - \mathrm{C(x)} = \mathrm{H(x)(x-1)(x-2)}$. Tehát $\mathrm{H(x)} = \frac{\mathrm{A(x)*B(x)-C(x)}}{Z(x)}$. Tehát gyakorlatilag a végeredményünk $\vec{s}$ lenne, de még nem oldottuk meg azt a problémát, hogy honann fogja tudni az ellenörző az egyenlet eredményét.

Ehhez viszont egy olyan problémát kell megoldanunk, hogy kiértékeljünk egy polinomot egy olyan helyen ahol nem tudjuk mi a polinom értéke!

Ha adott a következő polinomunk: $P(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ akkor ismernünk kell x-et ahhoz, hogy kiértékeljük. Ellenben, ha van egy függvényünk mely kiértékeli nekünk y-t akkor: $f(x) = y$ és $x \neq y$ akkor $f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$. Ekkor ha valaki ellenőrizni szeretné a P polinom érétkét x_0 pontban akkor megnézi mi f(x_0) beli értéke, legyen z. Ezt elküldi a küldőnek, aki nem tudja mi volt x_0.
Tehát ekkor még egyszer: P(x) = a_1x. Tehát: P(z) = a_1z. Tehát $\mathrm{P(z)} = (\mathrm{P(f}(x_0)) = a_1f(x_0) = f(a_1x_0) = f(P(x_0))$. Tehát x_0-t kiszámoltuk $f(P(x_0))$-ből. Tehát ha a vizsgáló harmadik fél ismeri P(z)-t előre a kódoló személytől, kiszámolhatja $f(P(x_0))$-t, mivel f, P, x_0 ismert. Ha a két eredmény egyezik akkor biztosak lehetünk abban, hogy a küldő kiszámolta x_0-t mivel egy x'_0-ra eltérő eredményt kaptunk volna. A vizsgáló félnek egyedül a polinom fokát kell még ismernie, ehhez a művelethez. És megoldottuk a problémát, úgy küldtünk függvény értéket, hogy azt más leellenőrizhette, és közben nem tudta meg mi volt az.

ERC 721-es tokenek esetében

Most, hogy tisztásztuk, hogy működik részletesen a protokoll a felbontható tokeneknél, felmerül a kérdés, hogy de mit csinálunk a felbonthatatlan tokenek esetében. Alább a vázlat:



ábra 2- Nightfall ERC721-es tokenekkel, forrás: nightfall whitepaper

Jól látszik, hogy egy kis token technikai részletnél nincs nagyobb változás, a titkosítási rész ugyanúgy működik tovább.

Források:

[1] L. Kasem (2019 07 30), *Using Blockchain To Securely Transfer And Verify Files*, Elérhető: (2021 04 14), forrás: devblogs.microsoft.com: https://devblogs.microsoft.com/cse/2019/07/30/using-blockchain-to-securely-transfer-and-verify-files/

[2] C. Konda, M. Connor, D. Westland, Q. Drouot, P. Brody (2019 05), *Nightfall* (whitepaper), Elérhető: (2021 04 09), forrás: github.com: https://github.com/EYBlockchain/nightfall/blob/master/doc/whitepaper/nightfall-v1.pdf

[3] Burián S. (2021 04 16), Bevezetés a blokkláncprogramozásba – jegyzet, Elérhető: (2021 04 16), github: https://github.com/gabboraron/bevezetes_a_blokklancprogramozasba/blob/main/README.md

[4] N. Reiff (202 08 06), What is ERC-20 and What Does It Mean for Ethereum?, Elérhető(2021 04 25), investopedia: https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/

[5] Győrfi A. (2020 12 22), Mik azok a Non-fungible tokenek(NFT)?, Elérhető: (2021 04 25), coincash.eu: https://hu.coincash.eu/blog/mik-azok-a-non-fungible-tokenek-nft

[6] S. Goldwasser, S. Micali, C. Rackoff (1989 02), The knowledge complexity of interactive proof systems, Elérhető(2021 04 25), forrás: mit.edu: http://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Proof%20Systems/The_Knowledge_Complexity_Of_Interactive_Proof_Systems.pdf

[7] C. Reitwiessner (2016 12 05), zkSNARKs in a nutshell, Elérhető: (2021 04 25), ethereum.org: https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/

[8] Ismeretlen, Homomorphic encryption, Elérhető: (2021 04 27), wikipédia https://en.wikipedia.org/wiki/Homomorphic_encryption

[9] Ismeretlen (2018 09 10), Introduction to zk- SNARKs (Part1), Elérhető: (2021 04 25), decentriq.com - academia https://blog.decentriq.com/zk-snarks-primer-part-one/