Elliptikus görbe kriptorendszerek II. rész

Elliptic Curve Cryptography – algoritmusok és a gyakorlat

Egy jó ideje egyre több helyen találkozhatunk az ECC betűhármassal, mint egy kriptorendszer jelölésével. A cikk első részében áttekintettük e viszonylag fiatal kriptorendszer elvi alapjait, most jöjjön néhány gyakorlati példa és algoritmus!

Remélem az első rész senkit nem rettentett el az ECC-től és az elmúlt egy hónap mindenkinek elég volt ahhoz, hogy a kissé száraz előkészítést magáévá tegye. Lássunk néhány algoritmust, melyek rettentően hasonlítanak néhány meglévőre...

Titkosítás és aláírás az elliptikus görbékkel

Az ECDLP-n alapuló rendszerek többsége aláíró (például ECDSA) vagy kulcscserélő (például ECDH) rendszer, mert gyors titkosításra ez a módszer is alkalmatlan, hasonlóan a többi nyíltkulcsos algoritmushoz. A továbbiakban eme algoritmusokkal ismerkedhetünk meg. (A folytatás jobb megértéséhez ajánlott ismeretek: Diffie-Hellman kulcscsere, ElGamal titkosítás valamint az üzenetpecsét algoritmusok ismerete – legalább nagyvonalakban. Ezek nélkül is érthető lesz a folytatás, csak nehezebb lesz párhuzamot találni a már meglévő algoritmusok és az EC-alapú algoritmusok között...)

ECDH – Elliptic Curve Diffie-Hellman kulcscsere

Az eredeti Diffie-Hellman a szimmetrikus titkosító rendszerek kulcsmegosztási problémáját oldatta meg. Hogyan is? A két résztvevő ugyanazokat a műveleteket végezte el egyező nyilvános és különböző titkos paraméterekkel, de azonos eredményt kaptak, melyet kulcsként használhattak. Az ECDH is ugyanígy működik, csak nem moduláris hatványozást használ, hanem a korábban megismert EC-műveleteket.

Alice és Bob megegyeznek egy ***E*** görbében és egy **G** pontban, utóbbit bázispontnak hívjuk. A továbbiakban eme paramétereket nyilvános rendszerparamétereknek tekintjük. Alice választ egy véletlen számot, (amely kisebb, mint a **G** pont rendje) és ugyanígy tesz Bob is: Alice száma legyen ***a***, Bobé legyen ***b***. Mindketten titokban tartják választásukat. A kulcscsere következő lépésében Alice kiszámolja ***a*G*** pontot, melyet elküld Bobnak, aki Alice műveletéhez hasonlóan kiszámolja ***b*G*** pontot és elküldi Alicenak. Végül Alice a Bobtól kapott ***b*G***-t megszorozza ***a***-val, így megkapja ***a*b*G*** pontot, valamint Bob az Alice-tól kapott ***a*G*** pontot szorozza meg saját titkos ***b*** számával, és eredményül ő is az ***a*b*G*** pontot kapja. A közös pont valamely tulajdonsága (például x vagy y koordinátája vagy éppen x + y) használható kulcsként. A kíváncsi Evenek az *ab**G*** pontot kellene kiszámolnia, de csak **G**, *a**G*** és *b**G*** pontokat ismeri, magukat a titkos *a* és *b* számokat nem. Az elliptikus Diffie-Hellman működését és lépéseit az alábbi egyszerű számpélda alapján követhetjük:

```
Nyilvános paraméterek:
  E: y² ≡ x³ + 5x + 8 mod 23  és  G(8,10)
Titkos paraméterek:
  a=7 (Alice titkos száma)
  b=3 (Bob titkos száma)
Kulcselőkészítés:
  Alice számol:          Bob számol:
  1G(8,10)               1G(8,10)
  2G(13,4)               2G(3,4)
  3G(20,9)               3G(20,9)
  4G(22,18)
  5G(6,1)
  6G(2,18)
  7G(7,15)
  aG = (7,15)            bG = (20,9)
Kommunikáció:
  Kicserélik eredményeiket aG-t megkapja Bob, bG-t pedig Alice
Kulcsegyeztetés:
  1bG (20,9)             1aG (7,15)
  2bG (12,18)            2aG (13,19)
  3bG (7,8)              3aG (6,1)
  4bG (22,5)
  5bG (8,13)
  6bG (13,4)
  7bG (6,1)
  abG (6,1)              baG (6,1)
```

ECElGamal – Elliptic Curve ElGamal titkosítás

Ahogy az eredeti ElGamal titkosítás a Diffie-Hellman algoritmus problémáján alapul, úgy építhető fel az elliptikus ElGamal is az ECDH-ra:

1. Alice és Bob választ egy ***E*** görbét és egy **G** bázispontot.
2. Mindketten választanak egy-egy véletlen ***a*** és ***b*** számot, mint titkos kulcsot.
3. Alice elküldi az ***a*G*** pontot, mint nyilvános kulcsot Bobnak.
4. Bob elküldi a ***b*G*** pontot, mint nyilvános kulcsot Alice-nak.

5. Ha Alice üzenni akar Bobnak, az üzenetet leképzi a görbe egy vagy több ***M*** pontjára és generál egy véletlen ***k*** számot, mint viszonykulcsot. Elküldi Bobnak a *(kG, M+k(bG)*) üzenetpárost.
6. Bob a következőképpen olvassa el az üzenetet: a kapott küldemény első felét megszorozza saját titkos ***b*** számával, így ***bkG***-t kap, amit egyszerűen kivon a küldemény második feléből.

Eve támadásának feltétele az lenne, ha Bob átküldött nyilvános kulcsából ki tudná számolni ***b*** értékét vagy a titkosított üzenet első részéből ***k*** számot. Jelenlegi ismereteink szerint egyikre sem képes elfogadható időn belül.

ECDSA – Elliptic Curve Digital Signature Algorithm
A következő algoritmus a FIPS 186-2-ben leírt DSA-hoz hasonlóan működik, hasonlóak a végzett műveletek, lépések is. Ahhoz, hogy Alice egy ***M*** üzenetet aláírva el tudjon küldeni, a következő paraméterek és eszközök szükségesek:

1. egy elliptikus görbe *mod q* felett (nyilvános paraméter)
2. egy **G** bázispont, melynek rendje *n* (nyilvános paraméter, $n \geq 160$ bit)
3. egy véletlen *d* szám (mely kisebb, mint *n-1*) és egy **Q**=*dG* pont. Alice kulcspárja (*d,***Q**), ahol *d* a titkos és **Q** a nyilvános kulcsa.

Az ECDSA aláíró algoritmusa

1. Alice választ egy ***k*** számot *1* és *n-1* között
2. Kiszámolja ***kG***=(x_1,y_1) pontot és ***r***=$x_1 \bmod n$. Ha a pont x koordinátája zérus ($x_1=0$), akkor új ***k*** számot választ. A pont x koordinátája lesz az aláírás egyik komponense, ezért jelöltük meg külön egy *r* betűvel.
3. Kiszámolja ***k*** multiplikatív inverzét *n*-re (vagyis $k^{-1} \bmod n$ értékét).
4. Kiszámolja a küldendő üzenet pecsétjét, melyre a szabvány az SHA-1 algoritmust ajánlja. Legyen hát ***e*** = *SHA-1(M)*!
5. Az aláírás másik alkotóeleme a következő kifejezés: ***s***=$k^{-1}(e + dr) \bmod n$. Abban a szerencsétlen (és meglehetősen ritka) esetben, ha ***s***=*0*, akkor az egész algoritmust elölről kell kezdeni. Itt azt is láthatjuk, hogy a 2. lépésben miért nem lehet ***r***=*0*, mert az aláírás nem tartalmazná a titkos kulcsot!
6. Az ***M*** üzenethez és Alice-hoz tartozó aláírás: az (***r,s***) értékpáros.

Az ECDSA ellenőrző algoritmusa
Feltételezzük, hogy Bob, mint az aláírás ellenőrzője rendelkezik a hitelesített(!) rendszerparaméterekkel és Alice hiteles nyilvános kulcsával. Bob a következő lépések végrehajtásával ellenőrizheti egy aláírás helyességét:

1. Ellenőrzi, hogy az (***r,s***) egész számok megfelelő intervallumban vannak-e? (Kisebbek-e *n-1*-nél?)
2. Kiszámolja a kapott üzenet pecsétjét. Ehhez ugyanazt az algoritmust kell használnia, amit Alice használt: ***e*** = *SHA-1(M)*.
3. Kiszámolja ***s*** multiplikatív inverzét *n*-re: $w = s^{-1} \bmod n$. Ezért nem hagyhattuk az aláírás 5. lépésében, hogy *s=0* legyen: nem létezne inverze!
4. Kiszámolja a következő részeredményeket: $u_1=ew \bmod n$ és $u_2=rw \bmod n$.
5. Végül kiszámolja a **P**$(x_1,y_1) = u_1\mathbf{G} + u_2\mathbf{Q}$ elliptikus pontot. Ha **P**=***O***, akkor biztosan nem jó az aláírás, egyébként legyen $v = x_1$!
6. Bob az aláírást csak akkor fogadja el, ha $v = r$.

Miért helyes az ellenőrzés?
Az aláírás-ellenőrzés helyességéhez az kell belátnunk, hogy az 5. pontban kiszámolt **P** pont nem más, mint az Alice által kiszámolt ***kG*** pont (aláírási folyamat második lépése).

Alice aláírásának egyik része a következő kifejezés:

$$s = k^{-1}(e + dr) \bmod n$$

Ha az egyenlet mindkét oldalát megszorozzuk $s^{-1}k$ szorzattal, akkor azt kapjuk, hogy

$$k \equiv s^{-1}(e+dr) \equiv s^{-1}e + s^{-1}dr \equiv we + wdr \equiv u_1 + u_2d \pmod n$$

Már csak egy lépés választ el attól, hogy a Bob által kiszámolt **P** pontra belássuk állításunkat:

$$P(x_1,y_1) = u_1G + u_2Q = u_1G + u_2dG = (u_1 + u_2d)G = kG$$

Vagyis, ha Bob a számítás eredményeképpen Alice véletlen pontját kapja vissza, azok x koordinátáinak is meg kell egyezniük. Ha Bob egy másik pontot kap eredményül, az aláírás nem fogadható el.

Pontok, görbék előállítása
Egy-egy ECC-rendszerhez szükség van egy ***E*** görbére, egy **G** bázispontra, véletlen pontokra stb. Az alábbiakban ezek előállítására láthatunk módszereket. Sajnos a görbék és pontok választásánál sok olyan tulajdonságra kell figyelni, amelyekről már volt szó, vagy eddig nem tértem ki rájuk és nem is fogok, azok komplexitása miatt. Ha ezen ismeretek hiányában kívánunk üzleti célú biztonságos implementációt készíteni, a szabványokban javasolt görbékből és bázispontokból válasszunk! Ne feledjük: ezek egyébként is nyilvános paraméterek! Például [**sec**] linken elérhető ajánlásokban 113 bittől 571 bitig találunk paramétereket, [**fedstd**] ajánlásban pedig a szabványosnak tekintett bitméretekre: 112, 128, 160, 192, 224, 256, 384, 521 bit. Úgy vélem, hogy az ECC alapteremtő magyarázatához és megértéséhez nem szükségesek a most „elfelejtett" tulajdonságok, akit pedig érdekel, az Interneten is tengernyi irodalmat találhat. (Személy szerint az IEEE P1363 szabványt ajánlom, én ebben találtam a legtömörebb, legegyszerűbb és „legimplementáció-barátabb" leírásokat.) Kérem a Kedves Olvasót, hogy nézze el ezt a hiányosságot nekem, de csak és kizárólag az ECC matematikai hátteréről több könyvet lehetne írni. És még néhányat a kriptográfiai alkalmazásukról, gyakorlati implementációikról... Csak két példa „elrettentésül":

1. Minden eddig leírt definíciót, szabályt és algoritmust még legalább kétszer le lehetne írni, mert az elliptikus görbéket nem csak a természetes számok (*mod p*) felett lehet értelmezni, hanem polinom-algebra alapján is, amelynek legalább kétféle ábrázolásmódja ismeretes. (Ilyenkor a modulus nem egy prímszám, mint eddig, hanem egy kettőhatvány. Szoftvermegvalósításban általában az előbbi, hardveres megoldásban az utóbbi a gyorsabb.)
2. A kedvelt és szemléletes Descartes-féle koordinátarendszer mellett az elliptikus görbe pontjait a projektív síkon is szokták ábrázolni. Ekkor a végtelenben lévő **O** pont az origóba kerül.

Görbekészítés
A görbék készítésére egyébként legalább háromféle módszer van. Az egyik speciális görbékkel dolgozik, amelyek együtthatói bizonyos szempontoknak megfelelnek: optimális hardvermegvalósítást tesznek lehetővé, vagy különlegesen ellenállóvá teszik a görbét valamelyik támadási forma ellen, stb. A második szerint a görbéket meghatározó együtthatókat véletlenszerűen választjuk meg:

Válasszunk egy prímszámot:
$p = 4294967861$ ($2^{32}+565$)
Majd egy véletlen a paramétert:
$a = 1234567890$
És egy véletlen b paramétert:
$b = 0987654321$
Ellenőrzés következik:
$4a^3 + 27b^2 \bmod p = 752670551349406800391749410 7 \bmod 4294967861 = 1240368550 \neq 0 \rightarrow$ OK!
A kész görbénk egyenlete:
$y^2 \equiv x^3 + 1234567890x + 987654321 \pmod{429467861}$

A harmadik módszer nem véletlen számokat használ, hanem egy kezdőértékből (*seed*) számított SHA-1 érték alapján állítja elő az együtthatókat. Az IEEE P1363 és a NIST egyaránt ezt az eljárást ajánlja az ilyen típusú görbék (*pseudo-random curves*) konstruálásához [**fedstd**]. Az álvéletlen megoldás előnye, hogy reprodukálható és így ellenőrizhető, hogy egy görbe ezzel a módszerrel készült-e vagy sem.

Pontkészítés
Ha már van egy görbénk, azon egy véletlen pontot is kereshetünk. Példaképpen a véletlen számokból készített görbénken keresünk egy pontot:

$y^2 \equiv x^3+1234567890x+987654321 \pmod{429467861}$
$x = 147896325$ (véletlenszerűen választott)
$y^2 \equiv 370713451 \pmod{4294967861}$
$y = ?$

Hát, izé… ennek nincs megoldása, vagyis nincs olyan szám, amelynek négyzetét a modulussal elosztva 370713451-et kapnánk maradékul. Próbáljuk meg újra egy másik x értékkel!

$x = 225589$
$y^2 \equiv 376919525 \pmod{4294967861}$
$y = 57372704$

Na, ezzel megvolnánk: **P**(*225589, 57372704*)! E pontnak van még néhány tulajdonsága, amit jó lenne tudni (például generátor-e? Ha nem, mennyi a rendje?), de ezekkel most nem foglalkozunk (lásd korábban, hogy miért nem).

Üzenet leképzése egy pontra és vissza

Néhány kriptográfiai algoritmus tartalmaz egy olyan lépést, amikor az üzenetet le kell képezni egy olyan alakra, amit az adott algoritmus kezelni tud. Esetünkben ez azt jelenti, hogy egy adott ***E*** görbe alkalmazása mellett Alice **P** ponttá tudja alakítani az *m* üzenetet és Bob egy megoldott pontból ki tudja venni az üzenetet. (Itt jegyzem meg: előfordulhat, hogy csak a pont x koordinátája közlekedik teljes egészében a kommunikációban. Az y-nak csak legnagyobb helyértékű bitje kíséri az x-et, mondván, az y kiszámolható. Ebben az esetben a pontot tömörített pontnak (*compressed point*) hívja az ANSI9.62, az IEEE P1363 és a SEC is. A három forrás legnagyobb különbsége, hogy a SEC csak használja a fogalmat, a többiek el is magyarázzák.)

A példához ismét a korábban készített görbénket fogjuk használni. Első próbálkozásunkkal alakítsuk ponttá a „Jó!" karaktersorozatot! Ehhez először számmá kell alakítani: a karakterek ASCII kódját hexa formában egymás mellé írjuk és egyetlen hexadecimális számként értelmezzük. Más módszer is használhatunk, a lényeg, hogy:

- kölcsönösen egyértelmű megfeleltetés legyen,
- a blokkméretnek kisebbnek kell lennie, mint a modulus hossza!

És ezután mi sem egyszerűbb, ez legyen az üzenetet képviselő **P** pont x koordinátája, csak ki kell számolni a hozzátartozó y-t! Lássunk neki!

m = „Jó!" = 0x4A 0xF3 0x21 = 0x4AF321 = 4911905

P(m,y) = (4911905, ?)

$y^2 \equiv x^3 + 1234567890x + 987654321 \pmod{429467861}$

$y^2 \equiv 43578828$

$y \equiv 165701469$

P(m,y) = (4911905, 165701469)

Természetesen felvetődhet a kérdés, hogy mi van akkor, ha az üzenet alapján nem létezik pont? A pontgenerálás első próbálkozása is ezért volt sikertelen. Mi a teendő akkor, ha y^2-nek nincs megoldása?

m = „Nem" = 0x4E 0x65 0x6D = 0x4E656D = 5137773

P(m,y) = (5137773, ?)

$y^2 \equiv x^3 + 1234567890x + 987654321 \pmod{429467861}$

$y^2 \equiv 109210672$

y ≡ nincs megoldása

Az ilyen esetekre felkészült alkalmazások nem tisztán az *m* üzenetet tekintik az x koordinátának, hanem az x koordináta felső bitjeibe helyezik el azt, majd az alsó bitekkel addig „szórakoznak", amíg eredményre nem jutnak: **P**(*m * eltolás + valami, y*). Ilyenkor az üzenetet kibányászni a (P_x / *eltolás*) egészrészeként lehet. Példánkban a modulus 29 bites, az eltolás legyen 6 bit, így 23 bites üzeneteket tudunk továbbítani. Próbáljuk meg még egyszer a „Nem" szót ponttá alakítani, az eltolás használatával (valami=10, ha nem jutunk eredményre, majd választunk másikat…):

m = „Nem" = 0x4E 0x65 0x6D = 0x4E656D = 5137773

$P(m*2^6+10,y) = (328817482, ?)$

$y^2 \equiv x^3+1234567890x+987654321 \pmod{429467861}$

$y^2 \equiv 275000646$

$y \equiv 125641602$

P(328817482, 125641602)

Ezt a pontot már Alice tetszőleges módon titkosíthatja és elküldheti Bobnak. Bob a dekódolás után szerencsés esetben ezt a pontot fogja visszakapni. Megragadja a pont x koordinátáját, elosztja 2^6-nal és az eredmény egészrésze: 5137773 ! Voilá!

Tényleg biztonságban vagyunk?

A kérdés megválaszolásához a bevezetőben szereplő gondolatokat kell továbbfűznünk az eddigi törési kísérletek és tapasztalatok fényében.

Certicom challenges

Az RSA Inc.-hez hasonlóan a Certicom Corporation [**certicom**] is írt ki törési versenyeket, melyek egy része mára megoldott, másik része még megoldásra vár [**certchall**]. A feladatok 1999 óta – a Certicom szándéka szerint – azt kívánják bizonyítani,

hogy az ECC erősebb, mint az RSA- vagy a DLP-probléma. Másrészt marketingfogás, amely megpróbálja a potenciális ipari felhasználók, fejlesztők és a kutatók figyelmét az ECDLP felé terelni. A versenykiírásban mintegy 20 nyilvános kulcs szerepel, a hozzájuk tartozó rendszerparaméterekkel együtt [**curlist**]. A feladat: meg kell keresni a titkos kulcsot!
A Certicom az alábbi három csoportra osztotta a feladványokat (zárójelben a Certicom által becsült megoldási idők, néhány ezer együttműködő gép esetére):

Excercises:	79 bites	(néhány óra)
	89 bites	(néhány nap)
	97 bites	(néhány hét)
Level I:	109 bites	(néhány hónap)
	131 bites	(néhány hónapnál sokkal több)
Level II:	163 bites	(jelenleg megoldhatatlan feladatok)
	191 bites	
	239 bites	
	359 bites	

Eddigi megoldások néhány technikai adatát tartalmazza a következő felsorolás. (Érdekes, hogy a különböző források kis mértékben ellentmondóak egymásnak, akárcsak az RSA törési versenyek esetében...)
2002. November 6. – ECCp-109 Challenge megoldása (prím modulus): 10300 résztvevő, 10000 számítógép, 1,5 év időtartam;
2000. Április 17. – ECC2K-108 Challenge megoldása (kettőhatvány modulus): 1300 résztvevő, 9500 számítógép, 4 hónap időtartam;
1999. Szeptember 28. – A 97-bites ECC Challenge megoldása: 200 résztvevő, 740 számítógép, 16000 MIPS-év számításigény. Mindez körülbelül fele az ötször hosszabb RSA-512 feltöréséhez használt teljesítménynek.

Pollard-ρ algoritmusa
Napjaink legjobbnak tartott algoritmusa a *Pollard-ρ* algoritmus (Pollard-ró). Az eljárás kis módosítással teljes mértékben párhuzamosítható, így ha 10 processzor áll rendelkezésre, 10-szer gyorsabban jut eredményre. Ha csak egy processzorunk van, $0.5*\sqrt{(\pi*p)}$ EC-összeadás (ahol *p* a modulus) kell a végrehajtásához, ha több, akkor ez a sok számítás megoszlik köztük. Akármilyen jó is az algoritmus, tetemes számításigénye van:

***p* mérete**	**$0.5*\sqrt{(\pi*p)}$**	**MIPS-év**
97 bit	2^{49}	$1{,}6\times10^{4}$
160 bit	2^{80}	$8{,}5\times10^{11}$
186 bit	2^{93}	$7{,}0\times10^{15}$
234 bit	2^{117}	$1{,}2\times10^{23}$
354 bit	2^{177}	$1{,}3\times10^{41}$
426 bit	2^{213}	$9{,}2\times10^{51}$

Összehasonlításul a faktorizálás becsült számításigénye a szokásos modulusok méretére:

modulus mérete	**MIPS év**
512 bit	3×10^{4}
768 bit	2×10^{8}
1024 bit	3×10^{11}
1280 bit	1×10^{14}
1536 bit	3×10^{16}
2048 bit	3×10^{20}

(1 MIPS-év az a számításigény, ami 1 darab 1 MIPS teljesítményű számítógéppel 1 év alatt teljesíthető.)

Az algoritmus további részleteitől és hátterétől most nagyvonalúan tekintsünk el, jelentősen túlmutat a cikk célkitűzésein.

Válasz a kérdésre: nem tudjuk!
Napjaink minden széles körben elterjedt kulcscserére, titkosításra vagy digitális aláírásra használt PKI algoritmusa a faktorizálás vagy a diszkrét logaritmus problémáján nyugszik. A két probléma hasonlít egymáshoz, amit az is jól jelez, hogy a legjobb faktorizáló algoritmusok (bizonyos feltételek teljesülése esetén) felhasználhatók a DLP-problémák megoldásában is. Némi egyszerűsítéssel azt is mondhatjuk, hogy egyező kulcsméret mellett egyező biztonságot nyújtanak.
Sajnos a DLP és az ECDLP már nem hasonlít ennyire egymásra, a viszonylag jó és újabb DLP-megoldó algoritmusok („*index kalkulus*”) egyszerűen nem használhatók ECDLP esetére: ott meg kell elégedni a régebbi módszerekkel (például *Pollard-ρ*). Ebből az is következik, hogy elégséges, ha a kulcsok e régi és lassú „trükköknek” ellenállnak, tehát rövidebbek is

lehetnek. Jelentősen rövidebbek. A minimálisan ajánlott kulcsméret ma 1024 bit az RSA esetében, 163 bit az ECC-rendszerekhez.
Semmi sem garantálja azonban, hogy ez holnap is így lesz. Semmi sem garantálja, hogy a közeljövőben valaki nem publikál egy olyan eljárást, amely valóban jó megoldást nyújt az ECDLP-re. Nem tudjuk, hogy elvileg sem működnek a DLP jó algoritmusai vagy csak a mi ismereteink a hiányosak. Igazság szerint az ECDLP-re ma is léteznek jó algoritmusok, de ezek mindegyike kizárólag valamilyen speciális tulajdonságú görbére és nem általánosan alkalmazható. Jelenlegi tudásunk szerint mindenestre az ECDLP alapú kriptorendszerek jobbak – gyorsabbak és biztonságosabbak –, mint az IFP-n vagy a DLP-n alapuló kriptorendszerek [**menezes**].

Virasztó Tamás
wacher@westel900.net