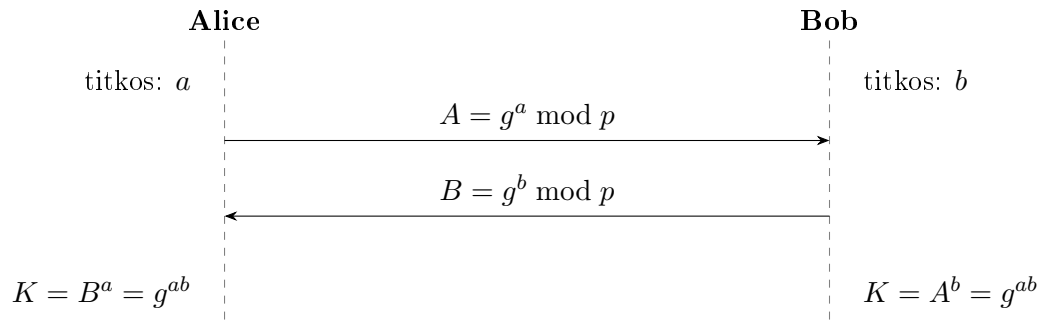


1. ábra: A Diffie–Hellman kulcscsere futása. A csatornán csak A és B halad át; a közös $K = g^{ab}$ titkot mindkét fél a saját titkos kitevőjével számolja ki.

5 Titkosítások feltörése

A legtöbb klasszikus titkosítási algoritmus biztoságát az adja hogy egy meglehetősen nagy véges test fölött működik. Nyilván ha a test összes elemét le tudjuk tárolni a memóriánkban és csinálunk egy asszociatív tömböt amibe az összes privát kulcs-publikus kulcs értéket gyorsan kereshető formában tároljuk, akkor mindenhatók vagyunk és ellenünk nem lehet biztonságos kriptorendszert tervezni. Hasonló a helyzet ha az értékeket nem tároljuk, de nagyon gyorsan ki tudjuk számolni. Igen ám, de ha csak egy 256 bites ECC kulcsot veszünk, akkor a tér mérete 2^{256} nagyságú, ami exponenciálisan sok memóriát vagy számítást (vagy a kettő szorzatát, erről majd később) igényli, ez pedig bonyolultságelméleti szempontból nagyon nehéznek tekintett. Az ilyen jellegű támadásokat hívják brute-force (kimerítő) támadásoknak, amelynek során valaki kipróbálja az összes lehetséges értéket. Nyilvánvaló tehát hogy a kulcs méretét olyan nagyra kell választani hogy a jelenlegi (és lehetőleg a jövőbeli) hardwareken ez lehetetlen legyen. Tovább bonyolítja a helyzetet, hogy a probléma nem univerzálisan redukálódik a kimerítő keresésre, hiszen minden titkosítási algoritmust valamint digitális aláírást külön kell vizsgálni. Gyakran a DLP probléma és a hamisított aláírás generálása például nem ekvivalens. Egy további példa lehet mondjuk az is hogy RSA aláírást hamisítani nem biztos hogy annyira nehéz mint megtalálni egy nagy szám prím osztóit, vagy eldönteni egy számról hogy prím szám-e, vagy hogy a prím osztói nagyok-e. Sok problémáról kiderült hogy nem ugyanabba a bonyolultsági osztályba esnek mint a prímtényezős felbontás vagy a DLP és sokról joggal hihetjük azt hogy ezek egyszerűbb problémák mint az előbb említett kettő. Ez azt jelenti tehát hogy elképzelhető, hogy holnap felkelünk és kiderül, hogy valaki előállt egy algoritmussal ami ügyesen tud ECDSA [7] aláírást hamisítani anélkül, hogy magát az alapul szolgáló elliptikus görbe diszkrét logaritmus problémát (ECDLP) megoldaná – az aláírás-hamisítás és az azt megalapozó matematikai probléma ugyanis nem feltétlenül ekvivalens. Jelenleg nagyon kevés eredmény ismert a digitális aláírások és a titkosítási algoritmusok bonyolultságelméleti hovatartozásáról. További probléma lehet az is, hogy nem minden eredmény publikus. Lehet hogy a mai publikált eredmények alapján azt hisszük, hogy a 2048 vagy a 4096 bites RSA kulcsok biztonságosak, mert a leghatékonyabb publikus algoritmusok ezeket a kulcsméreteket képtelenek kezelni, de mi a helyzet a nem publikált algoritmusokkal? Erre ugye nem tudunk válaszolni, a nemzeti kutatóintézetek (amerikai, kínai, orosz, stb.) nem feltétlenül publikálnak minden eredményt, sőt vannak olyan titkos katonai kapcsolódású intézetek ahol szigorúan tilos publikálni ezeket az eredményeket. Ezen okok miatt jó lenne tehát olyan kriptorendszereket használni, amelyek biztonsága nem függ az alkalmazott kulcsok bithosszától. A kriptográfiának van egyébként ilyen

iránya, de sajnos az derül ki hogy ezek az algoritmusok nem túl hatékonyak és a valós rendszerekben nem lehet őket megfelelően implementálni. Emiatt a mai titkosítási algoritmusok és digitális aláírások mind olyanok, amelyeknél a kulcsméret igen is számít biztonsági szempontból.

6 Számelmélet és a számítógép

Az első írásos emlékek, amelyekben a prímszám problémáról olvashatunk, nagyjából 2000 évre mennek vissza. Ekkor jelent meg Eratosztenész szitája, ami megoldást adott a prímszámok megtalálására (és ezáltal próbaosztással a prímtényezős felbontásra is). Eratosztenész (kb. i. e. 276–194) görög matematikus, földrajztudós, csillagász és könyvtáros volt, egy ideig az alexandriai könyvtár vezetője – ami akkoriban a hellenisztikus világ egyik legfontosabb tudományos központja volt.

6.1 Eratosztenész szitája

Az algoritmus egy adott n-ig megtalálja az összes prímszámot az alábbi lépésekkel:

1. Írjuk fel a $2, 3, 4, \ldots, n$ számokat.
2. Legyen $p = 2$ (a legkisebb prím).
3. Húzzuk ki p összes valódi többszörösét: $2p, 3p, 4p, \ldots$ (amelyek $\leq n$).
4. Keressük meg a következő ki nem húzott számot p után, ez lesz az új p.
5. Ha $p^2 > n$, álljunk meg – a megmaradt (ki nem húzott) számok mind prímek.
6. Egyébként ismételjük a 3. lépéstől.

Az algoritmus futásideje $O(n \log \log n)$, ami rendkívül hatékony kis n értékekre. Fibonacci rámutatott, hogy a szitát elég csak $\sqrt{n}$-ig futtatni, hiszen ha egy $m \leq n$ összetett számnak nincs $\sqrt{n}$-nél kisebb prímosztója, akkor m maga prím.

Ezek után a számelméleti kutatások pihentek néhány száz évet. Ma már ez közel triviális, de látszik hogy akkoriban meglehetősen sok idő kellet az ötletek beérésének. A következő eredmények 1800 környékén keletkeztek Gauss asztalán, amikor is leírta hogy valószínűleg a prímtényezős felbontás és a prím tesztelés nem ugyanabba a bonyolultsági osztályba tartoznak. Végül kiderült hogy igaza volt. Utána még Euler, Legendre, Dirichlet és Riemann is sokat foglalkozott a problémával és jelentőset is alkottak a területen. Ebben az időben kezdett el fejlődni a számításos számelmélet (computational number theory), amelynek segítségével sok matematikai sejtést tudtak tesztelni számítógépek segítségével, ez jelentősen megkönnyítette a matematikusok munkáját. Ekkor tevékenykedett Gödel, Turing, Kleene és Church is. 1960 és 1970 környékén születtek meg azok a bonyolultságelméleti eredmények, amelyekben a polinomiális idő, NP és hasonló fogalmak már formalizálva voltak. Fontos megjegyezni hogy 1973-ig a leghatékonyabb prímtesztelő algoritmus és prímtényezős felbontás még továbbra is Fibonacci eredménye volt. Az ez után következő algoritmusok hatékonysága főleg annak köszönhető hogy megjelent a bonyolultságelmélet és a matematikusok elkezdték számolni az eljárások lépésszámát és innentől kezdve a matematikusok nem feltétlenül a futásidőre koncentráltak hanem az algoritmusok bonyolultságára. A bonyolultságelmélet egy teljesen más megközelítést hozott be a matematikába. A

számelmélettel foglalkozó matematikusok (pl. Vaughan Pratt) sokszor hoztak ki számukra teljesen triviálisnak tűnő eredményeket, amiket ha bonyolultságelméleti szemüveggel nézünk akkor formabontók voltak a maguk nemében. Ekkoriban jelent meg és terjedt el a véletlen fogalma az algoritmikus számelméletben, ekkor született meg a random polinom futásidejű problémák osztálya (gondoljuk például a véletlenszerű prímtesztelő algoritmusokra). 1977-re amikor megjelent az RSA algoritmus a matematikusok már tisztán látták hogy eldönteni egy számról hogy prím-e könnyű, egy számnak megkeresni a prím osztóit viszont nehéz. A két probléma tehát elvált egymástól bonyolultságelméleti szinten. 1980-ban Adleman, Pomerance és Rumely előálltak egy majdnem polinom idejű determinisztikus prímtesztelő algoritmussal [9]. Az APR-teszt az egyik első „modern" általános prímteszt volt: nem használ véletlent, és ciklotomikus testek valamint Jacobi-összegek aritmetikájára épül. 1986-ban Goldwasser és Kilian újra elővették Pratt ötletét és egy elliptikus görbe alapú prímtesztelőt javasoltak [10], amellyel megmutatták hogy a prímek majdnem mindegyikére polinomiális várható időben találunk tanúsítványt – formálisan az n-bites prímek között a kivételek aránya aszimptotikusan eltűnik. A szépséghiba annyi volt, hogy egy be nem bizonyított feltételezést használtak. Ezek után Adleman és Huang [11] a Goldwasser-Kilian megközelítést általánosították: elliptikus görbék helyett magasabb dimenziós abeli varietásokat használtak, amivel az „almost all primes" korlátot sikerült eltüntetni, és feltétel nélküli randomizált polinomidejű prímtesztelést értek el minden prímre (RP). A végső áttörést Agrawal, Kayal és Saxena hozta 2002-ben [12], akik bebizonyították hogy a prímtesztelés P-ben van, azaz létezik determinisztikus polinomidejű algoritmus a problémára. Elméletileg ez óriási áttörés, hiszen lezárta a prímtesztelés bonyolultsági besorolásának évtizedes kérdését. A gyakorlatban azonban az AKS viszonylag nagy konstansai miatt a valós prímbizonyításra továbbra is az ECPP (Elliptic Curve Primality Proving) [14] vagy valamilyen APR-CL/Miller-Rabin kombináció a gyorsabb. Ezért a kriptográfiai alkalmazásokban jellemzően a probabilisztikus Miller-Rabin tesztet [15] használják, amely ugyan nem ad matematikai bizonyítékot, de kellően sok iterációval a tévedés valószínűsége elhanyagolhatóvá válik. Látjuk tehát hogy a prímteszteléssel egész jól haladtak a matematikusok, nem ez volt a helyzet viszont a prímtényezős felbontással. Gauss, Legendre, Seelhoff, Kraichik, Lehmer, Powers majd Morrison és Brillhart 1975-ben sokat dolgoztak a problémán, de kevés eredmény látott napvilágot. Ami biztosan kiderült, hogy az egyik legfontosabb tulajdonság, amit a prímtényezős felbontás keresésénél lehet használni az a szám simasága, vagyis hogy mennyi kicsi prím osztója van. Vegyük észre hogy az RSA esetében szándékosan két hatalmas prímet szorzunk össze, ez nyilván nem véletlen. Jelenleg az egyik leghatékonyabb módszer a General Number Field Sieve (GNFS) [16], amely Pollard eredeti ötletén alapul és Lenstra, Buhler, Pomerance és mások fejlesztették tovább, ennek a futásideje szubexponenciális. Persze ha változtatunk az architektúrán, akkor ezen lehet javítani. Egy néhány évtizede látott napvilágot Shor algoritmusa [13], amely polinom időben állítja elő egy összetett szám prím osztóit kvantumszámítógépen.

7 Schnorr aláírás

1989-ben Claus-Peter Schnorr német matematikus egy olyan digitális aláírás algoritmust mutatott be a CRYPTO '89 konferencián (a folyóiratcikk 1991-ben jelent meg [17]), amely jóval egyszerűbb és hatékonyabb volt mint az akkoriban széles körben elterjedt RSA alapú megoldások. Schnorr motivációja abból indult, hogy az RSA nagy kulcsmérettel dolgozott. Az akkori hardvereken – különösen chipkártyákon és beágyazott rendszereken – gyakorlatilag implementálhatatlanná tette az RSA aláírást. Schnorr felismerte, hogy a

diszkrét logaritmus probléma nehézségére építve lehet olyan aláírási sémát konstruálni, amely lényegesen rövidebb kulcsmérettel működik és bizonyíthatóan biztonságos (erről lent).

7.1 Az algoritmus alapjai

A Schnorr aláírás a diszkrét logaritmus problémán (DLP) alapul, egy $\mathbb{Z}_p^*$ multiplikatív csoportban, ahol p egy nagy prímszám. A Schnorr aláírást három primitív definiálja: kulcsgenerálás, aláírás számolás és aláírás ellenőrzés.

7.2 Kulcsgenerálás

1. Válasszunk egy nagy p prímszámot és egy q prímet úgy, hogy q osztója legyen $(p-1)$-nek, azaz $(p-1)$ maradék nélkül osztható q-val.
2. Válasszunk egy g elemet amelynek rendje q a $\mathbb{Z}_p^*$ csoportban, azaz $g^q \equiv 1 \pmod{p}$ és $g \neq 1$.
3. A titkos kulcs egy véletlenszerűen választott x egész, ahol $0 < x < q$.
4. A publikus kulcs: $y = g^x \bmod p$.

A nyilvános protokoll paraméterek: (p, q, g, y), a titkos kulcs pedig x.

7.3 Aláírás generálás

Adott egy m üzenet, amelyet alá szeretnénk írni:

1. Válasszunk egy véletlenszerű k értéket, ahol $0 < k < q$ (ezt általában noncenak hívják).
2. Számítsuk ki a commitmentet: $r = g^k \bmod p$.
3. Számítsuk ki a challenget: $e = H(r \| m)$, ahol H egy kriptográfiai hash függvény (pl. SHA-256) és $\|$ a stringösszefűzés.
4. Számítsuk ki a választ (response): $s = k - x \cdot e \bmod q$.

Ekkor a Schnorr aláírás az (e, s) pár.

7.4 Fiat–Shamir heurisztika és a “challenge” elnevezés eredete

Felmerülhet a kérdés, hogy miért hívjuk az e értéket “kihívásnak” (challenge), ha az aláíró maga számolja ki. Ennek megértéséhez tudni kell, hogy a Schnorr aláírás eredetileg egy három lépéses interaktív azonosítási protokoll volt:

1. Az aláíró (bizonyító) elküldi $r = g^k \bmod p$-t az ellenőrzőnek (commitment).
2. Az ellenőrző visszaküld egy véletlenszerű e értéket (challenge – kihívás).
3. Az aláíró válaszol: $s = k - x \cdot e \bmod q$ (response).

Ebben a felállásban az e tényleg a másik féltől jön, ezért hívjuk kihívásnak. Az ellenőrző ezzel teszteli hogy a bizonyító valóban ismeri-e a titkos kulcsot. Mivel a challenge véletlenszerű a bizonyító nem tudja előre kiszámolni a helyes választ anélkül hogy valóban ismerné x-et.

Ezzel viszont az a probléma hogy a protokoll interaktív, tehát a két félnek valós időben kommunikálnia kell egymással. Amikor digitális aláírást akarunk ebből csinálni, akkor más megközelítés kell (pl. egy e-mailt nem tudsz interaktívan aláírni).

1986-ban Amos Fiat és Adi Shamir [18] megmutatták, hogy bármilyen ún. Sigma-protokollt (Σ-protokoll) nem-interaktívvá lehet alakítani úgy hogy az ellenőrző véletlenszerű kihívását egy kriptográfiai hash függvénnyel helyettesítjük: $e = H(r\|m)$. A Sigma-protokoll egy három lépéses (three-move) interaktív bizonyítási séma: (1) commitment, (2) challenge, (3) response – pontosan olyan mint a fenti Schnorr azonosítási protokoll. Ezek a protokollok public-coin tulajdonságúak, ami azt jelenti hogy az ellenőrző üzenete (a challenge) véletlenszerűen választott érték. Az ellenőrzőnek nincs semmiféle stratégiája vagy titka, egyszerűen dob egy érmét (vagy többet, amivel a biteket beállítja) és elküldi az eredményt. Éppen ez teszi lehetővé a hash függvénnyel való helyettesítést. Ha az ellenőrző úgyis csak véletlenszámot küld, akkor a hash függvény kiszámíthatatlan kimenete tökéletesen helyettesíti. Mivel az aláíró nem tudja befolyásolni a hash függvény kimenetét (nem tudja úgy megválasztani r-t, hogy egy számára kedvező e értéket kapjon, amivel a titkos kulcs ismerete nélkül is érvényes aláírást tudna gyártani), a biztonságossági tulajdonság megmarad. Ez a Fiat–Shamir heurisztika, és ez teszi lehetővé hogy a Schnorr protokollból digitális aláírás séma legyen. Itt az aláíró saját magának gyártja le a véletlen értéket egy hash függvény segítségével.

7.5 Az aláírás ellenőrzése

Tegyük fel hogy az ellenőrző fél megkapja az (e, s) aláírást és az m üzenetet, valamint ismeri a publikus paramétereket (p, q, g, y). Ekkor a fél számoljon a következő módon:

1. $r' = g^s \cdot y^e \bmod p$.
2. $e' = H(r'\|m)$.
3. Ekkor, az aláírás érvényes, ha $e' = e$.

Az ellenőrzés azért helyes, mert:

$$r' = g^s \cdot y^e = g^{k-xe} \cdot g^{xe} = g^k = r \pmod{p}$$

Tehát ha az aláírás helyes, akkor $r' = r$, amiből $e' = H(r'\|m) = H(r\|m) = e$ következik.

7.6 Biztonság és annak korlátai

A Schnorr aláírás biztonsága a DLP nehézségén alapul, azaz a publikus kulcs $y = g^x \bmod p$ ismeretében x kiszámítása számítástechnikailag lehetetlen megfelelő paraméterválasztás mellett. Schnorr bizonyította [17], hogy az aláírási séma biztonságos az ún. random oracle modellben [19], azaz ha a hash függvényt idealizált véletlenszerű orákulum-ként kezeljük, akkor az aláírás hamisítása legalább olyan nehéz mint a DLP megoldása. Ezt a redukciót Pointcheval és Stern formalizálták [20]. Megmutatták, hogy ha létezik hatékony algoritmus amely képes Schnorr aláírást hamisítani a random oracle modellben, akkor abból konstruálható egy hatékony algoritmus amely megoldja a DLP-t. Fontos megjegyezni,

hogy ez a redukció nem szoros. A konstrukció során hatékonyságot veszítünk, ami azt jelenti hogy a gyakorlatban valamivel nagyobb paramétereket kell választani mint amit a DLP nehézsége indokolna. Egyes kutatók szerint a prefaktor annyira nagy, hogy a redukció gyakorlati értelme megkérdőjelezhető. Ez azt jelenti hogy ha a támadó az aláírást ε valószínűséggel hamisítja, a DLP-megoldó csak ε^2/q_H valószínűséggel működik (ahol q_H a hash lekérdezések száma). Így tehát a redukcióból kapott DLP-megoldó annyi lépést igényel, amennyiből közvetlen módszerekkel is meg lehetne oldani a DLP-t. Ennek ellenére a Schnorr aláírás gyakorlati biztonsága évtizedek óta megkérdőjelezetlen, csak a formális bizonyítás ereje vitatott. Fontos megjegyezni, hogy a k nonce értéknek minden egyes aláírásnál frissen és véletlenszerűen kell választódnia. Ha valaki kétszer ugyanazt a k-t használja két különböző üzenethez, abból a titkos kulcs triviálisan kiszámolható.

8 DSA és ECDSA

1991-ben az Egyesült Államok Nemzeti Szabványügyi és Technológiai Intézete (NIST) egy új digitális aláírás szabványt akart kidolgoztatni. Ezt szabadon akarták használni kormányzati és kereskedelmi célokra. A probléma az volt, hogy az RSA aláírás a Rivest, Shamir és Adleman cég szabadalma alatt állt, a Schnorr aláírás pedig szintén szabadalmaztatva volt. David W. Kravitz, az NSA (National Security Agency) munkatársa 1991-ben javasolta a Digital Signature Algorithm-ot (DSA) [21], amelyet a NIST 1994-ben FIPS 186 néven szabványosított. A DSA szándékosan úgy lett tervezve hogy kikerülje a Schnorr-szabadalmat, de hasonló matematikai alapokon nyugszik (DLP, $\mathbb{Z}_p^*$ csoport, q-rendű alcsoport). A kriptográfiai közösség egy része kritizálta a DSA-t, mondván hogy lényegében a Schnorr séma egy átdolgozott változata, de a NIST végül elfogadta és évtizedekig ez volt az amerikai kormányzati szabvány.

8.1 DSA kulcsgenerálás

1. Válasszunk egy q prímet (tipikusan 256 bites) és egy p prímet (tipikusan 2048–3072 bites) úgy, hogy q osztója legyen $(p-1)$-nek.
2. Válasszunk egy g elemet amelynek rendje q a $\mathbb{Z}_p^*$ csoportban: $g = h^{(p-1)/q} \bmod p$ valamely $h > 1$-re, ahol $g \neq 1$.
3. A titkos kulcs: véletlenszerű x, ahol $0 < x < q$.
4. A publikus kulcs: $y = g^x \bmod p$.

8.2 DSA aláírás generálás

Adott egy m üzenet:

1. Válasszunk véletlenszerű k-t, ahol $0 < k < q$.
2. Számítsuk ki: $r = (g^k \bmod p) \bmod q$.
3. Számítsuk ki: $s = k^{-1} \cdot (H(m) + x \cdot r) \bmod q$, ahol H egy hash függvény (pl. SHA-256).
4. Az aláírás: (r, s).

8.3 DSA aláírás ellenőrzés

1. Számítsuk ki: $w = s^{-1} \bmod q$.
2. Számítsuk ki: $u_1 = H(m) \cdot w \bmod q$ és $u_2 = r \cdot w \bmod q$.
3. Számítsuk ki: $v = (g^{u_1} \cdot y^{u_2} \bmod p) \bmod q$.
4. Az aláírás érvényes, ha $v = r$.

8.4 DSA és Schnorr összehasonlítás

A DSA és a Schnorr aláírás közötti fő különbség nem matematikai, hanem strukturális: a DSA-ban az r értéket a csoport elemből egy extra mod q lépéssel nyerik ki, és az s kiszámítása is eltér (inverz szorzás a Schnorr-féle egyszerű kivonás helyett). Ezek a módosítások éppen elegendők voltak ahhoz, hogy a NIST szabadalmijogi szempontból új algoritmusként tudja bemutatni a DSA-t. Biztonsági szempontból mindkettő a DLP nehézségére épül és hasonló paramétereket igényel.

8.5 Elliptikus görbék röviden

Az elliptikus görbék megértéséhez érdemes egy általánosabb fogalomból kiindulni: az algebrai varietásból. Egy algebrai varietás nem más mint egy polinom-egyenletrendszer megoldásainak halmaza. Például:

- Egy egyenes ($y = 2x + 1$) egy 1-dimenziós varietás (görbe).
- Egy kör ($x^2 + y^2 = 1$) szintén egy 1-dimenziós varietás.
- Egy gömb felülete ($x^2 + y^2 + z^2 = 1$) egy 2-dimenziós varietás (felület).

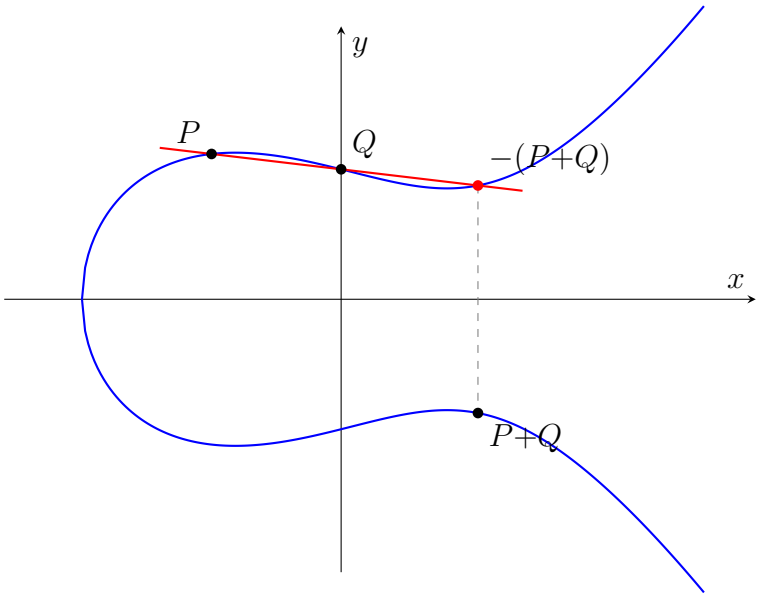
Általánosan: ha adottak $f_1, \ldots, f_m$ polinomok, akkor a $V = \{(x_1, \ldots, x_n) : f_i(x_1, \ldots, x_n) = 0,\ \forall i\}$ halmaz egy algebrai varietás. A varietás dimenziója határozza meg a geometriai jellegét: 0-dimenziós varietás pontok halmaza, 1-dimenziós varietás egy görbe, 2-dimenziós egy felület, és így tovább. A varietás genusza egy topológiai invariáns, ami a görbe "bonyolultságát" méri – informálisan a felületén lévő "lyukak" számát. Egy egyenesnek vagy kúpszeletnek (kör, ellipszis, hiperbola, parabola) a genusza 0, egy elliptikus görbének a genusza 1 (egy "lyuk", mint egy fánk felülete).

Az elliptikus görbék tehát az 1-genuszú sima algebrai görbék – ez az a speciális eset, amelyen természetes módon definiálható egy csoportművelet (pontösszeadás). Magasabb genuszú görbéken (genusz ≥ 2) és magasabb dimenziós ún. abeli varietásokon is definiálhatók hasonló csoportstruktúrák (pl. a Jacobian-csoport), amelyeket a kriptográfiában szintén vizsgálnak [24], de a gyakorlatban az elliptikus görbék (genusz 1) bizonyultak a legjobb kompromisszumnak a biztonság és a számítási hatékonyság között. Aki mélyebben szeretne megismerkedni az algebrai varietásokkal és az elliptikus görbék elméletével, annak ajánlott Silverman [25] és Washington [26] könyve.

Egy elliptikus görbét a következő egyenlet írja le (ún. rövid Weierstrass-forma):

$$y^2 = x^3 + ax + b$$

ahol a, b a görbe paraméterei, és a diszkrimináns $4a^3 + 27b^2 \neq 0$ (ez garantálja, hogy a görbének nincsenek szinguláris pontjai). A görbe pontjai – kiegészítve egy speciális



2. ábra: Két különböző pont összeadása a $y^2 = x^3 - 2x + 4$ görbén. A P-n és Q-n átmenő szelő (piros) a görbét egy harmadik pontban metszi; ezt az x-tengelyre tükrözve kapjuk $P + Q$-t.

"végtelenben lévő ponttal" ($\mathcal{O}$, az egységelem) – egy Ábel-csoportot alkotnak a pontösszeadás művelettel. A pontösszeadás geometriailag úgy működik, hogy amikor két pont összegét szeretnénk kiszámolni, akkor a rajtuk átmenő egyenes és a görbe harmadik metszéspontjának tükrözésével számolunk egy pontot és ezt hívjuk a két kiindulási pont összegének.

A pontösszeadás képletei. Legyen $P = (x_1, y_1)$ és $Q = (x_2, y_2)$ a görbe két pontja, és jelölje $P + Q = (x_3, y_3)$ az összegüket. A geometriai konstrukció (P-n és Q-n átmenő egyenes harmadik metszéspontjának x-tengelyre tükrözése) az alábbi algebrai képleteket adja. Két különböző pont összeadásakor ($P \neq \pm Q$) az összekötő egyenes meredeksége

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1},$$

a pontkétszerezésnél ($P = Q$, azaz $2P$ számítása) pedig a görbéhez P-ben húzott érintő meredekségét kell venni, amit az $y^2 = x^3 + ax + b$ implicit deriválásából kapunk:

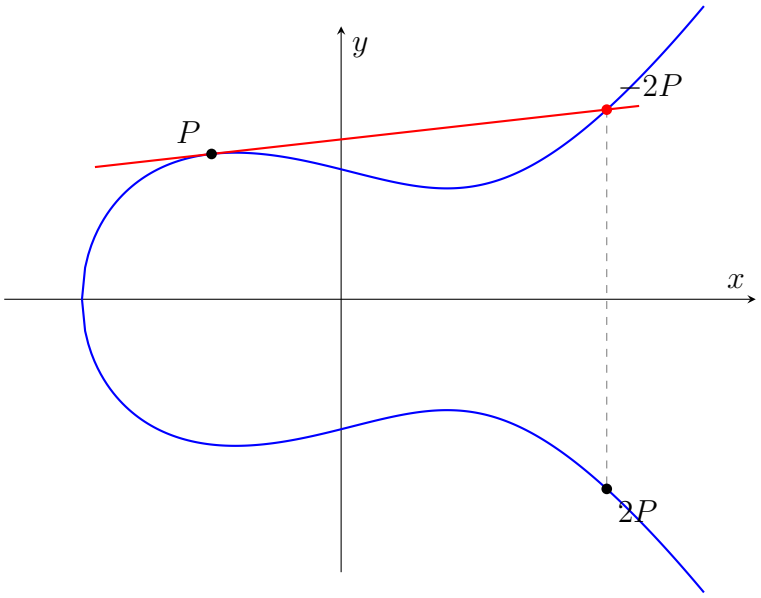
$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

Mindkét esetben az eredmény koordinátái:

$$x_3 = \lambda^2 - x_1 - x_2, \qquad y_3 = \lambda(x_1 - x_3) - y_1.$$

(Pontkétszerezésnél $x_2 = x_1$ helyettesítendő.) Speciális esetként, ha $P = -Q$ (azaz $x_1 = x_2$ és $y_1 = -y_2$), akkor az összekötő egyenes függőleges, nincs harmadik affin metszéspont, és az összeg a végtelenben lévő $\mathcal{O}$ egységelem.

A fenti geometriai konstrukciót a valós számok felett, a $y^2 = x^3 - 2x + 4$ görbén szemléltetjük (ábra 2 és ábra 3); ezek az ábrák a szelő–érintő szabály intuícióját adják. A valós számok felett azonban a koordináták szinte sosem egészek: már a $\lambda = (y_2 - y_1)/(x_2 -$



3. ábra: Pontkétszerezés ($2P$) a $y^2 = x^3 - 2x + 4$ görbén. A P pontban húzott érintő (piros) a görbét egy másik pontban metszi; ezt az x-tengelyre tükrözve kapjuk $2P$-t.

x_1) hányados és az azt követő négyzetre emelés is jellemzően irracionális értékeket ad (pl. a $P = (-1, \sqrt{5})$ pontból kiindulva $\sqrt{5}$ végigöröklődik a számításon).

A kriptográfiában viszont a görbét nem a valós számok, hanem egy véges test ($\mathbb{F}_p$) felett tekintjük, ahol a koordináták eleve a $\{0, 1, \ldots, p-1\}$ egészek, és minden művelet $\bmod p$ értendő – így az összeadás eredménye is mindig egész. (A szemléltető ábrák ilyenkor már nem rajzolhatók folytonos görbeként, a pontok egy szabálytalan rácson helyezkednek el, de az algebrai képletek változatlanul érvényesek.) Tekintsük példaként a $y^2 = x^3 + 2x + 2$ görbét az $\mathbb{F}_{17}$ test felett, és legyen $P = (6, 3)$, $Q = (0, 6)$. A moduláris osztást a $\bmod 17$ multiplikatív inverzzel végezzük. Két különböző pont összeadására:

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{6-3}{0-6} = 3 \cdot (-6)^{-1} \equiv 8 \pmod{17},$$

$$x_3 = \lambda^2 - x_1 - x_2 \equiv 8^2 - 6 - 0 \equiv 7, \qquad y_3 = \lambda(x_1 - x_3) - y_1 \equiv 8(6-7) - 3 \equiv 6 \pmod{17},$$

tehát $P + Q = (7, 6)$. A pontkétszerezésnél az érintő meredeksége

$$\lambda = \frac{3x_1^2 + a}{2y_1} = \frac{3 \cdot 36 + 2}{2 \cdot 3} = 110 \cdot 6^{-1} \equiv 7 \pmod{17},$$

amiből $x_3 \equiv 7^2 - 2 \cdot 6 \equiv 3$ és $y_3 \equiv 7(6-3) - 3 \equiv 1 \pmod{17}$, vagyis $2P = (3, 1)$. Minden koordináta egész, ahogy azt a véges test felett elvárjuk.

A véges test felett az elliptikus görbe pontjai véges csoportot alkotnak, és ezen a csoporton definiálható a DLP. Ha tehát adott P és $Q = nP$ (ahol nP az n-szeres pontösszeadás), az n megtalálása számítástechnikailag nehéz. Fontos még, hogy az elliptikus görbe DLP-re nem ismert szubexponenciális algoritmus (szemben a $\mathbb{Z}_p^*$-beli DLP-vel, ahol az index calculus [23] működik), ezért elvileg azonos biztonsági szinthez lényegesen kisebb kulcsméretek elegendők.