Elliptic Curves and Cryptography

Background in Elliptic Curves

We'll now turn to the fascinating theory of elliptic curves. For simplicity, we'll restrict our discussion to elliptic curves over Z_p, where *p* is a prime greater than 3. Keep in mind, though, that elliptic curves can more generally be defined over any finite field. In particular, the "characteristic two finite fields" $\mathbb{F}_2{}^{m}$ are of special interest since they lead to the most efficient implementation of the elliptic curve arithmetic.

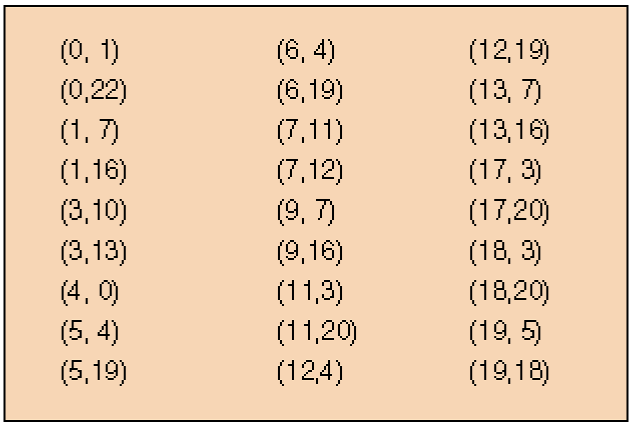
An elliptic curve *E* over Z_p is defined by an equation of the form

$$(1) \qquad y^2 = x^3 + ax + b,$$

where $a,b \in Z_p$, and $4a^3 + 27b^2 \neq 0 \pmod{p}$, together with a special point O, called the "point at infinity". The set $E(Z_p)$ consists of all points (*x*, *y*), $x \in Z_p$, $y \in Z_p$, which satisfy the equation in equation (1), together with O.

For example, let p =23 and consider the elliptic curve E: y^2=x^3+x+1 defined over Z_{23}. (In the notation of equation (1), we have a =1 and b =1.) Note that $4a^3 + 27b^2 = 4+4 = 8 \neq 0$, so E is indeed an elliptic curve. The points in $E(Z_p)$ are O and those listed in Figure 1.

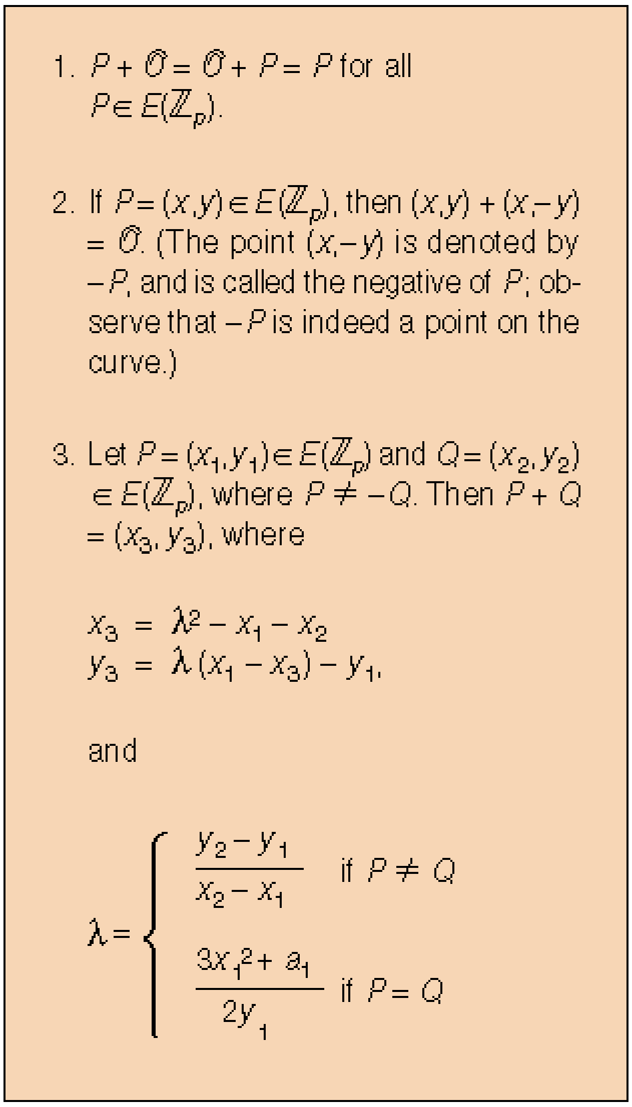
Figure 1: Points on the elliptic curve $y^2 = x^3+x+1$ over Z_{23}.



(0, 1)	(6, 4)	(12,19)
(0,22)	(6,19)	(13, 7)
(1, 7)	(7,11)	(13,16)
(1,16)	(7,12)	(17, 3)
(3,10)	(9, 7)	(17,20)
(3,13)	(9,16)	(18, 3)
(4, 0)	(11,3)	(18,20)
(5, 4)	(11,20)	(19, 5)
(5,19)	(12,4)	(19,18)

There is a rule for adding two points on an elliptic curve $E(Z_p)$ to give a third elliptic curve point. Together with this addition operation, the set of points $E(Z_p)$ forms a group with O serving as its identity. It is this group that is used in the construction of elliptic curve cryptosystems. The addition rule, which can be explained geometrically, is presented in Figure 2 as a sequence of algebraic formulae.

Figure 2: The addition rule.



1. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{Z}_p)$.

2. If $P = (x,y) \in E(\mathbb{Z}_p)$, then $(x,y) + (x,-y) = \mathcal{O}$. (The point $(x,-y)$ is denoted by $-P$, and is called the negative of P; observe that $-P$ is indeed a point on the curve.)

3. Let $P = (x_1, y_1) \in E(\mathbb{Z}_p)$ and $Q = (x_2, y_2) \in E(\mathbb{Z}_p)$, where $P \neq -Q$. Then $P + Q = (x_3, y_3)$, where

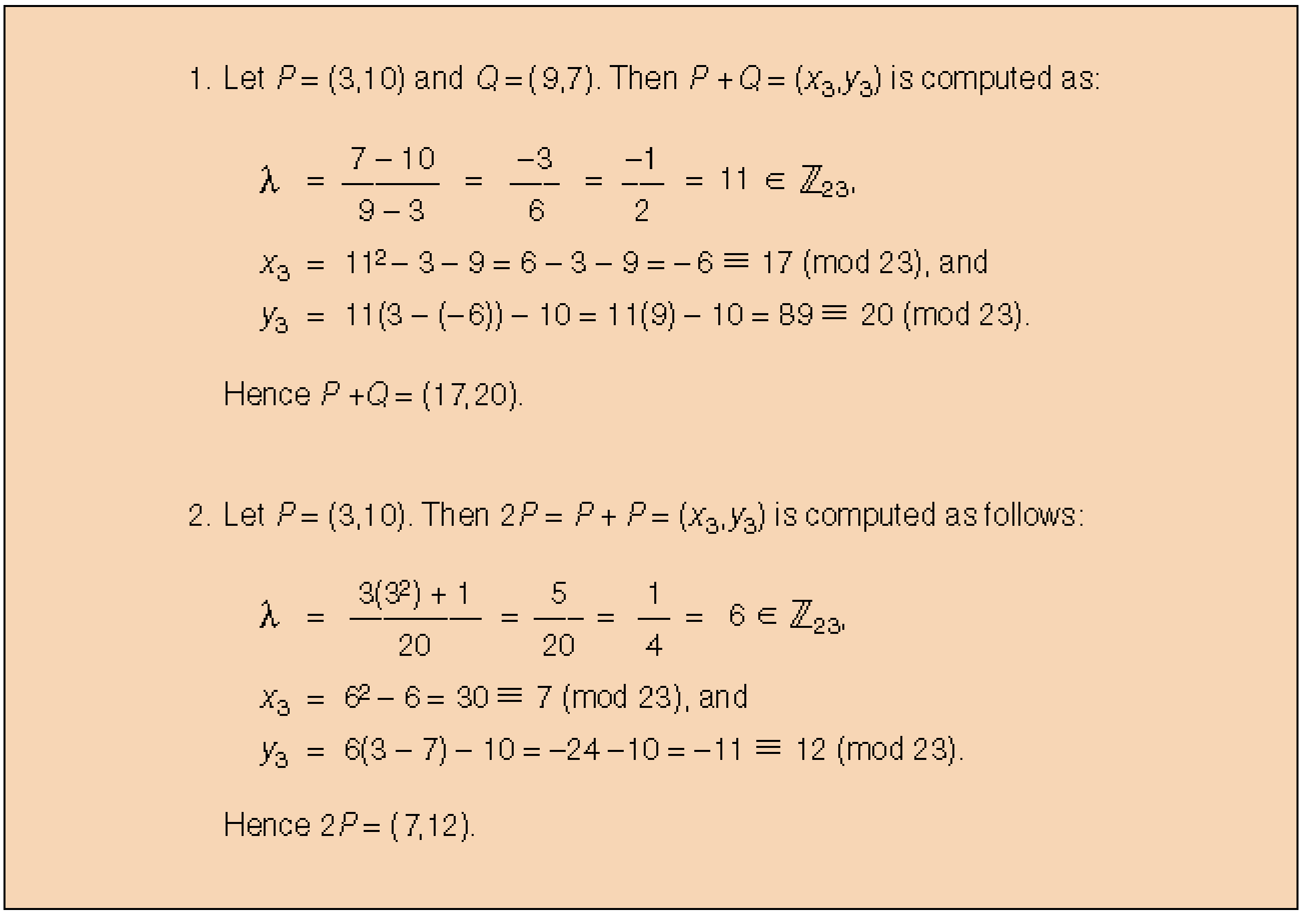
$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1, \end{aligned}$$

and

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{if } P \neq Q \\[2ex] \dfrac{3x_1^2 + a_1}{2y_1} & \text{if } P = Q \end{cases}$$

Examples of the addition rule are given in Figure 3. For historical reasons, the group operation for an elliptic curve $E(Z_p)$ has been called "addition." By contrast, the group operation in Z_p^* is "multiplication." The differences in the resulting additive notation and multiplicative notation can sometimes be confusing. Table 1 shows the correspondence between notation used for the two groups Z_p^* and $E(Z_p)$.

Figure 3: Examples of elliptic curve addition on the curve $y^2=x^3+x+1$ over Z_{23}.



1. Let $P = (3,10)$ and $Q = (9,7)$. Then $P + Q = (x_3, y_3)$ is computed as:

$$\lambda = \frac{7-10}{9-3} = \frac{-3}{6} = \frac{-1}{2} = 11 \in \mathbb{Z}_{23},$$

$$x_3 = 11^2 - 3 - 9 = 6 - 3 - 9 = -6 \equiv 17 \pmod{23}, \text{ and}$$

$$y_3 = 11(3-(-6)) - 10 = 11(9) - 10 = 89 \equiv 20 \pmod{23}.$$

Hence $P + Q = (17, 20)$.

2. Let $P = (3,10)$. Then $2P = P + P = (x_3, y_3)$ is computed as follows:

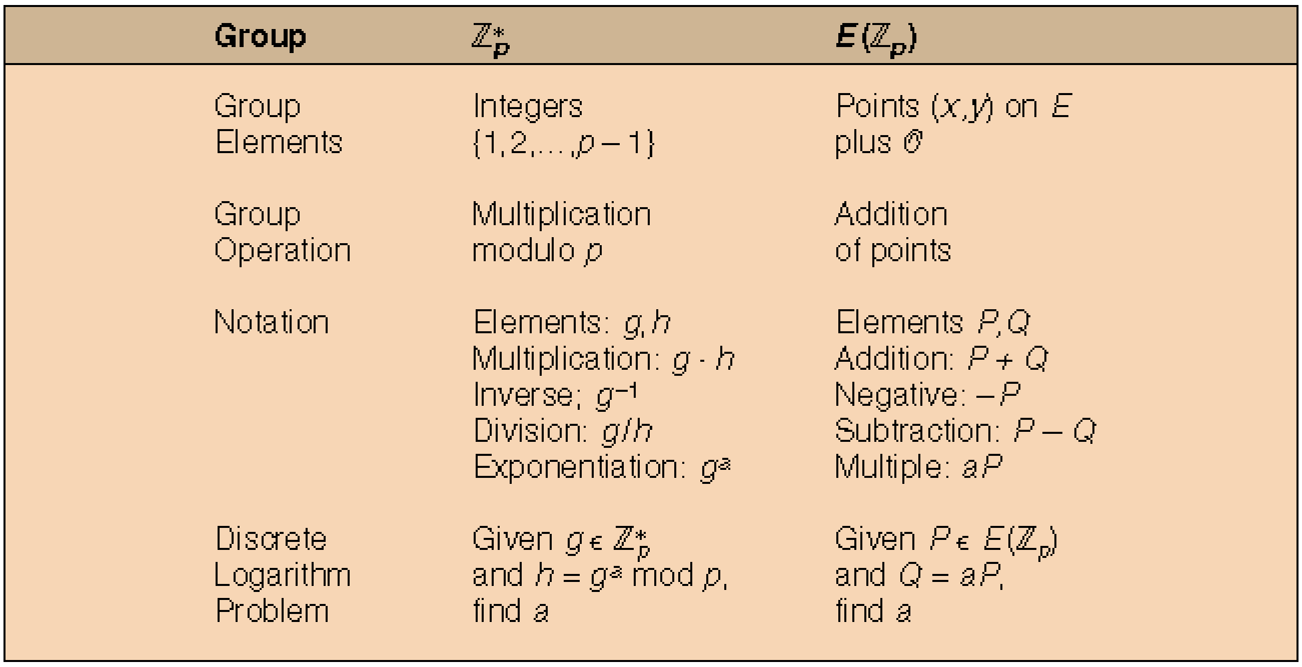
$$\lambda = \frac{3(3^2)+1}{20} = \frac{5}{20} = \frac{1}{4} = 6 \in \mathbb{Z}_{23},$$

$$x_3 = 6^2 - 6 = 30 \equiv 7 \pmod{23}, \text{ and}$$

$$y_3 = 6(3-7) - 10 = -24 - 10 = -11 \equiv 12 \pmod{23}.$$

Hence $2P = (7, 12)$.

Table 1: Correspondence between Z_p^* and $E(Z_p)$ notation



Group	$\mathbb{Z}_p^*$	$E(\mathbb{Z}_p)$
Group Elements	Integers $\{1, 2, \ldots, p-1\}$	Points (x, y) on E plus $\mathcal{O}$
Group Operation	Multiplication modulo p	Addition of points
Notation	Elements: g, h Multiplication: $g \cdot h$ Inverse; g^{-1} Division: g/h Exponentiation: g^a	Elements P, Q Addition: $P + Q$ Negative: $-P$ Subtraction: $P - Q$ Multiple: aP
Discrete Logarithm Problem	Given $g \in \mathbb{Z}_p^*$ and $h = g^a \bmod p$, find a	Given $P \in E(\mathbb{Z}_p)$ and $Q = aP$, find a

ECC Diffie-Hellman

Illustrate here the elliptic curve analog of Diffie-Hellman key exchange, which is a close analogy given elliptic curve multiplication equates to modulo exponentiation.

- can do key exchange analogous to D-H
- users select a suitable curve $E_q(a, b)$
- select base point $G = (x_1, y_1)$
    - with large order n s.t. $nG = O$
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A = n_A G$, $P_B = n_B G$
- compute shared key: $K = n_A P_B$, $K = n_B P_A$
    - same since $K = n_A n_B G$

Example:

$p = 211$; $E_q(0,-4)$, $\rightarrow$ $y^2 = x^3 - 4$; and $G = (2, 2)$, $240G = O$.

A: Private key $n_A = 121$, Public key $P_A = 121(2, 2) = (115, 48)$

B: Private key $n_B = 203$, Public key $P_A = 203(2, 2) = (130, 203)$

Shared secret key: $121(130, 203) = 203(115, 48) = (161, 69)$.

Figure 4. ECC Diffie-Hellman

Global Public Elements

$E_q(a, b)$	elliptic curve with parameters a, b, and q, where q is a prime or an integer of the form 2^m
G	point on elliptic curve whose order is large value n

User A Key Generation

Select private n_A	$n_A < n$
Calculate public P_A	$P_A = n_A \times G$

User B Key Generation

Select private n_B	$n_A < n$
Calculate public P_B	$P_B = n_B \times G$

Calculation of Secret Key by User A

$K = n_A \times P_B$

Calculation of Secret Key by User B

$K = n_B \times P_A$

ECC Encryption/Decryption

Several approaches to encryption/decryption using elliptic curves have been analyzed in the literature. This one is an analog of the ElGamal public-key encryption algorithm. The sender must first encode any message m as a point on the elliptic curve P_m (there are relatively straightforward techniques for this). Note that the ciphertext is a pair of points on the elliptic curve. The sender masks the message using random k, but also sends along a “clue” allowing the receiver who know the private-key to recover k and hence the message. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed hard.

- first encode any message m as a point on the elliptic curve P_m
- select suitable curve & point G as in D-H
- A & B select private keys $n_A<n$, $n_B<n$
- compute public keys: $P_A = n_A G$, $P_B = n_B G$
- A encrypts P_m : $C_m=\{kG, P_m + kP_B\}$,

  k: random positive integer

  P_B: B's public key
- B decrypts C_m compute:

  $P_m + kP_B - n_B(kG) = P_m + k(n_B G) - n_B(kG) = P_m$

Example:

$p = 751$; $E_q(-1, 188)$ $\rightarrow$ $y^2 = x^3 - x + 188$; and $G = (0, 376)$.

A $\rightarrow$ B: $P_m = (562, 201)$

A selects the random number $k = 386$.

B's Public key: $P_B = (201, 5)$.

A computes: 386(0, 376) = (676, 558) and

(562, 201) + 386(201, 5) = (385, 328).

A sends the cipher text: {(676, 558), (385, 328)}.

The Digital Signature Algorithm

The Digital Signature Algorithm (DSA) was proposed in August of 1991 by the U.S. National Institute of Standards and Technology (NIST) and became a U.S. Federal Information Processing Standard (FIPS 186) in 1993. It was the first digital signature scheme to be accepted as legally binding by a government. The algorithm is a variant of the ElGamal signature scheme. It exploits small subgroups in $Z_p^* = \{1, 2, ..., p-1\}$ in order to decrease the size of signatures. Figure 5 shows the key-generation, signature-generation, and signature-verification procedures for DSA.

Figure 5-1: DSA approach

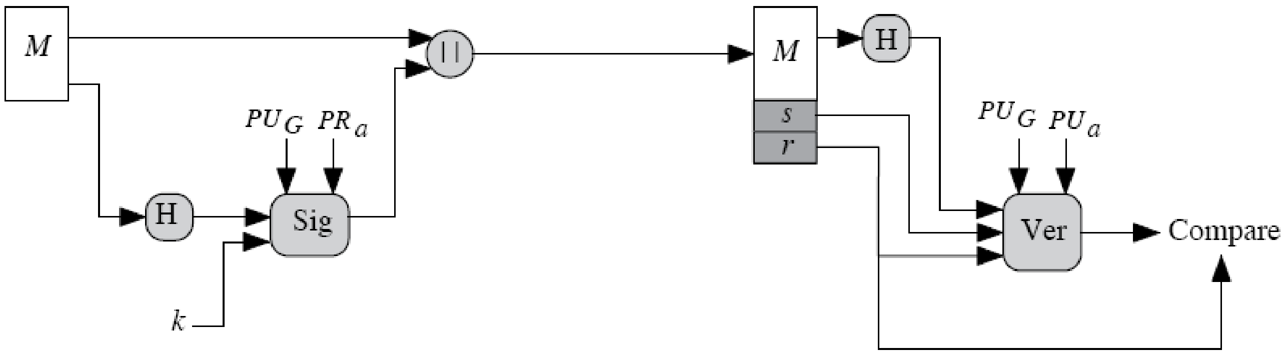


Figure 5-2: DSA Algorithm

Global Public Key Components

p	prime number where $2^{L-1} < p < 2^L$ for $512 \le L \le 1024$ and L a multiple of 64 i.e., bit length of between 512 and 1024 bits in increments of 64 bits
q	prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$ i.e., bit length of 160 bits
g	$= h^{(p-1)/q} \bmod p$ where h is any integer with $1 < h < (p - 1)$ such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x	random or pseudorandom integer with $0 < x < q$

User's Public Key

y	$= g^x \bmod p$

User's Per-Message Secret Number

k	= random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = \left[k^{-1}\left(\mathrm{H}(M) + xr\right)\right] \bmod q$

Signature $= (r, s)$

Verifying

$w = (s^{-\prime})^{-1} \bmod q$

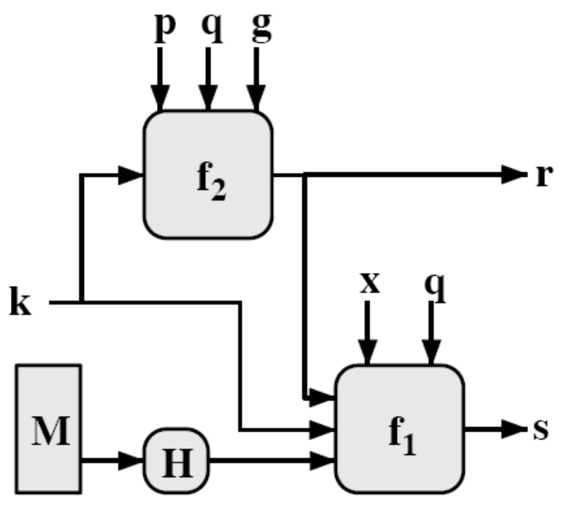
$u_1 = \left[\mathrm{H}(M')w\right] \bmod q$

$u_2 = (r')w \bmod q$

$v = \left[\left(g^{u1}y^{u2}\right) \bmod p\right] \bmod q$

TEST: $v = r'$

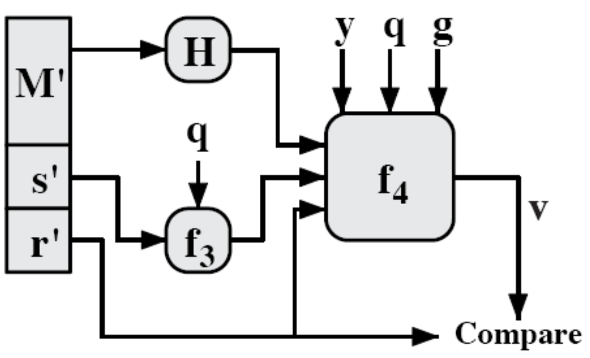
M	=	message to be signed
$\mathrm{H}(M)$	=	hash of M using SHA-1
M', r', s'	=	received versions of M, r, s



$s = f_1(\mathrm{H(M)}, \mathrm{k}, \mathrm{x}, \mathrm{r}, \mathrm{q}) = (\mathrm{k}^{-1} (\mathrm{H(M)} + \mathrm{xr})) \bmod \mathrm{q}$

$r = f_2(\mathrm{k}, \mathrm{p}, \mathrm{q}, \mathrm{g}) = (\mathrm{g}^{\mathrm{k}} \bmod \mathrm{p}) \bmod \mathrm{q}$

(a) Signing



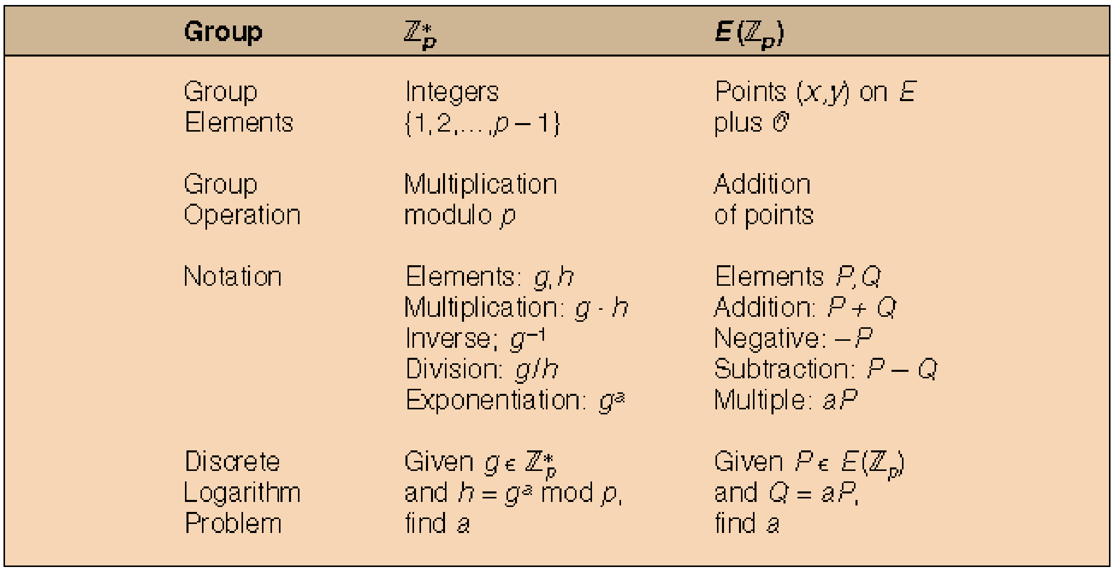
$w = f_3(\mathrm{s}', \mathrm{q}) = (\mathrm{s}')^{-1} \bmod \mathrm{q}$

$v = f_4(\mathrm{y}, \mathrm{q}, \mathrm{g}, \mathrm{H(M')}, \mathrm{w}, \mathrm{r}')$

$= ((\mathrm{g}^{(\mathrm{H(M')w}) \bmod \mathrm{q}}\, \mathrm{y}^{\mathrm{r'w} \bmod \mathrm{q}}) \bmod \mathrm{p}) \bmod \mathrm{q}$

(b) Verifying

Since r and s are each integers less than q, DSA signatures are 320 bits in length. The security of the DSA relies on two distinct (but related) discrete logarithm problems. One is the discrete logarithm problem in Z_p^* where the number field sieve algorithm applies. Since p is a 1024-bit prime, the DSA is currently not vulnerable to this attack. The second discrete logarithm problem works to the base g: given p, q, g, and y, find x such that $y \equiv g^x (\bmod p)$. For large p (1024 bits, for example), the best-known algorithm for this problem is the Pollard rho-method, which takes about $\sqrt{\pi q/2}$ steps. Since $q \approx 2^{160}$, the DSA is not vulnerable to this attack.

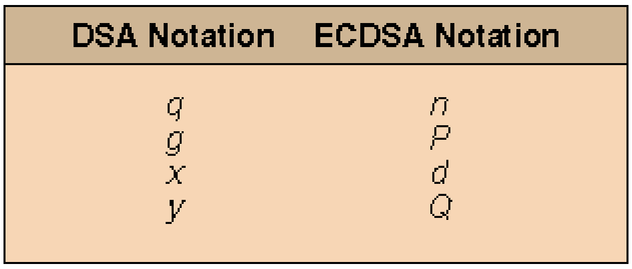


Group	$\mathbb{Z}_p^*$	$E(\mathbb{Z}_p)$
Group Elements	Integers $\{1, 2, \ldots, p-1\}$	Points (x,y) on E plus $\mathcal{O}$
Group Operation	Multiplication modulo p	Addition of points
Notation	Elements: g, h Multiplication: $g \cdot h$ Inverse; g^{-1} Division: g/h Exponentiation: g^a	Elements P, Q Addition: $P + Q$ Negative: $-P$ Subtraction: $P - Q$ Multiple: aP
Discrete Logarithm Problem	Given $g \in \mathbb{Z}_p^*$ and $h = g^a \bmod p$, find a	Given $P \in E(\mathbb{Z}_p)$ and $Q = aP$, find a

The Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is the elliptic curve analogue of the DSA. That is, instead of working in a subgroup of order *q* in Z_p^*, you work in an elliptic curve group $E(Z_p)$. The ECDSA is currently being standardized by the ANSI X9F1 and IEEE P1363 standards committees. Table 2 shows the correspondence between some math notation used Z_p in DSA and ECDSA. Tables 1 and 2 should make the analogies between the DSA and ECDSA more apparent (see Figure 6).

Table 2: Correspondence between DSA and ECDSA notation.

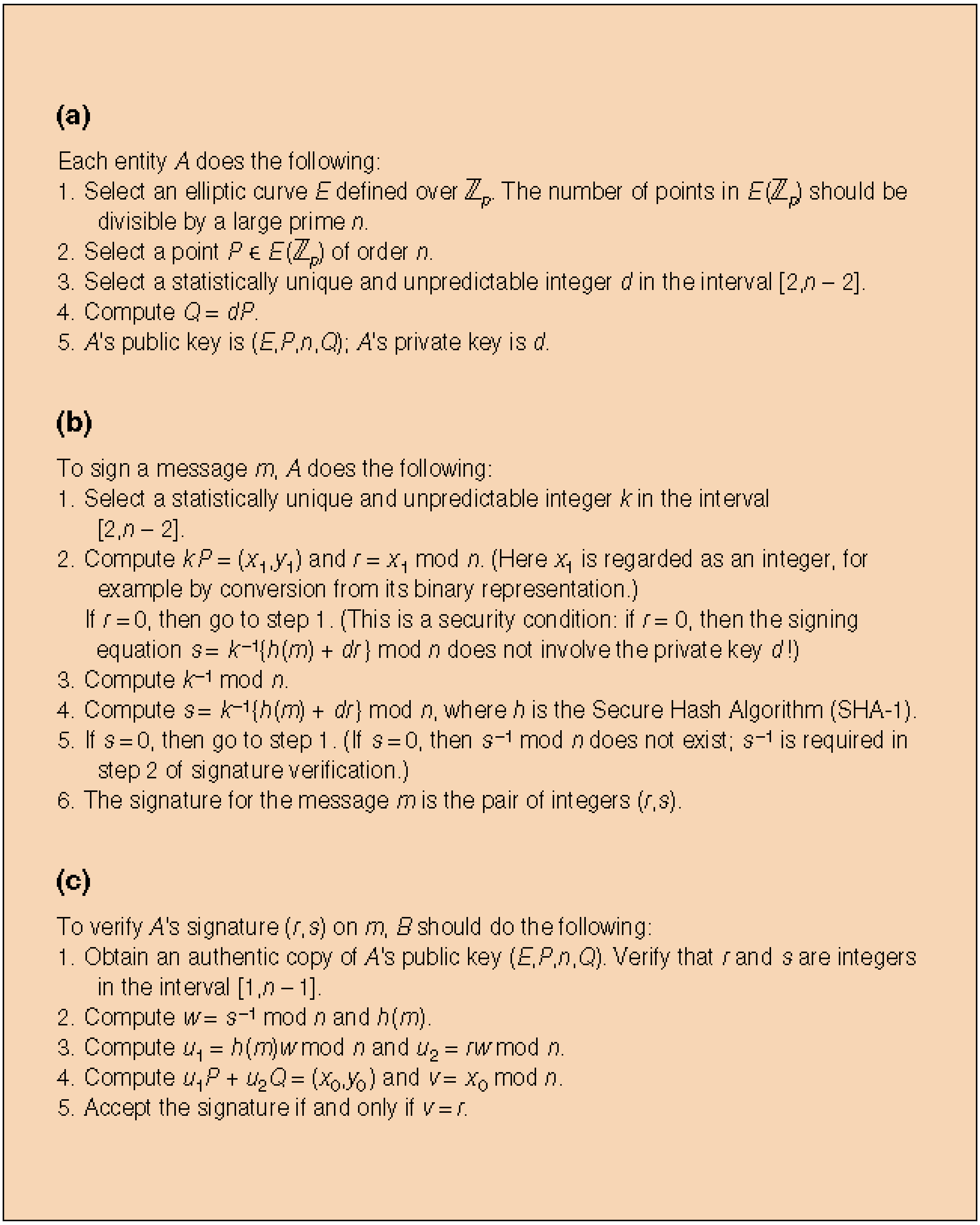


DSA Notation	ECDSA Notation
q	n
g	P
x	d
y	Q

The key-generation, signature-generation, and signature-verification procedures for ECDSA are given in Figure 6. The only significant difference between ECDSA and DSA is in the generation of *r*. The DSA does this by taking the random element (g^k mod *p*) and reducing it modulo *q*, thus obtaining an integer in the interval [1, q -1]. The ECDSA generates the integer r in the interval [1, n -1] by taking the x-coordinate of the random point kP and reducing it modulo n.

To obtain a security level similar to that of the DSA (with 160-bit q and 1024-bit p), the parameter n should have about 160 bits. If this is the case, then DSA and ECDSA signatures have the same bit length (320 bits).

Figure 6: (a) ECDSA key generation; (b) ECDSA signature generation; (c) ECDSA signature verification.



(a)

Each entity A does the following:

1. Select an elliptic curve E defined over $\mathbb{Z}_p$. The number of points in $E(\mathbb{Z}_p)$ should be divisible by a large prime n.
2. Select a point $P \in E(\mathbb{Z}_p)$ of order n.
3. Select a statistically unique and unpredictable integer d in the interval $[2,n-2]$.
4. Compute $Q = dP$.
5. A's public key is (E,P,n,Q); A's private key is d.

(b)

To sign a message m, A does the following:

1. Select a statistically unique and unpredictable integer k in the interval $[2,n-2]$.
2. Compute $kP = (x_1,y_1)$ and $r = x_1 \bmod n$. (Here x_1 is regarded as an integer, for example by conversion from its binary representation.)
   If $r = 0$, then go to step 1. (This is a security condition: if $r = 0$, then the signing equation $s = k^{-1}\{h(m) + dr\} \bmod n$ does not involve the private key d!)
3. Compute $k^{-1} \bmod n$.
4. Compute $s = k^{-1}\{h(m) + dr\} \bmod n$, where h is the Secure Hash Algorithm (SHA-1).
5. If $s = 0$, then go to step 1. (If $s = 0$, then $s^{-1} \bmod n$ does not exist; s^{-1} is required in step 2 of signature verification.)
6. The signature for the message m is the pair of integers (r,s).

(c)

To verify A's signature (r,s) on m, B should do the following:

1. Obtain an authentic copy of A's public key (E,P,n,Q). Verify that r and s are integers in the interval $[1,n-1]$.
2. Compute $w = s^{-1} \bmod n$ and $h(m)$.
3. Compute $u_1 = h(m)w \bmod n$ and $u_2 = rw \bmod n$.
4. Compute $u_1P + u_2Q = (x_0,y_0)$ and $v = x_0 \bmod n$.
5. Accept the signature if and only if $v = r$.

Instead of each entity generating its own elliptic curve, the entities may elect to use the same curve E and point P of order n. In this case, an entity's public key consists only of the point Q. This results in public keys of smaller sizes. Additionally, there are point compression techniques whereby the point $Q = (x_Q, y_Q)$ can be efficiently constructed from its x-coordinate x_Q and a specific bit of the y-coordinate y_Q. Thus, for example, if $p \approx 2^{160}$ (so elements in Z_p are 160-bit strings), then public keys can be represented as 161-bit strings.

Security Issues

The basis for the security of elliptic curve cryptosystems such as the ECDSA is the apparent intractability of this elliptic curve discrete logarithm problem (ECDLP): given an elliptic curve E defined over $\mathbb{Z}_p$, a point $P \in E(\mathbb{Z}_p)$ of order

n, and a point $Q \in E(\mathbb{Z}_p)$, determine the integer l, $0 \le l \le n-1$, such that $Q = lP$, provided that such an integer exists.

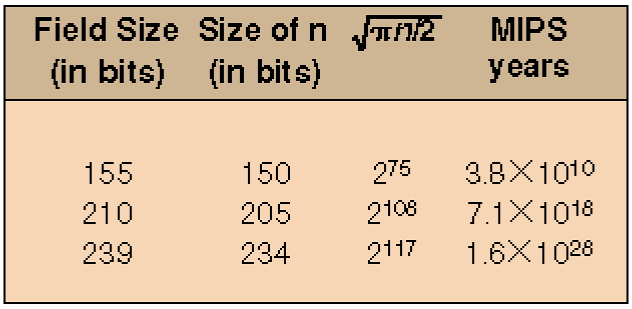
Over the past 12 years, the ECDLP has received considerable attention from leading mathematicians, and no significant weaknesses have been reported. An algorithm by Pohlig and Hellman reduces the determination of *l* to the determination of *l* modulo each of the prime factors of *n*. Hence, to achieve the maximum possible security level, *n* should be prime. The best-known algorithm to date for the ECDLP in general is the Pollard rho-method, which takes about $\sqrt{\pi n/2}$ steps, where a step is an elliptic curve addition. In 1993, Paul van Oorschot and Michael Wiener showed how the Pollard rho-method can be parallelized so that if *r* processors are used, the expected number of steps by each processor before a single discrete logarithm is obtained is $\sqrt{\pi n/2}/r$.

In considering software attacks on ECDLP, we assume that a million instructions per second (MIPS) machine can perform 4310^4 elliptic curve additions per second. (This estimate is indeed conservative -- an application-specific integrated circuit for performing elliptic curve operations over the field $\mathbb{F}_2{}^{155}$ described in "An Implementation of Elliptic Curve Cryptosystems over $\mathbb{F}_2{}^{155}$," by G. Agnew, R. Mullin, and S. Vanstone [*IEEE Journal on Selected Areas in Communications*, 11, 1993] has a 40-MHz clock rate and can perform roughly 40,000 elliptic additions per second.) Then the number of elliptic curve additions that can be performed by a 1 MIPS machine in one year is

$$(4\times 10^4)\bullet(60\times 60\times 24\times 365)\approx 2^{40}$$

Table 3 shows, for various values of n, the computing power required to compute a single discrete logarithm using the Pollard rho-method. A MIPS year is equivalent ot the computational power of a computer that is rated at 1 MIPS and utilized for one year.

Table 3: Computing power to compute elliptic curve logarithms with the Pollard rho-method.

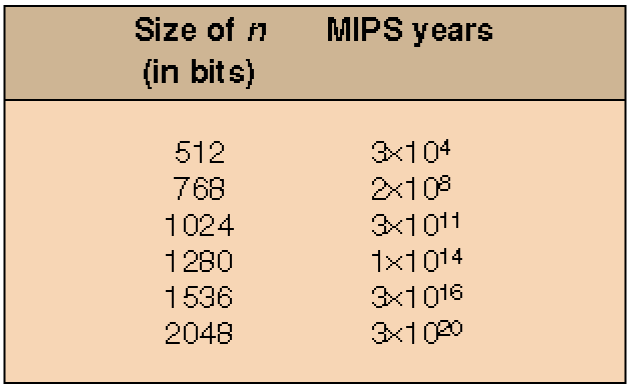


Field Size (in bits)	Size of n (in bits)	$\sqrt{\pi n/2}$	MIPS years
155	150	2^{75}	3.8×10^{10}
210	205	2^{108}	7.1×10^{18}
239	234	2^{117}	1.6×10^{28}

For instance, if 10,000 computers each rated at 1000 MIPS are available, and $n\approx 2^{150}$, then an elliptic curve discrete logarithm can be computed in 3800 years. Andrew Odlyzko has estimated that if 0.1 percent of the world's computing power were available for one year to work on a collaborative effort to break some challenge cipher, then the computing power available would be 10^8 MIPS years in 2004 and 10^{10}-10^{11} MIPS years in 2014.

To put the numbers in Table 3 in some perspective, Table 4 (see A. Odlyzko's "The Future of Integer Factorization," *CryptoBytes: The Technical Newsletter of RSA Laboratories*, Summer 1995; also available at http://www.rsa.com/) shows the estimated computing power required to factor integers with current versions of the general number field sieve.

Table 4: Computing power required to factor integers using the general number field sieve.



Size of *n* (in bits)	MIPS years
512	3×10^4
768	2×10^8
1024	3×10^{11}
1280	1×10^{14}
1536	3×10^{16}
2048	3×10^{20}

A more promising attack (for well-funded attackers) on elliptic curve systems would be to build special-purpose hardware for a parallel search using the Pollard rho-method. In "Parallel Collision Search with Cryptanalytic Applications" (to appear in *Journal of Cryptology*), P. van Oorschot and M. Wiener provide a detailed study of such a possibility. They estimated that if $n \approx 10^{36} \approx 2^{120}$, then a machine with m = 325,000 processors that could be built for about $10 million would compute a single discrete logarithm in about 35 days.

It should be pointed out that in the software and hardware attacks just described, computation of a single elliptic curve discrete logarithm has the effect of revealing a *single* user's private key. The same effort must be repeated to determine another user's private key.

In "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security" (January 1996, available from http://theory.lcs.mit.edu/~rivest/publications.html), M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, and

M. Wiener reported on the minimum key lengths required for secure symmetric-key encryption schemes (such as DES and IDEA). Their report comes to the following conclusion:

To provide adequate protection against the most serious threats -- well-funded commercial enterprises or government intelligence agencies -- keys used to protect data today should be at least 75 bits long. To protect information adequately for the next 20 years in the face of expected advances in computing power, keys in newly-deployed systems should be at least 90 bits long.

Extrapolating these conclusions to the case of elliptic curves, you see that n should be at least 150 bits for short-term security and 180 bits for medium-term security. This extrapolation is justified by the following considerations:

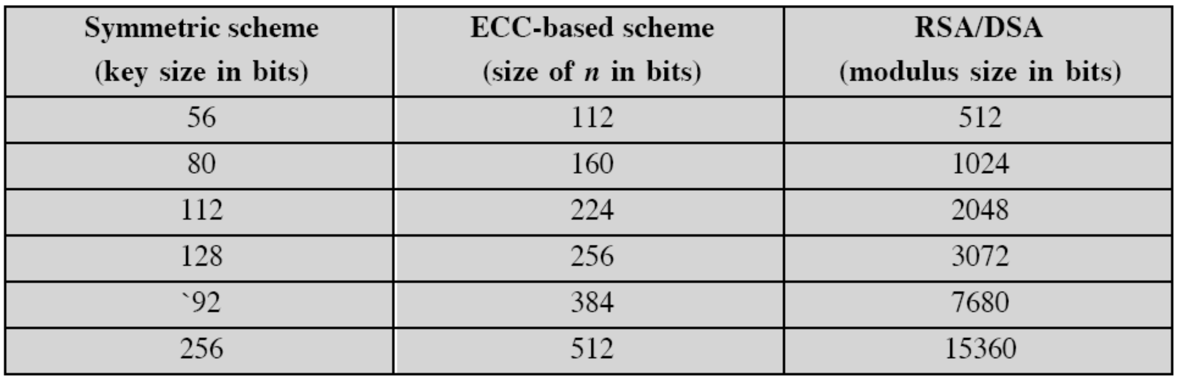
- Exhaustive search through a k-bit symmetric-key cipher takes about the same time as the Pollard rho algorithm applied to an elliptic curve having a $2k$-bit parameter n.

- Exhaustive searches with a symmetric-key cipher and the Pollard rho algorithm can be parallelized with a linear speedup.
- A basic operation with elliptic curves (addition of two points) is computationally more expensive than a basic operation in a symmetric-key cipher (encryption of one block).
- In both symmetric-key ciphers and elliptic curve systems, a "break" has the same effect: It will recover a single private key.

ECC Security

- Compared to factoring, can use much smaller key sizes than with RSA etc.
- For equivalent key lengths computations are roughly equivalent.
- Hence for similar security ECC offers significant computational advantages
- Comparable Key Sizes for Equivalent Security

Table 5: Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis



Symmetric scheme (key size in bits)	**ECC-based scheme (size of *n* in bits)**	**RSA/DSA (modulus size in bits)**
56	112	512
80	160	1024
112	224	2048
128	256	3072
`92	384	7680
256	512	15360