Elliptic Curve DSA (ECDSA): An Enhanced DSA

Don B. Johnson
Certicom Corp.
200 Matheson Blvd. West, Suite 103
Mississauga, Ontario, Canada, L5R 3L7.
Email: djohnson@certicom.ca

Alfred J. Menezes
Department of C&O
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1.
Email: ajmeneze@math.uwaterloo.ca

Abstract

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA), and is under consideration for standardization by the ANSI X9 committee. Unlike the normal discrete logarithm problem and the integer factorization problem, the elliptic curve discrete logarithm problem has no subexponential-time algorithm. For this reason, the strength-per-key-bit is substantially greater in an algorithm that uses elliptic curves. In this paper, we compare the draft ANSI X9.62 ECDSA to the ANSI X9.30 DSA, the latter of which is identical to FIPS 186 DSS.

1 Introduction

Since the introduction of the concept of public-key cryptography by Whitfield Diffie and Martin Hellman [11] in 1976, the cryptographic importance of the well-studied discrete logarithm problem's apparent intractability has been recognized. Taher ElGamal [12] first described how this problem could be utilized in public-key encryption and digital signature schemes. ElGamal's methods have been refined and incorporated into various protocols to meet a variety of applications, and one of its extensions forms the basis for the U.S. government digital signature algorithm (DSA).

We begin by introducing some basic mathematical terminology. A *group* is an abstract mathematical object consisting of a set G together with an operation $*$ defined on pairs of elements of G. The operation must have certain properties, similar to those with which we are familiar from ordinary arithmetic. More precisely:

1. (*closure*) $a * b \in G$ for all a, $b \in G$.
2. (*associativity*) $a * (b * c) = (a * b) * c$ for all $a, b, c \in G$.
3. (*existence of identity*) There exists an element $e \in G$, called the *identity*, such that $e * a = a * e = a$ for all $a \in G$.
4. (*existence of inverses*) For each $a \in G$ there is an element $b \in G$ such that $a * b = b * a = e$. The element b is called the *inverse* of a, and is denoted by a^{-1}.

A group G is said to be *abelian* if $a * b = b * a$ for all $a, b \in G$. The *order* of a group G is the number of elements in G.

For example, the integers modulo n, namely $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$, form a group of order n under the operation of addition modulo n. The (additive) identity of this group is 0. If p is a prime number, then the non-zero elements of $\mathbb{Z}_p$, namely $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$, form a group of order $p-1$ under the operation of multiplication modulo p. The (multiplicative) identity of this group is 1.

The *order* of a group element $g \in G$ is the least positive integer n such that $g^n = 1$. For example, in the group $\mathbb{Z}_{11}^*$, the element $g = 3$ has order 5, since

$$\begin{aligned} 3^1 &\equiv 3 \pmod{11}, \\ 3^2 &\equiv 9 \pmod{11}, \\ 3^3 &\equiv 5 \pmod{11}, \\ 3^4 &\equiv 4 \pmod{11}, \text{ and} \\ 3^5 &\equiv 1 \pmod{11}. \end{aligned}$$

The discrete logarithm problem, as first employed by Diffie and Hellman in their key agreement protocol, was defined explicitly as the problem of finding logarithms in the group $\mathbb{Z}_p^*$: given $g \in \mathbb{Z}_p^*$ of order n, and given $h \in \mathbb{Z}_p^*$, find an integer x, $0 \le x \le n-1$, such that $g^x \equiv h \pmod{p}$, provided that such an integer exists. The integer x is called the *discrete logarithm* of h to the base g. For example, consider $p = 17$. Then $g = 10$ is an element of order $n = 16$ in $\mathbb{Z}_{17}^*$. If $h = 11$, then the discrete logarithm of h to the base g is 13 because $10^{13} \equiv 11 \pmod{17}$.

These concepts can be extended to arbitrary groups. Let G be a group of order n, and let α be an element of G. The *discrete logarithm problem* for G is the following: given elements α and $\beta \in G$, find an integer x, $0 \le x \le n-1$, such that $\alpha^x = \beta$, provided that such an integer exists.

A variety of groups have been proposed for cryptographic use. There are two primary reasons for this. First, the operation in some groups may be easier to implement in software or in hardware than the operation in other groups. Second, the discrete logarithm problem in the group may be harder than the discrete logarithm problem in $\mathbb{Z}_p^*$. Consequently, one could use a group G that is smaller than $\mathbb{Z}_p^*$ while maintaining the same level of security. Such is the case with elliptic curve groups, which were first proposed for cryptographic use independently by Neal Koblitz [19] and Victor Miller [25] in 1985. The resulting elliptic curve cryptosystems (ECC) have smaller key sizes, smaller bandwidth requirements, less power consumption, and faster implementations. These features are especially attractive for security applications where computational power and integrated circuit space is limited, such as smart cards, PC cards, and wireless devices.

2 The Digital Signature Algorithm (DSA)

The DSA was proposed in August 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. The FIPS 186 standard is also referred to as the Digital Signature Standard (DSS). The DSA was the first digital signature scheme accepted as legally binding by a government. The algorithm is a variant of the ElGamal signature scheme. It exploits small subgroups in $\mathbb{Z}_p^*$ in order to decrease the size of signatures. The key generation, signature generation, and signature verification procedures for DSA are given next.

DSA key generation. Each entity A does the following:

1. Select a prime q such that $2^{159} < q < 2^{160}$.
2. Select a 1024-bit prime number p with the property that $q \mid p-1$. (The DSS mandates that p be a prime such that $2^{511+64t} < p < 2^{512+64t}$ where $0 \le t \le 8$. If $t = 8$ then p is a 1024-bit prime.)
3. Select an element $h \in \mathbb{Z}_p^*$ and compute $g = h^{(p-1)/q} \bmod p$; repeat until $g \neq 1$. (g is a generator of the unique cyclic group of order q in $\mathbb{Z}_p^*$.)
4. Select a random integer x in the interval $[1, q-1]$.
5. Compute $y = g^x \bmod p$.
6. A's public key is (p, q, g, y); A's private key is x.

DSA signature generation. To sign a message m, A does the following:

1. Select a random integer k in the interval $[1, q-1]$.
2. Compute $r = (g^k \bmod p) \bmod q$.
3. Compute $k^{-1} \bmod q$.
4. Compute $s = k^{-1}\{h(m) + xr\} \bmod q$, where h is the Secure Hash Algorithm (SHA-1).
5. If $s = 0$ then go to step 1. (If $s = 0$, then $s^{-1} \bmod q$ does not exist; s^{-1} is required in step 3 of signature verification.)
6. The signature for the message m is the pair of integers (r, s).

DSA signature verification. To verify A's signature (r, s) on m, B should:

1. Obtain an authentic copy of A's public key (p, q, g, y).
2. Verify that r and s are integers in the interval $[1, q-1]$.
3. Compute $w = s^{-1} \bmod q$ and $h(m)$.
4. Compute $u_1 = h(m)w \bmod q$ and $u_2 = rw \bmod q$.
5. Compute $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
6. Accept the signature if and only if $v = r$.

Since r and s are each integers less than q, DSA signatures are 320 bits in size. The security of the DSA relies on two distinct but related discrete logarithm problems. One is the discrete logarithm problem in $\mathbb{Z}_p^*$ where the number field sieve algorithm (see Gordon [17] and Schirokauer [32]) applies; this algorithm has a subexponential running time. More precisely, the running time of the algorithm is

$$O\left(\exp\left((c + o(1))(\ln p)^{1/3}(\ln \ln p)^{2/3}\right)\right), \qquad (1)$$

where $c \approx 1.923$, and $\ln n$ denotes the natural logarithm function. If p is a 1024-bit prime, then the expression (1) represents an infeasible amount of computation (see Section 5); thus 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 $y \equiv g^x \pmod{p}$. For large p (e.g., 1024-bits), the best algorithm known for this problem is the Pollard rho-method (see Section 5), and takes about

$$\sqrt{\pi q/2} \qquad (2)$$

steps. If $q \approx 2^{160}$, then the expression (2) represents an infeasible amount of computation (see Section 5); thus the DSA is not vulnerable to this attack. However, note that there are two primary security parameters for DSA, the size of p and the size of q. Increasing one without a corresponding increase in the other will not result in an effective increase in security.

3 Background in Elliptic Curves

We proceed now to give a quick introduction to the theory of elliptic curves. Chapter 6 of Koblitz's book [21] provides an introduction to elliptic curves and elliptic curve systems. For more details, consult Menezes' book [23].

For simplicity, we shall restrict this discussion to elliptic curves over $\mathbb{Z}_p$, where p is a prime greater than 3. We mention 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 implementations of the elliptic curve arithmetic.

An *elliptic curve* E over $\mathbb{Z}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b, \tag{3}$$

where $a, b \in \mathbb{Z}_p$, and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, together with a special point $\mathcal{O}$, called the *point at infinity*. The set $E(\mathbb{Z}_p)$ consists of all points (x, y), $x \in \mathbb{Z}_p$, $y \in \mathbb{Z}_p$, which satisfy the defining equation (3), together with $\mathcal{O}$.

Example 1 (*elliptic curve over* $\mathbb{Z}_{23}$) Let $p = 23$ and consider the elliptic curve $E : y^2 = x^3 + x + 1$ defined over $\mathbb{Z}_{23}$. (In the notation of equation (3), we have $a = 1$ and $b = 1$.) Note that $4a^3 + 27b^2 = 4+27 = 31 \equiv 8 \pmod{23}$, so E is indeed an elliptic curve. The points in $E(\mathbb{Z}_{23})$ are $\mathcal{O}$ and the following:

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

Addition Formula

There is a rule for adding two points on an elliptic curve $E(\mathbb{Z}_p)$ to give a third elliptic curve point. Together with this addition operation, the set of points $E(\mathbb{Z}_p)$ forms a group with $\mathcal{O}$ serving as its identity. It is this group that is used in the construction of elliptic curve cryptosystems.

The addition rule is best explained geometrically. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve E. Then the *sum* of P and Q, denoted $R = (x_3, y_3)$, is defined as follows. First draw the line through P and Q; this line intersects the elliptic curve in a third point. Then R is the reflection of this point in the x-axis. This is depicted in Figure 1. The elliptic curve in the figure consists of two parts, the ellipse-like figure and the infinite curve.

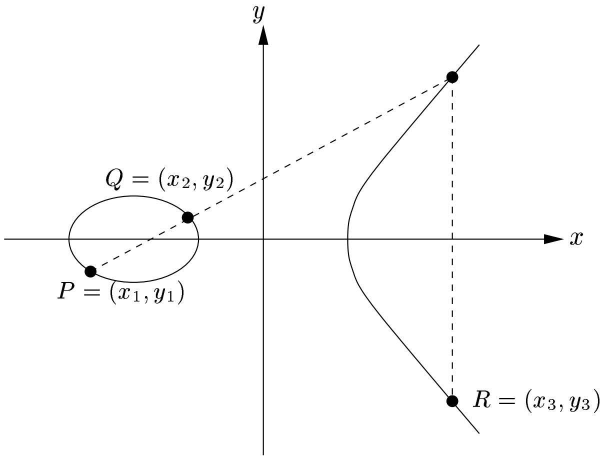

Figure 1: Geometric description of the addition of two distinct elliptic curve points: $P + Q = R$.

If $P = (x_1, y_1)$, then the *double* of P, denoted $R = (x_3, y_3)$, is defined as follows. First draw the tangent line to the elliptic curve at P. This line intersects the elliptic curve in a second point. Then R is the reflection of this point in the x-axis. This is depicted in Figure 2.

The following algebraic formulae for the sum of two points and the double of a point can now be derived from the geometric description.

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

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

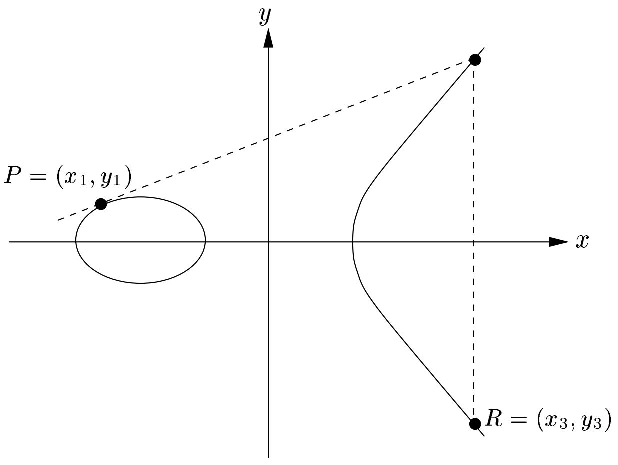

Figure 2: Geometric description of the doubling of an elliptic curve point: $P + P = R$.

and

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

Observe that the addition of two elliptic curve points in $E(\mathbb{Z}_p)$ requires a few arithmetic operations (addition, subtraction, multiplication, and inversion) in the underlying finite field $\mathbb{Z}_p$.

Example 2 (*elliptic curve addition*) Consider the elliptic curve defined in Example 1.

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

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

   Note that $2^{-1} = 12$ since $2 \cdot 12 \equiv 1 \pmod{23}$. Finally,

   $$\begin{aligned} x_3 &= 11^2 - 3 - 9 = 109 \\ &\equiv 17 \pmod{23}, \end{aligned}$$

   and

   $$\begin{aligned} y_3 &= 11(3 - 17) - 10 = -164 \\ &\equiv 20 \pmod{23}. \end{aligned}$$

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

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

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

   Note that $4^{-1} = 6$ since $4 \cdot 6 \equiv 1 \pmod{23}$. Finally,

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

   and

   $$\begin{aligned} y_3 &= 6(3 - 7) - 10 = -34 \\ &\equiv 12 \pmod{23}. \end{aligned}$$

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

Hasse's theorem states that the number of points on an elliptic curve is $\#E(\mathbb{Z}_p) = p + 1 - t$ where $|t| \leq 2\sqrt{p}$; $\#E(\mathbb{Z}_p)$ is called the *order* of the elliptic curve. In other words, the order of an elliptic curve $E(\mathbb{Z}_p)$ is roughly equal to the size p of the underlying field. There is a polynomial-time algorithm, due to Schoof [33], for counting the number of points on an elliptic curve. Although this algorithm is quite cumbersome in practice, several improvements have been proposed in recent years which make the algorithm practical. For some recent work in this area, see Lercier [22].

For historical reasons, the group operation for an elliptic curve $E(\mathbb{Z}_p)$ has been called *addition*. In contrast, the group operation in $\mathbb{Z}_p^*$ 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 $\mathbb{Z}_p^*$ and $E(\mathbb{Z}_p)$.

4 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 $\mathbb{Z}_p^*$, we work in an elliptic curve group $E(\mathbb{Z}_p)$. The ECDSA is currently being standardized within the ANSI X9F1, IEEE P1363, and ISO SC27 standards committees. Table 2 shows the correspondence between some math notation used in DSA and ECDSA. Using Tables 1 and 2, the analogies between the DSA and ECDSA should be more apparent.

The key generation, signature generation, and signature verification procedures for ECDSA are given next.

ECDSA key generation. 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.

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

Table 1: Correspondence between $\mathbb{Z}_p^*$ and $E(\mathbb{Z}_p)$ notation.

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

Table 2: Correspondence between DSA and ECDSA notation.

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 $[1, n-1]$.
4. Compute $Q = dP$.
5. A's public key is (E, P, n, Q); A's private key is d.

ECDSA signature generation. To sign a message m, A does the following:

1. Select a statistically unique and unpredictable integer k in the interval $[1, n-1]$.
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 3 of signature verification.)
6. The signature for the message m is the pair of integers (r, s).

ECDSA signature verification. To verify A's signature (r, s) on m, B should:

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

ANSI X9.62 mandates that $n > 2^{160}$. 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 size (320 bits).

Instead of each entity generating its own elliptic curve, the entities may elect to use the same curve E over $\mathbb{Z}_p$, and point P of order n; these quantities are then called *system parameters* or *domain parameters*. (In DSA, the analogous system parameters would be p, q and g.) In this case, an entity's public key consists only of the point Q. This results in public keys of smaller sizes.

ECDSA has a number of consistencies with the DSA. The important ones are:

1. Both algorithms are based on the ElGamal signature scheme and use the same signing equation: $s = k^{-1}\{h(m) + dr\} \bmod n$.

2. In both algorithms, the values that are relatively difficult to generate are the system parameters (p, q and g for the DSA; E, P and n for the ECDSA) which are public – their generation can be audited and independently checked for validity. This helps show that they were not produced to meet some secret (e.g., trapdoor) criteria. Generating a private key, given a set of system parameters, is relatively simple and

generating the associated public key is straightforward. Contrast this with the RSA algorithm, where the values that are difficult to generate (the primes p and q) must be kept private or destroyed – public auditing of the correct generation of p and q is not feasible without opening the private key.

3. In their current version, both DSA and ECDSA use the SHA-1 as the sole cryptographic hash function. This may be modified in the future by, for example, allowing a hash function which offers output values of variable lengths.

However, there are some significant differences and advantages of ANSI X9.62 ECDSA over the DSS, as follows:

1. ANSI X9.62 ECDSA specifies the steps of a procedure to validate the generated system parameters, while DSS does not explicitly state them. In one common scenario, a user would be given the DSA system parameters by a trusted third party where there is no need to validate the system parameters. However, another useful scenario is where one party has generated its own personal system parameters. In the latter scenario one may wish to verify that the supplied system parameters actually meet all security requirements before using them. Such a system parameter validation procedure could be added to the DSS.

2. In the DSS, the use of a canonical seeded one-way hash function to generate the system parameters *verifiably at random* is mandated, in order to ensure the system parameters do not meet some hidden trap-door criteria that might be hard to discern from examination of the system parameters by themselves. Given the input seed, anyone can validate that the DSA system parameters were indeed generated randomly. In the ECDSA, the analogous attacks on weak system parameters do not apply. This allows use of special elliptic curves with advantageous performance, such as Koblitz curves [20]. However, ANSI X9.62 also specifies a method for generating elliptic curves verifiably at random. Given the input seed to this method, anyone can validate that the elliptic curve was indeed generated randomly. Use of this random generation method can help mitigate concerns regarding the possible future discovery of new and rare classes of weak elliptic curves, as such rare curves would essentially never be generated.

3. ANSI X9.62 ECDSA specifies a public key validation routine. This allows anyone at any time to validate that a claimed ECDSA public key actually conforms to the arithmetic requirements for such a key. Namely, given a valid set of system parameters E, P and n, and a purported public key Q, one verifies that Q is a point on E, that $Q \neq \mathcal{O}$, and that $nQ = \mathcal{O}$. As the validation is 100%, successful execution of the routine demonstrates that an associated private key can logically exist, although, of course, it does not demonstrate that the private key actually does exist nor that the claimed owner actually owns the private key. Public key validation is a useful service that can help assure the owner of the associated private key that the public key output by a system is plausible, i.e., that no subtle undetected errors occurred during key generation. It is also a useful service for a potential user of a public key to perform – if a public key is bogus, it should not be used. Such a public key validation routine could be added to the DSS.

4. In ANSI X9.62 ECDSA, a method called *point compression* allows for a point on the elliptic curve (e.g., a public key Q) to be compactly represented by one field element and one additional bit, rather than two field elements. Thus, for example, if $p \approx 2^{160}$ (so elements in $\mathbb{Z}_p$ are 160-bit strings), then public keys can be represented as 161-bit strings. This can lead to a substantial reduction in the size of a public-key certificate, on the order of 25% when compared with other asymmetric algorithms.

5. At Crypto '96, Serge Vaudenay [38] demonstrated a theoretical weakness in DSA based on his insight that the actual hash function used in the DSA is SHA-1 mod q, not just SHA-1, where q is the 160-bit prime. This weakness allows the selective forgery of one message if the adversary can select the system parameters. This weakness does not exist in the DSA if the system parameters are selected as specified in the DSS. The ECDSA analog for q is n, the order of the base point P. If $n > 2^{160}$, then such an attack is not possible.

6. In the DSA, there is an optional check during signature generation on the components of the digital signature (r and s) are non-zero. This results in an extremely low probability that a conforming system which did not do the bounds check could generate a signature that would not

verify. However, this means that generically written code (i.e., code that does not take into account knowledge of the underlying DSA implementation) must verify the generated signature itself, if there must be 100% assurance that the signature will verify. The analogous bounds check has been made mandatory in ECDSA; 100% of the digital signatures that are generated will verify. Note, however, that there still may be situations where it is wise to explicitly verify a just-generated digital signature, such as when it is anticipated that the signature will be distributed widely. Explicit verification will help detect inadvertent implementation and application errors.

7. The private key d and the per-signature value k in ECDSA are defined to be *statistically unique and unpredictable* rather than merely *random* as in DSA. This is an important clarification and is a better statement of the security requirements. If k can be determined or if k repeats then an adversary can recover d, the private key. Of course, the use of a random value is explicitly stated as being allowed; however architecturally it is preferable to state the requirements rather than mandate a particular way to meet the requirements. For example, giving the requirements allows a high security implementation to filter the k values to ensure there are no repeats. This possibility is not allowed if k is required to be random. Also, stating the requirements gives more guidance to implementers and users regarding what constitutes a security concern.

5 Security Issues

The basis for the security of elliptic curve cryptosystems such as the ECDSA is the apparent intractability of the following *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, $0 \leq l \leq n-1$, such that $Q = lP$, provided that such an integer exists.

Over the past thirteen years, the ECDLP has received considerable attention from leading mathematicians around the world, and no significant weaknesses have been reported. An algorithm due to Pohlig and Hellman [28] reduces the determination of l to the determination of l modulo each of the prime factors of n. Hence, in order to achieve the maximum possible security level, n should be prime. The best algorithm known to date for the ECDLP in general is the Pollard rho-method [29] which takes about $\sqrt{\pi n/2}$ steps, where a *step* here is an elliptic curve addition. In 1993, Paul van Oorschot and Michael Wiener [27] showed how the Pollard rho-method can be parallelized so that if r processors are used, then the expected number of steps by each processor before a single discrete logarithm is obtained is $(\sqrt{\pi n/2})/r$.

Menezes, Okamoto and Vanstone [24] and Frey and Rück [16] showed how the ECDLP can be reduced to the DLP in extension fields of $\mathbb{Z}_p$, for which subexponential-time algorithms are known. However, this reduction algorithm is only known to be efficient for a very special class of curves known as *supersingular curves*. There is a simple test to ensure that an elliptic curve is not vulnerable to this attack; through this test, these curves are prohibited by ANSI X9.62 ECDSA.

Another weak class of elliptic curves are the so-called *anomalous curves*. These are curves E defined over $\mathbb{Z}_p$ for which $\#E(\mathbb{Z}_p) = p$. The attack on these curves was discovered independently by Semaev [35], Smart [36], and Satoh and Araki [31], and generalized by Rück [30]. As with supersingular curves, there is a simple test to ensure that an elliptic curve is not vulnerable to this attack; through this test, these curves are prohibited by ANSI X9.62 ECDSA.

It should be emphasized that the elliptic curves that succumb to one of the above two attacks are very rare. A prudent way then to guard against these attacks, and similar attacks against special classes of curves that may be discovered in the future, is to select the elliptic curve E at random (subject to the condition that $\#E(\mathbb{Z}_p)$ be divisible by a large prime).

Some other research on the ECDLP and related problems that has been reported in the literature include the following:

1. V. Miller [25].
2. L. Adleman, J. DeMarrais and M. Huang [1].
3. D. Boneh and R. Lipton [9].
4. A. Stein [37].
5. R. Balasubramanian and N. Koblitz [7].
6. R. Zuccherato [40].
7. R. Flassenberg and S. Paulus [15].
8. J. Voloch [39].

To encourage further research, Certicom Corp. has launched an ECC Challenge [10]. Prizes ranging in value up to $100,000 are being offered for solutions to specific instances of the ECDLP.

Software Attacks

We assume that a MIPS (Million Instructions Per Second) machine can perform 4×10^4 elliptic curve additions per second. This estimate is indeed high – an application-specific integrated circuit (ASIC) for performing elliptic curve operations over the field $\mathbb{F}_{2^{155}}$ described in [3] has a 40 MHz clock-rate and can perform roughly 40,000 elliptic additions per second. Also, the software implementation by Schroeppel et al. [34] on a SPARC IPC (rated at 25 MIPS) performs 2,000 elliptic curve additions per second. Then, the number of elliptic curve additions that can be performed by a 1 MIPS machine in one year is

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

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

Field size (in bits)	Size of n (in bits)	$\sqrt{\pi n/2}$	MIPS years
163	160	2^{80}	9.6×10^{11}
191	186	2^{93}	7.9×10^{15}
239	234	2^{117}	1.6×10^{23}
359	354	2^{177}	1.5×10^{41}
431	426	2^{213}	1.0×10^{52}

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

As an example, if 10,000 computers each rated at 1,000 MIPS are available, and $n \approx 2^{160}$, then a single elliptic curve discrete logarithm can be computed in 96,000 years. That is, a single private key can be recovered from a single public key. Andrew Odlyzko [26] has estimated that if 0.1% 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} to 10^{11} MIPS years in 2014.

To put the numbers in Table 3 into some perspective, Table 4 (due to Odlyzko [26]) shows the estimated computing power required to factor integers with current versions of the general number field sieve. (This is also roughly equal to the time it takes to compute discrete logarithms modulo a 1024-bit prime p.)

Size of integer to be factored (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}

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

Hardware Attacks

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. Van Oorschot and Wiener [27] provide a detailed study of such a possibility. In their 1994 study, 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 elliptic curve discrete logarithm in about 35 days. This is not a threat to secure implementations since ANSI X9.62 mandates that $n > 2^{160}$.

Discussion

It should be emphasized that in the software and hardware attacks described above, the 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 in order to determine another user's private key.

Blaze et al. [8] 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, we see that n should be at least 150 bits for short-term security and at least 180 bits for medium-term security. This extrapolation is justified by the following considerations:

1. 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.
2. Exhaustive searches with a symmetric-key cipher and the Pollard rho-algorithm can both be parallelized with a linear speedup.
3. 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).
4. In both symmetric-key ciphers and elliptic curve systems, a "break" has the same effect: it recovers a single private key.

6 Implementation Issues

Since the elliptic curve discrete logarithm problem appears to be harder than the discrete logarithm problem in $\mathbb{Z}_p^*$ (or the problem of factoring a composite integer n), one can use an elliptic curve group that is significantly smaller that $\mathbb{Z}_p^*$ (respectively, n). For example, an elliptic curve $E(\mathbb{Z}_p)$ with a point $P \in E(\mathbb{Z}_p)$ whose order is a 160-bit prime offers approximately the same level of security as DSA with a 1024-bit modulus p and RSA with a 1024-bit modulus n.

In order to get a rough idea of the computational efficiency of elliptic curve systems, let us compare the times to compute:

(i) kP where $P \in E(\mathbb{Z}_p)$, E is an elliptic curve, $p \approx 2^{160}$, and k is a 160-bit integer (this is an operation in ECDSA); and

(ii) $g^k \bmod p$, where p is a 1024-bit prime and k is a 160-bit integer (this is an operation in DSA).

Let us assume that a field multiplication in $\mathbb{Z}_p$, where $\log_2 p = l$, takes l^2 bit operations; then a modular multiplication in (ii) takes $(1024/160)^2 \approx 41$ times longer than a field multiplication in (i). Now, computing kP by repeated doubling and adding requires, on average, 160 elliptic curve doublings and 80 elliptic curve additions. From the addition formula we see that an elliptic curve addition or doubling requires 1 field inversion and 2 field multiplications. (The cost of field addition is negligible, as is the cost of a field squaring if the field $\mathbb{F}_{2^m}$ is used instead of $\mathbb{Z}_p$.) Assume also that the time to perform a field inversion is roughly equivalent to that of 3 field multiplications. (This is what has been reported in practice for the case of $\mathbb{F}_{2^m}$.) Then, computing kP requires the equivalent of 1200 field multiplications, or $1200/41 \approx 29$ 1024-bit modular multiplications. On the other hand, computing $g^k \bmod p$ by repeated squaring and multiplying requires, on average, 240 1024-bit modular multiplications. Thus, the operation in (i) can be expected to be about 8 times faster than the operation in (ii). It must be emphasized that such a comparison is indeed very rough, as it does not take into account the various enhancements that are possible for each system. Since multiplication in $\mathbb{F}_{2^m}$ is in fact substantially faster than modular multiplication in $\mathbb{Z}_p^*$, even more impressive speedups can be realized in practice.

Another important consequence of using a smaller group in elliptic curve systems is that low-cost and low-power consumption implementations are feasible in restricted computing environments, such as smart cards and wireless devices.

Another advantage of elliptic curve systems is that the underlying field ($\mathbb{Z}_p$ or $\mathbb{F}_{2^m}$) and a representation for its elements can be selected so that the field arithmetic (addition, multiplication, and inversion) can be optimized. This is not the case for systems based on discrete log (respectively, integer factorization), where the prime modulus p (respectively, the composite modulus n) should not be chosen to have a special form because this might render the underlying problem easy.

Summary

We have shown how ECDSA has many advantages over DSA besides the obvious one of being based on a harder problem. As ECDSA is based on DSA, we anticipate both increased understanding of the new elliptic curve technology and its advantages, and increased acceptance of the algorithm.

Acknowledgements

We thank Scott Vanstone for proposing ECDSA. We thank the ANSI X9.F1 and IEEE P1363 working groups for their work on ECDSA. We also thank Uri Blumenthal, IBM Research, for his perceptive questions of a review draft.

References

[1] L. Adleman, J. DeMarrais and M. Huang, "A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields", *Algorithmic Number Theory*, Lec-

ture Notes in Computer Science, **877** (1994), Springer-Verlag, 28-40.

[2] G. Agnew, R. Mullin, I. Onyszchuk and S. Vanstone. "An implementation for a fast public-key cryptosystem", *Journal of Cryptology*, **3** (1991), 63-79.

[3] G. Agnew, R. Mullin and S. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$", *IEEE Journal on Selected Areas in Communications*, **11** (1993), 804-813.

[4] ANSI X9.30-1, "American National Standard for Financial Services – Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry – Part 1: The Digital Signature Algorithm (DSA)", ASC X9 Secretariat – American Bankers Association, 1995.

[5] ANSI X9.30-2, "American National Standard for Financial Services – Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry – Part 2: The Secure Hash Algorithm (SHA) Revision 1", ASC X9 Secretariat – American Bankers Association, 1996.

[6] ANSI X9.62, "American National Standard for Financial Services – Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry – The Elliptic Curve Digital Signature Algorithm (ECDSA)", draft, ASC X9 Secretariat – American Bankers Association, December 1997.

[7] R. Balasubramanian and N. Koblitz, "The improbability that an elliptic curve has subexponential discrete log problem under the Menezes–Okamoto–Vanstone algorithm,", to appear in *Journal of Cryptology*.

[8] M. Blaze, W. Diffie, R. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener, "Minimal key lengths for symmetric ciphers to provide adequate commercial security", January 1996, available from `http://theory.lcs.mit.edu/~rivest/publications.html`

[9] D. Boneh and R. Lipton, "Algorithms for black-box fields and their applications to cryptography", *Advances in Cryptology – CRYPTO '96*, Lecture Notes in Computer Science, **1109** (1992), Springer-Verlag, 283-297.

[10] Certicom ECC Challenge, November 1997, `http://www.certicom.com`

[11] W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, **22** (1976), 644-654.

[12] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms", *IEEE Transactions on Information Theory*, **31** (1985), 469-472.

[13] FIPS 186, "Digital signature standard", National Institute for Standards and Technology, 1993. Available from `http://csrc.ncsl.nist.gov/fips/`

[14] FIPS 180-1, "Secure Hash Standard", National Institute for Standards and Technology, 1995 (supersedes FIPS PUB 180). Available from `http://csrc.ncsl.nist.gov/fips/`

[15] R. Flassenberg and S. Paulus, "Sieving in function fields", preprint, 1997.

[16] G. Frey and H. Rück, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves", *Mathematics of Computation*, **62** (1994), 865-874.

[17] D. Gordon, "Discrete logarithms in $GF(p)$ using the number field sieve", *SIAM Journal on Discrete Mathematics*, **6** (1993), 124-138.

[18] IEEE P1363, "Standard specifications for public-key cryptography", draft, 1997. Available from `http://stdsbbs.ieee.org/`

[19] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48** (1987), 203-209.

[20] N. Koblitz, "CM-curves with good cryptographic properties", *Advances in Cryptology – CRYPTO '91*, Lecture Notes in Computer Science, **576** (1992), Springer-Verlag, 279-287.

[21] N. Koblitz, *A Course in Number Theory and Cryptography*, 2nd edition, Springer-Verlag, 1994.

[22] R. Lercier, "Finding good random elliptic curves for cryptosystems defined $\mathbb{F}_{2^n}$", *Advances in Cryptology – EUROCRYPT '97*, Lecture Notes in Computer Science, **1233** (1997), Springer-Verlag, 379-392.

[23] A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[24] A. Menezes, T. Okamoto and S. Vanstone, "Reducing elliptic curve logarithms to logarithms in a finite field", *IEEE Transactions on Information Theory*, **39** (1993), 1639-1646.

[25] V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology – CRYPTO '85*, Lecture Notes in Computer Science, **218** (1986), Springer-Verlag, 417-426.

[26] A. Odlyzko, "The future of integer factorization", *CryptoBytes – The technical newsletter of RSA Laboratories*, volume 1, number 2, Summer 1995, 5-12. Also available from `http://www.rsa.com/`

[27] P. van Oorschot and M. Wiener, "Parallel collision search with cryptanalytic applications", to appear in *Journal of Cryptology*.

[28] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance", *IEEE Transactions on Information Theory*, **24** (1978), 106-110.

[29] J. Pollard, "Monte Carlo methods for index computation mod p", *Mathematics of Computation*, **32** (1978), 918-924.

[30] H. Rück, "On the discrete logarithm in the divisor class group of curves", preprint, 1997.

[31] T. Satoh and K. Araki, "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves", preprint, 1997.

[32] O. Schirokauer, "Discrete logarithms and local units", *Philosophical Transactions of the Royal Society of London A*, **345** (1993), 409-423.

[33] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p", *Mathematics of Computation*, **44** (1985), 483-494.

[34] R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems", *Advances in Cryptology – CRYPTO '95*, Lecture Notes in Computer Science, **963** (1995), Springer-Verlag, 43-56.

[35] I. Semaev, "Evaluation of discrete logarithms on some elliptic curves", to appear in *Mathematics of Computation*.

[36] N. Smart, "Announcement of an attack on the ECDLP for anomalous elliptic curves", 1997.

[37] A. Stein, "Equivalences between elliptic curves and real quadratic congruence function fields", presented at Pragocrypt '96.

[38] S. Vaudenay, "Hidden collisions on DSS", *Advances in Cryptology – CRYPTO '96*, Lecture Notes in Computer Science, **1109** (1996), Springer-Verlag, 83-88.

[39] J. Voloch, "The discrete logarithm problem on elliptic curves and descents", preprint, 1997.

[40] R. Zuccherato, "The equivalence between elliptic curve and quadratic function field discrete logarithms in characteristic 2", preprint, 1996.