Elliptic Curve Cryptography Support in Entrust

Author:	**Robert Zuccherato**
Date:	**May 9, 2000**
Version:	**1.0**

Entrust®

Securing Digital Identities
& Information

1 Introduction

There are three families of public-key cryptography in common use today. The most widely used systems are those based on integer factorization. In particular, the RSA cryptographic system is perhaps the most popular public-key algorithm. It is used in most web browsers (for SSL), email packages (for S/MIME) as well as within the Entrust family of products. Systems based on the discrete logarithm problem are also very popular as they can provide support for both digital signatures (with DSA) and key agreement (with the Diffie-Hellman algorithm). Traditionally Entrust has supported both of these families of cryptographic algorithms.

With the release of Version 5.0 of the Entrust product line, Entrust now also supports a third family of cryptographic algorithms. This family is based on arithmetic using elliptic curves. **Elliptic curve cryptography (ECC)** is a relatively new family of public-key algorithms that can provide shorter key lengths and, depending upon the environment and application in which it is used, improved performance over systems based on integer factorization and discrete logarithms.

This paper will give a brief introduction to elliptic curve cryptography, discuss its security and performance advantages and describe Entrust's support of this important type of cryptography.

1.1 ECC Background

The purpose of this section is to provide sufficient background material in ECC to understand the remainder of this document. For a more complete description of ECC the reader is referred to [P1363] and [X962].

All public-key cryptographic systems have their security based on certain mathematical problems that are difficult to solve. For example, RSA, since it is from the integer factorization family, has its security based on the difficulty of factoring integers. Given two large integers *p* and *q*, it is easy to compute their product

$$n = p \times q .$$

However, given a large integer *n* (where *n* has over 300 digits) it is extremely difficult, given today's resources, to factor *n* into two non-trivial factors *p* and *q*. For this reason, integer factorization is called a **one-way function**: it is easy to multiply integers, but hard to invert the operation (i.e. factor integers).

Similarly, ECC has its security based on a difficult mathematical problem. An elliptic curve can be thought of as a mathematical structure in which certain operations can be defined. These operations provide a one-way function that can be used to produce efficient cryptographic systems. The one-way function used in ECC is called the **elliptic curve discrete logarithm problem (ECDLP)**. The ECDLP is similar to the one-way function on which DSA and Diffie-Hellman are based, and hence, elliptic curve analogs of each of these algorithms have been defined. The most popular signature scheme which uses elliptic curves is called the **Elliptic Curve Digital Signature Algorithm (ECDSA)**, the most popular encryption scheme is called the **Elliptic Curve Integrated Encryption Scheme (ECIES)** and the most popular key agreement method is called **Elliptic Curve Diffie-Hellman (ECDH)**.

There are also two distinct, non-interoperable types of elliptic curve cryptography. The elliptic curve can be either **odd characteristic** (also called *modulo p* or *prime characteristic*) or **even characteristic** (also called *over a finite field with 2^m elements*). Odd characteristic elliptic curves are more suited to software implementations on a PC whereas even characteristic elliptic curves are more suited to hardware implementations.

Among even characteristic curves, there are a number of non-interoperable, different methods for representing public keys and performing arithmetic. The two most common methods are using a **polynomial basis** and using a **normal basis**.

2 Security of ECC

One of the advantages of using elliptic curve based cryptographic systems instead of integer factorization or discrete logarithm based methods is that they provide similar security levels using smaller key lengths.

Why is this? As mentioned in the previous section, the security of any public-key based cryptography is based upon the difficulty of solving certain mathematical problems. Thus, we can determine the amount of effort that would be required to break one of these public-key systems by looking at the effort required to solve these hard problems, using the best algorithms, software and hardware which are known. It should be noted that in the future new solutions to any of these problems might be discovered that drastically change the amount of effort required to solve them. The analysis below is based on the best methods known today.

Most people consider the integer factorization and discrete logarithm problems to have approximately equivalent security. Both of these problems have undergone intensive review and study by many of the world's top mathematicians and cryptographers. This can give us a sense of comfort that these problems are, in fact, difficult to solve. Actually, the best method known to solve each of these problems is the Number Field Sieve (NFS). The NFS is what is known as a **sub-exponential time method**. This means that the problem can be considered hard to solve, but not as hard as problems that only allow fully exponential solutions.

It is generally accepted that, based on the difficulty of solving the integer factorization problem and discrete logarithm problem, RSA, DSA and Diffie-Hellman keys should be at least 1024 bits long and that for very long-term security (20 years or more) 2048 bit keys should be used. Recently a large-scale effort was able to factor a 512-bit integer, thus showing that keys of this size are vulnerable to attack by large, sophisticated adversaries [RSA512].

On the other hand, solving the ECDLP is generally considered to be a much more difficult problem than factoring integers or solving the discrete logarithm problem. Because of the structure that is inherent within an elliptic curve, the types of solutions to these problems do not seem to apply to the ECDLP. The best method known to solve the ECDLP is an elliptic curve version of an attack developed by Entrust researchers for the discrete logarithm problem, known as the parallel collision search method [VOW]. This method is **fully exponential**, which means that that the ECDLP can be considered among the hardest types of problems to solve, using the best methods known today. One of the consequences of the ECDLP only having a fully exponential solution is that for every two additional bits of key used, attacking that key requires twice as much effort. Thus, attacking a 193-bit elliptic curve public key requires twice as much effort as attacking a 191-bit key.

Because it is relatively new, the ECDLP has not received as much attention from mathematicians and cryptographers as the integer factorization and discrete logarithm problem. Although, within the past few years, that has begun to change and a great deal of effort has been made at attempting to solve this problem. Since a great deal of research is still ongoing, it is difficult to directly compare the security levels provided by ECC with those provided by RSA, DSA and Diffie-Hellman, for example. However, it seems reasonable that for security equivalent to an RSA key with 1024 bits, one should use an elliptic curve with about 170 bits and that for security equivalent to an RSA key with 2048 bits, one should use an elliptic curve with about 230 bits.

The above discussion on the difficulty of attacking an ECC public key assumes that certain weak cases have been avoided when constructing the elliptic curve parameters. There are certain elliptic curves that are known to produce cryptographic systems with a substantially lower security level than the general case described above. These weak cases include:

- a class of curves known as supersingular elliptic curves;
- elliptic curves modulo p which contain exactly p points; and
- elliptic curves defined over a finite field with 2^m elements where m is not a prime.

Fortunately, each of these classes of weak curves is easy to identity and most standards bodies forbid their use. In order to guarantee that a given curve has not been intentionally constructed to somehow be weaker than expected and also to guard against possible future attacks against additional classes of weak curves, it is generally recommended to use elliptic curves which have been verifiably generated at random. This is the most conservative, or safest, option when choosing elliptic curves on which to base a system.

There exists another class of special elliptic curves which requires attention. This class of even characteristic curves, called **Koblitz curves**, allows more efficient implementations. For very resource constrained environments these curves are an attractive option. However, some cryptographers are concerned that the additional structure exploited in these curves to obtain the efficient implementations may also be used to efficiently attack them. In fact, Entrust researchers were one of two independent groups which were able to show that these curves provide a few bits less security than randomly generated curves [WZ]. While the slight weakness demonstrated should not, in itself, stop people from using these curves, it does raise the question of how secure these curves really are. Entrust recommends that these curves be used with caution.

3 Comparison with RSA

This section compares ECC public key sizes, signature and encryption lengths, and speed with those of RSA. Typical usage scenarios will be used to describe the effect these have on various implementations. The ECC system under consideration will use an odd characteristic 192-bit elliptic curve, which is the default used by the Entrust product line. The RSA system will use 1024-bit keys, which is also the default in the Entrust product line.

3.1 Size

The size of an ECC public key, ECDSA signature and an ECIES encryption will be compared with those produced by an RSA system.

3.1.1 Public Key Size

An RSA public key consists of an ordered pair *(n,e)* where *n* is a composite number, called the modulus, and *e* is the public exponent. In a 1024-bit RSA system, *n* will have 1024 bits. A common value for the public exponent is e=2^{16}+1. This is the value that Entrust uses. Thus, an Entrust RSA public key would require 128 bytes for the modulus and 3 bytes for the public exponent. The total size is then 131 bytes.

An ECC public key consists of a point on the elliptic curve. Each point is represented by an ordered pair of elements *(x,y)* each with 192 bits. For a 192-bit elliptic curve, the public key would then be represented by two 24-byte values, giving a total key size of 48 bytes.[1]

As can be seen from the numbers above, ECC does provide a significant reduction in public key size. This reduction can be crucial in many severely constrained environments where large public keys are not possible. However, in a PKI using X.509 certificates, the effect of using the smaller public keys is minimal. A typical size for an X.509 certificate would be about 1K (~1000 bytes). Thus, changing a user’s public key from RSA or DSA to ECC would reduce his/her certificate size by less than 10%.

Another important point to keep in mind is that each ECC public key is only valid in the context of certain parameters. These parameters must also be specified and transferred with integrity to the public key recipient (e.g. within an X.509 certificate). While there do exist certain curves which can be represented using short identifiers, in the general case, it will require an additional five 192-bit (24-byte) quantities to specify these parameters. Thus, it could take up to 110 additional bytes. RSA does not require any parameters be transferred with the public key.

3.1.2 Signature Size

An RSA signature consists of a single 1024 bit value. Thus, it can be represented in 128 bytes.

An ECDSA signature consists of two 192-bit values. Thus, it can be represented using two 24-byte values, for a total signature size of 48 bytes.[2]

[1] A method does exist to reduce the size of ECC public keys by almost a factor of 2. This method, called **point compression**, is a proprietary technique that is not available to all implementers of ECC, thus to ensure interoperability it is not recommended. For this reason, the size estimates given assume no point compression has been performed. If point compression was used, a public key could be represented using one 192-bit value and one additional bit. This would then require (24+1=) 25 bytes.

Again, the reduction in signature size is substantial and may be important for many constrained environments. However, as with public key size, the difference represents less than 10% of the size of a public key certificate. For larger signed messages, the difference would represent an even smaller percentage of the overall message.

3.1.3 Encryption Size

This section will compare the difference in size in transporting a 128 bit symmetric key using RSA and ECIES. This is the typical scenario when files are encrypted, for example. The encryption algorithm ECIES is specified in the ANSI X9.63 draft [X963].

A 128 bit symmetric key encrypted using RSA will consist of one 1024 bit value. Thus, it can be represented using 128 bytes.

A 128 bit symmetric key encrypted using ECIES will consist of an elliptic curve point, a 128-bit value and a 160-bit value. The elliptic curve point consists of two 192-bit values, so it can be represented using two 24-byte values, or 48 bytes.[3] The 128-bit value can be represented using 16 bytes and the 160-bit value can be represented using 20 bytes. Thus the encrypted symmetric key requires 84 bytes.

While ECIES does indeed produce smaller encrypted values than RSA, the difference is not as dramatic as for public keys and signature values. When considering that the symmetric key will then usually be used to encrypt much larger files, the advantage may become inconsequential.

3.2 Speed

This section will compare the time required to perform ECC signature and encryption operations with the time required for RSA signatures and encryption. Absolute timings for different cryptographic implementations can vary widely. These variations can be caused by a number of factors, including the quality of the implementation, the platform used, optimizations made which exploit certain special cases, or the use of proprietary or patented techniques not available to other implementers. For this reason it can often be misleading to directly compare any individual timings. It is better to consider general trends in timings for types of cryptographic systems. That is what we do here.

The first issue to consider is whether the implementation is in software or hardware. For a hardware implementation in custom silicon, even characteristic elliptic curves clearly allow the fastest implementations. This is due to the fact that the underlying arithmetic for even characteristic curves can be implemented using fewer gates, and thus in a smaller area, than the arithmetic for odd characteristic curves or for RSA. Thus, the advantage these curves have is lost if the implementation is in software or firmware on an embedded processor. Odd characteristic elliptic curves and RSA however can take advantage of the integer mathematics routines that inherently exist on most computers and therefore they should be used for software implementations. The decision as to which of these two to use in a software implementation will depend on the particular environment in which the cryptography will be used. The remainder of this section will examine some common, general software environments in which this choice must be made.

Using a small public exponent value (e.g. e=2^{16}+1), the RSA public key operations can be made very fast. This means that RSA signature verification and RSA encryption can be performed up to 40 times faster than an ECDSA verify or ECIES encryption operation. The RSA private key operations (i.e. signature generation and decryption) are generally slower than the public key operations. The ECC private key

[2] Signature sizes cannot be reduced using point compression. Thus, 48 bytes is the smallest signature size available for a 192-bit elliptic curve system.

[3] As with ECC public keys, the 48 bytes required to represent the elliptic curve point in an ECC encryption can be reduced to 25 bytes using point compression. Since this is a proprietary technique not available to all implementers of ECC, in order to ensure interoperability it is not recommended that point compression be used. For this reason, the size estimates given assume no point compression has been performed.

operations are generally faster than the public key operations. The situation can be summarized in the following table.

	RSA	ECC
Private Key Operations **(Signature Generation and Decryption)**	LESS FAST	FAST
Public Key Operations **(Signature Verification and Encryption)**	VERY FAST	LESS FAST

Let's now consider four common situations. First, let's consider the question of which algorithm would be best for CA signing keys. CA signing keys are used to sign certificates and CRLs. Each certificate and CRL only gets signed once, but is verified many times. In particular, every time that an end user's signature is verified or something is encrypted at least one certificate and CRL must be verified. Thus, it makes sense for CAs to try and minimize the amount of time end users spend on signature verification. In this situation, RSA appears to be a better choice for CA public keys.

Now let's consider end user signing keys. Each signature, regardless of the number of people to whom it will be sent or the number of times that it will be verified, will only be created once. In the general case however, it may potentially be verified many times, by many different people. Thus, it makes sense in this situation, again, to try and minimize the time that signature verification takes. Since signature generation will only be performed once, we are not as concerned about the amount of time that operation takes. Thus, again, RSA appears to be a more optimal choice for end user signature keys.

Let's now look at what happens with end user encryption keys. When encrypting a document, the actual document will get encrypted with a symmetric algorithm (e.g. DES, CAST, IDEA) and then the symmetric key will be individually encrypted using the public-key algorithm for each recipient. Thus, the person encrypting the document must perform a separate public-key encryption for each recipient and therefore may need to perform many public-key encrytions each time a document is encrypted. Each recipient, however, will only need to perform one decryption operation each time access to the document is required. Thus, in this situation it makes sense to try and minimize the amount of time the person encrypting the document must spend and, therefore, minimize the amount of time spent on encryption. Again, using RSA seems to make the most sense in this situation.

Finally, let's consider a general email application where a user wishes to digitally sign and encrypt an email for one recipient. Digitally signing the email requires one private key operation. Encrypting the email requires one public key operation to encrypt the symmetric key which will be used to encrypt the contents of the email. However, before using the recipient's encryption public key, the sender must first validate the recipient's public key certificate. Validating the certificate will involve verifying the signature on at least one certificate and also verifying the signature on at least one CRL. Thus, at least three public key operations must be performed before the email can get sent. In this situation, the difference between the time required for an RSA-based system and an ECC-based system is very small. Either choice will result in a very fast implementation.

4 Entrust Support for ECC

This section will describe Entrust's present and future support for ECC. In the 5.0 release of the Entrust product line, support for ECDSA was introduced. Initially only end user ECDSA keys are supported, but support for CA ECDSA keys is expected in a future release. Thus, Entrust now supports all three major digital signature algorithms, RSA, DSA and ECDSA. At the present time Entrust does not support either ECIES encryption or ECDH. These algorithms will be supported when market demand warrants.

Entrust's elliptic curve implementation is completely general. Any odd or even characteristic elliptic curve (that is not a member of one of the weak classes described in Section 2) is supported. However, in order to make things easier for the user and to promote interoperability, Entrust has chosen certain default curves that will be used if no other curve is specified. Since the environment is software based, Entrust has chosen odd characteristic elliptic curves to be the default type of curve.

As was mentioned in Section 2, a 170-bit elliptic curve system seems to provide approximately equivalent security to a 1024-bit RSA system. Since elliptic curves are relatively less studied than integer factorization or discrete logarithms, Entrust felt it was prudent to choose default curves which provide slightly more security than required, to allow for future advances in cryptanalytic attacks. The default odd characteristic curve has 192 bits and the default even characteristic curve is 191 bits. Both of these default curves are specified in ANSI X9.62 [X962] and can be represented by standardized short identifiers. In fact, Entrust recognizes all of the identifiers specified in the ANSI X9.62 standard and is X9.62 compliant. For even characteristic curves, however, Entrust only supports curves represented with respect to a polynomial basis. Normal bases are not supported.

5 References

[P1363] IEEE Std 1363-2000: Standard Specifications for Public-Key Cryptography.

[RSA512] “Factorization of RSA-155”, available at
http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html

[VOW] P.C. van Oorschot and M.J. Wiener, Parallel collision search with cryptanalytic applications, *Journal of Cryptology*, **12** (1999), pp. 1-28.

[WZ] M.J. Wiener and R.J. Zuccherato, Faster attacks on elliptic curve cryptosystems, *Selected Areas in Cryptography- 5th Annual International Workshop, SAC '9*8, Lecture Notes in Computer Science **1556** (1999), Springer-Verlag, pp. 190-200.

[X962] ANSI X9.62-1998, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[X963] ANSI X9.63, Public Key Cryptography for the Financial Services Industry: Elliptic Curve Key Agreement and Transport Protocols, draft, 1999.