



Elliptic Curve PKI

An exploration of the benefits and challenges of a PKI based on elliptic curve cryptography

February 2008

Entrust is a registered trademark of Entrust, Inc. in the United States and certain other countries. Entrust is a registered trademark of Entrust Limited in Canada. All other company and product names are trademarks or registered trademarks of their respective owners. The material provided in this document is for information purposes only. It is not intended to be advice. You should not act or abstain from acting based upon such information without first consulting a professional. ENTRUST DOES NOT WARRANT THE QUALITY, ACCURACY OR COMPLETENESS OF THE INFORMATION CONTAINED IN THIS ARTICLE. SUCH INFORMATION IS PROVIDED "AS IS" WITHOUT ANY REPRESENTATIONS AND/OR WARRANTIES OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, BY USAGE OF TRADE, OR OTHERWISE, AND ENTRUST SPECIFICALLY DISCLAIMS ANY AND ALL REPRESENTATIONS, AND/OR WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT, OR FITNESS FOR A SPECIFIC PURPOSE.

© Copyright 2008 Entrust. All rights reserved.

Table of Contents

1	Introduction	1
2	Foundation of ECC.....	2
3	Juxtaposition of ECC, RSA	3
	3.1 Cryptographic Strength.....	3
	3.2 Data Block Size.....	5
	3.3 Performance	7
4	Planning for the Future.....	9
5	Deploying an ECC PKI	9
	5.1 Interoperability	9
	5.2 Transitioning to ECC.....	13
6	The Decision Process.....	14
7	References.....	16
8	About Entrust	18

1 Introduction

Public-key cryptography has achieved enormous success in protecting users from identity theft resulting from transactions on the Web, e-mail correspondence and VPN sessions. But, it is well accepted in information security circles that cryptographic algorithm strength erodes with time, as the computing power available to cryptanalysts increases and the cryptanalytic techniques they use improve. Security architects respond to this trend by specifying larger key sizes, and since the computing power available to the legitimate user also increases with time, the attendant performance degradation is — at least partially — offset.

The impact on performance of increased key size is different for each public-key algorithm. And, although RSA is by far the most widely used public-key algorithm, it suffers serious performance impacts when key sizes increase much beyond those in common use today. For this reason, depending upon the characteristics of the operating environment, it is worthwhile considering switching to an algorithm with more favorable characteristics at the security levels that will become common in years to come.



There are three mainstream families of public-key algorithms. As noted above, the most widely used systems are those based on integer factorization; in particular, the RSA cryptographic system. Systems based on the discrete logarithm problem are also very popular and can provide support for both digital signatures (with the Digital Signature Algorithm) and key agreement (with the Diffie-Hellman algorithm).

There is a third family of public-key algorithms that is rising in importance as key sizes for the other two families increase. This family is based on arithmetic using elliptic curves. First described in 1985, Elliptic curve cryptography (ECC) is a relatively new family of public-key algorithms that can provide shorter key lengths and, depending upon the operating platform and the applications for which they are used, may provide improved performance over systems based on integer factorization and discrete logarithms.

This paper gives a brief introduction to elliptic curve cryptography, discusses its security and performance characteristics and considers the advantages of, and challenges associated with, rolling out, or transitioning to, a PKI based on ECC.

2 Foundation of ECC

The purpose of this section is to provide a background in ECC sufficient to understand the remainder of the paper. For a more complete description, the reader is referred to [\[P1363\]](#), [\[SEC1\]](#), [\[SEC2\]](#), [\[X962\]](#) and [\[X963\]](#).

The security of all public-key cryptographic systems is based on a mathematical problem that is difficult to solve. For example, the security of RSA is based on the difficulty of factoring large integers.

Given two large integers, p and q , it is easy to compute the product:

$$n = p \times q$$

However, given a large integer n (where n has more than 300 digits) it is extremely difficult, given current knowledge and resources, to factor n into its 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 large integers). This hard problem is called the integer factorization problem.

Similarly, the security of ECC is based on a difficult mathematical problem. An elliptic curve can be thought of as a mathematical structure called a “group,” consisting of a set of elements and an operation on pairs of those elements. The number of elements in the group is often referred to as the size of the ECC cryptographic system, even though it is not exactly the size of the ECC public key or signature. The group operation can be used to create a one-way function that can, in turn, form the basis of an efficient cryptographic system.

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 (which is called simply the discrete logarithm problem), and hence, elliptic curve analogs of each of these algorithms have been defined. The most popular signature scheme that uses elliptic curves is called the Elliptic Curve Digital Signature Algorithm (ECDSA), and the most popular key agreement scheme is called Elliptic Curve Diffie-Hellman (ECDH).

There are also two distinct, non-interoperable, types of elliptic curve cryptography. Elliptic curve cryptography 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” or “binary”). Odd characteristic elliptic curves are well suited to software implementations on a PC, whereas even characteristic elliptic curves are better suited to custom hardware 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 of silicon, than the arithmetic for odd characteristic curves or for RSA.

Because the underlying arithmetic operation for even characteristic ECC is not well suited to a general-purpose processor, the advantage these curves offer is lost when the algorithm is implemented in software or firmware. Odd characteristic elliptic curves and RSA, however, can take advantage of the integer mathematics routines that exist on most general-purpose processors.

For a hardware implementation in custom silicon, even characteristic elliptic curves clearly achieve lower cost than their odd-characteristic counterparts. However, odd-characteristic curves are the most widely used, and therefore they have some advantages when considering interoperability. In addition, most researchers tend to feel that odd characteristic curves are more resistant to attack than even characteristic curves.

Of course, where software implementations are required to interoperate with constrained devices that implement only even-characteristic curves, those same even-characteristic curves must be implemented on the software platforms as well. But, where practical, it is recommended that odd characteristic elliptic curves be chosen.

3 Juxtaposition of ECC, RSA

This section compares the public key sizes, signature and encryption block sizes and operating speeds for the ECC and RSA algorithms, when implemented at equivalent cryptographic strength. Typical usage scenarios are used to explore the implications of these characteristics in a variety of implementation environments. But, first we must determine the key size required by each algorithm for a specified cryptographic strength.

3.1 Cryptographic Strength

Probably the most well-accepted guidance on the cryptographic strength required for a particular application exists in the NIST key management guidelines [KMG]. Table 4 of the guidelines contains recommendations for cryptographic strength, based on the time for which protection is required to be maintained, and part of that table is reproduced here.

Table 1: Cryptographic Strength

Protection period	Strength (bits)
Up to 31 Dec 2010	80
Up to 31 Dec 2030	112
Beyond 31 Dec 2030	128

Cryptographic strength is measured in “bits,” as the key size of an “ideal” algorithm. An ideal algorithm is one for which no more efficient attack strategy exists than exhaustive search of the key space. Because more effective attacks than exhaustive search exist for all public-key algorithms, they are not “ideal,” according to this definition. So, they require keys of greater length than the cryptographic strength they provide.

As mentioned earlier, these recommendations apply to **end-user** keys, and NIST further recommends that the size of CA keys and end-user keys that protect particularly sensitive data should be set at the 112-bit level, even today.

These recommendations were arrived at by taking into account Moore’s Law and projected improvements in cryptanalytic techniques that were extrapolated from past experience. Since these recommendations are consistent with current research, we will adopt this table of equivalences and timelines for our present purposes.

As mentioned in the previous section, the security of any public-key cryptographic system is based upon the difficulty of solving a hard mathematical problem. Thus, we can determine the effort required to break public-key systems by looking at the effort required to solve the associated hard problem — using the best algorithm, the best software and the best hardware available today.

It should be noted, however, that at any time in the future, new solutions to any or all of these problems may be discovered, and those discoveries may drastically reduce the amount of effort required to solve the associated problem. But, the analysis presented here is based on the best methods known today.

There is general agreement that the integer factorization and discrete logarithm problems provide approximately equivalent security at the same key size¹. Both of these problems have undergone intensive review and study by many of the world's top mathematicians and cryptographers. So, it is widely accepted that they are, in fact, difficult to solve.

Actually, the best known method for solving both of these problems is the Number Field Sieve (NFS). But, the NFS is what is known as a sub-exponential time method. This means that, as the size of the number grows, more and more bits must be added in order to achieve the same relative increase in difficulty. So, these problems can be considered hard, but not as hard as problems for which there exist only fully exponential solutions.

Solving the ECDLP is generally considered to be much more difficult than factoring integers or solving the discrete logarithm problem. Because of the different structure that is inherent in an elliptic curve, the solutions available for factorization and solving discrete logarithms cannot be applied to the ECDLP. The best methods for solving the ECDLP are elliptic curve variants of a class of attacks on the general discrete logarithm problem, and they take time proportional to the square-root of the size of the group. That is, in order to double the cryptographic strength, two bits have to be added to the key, regardless of the key size.

So, attacking a 193-bit elliptic curve requires twice as much effort as attacking a 191-bit curve. The increased effort may translate to the cost of the computing equipment required, to the time it takes to complete the attack or to some combination of the two. Because the best methods for attacking the ECDLP are fully exponential, the ECDLP can be considered among the hardest problems to solve

A segment of Table 2 of the NIST recommendations, below, shows the key size required to achieve a specified cryptographic strength for both ECC and RSA.

Table 2: Required Key Sizes

Strength (bits)	ECC Size (bits)	RSA Size (bits)
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Taking the two tables together, one can see that keys used for short-term protection of end-user data should, today, be 160 bits for the ECC algorithm and 1024 bits for the RSA algorithm.

Over the past few years there have been a number of proposals for devices that are specifically designed to perform parts of the NFS [S], [B] and [ST]. These proposals do not affect how the NFS works (it would still be a sub-exponential attack). However, if these devices were ever to be built, they would significantly decrease the amount of time required to attack RSA keys.

For example, it was conjectured in [B] that the proposed machine could increase the size of keys that can be successfully attacked by a factor of 3. Thus, RSA keys of more than 1500 bits could be vulnerable today. Likewise, in [ST] it was estimated that the proposed machine could attack a

¹ The discrete logarithm problem can be solved either by methods very similar to those that solve the integer factorization problem or by methods very similar to those that solve the ECDLP. And, which of these two types of attack on the discrete logarithm problem are more effective depends upon the particular instance of the problem being solved. However, for the remainder of this paper we will assume that the effort required to solve the discrete logarithm problem and the effort required to solve the integer factorization problem are approximately the same.

1024-bit RSA key in one year for a \$20 million one-time design/start-up cost and a \$10 million manufacturing cost. None of these proposals represents an actual attack against RSA. They are merely well-thought-out approaches. However, the potential for dramatic improvements in factorization over the next few years exists, and this must be taken into account when evaluating the security of RSA.

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 that 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 identify 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 that have been verifiably generated at random. This is the most conservative, or safest, option when choosing elliptic curves on which to base an implementation.

There exists another class of special elliptic curves that deserve a mention. This class of even characteristic curves, called Koblitz curves, allows for more efficient implementation. 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 improved efficiency may also be used to attack them more efficiently.

In fact, Entrust researchers were one of two independent groups² that were able to show that these curves provide a few bits less security than randomly generated curves [WZ]. While this slight weakness should not, in itself, stop people from using these curves, it does raise the question of how secure they really are. We recommend that they be used with caution.

ANSI [X962] and NIST [FIPS186] have collaborated to address concerns about inadvertently choosing curves with known weaknesses by publishing a set of curves of each type (odd and even characteristic and Koblitz curves) for use by U.S. financial institutions and government agencies. The set spans the range of security levels described in the NIST key management guidelines. The U.S. National Security Agency (NSA) has chosen a subset of the NIST curves for the suite of algorithms it calls "Suite B." Suite B is approved by NSA for protecting national security information.

3.2 Data Block Size

In this section, the size of an ECC public key and ECDSA signature block will be compared with their RSA equivalents.

3.2.1 Public Key Size

An RSA public key is an ordered pair (n,e) , where n is a composite number, called the modulus, and e is called the public exponent. In a 1024-bit RSA system, n has 1024 bits. A common value for the public exponent is $e = 2^{16} + 1$. Thus, a 1024-bit RSA public key requires 128 bytes for the

² The results were independently discovered by Gallant, Lambert and Vanstone. See [GLV].

modulus and 3 bytes for the public exponent. The total size is then 131 bytes. Similarly, in a 15,360-bit RSA system, the public key would require 1,923 bytes for the public key.

An ECC public key is a point on the elliptic curve. Each point is represented by two coordinates (x,y), both of which are the same size as the underlying finite field. Usually, this is the same as the size of the curve. Thus, for a 160-bit elliptic curve, the public key consists of two 20-byte values, giving a total key size of 40 bytes. Similarly, a 512-bit elliptic curve has a public-key size of 128 bytes.³

As can be seen from the numbers given above, ECC can provide a significant reduction in public key size, particularly at higher cryptographic strengths. This reduction can be important in many severely constrained environments, where large public keys cannot be supported, or where certificate size is critical.

In a PKI based on X.509 certificates, the effect of including the smaller public key is less dramatic. A typical size for an X.509 certificate is about 1K (~1000 bytes). So, changing an end-user's public key from 2048-bit RSA or DSA to 224-bit ECC reduces the certificate size by approximately 20 percent.



The sender and receiver in an ECC PKI must agree on the curve parameters. And, these may be conveyed with the public key in the certificate. This approach further reduces the size benefit of ECC. However, there are two alternative approaches, which can restore that benefit. The first uses certificate fields to inform the verifier that the curve parameters for the end-user key are the same as those for the CA's key. This approach offers a benefit when the entire PKI is based on ECC and cryptographic strength does not increase as one traverses the certificate path toward the root⁴. These conditions are only likely to apply in very special circumstances. So, this approach is not likely to be widely followed.

The second alternative is likely to be more popular, and it allows curves and their parameters to be specified by a standard and simply identified in the certificate. This approach maximizes the size benefit of ECC.

3.2.2 Signature Block Size

An RSA signature created with a 1024-bit public key consists of a single 1024-bit value. Thus, it can be represented in 128 bytes. Similarly, an RSA signature created with a 15,360-bit public key requires 1,920 bytes.

An ECDSA signature created using a 160-bit curve consists of two 160-bit values. Thus, it can be represented using two 20-byte values, for a total signature size of 40 bytes⁵. Similarly, a signature created using a 512-bit curve requires 128 bytes.

³ A method does exist to reduce the size of an ECC public key by almost a factor of 2. This method, called point compression, has not been widely implemented, and so it is not recommended when interoperability is a consideration. For this reason, the size estimates given here assume no point compression has been applied. However, if point compression were to be applied, a 160-bit public key could be represented as one 160-bit value and one additional bit. This would then require (20+1=) 21 bytes. A 512-bit public key could likewise be represented by (64+1=) 65 bytes.

⁴ It is worth noting that different signature algorithms can be used at different levels in the PKI. The only drawback is that the relying party software must be capable of processing the signature algorithms it encounters at all levels.

⁵ The signature is not a point on the curve, so its size cannot be reduced by point compression.

Again, the reduction in signature size, especially for higher cryptographic strengths, is substantial, and may be important in constrained environments, where small certificates are essential or small amounts of data are being signed. As before, the benefit for smaller key sizes may represent approximately 20 percent of the size of an X.509 public-key certificate. These benefits are cumulative, though. So, if both the CA and the end-user use ECC, then the end-user's certificate may be reduced in size by as much as 40 percent.

For larger signed messages, like an e-mail message, for example, the difference may represent an insignificant percentage of the overall message size.

3.3 Performance

This section compares the computing resources required to perform ECC signature and encryption operations with the computing resources required for the equivalent RSA operations, in terms of the relative time it takes to perform the operations. Absolute timings for different cryptographic implementations can vary widely. And these variations can be caused by a number of factors, including the quality of the implementation, the characteristics of the execution platform, optimizations made to exploit certain special cases and the use of proprietary or patented techniques that may not be available to all implementers.

In addition, public- and private-key operations differ in efficiency: for RSA, public-key operations are significantly more efficient than private-key ones, whereas for ECC, private-key operations are slightly more efficient than public-key ones. Therefore, the performance of one algorithm relative to another depends upon the profile of operations used by the application. Profiles that involve significantly more public-key operations than private-key and key generation operations will favor RSA over ECC. Other mixes will tend to favor ECC.

For these reasons, it can be misleading to compare empirical timing values for different algorithm implementations. So, for present purposes, we consider relative timings for each cryptographic system at a range of strengths.



Let's compare the amount of time required to do two important operations (signature generation and key generation) for both ECDSA and RSA, at various key sizes. Since absolute comparison is difficult, as mentioned above, we will represent the amount of time required for each of these operations at the 80-bit security level by the constants *A*, *B*, *C* and *D*. For the purposes of this comparison, we will assume a naïve implementation of each of the algorithms. In reality, state-of-the-art implementations can achieve better performance. But, this comparison is sufficient to indicate the relative magnitude of the differences.

Each of the operations of RSA signature generation, ECDSA signature generation and ECDSA key generation have running times that increase approximately with the cube of the key size. The running time of RSA key generation, on the other hand, increases with the key size raised to the power of 4. This gives the following comparison of running times as key size varies.

Table 3: Key Generation Running Times

ECC Group Size (in bits)	ECDSA Sig. Time	ECDSA Key Gen. Time	RSA Key Size (in bits)	RSA Sig. Time	RSA Key Gen. Time
160	A	B	1024	C	D
224	2.7·A	2.7·B	2048	8·C	16·D
256	4.1·A	4.1·B	3072	27·C	81·D
384	13.8·A	13.8·B	7680	422·C	3,164·D
512	32.7·A	32.7·B	15360	3,375·C	50,625·D

Notice that, as key sizes increase from the 80-bit level to the 256-bit level, the amount of time required to perform an ECDSA signature generation or key generation operation increases by a factor of almost 33. The running times for RSA, on the other hand, increase by a factor of more than 3,000 for signing and more than 50,000 for key generation. The reason for the difference between RSA and ECDSA lies in the existence of a sub-exponential attack on RSA. This rapid increase is then cubed for signature generation and raised to a power of 4 for key generation.

As key sizes increase over time, performance degrades. But, even the dramatic degradation associated with RSA would be more than compensated for by the operation of Moore's Law. Such a conclusion is only valid, however, if Moore's Law continues to operate for several decades into the future, which is not, by any means, a certainty.

As was mentioned above, direct comparison of performance between different cryptographic families is difficult. However, it is interesting to estimate where, approximately, the crossover point in performance between RSA and ECC lies. For application profiles that make heavy use of public-key operations, compared to private-key and key generation operations, the crossover point is around the 128-bit cryptographic strength level (i.e., 3072 bits for RSA and 256 bits for ECC). This profile is the one that shows RSA in the most favorable light. So, the crossover point occurs at a lower strength level for all other profiles.

It is worth noting that the experts at the U.S. Government's National Security Agency have determined that beyond the 1024-bit public-key systems in common use today, rather than increase key size, a switch to ECC is warranted.

For many severely constrained environments the crossover point is also lower than 128 bits. Furthermore, there exist a number of platforms that cannot support the large integer arithmetic required for even moderately sized RSA operations (1024- or 2048-bit). And, in these environments, RSA is not an option.

4 Planning for the Future

Cryptanalytic computing power increases and cryptanalytic techniques improve over time. So, cryptographic key sizes increase inexorably. This makes it unlikely that today's 1024-bit RSA end-user keys will still be considered secure 30 years from now. However, the rate at which key sizes increase varies by algorithm. And, barring any unforeseen developments, RSA key sizes will increase at a faster rate than those of ECC.

As key sizes increase, so do the sizes of signatures and public keys, and so does the time required to perform cryptographic operations on a particular computing platform. This rate of increase will be considerably faster for RSA than it is for ECC.

If data only had to be protected for a short period of time, the increasing key size would not be an immediate concern: we could simply upgrade to a larger key size or a new algorithm when the protection provided by the current keys and algorithm becomes inadequate. However, in many applications, sensitive information must be protected for a very long period of time (sometimes as much as 30 years or more).

Also, unless integrity and confidentiality protection are renewed periodically, the certificates and revocation information in use today must remain secure for the same period of time.

Furthermore, to the extent that it is possible, we should engineer systems to accommodate the types of end-user devices that will come into use during the lifetime of the system. In other words, for applications with long-term protection requirements, we need to deploy systems today that will retain their usefulness and remain secure for periods of time possibly as long as 30 years. Clearly, 1024-bit RSA cannot satisfy this requirement, and we are driven to consider ECC as an alternative.

5 Deploying an ECC PKI

This section examines various topics relating to the deployment of a PKI based upon ECC. As we have seen, ECC has some theoretical advantages over RSA. But, is there a price to pay? This section focuses on the practical considerations. These include interoperability issues, specific requirements of an ECC infrastructure and the challenges of transitioning from an RSA infrastructure to one based upon ECC.

5.1 Interoperability

In order for a public key infrastructure (PKI) to be truly an infrastructure, it must be transparent and seamless. This places onerous requirements, not only on its usability, but also on its integration with applications and its internal and external interfaces. This section examines these integration and interoperability issues. First we explore the extent of support for ECC in applicable standards. Then we look at the availability of products that can be used with an ECC PKI. Finally, we review the issues that may arise when interoperating across domain boundaries.

A great deal of effort has been devoted in recent years to the standardization of elliptic curve cryptography. This effort continues. However, there are still some areas where work remains to be completed. Even where the standardization work has been completed, the results have not fully flowed through into commercial products. Until these deficiencies have been addressed, it will not be a straightforward matter to deploy a general-purpose ECC infrastructure.

In the following sections, we review the state of security standards and the extent to which they incorporate elliptic curve cryptography.

5.1.1 Standards

Standard Curves

RSA is a specific cryptographic transformation, described only by the key. Elliptic curve cryptography, on the other hand, describes a **class** of cryptographic transformations, a particular instance of which is described by a set of parameters, in addition to the key. The parameters describe the particular curve and the particular underlying finite field. So, in order for products to interoperate using ECC, in addition to all of the factors that must normally be agreed between the communicating parties, it is necessary to agree a particular member of the class of ECC transformations.

This additional need for agreement has the potential to severely hamper interoperability. But, the situation is not as ominous as it might seem. The U.S. National Institute for Standards and Technology has published a set of ECC algorithms that satisfies a broad range of applications, and a broad range of security levels [[FIPS186](#)]. This suite of algorithms has been the focus of standards development related to ECC. Members of the suite represent all of the various security levels that are included in the NIST table of strengths, odd and even characteristic curves, Koblitz curves and randomly generated curves. Security architects should select from the set of NIST algorithms.

Cryptographic Standards

For signatures, the most widely used and standardized algorithm is the Elliptic Curve Digital Signature Algorithm (ECDSA). It is the ECC analog of the Digital Signature Algorithm (DSA) and is currently specified in a number of international standards ([\[X962\]](#), [\[P1363\]](#), [\[ISO14888-3\]](#) and [\[ISO15946-2\]](#)) as well as the NIST specification [[FIPS186](#)].

The situation is similar for ECC key agreement algorithms, in which the Elliptic Curve Diffie-Hellman (ECDH) algorithm is the most widely used key agreement algorithm and is specified in a number of international standards ([\[X963\]](#), [\[P1363\]](#) and [\[ISO15946-3\]](#)). There is one complication, however. These standards define different and incompatible ways of deriving the symmetric key from the result of the public-key operation. Fortunately, most application-level standards specify which key derivation function to use, and this effectively enables interoperability.

There is one additional algorithm that has received quite a bit of support for key agreement. The Elliptic Curve Menezes-Qu-Vanstone (ECMQV) algorithm is not very widely used, but it is allowed in many standards ([\[X963\]](#), [\[P1363\]](#) and [\[ISO15946-3\]](#)), and it can provide some performance advantages in some environments. Besides being not as widely supported, the main drawback of this algorithm is that it requires both parties to have a static, certified key pair. Thus, it is not as practical for most applications as ECDH, and it is likely to cause interoperability problems. These problems will primarily arise in mixed algorithm environments and when only one of the entities has a certified public key (e.g., SSL server authentication). Therefore, it is not recommended that ECMQV be used.

Message-based ECC encryption algorithms typically involve the sender performing an ephemeral-static ECC Diffie-Hellman key agreement and then encrypting the data with the resulting shared key and a symmetric encryption algorithm. The Elliptic Curve Integrated Encryption Scheme (ECIES) is perhaps the most popular scheme currently specified ([\[X963\]](#), [\[P1363a\]](#)). However, there are a number of different schemes for ECC encryption, including those based on ephemeral-static ECC Diffie-Hellman using the derived key to protect a per-message AES key. It is recommended that implementations use ECDH and the AES Key Wrap method.

PKI Standards

The Internet Engineering Task Force (IETF) Public-Key Infrastructure-X.509 (PKIX) Working Group specifies the X.509 PKI for the Internet and is the primary standards body for PKI operational standards. It profiles X.509 certificates and develops protocols to support them. Algorithms and identifiers for PKIX are specified in [\[RFC3279\]](#) and [\[RFC4055\]](#), which contain sections on how to include ECDSA and ECDH public keys in certificates, and specify how to use ECDSA signatures. Following these RFCs should enable interoperability at the PKI operational level, and should be followed when certifying ECC public keys in an X.509 PKI.

S/MIME

The IETF S/MIME Mail Security Working Group specifies Internet standards for secure e-mail based upon PKCS #7/Cryptographic Message Syntax [\[RFC3852\]](#). In addition, [\[RFC3278\]](#) specifies the use of ECC algorithms in CMS. It allows the use of ECDSA, ECDH and ECMQV, with encryption obtained by combining ECDH or ECMQV with CMS key wrapping, which is specified in [\[RFC3852\]](#), and other RFCs, depending on the symmetric cipher used. Thus, the state of standards for ECC in S/MIME is stable and [\[RFC3278\]](#) should be followed for interoperability purposes.

SSL/TLS

The IETF Transport Layer Security (TLS) Working Group defines the TLS protocol that is based upon the Secure Sockets Layer (SSL) protocol. There is an Internet-RFC that describes how to integrate ECC with TLS (see [\[RFC4492\]](#)). It is recommended that solutions that use ECC for TLS be based on this RFC.

For wireless applications, the Open Mobile Alliance (OMA) WTLS protocol supports ECC and is quite stable [\[WTLS\]](#). This standard can be implemented without risk of premature obsolescence.

XML Signatures and Encryption

The W3C XML Signature and XML Encryption Working Groups were responsible for defining how to sign and encrypt documents (or portions of documents) expressed in Extensible Markup Language (XML). These groups have both advanced all of their charter deliverables and have basically ceased operation. The functionality they specify is extensively used for securing Web services through standards such as WS-Security [\[WSS\]](#). The use of ECC signatures is described in [\[RFC4050\]](#). There is no active document describing how to use ECC encryption with XML encryption and thus this combination should be avoided until this deficiency has been resolved.

IPSEC

The IETF IP Security Protocol (IPSEC) Working Group is responsible for providing security at the IP layer. [\[RFC4306\]](#) (Internet Key Exchange, or IKE) is widely used as the protocol for authentication and key establishment within IPSEC. Thus, most Virtual Private Networks (VPNs) in use today use this protocol. The use of ECDH and ECDSA are fully supported for key agreement and digital signature within IKE ([\[RFC4753\]](#), [\[RFC4754\]](#) and [\[RFC4869\]](#)).

PKCS #11

The Public-Key Cryptography Standard (PKCS) #11 [\[PKCS11\]](#) describes an API for accessing devices that hold cryptographic information and perform cryptographic functions. This is the most popular standard for interfacing with cryptographic tokens, smart cards and hardware security modules (HSMs). PKCS #11 specifies an API for the use of ECDSA signatures and ECDH or ECMQV key agreement. Thus, PKCS #11 should be used in order to achieve interoperability when accessing an ECC-enabled cryptographic token.

5.1.2 Support for ECC in Commercial Products

All the major CA software products support ECDSA, both for certificate and CRL signing and for end-user public keys. So, for applications that only require authentication and digital signatures, it should not be difficult to source a suitable CA product. The slower pace of standardization for ECC-based key agreement adds some uncertainty for applications that also require encryption. However, the major CA software suppliers all have advanced plans to support ECDH keys in end-user certificates. So, planning in this area carries only minor risk. Implementation, on the other hand, must await realization of these plans in shipping products.

Many applications, particularly those that extend beyond the enterprise, rely on a Web browser for end-users to access resources securely. The major browsers and operating systems all provide support for ECC in their most recent versions. But, there is still a sizable user population using older versions of these products. And it will be many years before there is complete support for ECC across the general user population.

Even the browser versions that do support ECC currently do not have an embedded ECC root certificate (although this situation is just starting to change). So, either an ECDSA root key has to be imported, or one of the existing embedded RSA roots will have to be used. Effectively, this means that ECC is not yet a viable option for securing consumer applications. And this situation is likely to persist for many years. Enterprise and extranet applications, on the other hand, can more readily be secured with ECC today.



ECC does not yet enjoy the same level of support amongst security-enabled commercial applications as RSA does. However, this situation is changing rapidly, and any discrepancy will probably be eliminated within a couple of years. For in-house applications, ECC toolkits are available from the major CA software suppliers and from toolkit specialists. As previously mentioned, support for signatures is more mature than support for encryption, and it will likely take a couple of years before this inequality is fully eliminated.

Many applications that run on versions of Microsoft's Windows OS rely on the services of the Microsoft Cryptographic API (CAPI). CAPI is a framework into which Cryptographic Service Providers (CSPs) can be plugged. And, it is the CSPs that actually perform the cryptographic operations. The framework supports the use of ECC for signatures and encryption, and the CSP that ships with the OS includes support for some of the "Suite B" ECC algorithms. Other CSPs that support ECC are also available.

Similarly, Java applications that include security operations rely upon a Cryptographic Package Provider, or "crypto provider" for short, to supply the necessary cryptographic functions. The API provided in the Java Cryptography Architecture (JCA) and the Java Cryptography Extensions (JCE) supports ECC, but the default crypto provider does not. There are, however, a number of crypto providers that do include ECC functions.

Some applications, particularly those within the enterprise, may require the use of cryptographic tokens, usually smart cards, for secure credential storage and two-factor authentication. As already mentioned, PKCS #11 fully supports ECC, and so the necessary standards are in place to provide ECC functionality on a smart card. However, there are only a small number of smart card products that implement ECDSA and ECDH. So, care must be taken in choosing a supplier. The same considerations apply to the choice of an HSM.

The above discussion of available end-user applications is focussed on the desktop. However, when considering other types of end-user devices (e.g., PDAs, mobile phones) the situation improves somewhat. There are a number of toolkits available for implementing ECC-enabled applications on mobile devices, and many devices actually come with built-in support for ECC. The more efficient implementations that are possible with ECC make it quite an attractive choice for these devices.

So, it is much more likely that ECC-enabled applications will be available for constrained devices than for PCs. Therefore, an ECC PKI is more feasible today if it is primarily intended to serve constrained devices. It should be anticipated, however, that ECC capabilities will vary from platform to platform.

5.1.3 Interdomain Communication

As we have seen, some uncertainty exists over implementing ECC within a closed domain if a variety of standard-compliant commercial products are to be used. The degree of uncertainty increases, however, if communication is required across domain boundaries with users and devices whose ECC capabilities are either unknown or uneven.

The situation eases if all parties implement only standard ECC parameters. And this is likely to be the popular approach. But, proprietary parameters may still be encountered. So, it would be prudent to investigate this aspect, to the extent that it is possible, early in the planning phase.

In case remote end-systems that do not support ECC should be encountered, local end-systems should retain the capability to communicate using RSA, DSA and Diffie-Hellman. Such systems are not only required to support all these algorithms, they must also have certified keys for each.

5.2 Transitioning to ECC

There are a number of factors that have to be considered when planning a transition from an RSA environment to an ECC infrastructure. This section highlights some of these factors.

Most modern security infrastructures are designed to be algorithm-independent. So, in theory, it is possible to switch from one algorithm to another with relative ease. In reality, a great many details have to be aligned, and achieving this alignment can consume a large amount of effort. Therefore, from the outset, one must ensure that the benefit justifies the effort.

Once it has been determined that a transition to ECC is justified, all of the considerations outlined earlier must be taken into account: the key sizes and curves must be chosen; the applications to be supported must be catalogued and investigated; and the necessary infrastructure products, application software and hardware must be identified, sourced, tested and deployed.

The prudent architect will replicate the operational environment in a laboratory setting in order to confirm the compatibility of the chosen products and rehearse the transition.

Software capable of encryption and signature verification using ECC should be deployed to all users before ECC keys are issued and data is encrypted or signed. Then, ECC should be enabled for users in a phased manner, so that teething troubles can be identified before a large user-population is affected. Preparations such as help-desk training and updating affected policies and practices should be completed prior to the system going live.

Because the deployment of ECC-enabled software takes place before any ECC keys are issued, it is necessary that all new applications be capable of processing the existing crypto algorithms in

addition to the ECC ones. And if the planned architecture contains devices that only support ECC, then these cannot be introduced until the later stages of the rollout.

Once all end-user software and hardware have been deployed, we are ready to make the transition. This can be achieved as a normal key rollover operation. CAs typically accomplish this by using the old (RSA) root key to create a link certificate containing the new (ECC) root key, and similarly using the new root key to create a link certificate containing the old root key. Users that already trust the RSA root key can then verify all future certificates created by the ECC root key, and vice versa.

The fact that the two keys are associated with different algorithms should not affect the relying party software's ability to verify the certificate path. Once the link certificates have been published, the new ECC root key can start certifying end-users' ECC public keys. Typically, when relying party software that trusts the old (RSA) root key encounters a link certificate, it will automatically import the new (ECC) root key into its root key store. In this way, end-users are transparently transitioned to the ECC root.

Older versions of infrastructure software may not be capable of issuing a link certificate for an ECC root. So, it may be necessary to upgrade the infrastructure software prior to performing the key rollover operation.

6 The Decision Process

While ECC can offer substantial benefits in terms of security level and the demands it places on computing resources, its implementation is not yet as straightforward as more well-established cryptosystems. The benefits must be weighed against the costs to determine which algorithm is best-suited to the situation. This section summarizes the decision process.

First, one must consider the level of security that is required. Does the data need to be protected just for the short term (until 2010), medium term (until 2030) or long term (beyond 2030)? The answer to this question will give you the approximate level of security by reference to *Table 1 in Section 3.1*.

If the planned infrastructure must remain operational for a long period of time, then this, too, should be taken into account. For example, if a root key is to be embedded in hardware that must remain in service for the long term, then one would probably choose 128 bits of security, or greater, even though the data it protects may only need protection for a short period.

In an organization that tries to avoid rapid and unplanned reactions to external events (such as unexpected improvements in cryptanalytic techniques) an even higher security level may be chosen.

Next, the computing power available on the target platforms must be considered. Given the security level arrived at above, is it possible for the platforms to support the required RSA key size? One must consider not only the public- and private-key operations, but also the key generation operation.

For example, if the required security level corresponds to 7680 bits of RSA, and the platform includes a server that generates keys for many users, then it may not be feasible to choose RSA. Similarly, if the required security level corresponds to 2048 bits of RSA and the platform is a



constrained device, then it may take hours for it to generate an RSA key of that size. If RSA is not feasible for the given key size, then ECC is the best choice. The security level required for medium- to long-term use will disqualify RSA for many applications. However, in situations where RSA is viable, then it is the better choice in order to minimize costs and interoperability concerns.

If, based upon security and performance considerations, ECC appears to be the most appropriate choice, then we must turn our attention to the practical issues. Are the applicable standards stable? Is it possible to obtain the necessary software and hardware products? Sometimes, ECC may be the preferred choice, but the architect is driven to use RSA because of its more complete support in commercial products.

The interoperability issues must now be considered. Does the application need to support communications with end-users who don't have ECC-enabled software? Is it possible to deploy ECC-enabled software to all affected end-users? Will the end-user application software be able to work with the infrastructure? If the answers to these questions do not support the use of ECC, then RSA should be used.

Conversely, if an RSA implementation is being considered, one must ask whether the infrastructure or applications need to be hosted on, or communicate with, constrained devices that are incapable of implementing RSA at the required security level. If this is the case, and the other considerations allow it, then ECC may be the more prudent choice.

Currently, ECC tends to be more useful in dedicated applications that involve resource-constrained platforms, do not require external interoperability and do not share a common infrastructure with other applications. In these environments, ECC can be deployed as easily as any other algorithm and has many advantages in terms of efficiency and the level of security provided. For these environments, ECC is likely the preferred choice.

To summarize, ECC should be used if:

- 1) The security level required makes RSA infeasible, the applications and infrastructure are available (at acceptable cost), the application doesn't require substantial external interoperability and all of the pieces will work together, or
- 2) Constrained devices that are incapable of processing RSA need to be supported.

If any of these conditions do not apply, then RSA is likely the best choice.

Of course, none of this discussion precludes the architect from making compromises in order to arrive at the optimum solution. For example, if the interoperability issues associated with ECC can't be dealt with, and the required security level corresponds to an RSA key size that will not support the specified performance, then the architect may decide to make do with a lower security level or diminished performance.

Or, if the performance requirement is absolutely inflexible, then the architect may have to live with the interoperability issues. In the end, it is up to the organization that is implementing the infrastructure and deploying the application to decide what, exactly, its priorities are.

It should be noted, however, that, whenever possible, regardless of which particular algorithm is actually used, products should be chosen that are capable of supporting both RSA and ECC. The ability to process RSA is required for current interoperability and allows users to verify signatures of, and encrypt messages for, other users and decrypt and verify archived data. The ability to process ECC is recommended in order to support future interoperability and transition to ECC, as the need arises. In an environment in which many algorithms are in use, the system design is greatly simplified if all devices can implement all algorithms.

7 References

[B] - D.J. Bernstein, Circuits for integer factorization: a proposal, manuscript, 2001, available at <http://cr.yp.to/papers.html>

[FIPS186] - National Institute of Standards and Technology, Federal Information Processing Standards Publication 186-3 (draft), Digital Signature Standard (DSS), March 2006.

[GLV] - R.P. Gallant, R.J. Lambert, S.A. Vanstone, Improving the parallelized Pollard lambda search on anomalous binary curves, *Mathematics of Computation*, 69 (2000), pp. 1699-1705.

[ISO14888-3] - ISO/IEC 14888-3. Information technology – Security techniques – Digital signatures with appendix – Part 3: Certificate-based mechanisms.

[ISO15946-2] - ISO/IEC 15946-2. Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 2: Digital signatures.

[ISO15946-3] - ISO/IEC 15946-3. Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 3: Key establishment.

[KES] - National Institute of Standards and Technology, Special Publication 800-56a: Recommendation For Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, March 2007. Available at: <http://csrc.nist.gov/publications/PubsSPs.html>

[KMG] - National Institute of Standards and Technology, Special Publication 800-57: Recommendation for Key Management - Part 1: General (Revised), March 2007. Available at: <http://csrc.nist.gov/publications/PubsSPs.html>

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

[P1363a] - IEEE P1363a: Standard Specifications for Public Key Cryptography: Additional Techniques.

[PKCS11] - PKCS #11 v2.20, Cryptographic Token Interface Standard, November 2001. Available at: <http://www.rsasecurity.com/rsalabs/pkcs/>

[RFC3278] – S. Blake-Wilson, D. Brown, P. Lambert, Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS), RFC 3278, April 2002.

[RFC3279] – W. Polk, R. Housley, L. Bassham, Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 3279, April 2002.

[RFC3852] – R Housley, Cryptographic Message Syntax (CMS), RFC 3852, July 2004.

[RFC4050] – S. Blake-Wilson, G. Karlinger, T. Kobayashi, Y Wang, Using the Elliptic Curve Signature Algorithm (ECDSA) for XML Digital Signatures, RFC 4050, April 2005.

[RFC4055] – J. Schaad, B. Kaliski, R. Housley, Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 4055, June 2005.

[RFC4306] – C. Kaufman, Internet Key Exchange (IKEv2) Protocol, RFC 4306, December 2005.

[RFC4492] – V. Gupta, S. Blake-Wilson, B. Moeller, C. Hawk, N. Bolyard, Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS), RFC4492, May 2006.

[RFC4753] – D. Fu, J. Solinas, ECP Groups for IKE and IKEv2, RFC 4753, Jan 2007.

[RFC4754] - D. Fu, J. Solinas, IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA), RFC 4754, Jan 2007.

[RFC4869] – L. Law, J. Solinas, Suite B Cryptographic Suites for IPsec, RFC 4869, April 2007.

[RSA576] - Factorization of RSA-576, available at <http://www.rsa.com/rsalabs/node.asp?id=2096>

[S] - A. Shamir, Factoring large integers with the TWINKLE device (extended abstract), proc. CHES'99, LNCS 1717, pp. 2-12, Springer-Verlag, 1999.

[SEC1] - Elliptic Curve Cryptography, Version 1.0, September 20, 2000. Available at: http://www.secg.org/download/aid-385/sec1_final.pdf

[SEC2] - Recommended Elliptic Curve Domain Parameters, Version 1.0, September 20, 2000. Available at: http://www.secg.org/download/aid-386/sec2_final.pdf

[ST] - A. Shamir and E. Tromer, Factoring large numbers with the TWIRL device, Advances in Cryptology – CRYPTO 2003, LNCS 2729, pp. 1-26, Springer-Verlag, 2003.

[WSS] – Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), 01 November 2006. Available at: <http://www.oasis-open.org/committees/download.php/21255/wss-v1.1-spec-errata-os-SOAPMessageSecurity.pdf>

[WTLS] - Wireless Transport Layer Security Specification, Wireless Application Protocol Forum, WAP-261-WTLS-2001-04-06-a. Available at: <http://www.wapforum.org/what/technical.htm>

[WZ] - M.J. Wiener and R.J. Zuccherato, Faster attacks on elliptic curve cryptosystems, Selected Areas in Cryptography - 5th Annual International Workshop, SAC '98, Lecture Notes in Computer Science 1556 (1999), Springer-Verlag, pp. 190-200.

[X962] - ANSI X9.62-2005, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[X963] - ANSI X9.63-2002, Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

8 About Entrust

Entrust [NASDAQ: ENTU] secures digital identities and information for consumers, enterprises and governments in 1,700 organizations spanning 60 countries. Leveraging a layered security approach to address growing risks, Entrust solutions help secure the most common digital identity and information protection pain points in an organization. These include SSL, authentication, fraud detection, shared data protection and e-mail security. For information, call 888-690-2424, e-mail entrust@entrust.com or visit www.entrust.com.