

STANDARDS FOR EFFICIENT CRYPTOGRAPHY

Review of SEC1: Elliptic Curve Cryptography

Dan Boneh

dabo@cs.stanford.edu

Executive Summary

At the request of Certicom Research, I performed a review of the SEC standard for Elliptic Curve Cryptography. The attached review is based on a study of version 0.4 of the proposed standard dated August 1999. The topics examined during the review include:

- Design of cryptographic algorithms. Choice of cryptographic parameter sizes.
- Security analysis. Relation to known and potential attacks.
- Relation to alternative proposals in the crypto research community.
- Clarity of presentation.

The standard is very well written. It is easy to follow and carefully avoids any ambiguities that might arise during an implementation. Most of the cryptographic algorithms discussed in the standard have been around for several years and have been thoroughly analyzed by the research community. Similarly, recommended parameter sizes are based on practical experience with the Certicom challenges and recommendations of the research community. The available range of parameter sizes is adequate for both the “paranoid” and “not so paranoid” engineer.

All cryptographic algorithms discussed in the standard include tests designed to defeat known attacks. I was pleasantly surprised to see that no essential tests have been left out. The standard itself is conservative in its design and takes potential future attacks into consideration. An in depth discussion is provided in the body of the report.

Introduction

Elliptic Curve Cryptography (ECC) is pivotal to the deployment of cryptography on handheld devices. No other public key system scales as efficiently to provide varying levels of security. As a result there is a clear need for an efficient, scalable, interoperable standard. The SEC1 Elliptic Curve Cryptography standard is carefully designed to be such a standard. To place the standard in context I begin by explaining why, in my opinion, ECC is essential for handheld security. In the following section I discuss the choice of algorithms used in the standard as well as the choice of parameters. In the last section I give some comments that Certicom may wish to incorporate prior to release of the standard.

In the early 90's the most commonly deployed public key systems were RSA (based on arithmetic modulo composites) and regular Diffie-Hellman (based on arithmetic modulo primes). At the time, 512 bit keys were commonly used. For instance, the initial Digital Signature Standard proposal only had provisions for 512 bit keys. As can be expected, both factoring algorithms and computer hardware has been steadily improving. As a result, in August 1999 the famous RSA-512 challenge was broken demonstrating that 512 bits keys are inadequate for both RSA and Diffie-Hellman. Clearly larger keys are necessary. The question is, how large?

The difficulty with using RSA and Diffie-Hellman in practice is the existence of "sub-exponential" attacks. These attacks imply that one must use keys that are significantly longer than 512 bits to provide an adequate level of security. For example, to just *double* the amount of work for factoring the RSA modulus one must use a 543-bit key (using NFS as the factoring algorithm). NIST recommends using a 3072-bit modulus to obtain a level of security comparable to that provided by a 128 bit symmetric key. Such large keys make it impossible to use RSA and regular Diffie-Hellman on handheld devices such as cell phones, pagers, and PDAs. For example, generating a 1024 bit RSA signature on the PalmPilot takes approximately 42 seconds! [1]

Enter Elliptic Curve Cryptography. ECC provides public key systems for which, as far as we know, there do not exist sub exponential attacks. Consequently, much shorter keys are sufficient for obtaining an adequate level of security. For example, a 256-bit ECC key is sufficient for providing a level of security comparable to that provided by a 128 bit symmetric key. It is precisely this reason that makes ECC critical for cryptography on handheld devices. Indeed, on the PalmPilot generating a 163-bit ECC signature takes only 0.8 seconds (verifying the signature takes 2.5 seconds).

Several products requiring handheld security are already using ECC. I am confident that in the future we will see much heavier use of ECC on handhelds. It is worth pointing out that handhelds seem to follow Moore's law differently than desktops and mainframes. When 18 months pass and processor speed doubles, handheld vendors typically do not run the device at twice the clock rate. Instead, they keep the clock rate fixed and reduce the *size* of the device. This has been a constant trend in the cell phone industry. As a result, it is likely that public key cryptography will be cumbersome on handhelds for many years to come and consequently ECC will remain the primary cryptosystem for handhelds.

Algorithms used in the standard

The standard describes how to implement basic ECC cryptographic primitives. These include: (1) Parameter generation, (2) Key generation, (3) Signatures, (4) Encryption, and (5) Key agreement. Below we comment on the choice of algorithms and implementations for each of these primitives.

Overall, it is clear that a great deal of work was spent on ensuring that the standard is interoperable with existing proposals. The algorithms and parameters were chosen so that they offer the specified level of security. Specific checks are in place to ensure that weak parameters are avoided. Furthermore, the choice of parameters is done to optimize efficiency.

1. Parameter generation

The first step in setting up ECC is curve and field generation. The standard supports two types of fields: prime fields, denoted by F_p , and extensions of F_2 , denoted by F_{2^m} .

When generating a field F_p one picks a prime whose bit length is one of eight possible values. The smallest allowable field is of size 112 bits and the largest is of size 521 bits. The smallest field size provides a level of security comparable to a 56-bit symmetric key, while the largest field size provides a level of security comparable to a 256-bit symmetric key. The lowest level of security is adequate when export restrictions are in place. The highest level of security is appropriate for use with the upcoming AES encryption standard. We note that achieving 256-bit security using RSA requires a modulus of size 15360 bits --- hardly appropriate for use on a cell phone. With ECC a 512-bit modulus is within reason.

When generating a field F_{2^m} the standard specifies a list of nine different fields. As before, the smallest field provides approximately 56-bit security. The largest field provides approximately 256-bit security. The list of nine possible fields was chosen so as to optimize efficiency. The listed fields have special properties that can be used to speed up ECC operations.

It is worth pointing out that the standard is conservative in its choice of fields F_{2^m} . The list of nine fields all use a *prime* value for m . Currently, we do not know of a reason why F_{2^m} with m composite should be avoided. However, it is quite possible that the ECC discrete log on F_{2^m} with a composite m is only as hard as the ECC discrete log problem on the largest subfield of F_{2^m} . For example, the ECC discrete log problem on $F_{2^{169}} = F_{2^{13 \times 13}}$ might only be as hard as the ECC discrete log problem on $F_{2^{13}}$. For this reason it is best to use a field F_{2^m} for which m is prime. Such fields have no non-trivial subfields. By avoiding composite extension fields the standard pays a small cost in performance, but avoids a likely catastrophe in case a subfield discrete log algorithm is discovered. This is a sound design principle.

Once the field is agreed upon, the next step is to generate an elliptic curve over the field. Ideally, the size of the curve should be prime, but the standard also permits curve of size four times a prime. This could result in a loss of 1 bit of security. For example, if one uses a field F_p with p a 256-bit prime then a curve of prime size would offer 128-bit security. On the other hand, a curve of size four times a prime offers only 127-bit security. The benefit is that curve generation is somewhat simpler and is worth the price of 1 bit in security.

Over the past 10 years a number of weak ECC curves were discovered. These are analogous to weak RSA moduli that should be avoided. A randomly generated curve is extremely unlikely to be a weak curve and probably does not even need to be tested. Nevertheless, the standard requires a list of several curve validation tests no matter how the curve is generated. These tests rule out weak curves susceptible to the MOV, FR, and Smart et al. attacks. At the present time, the list of validation tests is comprehensive and quite adequate.

We note that curve generation is usually a costly task. The standard points to a GEC 1 specification (that I have not seen) that contains a list of suggested curves. Hopefully these curves will be made freely available to the public so as to simplify the deployment of ECC in the real world.

The domain parameters used to describe the chosen curve are adequate. They contain all required information necessary to validate and use the curve for ECC operations.

2. Key generation

Once ECC parameters are agreed upon (field and curve) the next step involves public/private key generation. The basic key generation procedure is simple and adequately described in the standard. We note that one pitfall in key generation is the generation of random bits from which the private key is constructed. The standard does not specify how the random bits are obtained and leaves it up to the implementer to use a strong random (or pseudorandom) bit generator.

The procedure for validating public keys is adequate. The validation steps include a test designed to avoid subgroup attacks. I was happy to see this test since it illustrates a sound design methodology.

3. Signatures

The standard supports the ECDSA signature mechanism. ECDSA is the elliptic curve analogue of the Digital Signature Algorithm (DSA). DSA was designed by NIST in 1989 and published as FIPS-186. It has gone through extensive review by the research community and appears to be an adequate signature algorithm. Signature schemes closely related to DSS were shown to be secure in the random oracle model [2].

The original DSS standard uses the group Z_p^* . The ECDSA algorithm is obtained by replacing Z_p^* by the group of points of an elliptic curve over F_p or F_{2^m} . Assuming the discrete log problem on the chosen elliptic curve is hard, the security of ECDSA is analogous to that of DSS. Clearly ECDSA performs much better than DSS due to the reduced field size. Thus, choosing ECDSA for the standard is natural from both a security and performance point of view.

We note that a few weak attacks on DSS were identified in the past. These are all easily prevented by proper choice of parameters. The standard carefully includes tests to ensure adequate security. Similarly, it includes all necessary tests for properly verifying an ECDSA signature.

4. Encryption

The standard defines the ECAES public key encryption standard. The encryption scheme is a hybrid scheme relying on EC-Diffie-Hellman, symmetric encryption, and a MAC.

ECAES is modeled after a recent proposal due to Abdullah, Bellare, and Rogoway called DHAES. However, there are some differences between ECAES and DHAES. For example, the key derivation mechanism follows the X9.63 ANSI standard rather than DHAES proposal. Other differences are significant only when encryption is done in a non-prime order group. Since the standard does not permit the use of non-prime order groups, the differences are mostly irrelevant.

ECAES combines Diffie-Hellman, symmetric encryption and MACing using methods that are common practice. An analysis included in the DHAES proposal suggests that this approach provides a high level of security. However, the analysis is based on very strong interactive assumptions and hence should be taken with a grain of salt.

Overall, the encryption method used in the standard is sound following existing proposals and common practice. The encryption scheme is designed for efficiency. For example, the standard does not support a recent cryptosystem designed by Cramer and Shoup (Crypto '98). The Cramer-Shoup system offers a similar level of security as ECAES, but is based on weaker assumptions. Unfortunately, the Cramer-Shoup system is about three times as slow. Similarly the standard does not support other Diffie-Hellman encryption modes such as the one suggested by Fujisaki and Okamoto (Crypto '99). Support for both Cramer-Shoup and other encryption modes can be easily added in the future if there is sufficient demand.

5. Key agreement

The standard supports two basic key agreement methods. Diffie-Hellman and MQV. Both schemes are very flexible and support multiple modes of operation. There is not much to say here since the standard uses mechanisms that have been around for a long time and have already been included in other standards.

Comments

- Middle of page 4. Sentence should read:
Additive inverses and multiplicative inverses in F_p can be calculated efficiently.
Multiplicative inverses are computed using the extended Euclidean algorithm.
- Middle of page 4. Allowable field sizes for F_p are $\{112, 128, 160, 192, 224, 256, 384, 521\}$.
Why not also allow 512 bits ??
- Page 11. Algorithm 2.3. step 2.2.2. This step is done inefficiently !! It requires an unnecessary multiplication and inversion. The step could be done more efficiently as follows:
If $q=2^m$ set $\underline{y}_p=0$ if $x_p=0$. Otherwise write $x_p = z_{m-1} x^{m-1} + \dots + z_1 x + z_0$ and $y_p = w_{m-1} x^{m-1} + \dots + w_1 x + w_0$. Let k be the smallest positive integer such that $z_k \neq 0$. Set $\underline{y}_p = w_k$.
- Page 12. Algorithm 2.3.4. Step 3 of the algorithm is missing a step:
3.5 Check that the point $P=(x_p, y_p)$ satisfies the curve equation. (!!!)
3.6 Output $P=(x_p, y_p)$.
- Page 28. Section 3.6. Why not also support the TLS key derivation algorithm?
- Page 32. Why not support DES-X as one of the symmetric encryption algorithms?
- Page 63. The reference [18] should probably be to the DDH survey paper. Actually, both the current [18] and the survey paper could be referenced.
D. Boneh, "The decision Diffie-Hellman problem", in proceedings of ANTS III, Springer-Verlag Lecture Notes in Computer Science (LNCS), Vol. 1423, pp. 48-63, 1998.
- Page 69. Middle of page. Should also reference Kocher's power analysis. Power analysis is more powerful than the timing attacks.
- **General comment:** in many places the standard says: "test that $x \in [1, n-1]$ ". These are critical to the security of the schemes describes in the standard. Both the lower bound and upper bound tests must not be ignored in any implementation. Unfortunately, implementers may not realize the importance of both the lower and upper bound tests and may choose to ignore one (or both) of them. It might be worth specifically pointing out in the beginning of the standard that implementations must include all tests specified in the body.

Bibliography

- [1] D. Boneh, N. Daswani, “Experimenting with electronic commerce on the PalmPilot. In proceedings of the third Financial Cryptography conference, Springer-Verlag, Lecture Notes in Computer Science (LNCS).
- [2] D. Pointcheval, J. Stern, “Security proofs for signatures”, In proceedings of Eurocrypt ’96.