

ECC, Future Resiliency and High Security Systems

March 30, 1999

Don B. Johnson, Certicom

djohnson@certicom.com

Revised July 6, 1999 to correct typo in RSA key generation description and clarify low exponent RSA discussion.

Preface

This paper presents some recent findings on additional advantages of using elliptic curve cryptography (ECC). In addition to some well-known advantages that suggest use of ECC is ideal in constrained environments (sometimes called low-end systems), ECC has attributes that suggest it is also ideal for use in high security environments (sometimes called high-end systems).

Introduction

Today, there are three hard arithmetic problems on which many asymmetric cryptosystems are based:

1. The integer factorization problem (IFP), e.g., RSA, or Rabin-Williams.
2. The (normal) discrete logarithm problem (DLP), e.g., DSA or Diffie-Hellman.
3. The elliptic curve discrete logarithm problem (ECDLP), e.g., ECDSA or ECDH.

Use of the phrase “hard” in the above paragraph means that the known best ways to solve each problem are at least subexponential. These three hard problems are used by ANSI X9, ISO SC27, IEEE P1363, IETF, and other standards bodies as the basis for specifications of asymmetric cryptography algorithms.

Each particular algorithm has certain security and performance attributes, which translate into advantages and disadvantages relative to a solution that meets a user’s needs. One of the tasks of a cryptosystem designer is to weigh the advantages and disadvantages and select the algorithmic tools that best address the problem to be solved. The first part of this paper summarizes some advantages of ECC in constrained environments. The second part presents some recent work that suggests that use of ECC has many advantages in environments with high security requirements.

SHORT REVIEW OF ECC ADVANTAGES IN CONSTRAINED SYSTEMS

A constrained system is one in which considerations of time (i.e., performance of key generate, signature generate, signature verify, etc.), space (i.e., ROM, RAM, bandwidth, code size, data size, etc.), cost (i.e., energy, money, etc.) or a combination thereof constrains the ability of a solution to meet security

objectives. To paraphrase an English Leather advertisement from years ago, “All my constrained systems use ECC as their asymmetric algorithm, or they use nothing at all.”

Key Size, Signature Size, and Certificate Size

The known best method to solve the ECDLP is fully exponential, while the known best methods to solve the IFP and the DLP are subexponential. Therefore ECC keys of about 161 bits (when using a technique called point compression) are about equivalent in strength to RSA or DSA keys of about 1024 bits, which is a significant savings. Furthermore, an ECDSA signature of about 322 bits is about equal in strength to an RSA signature of about 1024 bits.

As a concrete example, assuming one requirement is to meet an ANSI X9 standard, then the relative sizes of bytes of the user’s public key and CA’s signature using the most compact form for each is as presented in the following table.

	RSA	DSA	ECDSA
User’s Public Key	128	128	21
CA’s Signature	128	40	41
Total	256	168	62

Table 1 Size of User’s Public Key and CA’s Signature in Certificate (in bytes)

Key Pair Generation Complexity

To generate an RSA key pair, two random primes (p and q) are generated (sometimes with certain additional properties to eliminate certain types of attacks or for other reasons). The public modulus $n = pq$. A value for the public exponent e is selected (often a fixed value but sometimes randomly) such that e is relatively prime to both $(p-1)$ and $(q-1)$ and the value of the private exponent d is determined by solving the equation $ed = 1 \pmod{\text{LCM}((p-1)(q-1))}$, where LCM is the least common multiple. This means that an RSA key pair generation engine must have the ability to obtain random numbers from a quality random number generator (RNG), use those random numbers to calculate two prime candidates, test the prime candidates for primality and any other required conditions, test for relative primality, and solve the modular equation to determine the private exponent d [X9.31]. This is a relatively complex process (in big-O notation, it is a fourth order polynomial in the size of $\log n$), all which must be done in a secure way and without error.

Contrast this with ECC key pair generation. There are two parts to ECC key pair generation: (1) generation of a valid set of domain parameters and (2) generation of a key pair associated with that set. To generate a set of public domain parameters, a finite field is selected, an elliptic curve is generated over that field

(perhaps randomly), a basis is selected, an order for the curve is determined via point counting and a generator point G is determined with a large prime order. This is also a complex process (in big-O notation, calculating the order of a random curve is at least an eighth order polynomial in the log of the number of field elements). A less complex alternative is to use the Complex Multiplication method [P1363] to first select an order for the curve and then find a curve with the specified order.

In any case, as the domain parameters are public, anyone at anytime can validate the domain parameters values to ensure they conform to the specifications of the relevant standard, for example, ANSI X9.62 ECDSA. This means that any error during the calculations done during domain parameter generation can be detected in a straightforward manner. For example, services that a Certification Authority (CA) may choose to provide for its clients are: (1) generate a set of valid domain parameters and/or (2) validate a candidate set of domain parameters generated by someone else.

Given a valid set of ECC domain parameters, ECC key pair generation is straightforward. A random number of a certain size is generated, this becomes the user's private key d . The user's public key Q (a point on the elliptic curve) is dG , where dG is the scalar multiple of d times the generator point G , that is G "added" to itself d times, where "added" indicates the operation in the elliptic curve group. This means that ECC key pair generation, given a valid set of domain parameters, only needs the ability to obtain a random number and the ability to do a scalar multiplication of an elliptic curve point by that random value, which is the normal ECC operation. ECC key pair generation (given a valid set of domain parameters) is therefore about as simple a calculation as can be imagined. This means that ECC key pair generation is ideal to use in constrained devices.

In summary, RSA key generation in a single operation: (1) uses a quality RNG to determine a random abstract group which must be kept secret and (2) determines the private key; while ECC key generation uses two distinct operations: (1) domain parameter generation to determine a public (possibly random) abstract group and (2) uses a quality RNG to generate a private key. Both rely on a quality RNG, but RSA key generation needs additional code inside a secure boundary to generate the secret abstract group structure. When the Public Key Infrastructure (PKI) model being used assumes that each user generates his or her own private/public key pair (as contrasted with a trusted third party (TTP) generating the key pair for each user and injecting the private key into each user's device), choosing ECC means that more systems will be able to meet that assumption than if RSA were chosen.

Bullet Certificates

In the general PKI model, a user generates his own key pair and asks various Certificate Authorities to certify his or her public key. However, often there is a primary CA and sometimes only one relevant CA. It would be nice to be able to optimize the processing for use with this primary or sole CA. Such optimization is an attraction of a recent idea that Certicom terms bullet certificates. Bullet certificates (as their name implies) are smaller than normal certificates. Recall from Table 1 that in a certificate, the size of an RSA key and signature was 256 bytes, a DSA key and signature was 168 bytes and an ECDSA key and signature was 62 bytes; the same information in an ECC bullet certificate takes 21 bytes.

Elliptic Curve Secure Messaging (ECSM)

Another possible optimization is based on the realization that often both message signing and message encryption are done on the same message. It turns out that it is possible to achieve the security properties of both encryption and signatures in a single process which Certicom terms Elliptic Curve Secure Messaging (ECSM) and which is more efficient than doing both encryption and signing separately.

ECC ADVANTAGES IN HIGH-SECURITY SYSTEMS

A high-security system is one where considerations of time, space, and cost are overshadowed by the requirement for security. For example, a company may exist such that its digital signature capability IS the company. In such an environment, it may be reasonable to spend resources of time, space, and cost in an effort to achieve higher security assurances.

Forward Secrecy

Suppose an adversary steals your cryptosystem. Of course, you notice this and tell the appropriate parties to revoke all your keys; this prevents the adversary from pretending to be you and creating messages that seem to be from you in the future. However, the adversary has thought ahead and has a copy of all your old messages. He wants to read as many of these messages as he can. Tamper protection is a nice concept, but most current thinking on this subject is that if an adversary has physical access to your system and is willing to pay the cost, he can break in (that is, there is no such thing as a tamper-proof cryptosystem, just levels of tamper resistance). The adversary attacks your system using physical means and reveals all your current keys. This means he can read your messages for the session that was current at the time the cryptosystem was stolen, if there was a session at that time. It would be a very nice feature if the adversary could not read any of the messages in your old sessions. This security attribute is called forward secrecy.

In its essence and avoiding most of the technical details, forward secrecy is achieved through use of ephemeral keys, that is, keys that are generated anew for each session and destroyed when the session is ended. Destruction for any key is straightforward. However, the previous section on key pair generation mentioned that ECC key pair generation (given a valid set of domain parameters) is about as simple as is possible to conceive, while RSA key pair generation is much more complex. This means that achieving the security attribute of forward secrecy is relatively straightforward when using ECC, but is more complicated to achieve and suffers a significant performance cost when using RSA.

Security Analysis

For RSA, the strength of the algorithm increases by increasing the primary security parameter, which is the size of the modulus n . For example, ANSI X9.31 mandates a minimum size of n of 1024 bits.

For DSA, there are two primary security parameters, the size of the prime p (the order of the arithmetic field) and the size of the prime q (the order of the subgroup generated by the generator g). For example, the revision of ANSI X9.30 DSA now being discussed by ANSI X9 will mandate a minimum size for p of 1024 bits and a minimum size for q of 160 bits. To increase the effective security, both must be increased appropriately; increasing one without the other is not effective.

For ECC, the primary security parameter is n , the order of the generating point G . For example, ANSI X9.62 mandates a minimum size for n of 161 bits.

Assume for purposes of this discussion that the known best method to attack a symmetric key is by brute-force key exhaustion, which is the ideal goal of a symmetric key algorithm. Note that this attack is inherently able to be run in parallel, if a single cracker machine is expected to take m years and an adversary has k cracker machines, then his expected time to crack a key is m/k years. One would like to be able to calculate the appropriate size for the asymmetric key used to protect a n -bit symmetric key.

For an elliptic curve cryptosystem, the known best general-purpose attack is a square-root attack based on the Pollard rho algorithm [X9.62]. Note that this attack can also be run in parallel. If one assumes that a symmetric algorithm encryption takes about the same time as an elliptic curve scalar multiplication (this is a very conservative assumption), this means that one should use an ECC keysize of about double the symmetric keysize. For 128-bit AES, an appropriate ECC key size is 256 bits. This is a very simple and straightforward calculation.

The Pollard rho algorithm is also the basis of the known best general-purpose method to attack the subgroup of size q in the DSA, this means that the size of q should be about 256 bits. However, the known best method to attack either the

security associated with the size of DSA's p parameter or the size of RSA's modulus n is a complicated formula based on the General Number Field Sieve method for taking discrete logarithms or factoring. As specified in the current draft of the revision of X9.30 DSA, for 128-bit AES, an appropriate size of p is 3072 bits. An attack on the DSA field p is considered (very) slightly harder than an attack on the RSA modulus n, but for simplicity, just assume the same difficulty. As can be seen from the following table, for very high levels of security, the large key sizes for RSA and DSA keys will simply be unwieldy, ECC is the most practical choice. However, note that many experts believe that 192 or 256 bits of symmetric key security may never be needed.

Symmetric	56	80	112	128	192	256
RSA n	512*	1024	2048	3072	7680	15360
DSA p	512*	1024	2048	3072	7680	15360
DSA q	112*	160	224	256	384	512
ECC n	112*	161	224	256	384	512

Asterisk (*) means below minimum key size specified by ANSI X9 standard.

Table 2: Approximate equivalence of keys in bits to known best general attacks

Another aspect to consider when discussing security is the reduction of one hard problem to another. It has been shown that if the Elliptic Curve Discrete Logarithm Problem (ECDLP) is fully exponential in complexity, then solving the elliptic curve Diffie-Hellman problem (ECDHP) is also fully exponential; that is, the two problems are essentially equivalent [Bo6]. It has not yet been shown that solving the RSA problem is equivalent to solving the integer factorization problem (IFP) or that solving the Diffie-Hellman Problem (DHP) is equivalent to solving the Discrete Logarithm Problem (DLP). Informally, this means that breaking RSA or Diffie-Hellman may be easier than solving their underlying hard problem.

Performance Considerations

For RSA, public key operations should be expected to be faster than private key operations, for the following reasons. Use of a private key is expected to need to do modular exponentiations on the order of about 1.5 times the modulus size, for a 1024 bit modulus, this is about 1536 operations. This can be reduced by the use of the Chinese Remainder Theorem [X9.31]; but a large reduction in the size of the private key d (for example, half or more) is dangerous, and is specifically excluded by ANSI X9.31 [X9.31]. A random public exponent of the maximum size (864 bits) allowed by X9.31 for a 1024-bit modulus would be expected to take about 1,296 modular exponentiations. A method to speed up RSA public key operations is to select a small value for the public exponent. Some values that are suggested are 3, 17 and 65537, all chosen because they are prime and there are only two ones in the binary representation of each number. This

means that the number of modular exponentiations needed is respectively, 2, 5, and 17, each a dramatic reduction from 1,296.

The insightful attacks due to Hastad and Coppersmith on low exponent RSA encryption show that use of a low public exponent means reliance on a good method to format the encryption payload and on the output of a quality random number generator [X9.44]. Use of a larger public exponent can be thought of as a second line of defense; for example, if there were fears about the possibility of the random number generator being attacked.

The value for the private exponent d is determined by the values for the public modulus n and the public exponent e . For example, the use of a public exponent of 3 means that the high order half of the private exponent d is able to be completely known by an adversary (the formula is $2n/3$) [X9.31]; of course, this is not enough to break an RSA cryptosystem by itself but could be of use in validating results from some other attack, such as a “side-effect” attack such as timing or power analysis. Also, a private key is often protected by encrypting it using a symmetric key, using a public exponent of 3 means that an adversary is thereby given some plaintext/ciphertext pairs for this encryption. This is undesirable.

Furthermore, perhaps surprisingly, when using a low RSA public exponent, only the low-order $\frac{1}{4}$ of the private exponent d is needed to be known by an adversary in order to be able to recover the entire value in polynomial time (that is, in a feasible amount of time) [Bo4]. In this case, low exponent means that exhaustive search on all values less than e is feasible; for example, this concern applies to use of values of 3, 17 or 65537. Given the recent results of the DES Cracker, this suggests use of 64 bits or more for the RSA public exponent is indicated for use by those that want to mitigate this potential concern. Use of a 64-bit value for e would be expected to need about 92 modular exponentiations, which is still much less than 1,296 expected for a maximum size RSA exponent.

The potential number of RSA keys is also affected by the use of a small public exponent. For a public exponent of 3, the reduction is to about 44% of the theoretical key size; for a public exponent of 65537 or greater, the reduction is negligible [X9.31]. Even for a value of 3, this is not a dramatic reduction; but, taken together, the above ideas at least suggest that an adversary might have a greater chance of success in attacking a deployed system if attacking an RSA public key with a low exponent than in attacking an RSA public key that does not, if simply for the reasons that more is able to be known.

In fact, there are indications that breaking low exponent RSA may not be equivalent to factoring [Bo2]. In this case, low means that exhausting $2^{**}e$ is feasible, where $**$ denotes exponentiation, for example, use of the values 3 or 17. That is, it is possible that the RSA algorithm using a large public exponent is secure, but use of a low public exponent in RSA is not secure. If such should

turn out to be the case, then the prudence of those who decline to use low exponent RSA would be justified. At the least, one should distinguish between “normal” RSA and “low exponent” RSA and know which type one is using.

For ECC, private key operations should be expected to be faster than public key operations. One reason for this is because a private key can use precalculated values to improve performance. However, the same idea will work for an ECC public key, an ECC certificate can contain precalculated values in a helper field that assists the calculations using the public key. This can improve performance dramatically. This idea is also possible for DSA keys, but is not as practical. Also, an ECC private key, if available, can be used to verify a signature that it was used to generate; this is very fast and is useful, for example, when a CA is recertifying a certificate. Notice also that protecting an ECC private key by encrypting it with a symmetric key does not lead to an adversary obtaining any known plaintext/ciphertext pairs for this encryption.

Calculation Errors

As mentioned above, one method to improve performance of an RSA private key is through use of a method based on the Chinese Remainder Theorem [X9.31]. However, if this CRT calculation gets a single undetected bit error and the results of the calculation are known (for example, if a bad RSA signature is transmitted), then the entire RSA private key is exposed [Bo3]. This suggests that anytime an RSA signature fails to verify, a person with larceny in their heart might wish to see if the RSA private key is revealed. If this ever happened and assuming the message and signature were transient, discovering how the RSA private key was revealed could be problematic.

For ECC, the analogous known best methods take many undetected bit errors [Bo3]. An error in calculation simply results in a signature that will not verify. For example, suppose there was a bit flip in an ECC private key during a signature calculation. The signature will be valid for the altered private key, but the public key will not correspond to it so the signature will not verify. If the bit flip was transient, then the next signature will be able to be verified with the public key.

Public Key as an Abstract Group Operation

In ECC, a user’s private key is a random number of a certain size. In many ways this is an ideal secret, similar to the value of a symmetric key. The user’s public key is the result of the operation of multiplying the base point G by the user’s private key. The abstract group that the user’s public key has meaning in is determined by the set of domain parameters. As noted before, all information about the structure of the abstract group is public and can be validated by anyone at anytime to ensure it conforms to the specifications (that is, it is suitable for cryptographic use). The user of an ECC public key can know all the details about the group operation that use of such a key entails.

For RSA, the private key is d , the private exponent. However, besides d , the values of p and q (the factors of the public modulus n) must be kept secret. Also the value of $\phi(n)$ must be kept secret, otherwise n can be factored to recover p and q . However, the value of $\phi(n) = (p-1)(q-1)$ is the number of elements (i. e., the order) of the abstract group. This means that fundamental information about the abstract group must be kept secret. The user of an RSA public key cannot know all the details about the group operation that use of such a key entails, he does not even know how many elements are in the group. This means the user of an RSA public key is required to trust that “things are OK” in a way that a user of an ECC public key is not. Public key validation (discussed below) addresses many of these trust issues and is a current research topic.

Public Key Certification and Validation

For any public key cryptosystem, services that a CA may choose to provide as part of the public key certification process include (1) having the claimed owner demonstrate proof of possession (POP) of the associated private key and (2) running arithmetic tests to show that the claimed public key conforms to the specifications of the relevant standard; this is known as public key validation (PKV). In either case, if these tests are not done, then either the owner of the private key or the user of the public key may be open to attacks.

As an example of the usefulness of POP, consider the case where Alice and Bob have certified public keys, but the CA does not do POP or check for duplicate public keys. Alice signs a message and encrypts it using Bob’s encryption public key. Eve intercepts the message. She cannot read it, but wants to know information about what Alice said to Bob. Eve gets Alice’s public key certified as hers (as POP is not done) and sends Alice’s message to Bob along with Eve’s certificate. The signature verifies successfully. Bob thinks the message came from Eve, even though Eve has no idea what it says. Bob now responds to Eve. Eve may be able to discern what Alice’s message said by the nature of Bob’s response. This is a simple example; there are many variations.

POP is often accomplished by using the private key in a natural way in an exchange with the CA. For example, Ann may show she owns her private signature key by signing the certificate request to the CA. The CA extracts the public key and verifies the signature on the request. Another way to do POP is to do a Zero-Knowledge Protocol (ZNP) with the verifier. Showing proof of possession of a private key is a straightforward process for any asymmetric algorithm.

As an example of the usefulness of PKV, in RSA the assumption is that the modulus n cannot be factored. If the key pair generation calculations are done according to specifications, then the RSA modulus should not be able to be factored according to current knowledge, but what if there was an error during the calculation that allows it to be easily factored. As a simplistic example, what

if n , instead of being the product of two primes, was itself prime? The answer is that all operations using this “RSA” public key would be totally insecure, yet any RSA public key operation (e.g., signature verification or encryption) would appear to work just fine.

Of course, a public key might also be invalid due to a deliberate act of an adversary. For example, Eve might be a third party adversary or even a first party adversary, appearing to participate as a normal user. Other examples of the importance of PKV include the Lim-Lee attack or Vanstone small subgroup attacks on Diffie-Hellman key agreement [X9.42, X9.63].

Recall from the key pair generation discussion that ECC divides the key pair generation problem into two parts, a complex but publicly auditable part and a simple private part, while RSA requires that key pairs be generated by a relatively complex process that must be private. If one makes the plausible assumption that there must be a certain level of complexity in any public key algorithm, ECC makes almost all the complex key pair generation calculations auditable in public information while RSA must make all of its complex key pair generation calculations in private. This has important implications for public key validation.

Demonstrating to another party (for example, a CA) that a candidate RSA public key conforms to the specifications of a standard requires the use of the private key owner acting as an oracle to answer queries from the other party [Jo, Si]. This means that RSA public key validation (beyond some basic tests) is an online process. Also, this process may leak some information about the RSA private key, not enough to cause a break considered by itself, but such information leakage is certainly an undesirable side effect of RSA PKV. Furthermore, as the method for RSA PKV is so complex, there does not (today) exist an ANSI X9, IEEE, or ISO SC27 standard that specifies a method of even minimally validating an RSA public key.

Contrast this with ECC public key validation, as specified in ANSI X9.62, IEEE P1363 and ISO SC27 standards. The validation process is offline and can be done by anyone at anytime just by examining the public key. The validation process is 100%, in the sense that successful validation demonstrates that an associated ECC private key can logically exist. There is no need to use the private key owner as an oracle to answer queries. In some sense, ECC public key validation is ideal; it meets all objectives and has no security downside.

Everyone knows that it is good engineering practice to check calculations. It is often best to do this in a publicly auditable manner. For ECC keys, accomplishing this is straightforward; for RSA keys, it appears to be difficult and complex and is a current research topic.

Multi-Party Private Key Generation

A scenario that is useful in high security applications is to allow a multiple party signature. For example, I might want to allow any two people among the CEO, CFO and CIO of a company to sign a contract, but not allow just one. This can be handled by distributing a public key for each party and saying that two different valid digital signatures are required for the contract to be binding. For example, requiring two signatures on a high value check is a common practice. However, note that this example requires that three public keys be associated with the company; furthermore, the verifier is required to know that the contract is binding only if it has two or more digital signatures.

In many situations, it may be more desirable to have a single public key associated with the company, rather than three. What is needed in this second scenario is threefold: (1) a way to generate the key pair in a special way, so that each owner has only a partial private key and (2) a way to use these partial private keys to form partial signatures that somehow can be combined into a valid signature that will verify using the single public key and (3) assurance that a partial owner cannot cheat somehow to become a full owner.

Recall that, for ECC, the private key is a random number of a certain size. A random number can be split up and shared in many ways between parties such that combining the right number of parties allows reconstruction of the random number. For example, bitwise Boolean exclusive-or can be used to combine key parts as well as modular addition, etc. This means there is great flexibility for the multi-party algorithm designer in the format chosen for the private key parts in meeting the security requirements. This flexibility allows the choice of a method that best meets other requirements, whatever they may be. For ECC, as the abstract group structure is public, it is also straightforward to check another's calculations to see if there are any errors.

For RSA, requirement (3) implies that no partial owner can determine the factorization of n ; that is, each partial owner is required to "operate in the dark" in a way similar to the user of an RSA public key. For RSA, as some of the group information cannot be known by anyone, it is more difficult to check another's calculations to detect errors. At Crypto '97, a complex but feasible way of generating a multi-party RSA key pair was presented [Bo5].

Zero-Party Private Key Generation

If single-party private keys are for normal users and multi-party private keys may have uses for high-security applications, a natural question is: "Is it possible to have a zero-party private key?" It turns out that it is straightforward to generate a valid ECC public key for which no one knows (or ever knew) the associated private key and that this fact can be validated by anyone. First, a random curve is generated, as part of a valid set of ECC domain parameters. Second, a

random point on the curve is selected by some verifiable procedure, such as the use of a seeded hash. Third, this random point is multiplied by the cofactor (one of the ECC domain parameters), this ensures the result (the candidate public key) is a point on the subgroup with large prime order. Fourth, public key validation is done on the candidate public key to ensure that a private key can logically exist, that is, that there were no errors during the calculation.

This is what was done in the Certicom ECC challenge to provide assurance that any reported break could only have been done by honest means, as there were no insiders with secret knowledge. With prizes up to \$100,000 for a single break, this was important. For more information on the Certicom ECC challenge, see <http://www.certicom.com/chal/contents.htm>.

Another use of is to provide a method for key recovery that is known to take an expected amount of time to recover the key. As no one ever knew the value of the private key and everyone can see that no one ever knew, no cheating is possible; any recovery of the key must be honest. This helps ensure that any key recovery must take some work to accomplish and helps allay fears of widespread key recovery by corrupt escrow agents. Another use is to store data away in a “cryptographic time capsule” so that it can be opened in the future after doing a known amount of work.

For RSA, generation of such a public key appears far from straightforward. At least one researcher in the field thinks “it may be impossible” [Bo7]. An indication of how difficult it might be can be surmised by recalling the simpler topic of multi-party private key generation.

Random Number Generator Failure

A fundamental requirement for the security of most cryptosystems is the availability of a good source of random numbers. The recent attack on the RNG in Netscape Navigator highlighted this requirement. As noted above, both ECC and RSA key pair generation require the use of a quality RNG. Any cryptosystem relying on a totally compromised RNG (for example, if the output is stuck to a single value) is insecure.

Concerns about the potential for failure of an RNG have led to the specification in FIPS 140-1 of validation tests for an RNG [FIPS140]. The continuous output test checks to see if the RNG output that was just generated is equal to the last output generated; if so, a failure is indicated. Also, some specific statistical tests are specified, these test 20,000 bits of RNG output on startup and at any selected time in various ways. If any of the statistical tests fail, a failure is indicated. Each FIPS 140-1 statistical test has a rejection rate of about 1 in a million; that is, there is less than a one in a million chance that the statistic will indicate failure but the output was actually random.

The FIPS 140-1 tests go a long way toward addressing concerns with an RNG. However, no tests of an RNG can detect every possible error. The “Chilling” Flaw in an RNG discussed below is a scenario where the FIPS 140-1 tests will almost certainly pass, but the user may be compromised. The specific numbers used in the following discussion are intended to simplify the discussion by making the calculations concrete, an actual attack might use larger numbers.

“Chilling” Flaw in Random Number Generator

Suppose an organization decides to distribute one million smartcards to all its clients. Most of the time things go well; however, a manufacturing defect damages (that is, “chills”) the RNG on 100 smartcards so that each card only produces 10,000 different 256-bit random numbers and all 100 cards produce the same 10,000 numbers. (This defect might be inadvertent or deliberately caused by an insider.) The smartcard appears to work fine, so the defect is not likely to be detected by normal functional tests. Note also that the FIPS 140-1 tests are expected to pass, as a repeat might be expected to occur after 100 RNG outputs, but this is 25,600 bits; more than the 20,000 tested by the FIPS 140-1 tests. To simplify the discussion, assume the adversary somehow knows which cards have a high potential for being “chilled”; perhaps he tested his smartcard extensively and discovered it was “chilled” or perhaps he paid the insider to cause the defect. Now let us consider what happens when RSA and ECC are used.

“Chilled” RNG with RSA Allows GCD Attack

For RSA, 10,000 random numbers means 10,000 different primes can be generated, this means about 100,000,000 different RSA moduli can be generated using these 100 cards. However, after about 50 RSA moduli are generated (using 100 primes) one would expect a repeat of some prime, due to the birthday phenomenon, as there are only 10,000 primes from which to select.

The adversary obtains the 100 RSA moduli from the 100 “chilled” cards and calls a greatest common divisor (GCD) routine among every possible pair of moduli, this is about 9900 GCD calculations. The expectation of the adversary is close to 100% that he will find at least one pair of moduli with a common prime, thereby cracking two RSA keys. He then picks the RSA key with the larger monetary value and proceeds to defraud the legitimate owner. The legitimate owner complains, but the CA may have no idea what is going on.

Consider the situation of the organization and what they might try to do to detect what was happening, let alone thwart it. They might ask the CA to check for duplicate RSA moduli, but in this particular example each of the 100 “chilled” smartcards would need to generate 100 moduli before there was an expected repeat which could be detected. The organization might consider doing a GCD

calculation among all possible pairs of moduli, but this takes about one trillion GCD calculations, which would take time and money to do.

RSA Rekey May Not Address Problem

The most likely result is that the user assumes the adversary got lucky in guessing his private key, revokes his public key and generates another key pair. But now an interesting thing happens, whenever one of the 100 “chilled” smartcards generates a new RSA key pair, there is a chance for a new match whereby the adversary gets to attack a new RSA key. The normal procedure of revocation and generation of a new key pair when one suspects a compromise might actually REDUCE security in this “chilling” scenario.

“Chilled” RNG with ECC Allows Detection of Problem

For ECC, 10,000 random numbers means 10,000 different private keys are possible. A repeat is expected after about 100 private keys are generated. A CA can detect the “chilled” RNG by checking for duplicate public keys, as the public keys for two different users will be the same. If the private key is used as a per-message key (for example, k in ECDSA), then the associated public key (r in ECDSA) will repeat and the user can detect this in a straightforward way.

In summary, both ECC and RSA rely for security on the output of a quality random number generator. If the RNG is compromised totally, then no algorithm is secure; however, this should be able to be detected using RNG validation tests. The interesting result is when the generator is partially compromised such that it still passes the RNG validation tests. ECC allows detection of a partially compromised RNG in a straightforward way, while RSA has a “gray area” where the user’s security can be compromised but detection may be difficult.

Future Resiliency

Cryptographic algorithms get invented and then standardized. Cryptosystems are then built by vendors and bought by various organizations and deployed. At some time later, an adversary attempts to attack a deployed cryptosystem. However, this is an unfair battle. The adversary gets to work in the future, when more cryptanalytic knowledge will be available. The algorithm inventor and system designer can make assumptions about what will be possible in the future, but no one can know for sure what will be possible. Sometimes surprises occur. One of the reasons cryptography is so interesting and challenging is that 99 cryptographers can look at an algorithm, protocol, or system and see nothing wrong, but a single cryptographer with a new insight can sometimes break it. The goal of future resiliency is to try to design a system today to try to address the potential for a nasty surprise in the future.

What Can Go Wrong?

In the future there may be an advance in a special purpose attack (where a subset of keys can be attacked) or an advance in a general-purpose attack (where essentially all keys can be attacked). Both types of advances have occurred historically. An example of a special purpose attack is the recent anomalous attack, where a method was discovered to attack an elliptic curve if the order of the curve exactly equaled the order of the underlying field. An example of a general-purpose attack is the invention of the General Number Field Sieve, this was a dramatic improvement in the ability to factor integers.

Handling Known Special-Purpose Attacks

How does one design a system to handle known special purpose attacks? The answer is that one excludes the special weak cases either in a deterministic or probabilistic way. An example of a probabilistic method is found in ANSI X9.30 DSA that requires the use of a canonical seeded hash to generate the p and q domain parameters. This was due to an attack on the initial version of DSA by Dan Gordon using specially constructed (but rare) values for p and q . One example of a deterministic method is found in ANSI X9.31 rDSA (RSA and RW signatures) where the difference between the primes p and q (the factors of the public modulus n) is required to be large [X9.31]. This is done because if the difference is small, then the square root of the public modulus n can be used as an estimate for the values of p and q . Another example of a deterministic method is in ANSI X9.62 where the MOV attack [MOV] and the anomalous attack [Sm, SA, Se] are excluded by checking an exclude list for the conditions that would allow these attacks to succeed and rejecting such curves for cryptographic use.

Handling a Future Advance in a Special-Purpose Attack

How does one design a system to handle future special purpose attacks? One method is to use a random key pair generation method that will hopefully exclude the special purpose attack on probabilistic grounds. If the attack works only on rare keys, this will be successful, at least most of the time. However, it seems clear that the preferred method would be to add a deterministic check to exclude the newly-discovered weak cases. A natural question to ask is “How feasible is it to add a check for a newly-discovered weak condition?”

For RSA, the answer is murky. Recall that RSA key pair generation is complex and that information about the RSA abstract group must be kept secret. Recall also that RSA public key validation is very limited in capability if one cannot use the owner of the private key as an oracle to answer queries and that oracle queries (when available) can leak information about the private key. This means that adding an exclusion test to an existing system to filter out a newly-discovered attack is problematic, such a capability may not be feasible and might

be able to be misused. It is unclear if tests for a newly discovered attack will be able to be run, except by essentially running the attack itself on the RSA public key. In a nightmare scenario, it may be difficult to run tests on existing RSA public keys, as a CA may be swamped trying to help.

The conservative assumption is that users may need to treat a special purpose attack on a significant fraction (exact value determined by the user) of all RSA keys as a general purpose attack on all RSA keys, as users may simply not be able to determine if their keys are affected or not. This has some potential implications: (1) the need to stop using the RSA algorithm or accept the risk of the attack and (2) the need to immediately deploy systems with an updated RSA key generation routine. Even worse, increasing the keysize may not help, if the fraction of RSA keys able to be attacked holds relatively constant as key sizes are increased.

For ECC, as the domain parameters (and therefore the abstract group structure) are public, it is straightforward to add a new condition to the exclude list in the domain parameter validation routine. The updated domain parameter validation routine can be run by a user or by a third party (such as a CA) on behalf of the user. One can know in a straightforward way whether one is using a set of domain parameters susceptible to the new-discovered attack.

Some may say that what one cares about is cracking an ECC private key, and not the domain parameters. This is true, however, recall that the difficulty of the DLP or ECDLP is independent of the generator [HAC 3.53]. This means that any algorithm which computes logarithms using one particular generator can be used as a subroutine to compute logarithms using any other generator. As the generator for ECC has prime order, this means any ECC public key is also a generator. Therefore, breaking one ECC public key implies the ability to break all ECC public keys using the same set of domain parameters. This implies that either all ECC public keys from a specific set of domain parameters can be broken or none can be, there is no special strength or weakness about any particular (valid) public key and any apparent structure in the associated private key that might hypothetically allow an attack is an illusion.

Another way to view this is that the specific value for a private key depends on the generator, using a different generator results in an entirely different value for the private key, that is, the specific value for a private key is arbitrary. This means that the code for ECC key pair generation does not need to change when an attack is discovered, only the input domain parameters need to change.

If a weak set of domain parameters is found, it can be revoked (along with all its associated public keys) and replaced with either an existing strong set or a new strong set of domain parameters. The unlucky users (if any) are told they must generate new key pairs using a set of domain parameters resistant to the new attack. In a nightmare scenario, where a significant fraction of keys are open to a

newly discovered attack, at least it is able to be known which keys are open to the attack and the actions that will need to be taken are known ahead of time. A cryptosystem designer can prepare for the possibility of the future discovery of an attack by including the capability in the system for using different sets of domain parameters. That is, ECC systems are able to be designed so they are future resilient to the discovery of new special-purpose attacks.

Handling Known General-Purpose Attacks

How does one design a system to address known general purpose attacks today? One answer is to increase the primary security parameter(s) so that the attack is again infeasible. An example of this is where the invention of the GNFS resulted in the size of RSA keys thought to be infeasible to attack increasing from 512 bits to 1024 bits. However, this clearly depends on the nature of the attack.

Another answer is to use a different algorithm that is not open to the attack. A method based on the knapsack hard problem was one of the first public key systems proposed, once shown to be flawed it was dropped from consideration [HAC].

Handling a Future Advance in a General-Purpose Attack

How does one design a system to address future general attacks? One answer is to use longer keys for critical uses, for example, a normal user might use a 1024-bit RSA key, but a CA might use a 2048-bit RSA key. As we have seen, there may be a flaw in such thinking, for example, if RSA has a special-purpose attack discovered in the future on a large fraction of keys where the large fraction remains stable even as the keysize is increased.

Another answer for the user with very high security requirements (some might say paranoid security requirements) is to use multiple algorithms for critical uses. For example, use two signatures instead of just one; all users are instructed to accept a message as valid only if both signatures verify. This is similar to the idea of having multiple boundaries of protection in the physical world. If one algorithm or key is broken somehow, another algorithm and key remain to protect the critical message. Some companies either have or are thinking of having a signature key that essentially IS the company. It may be prudent to use multiple algorithms for that critical signature key.

Of course, using three algorithms based on all three different hard problems mentioned above might be the most secure but often there are other considerations. Which two algorithms in combination might provide the best security for those with very high security requirements? Recall the following:

1. RSA has one primary security parameter (the size of the modulus n) for which the known best attack is based on the GNFS.

2. DSA has two primary security parameters (the sizes of the domain parameters p and q) of which the known best attack on the first is based on the GNFS and the known best attack on the second is based on the Pollard rho algorithm.
3. ECC has one primary security parameter (the size of the order of the generator of the elliptic curve subgroup) and the known best attack is based on the Pollard rho algorithm.

The Most Resilient Two-Algorithm Signature

It seems most plausible to hypothesize a general-purpose advance corresponding to increasing the existing performance of the GNFS or the Pollard rho algorithm. If the former scenario would occur, both RSA and DSA would likely be affected; if the latter scenario would come to pass, both DSA and ECC would likely be affected. Furthermore, recall that both DSA and ECC (e.g., ECDSA) are based on similar ideas, disparity of algorithms is a desirable trait when seeking to achieve future resiliency, in case some specific aspect of a design should prove an avenue for attack. Given this analysis, using two signatures, RSA and ECC, would seem to cover either potential general advance scenario and also use algorithms based on disparate designs and therefore represent the best two-algorithm approach.

Furthermore, as ECC allows short signatures, it is possible to replace the hash value in a RSA signature with an ECDSA signature. This would result in a signature of the same size as an RSA signature by itself, but would be future resilient by incorporating ECC, that is, the combined signature would be based on two different hard problems.

Summary

Advantages of ECC in a constrained environment include the following:

1. Shorter keys - 161-bit ECC is about 1024-bit RSA/DSA.
2. Shorter signatures – 322-bit ECC is about 1024-bit RSA.
3. Shorter certificates – 256-byte RSA or 168-byte DSA is about 62-byte ECC.
4. Simple generation of key pair, given a valid set of domain parameters.
5. Bullet certificate – implicit certificate from one CA, 21-byte ECC.
6. ECSM – combined message encryption and signature.

Advantages of ECC in a high-security environment include the following:

1. Easy to achieve the security attribute of forward secrecy.
2. Simple mapping of symmetric keysize to appropriate ECC keysize.
3. Reasonable ECC keysizes to protect larger AES keysizes.
4. Proven equivalence between the ECDLP and the ECDHP.
5. Improved ECC public key performance via helper fields in a certificate with no additional risk.
6. A single-bit error in a signature calculation does not reveal ECC private key.

7. ECC private key is an ideal secret, a random number of a certain size.
8. ECC private key is resilient to partial key exposure, leaking information about the private key reduces the keyspace by the amount of information leaked.
9. ECC public key validation is ideal, validation is offline and is 100% (successful validation shows that an associated private key can logically exist, yet gives no information about the value of the associated private key).
10. Straightforward generation of a multi-party ECC private key.
11. Simple generation of a valid ECC public key where it can be publicly audited that no one knows the associated private key without doing an honest attack.
12. Straightforward detection of a “chilling” attack on random number generator.
13. Future resilient to the discovery of a new special-purpose attack on ECC.
14. Future resilient as a component (with RSA) of a two-algorithm signature that addresses fears of an advance in a general-purpose attack leading to catastrophic algorithm failure.

Acknowledgements

Scott Vanstone and Dan Boneh each provided many insightful comments to improve the technical content. Steve Howard provided comments that helped improve understandability. Certicom provided the environment to conduct research in these very interesting areas of cryptography.

References

[Bo1] D. Boneh, “Twenty years of attacks on the RSA cryptosystem,” 1999, available from <http://theory.stanford.edu/~dabo/index.html>.

[Bo2] D. Boneh and R. Venkatesan, “Breaking RSA may not be equivalent to factoring,” 1998, available from <http://theory.stanford.edu/~dabo/index.html>.

[Bo3] D. Boneh, R. DeMillo, R. Lipton, “On the importance of checking cryptographic protocols for faults,” 1997, available from <http://theory.stanford.edu/~dabo/index.html>.

[Bo4] D. Boneh, G. Durfee, and Y. Frankel, “Exposing an RSA Private Key Given a Small Fraction of Its Bits,” 1998, available from <http://theory.stanford.edu/~dabo/index.html>.

[Bo5] D. Boneh and M. Franklin, “Efficient Generation of shared RSA keys,” 1997, available from <http://theory.stanford.edu/~dabo/index.html>.

[Bo6] D. Boneh and R. Lipton, “Searching for Elements in Black Box Fields and Applications,” 1996, available from <http://theory.stanford.edu/~dabo/index.html>.

[B07] Dan Boneh, personal communication, February, 1999.

[FIPS140] Federal Information Processing Standard 140-1, available from <http://www.nist.gov>.

[HAC] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, "Handbook of Applied Cryptography, CRC Press, 1997.

[Jo] D. Johnson, presentation on Public Key Validation to ANSI X9.F.1 working group, 1997.

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

[P1363] IEEE P1363 Standard Specification For Public Key Cryptography (draft), 1999, available from <http://grouper.ieee.org/groups/1363/index.html>.

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

[Se] I. A. Semaev, "Evaluation of discrete logarithms on a group of p-torsion points of an elliptic curve in characteristic p," Mathematics of Computation, Vol. 67, No. 221, Jan. 1998, pp. 353-356.

[Si] R. Silverman, presentation on RSA Public Key Validation to ANSI X9.F.1 working group, 1997.

[Sm] N. Smart, Posting to sci.math.research, September 30, 1997.

[X9.30] ANSI X9.30-1999 Public Key Cryptography for the Financial Services Industry: The Digital Signature Algorithm (DSA) draft, available from <http://www.x9.org>.

[X9.31] ANSI X9.31-1998 Digital Signatures using Reversible Public Key Cryptography for the Financial Services Industry (rDSA), available from <http://www.x9.org>.

[X9.62] ANSI X9.62-1999 Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), available from <http://www.x9.org>.

Biography

Don B. Johnson is Director of Cryptographic Standards for Certicom, is a member of Certicom Research, and sits on the Advisory Board of the Standards for Efficient Cryptography Group (SECG). He participates in ISO SC27, ANSI X9, IEEE P1363 and other standards bodies. He has over 40 patents and patent applications and is an inventor of the unified model of key agreement found in ANSI X9.42 and IEEE P1363, methods for public key validation, the CDMF (40-

bit DES) algorithm, and the reverse signature concept found in ANSI X9.44 and IEEE P1363. He was the editor of the X9.62 Elliptic Curve Digital Signature Algorithm (ECDSA) standard. He is a member of the IACR and the IEEE Standards Association.