One Key to Rule Them All

Nigel P. Smart
Extended Enterprise Laboratory
HP Laboratories Bristol
HPL-1999-26
March, 1999

cryptographic keys

We show how to specify an elliptic curve public key, RSA public key and DSA public key all in a single 2048 bit block. The method gives a wide choice of finite fields and curves for use in the ECC system and introduces no known security weaknesses. The method hence allows algorithm type to be decided at run time, rather than at the time the public keys are distributed. However, this is done without the need for very large key lengths.

ONE KEY TO RULE THEM ALL

N.P. SMART

ABSTRACT. We show how to specify an elliptic curve public key, RSA public key and DSA public key all in a single 2048 bit block. The method gives a wide choice of finite fields and curves for use in the ECC system and introduces no known security weaknesses. The method hence allows algorithm type to be decided at run time, rather than at the time the public keys are distributed. However, this is done without the need for very large key lengths.

Suppose Alice wishes to publish her public key in some directory, so that someone else, say Bob, can verify Alice's digital signatures, or send Alice secret messages. At present Alice first has to decide on which of the three standard public key algorithms to use: RSA, DSA or ECC. Given this Alice then passes to the directory, or CA, the bit pattern which represents her public key for that given algorithm. If Alice wished to keep the choice of algorithm open, until the key was actually used, then she would need to give the bit patterns of three independent public keys.

Such a situation could arise where we do not know how the public key is to be used. For example, if Bob is going to verify signatures from Alice, or send messages to Alice, using a small constrained device then RSA is probably to be preferred as the public key operations are much faster with RSA. If however, Alice's operations, whether signing or decryption, are to be performed on a small constrained device then ECC is probably to be the preferred option. There may even be some situations where DSA is the preferred option.

In any case if Alice wishes to make available all three public keys in the directory then, with current recommendations for key sizes, she will require 1024 bits to specify her RSA key, another 1024 bits to specify her DSA key (assuming a preagreed finite field is used) and another 170 bits to specify an ECC public key (assuming a preagreed curve is used). Hence a total of 2218 bits are required.

If the system require that users should use different finite fields for DSA or different curves for ECC then the number of bits required increases, to at least 3500. In fact Alice may prefer to use elliptic curves over odd characteristic fields, since Alice may be using a PC based environment to perform her operations in. Whilst another user may prefer elliptic curves over even characteristic fields, since they may be using a dedicated hardware device. It is also known that using the same finite field for DSA over a large number of users creates an attractive weakness which could be exploited by an admittedly rich adversary. If all three keys where stored in their standardized ASN.1 notation then the amount of storage required would be even larger.

Another problem with using separate keys occurs in the (admittedly) unlikely event that one of the three main public key algorithms falls to an as yet unknown attack. If this "doomsday" scenario occurred then all public keys and the associated public key infrastructure would need to be revoked and redeployed for all keys which used the given public key algorithm. If however a public key was used which did not depend on the algorithm choice then the public key infrastructure would not need to be revoked and redeployed, users would just switch from using the insecure algorithm to one of the secure ones. Admittedly legacy signatures could then be forged (we shall not address this problem but note it can be solved using a trusted time stamping authority). However we emphasize that future signatures, secure communication and key agreement could proceed at no extra cost.

Using the *superkeys* introduced in this note there is another way to secure signatures against a future doomsday scenario. By signing the document with the three algorithms of the superkey in parallel Alice can ensure that the signatures remains valid until the last of the algorithms is made insecure. For electronic legal documents that are needed to remain in force over many years (or even decades) this provides added security.

A similar process can be carried out for encryption of master keys, where the master keys need to be stored for a long length of time. Encryption can be triple locked with the superkey by applying the three algorithms in sequence.

Another advantage arises from being able to use, say, ECC to perform Alice's signature operations and RSA to perform her decryption operations, since it is not considered good practice to use the same key for both signing and decryption. But by using a superkey one uses the same public key but in two different contexts.

In this note we explain how to embed an RSA key,a DSA key and an ECC key into one 2048 bit string. Such a string will give a wide choice of finite fields to use for ECC, no restriction on which curve to use over this finite field and a virtually unique finite field in which to implement DSA. Hence this 2048 bit string can be considered a public key to end all public keys, or to paraphrase Tolkien [4]

One key to rule them all,
One key to find them,
One key to bring them all
and in the darkness bind them.

We shall first specify the elliptic curve public key, then the DSA public key and finally the RSA public key. We let n denote the number of bits of security in the ECC public key and m denote the number of bits of security in the RSA public key. The DSA key will be a field of order around 2^m with a multiplicative subgroup of order around 2^n.

The method works in two main stages. First the elliptic curve parameters and keys are produced in a way which means they occupy as few a number of bits as possible. Secondly the bit pattern of the elliptic curve key is embedded into a RSA key using a technique of Lenstra [2].

In summary our method allows a relatively short bit string to be interpreted as a variety of public keys for different algorithms. This allows additional functionality of the public key infrastructure by

1. Allowing users to decide which public key algorithm to use dynamically rather than having this dictated by the entity deploying the public key infrastructure.
2. The algorithm choice can depend on the environment rather than the identity of the user.
3. Providing the ability to cope with the scenario of a public key system being declared weak.

4. Provides a mechanism to triple lock an encryption or triple sign a message with only relatively small key size. Thus giving added security over a longer period of time.

In addition other benefits arise from our method of creating ECC keys and ECC system parameters with a small amount of space. For example the current ASN.1 definition of elliptic curve parameters and keys in X9.62 [5] means that keys can take up a lot of bandwidth. This can be a problem, given that ECC will be used in small constrained devices, where bandwidth is a problem. Using the small key lengths derived from our method one can achieve the benefit of users using different fields and curves without the disadvantages of preagreement and/or increased bandwidth.

We end this introduction with a short note on the notation we shall use in the rest of this paper. If x and y are bit strings we let $x \mid\mid y$ denote the concatenation of the bit strings. If x is a bit string we let $\{x\}$ denote the number or finite field element represented by x (with least significant bit last) and for a number or finite field element y we let $[y]$ denote the bit string which represents y. Clearly this notation will clash with other notations, such as a set of one element and the multiplication by y map, however we assume the reader is intelligent enough to understand the meaning from the context.

1. The Elliptic Curve Public Key

All the details we require about elliptic curve systems can be found in the book [1].

1.1. **The Finite Field.**

We first need to specify which finite field to use and whether it is of even or odd characteristic. We wish to choose an elliptic curve, E, over a field $\mathbb{F}_q$ of order around 2^n. Note that in all current practical elliptic curve systems, one would choose $n \leq 255$.

To specify an even characteristic field can then be done in the 8 bits needed to represent n, assuming an Optimal Normal Basis is used to represent the field elements. Alternatively, if we restrict to fields with a trinomial basis then we can represent the field using

$$X^n + X^c + 1$$

so we require another 8 bits to represent c. In practice it is common to take n to be odd in this situation, which we shall indeed do (for reasons to become apparent later). Hence to specify the even characteristic field requires at most 15 bits, since we are assuming n is always odd.

In many systems for odd characteristic fields we use fields of prime order equal to

$$q = 2^n + c.$$

If c is small this gives a very efficient way of performing the field operations with no known loss of security. We shall restrict to fields of the above form with $n \leq 255$ and $1 \leq c \leq 255$. There are 174 such primes with $150 \leq n \leq 255$ and $1 \leq c \leq 255$, which we list in Table 1. Hence we have more choice of odd characteristic fields than even characteristic fields. In addition we will require 15 bits to represent q, 8 bits for n and 7 bits for c, since clearly c must be odd.

Since both odd and even fields can be represented using 15 bits we can represent the choice of field in 16 bits by using a single additional bit to specify whether we are in the odd or even case. We shall call the resulting bit string f.

1.2. **The Elliptic Curve.** Having chosen a field, we now find an elliptic curve over $\mathbb{F}_q$ using Schoof's algorithm [3]. Notice that we are not specifying a special elliptic curve and so we are not compromising security with our choice of elliptic curves. The elliptic curve is given by an equation of the form

$$\begin{array}{ll} Y^2 = X^3 + aX + b & q \text{ odd}, \\ Y^2 + XY = X^3 + a_2X^2 + a_6 & q \text{ even}. \end{array}$$

Since in the even characteristic case we have insisted that n is odd, we can choose $a_2 \in \{0, 1\}$.

In the odd characteristic case we can assume that a will sit in an eight bit word. To see why this is so, notice that $Y^2 = X^3 + aX + b$ is isomorphic to $Y^2 = X^3 + au^4X + bu^6$ for some $u \in \mathbb{F}_q$. Now if $q \equiv 1 \pmod 4$ then we have a 25 percent change of replacing a by any number a', by simply trying to extract the fourth root of a'/a. If $q \equiv 3 \pmod 4$ then we have an even better chance, namely 50 percent, of this working for any given a'. If we insist that a is specified by only 8 bits then we have an, at most,

$$.75^{256} \approx 10^{-32}$$

chance that our curve found by Schoof's algorithm cannot be put in a form with $a \notin \{1, \ldots, 256\}$. If we are so unlucky we could always try and find another curve.

Since b and a_6 both require n bits to specify them, we can represent the elliptic curve E using only $n + 8$ bits. We call the resulting string e.

1.3. **The Group Order.** We assume that the curve order is divisible by a large prime l and

$$N_q = \#E(\mathbb{F}_q) = \begin{cases} l & q \text{ odd}, \\ 2l & q \text{ even and } a_2 = 1, \\ 4l & q \text{ even and } a_2 = 0. \end{cases}$$

This is common practice and such elliptic curves are quite easy to find using Schoof's algorithm for the values of q in use today. By Hasse's theorem we have that $t = q + 1 - \#E(\mathbb{F}_q)$ is bounded by $|t| \leq 2\sqrt{q}$. Hence to specify the group order, and hence l, requires $2 + n/2$ bits. The resulting bit string we denote by o.

1.4. **The Generating Point.** We now need to specify a point which generates a subgroup of order l. We pick a random bit string x of length 7 and consider the field element $\{x\}$ represented by x. With probability $1/2$ we can find a field element $y \in \mathbb{F}_q$ such that $(\{x\}, y) \in E(\mathbb{F}_q)$. If we then compute

$$P = [N_q/l](\{x\}, y)$$

then with probability around 2^{2-n} we have $P = \mathcal{O}$ and we reject this value of x and start again. Otherwise we have found a point P which is a non-trivial element of order l in $E(\mathbb{F}_q)$. So we can represent the generating point, P, by compressing the elliptic curve point $(\{x\}, y)$ into a bit string, p, of length 8 bits. With overwhelming probability such a method is guaranteed to find a suitable generating point.

1.5. **The Public Key.** We finally need to construct the public key. We produce a random number k, the private key, with $1 < k < l$ and compute the public key

$$Q = [k]P.$$

By compressing Q we can represent Q by a bit string q of length at most $n + 1$ bits.

Hence to represent all the parameters of our elliptic curve public key we require the bit string

$$X = f \mid\mid e \mid\mid o \mid\mid p \mid\mid q$$

of length $t = 16 + (n+8) + (2 + n/2) + 8 + (n+1) = 35 + 5n/2$ bits.

2. The DSA Public Key

Using the prime l above, which is of order 2^n, we form the number

$$T = l\{X\}^{\lfloor (m-n)/t \rfloor + 1} + 1.$$

Since $\{X\}$ is a number of bit length t the bit length of T is

$$\log_2(T) \approx \log_2(l) + \frac{m-n}{t} \log_2\{X\} \approx n + m - n = m.$$

We now repeatedly add l onto T, until we come across a prime number, $P = T + \lambda l$. By the prime number theorem such a prime number will occur, on average, after m such additions, so we can assume that the bit length of λ is $\log_2 m$. The resulting prime P will be the field in which our DSA public key will lie, clearly it is deterministically derived from the bit string X and the prime l will divide $P-1$. We can assume that l^2 does not divide $P-1$, which is such a rare event we can safely ignore it.

We now choose a random number g of bit length less than 8 and compute

$$h \equiv g^{(P-1)/l} \pmod{P}.$$

With probability 2^{-n} we obtain 1, in which case we choose another random g of bit length less than 8 and repeat the calculation until h is not equal to one. The number h is then a generator of the subgroup of $\mathbb{F}_P$ of order l.

Our DSA public key is then the number

$$z = h^k \pmod{P}.$$

Notice that we have used the same private key for the ECC and DSA schemes, this is not necessary and is just done here for convenience and so we do not need to find another letter. In addition it means the holder of the private key is less likely to forget which private key corresponds to ECC and which to DSA.

We let Z denote the bit string $[z]$ of length approximately m bits and set

$$Y = X \mid\mid [\lambda] \mid\mid [g].$$

So Y is a bit string of length $43 + 5n/2 + \log_2 m$ bits.

3. The RSA Public Key

Using the method in [2], which apparently has been known for a long time, we can construct an m bit RSA modulus, N, which contains the bit string Y as either the leading or trailing portion of $[N]$. This is possible assuming

$$m > 86 + 5n + 2\log_2 m.$$

As the RSA public exponent we make the standard choice of 65537. Finally we represent all three public keys using the bit string

$$[N] \mid\mid Z$$

of length $2m$. In 'real life' we have typical values of $n = 160$ and $m = 1024$, which satisfy the above inequality with ease.

4. Security

Clearly our choice of elliptic curve parameters do not affect the security of the system. Neither does our choice of DSA parameters, although the choice of such a value of P could possibly lead to small degradations in performance when compared to sparse values of P used in some DSA signature implementations.

The discussion in [2] should convince the reader that the resulting RSA public keys provide no loss in security compared to more general RSA moduli.

References

[1] I.F. Blake, G. Seroussi and N.P. Smart. *Elliptic Curves in Cryptography.* To appear, CUP, 1999.

[2] A. Lenstra. Generating RSA moduli with a predetermined portion. In *Advances in Cryptography, ASIACRYPT 98*, 1–10. Springer–Verlag, LNCS 1514, 1998.

[3] R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comp.*, **44**, 483–494, 1985.

[4] J.R.R. Tolkien. *The Lord of the Rings.* George Allen and Unwin, 1954.

[5] ANSI X9.62 Public Key Cryptography for the financial services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA) Draft Standard, 1998.

Hewlett-Packard Laboratories,Filton Road, Stoke Gifford, Bristol, BS12 6QZ, U.K.
E-mail address: `nsma@hplb.hpl.hp.com`

TABLE 1. The 174 primes of the form $2^n + c$ with $150 \leq n \leq 255$ and $1 \leq c \leq 255$

n	c		n	c
150	147, 163		151	253
152	27		153	45, 115, 133
154	97, 189, 253		155	7
156	19, 49, 225		157	117, 135, 151, 231
158	85		159	87, 117
160	49, 55, 79, 85, 97, 133, 135, 225		161	105, 177
162	7, 37, 73, 129		164	87, 99, 189, 193
165	15, 115, 193		166	49, 207
167	27, 63		168	205, 249
169	57, 123, 153		170	129, 223
171	133		172	27, 67, 103
173	21		174	15, 49, 169
175	235		176	67, 189
177	7, 157, 213, 247		178	33, 75, 169, 229, 247
180	33, 177		181	57, 223
182	7, 217		184	163
185	217		186	49, 57, 109, 127, 163
189	3		190	67, 163, 183
192	43, 225, 255		193	25, 97, 193, 237
194	67, 189, 199		195	115
196	69		197	133, 235
199	81		200	25
202	79, 153, 163, 207, 247		204	73
205	223		206	79
207	175		208	133, 135, 193
209	43, 111, 121		210	127, 133
211	57, 81, 163, 223		212	63, 67, 177
213	157		214	117
215	45		216	159, 189
218	43		219	127
221	13, 91		222	67, 169, 205, 249
223	61, 163, 247		225	157
227	37		228	45, 73, 117
229	27, 81		230	15, 189
231	91		232	93
233	87		235	27
236	219		238	249
239	15		240	93
243	241		244	97, 99
246	15, 85		247	231
249	241		251	55, 81, 223
255	105			