Scalar Blinding on Elliptic Curves with Special Structure

Scott Fluhrer

Cisco Systems

August 11, 2015

1 Abstract

This paper shows how scalar blinding can provide protection against side channel attacks when performing elliptic curve operations with modest cost, even if the characteristic of the field has a sparse representation. This may indicate that, for hardware implementations, random primes might not have as large of an advantage over special primes as previously claimed.

2 Background

Elliptic curves are a useful tool within cryptography. An Elliptic Curve is a mathematical group, and some Elliptic Curves have this useful property: given a group member (point) G and an integer k, the point $H = kG$ can be computed in time proportional to $\log k$; however given two points G, H, computing the integer k such that $H = kG$ takes time proportional to $\sqrt{k}$ (using the best known algorithm). By selecting k (and the Elliptic Curve) to be an appropriate size, we can make finding H given k and G (known as point multiplication) relatively quick, while making finding k given H and G (known as the discrete logarithm problem) infeasible.

The most common Elliptic Curves used in practice are defined over a prime field $GF(p)$, for a large (perhaps 256 bit) prime p (the characteristic) that we pick when we generate the curve. One thing this means in practice is that when we compute a point multiplication kG, we spend the majority of the time computing the modular multiplication $a \times b \pmod{p}$ for two values $a, b \in GF(p)$.

To accelerate this operation, one approach is to select a prime of the form $p = 2^e - c$, where c has a simple representation in binary, and is considerably smaller than 2^e. This allows us to accelerate the computation of the modular reduction by taking advantage of the identity:

$$a \cdot 2^e + b \equiv a \cdot c + b \pmod{2^e - c}$$

If the binary representation of c is simple enough, we can compute $a \cdot c$ without doing a full multiply, and hence we can compute this modular reduction significantly faster than we could for an arbitrary prime. This allows us to compute the modular multiplication of two numbers in not much more time than it takes to perform a bignum multiplication of those two numbers. Examples of Elliptic Curves that allow this optimization include the so-called NIST curves[7], Curve25519[2], and the Microsoft NUMS curves[1].

If we instead select a random prime without such a special structure (such as was done when defining the Brainpool curves[6]), there are still some optimizations we can do beyond the obvious 'perform a multiply, and then perform a generic modulo reduction'. We can implement Montgomery Multiplication, which replaces the modulo operation with some multiplies and shifts; the net result is that a multiplication followed by a modular reduction can be done in the time of approximately two bignum multiplications; in other words, modular multiplication of a special form prime can be done approximately twice as fast as an arbitrary prime.

If this were the only consideration, the choice of whether we should use a prime with special structure would be an obvious one. However, there is another issue. Sometimes, Elliptic Curves are implemented by hardware that needs to operate in hostile environments, and can be expected to be subject to side channel attacks, such as Differential Power Analysis. In these types of attacks, the cryptanalyst runs the system, performing the same operation repeatedly, and takes careful measurements of power consumed (or EM radiation emitted) on a cycle-by-cycle basis; by statistically combining these measurements, the attacker hopes to recover the internal states (which includes the private key).

To combat these sorts of attacks, one of the strategies that we need to employ is blinding; we include random data in our computations, and while the end results is independent of the random value, the intermediate values are strongly dependent, and thus the correlations between the intermediate states and anything that the attacker wants (such as the private key) is much weaker.

One such method of blinding Elliptic Curve calculations (first published by Coron[4]) takes advantage of a property of Elliptic Curve groups; we know an integer n such that $nG = 0$ (this value n is known as the order of the point G). Coron's method to compute kG would be to select a random value t and computing first $nt + k$, and then $(nt + k)G$. Everytime we would perform a point multiplication, we would select a random t, and hence the bits of the integer we're giving to the point multiplication logic are independent of the integer k we're actually multiplying by. Because the time taken by the point multiplication is proportional to the log of the integer, this blinding method increases the time by a value proportional to be size of t; if we select (for example) a 64 bit t, this increase is relatively small compared to the time we would have taken computing kG anyways.

However, as observed in [3], this straight-forward approach turns out not to work as well from primes with special structure. The order of the curve is

always within the Hasse Interval; that is, we have:

$$p + 1 - 2\sqrt{p} \leq hn \leq p + 1 + 2\sqrt{p}$$

where h is the cofactor of the curve, and is usually a small power of 2. What this implies is that $n \approx p/h$, and if the upper bits of p have a sparse structure, then the upper bits of n will as well. In other words, if p is a special structure prime, and if $t < \sqrt{p}$, then some of the bits of $nt + k$ will be strongly correlated to some bits of k, and hence this supposed blinding operation does leak some information about k. This would appear to imply that primes with special structure would require significantly larger t values than random primes. And because the time taken to do a point multiplication is proportional to the length of the integer being multiplied, this would appear to imply that primes with special structure can be slower than random primes when implemented on hardware.

3 Scalar randomization with fields with special structure

One common way to compute the point multiplication kG is to express k in base b, as:

$$k = d_i b^i + d_{i-1} b^{i-1} + d_{i-2} b^{i-2} + ... + d_2 b^2 + d_1 b^1 + d_0 b^0$$

and then perform the computation:

$$kG = d_0 G + b \cdot (d_1 G + b \cdot (d_2 G + ... + b \cdot (d_{i-1} G + b \cdot (d_i G)))...)))$$

In the straight-forward way, this takes $b - 2$ additions to evaluate the values $(0G, 1G, 2G, ..., (b-1)G)$, and then i cycles of multiplying an intermediate point by the small integer b and adding the point that corresponds to the next digit. There are a number of variants to this approach, both to try to achieve constant time, and to reduce the number of additions required (for example, by using the digits in the range $(-b/2, b/2)$, taking advantage of the fact that we can compute the inverse $-G$ cheaply within an Elliptic Curve group).

The obvious choice is to make b a power of two (so $b = 2^m$); this yields two immediate advantages:

- If k is already expressed in binary, the decomposition into the form $(d_i, d_{i-1}, ..., d_0)$ is just extracting bits
- The operation of multiplying a point by b can be efficiently done by doing m point doublings

However, if we look at that value of n expressed in such a base b if the prime has special structure, we see a regular pattern. For example, the value of n

for the Elliptic Curve Curve25519 (which has the special form prime $2^{255} - 19$) expressed in base $b = 32$ is:

04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 27 27 28 29 29 08 23 23 19 19 11 05 16 04 19 03 03 09 14 15 11 20 31 13

Note the long string of zero's at the beginning; these are what makes scalar randomization less effective. As one might expect, $tn \approx t2^{252} + t2^{124.4}$, and if $t < 2^{128}$, then bits 251 and below of $k + nt$ will be strongly correlated to the corresponding bits of k (because the bits of nt with nontrivial contributions to those bits of the sum will be zero). Other special form primes don't have quite as striking of a form (I chose Curve25519 because the form of its n makes it quite obvious), but the other special form primes also have long strings of 0's or $b - 1$ digits at the beginning, which yields the corresponding weakness.

However, let us consider what happens if we consider a b which is not a power of 2. For example, if we were to take the same n expressed in base $b = 48$, we get:

01 28 34 41 23 00 42 31 16 04 20 44 19 13 01 16 37 17 42 41 16 36 22 39 40 06 14 29 09 44 17 13 24 04 31 28 46 05 19 16 46 07 18 12 42 13

Here, we don't get any regular pattern, and this value would, at first glance, appear random. And, in fact, if we look at the digits of $k+nt$ (for modest random t) expressed in base 48, we don't detect any correlation between the digits of that and the digits of k expressed in base 48. This implies that blinding with this value (in base 48) is likely as effective as blinding based on a random prime in base 32.

This is not only true of Curve25519; the same thing happens for the special prime curve P256, which has n in base $b = 32$ as:

01 31 31 31 31 31 31 16 00 00 00 00 00 03 31 31 31 31 31 31 31 31 31 31 31 31 29 28 28 27 29 10 27 09 24 23 19 26 02 15 07 14 14 10 24 11 30 06 06 09 10 17

(where the streaks of 31's and 00's cause a similar effect to the long string of 00's in the Curve25519 n). In contrast, in base $b = 48$, it is:

25 27 29 39 32 12 33 29 24 47 03 05 12 17 33 24 45 02 26 34 11 37 13 27 43 25 45 05 37 02 04 08 39 09 42 06 04 27 15 01 47 30 12 25 23 01

Of course, once we consider nonpower-of-2 bases, we lose the two advantages that we formally had; let us examine how we can accommodate the loss of these two advantages.

4 Costs of point multiplication using a non-power-of-2 base

The first thing we want to look at is how much more expensive it is to use a base which is not a power of 2. After all, a base which is a power of 2 allows us to implement the fixed multiplication by b by using a handful of doublings, which is the most efficient method possible. However, it turns out there are other bases that are almost as cheap.

To make a concrete comparison, we'll outline the costs with both a power-of-2 base, and a nonpower-of-2 base. In both cases we'll assume that we're dealing with an Elliptic Curve with a 256 bit subgroup, and that we'll use a 64 bit blinding value r, yielding an exponent which is 320 bits long. We'll use the radix method, using a balanced representation of the digits (that is, the digits are values in the range $[-b/2, b/2]$, and we'll assume that the addition-by-0 is masked somehow (whether the low-level addition routines handle it without a special case, or because in that case we'll add by an arbitrary point and discard the result).

To implement this using radix $b = 32$ (which is optimal over all powers-of-2 in this scenario), we'll first compute the digits $(-16G, -15G, ..., 15G, 16G)$ with 7 doublings, 7 additions[1] and 15 negations[2]. Then, we implement the actual addition chain; in this case, 320 bits is 64 digits[3]; this is 63 rounds of multiplying the current point by 32 (which is 5 doublings), and adding in the next digit (which is a single addition); this step takes us 315 doublings, and 63 additions, for a total of 322 doublings, 70 additions, and 15 negations.

Now, let us look at the radix $b = 48$ case; computing the digits $(-24G, -23G, ..., 24G)$ requires 11 doublings, 11 additions and 23 negations. Then, we implement the actual addition chain; in this case, 320 bits can be expressed in 58 base-48 digits; this is 57 rounds of multiplying the current point by 48 (which can be implemented by 5 doublings and an addition), and adding in the next digit (which is a single addition); this step requires 285 doublings, and 114 additions, giving us a grand total of 296 doublings, 125 additions and 23 negations.

If our Elliptic Curve representation makes addition as cheap as doubling (which some do), and we ignore the negations (which are comparatively cheap), then the base-32 method turns out to be 7.3% faster than the base-48 method. If we instead assume that a doubling is 80% of the cost of an addition (another common assumption), then the base-32 method turns out to be 10.4% faster than the base-48 method. In other words, from this perspective, we can implement the blinding on a special format prime, and be within 7-10% of the performance of a random prime. These results are fairly stable if we tweak our assumptions (e.g. change the size of the group order or the size of t)

When we multiply by a fixed point (for example, the curve generator G), one common optimization is to precompute various multiples $k_i G$ for various values k_i, and use those to accelerate the point multiplication process. While this works even better with bases that aren't powers of 2 (as we no longer need to perform multiplications by the fixed value b, and not restricting ourselves to power of 2 bases often allows us to fine-tune the base better), this technique does require us to store some precomputed tables, and hence is less likely to be considered useful for a hardware implementation. Hence, other than this quick

[1] In some elliptic curve representations, the operation of adding a point to itself (doubling) is cheaper than adding two distinct points (additions), hence we track those two operations separately

[2] Negation is such a cheap operation within Elliptic Curves that we typically don't count it

[3] Normally, it would be 65, because of the signed representation; however we could assume that t is a signed value as well, and that would bring us to the 64 digit level

note, we will ignore the possibility.

5 Working with exponents in nonpower-of-2 bases

The other advantage that we discard if we work in an odd base is the fact that have to do something to convert our multiplier (which is in binary) into the base. The obvious approach would be to compute $k + tn$ in binary, and then do a base conversion into our desired format. The problem with that is that the digits of $k + tn$ will be expressed as a temporary, and thus will be subject to the same side channel attacks that we are trying to avoid.

However, there are ways to avoid this issue. To demonstrate this, we will review two different representative protocols, and give a possibility of how this can be addressed in both of them. These are certainly not the only algorithms we would like to do point multiplication with; however these two should demonstrate the range of options that are possible.

One note: the above point multiplication analysis assumed a balanced base-48 notation, while the below will assume a standard base-48 notation. This is because standard base-48 notation is easier to do arithmetic in, while it is not difficult to convert to a balanced notation, if that would be helpful to the point multiplication logic.

5.1 The case of ECDH/ECIES

The easiest case to handle is the case of ECDH and ECIES. In these cases, the integer that we multiply by is just a random number that we pick, and has no correlation with any other value (with the exception that we multiply two different points by the same integer).

In this case, we can avoid the initial problem (how do we convert the binary integer into base-48 without giving a side channel attack) simply by selecting the initial random number in base-48. That is, we never explicitly express the multiplier in binary; instead, we pick a series of random values between 0 and 47, and use those as the base 48 digits. As for how to select such a random value between 0 and 47, it can be noted that a rejection method (where you generate 6 random bits as a value between 0 and 63 repeatedly until the selected value is in range) is safe; it is not constant time, however the time taken is uncorrelated the value eventually selected, and hence the timing doesn't leak any data we care about.

The other step is to apply the blinding factor, that is, compute $k + nr$ is a way that has minimal correlation to k; this can be done by computing nr in binary, and then converting that to base-48 (and as nt has no correlation to k, we have less concern about leaking data during the conversion process); once that is done, we can perform a constant time addition of nr to k in base-48.

5.2 The case of ECDSA Signature Generation

Another case is where we are attempting to implement ECDSA, and in particular, the signature generation process. Here, we pick a random value k, and compute both the x-coordinate of $r = kG$ (for the generator point G), and $s = k^{-1}(z + rd)$ (where z, r and d are integers).

Because we need to do computations on k beyond using it to do point multiplication, the strategy of generating it in base-48 is less attractive. The obvious idea of computing $k + bn$, and then converting that to base-48 (for a random blinding factor b), and then using our base-48 style point multiplication also doesn't work, because we initially express $k + bn$ in binary, and the the intermediate bits of that will be correlated to the bits of k, and that's what we're trying to avoid.

However, it is still possible by adding a few extra blinding factors. Consider this randomized procedure:

- Select random values a, b from the range $[0, n)$, u from the range $[1, n)$ and t from the range $(0, 2^{64})$ (t will be the Coron blinding factor).
- Compute $t_1 = a + tn$
- Convert both t_1 and b into base-48, giving t_3, and t_4 Add t_3 and t_4 together as base-48 numbers, giving t_5
- Compute $t_5 G$ (using the base-48 point multiplication algorithm outlined earlier), with r being the x-coordinate of the resulting point
- Compute $u_1 = au \bmod n$ and $u_2 = bu \bmod n$
- Compute $u_3 = u_1 + u_2 \bmod n$, and then compute $u_4 = u_3^{-1} \bmod n$
- Compute $s = u_4 u(z + rd)$ (where z, r and d have the normal meanings for ECDSA; z is the hash, r is the x-coordinate computed previously, and d is the ECDSA private key).

If you go through this procedure, it should be clear that this is the ECDSA signature algorithm (with $k = a + b \bmod n$). It should also be clear that the value of k is selected without a bias. In addition, the internal bits of all the intermediate values are uncorrelated to the bits of k (in fact, except for t_5, the value of all intermediates are distributed independently of k), hence we have achieved blinding against first order side channel attacks. In addition, the operations that we have added over the straight-forward ECDSA signature generation with Coron blinding (generating $2 \log n$ additional random bits, three additional multiplications, one additional binary addition, one addition in base-48, and two base conversions) are relatively cheap (say, compared to computing the multiplicative inverse), and so we haven't increased the expense significantly.

6 Summary

In Requirements for Standard Elliptic Curves[5], the designers of the Brainpool curves gives two justifications for selecting a random prime; one is that a special prime does not give an special performance advantages in their environment, and secondly, the special primes make blinding operations more difficult. This paper has shown that the effort required to perform blinding when using a special prime has been overestimated; there appears to be ways to perform the required blinding at modest additional expense.

References

[1] C. Costello P. Longa-M. Naehrig B. Black, J. Bos, 2014.

[2] Daniel Bernstein. A state-of-the-art Diffie-Hellman function. Web Site, 2005.

[3] Mathieu Ciet and Marc Joye. (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In *Information and Communications Security*, pages 348–359. Springer Science Business Media, 2003.

[4] Jean-Sébastien Coron. Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems. In *Cryptographic Hardware and Embedded Systems*, pages 292–302. Springer Science Business Media, 1999.

[5] Dr. Joern-Marc Schmidt Dr. Torsten Schuetze Dr. Manfred Lochter, Dr. Johannes Merkle. Requirements for Standard Elliptic Curves, 2014.

[6] M. Lochter and J. Merkle. Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. Technical report, mar 2010.

[7] NIST. RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE. NIST Web Site, 1999.