Ed3363 (HighFive) – An alternative Elliptic Curve

Michael Scott

MIRACL Labs
mike.scott@miracl.com

Abstract. We propose a new Elliptic curve at a security level significantly greater than the standard 128 bits, that fills a gap in current proposals while bucking the expected security vs cost curve by exploiting the new trick recently described by Granger and Scott. This essentially reduces the cost of field multiplication to that of a field squaring.

1 Introduction

If a non-cryptographer were asked to guess how much stronger TOP SECRET cryptography is compared with commercial strength cryptography, I would imagine that most would suggest a hundred times, maybe a thousand times, maybe even a million times. But I think many would be surprised that in fact its at least 9,223,372,036,854,775,808 times, a number so big that it is unspeakable. But thats the difference between an elliptic curve at the 128-bit level of security and an elliptic curve at the 192-bit level. Most might consider this a little excessive.

One hard-to-refute argument against using a higher level of security is that if it doesn't cost very much – hey, why not. In fact the extra cost of going from 128-bit to 192 bit security depends a lot on the type of cryptography we are considering.

For a symmetric block cipher like the AES the cost is just another two rounds on top of the standard ten rounds, so there is a mere 20% overhead.

For elliptic curve crypto, assuming the simplest algorithm for field multiplication of quadratic complexity, the overhead is 237.5%

For RSA crypto, assuming Karatsuba methods for field multiplication, the overhead is a whopping 968%

These costs can be significant. Its one thing for someone to propose the use of an elliptic curve at the TOP SECRET level, as they may have the spare capacity to implement it on their hardware, but spare a thought for the person tasked to build a compatible product on a less powerful platform. If unnecessarily high levels of security start excluding certain devices from being secured, they may do more harm than good.

Another downside of the bigger-is-always-better position is that it might appear to endorse snake oil crypto merchants, and their arguments for 1,000,000 bit security products.

Also as we are all aware realistically an attacker will not take on the crypto at even the 128-bit level of security – as recent revlations have shown other methods of attack on an implementation come in at a much lower price than a full frontal assault on the crypto, either mathematical or brute force.

So where did these TOP SECRET recommendations come from? An interesting insight is offered in the recent paper by Miele and Lenstra [7] who state that "With 128-bit security more than sufficient for the foreseeable future, it is not clear either what purpose is served by higher security levels, other than catering to TOP SECRET 192-bit security from [8]. In this context it is interesting to note that 256-bit AES, also prescribed by [8] for TOP SECRET, was introduced only to still have a 128-bit secure symmetric cipher in the post-quantum world (...), and that 192-bit security was merely a side-effect that resulted from the calculation $(128 + 256)/2$ (...). In that world ECC is obsolete anyhow."

Recently there has been a drift away from the very highest suggested security level of AES-256, arguably as a belated recognition that this has no place in a pre-quantum world. Proposals for new standard high security elliptic curves have tended to fall into the gap between AES-192 and AES-256. For example Curve41417 [2] and Mike Hamburg's Ed448 "Goldilocks" curve [6].

Nevertheless there is an undeniable demand for a higher level of security. Here we attempt to satisfy this demand by suggesting a new elliptic curve, which because of the fortunate form of its modulus, holds out the possibility of incurring only a very modest extra cost, while being 1,099,511,627,776 times more secure. That is about a trillion times stronger than the commercial standard.

2 Picking a curve

One downside of elliptic curve cryptography is the need to base our protocols on a particular curve. This issue simply doesn't arise for RSA and finite field methods. Its an extra hurdle of trust we need to get over. Who chose it, how and why?

Clearly we have to be selective about the elliptic curves that we choose to use. Pick one completely at random, and its group order will probably be composite and hence any scheme built upon it can be easily cryptanalysed. However just how selective we should be has been the subject of much debate. Typically we would like to select on the basis of better performance and better security against even the most obscure attacks. The counter-argument is that by misfortune we may pick our curve from a subset that has a currently unknown weakness that doesn't apply to the larger enclosing family of curves. Often these concerns cannot be scientifically answered, as we just don't know. Its a judgement call, made easier only by the lapse of time during which no weakness has been found. But the same is true of much of cryptography – belief in the security of the AES for example is not based on any scientific proof, but rather on the basis of the lapse of time with no significant weakness having been found by very many clever people studying it.

Here are some classes of special curves that have been proposed.

– Endomorphism curves (faster)
– CM curves (quicker to generate)
– Curves with special primes (faster)
– Curves with strong twists (more secure)

Our judgement call is that there is nothing wrong with any of these curves. However endomorphism curves have patent issues which will take a few years to eliminate (but those years will eventually only add to our confidence in them). Since we are not planning on generating random curves in real time [7], there is no point in using the CM method to find them. Our curve will use a special prime for the performance advantage that it brings, and will have a strong twist as although this property hugely increases the time it takes to find a suitable curve we are convinced that this adds security against plausible attacks.

Since we are proposing a new curve it makes sense to pick one with the best elliptic curve parameterisation, that is an Edwards curves [1]. A minor issue with the Edwards parameterisation is that the curve order is at best four times a prime, $4q$. Since our choice of modulus $p = 5 \bmod 8$, if the curve order is $0 \bmod 4$ then the twist order must be at best $0 \bmod 8$, and visa versa. Points (x, y) on an Edwards curve where $x, y \in \mathbb{F}_p$ satisfy the equation

$$y^2 + x^2 = 1 + dx^2y^2$$

A sine qua non of trusted elliptic curve selection is that the method of selection should be absolutely transparent, such that the actual curve chosen is largely outside of the control of the proposer. This quality is known as non-rigidity [4]. Here we follow the now standard approach of searching for the curve with the minimum absolute value for a non-square d such that both the curve and the twist have orders four or eight times a prime.

However we had another criteria in mind which was to exploit the Granger-Scott idea [5] for fast modular arithmetic. This guided our choice of the modulus p. We also wanted a curve that would be implemented just as conveniently on both 32-bit and 64-bit architectures. Whereas a nice fit to a 64-bit architecture facilitates record-setting, in our view a good fit to a 32-bit architecture is probably more important in practise. As a bonus we wanted our implementation to be efficient even when written in a portable high-level language.

3 Our Modulus

Our chosen modulus is $p = 2^{336} - 3$. Note that elements modulo this 336-bit prime can be represented either by 6 digits to the base 2^{56} on a 64-bit architecture, or by 12 digits to the base 2^{28} on a 32-bit architecture. Our description here will assume the former, although the extension to the latter is straightforward. Given the non-saturated representation, we can add and subtract multiple digits and products of digits without fear of overflow.

As pointed out in [5] the product of two such numbers modulo $2^{336} - 1$ can be written simply as

$$\begin{aligned}
&[x_0y_0 + x_1y_5 + x_2y_4 + x_3y_3 + x_4y_2 + x_5y_1,\\
&\ x_0y_1 + x_1y_0 + x_2y_5 + x_3y_4 + x_4y_3 + x_5y_2,\\
&\ x_0y_2 + x_1y_1 + x_2y_0 + x_3y_5 + x_4y_4 + x_5y_3,\\
&\ x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0 + x_4y_5 + x_5y_4,\\
&\ x_0y_4 + x_1y_3 + x_2y_2 + x_3y_1 + x_4y_0 + x_5y_5,\\
&\ x_0y_5 + x_1y_4 + x_2y_3 + x_3y_2 + x_4y_1 + x_5y_0].
\end{aligned} \tag{1}$$

where x_i and y_i are the digits of the numbers being multiplied. The clever bit is the observation that, given $s = \sum_{i=0}^{5} x_i y_i$ this can be written as

$$\begin{aligned}
&[s - (x_1 - x_5)(y_1 - y_5) - (x_2 - x_4)(y_2 - y_4),\\
&\ s - (x_1 - x_0)(y_1 - y_0) - (x_2 - x_5)(y_2 - y_5) - (x_3 - x_4)(y_3 - y_4),\\
&\ s - (x_2 - x_0)(y_2 - y_0) - (x_3 - x_5)(y_3 - y_5),\\
&\ s - (x_2 - x_1)(y_2 - y_1) - (x_3 - x_0)(y_3 - y_0) - (x_4 - x_5)(y_4 - y_5),\\
&\ s - (x_3 - x_1)(y_3 - y_1) - (x_4 - x_0)(y_4 - y_0),\\
&\ s - (x_3 - x_2)(y_3 - y_2) - (x_4 - x_1)(y_4 - y_1) - (x_5 - x_0)(y_5 - y_0)].
\end{aligned} \tag{2}$$

Observe that the number of partial products to be calculated has fallen from 36 to 21, which is the same number required for a modular squaring using standard methods. However we are not quite done yet, as our modulus is the prime $2^{336} - 3$, not the non-prime $2^{336} - 1$.

The actual product modulo $2^{336} - 3$ should be

$$\begin{aligned}
&[x_0y_0 + 3x_1y_5 + 3x_2y_4 + 3x_3y_3 + 3x_4y_2 + 3x_5y_1,\\
&\ x_0y_1 + x_1y_0 + 3x_2y_5 + 3x_3y_4 + 3x_4y_3 + 3x_5y_2,\\
&\ x_0y_2 + x_1y_1 + x_2y_0 + 3x_3y_5 + 3x_4y_4 + 3x_5y_3,\\
&\ x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0 + 3x_4y_5 + 3x_5y_4,\\
&\ x_0y_4 + x_1y_3 + x_2y_2 + x_3y_1 + x_4y_0 + 3x_5y_5,\\
&\ x_0y_5 + x_1y_4 + x_2y_3 + x_3y_2 + x_4y_1 + x_5y_0].
\end{aligned} \tag{3}$$

Which can be calculated at the cost of some extra additions as

$$\begin{aligned}
&[3[s - (x_1 - x_5)(y_1 - y_5) - (x_2 - x_4)(y_2 - y_4)] - 2x_0y_0,\\
&\ 3[s - (x_2 - x_5)(y_2 - y_5) - (x_3 - x_4)(y_3 - y_4)] - (x_1 - x_0)(y_1 - y_0) - 2(x_0y_0 + x_1y_1),\\
&\ 3[s - (x_3 - x_5)(y_3 - y_5)] - (x_2 - x_0)(y_2 - y_0) - 2(x_0y_0 + x_1y_1 + x_2y_2),\\
&\ s - (x_2 - x_1)(y_2 - y_1) - (x_3 - x_0)(y_3 - y_0) - 3[(x_4 - x_5)](y_4 - y_5) + 2(x_4y_4 + x_5y_5),\\
&\ s - (x_3 - x_1)(y_3 - y_1) - (x_4 - x_0)(y_4 - y_0) + 2x_5y_5,\\
&\ s - (x_3 - x_2)(y_3 - y_2) - (x_4 - x_1)(y_4 - y_1) - (x_5 - x_0)(y_5 - y_0)].
\end{aligned} \tag{4}$$

Further manual optimization of these equations provide some extra savings. Finally the digits in our product require carry propagation, and final reduction. We omit the details.

4 Our New Curve

Following the process described above we find the Edwards curve

$$y^2 + x^2 = 1 + 11111x^2y^2$$

Fortuitously our constant d has a nice easily memorised form – five ones, like the fingers on your hand. Hence the name. The number of points on the curve is $8q$ where q is the prime

$$\texttt{200000000000000000000000000000000000000000071415FA9850C0BD6B87F93BAA7B2F95973E9FA805}_{16}$$

A generator point of order q is

$$[\texttt{C}_{16}, \texttt{C0DC616B56502E18E1C161D007853D1B14B46C3811C7EF435B6DB5D5650CA0365DB12BEC68505FE8632}_{16}]$$

5 Implementation

Source code for both 32 and 64-bit versions may be found here.[1]

Observe that the 32-bit code is effectively written in standard portable C. The 64-bit code requires partial support for a 128-bit integer type, which unfortunately is not part of the standard, although the popular Gnu GCC compiler does support it. Translation of the 32-bit code to any other language should be straightforward.

The code implements variable point multiplication, which is the core operation at the heart of elliptic curve cryptography. To avoid side-channel attacks (including cache attacks) we use the fixed window method described in [2] using a window of size 4. Points are represented in extended {X,Y,Z,T} coordinates and manipulated using formulae from the EFD [3]. For modular inversion as required to convert from projective to affine coordinates we use a constant time method based on the well known fact that $x^{-1} = x^{p-2} \bmod p$, manually optimized to minimize the number of field multiplications.

On a 64-bit Intel Haswell processor point multiplication takes 333,000 cycles approximately. On a 32-bit ARM Cortex A7 based Raspberry pi version 2, clocked at 900MHz, it takes 3.7ms. We would assume that these timings could be improved upon.

Thanks to Rob Granger for inspiring a name for the curve.

[1] See `indigo.ie/~mscott/ed336_32.cpp` and `indigo.ie/~mscott/ed336_64.cpp` for the 32-bit and 64-bit code respectively.

References

1. D. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *Asiacrypt 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer-Verlag, 2007.
2. D.J. Bernstein, C. Chuengsatiansup, and T. Lange. Curve41417: Karatsuba revisited. In *CHES 2014*, volume 8731 of *Lecture Notes in Computer Science*, pages 316–334. Springer-Verlag, 2014.
3. D.J. Bernstein and T. Lange. Explicit formulas database, 2015. `http://hyperelliptic.org/EFD`.
4. D.J. Bernstein and T. Lange. Safecurves: choosing safe curves for elliptic-curve cryptography, 2015. `http://safecurves.cr.yp.to`.
5. R. Granger and M. Scott. Faster ECC over $\mathbb{F}_{2^{521}-1}$. In *Public-Key Cryptography – PKC 2015*, volume 9020 of *Lecture Notes in Computer Science*, pages 539–553. Springer Berlin Heidelberg, 2015.
6. M. Hamburg. Ed448-Goldilocks, a new elliptic curve. Cryptology ePrint Archive, Report 2015/625, 2015. `http://eprint.iacr.org/2015/625`.
7. A. Miele and A. K. Lenstra. Efficient ephemeral elliptic curve cryptographic keys. Cryptology ePrint Archive, Report 2015/647, 2015. `http://eprint.iacr.org/`.
8. NIST. Workshop on elliptic curve cryptography standards, 2015. `http://www.nist.gov/itl/csd/ct/ecc-workshop.cfm`.