An Analysis of High-Performance Primes at High-Security Levels

Patrick Longa

Microsoft Research

In the position paper [4], it is stated that "*we have not observed any significant performance-based reason to go moving the security goalposts in the next generation of curves*". This statement is in response to the fact that several proposals use curves whose underlying prime fields have bitlengths that are different from the majority of prime field curves in previous standards. In this talk we will discuss ongoing work that aims to quantify how much performance one can gain by using handpicked primes that are not rigidly generated and that do not have traditional sizes, e.g. primes that are not 64-bit aligned. Our goal is to quantify the performance gap between rigidly generated primes and handpicked primes, and our hope is that this will provide NIST with a meaningful analysis that will aid their selection of prime fields, should new curves be chosen.

As the main case studies, we will consider the two NUMS curves targeting the 192 and 256-bit security levels, namely numsp384t1 and numsp512t1 (see [3] for details), and the two curves below that were chosen by allowing flexibility in the bitlengths of the primes. For all the curves, we wanted the primes to be chosen consistently (with the same shape) across the security levels and to offer high performance on a wide range of platforms.

- **Ted37919.** This curve is defined by the equation $E_{-1,d}/\mathbb{F}_p : -x^2 + y^2 = 1 + dx^2y^2$, where $p = 2^{379} - 19$ and $d = 143305$. The curve and its quadratic twist have cardinalities $\#E_{-1,d} = 8r$ and $\#E'_{-1,d} = 4r'$, where r and r' are 375- and 376-bit prime numbers, respectively. On average, Pollard's rho algorithm [7] requires around 2^{188} group operations to solve the ECDLP on Ted37919.
- **Ed48817.** This curve is defined by the equation $E_{1,d}/\mathbb{F}_p : x^2 + y^2 = 1 + dx^2y^2$, where $p = 2^{488} - 17$. The curve and its quadratic twist have cardinalities $\#E_{1,d} = 4r$ and $\#E'_{1,d} = 4r'$ (resp.), where r and r' are 486-bit prime numbers. On average, Pollard's rho algorithm requires around 2^{243} group operations to solve the ECDLP on Ed48817.

Curve generation and selection criteria. The curve parameters for Ted37919 and Ed48817 have been generated using the procedure in [4]. We use pseudo-Mersenne primes of the form $2^\alpha - \gamma$ to define the underlying fields. These primes achieve excellent performance that scales well with the bitlength (as the security level increases). Selecting primes (and then curves) of the same shape across security levels can give rise to multiple potential benefits such as code and algorithm sharing, and may in general ease the implementation and maintenance effort.

Given the two s-bit security levels with $s = 192$ and 256, the primes were selected by searching for a prime $p = 2^\alpha - \gamma$ using $\alpha = 2s, 2s-1, 2s-2, \ldots$ for a given s until $\log_2 \gamma \leq 5$ bits. We discarded primes such that $p \equiv 1 \pmod 8$ which are widely agreed to have unwieldy square root algorithms. The tight restriction on γ avoids that multiplications between limbs (in "non-canonical" form[1]) and γ spill over into an extra computer word. Doing a top-to-bottom search allows us to stay right below the standard security levels so as not to waste

[1] In this note, the term *canonical* refers to representations that use the minimum numbers of computer words required to represent field elements, and the term *non-canonical* refers to redundant representations that use extra words in order to reduce carry handling operations. Which type of representation is more efficient depends on the characteristics of a particular implementation, platform or application.

any bits in certain scenarios. For example, hardware implementations of Ed48817 can use multipliers with close-to-optimal size for targeting the 256-bit security level, which might introduce savings in area in contrast to fields defined over the extended prime $2^{521} - 1$. In another example, a 32-bit non-vectorized implementation of Ted37919 requires 12 limbs whereas curves like Curve41417 [1] requires 13 limbs, which has an impact on performance. We note, however, that the search outlined above would output the prime $p = 2^{489} - 21$ before the prime $p = 2^{488} - 17$. Our experiments indicate that the latter is favorable in most scenarios.

Taking into account relevant compatibility considerations, the selection of Ted37919 and Ed48817 roughly match standard security levels. The reduction in bit-security represents a trade-off between efficiency and security. This is in contrast to the criteria used by NUMS curves which maximizes security instead of changing the size of primes to attain an efficiency gain.

Implementation aspects. A point in favor of the two chosen finite fields is that they support efficient implementations using either *canonical* or *non-canonical* representations. This was carefully considered in the selection of these curves as preferring one style of representation over the other is strongly tied to the particular characteristics of a given platform or application. Below, we briefly detail relatively simple yet efficient alternatives for computing the field arithmetic.

– In the case of canonical representations, given two integers x and y such that $0 \leq x, y < 2^\alpha - \gamma$, one can compute $x \cdot y \bmod (2^\alpha - \gamma)$ by first computing the product and writing this in a radix-2^α system as $x \cdot y = z_h \cdot 2^\alpha + z_\ell$. A first reduction step, based on the shape of the modulus, is $z_h \cdot 2^\alpha + z_\ell \equiv z_\ell + z_h \cdot \gamma \pmod{2^\alpha - \gamma} = z$, where $0 \leq z < (\gamma + 1)2^\alpha$. If this step is repeated, the result is such that $0 \leq z < 2^\alpha + \gamma^2$, which can finally be brought into the desired range by applying an additional correction modulo p using subtractions. Performance can be enhanced further by using incomplete reduction: instead of reducing z to the range $[0, 2^\alpha - \gamma)$ after every multiplication, one can more efficiently let z stay in an extended range $[0, 2^\beta)$, where $\alpha < \beta < 2s$ (at a target security level of s bits), enabling the use of additions without modular corrections. When using the canonical representations, Ted37919 requires 48, 12 and 6 limbs for 8-, 32- and 64-bit platforms. Similarly, Ed48817 requires 61, 16 and 8 limbs for 8-, 32- and 64-bit platforms.
– In the case of non-canonical representations, one can represent field elements as $\{54|54|54|54|54|54|55\}$-bit limbs on 64-bit machines. Similarly, Ed48817 can make use of a representation with $\{54|54|54|54|54|54|54|55|55\}$-bit limbs. Both cases produce little overhead in carry handling and perform efficiently. On 32-bit platforms such as ARM with NEON, Ted37919 can use a representation with $\{27|27|27|27|27|27|27|27|27|27|27|27|27|28\}$-bit limbs and Ed48817 can use $\{27|27|27|27|27|27|27|28|27|27|27|27|27|27|27|28\}$-bit limbs. Interestingly, note that the latter representation is symmetric; this favors the use of 2-way Karatsuba. These representations for Ted37919 and Ed48817 require 7 and 9 limbs (resp.) for 64-bit platforms and 14 and 16 limbs (resp.) for 32-bit platforms.

Performance results: We use MSR ECCLib [6] to obtain performance results for the NUMS curves numsp384t1 and numsp512t1. We have implemented Ted37919 and Ed48817 on a 64-bit architecture using the non-canonical representation with $\{54|54|54|54|54|54|55\}$-bit limbs. Table 1 displays the results for variable-base scalar multiplication for Intel Sandy Bridge, Intel Haswell and AMD Steamroller processors in comparison with other relevant curves in the literature [5] (performance results for Ed448-Goldilocks were obtained by running eBACS' Supercop on the targeted platforms [2]).

As we can see, the curves chosen by allowing some "wiggle room" in the generation are somewhat more efficient. However, the performance of the NUMS curves is still very

Table 1. Clock cycles to compute scalar multiplication on 64-bit Intel processors.

curve	Est. bit security	Intel Sandy Bridge	Intel Haswell	AMD Steamroller
Ted37919	188	491,000	407,000	675,000
numsp384t1	191	611,000	504,000	717,000
Ed448-Goldilocks	223	667,000	532,000	990,000
Ed48817	243	1,091,000	916,000	1,319,000
numsp512t1	255	1,320,000	1,136,000	1,523,000

good, especially if one takes into account the additional security, rigidity and compatibility properties that are gained in return. Also, it should be taken into account that the benchmark comparison above is somewhat unfair against the NUMS curves because MSR ECCLib is a portable and generic library. Whenever suitable, it could be possible to improve NUMS' results by writing platform-specific implementations. The results in Table 1 also reveal that, for the studied platforms, there is no strong argument (from a performance perspective) to deviate from the widely-used standard security levels, namely 192 and 256-bit security levels.

References

1. D. Bernstein, C. Chuengsatiansup, and T. Lange. Curve41417: Karatsuba revisited. In Lejla Batina and Matthew Robshaw, editors, *Proceedings of CHES 2014*, volume 8731 of *LNCS*, pages 316–334. Springer, 2014.
2. Daniel J. Bernstein and Tanja Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to, accessed March 12th 2015.
3. Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Specification of Curve Selection and Supported Curve Parameters in MSR ECCLib. Tech Report no. MSR-TR-2014-92, 2014. http://research.microsoft.com/apps/pubs/default.aspx?id=219966.
4. Craig Costello, Patrick Longa, and Michael Naehrig. A brief discussion on selecting new elliptic curves. Tech Report no MSR-TR-2015-46, 2015. http://research.microsoft.com/apps/pubs/default.aspx?id=246915.
5. Mike Hamburg. Ed448-Goldilocks, a 448-bit Edwards curve, 2014. http://sourceforge.net/p/ed448goldilocks/code/ci/master/tree/.
6. Microsoft Research. MSR Elliptic Curve Cryptography Library (MSR ECCLib), 2014. Available at: http://research.microsoft.com/en-us/projects/nums.
7. J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.