Slothful reduction

Michael Scott

MIRACL.com
mike.scott@miracl.com

Abstract. In the implementation of many public key schemes, there is a need to perform modular arithmetic. Typically this consists of addition, subtraction, multiplication and (occasionally) division with respect to a prime modulus. To resist certain side-channel attacks it helps if implementations are "constant time". As the calculations proceed there is potentially a need to reduce the result of an operation to its remainder modulo the prime modulus. However often this reduction can be delayed, a process known as "lazy reduction". The idea is that results do not have to be fully reduced at each step, that full reduction takes place only occasionally, hence providing a performance benefit. Here we extend the idea to determine the circumstances under which reduction can be delayed to the very end of a particular public key operation.

1 A basic observation

Consider a typical modular squaring operation. We require the square of $x \in \mathbb{F}_p$, that is $x \leftarrow x^2 \bmod p$. Assume that x is such that $x^2 < p.R$, where R is some value greater than p. Assume that the modular reduction algorithm used is such that after completion $x < 2.p$. In fact this is exactly how the well known Montgomery reduction algorithm works [6] (unless a conditional final subtraction is performed).

Now if x is not fully reduced prior to the squaring, say if we can only be sure that $x < 8.p$, then as long as $R > 64.p$, the output x is such that $x < 2.p$. Therefore we contend that a calculation that consists of a combination of modular additions, subtractions and multiplications may never need explicit reduction, due to the tendency of the squaring/multiplication code to force an implicit (albeit incomplete) reduction.

2 Finite field element representation

It is commonly assumed that a "packed-radix" representation is used for field elements, where numbers are represented to a base equivalent to the word-length of the computer. So for example a 256-bit number would be represented as eight 32-bit digits, or four 64-bit digits. But clearly to support any kind of lazy reduction some kind of redundant representation must be used which allows elements to increase beyond the bit length of the modulus. Indeed it is now

common for moduli to be proposed which are a few bits shy of the maximum possible for a fixed length representation in computer words. For a justification in the context of elliptic curve cryptography see [3]. For example rather than a 256-bit modulus, a 254-bit modulus may be preferred for the two bits of redundancy it allows. A disadvantage is the small loss of security that arises from using a smaller modulus.

The required redundancy arises more naturally if a "reduced radix" representation is used. Here elements are represented to a base somewhat less than the full word-length. So for example on a 64-bit architecture a base of 2^{60} might be used. This brings its own advantages as carry propagation can be delayed as well, a process which might be referred to as "lazy carrying", not to be confused with lazy reduction. Also a faster method for modular multiplication applies [8].

Our point is that using a reduced radix representation, common modulus sizes will no longer have a tight fit into a fixed number of computer words. So for example an element in $\mathbb{F}_p$, where p is 256 bits, may be represented as four 60-bit least-significant digits, and a most significant digit of only 16-bits, leaving a large amount of redundancy in the top word.

Note that our idea applies to any representation with sufficient redundancy (but as we will see more than two bits will be required).

We refer to the extent that an element x has crept beyond its modulus as its "excess" the integer E, where $x < E_x.p$.

The next issue is to fix the value of R. For multi-precision Montgomery reduction the ideal choice is 2^n where $p < 2^n$, and n is a multiple of the base of the representation. So for our 64-bit reduced-radix example above, a natural choice would be 2^{300}, where $300 = 5.60$. For the packed-radix case the natural choice would be 2^{256}

If Montgomery modular reduction is to work correctly then it is clearly sufficient when multiplying x by y that $E_x.E_y.p < R$. Observe that numbers less than R are representable using the same number of words as elements in $\mathbb{F}_p$.

2.1 Special moduli

For elliptic curve cryptography often a prime of special form is used, for which a faster reduction method applies. A common choice is a pseudo-Mersenne prime of the form $2^m - c$. The product to be reduced is split into a lower and higher part, with the lower part of length m bits. The top part is multiplied by c and added to the lower part. Finally the small excess beyond m bits is extracted, multiplied by c and added to the total. It is easy to see that the output may not be fully reduced, but will certainly be less than $2.p$.

Now when the product is split, as long as it is less than $p.R$, then the top part will be less than R, and hence representable. So basically the same rule applies as for Montgomery's method: As long as the input is less than $p.R$, then the reduced output will be less that $2.p$.

3 Modular arithmetic

Multiplication has already been discussed. Addition is carried out without reduction. Therefore if calculating $z = x + y$, then (assuming the worst case) $E_z = E_x + E_y$.

Subtraction is negation, followed by addition. To find $-y$ we subtract y from a suitable multiple $r = E_y$ of p and then $-y = r.p - y \bmod p$. If calculating $z = x - y$, then again $E_z = E_x + E_y$. Note that suitable multiples of the modulus may be precalculated and stored. Obviously the fewer the better.

4 Edwards curves

As an example we will consider an implemention of point multiplication on the Edwards curve [2],

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

using some kind of double-and-add algorithm. Using projective coordinates the calculations for point doubling and point addition are a mixture of modular additions, subtractions and multiplications. We want to determine the circumstances under which explicit reduction can be delayed until the very end of the point multiplication (assuming that a fully reduced output is desired).

From the explicit formula database [5] we find formulae for point doubling of an input point in projective coordinates (X, Y, Z). A common trick used when deriving these formulae is to use the identity $2XY = (X+Y)^2 - X^2 - Y^2$, which swaps a multiplication for a squaring. However if using a reduced-radix representation, multiplications and squarings have essentially the same complexity [8]. So we will not use this trick here, as it has the unfortunate side-effect of artifically inflating the excesses.

The formula is broken down into atomic operations, and simplified for our purposes. See Table 1. On each line the excess of the output value is recorded. We can assume that the curve constant d is fully reduced. For reasons that will become clear we assume that the initial excesses of X, Y and Z are all equal to 2.

Some observations: This operation is stable, in that the excesses of the outputs are the same as those of the inputs. Therefore multiple doubling operations can follow one another without affecting the worst case excesses recorded here. In the Notes column of the table, for subtractions we note the multiple of the modulus required when negating. Here $4p$ will work in all cases. After each multiplication or squaring we note the product of the excesses of the inputs. The maximum value for M is significant here – it will determine the minimum value for R.

Next is the point addition formula (Table 2). Again the overall operation is stable, and again we see that any combination of additions and doublings will not change these values. In this case a precomputed $2p$ is sufficient for all negations.

Operation	Excess	Note
$A = X$	2	
$B = Y$	2	
$C = Z$	2	
$D = A^2$	2	$M = 4$
$B = B^2$	2	$M = 4$
$C = C^2$	2	$M = 4$
$C = C + C$	4	
$D = B + D$	4	
$B = B + B$	4	
$B = D - B$	8	$r = 4$
$A = X.Y$	2	$M = 4$
$A = A + A$	4	
$C = D - C$	8	$r = 4$
$X = A.C$	2	$M = 32$
$Y = B.D$	2	$M = 32$
$Z = C.D$	2	$M = 32$

Table 1. Edwards Point doubling

Operation	Excess	Note
$A = X_1$	2	
$B = Y_1$	2	
$C = Z_1$	2	
$D = X_2$	2	
$E = Y_2$	2	
$F = Z_2$	2	
$C = C.F$	2	$M = 4$
$G = A + B$	4	
$H = D + E$	4	
$A = A.D$	2	$M = 4$
$B = B.E$	2	$M = 4$
$G = G.H$	2	$M = 16$
$G = G - A$	4	$r = 2$
$G = G - B$	6	$r = 2$
$G = G.C$	2	$M = 12$
$H = A.B$	2	$M = 4$
$H = d.H$	2	$M = 2$
$B = B - A$	4	$r = 2$
$B = B.C$	2	$M = 8$
$C = C^2$	2	$M = 4$
$A = C - H$	4	$r = 2$
$C = C + H$	4	
$X_3 = A.G$	2	$M = 8$
$Y_3 = B.C$	2	$M = 8$
$Z_3 = C.A$	2	$M = 16$

Table 2. Edwards Point addition

Overall the maximum value for M is 32. Now if $R = 2^n$ and $p < 2^m$, then we can conclude than as long as $n - m$ is greater than or equal to 5 bits, then these functions when combined to calculate a point multiplication, will work correctly. For a packed-radix representation, a 251-bit modulus is therefore a better choice than 256-bits or 254 bits. For a reduced-radix representation it is not difficult to meet this constraint for any size of modulus.

It might be regarded as inconvenient to have to precompute both $2p$ and $4p$. In fact $4p$ can be used for all the negations that arise in the point addition formula without increasing the maximum value of M.

To be concrete, to implement any Edwards curve modulo a 256-bit prime on a 32-bit processor, choose a base of 2^{29} in which case nine digits are required to represent field elements. Setting $R = 2^{9.29} = 2^{261}$ our conditions are met, as $32 = 2^{261-256}$. In all cases modular subtraction of $x - y$ can be calculated as $4p - y + x$. Since explicit reduction is never required, reduction is not merely lazy, it is slothful. Conditional subtractions by the modulus are never required, and therefore the code can easily be made constant time, independent of the data being processed.

5 Weierstrass curves

For a more complex example we consider the exception-free formulae for arithmetic on a prime-order Weierstrass curve

$$y^2 = x^3 - 3x + b$$

as recently described by Renes, Costello and Batina [7]. We start with point doubling (Table 3). In this case the output values are not the result of modular multiplications, and so we cannot assume that the calculation is necessarily stable. However for an input point (X, Y, Z) with input excesses of 4, 4 and 4 respectively, the outputs have the same excesses (or less), and therefore we conclude that this function is stable.

For point addition (Table 4) on the same curve, we assume the same input excesses for (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) of 4, 4 and 4 respectively, as the inputs may be result of previous doubling operations.

In this case the maximum $M = 676$ and so R would need to be 10 bits greater than the prime modulus to permit slothful reduction. Using a reduced radix representation this is an easy condition to meet on a 64-bit processor. On a 32-bit computer 256-bit field elements could be represented as ten digits to a base of 2^{27}, which allows $R = 2^{270}$ with a comfortable safety margin. There is a small cost here given that the field element representation has increased from nine to ten digits. But this may be regarded as a worthwhile trade-off for a constant time implementation.

For this example more multiples of the modulus are required to support modular negation in all cases. Again with care some r multiples can be adjusted upwards to coincide with others, without increasing the maximum value of M, so that less multiples may be needed. We leave this as an exercise for the reader.

Operation	Excess	Note	Operation	Excess	Note
1. $t_0 = X^2$	2	$M = 16$	2. $t_1 = Y^2$	2	$M = 16$
3. $t_2 = Z^2$	2	$M = 16$	4. $t_3 = X.Y$	2	$M = 16$
5. $t_3 = t_3 + t_3$	4		6. $Z_3 = X.Z$	2	$M = 16$
7. $Z_3 = Z_3 + Z_3$	4		8. $Y_3 = b.t_2$	2	$M = 2$
9. $Y_3 = Y_3 - Z_3$	6	$r = 4$	10. $X_3 = Y_3 + Y_3$	12	
11. $Y_3 = X_3 + Y_3$	18		12. $X_3 = t_1 - Y_3$	20	$r = 18$
13. $Y_3 = t_1 + Y_3$	20		14. $Y_3 = X_3.Y_3$	2	$M = 400$
15. $X_3 = X_3.t_3$	2	$M = 80$	16. $t_3 = t_2 + t_2$	4	
17. $t_2 = t_2 + t_3$	6		18. $Z_3 = b.Z_3$	2	$M = 4$
19. $Z_3 = Z_3 - t_2$	8	$r = 6$	20. $Z_3 = Z_3 - t_0$	10	$r = 2$
21. $t_3 = Z_3 + Z_3$	20		22. $Z_3 = Z_3 + t_3$	30	
23. $t_3 = t_0 + t_0$	4		24. $t_0 = t_3 + t_0$	6	
25. $t_0 = t_0 - t_2$	12	$r = 6$	26. $t_0 = t_0.Z_3$	2	$M = 360$
27. $Y_3 = Y_3 + t_0$	4		28. $t_0 = Y.Z$	2	$M = 16$
29. $t_0 = t_0 + t_0$	4		30. $Z_3 = t_0.Z_3$	2	$M = 120$
31. $X_3 = X_3 - Z_3$	4	$r = 2$	32. $t_0 = t_0 + t_0$	8	
33. $t_1 = t_1 + t_1$	4		34. $Z_3 = t_0.t_1$	2	$M = 32$

Table 3. Weierstrass Point Doubling

Operation	Excess	Note	Operation	Excess	Note
1. $t_0 = X_1X_2$	2	$M = 16$	2. $t_1 = Y_1Y_2$	2	$M = 16$
3. $t_2 = Z_1Z_2$	2	$M = 16$	4. $t_3 = X_1 + Y_1$	8	
5. $t_4 = X_2 + Y_2$	8		6. $t_3 = t_3t_4$	2	$M = 64$
7. $t_4 = t_0 + t_1$	4		8. $t_3 = t_3 - t_4$	6	$r = 4$
9. $t_4 = Y_1 + Z_1$	8		10. $X_3 = Y_2 + Z_2$	8	
11. $t_4 = t_4X_3$	2	$M = 64$	12. $X_3 = t_1 + t_2$	4	
13. $t_4 = t_4 - X_3$	6	$r = 4$	14. $X_3 = X_1 + Z_1$	8	
15. $Y_3 = X_2 + Z_2$	8		16. $X_3 = X_3Y_3$	2	$M = 64$
17. $Y_3 = t_0 + t_2$	4		18. $Y_3 = X_3 - Y_3$	6	$r = 4$
19. $Z_3 = bt_2$	2	$M = 2$	20. $X_3 = Y_3 - Z_3$	8	$r = 2$
21. $Z_3 = X_3 + X_3$	16		22. $X_3 = X_3 + Z_3$	24	
23. $Z_3 = t_1 - X_3$	26	$r = 24$	24. $X_3 = t_1 + X_3$	26	
25. $Y_3 = bY_3$	2	$M = 6$	26. $t_1 = t_2 + t_2$	4	
27. $t_2 = t_1 + t_2$	6		28. $Y_3 = Y_3 - t_2$	8	$r = 6$
29. $Y_3 = Y_3 - t_0$	10	$r = 2$	30. $t_1 = Y_3 + Y_3$	20	
31. $Y_3 = t_1 + Y_3$	30		32. $t_1 = t_0 + t_0$	4	
33. $t_0 = t_1 + t_0$	6		34. $t_0 = t_0 - t_2$	12	$r = 6$
35. $t_1 = t_4Y_3$	2	$M = 180$	36. $t_2 = t_0Y_3$	2	$M = 360$
37. $Y_3 = X_3Z_3$	2	$M = 676$	38. $Y_3 = Y_3 + t_2$	4	
39. $X_3 = t_3X_3$	2	$M = 156$	40. $X_3 = X_3 - t_1$	4	$r = 2$
41. $Z_3 = t_4Z_3$	2	$M = 156$	42. $t_1 = t_3t_0$	2	$M = 72$
43. $Z_3 = Z_3 + t_1$	4				

Table 4. Weierstrass Point addition

6 Extension field arithmetic

Slothful reduction may also find application in extension field arithmetic, as required in pairing-based cryptography. Consider for example powering in the extension field $\mathbb{F}_{p^2}$ using some square-and-multiply algorithm. Assuming p is congruent to 3 mod 4, then -1 will always be a quadratic non-residue, and elements can be represented as $u + iv$, where $u, v \in F_p$, and $i = \sqrt{-1}$.

To find the square of an element, we calculate $u \leftarrow (u+v)(u-v)$ and $v \leftarrow 2uv$.

Operation	Excess	Note
$A = u$	4	
$B = v$	6	
$C = A.B$	2	$M = 24$
$D = A + B$	10	
$E = A - B$	10	$r = 6$
$u = D.E$	2	$M = 100$
$v = C + C$	4	

Table 5. Squaring in $\mathbb{F}_{p^2}$

To multiply two elements $u_1 + iv_1$ and $u_2 + iv_2$ a Karatsuba method applies, and $u_3 = u_1.u_2 - v_1.v_2$ and $v_3 \leftarrow (u_1 + v_1)(u_2 + v_2) - (u_1.u_2 + v_1.v_2)$.

First we take the same approach as above. To achieve stability we must it seems assume initial excesses for u and v of 4 and 6 respectively. See Tables 5 and 6.

Operation	Excess	Note
$A = u_1.u_2$	2	$M = 16$
$B = v_1.v_2$	2	$M = 36$
$C = u_1 + v_1$	10	
$D = u_2 + v_2$	10	
$E = C.D$	2	$M = 100$
$F = A + B$	4	
$u_3 = A - B$	4	$r = 2$
$v_3 = E - F$	6	$r = 4$

Table 6. Multiplication in $\mathbb{F}_{p^2}$

But this may not be the best approach. Aranha et al. [1] have described in detail a potentially faster way in which to do multiplication in $\mathbb{F}_{p^2}$, which involves the use of lazy reduction in a different sense to which we have used it here. The idea is that if required to calculate say $ab + cd \bmod p$, then it makes sense to delay the reduction until after the addition, so that only one reduction is required rather than two. This is particularly true if reduction following multiplication is

slow, as would be the case for a p of no exploitable special form, as for example arises in pairing-based cryptography.

First we re-describe squaring with smaller initial excesses for u and v of 2 and 2 respectively, but for which the Aranha et al. idea does not apply. See (Table 7). We also re-order the instructions to our advantage.

Operation	Excess	Note
$A = u$	2	
$B = v$	2	
$C = A + A$	4	
$C = C.B$	2	$M = 8$
$D = A + B$	4	
$E = A - B$	4	$r = 2$
$u = D.E$	2	$M = 16$
$v = C$	2	

Table 7. Squaring in $\mathbb{F}_{p^2}$

Next we apply our methodology to the Aranha et al. method for multiplication (Table 8).

Operation	Excess	Note
1. $\mathbf{A} = u_1.u_2$	4	
2. $\mathbf{B} = v_1.v_2$	4	
3. $C = u_1 + v_1$	4	
4. $D = u_2 + v_2$	4	
5. $\mathbf{E} = C.D$	16	
6. $\mathbf{F} = \mathbf{A} + \mathbf{B}$	8	
7. $\mathbf{A} = \mathbf{A} - \mathbf{B}$	8	$r = 4$
8. $\mathbf{E} = \mathbf{E} - \mathbf{F}$	16	
9. $u_3 = \mathbf{A}$	2	$M = 8$
10. $v_3 = \mathbf{E}$	2	$M = 16$

Table 8. Faster Multiplication in $\mathbb{F}_{p^2}$

Here double precision numbers are represented in bold-face. The product of two values x and y with excesses of E_x and E_y respectively results in a double precision product $\mathbf{z} = x.y$, where $z < E_x.E_y.p^2$, and so the double precision excess is $E_z = E_x.E_y$.

Modular subtraction in line 7 is calculated as $A - B = 4p^2 - B + A$ (and so $4p^2$ should be precalculated), but the modular subtraction in line 8 clearly requires only simple non-modular subtraction (as $\mathbf{E} > \mathbf{F}$). Note that the maximum M value is now reduced to 16. The delayed double-precision reduction takes place in lines 9 and 10.

7 Unstable calculations

Unfortunately not all algorithms are stable in the sense used here. Consider for example the application of our original methodology to the fast algorithm for squaring in the cyclotomic subgroup of sixth degree extensions, as proposed by Granger and Scott [4]

$$a \leftarrow 3a^2 - 2\bar{a}$$
$$b \leftarrow 3\sqrt{i}.c^2 - 2\bar{b}$$
$$c \leftarrow 3b^2 - 2\bar{c}$$

Consider now a sequence of squarings of some initial value. Observe that $2\bar{a}$ contributes to the new value of a after each squaring. So clearly after each sequential squaring, the excess of a increases monotonically. Therefore in this case explicit reduction will be required, and fully slothful reduction will not be possible.

8 Conclusion

We have described a simple cook-book methodology that could be used to assist a programmer in implementing certain cryptographic functions efficiently, and in constant time. However we are not claiming that this is the only way to achieve this outcome. Basically we tailor the representation of finite field elements to fit the needs of the cryptographic algorithm, rather than force an algorithm unto a predetermined form of representation.

The method applies also to other moduli types, in particular to generalised Mersenne primes. The bound on the output of a product, that it be less than $2p$, can be significantly tightened in particular cases. For example for Montgomery reduction if $E_x.E_y.p$ is much less than R, then $2p$ can be reduced to $\delta.p$ for some $\delta < 2$.

References

1. D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. Lopez. Faster explicit formulae for computing pairings over ordinary curves. In *Eurocrypt – 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer-Verlag, 2011.
2. 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.
3. J. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, 6(4):259–296, 2016.
4. R. Granger and M. Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In *PKC – 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 2010.
5. T. Lange. Explicit formula database. `http://hyperelliptic.org/EFD/`.
6. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
7. J. Renes, C. Costello, and L. Batina. Complete addition formulas for prime order elliptic curves. In *Eurocrypt – 2016*, volume 9665 of *Lecture Notes in Computer Science*, pages 403–428. Springer-Verlag, 2016.
8. M. Scott. Missing a trick: Karatsuba variations. *Cryptography and Communications*, 2017. `https://link.springer.com/article/10.1007%2Fs12095-017-0217-x`.