Alternative cubics' rules with an algebraic appeal

Daniel R. L. Brown[*]

Rough draft, June 26, 2015

Abstract

Two alternating vector operations on a cubic hypersurface are given simple expressions. Direct use of the first operation's expression seems less efficient than state-of-the-art elliptic curve cryptography. The second expression seems mainly interesting towards an elementary exposition about elliptic curve theory.

1 Cubic-secant intersection: a vector expression

A cubic homogeneous polynomial (cubic form) e acts on vector x by:

$$e(x) = e(x_0, x_1, \dots) = \sum_{0 \le i \le j \le k} e_{i,j,k} x_i x_j x_k.$$

Fix e. For any vectors x and y, define:[1]

$$x \mathbin{\$} y = e(x-y)(x+y) + e(x+y)(x-y), \tag{1}$$

except in characteristic two, where one first divides by two, as detailed in §2.2. Say x is *on* (hypersurface) e, and write $x \in e$, if $e(x) = 0$ and $x \neq (0, 0, \dots)$. Properties of binary operation \$ include:

Proposition 1. *If $e(x) = e(y) = 0$, then $e(x \mathbin{\$} y) = 0$.*

Proposition 2. *For field F, if $x, y, z \in e$, and $z \in (Fx + Fy) \setminus (Fx \cup Fy)$, then $x \mathbin{\$} y \in Fz$.*

Proposition 3. *If $\dim(Fx + Fy) = 2$ and $x, y \in e$ and $x \mathbin{\$} y = (0, 0, \dots)$, then $Fx + Fy \subset e$.*

[*]dbrown@certicom.com

[1]Using the usual vector operations: $(x+y)_i = x_i + y_i$ and $(ax)_i = ax_i$.

Relation to chord-and-tangent addition rule These three propositions essentially show that if the vector $x \$ y$ is a nonzero vector, then it represents the third projective point on intersection of the hypersurface e with the projective line $Fx + Fy$ through x and y (the secant line).

In particular, when restricted to a three dimensional subspace of vectors, where e defines a plane cubic curve, which can often yield an elliptic curve, the operation \$ corresponds to an intermediate step in the well-known geometric chord-and-tangent rule for the addition law on elliptic curves.

So, the vector operation \$ corresponds a well-known projective point operation. Indeed, Niven, Zuckerman, and Montgomery [NZM91] attribute the first use of a chord (secant) to find a new point on a cubic to a manuscript of Newton.

Of all the vector operations corresponding to this geometric point operation, the operation \$ might have simplest expression (1) at least among expression permitted to apply e as a function.

Absence of the expression in some earlier works Most introductory texts on elliptic curves provide Jacobi's geometric description of the elliptic curve group law and usually an explicit computational description in affine coordinates. Most cryptographic implementations use projective coordinates, and therefore explicit computational descriptions of elliptic curve operations are needed. For example, Bosma and Lenstra [BL95] provide explicit group laws for elliptic curves given by Weierstrass equations. They expand the polynomials fully as a sum of monomials times constants, which may be satisfactory for a computational standpoint.

More recent work has sought more efficient explicit projective coordinates description of elliptic curve group operations. Bernstein and Lange [BL15] maintain an excellent database of such operations, with state-of-the-art efficiency and security. These formula generally assume a special form of curve, but one can often transform an arbitrary curve into one of these special forms, so they are usually general enough for cryptography.

This operation \$, and its expression in (1) continues in the trend of elementary projective formulas, though it emphasizes brevity and generality rather than efficiency. As such, \$ does not immediately merit inclusion in [BL15] due to it lacking any demonstration of efficiency and due to its intermediacy with respect to the usual basic elliptic curve operation.

Illustration of the operation Figure 1 illustrates an example with four-dimensional vectors projected onto the two-dimensional plane. The projection through the zero vector maps the four-space to three-space, which is then projected onto the plane. The figure aims to convey the sense of the projective three-space by showing depth: portions of geometric objects are obscured by others in front of them. A subset of vectors on e is drawn as a strip bent into the shape of the letter S. Three vectors on e, namely x, y, and $x \mathbin{\$} y$ are shown, which appear collinear, since they belong to the same two-dimensional space $Fx + Fy$, which is drawn as four collinear line segments: with the three gaps being obscured by a strip cut out of the cubic surface. The vectors $x + y$ and $x - y$ are also drawn, and belong to the same two-dimensional space $Fx + Fy$. Obviously, the figure in no way whatsoever proves that our formula for $x \mathbin{\$} y$ actually belongs to the curve e.

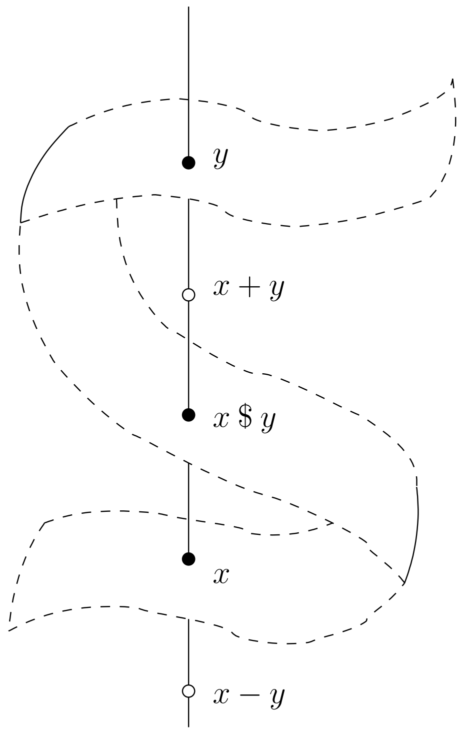

Figure 1: The operation \$ seen on a strip of a projective cubic surface

Figure 1 includes some liberties. The positioning of the five points in the figure on the line $Fx + Fy$ in merely schematic, and might not be consistent with any actual projection. Also, the rims of the cubic strip, shown in solid, are interpolated curves drawn with xfig program, and, as such, are probably something akin to spliced together Bezier curves, with might not accurately represent how any cubic strip would look.

The symbol \$ that this report uses for this operation is a miniature version of Figure 1, and may thus have mnemonic value. The two historical facts that Newton worked at the mint and that Newton used the geometric operation which Figure 1 and \$ represent, may also help in remembering,

Naming the operation and expression Presumably, the actual vector operation \$ and its expression in (1) are already known and named, given the extensive body of research into elliptic curves. Accordingly, they should be named per such precedents.

Occasionally, names for formula ease spoken discussion. Temporarily, until determining the preceding names, some interim short names can be used. The operation \$ can temporarily be called the *bisectant* operation. The term bisectant merges the terms intersect, secant and bisect. The term bisect is motivated by the vector operation $(x + y)/2$, which is projectively equivalent to $x + y$, which appears in the expression (1). Because many different expressions are possible for the same operation, the expression in (1) may also need naming. Temporarily call the right hand side of (1) the *primeval* or *simplex* or *exemplary* expression for the bisectant operation.

1.1 Inefficiency of expression (1)

This section looks at the efficiency of directly using expression (1) for computing the the operation \$. The conclusion is that it is almost certainly considerably less efficient than the state-of-the-art elliptic curve operations.

For example, Hisil, Wong, Carter and Dawson [HWCD08] provide an addition rule for twisted Edwards curves that uses eight field multiplications, in a sequential (single thread of) computation, and in two field multiplications using four threads of parallel computation.

By contrast, the most efficient application of (1) described below, requires a special form of cubic e and uses eight (resp. one) field cubing(s) and eight (resp. one) field multiplication(s) in one (resp. eight) thread(s) of parallel computation. Essentially, this consumes more than twice the computational resources of the HWCD addition law.

The resulting estimates are so far from competing with methods like Hisil–Wong–Carter–Dawson that it seems not worth trying to make an empirical measurement by actually implementing \$ according to (1).

1.1.1 Chord rule cost comparison

Recall from the traditional formulas for elliptic curves in short Weierstrass form that, in an elliptic curve addition of distinct points, most of the computation corresponds to the step of intersecting the secant with the cubic, which is known as the chord rule. The remaining computation is, geometrically, just a reflection in the horizontal axis, which corresponds to a very simple linear vector operation.

If the same pattern holds in the general setting, then we can examine the cost of the intersecting the secant and the cubic using \$ as described (1), keeping in mind that a small extra amount of work may be added later, corresponding to the linear reflection operation used in the Weierstrass standard equation.

1.1.2 Non-parallel implementation: too expensive

Consider first the cost of \$ in a system that has no parallel operations. Then it seems that the two scalar-vector multiplications in (1) already cost $2d$ field multiplications where d is the dimension of the vector space. The vector space dimension must be at least 3. So this is six field multiplications, which is already nearly the eight used by [HWCD08], and yet we have not even accounted for the cost of evaluating the cubic e twice.

These six or more field multiplications must be evaluated after two evaluations of e. With three dimensional vectors, most useful cubics e have at least four monomial terms. Naively, such an e would be evaluated in at least eight field multiplications, two per term, with perhaps also further field multiplications by a constant. Since e must be evaluated twice, that gives at least sixteen field multiplications. Adding the previous six, gives 22 field multiplications.

Using a Legendre form for an elliptic curve, such as:

$$e(x) = x_1^2 x_2 - x_0(x_0 - x_2)(x_0 - \lambda x_2)$$

for some λ, can allow e to be evaluated with three general field multiplications, one squaring, and one field multiplication by a constant ($3M + 1S + 1K$). Doing this twice, plus the previous six field multiplications, gives $12M + 2S + 2K$, which is still far worse than [HWCD08].

Note that using a Legendre equation here imposes the condition that the elliptic curve groups have a subgroup of order four.

One can also work in extended coordinates using vectors of dimension four instead of dimension three. In this case, the Legendre equation can be written as $e(x) = x_1^2x_2 - x_0(x - x_2)x_3$. In the three-dimensional subspace V defined by $x_3 = x_0 - \lambda x_2$, the effect is identical to the previous Legendre equation. This approach replaces the two field multiplication by λ in the two evaluations of e, by two general field multiplications to handle the fourth coordinate in the scalar-vector multiplications in the expression (1). So, this case is worse than the previous.

Trying extended coordinates again, one can instead consider a Fermat cubic:

$$e(x) = x_0^3 + x_1^3 + x_2^3 + x_3^3.$$

The main hope here is that the using four field cubings ($4C$) above to evaluate e are more efficient than three field multiplications and a squaring ($3M + 1S$) in the Legendre equation case. That might not generally be the case (though, later we will re-consider Fermat cubics in parallel computation setting).

Note the use of Fermat cubic surface may impose some conditions on the elliptic curves that intersect the surface.

Regardless, all the approaches above use more field multiplications than even the old-fashioned standard approaches to point addition for short Weierstrass equations. In other words, the only hope for an efficiency advantage from (1) is in the case of parallel implementations.

1.1.3 Parallel implementation

In this section, we will consider an approach to implementing \$ using 8 threads of parallel computation and the Fermat cubic form.

Simplistic view of parallel computation This report takes a simplistic view of parallel computation. Many different types of parallel computation are available on modern devices, such as pipelines, superscalaring, hyperthreading, single-instruction-multiple-data (SIMD) CPU instructions, multiple CPU cores, and graphics processing units. This report discusses parallel computation in terms of SIMD operations. In processors without explicit SIMD instructions, some of the gains from SIMD operations might also be achievable with pipelining or superscaling.

Generally, we consider the number of parallel threads of computation as an important cost. Suppose one is comparing two algorithms, and that they have essentially identical computation cost, except that one that uses more threads of parallel computation. The one using fewer threads is preferred for two reasons. Some systems many not have enough parallel capability to support the larger number of threads. In other systems, the algorithm using fewer threads frees up computational resources that can be used for other simultaneous computations.

Expanded addition laws Bosma and Lenstra [BL95] find addition laws for any elliptic curve that have bi-degree $(2, 2)$. If we allow an unlimited number of parallel threads of computation, then each monomial term in the addition can be computed in a parallel thread using two parallel field multiplication plus one field multiplication by a constant. The main disadvantages of this approach is the high number of threads required, and perhaps the one-time difficulty for the programmer to make sure all the terms are entered correctly.

Field implementation Consider a finite field implementation with the following properties. Each field elements has a set of representations, as a sequence of computer words, where a word is some kind of data type on which the computer naturally acts, such as an unsigned integer, or a floating point number. The field representatives can be added, subtracted and multiplied using a pre-determined set of computer operations on these words. In particular, the set of computer operations does not depend on the values of the field elements. Call this a constant operation implementation.

Such field implementations are now well-known. Because the operations are constant, parallel computation is potentially useful. If multiple similar field operations are done in parallel, then they complete at the same time, which avoids branching and stalling.

SIMD word operations Many modern devices' central processing units (CPU) have an instruction set that include Single Instruction Multiple Data (SIMD) operations. The SIMD instructions perform vector operations at the level of computer words: one vector of words can be added to another vector of words. Sometimes SIMD instructions allow word-by-word multiplication of vectors. Some SIMD instruction sets allow dot product of two vectors of words to computed.

SIMD field operations Given a constant operation implementation of a field operation, such as field addition, one can try to convert this into the corresponding vector operation, such as vector addition, by replacing each word operation by the corresponding SIMD word operation.

For example in the expression (1), computing the vector sum $x+y$ might be done with a sequence of SIMD word operations, whereby the addition is done simultaneously.

Fermat cubic surfaces The expression (1) for \$ describes two evaluations of e. Therefore, if we are free to choose the form of e, we may want to consider a form that is most efficient, and following the approach above, best suited to a SIMD-based vector implementation.

To this end, consider a Fermat cubic surface, such as $e(x) = x_0^3 + x_1^3 + x_2^3 + x_3^3$. An evaluation of e can be done with SIMD-version of the field cubing operation: the field cubes x_i^3 being computed simultaneously. A dedicated field cubing operation is potentially faster than two general field multiplications.

The evaluation of \$ as described above seems to require 8 threads of parallel computation. In each thread, there is one field cubing, one field multiplication, as well as some field additions and subtractions. Summarize this cost as $(1M + 1C)_8$.

The Hisil–Wong–Carter–Dawson (HWCD) [HWCD08] addition rule used four threads of parallel computation, with each thread using two field multiplications and some field additions and subtractions. Summarize this cost as $(2M)_4$.

Because a field cubing is expected to be considerably slower than a field multiplication, the approach above should be considerably slower than than HWCD addition rule, even when it uses twice as many parallel threads of computation as HWCD does.

1.2 Diffie–Hellman using \$

This section considers using the operation \$ to implement elliptic curve Diffie–Hellman (ECDH). More precisely, one must also specify a three-dimensional subspace V of vectors.

As shown above, there seems to be no efficiency benefits to implementing ECDH this way. The main benefit would be therefore be some kind of simplicity.

Oddly, it does not seem necessary to specify the identity element. Consequently, the implementation ECDH is defined over up to nine possible group structures on the plane cubic curve.

A severe *caution* to such an implementer is in order. The simplicity of \$ in no way buys any protection from the existing attacks and various risk of implementation faults. All the usual protections for ECDH must be considered.

1.2.1 Review of traditional EC groups

This subsection reviews some basics of the traditional theory of elliptic curve groups and how it relates to structure given by \$. This relation is what allows one to show that how \$ can be used to implement ECDH.

Comparing \$ to the usual EC group law The usual approach to elliptic curve cryptography (ECC), especially elliptic curve Diffie–Hellman (ECDH), uses the group structure on the elliptic curve. The cubic e alone is enough to define the operation \$ but it is not enough to pin down an elliptic curve group structure. To define a group, one also needs:

- a three-dimensional subspace V of vectors, thereby narrowing the cubic hypersurface to a (projective) plane cubic curve $e \cap V$, and
- a point $o \in e \cap V$ selected as the identity element of the group.

With this additional information, the well-known theory of elliptic curves define a group operation $\oplus_{o,V}$ on the projective points represented by the vectors in $e \cap V$, such that o represents the identity element of the group.

Fix o and V, and abbreviate to the group operation to $\oplus$, let $\ominus$ indicate the associated negation operation. We henceforth assume that all the vectors encountered belong to the three-dimensional subspace V, and refer to e as a curve, though we really mean $e \cap V$.

In some cases, V will be define explicitly by a system of linear equations, and in other cases, it may be defined implicitly by three vectors $u, w, z \in e \cap V$ that also form a basis of V.

Inflective identity point Conventional approaches to elliptic curve groups select an additive identity point o that is an inflection point. For example, in the traditional Weierstrass form of an elliptic curve, the identity point is selected as the point[2] $(0 : 1 : 0)$ represented by vector $(0, 1, 0)$. In affine coordinates, this is a point at infinity.

Furthermore, part of the generality of the Weierstrass form of an elliptic curve stems from the following well-known transformation of an arbitrary

[2]Thanks to Samuel Neves for noticing my mistake in the previous draft which had $(0 : 0 : 1)$ here.

plane cubic curve into Weierstrass form [ST92, I.3]. First determine an inflection point o on the cubic curve. Then apply a bi-rational transformation that maps o to $(0, 1, 0)$, with some other properties (such as the line at infinity being tangent to the transformed curve).

An efficiency, or at least simplicity, advantage of using an inflective identity is that $o \mathbin{\$} x$ can be re-scaled to be a linear function of x (this applies even if the curve equation is not transformed to Weierstrass form).

For example, suppose that $e(x) = x_0^3 + x_1^3 + x_2^3 + x_3^3$, and we define V by $a_0x_0 + a_1x_1 + a_2x_2 + a_3x_3 = 0$ for some vector a. If $a_2 = a_3$, then it seems that the vector $(0, 0, 1, -1)$ corresponds to an inflection point on the curve. If we make o the additive identity, then we can use the linear vector operation $x = (x_0, x_1, x_2, x_3) \mapsto \ominus x = (x_0, x_1, x_3, x_2)$ for negation of projective points. Then we can define $x \oplus y$ as $\ominus(x \mathbin{\$} y)$.

We henceforth assume an implicit choice of inflective point o and corresponding operation $\oplus$. Recall that a cubic may have up nine inflection points, so the implicit addition $\oplus$ could represent any of these.

Doubling laws The conventional approaches to ECC also make use of a doubling law in forming multiple of a point under the group law.

The usual geometric description of doubling involves the tangent line. Doubling requires a choice of o, so is not computable using \$ alone. All is not lost, however. Negated doubling can be achieved using only the operation \$ and some auxiliary vectors on the curve as described later below.

The \$ operation is alternating: $x \mathbin{\$} y = -(y \mathbin{\$} x)$. In particular $x \mathbin{\$} x = (0, 0, \dots)$, so unfortunately this most natural way of computing the negated double fails. An implicit, non-constructive, way to compute the negated double of x, is to find y such that $x = x \mathbin{\$} y$. Further below, we give a more explicit constructive method.

1.2.2 Using \$ and V

This section proceeds to pursue the exercise of describing a form of ECDH, while only using the operation \$ and the three-dimensional vector subspace V.

Adding any four points Consider the vector $v = (w \mathbin{\$} x) \mathbin{\$} (y \mathbin{\$} z)$. Then v is nonzero if:

- w and x represent distinct projective points, and
- y and z represent distinct projective points, and

- $w \mathbin{\$} x$ and $y \mathbin{\$} z$ represent distinct projective points.

If v is nonzero, then v represents projective point $w \oplus x \oplus y \oplus z$, for the implicit $\oplus$ mentioned above. In this case, the ambiguity of $\oplus$ vanishes: the resulting of $w \oplus x \oplus y \oplus z$ is the same for each of the nine inflection points o. The vanishing of the ambiguity happens because an alternative inflection point o' has order three. For example, $o' \oplus o' \oplus o' \oplus o' = o'$.

If $v = (0, 0, \dots)$, then v fails to represent $w \oplus x \oplus y \oplus z$. This case is called an *exception*, using the terminology of Bosma and Lenstra [BL95].

Note this computation has a cost of 3\$ operations. This cost is somewhat promising for this exercise, because it appears as though we can trade three implicit $\oplus$ operations for three \$, provided that we can arrange to avoid any exceptional cases. Nevertheless, recall that we are not aiming for great efficiency here, so we do not further pursue that level of efficiency.

Pre-computed auxiliary vectors The following calculations will use two pre-computed arbitrary vectors $w, z \in e \cap V$. These can be included in the specification of V, or they can be found as follows.

To choose a random $z = (z_0, z_1, z_2, \dots)$, choose z_0 and z_1 as arbitrary, even random, field elements. Think of z_0 and z_1 as constants. Using the description of the space V to describe any extra coordinates $z_3, z_4, \dots$ as linear functions of z_2. Then $e(z) = 0$ can be viewed as a univariate cubic polynomial in variable z_2. Solve for z_2. If this univariate cubic has no roots, then try again.

In this remainder of this section, consider w and z to be fixed vectors.

Negated doubling Vector $u = (w \mathbin{\$} z) \mathbin{\$} ((w \mathbin{\$} x) \mathbin{\$} (y \mathbin{\$} z))$, if non-zero, represents the point $\ominus(x \oplus x)$, the negated doubling of x, under any of the implicit group structure mentioned above. (where o is an inflection point)

The use of the points w and z can be called bootstrapping, since using \$ and x alone leads nowhere. This booting process implies exceptions: for a few values of x, the negated doubling approach above yields the zero vector.

For convenience, write this as $u = \bigoplus^{-2} x$, or as $\bigoplus_{w,z}^{-2} x$ to emphasize the dependence on w and z as a vector operation, or as a projective point operation where w and z only matter for defining the exceptions.

Not to pinch pennies, but the total cost of operation $\bigoplus^{-2}$ is 5\$.

Quadrupling to boot Repeating the negated doubling operation twice gives: $\bigoplus^{-2} \bigoplus^{-2} x = \bigoplus^{4} x$.

This is the second and last phase of the booting process, and introduces some further exceptions.

Overall, booting has cost 10\$ and yielding the vector representation of x, $\bigoplus^{-2} x$ and $\bigoplus^{4} x$, albeit with an intangible cost that there are few x for which the booting process fails.

Re-scaled double-and-add The following variant of double-and-add can be used to compute the point $\bigoplus^{3n'+1} x$ given $\bigoplus^{3n+1} x$ for $n' \in \{2n, n+1\}$:

$$
\begin{aligned}
\bigoplus^{3(2n)+1} x &= \left(\left(\bigoplus^{-2} x\right) \$ \left(\bigoplus^{3n+1} x\right)\right) \$ \left(\left(\bigoplus^{3n+1} x\right) \$ \, x\right), \\
\bigoplus^{3(n+1)+1} x &= \left(\left(\bigoplus^{4} x\right) \$ \left(\bigoplus^{3n+1} x\right)\right) \$ \, x, \\
\bigoplus^{3(n+0)+1} x &= \left(\left(\bigoplus^{1} x\right) \$ \left(\bigoplus^{3n+1} x\right)\right) \$ \, x.
\end{aligned}
$$

A formula with $n' = n + 0$ with cost equal to the formula for $n' = n + 1$ is included above to help make the overall computation run in roughly constant time. Recall that double-and-add works as follows: to compute the point for n' apply the formula above with $n = n' - 1$ if n' is odd and $n = n'/2$ if n is even (with an intermediate $n = n' - 0$ if constant-time implementation is warranted). Therefore, the bits in the binary expansion say which of the formulas above to apply.

Note that the computations above do not use points w and z, and generally do not add any new exceptions. More precisely, exceptions only arise given a dependency between n and the order of the point x in the elliptic curve group. Izu and Takagi [IT03] considered this problem for various elliptic curve addition laws (and addition chain methods).

The cost of the approach is about $5(3 + \lfloor \log_2(n) \rfloor)\$$. This is considerably more \$ than the number of $\oplus$ operations that a traditional ECDH would use.

Elliptic curve Diffie–Hellman Alice chooses her private ECDH key as an integer $3a + 1$ for some random integer a that can represent a unbiased value in the prime field F. Bob choose b similarly.

Let v be some point of V that Alice and Bob agree to use. Then Alice computes $x = \bigoplus^{3a+1} v$ and sends x to Bob. Bob computes $y = \bigoplus^{3b+1} y$ and sends y to Alice. Alice computes the point $\bigoplus^{3a+1} y = \bigoplus^{(3a+1)(3b+1)} v$. Similarly, Bob computes $\bigoplus^{3b+1} x = \bigoplus^{(3b+1)(3a+1)} v$. Alice and Bob obtain vector representation of the same projective point.

Alice and Bob must use the same method of representing points, because their computed vectors may differ. Also, because of known attacks, such as [NSS04], Alice and Bob should send each other the values x and y using the canonical representations mentioned above.

1.3 Personalized group selection

Alice and Bob may want to use a personalized Diffie–Hellman group for key exchange. Such a practice goes against the trend of using a common but carefully vetted Diffie–Hellman group, and is generally deemed as dubious. Perhaps Alice and Bob do not trust the common group. Perhaps Alice and Bob think that using a personalized group confers to them the benefit that an adversary cannot amortize the cost of attack against private keys across multiple users or key exchanges.

Normally, a personalized Diffie–Hellman group is very costly to generate, at least if one tries to follow the common strategy for a common group. For example, one must perform point-counting, primality-checking and so on.

In the case of elliptic curve Diffie–Hellman groups, an alternative strategy is available. For a given fixed field, a random projective plane cubic is likely to define an elliptic curve. If the field size is chosen large enough (larger than usual for common curves), then the elliptic curve group order can be expected to have a prime factor of large size, just by chance. Alice and Bob can then just trust to chance that the random curve that use is secure.

A straightforward way for Alice and Bob to choose a pseudorandom elliptic curve Diffie–Hellman group would be to use some standard curve form, such as a short Weierstrass equation like $y^2 = x^3 - 3x + b$, and to then choose b as the hash of their identities and some other information, such as the date.

An alternative approach using the operation \$ is as follows. Alice and Bob would choose their preferred cubic, such as $e(x) = x_0^3 + x_1^3 + x_2^3 + x_3^3$, but then select a three-dimensional subspace V in a pseudorandom manner, depending on their identities and the date.

Alice and Bob may further want an unpredictable curve, say, to avoid the risk of an adversary pre-computing in advance some attack information for their curve. In this case, they need to generate some unpredictable nonces, and then derive the curve from the nonces. We must assume that they can authenticate the nonces, which they might do with digital signatures.

The resulting pseudorandom vector space V can be specified either by its defining equations, or by selection of some basis. For example of the latter, Alice might choose a basis vector w, and Bob might choose a basis vector z. Alice sends w to Bob and Bob sends z to Alice. Alice choose secret integers $a \equiv 1 \bmod 3$ and Bob chooses b similarly. Alice computes $x = \bigoplus^a v$ and sends x to Bob, where v is the third basis element of V, chosen by some pre-arranged method. Bob computes $y = \bigoplus^b v$ and sends y to Alice. Two of these four communications can be merged, resulting in three passes between Alice and Bob.

1.4 An alternating quaternary operation?

If V is a three-dimensional subspace of the vectors on which e acts, then equip V with a vector cross product, written as $\wedge$, by fixing some 3-basis. For $w, x, y, z \in e, V$, let

$$v(w,x,y,z) = \frac{(w \mathbin{\$} x) \mathbin{\$} (y \mathbin{\$} z)}{\left(\frac{e((w \wedge x) \wedge (y \wedge z))}{\left|\begin{pmatrix} w \\ x \\ y \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix}\begin{pmatrix} y \\ z \\ w \end{pmatrix}\begin{pmatrix} z \\ w \\ x \end{pmatrix}\right|}\right)^3}, \tag{2}$$

interpreting w, x, y, z as row vectors, in the 3-basis over which $\wedge$ is defined, to form square matrices like $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$. In other words, the determinant $\left|\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right| = x \cdot (y \wedge z)$ is the triple product of x, y, and z. Strictly speaking, v is only defined where the denominator in (2) is nonzero.

Apparent properties of v A few simple numerical calculations suggest that the function v is

- an alternating function of its four inputs (swapping two inputs negates its value),
- equivalent in its action on surface $e \cap V$ to a polynomial vector function of multi-degree $(4, 4, 4, 4)$ in the coordinates.[3]

From alternation to quasi-associativity A consequence of the alternating property of function v is a kind of quasi-associativity of the operation \$ in the form:

$$(w \mathbin{\$} x) \mathbin{\$} (y \mathbin{\$} z) \propto (w \mathbin{\$} y) \mathbin{\$} (x \mathbin{\$} z), \tag{3}$$

[3]Achieving this polynomiality seemed to require including the matrices and determinants in (2).

where $u \propto v$ means that $Fu \subseteq Fv$ or $Fv \subseteq Fu$, provided that at least one of the corresponding denominators in (2) does not vanish[4]. The relation $\propto$ is transitive over nonzero vectors, and its equivalence classes (of nonzero vectors) correspond to projective points.

Basic point operation properties of \$ A potential theoretical application of quasi-associativity is to help establish associativity of another operation. But first, we note that Proposition 2 implies a kind of involutionary property:

$$(x \,\$\, y) \,\$\, y \propto x. \tag{4}$$

We also note the more obvious commutativity property $y \,\$\, x \propto x \,\$\, y$, which follows from the operation \$ being alternating on its vectors inputs: $y \,\$\, x = -x \,\$\, y$.

Jacobi's construction of the group law Fix $o \in e$ and define the operation $\oplus$ for any $p, q \in e$ by

$$p \oplus q = o \,\$\, (p \,\$\, q). \tag{5}$$

Jacobi first introduced the operation $\oplus$ as a group law for an cubic curve in 1835, (according Husemöller [Hus04, §5]).

Jacobi's now famous addition rule is illustrated schematically in Figure 2. The dashed lines and curves indicate a plane cubic curve. The curve is nearly singular, so it looks like a union of a horizontal line and a circle in the figure. The solid line segments indicating two chords, and cover a middle portion of the nearly horizontal part of the curve. Small solid disks represent the points involved, as labeled. The similarity between the symbol $\oplus$ and the figure is, of course, contrived and atypical, but may nonetheless have some mnemonic value.

From quasi-associativity to associativity It is also well-known [ST92, Ex. 1.11] that certain properties of an operation like \$ yield associativity of

[4]If the polynomial-action property of v holds, then this proviso can be dropped since the numerator will be zero making (3) hold vacuously.

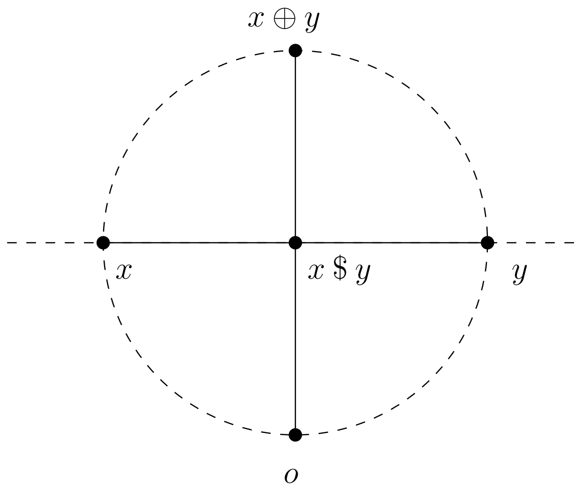

Figure 2: Jacobi's addition rule

$\oplus$. For example, if $o, p, q, r \in V$, then:

$$\begin{aligned}
p \oplus (q \oplus r) &= o \$ (p \underbrace{\$}_{\text{commute}} (o \$ (q \$ r))) \\
&\propto o \$ ((o \$ (q \$ r)) \$ \underbrace{p}_{\text{involution}}) \\
&\propto o \$ ((o \$ (q \$ r)) \$ ((p \$ q) \$ q)) \\
&= o \$ ((\underbrace{o}_{w} \$ \underbrace{(q \$ r)}_{x}) \$ (\underbrace{(p \$ q)}_{y} \$ \underbrace{q}_{z})) \\
&\propto o \$ ((\underbrace{o}_{w} \$ \underbrace{(p \$ q)}_{y}) \$ (\underbrace{(q \$ r)}_{x} \$ \underbrace{q}_{z})) \\
&\propto o \$ ((o \$ (p \$ q)) \$ ((r \$ q) \$ q)) \\
&\propto o \$ ((o \$ (p \$ q)) \$ r) \\
&= (p \oplus q) \oplus r,
\end{aligned} \tag{6}$$

which establishes that $p \oplus (q \oplus r) \propto (p \oplus q) \oplus r$ in the cases where all the intermediate vectors are nonzero, because transitivity applies making chain of $\propto$ carry through from start to finish. Exceptions to the argument occur if any of the intermediate vectors are zero.

From associativity to alternation? The arguments above do not seem to work in the converse direction. Associativity of $\oplus$, which holds at the level of point operations, does not seem to immediately imply the alternation of v as a vector operation.

2 Proofs of the propositions

Presuming that the three propositions are correct, one should expect that the formula \$ and the proofs of its properties are already known.

Cursory searches of a few introductory texts did not yield such proofs. Therefore, for completeness, this report proves the basic properties of the operation \$.

2.1 Proofs for non-binary fields

For *non-binary* fields (meaning field of zero or odd characteristic with $1+1 \neq 0$), the proofs of the first three propositions are made simple by using the following lemma.

Lemma 1. *If* $e(x) = e(y) = 0$*, then*

$$ax \mathbin{\$} by = (ab)^2(x \mathbin{\$} y) \tag{7}$$

for all $a, b \in F$*.*

Proof. In the expansion of the coordinates of the right side of equation (1) as polynomials in the coordinates of x and y, the scalar multipliers $e(x-y)$ and $e(x+y)$ each have total degree 3, while the the vectors x and y each have total degree 1. So, the total degree of $x \mathbin{\$} y$ is $3 + 1 = 4$.

Consider the degrees in the coordinates of x separately from the degrees in the y coordinates. Each non-zero monomial term in the polynomial expansion $x \mathbin{\$} y$ has a degree m in the coordinates of x and a degree n in the coordinates of y. The pair (m, n) is called the bi-degree of the term. Because the total degree is four: the bi-degree (m, n) obeys $m + n = 4$.

Swapping the two terms on the right of equation (1) shows that $x\$(-y) = x\$y$. This makes $x\$y$ an even function of the vector y. Evenness implies that terms in the polynomial expansion have even degree in the y coordinates[5]: so n is even. Consequently, $m = 4 - n$ is even too.

The monomial terms with bi-degree $(m, n) = (4, 0)$ in the polynomial expansion $x \mathbin{\$} y$ do not depend on y. Therefore, the bi-degree $(4, 0)$ sum to $x \mathbin{\$} (0, 0, \dots)$. Setting $y = (0, 0, \dots)$ on the right of equation (1) shows

[5]This argument may implicitly require working in an infinite extension of the field F.

that $x \mathbin{\$} (0,0,\dots) = 2e(x)x$. If $e(x) = 0$, then terms with $m = 4$ make no contribution to $x \mathbin{\$} y$ and can effectively be dropped for the purpose of evaluation of $x \mathbin{\$} y$.

Swapping x and y on the right of equation of (1) has the effect of negating both term, so $y \mathbin{\$} x = -(x \mathbin{\$} y)$. In terms of a polynomial expansion, this implies a kind of symmetry, up to signs, between x and y in the nonzero monomial terms. Because the terms of bi-degree $(4, 0)$ can be dropped if $e(x) = 0$, as shown above, this symmetry implies the terms with bi-degree $(0, 4)$ can be dropped when $e(y) = 0$.

The remaining terms all have bi-degree $(m, n) = (2, 2)$. Consequently, each term scales by $(ab)^2$ when x is scaled by a and y by b. The sum of terms scales similarly by linearity. $\square$

The bi-homogeneity of the operation \$ relative to the cubic e ensures that it is well-defined as an operation of projective points.

The bi-homogeneity of the operation \$ enables scaling of x and y to ax and by. This scales $x + y$ to $ax + by$. Setting $ax + by$ to any target point in $Fx + Fy$ is a key step in the next three proofs.

Proof of Proposition 1. The goal here is to show that if $e(x) = e(y) = 0$, then $e(x \mathbin{\$} y) = 0$.

If x and y are linearly dependent, then $y = ax$, or $x = ay$ for some scalar a. In the first case, $x \mathbin{\$} y = a^2(x \mathbin{\$} x) = e(0,0,\dots)(2x) + e(2x)(0,0,\dots) = (0,0,\dots)$, so $e(x \mathbin{\$} y) = 0$. The case $x = ay$ has the same argument. Otherwise, x and y are linearly independent, which is assumed for the rest of the proof.

If $x \mathbin{\$} y \in Fx$ or $x \mathbin{\$} y \in Fy$, then $e(x \mathbin{\$} y) = 0$ by homogeneity. Otherwise, $x \mathbin{\$} y \notin Fx \cup Fy$, which is assumed for the rest of the proof.

Because $x \mathbin{\$} y \in F(x + y) + F(x - y) \subseteq Fx + Fy$, there exists a, b, such that $x \mathbin{\$} y = ax + by$. By the assumption that $x \mathbin{\$} y \notin Fx \cup Fy$, it follows that $a, b \neq 0$. Write $z = ax + by = x \mathbin{\$} y$ and $w = ax - by$. Vectors z and w are linearly independent because x and y are. From Lemma 1:

$$\begin{aligned}(ab)^2 z &= (ab)^2 (x \mathbin{\$} y)\\ &= ax \mathbin{\$} by\\ &= e(w)z + e(z)w.\end{aligned} \tag{8}$$

Therefore $(e(w) - (ab)^2)z + e(z)w$ is zero vector. By the linear independence of w and z, both both coefficients in the linear combination must be zero. In particular, $e(z) = 0$. $\square$

Proof of Proposition 2. The task here is to show that if $x, y, z \in e$ and $z \in (Fx + Fy) \setminus (Fx \cup Fy)$, then $x \,\$\, y \in Fz$.

Writing $z = ax + by$ for $a, b \neq 0$, applying Lemma 1:

$$\begin{aligned}(x \,\$\, y) &= (ab)^{-2}(ax \,\$\, by) \\ &= (ab)^{-2}(e(ax - by)z + (ab)^{-2}e(z)(ax - by)) \\ &= (ab)^{-2}e(ax - by)z,\end{aligned} \tag{9}$$

which shows that $x \,\$\, y \in Fz$. □

Proof of Proposition 3. The task here is to show that if $\dim(Fx + Fy) = 2$, and $x, y \in e$, and $x \,\$\, y = (0, 0, \dots)$, then $Fx + Fy \subset e$.

Let $z \in Fx + Fy$. If $z \in Fx$, then $e(z) = 0$ by homogeneity. Similarly, if $z \in Fy$. Otherwise, $z = ax + by$ for some $a, b \neq 0$. Let $w = ax - by$. Vectors z and w are linearly independent. From Lemma 1,

$$\begin{aligned}(0, 0, \dots) &= x \,\$\, y \\ &= (ab)^{-2}(ax \,\$\, by) \\ &= (ab)^{-2}(e(w)z + e(z)w)\end{aligned} \tag{10}$$

By linear independence of z and w, we must have $e(w) = e(z) = 0$. In particular $e(z) = 0$, which holds for any $z \in Fx + Fy$. □

2.2 Proofs for all fields

In fields of characteristic two, equation (1) always evaluates to the zero vector, since $1 = -1$ causes both terms to be equal, canceling each other. Furthermore, the proofs of the first three propositions use trick that $ax + by$ and $ax - by$ are independent if x and y are, which does not work in characteristic two. So, in this section, we amend the definition of the operation \$, and provide rather different proofs.

Consider the ring $R_x = \mathbb{Z}[x_i, e_{i,j,k}]$, and consider $e(x)$ as an element of R. Notice that e is cubic in the x_i but linear in the $e_{i,j,k}$. For any other ring S and any vector z, we can define $e(z)$ by mapping x_i to z_i, and some appropriate mapping of the coefficients $e_{i,j,k}$ into the ring S: this is called the evaluation of e at vector z.

Consider the ring $R_{x,y} = \mathbb{Z}[x_i, y_i, e_{i,j,k}]$ which extends R_x. We evaluate $e(z)$ for a vector z with coordinates in $R_{x,y}$ by mapping $e_{i,j,k}$ in R_x to its copy in $R_{x,y}$. Now define the operation \$ as:

$$x \,\$\, y = (e(x - y)(x + y) + e(x + y)(x - y))/2, \tag{11}$$

where the division above is possible for integer coefficients, because each monomial of given degrees appears in both terms, only differing by signs. When monomials collected by common degrees in variables, the coefficients will be even integers. Therefore $x \,\$\, y \in R_{x,y}$.

When x and y are vectors over an arbitrary ring S, we can define $x \,\$\, y$ using the usual substitution of variables by coordinates and constants. This provides our modified definition of the operation \$ for any vector space, and any given cubic (where the $e_{i,j,k}$ are assigned constant values).

In fields of odd or zero characteristic, the new operation \$ is just our the old operation \$ divided by two. Because two is invertible in this fields, all the old propositions apply to the new operation \$, and the effect on projective points is identical.

Let

$$f(x, y) = (e(x + y) - e(x - y))/2, \tag{12}$$

where the division by two is integral by the same arguments as for the new operation \$, so $f(x, y) \in R_{x,y}$. Then:

$$f(y, x) = (e(y + x) - e(y - x))/2 = (e(x + y) + e(x - y))/2 \tag{13}$$

from the fact that $e(-z) = -e(z)$. It follows that:

$$e(x + y) = f(x, y) + f(y, x) \tag{14}$$

Next consider that

$$f(x, -y) = (e(x - y) - e(x + y))/2 = -f(x, y), \tag{15}$$

so $f(x, y)$ is odd as a function of y. This means that, as a polynomial, all its terms have total odd degree in the variables y_i.

Lemma 2. *If $e(z) = 0$, then $f(ax, bz) = a^2 bf(x, z)$ for all scalars a, b.*

Proof. We showed above that the polynomial $f(x, y)$ only has terms of odd degree in y. The total degree of f is 3, like e, so y degree of each term is either 3 or 1.

In terms of y-degree 3, the x-degree is 0. The sum of these terms is $f(0, y)$ (where 0 here is the zero vector). But $f(0, y) = (e(y) - e(-y))/2 = e(y)$. Therefore, these y-cubic terms make no contribution $f(x, z)$, since $e(z) = 0$.

In particular, the terms contributing to $f(x, z)$ have degree 1 in z and degree 2 in x. $\square$

In fact, one can expand f as follows:

$$f(x,y) = e(y) + \sum_{0\le i\le j\le k} e_{i,j,k}(x_i x_j y_k + x_i y_j x_k + y_i x_j x_k), \tag{16}$$

which holds as an equation in $R_{x,y}$.

For the rest of this section, we revert to the notation x and y vectors in an arbitrary ring: so x and y are no longer restricted to be vectors in $R_{x,y}$. This allows use to write $e(x) = e(y) = 0$ and to state claims similar to the rest of this report.

Lemma 3. *If $e(x) = e(y) = 0$, then $e(ax+by) = a^2bf(x,y) + ab^2f(y,x)$ for all scalars a, b.*

Proof. Combine Lemma 2 and equation (14). □

Returning to our new operation \$ definition, by regrouping terms it follows that:

$$x \mathbin{\$} y = f(x,y)y - f(y,x)x \tag{17}$$

In this form, one easily sees that the new operation \$ is also bi-quadratic, as per Lemma 1.

Second proof of Proposition 1. The goal here is to show that if $e(x) = e(y) = 0$, then $e(x \mathbin{\$} y) = 0$. Expand as follows:

$$\begin{aligned} e(x \mathbin{\$} y) &= e((-f(y,x))x + f(x,y)y) \\ &= (-f(y,x)^2 f(x,y))f(x,y) + (-f(y,x))(f(x,y)^2)f(y,x) \\ &= 0, \end{aligned} \tag{18}$$

where: the first equality above uses equation (17) with the order of terms swapped; the second equality uses Lemma 3; and the third equality follows by arithmetic. □

Second proof of Proposition 2. The task here is to show that if $x, y, z \in e$ and $z \in (Fx + Fy) \setminus (Fx \cup Fy)$, then $x \mathbin{\$} y \in Fz$.

If $z \in Fx + Fy$ with $z \notin Fx \cup Fy$, then there exists nonzero scalars such that $z = ax + by$. Now:

$$\begin{aligned}
x \,\$\, y &= f(x,y)y - f(y,x)x \\
&= (ab)^{-2}(f(ax,by)by - f(by,ax)ax) \\
&= (ab)^{-2}(f(ax,by)by + (f(ax,by) - f(ax,by) - f(by,ax))ax) \\
&= (ab)^{-2}(f(ax,by)by + (f(ax,by) - e(ax+by))ax) \\
&= (ab)^{-2}(f(ax,by)by + (f(ax,by) - e(z))ax) \\
&= (ab)^{-2}(f(ax,by)by + f(ax,by)ax) \\
&= (ab)^{-2}f(ax,by)(ax+by) \\
&= (ab)^{-2}f(ax,by)z
\end{aligned} \tag{19}$$

which shows that $x \,\$\, y \in Fz$. □

Second proof of Proposition 3. The task here is to show that if $\dim(Fx + Fy) = 2$, and $x, y \in e$, and $x \,\$\, y = (0, 0, \dots)$, then $Fx + Fy \subset e$.

The vector $x \,\$\, y = f(x,y)x - f(y,x)y$ is a linear combination of independent vectors x and y and can only be zero if the coefficients of x and y are both zero. So, $f(x,y) = f(y,x) = 0$.

Suppose $z \in Fx + Fy$, which means that there exists nonzero scalars such that $z = ax + by$. Now:

$$\begin{aligned}
e(z) &= e(ax+by) \\
&= f(ax,by) + f(by,ax) \\
&= a^2bf(x,y) + ab^2f(y,x) \\
&= 0.
\end{aligned} \tag{20}$$

which shows that e vanishes on $Fx + Fy$. □

References

[BL95] W. Bosma and H. W. Lenstra, Jr. Complete systems of two addition laws for elliptic curves. *J. Number Theory*, 53(2):228–240, 1995.

[BL15] Daniel J. Bernstein and Tanja Lange. Explicit-formulas database. `hyperelliptic.org/EFD`, 2015.

[Hus04] Dale Husemöller, editor. *Elliptic Curves*, volume 111 of *Graduate Texts in Mathematics*. Springer, 2nd edition, 2004.

[HWCD08] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted edwards curves revisited. In Josef Pieprzyk, editor, *Advances in Cryptology – ASIACRYPT 2008*, number 5350 in LNCS, pages 326–343. Springer, 2008. See also `eprint.iacr.org/2008/522`.

[IT03] Tetsuya Ize and Tsuyoshi Takagi. Exceptional procedure attack on elliptic curve cryptosystems. In Yvo G. Desmedt, editor, *Public key cryptography – PKC 2003*, number 2567 in LNCS, pages 224–239. Springer, 2003.

[NSS04] David Naccache, Nigel P. Smart, and Jacques Stern. Projective coordinates leak. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, number 3027 in LNCS, pages 257–267. Springer, 2004.

[NZM91] Ivan Niven, Herbert S. Zuckerman, and Hugh L. Montgomery. *An Introductio to the Theory of Numbers*. John Wiley & Sons, Inc., 5th edition, 1991.

[Sch91] Norbert Schappacher. Developpment de la loi de groupe sur une cubique. *Progress in Mathematics*, 91:159–184, 1991.

[ST92] Joseph H. Silverman and John Tate. *Rational Points on Elliptic Curves*. Undergraduate Texts in Mathematics. Springer, 1992.

A History

Schappacher [Sch91][6] discusses the history of the elliptic curve group law. Schappacher notes that Sylvester describes a procedure for deriving a sequence of elliptic curve points from a single point, naming them by integers corresponding to what modern notation describes as multiplies under the group law. But Sylvester does not fully describe the group, and instead, a series of secants, very similar the procedure that this report describes for computing a subset of the multiples of a point.

In the complete works of Sylvester, one can find his name for the point obtained from two distinct points by intersecting the secant with the cubic: the *connective*. Sylvester refers to the connective in the context of projective points, rather than a more explicit vector operation. (Again, there are many

[6]Thanks to Samuel Neves for drawing my attention to Schappacher's article

vector operation corresponding to a single point operation, by scaling the points.)

B Simplistic code samples

In this section, some simplistic code samples using $ and a form of elliptic Diffie–Hellman described. The sample code is not optimized for efficiency. Worse yet, the sample code include no protections against side channels and other implementation security risks.

The sample code mostly omits explanatory comment: but this report is essentially the comment.

B.1 J script

The J programming language, an interpreter, can be useful for quickly testing some certain mathematical ideas. It has built-in big integers, multi-dimensional array manipulation, and a tacit syntax. The tacit syntax makes for terse and difficult reading of code, but for quick programming, once one accustomizes to it.

```
p   =: - 1 - 2x^521
inv =: (p&|@^)&(p-2)
aff =: p | }. (* inv) {.
prj =: 1 & ,
e   =: p | ([: +/ ^&3) " 1
s   =: [: +/ [: (e*|.) + ,: -
S   =: (p | s) " 1
w   =: 3  4   5 _6x
z   =: 1 12 _10 _9x
dbn =: (w S z) S (w & S) S (S & z)
btr =: dbn^: 0 1 2 2
sum =: _2&(S/\)^:2
dub =: }: (, sum) 0 3 3 1&{
itr =: (}:@:]) , (S/)@:(({~(0 3,+:))~dub)
rec =: ]`(({.@[)itr($:~}.)~)@.(0<#@[)
exp =:  [: aff [: {: (#:@:]) rec (btr@:prj@:[)
g   =: aff 1 6 8 _9x
```

Table 1: J code for ECDH using $

Table 1 provides a script. The last line includes an example base point. I took all the three points from Elkies' web page on the Fermat cubic, which makes it convenient to change the field size. The J verb `exp` implements exponentiation but the exponent is recoded.

B.2 C++ code (unsecured)

Not being a software developer, I seldom use C or C++. Operator overloading in C++ makes it easy to write code with appearance of standard mathematical notation. The algorithm for elliptic curve Diffie–Hellman described in this report is implemented in Table 2.

```
// May leak secrets
# include "vecctor.hh"
static scalar e (vector x) {return sum(cube(x));}
# define $ |
static vector operator $ (vector x, vector y)
{return e(x-y)*(x+y) + e(x+y)*(x-y);}
# define w vector_int(3, 4,  5,-6)
# define z vector_int(1,12,-10,-9)
static vector negative_double (vector x)
{return (w $ z) $ ((w $ x) $ (x $ z));}
typedef struct {
# define   EXP_LENGTH  521
  bool bit[EXP_LENGTH];
} bit_array;
point operator ^ (point base, bit_array exponent)
{vector x1, x_2, x4, y;
  x1 = projective (base);
  x_2 = negative_double (x1);
  x4 = negative_double (x_2);
  y = x4;
  for (int i=0; i<EXP_LENGTH; i+=1){
    y = (x1 $ y) $ (y $ x_2);
    y = ((exponent.bit[i] ? x4 : x1) $ y) $ x1;}
  return affine(y);
}
```

Table 2: C++ code usable for ECDH

This code is incomplete in that it requires an implementation of the vector space. The interface it uses to the vector space implementation is describe in the file `vecctor.hh` given in Table 3. A toy implementation of a

```
# include "vecctor_internal.hh"  // vector, point, scalar
vector operator + (vector, vector);   // add vectors
vector operator - (vector, vector);   // subtract vectors
vector operator * (scalar, vector);   // scalar vector
vector vector_int (int, int, int, int);  // initialize small vector
point  affine     (vector);  // projective vector --> affine point
vector projective (point);   // affine point --> projective vector
vector cube (vector);   // cube each vector coordinate
scalar sum  (vector);   // sum of vector coordinates
```

Table 3: File `vecctor.hh` with interface used by C++ code sample

vector space over the field of very small size 521 seems to interoperate with the previous J code when `p` is redefined to be 521.

M. Scott's recent C++ implementation of the field of size $2^{521} - 1$ can be adapted to fit this interface above. On a rather old personal computer, the C++ code from Table 2 runs about 10 times slower than Scott's ECDH implementation of Weierstrass curve P-521. This may be due to Scott' use of faster addition rules and to the better windowing. Although this speed is far from optimal, it may be adequate for end-to-end encryption between end users of high end devices.

Scott's field implementation together with Table 2 has not yet been successfully tested interoperabilty with the J code from Table 1, but perhaps only due to formatting differences.

The sample C++ code includes no effort into ensuring a secure implementation. For example, it may cause severe side channels, it may leave secret values in memory, and so on.