SAND REPORT

SAND2003-4282
Unlimited Release
November 2003

Algorithms for Improved Performance in Cryptographic Protocols

Richard Schroeppel and Cheryl Beaver

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm

SAND2003-4282
Unlimited Release
Printed November 2003

Algorithms for Improved Performance in Cryptographic Protocols

Richard Schroeppel and Cheryl Beaver
Cryptography and Information Systems Surety Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0785
rschroe, cbeaver@sandia.gov

Abstract

Public key cryptographic algorithms provide data authentication and non-repudiation for electronic transmissions. The mathematical nature of the algorithms, however, means they require a significant amount of computation, and encrypted messages and digital signatures posess high bandwidth. Accordingly, there are many environments (e.g. wireless, ad-hoc, remote sensing networks) where public-key requirements are prohibitive and cannot be used. The use of elliptic curves in public-key computations has provided a means by which computations and bandwidth can be somewhat reduced. We report here on the research conducted in an LDRD aimed to find even more efficient algorithms and to make public-key cryptography available to a wider range of computing environments. We improved upon several algorithms, including one for which a patent has been applied. Further we discovered some new problems and relations on which future cryptographic algorithms may be based.

Acknowledgement

The authors would like to thank Don Gallup, Nathaniel Blair-Stahn, Erin McNicholas and Livia Miller for their programming efforts. We also thank Bill Gosper for his research contributions.

Contents

Figures

Algorithms for Improved Performance in Cryptographic Protocols

1 Introduction

Cryptographic protocols provide means to protect critical data across insecure communication lines. Protocols such as HTTPS (which uses SSL) are used by browsers such as Netscape to give secure web sites the ability to provide services like e-commerce. Such protocols are too slow and bandwidth consuming for many technologies (e.g., wireless or ad-hoc networks). New military applications like minimally manned warfare and advanced logistics will require fast reliable cryptographic protocols to ensure the integrity of communications. As technology provides us with a greater ability to access, quickly and efficiently, all kinds of data, cryptographic protocols are essential to protect sensitive data from interception or modification. Public key algorithms provide security functionality such as non-repudiation and unique source authentication that symmetric key algorithms cannot provide.

Public key cryptography is distinguished by the fact that a different key is used to encrypt data than is used to decrypt the data. Because of this, the encryption key can be made public (hence the name public key) while the decryption key is kept private. Anyone can encrypt messages using the public key and only the person with knowledge of the corresponding private key can decrypt the messages. The owner of a private key can also use it to generate digital signatures (these signatures work much in the same way as usual signatures) that anyone can verify using the corresponding public key. The special knowledge of the private key in conjunction with cryptographic algorithms for digital signatures and encryption allows a means by which people (or any source of data - e.g. a website, sensor, cell phone) can uniquely identify themselves, authenticate data and provide receipts that cannot be repudiated even by the source. These capabilities are unique to public key cryptography and are extremely significant in our electronic world.

Public key cryptography works because the encryption and decryption keys are different but mathematically related. These mathematical relationships can be exploited. This has a two-fold ramification. First, the algorithms used to provide the encryption/digital signatures require mathematical operations that are relatively difficult for the computer to do when compared to the machine-type operations of traditional symmetric key cryptography. Second, since mathematical operations have "nice" properties and relationships, attacks against the cryptosystems are more sophisticated and so very large numbers have to be used in the computations to ensure security. This combination of difficult computations with very large numbers translates to large keys, lots of computer processor time, and high bandwidth requirements during transmissions for systems that use public key cryptography. For example, the RSA algorithm to provide encryption/digital signatures requires many modular exponentiations using numbers of size varying from 768-bits (lower security) to 2048-bits (high security). By comparison, elliptic curve algorithms use numbers of size 140-200-bits to achieve the same security. This is possible because elliptic curve algorithms use arithmetic operations defined on elliptic curves, rather than traditional modular arithmetic. The attacks on the usual algorithms don't translate over to elliptic curves and so smaller parameters can be used. Using elliptic curve algorithms in place of these traditional algorithms can speed up the computation time by a factor of ten.

This research is motivated by the success of using arithmetic on elliptic curves as a means to

reduce key size and speed up computations in public key algorithms. We present new algorithms that further exploit certain properties of elliptic curves and provide additional speedups of signature algorithms. For one of these algorithms, a patent has been applied for, and another has been used in a Sandia design for a fast digital signature chip. In addition, we explore the use of other groups as the basis for new cryptosystems. In particular, we examine the use of Dilogarithm and Recurrence Sequences.

The report is organized as follows: In Section 2 we introduce elliptic curves and discuss their applications to cryptography. In Sections 3,4,5, and 6 we discuss fast elliptic curve algorithms. Section 7 address speeding up computations in finite fields. Then in Section 9 we introduce the Dilogarithm problem and in Section 10 we discuss recurrence sequences and their possible applications.

2 Elliptic Curves

An elliptic curve consists of pairs $(x, y) = P$ (points), which are solutions to a certain cubic equation together with a distinguished point, $\mathcal{O}$, called the origin (point at infinity). What makes elliptic curves interesting to the field of cryptography is that there is an addition law that can be defined on the points of the curve that makes the set of points form a group in which the discrete logarithm problem is hard. The discrete logarithm problem is defined on the elliptic curve group as follows: Given an elliptic curve E and points $G, P = nG \in E$, find n. Many public key encryption and digital signature algorithms base their arithmetic in groups where the discrete logarithm problem is defined and base their security on the difficulty of solving this problem. Originally, these algorithms were defined using modular arithmetic over finite fields. However, Victor Miller [24] and Neal Koblitz [15] noticed that elliptic curve arithmetic could be used just as easily. Furthermore, traditional attacks on discrete log-based systems over finite fields do not carry over when the base group is an elliptic curve. For this reason, smaller key sizes/parameters can be used and the algorithms are often more efficient.

In addition to smaller parameters contributing to more efficient algorithms, further tricks can be applied to speed up calculations. The most intense computation in any elliptic curve cryptographic algorithm is the calculation of a large multiple of a point. There are a number of ways to speed this up: choosing special curves and fields to make the computations easier, improving the elliptic curve algorithms and improving the algorithms in the underlying fields.

2.1 Elliptic Curve Point Addition

In the most general case, an elliptic curve E is defined over a field K by the equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_i \in K$ and E consists of points $P = (x, y)$ where $x, y \in K$ together with a special point $\mathcal{O}$ at infinity. For a general introduction to elliptic curves, see [Sil86]. For use in cryptography, we consider only those points whose coordinates lie in some finite Galois field. Typically, the field is either $GF(p)$ for some large prime p, or $GF(2^n)$. For our purposes, we consider the case where the field of definition for the points on the curve is generally $GF(2^n)$. The arithmetic is faster and, in

that case, the curve E can be given by an equation of the following form

$$E : y^2 + xy = x^3 + ax^2 + b, \tag{1}$$

with $a, b \in GF(2^n)$. For convenience, if $P = \mathcal{O}$ we write its coordinates as $P = (0, 0)$ (note $(0, 0)$ does not satisfy the curve equation, this is just for bookkeeping in our algorithms).

2.1.1 Affine vs. Projective Coordinates

The equation for E can also be given in projective coordinates:

$$E : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{2}$$

The points of the curve are equivalence classes: $P = [X, Y, Z] \sim P' = [X', Y', Z']$ if there is a $\lambda \neq 0$ such that $X' = \lambda X$, $Y' = \lambda Y$, and $Z' = \lambda Z$. Note that if $Z \neq 0$, then there is always a representative of the class with $Z = 1$. These equivalence classes where $Z \neq 0$ correspond to the points on the curve as defined by equation (1) via the change of coordinates $x = X/Z$, and $y = Y/Z$. The only solution to the equation (2) for $Z = 0$ is the class containing $X = Z = 0, Y = 1$ and this corresponds to the special point $\mathcal{O}$ at infinity included with the equation (1). Throughout the rest of the paper we assume the curve is given in terms of affine coordinates, although most of the algorithms can be recast in terms of projective coordinates and in some cases this gives some efficiency improvements.

2.1.2 Point Addition

Suppose $M_1 = (x_1, y_1)$ and $M_2 = (x_2, y_2)$ are two points on the curve E. Then, the addition of the points, $M_3 = M_1 + M_2$ can be described geometrically (see Fig.1): Draw a line through the points M_1 and M_2. Since the curve is a cubic, the line will intersect the elliptic curve at exactly one other point, P. The point M_3 is the point on the curve defined by the reflection of P about the x-axis. An algebraic description of the algorithm is next.

Alg. 1. Point Addition

Input: $a, A = (x_A, y_A), B = (x_B, y_B)$
Output: $C = (x_C, y_C)$

1. *If $A = \mathcal{O}$ (e.g. $x_A = y_A = 0$), then output $C = B$ and stop*
2. *If $B = \mathcal{O}$, then output $C = A$ and stop*
3. *If $x_A \neq x_B$ then*
   (a) *Set $\lambda = (y_A + y_B)/(x_A + x_B)$*
   (b) *Set $x_C = a + \lambda^2 + \lambda + x_A + x_B$*
   (c) *GOTO step 7*
4. *If $y_A \neq y_B$ then output $C = \mathcal{O}$ and stop*
5. *If $x_B = 0$ then output $C = \mathcal{O}$ and stop*

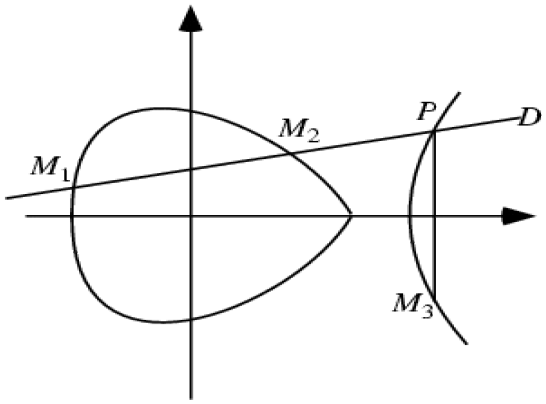

Figure 1. Point Addition: $M_1 = M_2 = M_3$.

6. *Set*

   (a) $\lambda = x_B + y_B/x_B$

   (b) $x_C = a + \lambda^2 + \lambda$

7. $y_C = (x_B + x_C)\lambda + x_C + y_B$

8. $C = (x_C, y_C)$

We note that the algorithm gives a different formula depending on whether the points to be added are different (Step 3) or the same (Step 6). In the latter case, we say the point is being doubled. If E is given by equation (1), then the negative of a point, $P = (x, y)$, is given by $-P = (x, x \oplus y)$. Subtraction $Q - P$ of points is simply Q plus the negative of P. We note that the special point $\mathcal{O}$ acts like the zero of the addition law. In particular, $\mathcal{O} + \mathcal{P} = \mathcal{P} + \mathcal{O} = \mathcal{P}$ and $P - P = \mathcal{O}$.

We write $E(K)$ for the set of points with coordinates in the field K. $E(K)$ forms a group under this addition law with identity element $\mathcal{O}$. The order of the group, written $\#E(K)$, is the number of points. Multiplication is defined on E as repeated addition: $nP = P + P + \cdots + P$ (n times).

2.2 Elliptic Curves In Cryptography

Any cryptographic algorithm whose security is based on the discrete logarithm problem can be recast into an elliptic curve based algorithm with security based on the elliptic curve version of the discrete logarithm problem. The most common such algorithm is the elliptic curve version of the Digital Signature Algorithm or ECDSA. Other examples include analogs of Diffie-Hellman key exchange and El Gamal type systems. In all cases, the main components needed are a *secure* curve (see Section 2.4) and an algorithm to compute a multiple of a point. We describe the ECDSA below to give a flavor for these types of operations found in EC cryptographic algorithms.

2.2.1 ECDSA

We describe the Elliptic Curve Digital Signature and Verification Algorithms for an elliptic curve E defined over a field $F = GF(q)$. Here q is either a large prime or a large power of 2. Let r be a large prime divisor of the order of E and let $G \in E$ be a point of order r. Let s be the private key and $W = sG$ the public key. Denote by f the message representative to be signed (the message or hash of message).

ECDSA Signature Algorithm

1. Generate a random integer $u \in [1, .., r-1]$ and set $V = uG$. Write $V = (x_V, y_V)$.
2. Compute an integer $c \equiv x_V \pmod{r}$. If $c = 0$ go back to step 1.
3. Compute the integer $d = u^{-1}(f + sc) \pmod{r}$. If $d = 0$ go back to step 1.
4. Output the signature pair (c, d).

ECDSA Verification Algorithm

1. If c or d is not in the range $1..r-1$ output invalid and stop.
2. Compute integers $h = d^{-1} \pmod{r}$, $h_1 = fh \pmod{r}$, and $h_2 = ch \pmod{r}$
3. Compute the elliptic curve point $P = h_1G + h_2W$. If $P = O$, output invalid and stop, else $P = (x_P, y_P)$
4. Compute an integer $c' \equiv x_P \pmod{r}$
5. If $c' = c$ output *valid*, else output *invalid*.

2.3 Computational Considerations

Step 1 in the signature algorithm and Step 3 in the verification algorithm are the most time consuming, power-intensive steps in the algorithm. These steps involve taking the multiple of an elliptic curve point. In fact, other than those steps, the only other computations done are some computations modulo r. There are well known tricks for optimizing the modulo r computations and so we concentrate on making the point multiplications more efficient. In a typical cryptographic system, the quantities involved in these computations (e.g. the finite field elements, the point coordinates, and the point multipliers) are a few hundred bits long.

2.3.1 The Number of Arithmetic Operations in Point Addition

The basis of the point multiplication is the point addition or doubling. When dissecting these algorithms for binary fields, we note that the time consuming operations are really the multiplies and the reciprocals (addition is just an xor and squaring is almost free). Hence, one way to compare the complexity of algorithms is to determine the number of multiplies and reciprocals used in each point addition formula and to look for algorithms that minimize point additions. In the general addition formula (2.1.2), there is a reciprocal required in step (3a) to compute $1/(x_A + x_B)$ followed

by a multiplication to compute $(y_A + y_B)/(x_A + x_B)$. There is another multiplication to compute the y coordinate in (7). The total work for a point addition (and also doubling) as given by this algorithm is roughly 2 Multiplies (M) and 1 Reciprocal (R), or $2M + R$. There are algorithms that offer cheaper alternatives: The work can be reduced to $M + R$ in [35, 42], or even just R [36] if special values of the curve parameter b are chosen. Point halving can be used instead of doubling [14, 37, 39] to reduce the work to M plus some side calculations (see Section 4). Reciprocals are generally more expensive than Multiplies. This is true whether we measure cost as simply execution time of a computer program, or hardware circuit delay, or circuit area or power consumption. Using program execution time as our cost metric, the relative cost ratio of reciprocals to multiplications, R/M, varies widely depending on circumstances. In [35], the reported ratio is 2.5. In work with multi-level field towers (see Section 7.1), the ratio is around 1.5. In [12] ratios are described to be around 10, and values as high as 60 have been reported.

2.4 Secure Curves and Point Counting

Prior to using an elliptic curve in a cryptographic algorithm, we need a well chosen, secure curve. As noted above, the curve is secure as long as the discrete logarithm problem is hard in the group of points over the finite field of interest. If the order of the group is N (i.e. the number of points in the group) where $N = \prod p_i$ with p_i prime, then the discrete log problem can be broken down into the easier discrete log problems over groups of order p_i. Hence, in order for the curve to be secure, we need the curve order to have a large prime divisor. The definition of "large" varies depending on the current state of the art in computing power and finding discrete logarithms. Currently the best attacks are square root in time. Always check current recommendations from experts when choosing curve parameters. In general, the order having a large prime divisor is a sufficient condition, but there are some additional properties that may cause some curves to be "weak" (e.g. [23]).

The number of points on an elliptic curve E with coordinates in a finite field $F = GF(q)$ (where $q = p^n$ for some prime p) is given by the formula

$$\#E(F) = q + 1 - t \tag{3}$$

Here t is the trace of the Frobenius map: $Fr(x) = x^q$. The first polynomial time algorithm for point counting was due Schoof [33]. The idea was to compute t modulo ℓ for small primes ℓ and then use the Chinese remainder theorem to calculate t. Improvements to the algorithm were given by Elkies and Atkin that led to the SEA algorithm [34]. Other improvements were given in [26, 5]. The run time for these algorithms over fields of small characteristic is approximately $O(log^{4+\epsilon}q)$ time.

In 2000, Satoh [31] came up with an alternative method for point counting which effectively solved the point counting problem in the sense that it is no longer considered hard or time consuming. His idea was to lift the curve E to the curve $\tilde{E}$ over the p-adic ring $\mathbf{Z}_q$ which reduced to E. The trace of the frobenius could then be computed easily. Variants and improvements of this idea have appeared since then (e.g. [7, 48, 8, 32, 17]) the main variant being how the curve is lifted. The best run time for these algorithms over fields of small characteristic is currently about $O(log^{2+\epsilon}q)$ (e.g. [17]).

For this project we implemented a version of Satoh's algorithm for elliptic curves over fields of characteristic two. For more information or copies of the software contact the authors.

3 Faster Point Multiplication

There are many ways to make point multiplication more efficient. The traditional tricks for speeding up modular exponentiation will also apply to elliptic curve multiplication. Some common techniques for faster point multiplication inclue precomputing values (e.g. [3], [22]), windowing methods, addition and subtraction chains (e.g. [27]), mixing projective and affine coordinate point representations [4], and multi-doubling [21]. Another Sandia project developed a particulary fast method for computing a multiple of a point where the base point is known [47]. Other methods include choosing particular curves over which computations may be easier (e.g. Koblitz curves [16, 42, 44]), or for which an eficiently computable endomorphism may speed up the point multiplication (e.g. [9, 44]) or working over special fields (e.g [45, 43, 28]). In the latter cases, one must be careful that the choice of special curve or field does not lead to reduced security (see [23, 10]).

4 Point Halving

One way to modify the point multiplication algorithm is to use a point halving algorithm in place of a doubling algorithm. The idea of "halving" a point $P = (x_P, y_P)$ is to find a point $Q = (x_Q, y_Q)$ such that $2Q = P$. Note this is the inverse of the point doubling problem. The point halving can nevertheless be used in algorithms by a simple adjustment on the base point of the elliptic curve used. The algorithm offers a speed-up in software of a factor of about two to three over the point doubling algorithm. We follow the algorithm of [37] developed by the first author.

For this algorithm we sometimes write the coordinates of the points $P \in E$ as (x_P, r_P) where $r_P = y_P/x_P$. In fact, we use the (x_P, r_P) form whenever possible, but the input and output of the point addition algorithm need the Y coordinate, so the halving algorithm must handle Y outputs and inputs. When the y_Q output is not required, the point halving algorithm needs only one field multiplication. It is most efficient when point halvings are consecutive. If a signed sliding window multiplication method is used, there are about five halvings between additions.

Alg. 2. *Point Halving over* $GF(2^m)$

Input: $P \in E$ **Output:** $Q = \frac{1}{2}P \in E$

1. $M_h = Qsolve(x_P + a)$, *where* a *is the curve parameter*
2. $T = x_P * (M_h + r_P)$ *or* $T = x_P * M_h + y_P$
3. *If parity(*T *and* t_m*)*$= 0$, *then* $M_h = M_h + 1$; $T = T + x_P$

   Here t_m *is a mask that depends upon the modulus polynomial. In our case,* $t_m = (u^{51} + 1, 0)$.
4. $x_Q = \sqrt{T}$
5. $r_Q = M_h + x_Q + 1$
6. *If needed,* $y_Q = x_Q * r_Q$

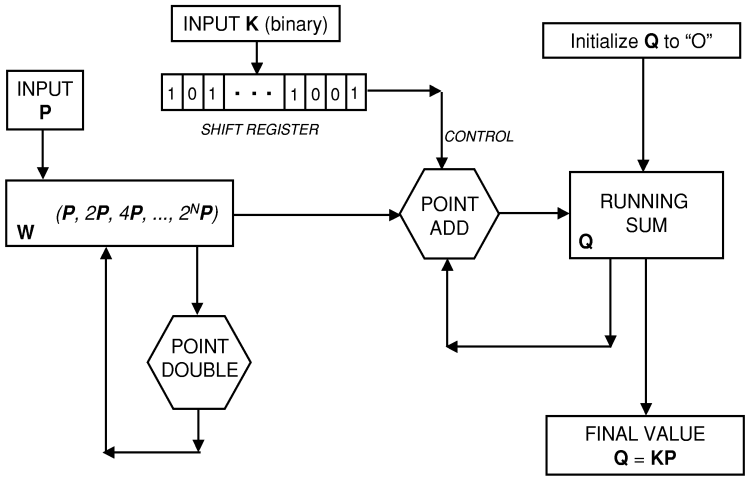

Figure 2. Double and Add

The $Qsolve(a)$ in step 1 is an algorithm to find the solution, x, to the quadratic equation $x^2 + x = a$. For more details, see section 7.2. The point halving algorithm only requires one Multiply and some side computations ($\sim 1.3M$)(note there are no reciprocals required) and hence often gives a savings over doubling when computing the multiple of a point.

5 Double And Add

Point multiplication is generally done by utilizing two operations: the doubling of a point and the addition of two different points. As noted in the elliptic curve addition algorithm, there are slightly different operations for each. One of the simplest ways to compute a multiple of a point is to use the "double and add" algorithm:

Input: An elliptic curve E, a point $P \in E$ and a multiplier, k.
Output: The point $Q = kP$.

1. Initialize $Q = \mathcal{O}$ and $W = P$.
2. Write out the binary expansion of $k = k_m ... k_1 k_0$.
3. For i from 0 to m do:
   (a) if $k_i = 1$, $Q = Q + W$
   (b) $W = 2W$
4. Output Q

Figure 2 shows a block diagram illustrating this simple method to compute a scalar multiple. The input point P is placed into the variable W, which is repeatedly doubled, producing $2P$, $4P$, $8P$, etc. The input K (as a binary number) is placed in a shift register. The bits are shifted off the low end, and control whether the point-addition box is active. When the shifted bit from K is 0, the point-add is inactive. When the shifted bit from K is 1, the point-add is active, and the point in W is added to the running sum Q. Q holds the running sum of selected points from W. Q is initialized to $\mathcal{O}$. When all bits of K are shifted out of the register, the final value of Q is the answer, $Q = KP$.

6 The Reciprocal Sharing Trick

The main result of our elliptic curve work on this project is the development of an algorithm to combine points using a Reciprocal Sharing Trick. The idea is to rearrange the steps of the "Double and Add" algorithm presented above. Every time where we would add W to Q, we instead save W in a list. At the end of the modified algorithm, we add all the values from the list together. This, of course, will compute the same value Q as before. The advantage of this new method is that we can be more efficient about adding together multiple elliptic curve points if we organize the addition as a binary tree. The trick gives an improvement when the Reciprocal to Multiply ratio, R/M, exceeds 3. The idea in effect changes the cost of point addition from $2M + R$ to $5M + \epsilon * R$. The method has been documented in a patent application [40].

Assume the points are named $A, B, C, \ldots$. We group the points to be added into pairs, $A + B$, $C + D$, $E + F$, etc. (There may be a point left over; it goes into the next level.) Each of these point additions will need to compute a reciprocal: $1/(x_A + x_B)$ for $A + B$, $1/(x_C + x_D)$ for $C + D$, etc. To improve efficiency, we use a trick originally invented by Peter Montgomery [25], called the Reciprocal Saving Trick (RST). The trick computes a group of N reciprocals simultaneously. It computes one actual reciprocal, and uses $3N - 3$ auxiliary multiplications. The net savings is $(N - 1)(R - 3M)$. The details of the RST trick are explained in Section 6.5. Having computed the reciprocals needed for the point-pair additions, we complete the calculations for each addition. This has reduced the number of quantities we need to combine by half. We repeat the pairing procedure, combining point pairs to make point quads, and keep combining pairs until all points have been combined into one total. This gives the required point sum, $Q = kP$. We've used about log_2(number of points to sum) reciprocals, and the remaining reciprocals have been replaced by multiplies (3 each).

Figure 3 shows a block diagram illustrating an example of a method to compute a scalar multiple of an elliptic curve point using this RST idea. The inputs P and K are the same as in Figure 2, as is the variable W and the shift register that holds K. The variable W is repeatedly doubled, as in Figure 2. The bits of K determine which values of W are stored in a list of saved points. When all bits of K are shifted out of the register, the list of saved points is complete. The points in the list are summed in a binary tree. The list points are paired. (Note: Figure 3 shows an odd number of points in the list, so the leftover point is forwarded to the next level of the summing tree.) Each point pair is added using the Reciprocal Saving Trick, which is described in Section 6.5. Each column of octagons represents one level of the binary summing tree. Each level groups all of its reciprocals for sharing; this is represented by the line labeled "RST" passing vertically through the octagons. Each level has one-half as many points as the previous level. The final octagon has nothing to share reciprocals with, so it has no RST line. Only one actual reciprocal is computed for each level of the summing tree (according to the RST algorithm).

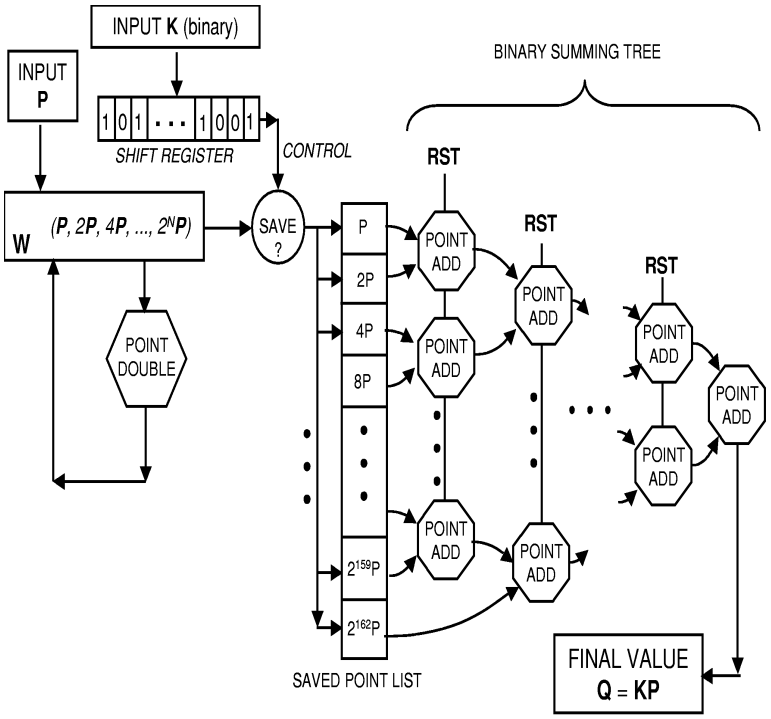

Figure 3. Double and Add with Reciprocal Savings Trick

In other words, to speed up the simple algorithm illustrated in Figure 2, instead of keeping a running sum for the result, we keep a list of all the points we need to add. Then, we simply pair off the points into a binary tree structure, so that we can compute several reciprocals simultaneously when we do the additions. Note that we still have to perform all the point doublings sequentially, se we can't apply the reciprocal saving trick to speed up that step.

There are a number of variations for using this method besides just that given in the description of Figure 3. In one variation, a more complex control based on K assigns coefficients of $+1$ or -1 to each list value. The points can be negated as they are placed in the list; or a "sign bit" can be kept for each list point, and used in the first level of point additions. In another variation, point halving (Section 4) could be used in place of doubling. The bits of K are used in reverse order. Yet another variation is to save all of the values of W in the list, and the bits of K exercise their control function later, selecting which of the list values are forwarded into the summing tree. When P is known in advance, the list can be precomputed. When a Koblitz curve is used [16], and K is expressed in radix τ (instead of binary), the list can be virtual. Each list point has X and Y coordinates that are 1-bit rotations of the previous point.

6.1 Special Cases

Special handling is needed in some rare situations: When either point of a pair $A+B$ to be combined is $\mathcal{O}$, or when two equal points must be combined, or when a point must be combined with its negative. $\mathcal{O}$ points can simply be dropped from the calculation, or excluded when the original list of points to be combined is generated. If the designer decides to retain them, then it may be useful to use $1/1$ as a place holder for the required reciprocal. The cases of equal points and negative points are detected when either the X coordinates are equal, or the denominator $x_A + x_B$ (or $x_A - x_B$ in other fields) is 0. For the negative points case, the point pair can simply be discarded, and $\mathcal{O}$ used as a place holder if the algorithm requires one. $1/1$ can be used as a place holder in the reciprocal algorithm when needed. For doubling, the doubling formula requires a different reciprocal ($1/x_A$ for $GF(2^N)$ fields), which is used in place of $1/(x_A + x_B)$ in the reciprocal saving calculation. In some systems the designer may know that some of these special cases are impossible, and can omit the special handling provisions.

Other fields than $GF(2^N)$ use similar but different elliptic curve equations, and the addition and doubling formulas are similar. The reciprocals required are usually for $x_A - x_B$ instead of $x_A + x_B$.

Figure 4 shows the arithmetic used for conventional point addition. Figure 5 shows an example of a method for point addition using RST. Most of the arithmetic is the same as in Fig. 4, except for the vertical band down the middle. The denominator $M_{den} = x_S + x_T$ is computed just as in Fig. 4. The reciprocal part is different, however. In Fig. 4 the reciprocal M_{rec} is computed as $1/M_{den}$. In Fig. 5, all the M_{den} values in a sharing group (one column of the summing tree in Fig. 3 with the "RST" line drawn through) are used as inputs to the Montgomery Reciprocal Saving Trick. The output of the RST is the set of reciprocals, the required Mden values. The rest of the point addition process is the same as in Fig. 4.

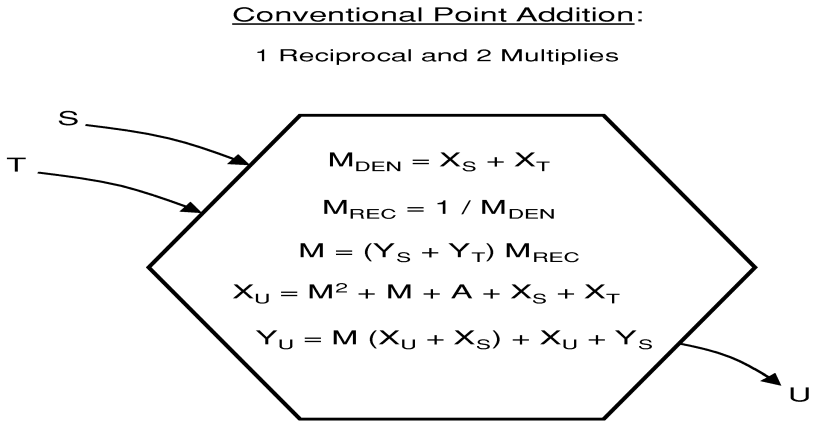

Figure 4. Conventional Point Addition

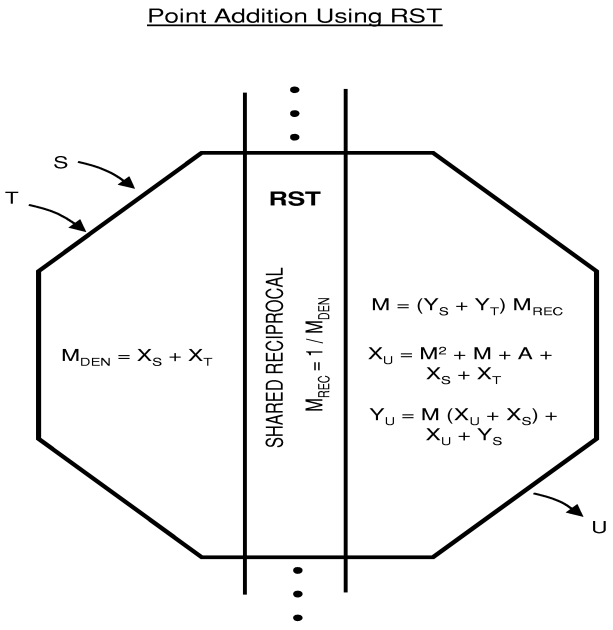

Figure 5. Point Addition with RST

6.2 Timing Results

The RST has been programmed on a PC in the C language. A typical test shows a 35% speed improvement. In the table below, the results are average time in seconds, EC:AddSeveralPoints refers to the present invention, and the computations were done on an elliptic curve over the finite field $GF(2^{233})$ with field polynomial $f(x) = 1+x^{74}+x^{233}$. The results show that the more points you have to add, the better the speed-up you get (as would be expected), and that when a computation involves about 100 points, the speed-up is better than 30%

$Elts.inArray$	$EC : AddPoints$	$EC : AddSeveralPoints$
25	0.032366167	0.025236667
50	0.066622	0.050608667
75	0.099631	0.0747485
100	0.122037667	0.082580833

6.3 More Applications

The idea of reciprocal sharing applies to many algorithms. For example it can be used in conjunction with almost all of the ideas referenced in Section 3. See [40] for more details.

6.4 Protection against Side Channel Attacks

One benefit of this RST idea is that in hardware it helps protect against side-channel attacks against the secret scalar multiplier K. With the normal algorithm for scalar multiplication, an attacker can often carefully observe the device and determine whether it is executing a point addition or a point doubling. The pattern of doublings interleaved with some additions can be used to recover the bits of K. The anual CHES conferences (Cryptographic Hardware and Embedded Systems) have many papers on more sophisticated versions of this attack. The present invention reorders the point additions to come after the point doublings. All the attacker learns from his observations is the number of doublings and additions, rather than the interleaved pattern. (Even this can be concealed by always doing the maximum number of doublings and additions, but discarding some of the values.) We might design the chip to always write the doubling point $2^N P$ to the list memory, even when the point will not be used in the final sum, and selectively advance the list pointer when we want to retain a point for the final sum. The idea of reordering additions and doublings to protect against side channel attacks can be used independently of whether reciprocal saving is being used. The reordering can be random or pseudorandom, and need not put all the additions last; simply saving a few points to be added and doing point additions as a sporadic operation will offer some protection.

6.5 Montgomery's Reciprocal Sharing

The Reciprocal Saving Trick (RST) is a trick (i.e., algorithm) invented by Peter Montgomery [25]. When you need to compute lots of reciprocals (in a mathematical field or ring), you can replace all

but one of the reciprocals with three field multiplications each. Hence, a calculation of n reciprocals can be replaced with a calculation of 1 reciprocal and $3(n-1)$ multiplications. Here's the formula underlying the trick. We need reciprocals $1/U$ and $1/V$. Instead of computing them directly, we compute $1/(UV)$. Then we compute $1/U$ as $V*(1/(UV))$, and $1/V$ as $U*(1/(UV))$. We've saved one Reciprocal, at a cost of three Multiplications: $U*V, V*(1/(UV))$, and $U*(1/(UV))$. If our system has a R/M ratio greater than 3, then this substitution is a win. In other words, if a reciprocal takes more work than 3 multiplications, this saves us time when we have a lot of reciprocals to compute.

The reciprocal saving trick can be extended to computing several reciprocals. The extension can be done either as a straight recursion, or as a linear algorithm. As an example of the linear case for three reciprocals $1/U, 1/V, 1/W$: Compute $UV, UVW, 1/(UVW), UV*(1/(UVW)) = 1/W, W*(1/(UVW)) = 1/(UV), V*(1/(UV)) = 1/U, U*(1/(UV)) = 1/V$. The cost here is six multiplications and one reciprocal, giving a cost of three multiplications for each reciprocal saved.

The pattern is the same for more reciprocals. Compute the product (UVW) of all the numbers to reciprocate; reciprocate that product $(1/(UVW))$; multiply the reciprocal by the two factors that went into the final product (UV and W) to create the reciprocals of those factors (reversed) $(UV*1/(UVW) = 1/W$, and $W*1/(UVW) = 1/(UV))$. Reciprocals of single terms $(1/W)$ are wanted answers, and the reciprocals of compound terms $(1/(UV))$ are further multiplied by the factors of their denominator (U and V) to split them up again as deeply as necessary. Although the grand product can be created by multiplying one factor at a time, $U*V \rightarrow UV, UV*W \rightarrow UVW$, this is not a requirement, and other orders may be preferable. For example, on a parallel processor we might begin to compute four reciprocals U, V, W, X by computing the product $UVWX$ as $(U*V)*(W*X)$, so that the multiplications $U*V$ and $W*X$ can execute in parallel, and the individual reciprocals can be unwound partly in parallel.

Other considerations that may influence the choice of computing order are the schedule of availability of the factors, arrangement of memory use, and simplicity of the algorithm (linear is easier to design).

6.6 Other Point Addition Considerations

During the course of this research we explored other methods to more efficiently combine elliptic curve points. Although none were as successful as the reciprocal savings trick, the ideas are worth mentioning and perhaps when combined with a new idea will lead to improved algorithms.

One avenue that we explored was combining several elliptic curve points in one operation. The scalar multiplication operation combines around a hundred points with addition or subtraction, and does about two hundred point doubling (or halving, or tripling, etc.) operations. We examined several ways of combining these operations in groups to look for faster computations.

We also looked at adding three unrelated points $(A+B+C)$ in one step. The mathematical approach is to compute the formula for the sum in two steps: $A+B$, then $(A+B)+C$. The triple sum formula is then simplified algebraicly, and we look for ways to compute it with the minimum number of multiplications and reciprocals. If we can do this with less total work than the direct two point additions ($A+B$ and $(A+B)+C$), this will lead to an efficiency improvement.

A simple analogy, using circular functions instead of elliptic curves is given by the tangent function. Suppose we are given $\tan A$ and $\tan B$, and need to compute $\tan(A+B)$. The formula

for the tangent of the sum of two angles A and B is

$$\tan(A+B) = \frac{\tan A + \tan B}{1 - \tan A \tan B}$$

If we assume free addition, the cost of tangent addition is two multiplications and one reciprocal, $2M + R$.

The triple addition formula is

$$\tan(A+B+C) = \frac{\tan A + \tan B + \tan C - \tan A \tan B \tan C}{1 - \tan A \tan B - \tan A \tan C - \tan B \tan C}.$$

This can be computed with four multiplications and one reciprocal, $4M + R$. The cost of two plain additions, $A + B$ and then $(A + B) + C$, is $(2M + R) + (2M + R) = 4M + 2R$, so direct triple addition saves $1R$, one reciprocal.

Earlier work in the field showed that direct-formula multiplication of a point by 8, 16, or 32 is sometimes better than repeated doubling. The direct formulas are very complicated, and use more multiplications and fewer reciprocals than repeated doubling. Whether they are advantageous depends on the relative costs of multiplication and reciprocal, the ratio R/M.

We examined triple sum $(A+B+C)$, double-and-add $(2A+B)$, and triplication $(3A)$. Of these, only triplication provided a slight improvement. To use triplication effectively, the scalar multiplier must be represented in balanced ternary (base 3) notation, with digits 0, 1, and -1. The number of digits is less than binary by a factor of $log_2(3) = 1.58$, but 0 digits are less likely. The average number of non-zero digits (2/3 * 1/1.58)*(length of scalar multiplier) is slightly smaller with ternary than binary (1/2)*(length), but the comparison goes the other way when blocks of 0s are taken into account. This means that the ternary scheme will need more addition/subtraction steps than binary when the natural signed-sliding-window method is used. These extra adds defeat the small advantage of triplication. (Since the RST patent idea above cheapens addition, it may make this idea work.) One place where triplication has newly recognized importance is in $GF(3^N)$ fields. These are being used in new identity-based encryption schemes [2], and some $GF(3^N)$ curves have very efficient triplication formulas. In these schemes, it will make sense to use the balanced ternary representation of the scalar multiplier, since triplication is cheap.

After we did this work, Peter Montgomery found a small improvement in computing the combination $2A+B$ (see [6]). He used the novel scheme $(A+B)+A$. (We used the more natural approach of first computing $2A$, followed by adding B. Montogomery's approach is unnatural because it uses two point additions, and point addition is generally a little more expensive than doubling.) His scheme allows one multiplication to be eliminated from the calculation, by not computing the Y-coordinate of the intermediate value $A + B$. The net saving is a few percent. One advantage of Montgomery's result is that it works with fields of any characteristic. We note that this work is largely trumped by the RST patent idea, which cheapens point addition for large groups of points.

7 Finite Field Arithmetic and Field Towers

Underlying all elliptic curve arithmetic is the finite field arithmetic. Both $GF(p)$ with p a large prime and $GF(2^N)$ fields are commonly used in elliptic curve cryptography. A careful choice of field

may speed up computations. For example choosing a *Mersenne* prime field will help with $GF(p)$ computations [43]. We discuss some improvements for $GF(2^N)$ fields here.

The finite field

$$GF(2^m) \cong GF(2)[x]/f(x) = \{a_0 + a_1 x + \cdots + a_{m-1}x^{m-1} (\text{mod } f(x)) \mid a_i \in GF(2)\}$$

where $f(x)$ is an irreducible binary polynomial of degree m. An element $a \in GF(2^m)$ can therefore be represented as an $m-$tuple $a = (a_0, a_1, ..., a_{m-1})$ of zeros and ones. Addition of two elements is a bitwise exclusive-OR (XOR) operation:

$$a, b \in GF(2^m),\ a + b = (a_0 \oplus b_0, a_1 \oplus b_1, ..., a_{m-1} \oplus b_{m-1})$$

and multiplication is like a plain multiplication without any carries but with the XOR accumulation only. The result of the multiplication must, however, be reduced by the field polynomial $f(x)$. As the degree m of the field gets large, the multiplication can become time-consuming and the representation of the numbers can become cumbersome. For a general reference on finite field arithmetic, see [13].

There are a number of ways to speed up computations in $GF(2^N)$ fields. One way is to choose a field where the polynomial $f(x)$ can be represented by a trinomial to make the reduction step in the multiplication easier. The representation of the field elements as described above uses a *polynomial* basis. A *normal basis* representation is sometimes used instead. Squaring and square root in a normal basis representation is almost free (it's just a rotation of the bits). See [13] for more details.

7.1 Field Towers

If m is a composite number, we can use field towers to speed-up the computations. Suppose $m = ns$. Then we can think of $GF(2^m) = GF((2^n)^s)$ as a degree s extension of $GF(2^n)$. The elements are $a \in GF(2^m), a = (\alpha_0, \alpha_1, ..., \alpha_{s-1})$, where $\alpha_i \in GF(2^n)$. For example, suppose $m = 156 = 12 * 13$. Then we can represent $GF(2^{156})$ as $GF((2^{13})^{12})$. The addition and multiplication of two elements $\alpha = (a_0, ..., a_{11})$ and $\beta = (b_0, ..., b_{11}), a_i, b_j \in GF(2^{13})$ uses the underlying $GF(2^{13})$ arithmetic which is much simpler than the arithmetic in $GF(2^{156})$. If m is highly composite, there are several possibilities for writing $GF(2^m)$ as an extension of a smaller field. We may even write $GF(2^m)$ as a series of field extensions. For instance, $GF(2^{156}) = GF(((2^3)^4)^{13})$, or $GF(2^{156}) = GF(((2^6)^2)^{13})$, etc. We can speed up the arithmetic considerably in such fields. Thinking of $GF(2^{156})$ as a three-level tower would make the corresponding arithmetic about three times faster than the arithmetic in the field, using a polynomial basis. We note that one must be careful if using field towers as there are some choices where security is compromised (see [46]).

7.2 Quadratic Solve

If a hardware chip is being developed to carry out fast elliptic curve computations, it can be the case that there are special implementations of finite field algorithms that give savings. The algorithms may be optimized differently depending on which field is being used. As an example of an efficient implementation of an elliptic curve signature for hardware we cite [39]. That implementation used the following special circuit we developed for solving a quadratic equation as needed in the point halving algorithm (Step 1 in Alg.2).

The circuit has a relatively small number of XOR gates (387) and depth (35). The full circuit and detailed derivation are in [38]. The algorithm is specific to the field $GF(2^{89})$, but the idea could work to create optimal Qsolve circuits for other fields.

Alg. 3. *Qsolve*

Input: $a = (a_{00}, ..., a_{88}) \in GF(2^{89})$
Output: $z = (z_{00}, ..., z_{88}) \in GF(2^{89})$ *where* $a = z^2 + z$

Except for odd z in the range $z_{01} - z_{19}$ (which are computed directly), the bits of z are computed from the following equations:

a_{even} bits:

$$\begin{aligned} a_{00} - a_{36} : a_{2n} &= z_{2n} \oplus z_n \oplus z_{n+70} \\ a_{38} - a_{74} : a_{2n} &= z_{2n} \oplus z_n \oplus z_{n+51} \\ a_{76} - a_{88} : a_{2n} &= z_{2n} \oplus z_n \end{aligned}$$

a_{odd} bits:

$$\begin{aligned} a_{01} - a_{37} : a_{2n+1} &= z_{2n+1} \oplus z_{n+45} \\ a_{39} - a_{87} : a_{2n+1} &= z_{2n+1} \oplus z_{n+45} \oplus z_{n+26} \end{aligned}$$

This derivation uses several observations to reduce the number of gates.

1. QSolve is linear, so we could precompute QSolve(u^N) for each N. The runtime circuit XORs together the appropriate subset for a general polynomial (see [29] for one method of doing the precomputation). This is fast, but uses a lot of gates. We traded speed for size, getting a slower but smaller circuit.
2. We reduced the number of required QSolve(u^N) values by removing some powers of u from the problem. For example, the substitution QSolve(u^{2N}) $\Rightarrow u^N + QSolve(u^N)$ eliminates even powers of u. The substitution $u^N \Rightarrow u^{N-38} + u^{N+51}$ removes some odd powers of u. After repeated substitutions like these, QSolve(u^N) is only needed for odd N in the range 1...19.
3. Only some of the answer bits are required: z_{odd} in the range $z_{01}...z_{19}$. This reduces the number of gates considerably. The remaining bits can be recovered by solving the bit equations for QSolve. For example, we compute z_{45} from the equation $a_{01} = z_{01} \oplus z_{45}$.
4. We assume that a_{00} is equal to a_{51}. The actual value of a_{00} is ignored. Furthermore, z_{00} is irrelevant, and is set equal to 0.

Our minimal size QSolve circuit used only 287 XOR gates, but had depth 65. We moved back from this extreme point on the speed-size tradeoff curve to a circuit with 387 XOR gates and depth 35.

8 Available Software

As part of this project many efficient elliptic curve algorithms were implemented in C, including the ones described in this report. Contact the authors for more information on obtaining and using the code.

9 Dilogarithms

The fact that elliptic curve groups can be used to speed up cryptographic computations caused us to consider if there were other groups on which to base security that would be usable in cryptographic applications. This section reports on a generalization of the discrete logarithm, called the Modular Dilogarithm. The modular dilogarithm is a discrete analog of the Dilogarithm, a special function defined over the complex variables. The modular dilog satisfies the same functional equations as the complex dilog, including some equations involving discrete logarithms. The modular dilog can be defined modulo a prime, or a prime power, or in a finite field. For modular dilogs defined modulo a prime P, the range of values seems to be modulo $P^2 - 1$. The modular dilog is harder to compute than the discrete log. Modular Trilogarithms may also exist.

First, in Section 9.1 we give a brief summary of the properties of the classical dilogarithm defined on the complex numbers. In Section 9.2 we give an example of the Modular Dilogarithm, and show which of the properties of the complex function carry over to the modular case. Then in Section 9.3 we explain how we compute modular dilogs. In Section 9.4 we discuss extensions to trilogs and some other directions. Then in Section 9.5 we pose some questions that have been suggested by this discovery . The most important is "Why do these exist at all?" Finally in Section 9.6 we discuss the prospects of this discovery. We note that we have no proofs. All the work is empirical.

9.1 The Classical Dilogarithm Function

Most of this material is drawn from Lewin [19]. The complex dilogarithm $Li_2(z)$ may be defined by the power series

$$Li_2(z) = \sum_{n=1}^{\infty} \frac{z^n}{n^2}$$

or by the integral

$$Li_2(z) = \int_{t=0}^{z} -\log(1-t)/t \; dt$$

The equivalence of the definitions is established by term-by-term integrating the power series for $-\log(1-t)/t$.

The power series converges within (and on) the unit circle $|z| \leq 1$. The Riemann sheet structure is complicated: The integrand is defined everywhere except $t = 0$ and 1, and the singularity at 0 is removable on the principal sheet. The singularity at 1 is a log-spiral-staircase, and the non-principal sheets have simple poles at 0. The principal sheet is the complex plane with a branch cut along the positive real axis for $x > 1$.

The dilogarithm function was apparently first considered by Leibniz, shortly after calculus was discovered. It was later studied by Euler and Landen. Euler found that

$$Li_2(1) = \zeta(2) = \frac{\pi^2}{6}$$

Several simple functional equations were discovered:

$$Li_2(z) + Li_2(-z) = \frac{Li_2(z^2)}{2}$$

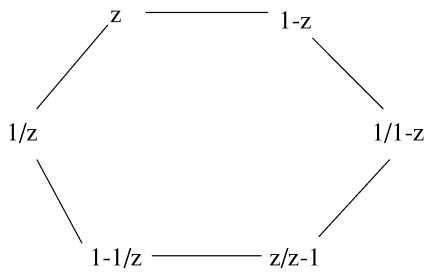

Figure 6. Dilogarithm Relations

This is obvious from adding up the power series for $Li_2(z)$ and $Li_2(-z)$. The z^{odd} terms cancel. The z^{even} terms are $Li_2(z^2)/2$. A generalization of this trick gives

$$Li_2(z) + Li_2(wz) + Li_2(w^2z) = \frac{Li_2(z^3)}{3}$$

where $w = exp(2\pi i/3)$ is a complex cube-root of 1. Analogous results hold for higher roots of 1.

The next two functional equations are established by differentiation.

$$Li_2(z) + Li_2(1-z) = \frac{\pi^2}{6} - \log(z)\log(1-z)$$

For complex z, the branches of the logarithms are chosen by walking from $z = 1/2$.

$$Li_2(z) + Li_2(1/z) = -\frac{\pi^2}{6} - \frac{(\log(-z))^2}{2}$$

For complex or positive z, the branch of the logarithm is selected by walking from $z = -1$

These functional equations are enough to compute $Li_2(z)$ anywhere in the complex plane (principal branch), by bringing z inside the unit circle and using the power series. Our work generally ignores branches, so we don't include the twiddle terms necessary for non-principal branches.

The transformations $z \to 1/z$ and $z \to 1-z$ can be combined to generate a set of six related quantities whose dilogarithms differ by (or sum to) logarithmic terms: $z, 1/z, 1-z, 1-1/z, 1/(1-z), z/(z-1)$. They can be arranged in a hexagon with alternate edges representing the two transformations (See Figure 6).

For example, we can set $z = 1/3$, and relate the value of Li_2 at $1/3$ to those at $2/3, 3/2, -1/2, -2$, and 3. Manipulation of the functional equations allows closed-form values to be determined for Li_2

at $z = 1, -1, 1/2$, and $1/\phi$ and $1/\phi^2$, where $\phi = 1.618...$ is the golden ratio.

$$\begin{aligned} Li_2(1) &= \pi^2/6 \\ Li_2(-1) &= -\pi^2/12 \\ Li_2(1/2) &= \pi^2/12 - \log(2)^2/2 \\ Li_2(1/\phi) &= \pi^2/10 - \log(\phi)^2 \\ Li_2(1/\phi^2) &= \pi^2/15 - \log(\phi)^2 \end{aligned}$$

In 1809, Spence made a major advance, a true leap of imagination. The dilogarithm is sometimes called Spence's function in his honor. He found a relationship equivalent to

$$\begin{aligned} Li_2(xy) = \quad & Li_2(x) \quad + Li_2(y) + Li_2((xy - x)/(1 - x)) \\ & + \quad Li_2((xy - y)/(1 - y)) + 1/2(\log((1 - x)/(1 - y)))^2 \quad [5term] \end{aligned}$$

valid when $x, y, xy < 1$, or if all the dilog arguments are inside the unit circle. Slight variations apply outside this range. This particular form of the functional equation is due to Hill. The equation is called 5term since it contains 5 dilogarithm terms.

Subsequently, many authors have studied the dilogarithm and related functions, and produced an abundance (nay, a plethora) of functional equations. None can really be called simple, in contrast to the simple relationship for logarithms. Lewin [18] contains a good summary of pre-1960 work, including a good bibliography. [19] is an updated second edition. Lewin has edited a sampler of recent work [20].

Other dilog functional equations have more terms and/or more variables, both with and without side conditions. It has been established that all can be derived from the basic five-term functional equation.

Considerable effort has gone into looking for functional equations without log terms, and into defining closely related functions without the singularities, or whose functional equations don't have the log terms. For example, the contributors to [20] define the Bloch-Wigner Dilogarithm, and several versions of the Rogers Dilogarithm. Each formulation has its advantages and disadvantages; there's no clearly superior choice. A typical sacrifice is the analyticity of the new function. Like a wrinkle in a rug, the trouble spots can be moved around but not eliminated.

The work described here began with Newman's six-term symmetric functional equation for the co-dilogarithm, $cLi_2(x) = Li_2(1 - x)$. This equation has the nice property of containing no extra logarithm terms.

$$2[cLi_2(x) + cLi_2(y) + cLi_2(z)] = cLi_2(xy) + cLi_2(xz) + cLi_2(yz) \quad [6term]$$

provided that $x + y + z = xyz + 2$, or equivalently, $1/(1 - x) + 1/(1 - y) + 1/(1 - z) = 1$. [This equation is also true for plain old logarithms, without any relationship among x, y, z except $xyz \neq 0$. Any multiple of log can be added to a solution of the functional equation to get another solution valid on the nonzero subdomain.]

There doesn't seem to be an algebraic way to go from the six-term equation to the five-term equation.

9.2 Modular Dilogarithms

We have invented/discovered the Modular Dilogarithm function, $D()$. The argument of D is an integer N modulo a prime P. The value is an integer $D(N) \pmod{P^2-1}$.

Example with $P = 19$:

N	0	1	2	3	4	5	6	7	8	9	10	11	12
$log(N)$	–	0	1	13	2	16	14	6	3	8	17	12	15
$D(N)$	120	30	345	358	74	26	344	258	327	108	265	162	3
$E(N)$	0	0	0	352	56	224	56	192	288	312	160	168	72

N	13	14	15	16	17	18	(mod 19)
$log(N)$	5	7	11	4	10	9	(mod 18)
$D(N)$	132	326	44	236	232	345	(mod 360)
$E(N)$	48	344	296	344	208	0	(mod 360)

D satisfies five functional equations, all modulo 360:

$$D(x) + D(1-x) = D(0) + D(1) + K\log(x)\log(1-x)$$

$$2[D(x) + D(1/x)] = -2D(1) + K\log(-x)^2 \quad x \neq 0$$

$$2[D(x) + D(-x)] = D(x^2)$$

$$\begin{aligned} D(xy) = \quad & D(x) \quad + D(y) - D((x-xy)/(1-xy)) - D((y-xy)/(1-xy)) \\ + \quad & D(0) + K\log((1-x)/(1-xy))\log((1-y)/(1-xy)) \quad xy \neq 1 \end{aligned}$$

$$\begin{aligned} 2[D(1-x) + D(1-y) \quad + \quad & D(1-z)] = D(1-xy) + D(1-xz) + D(1-yz) \\ with\ z \quad = \quad & (2-x-y)/(1-xy) \quad xy \neq 1 \end{aligned}$$

Here $K = 20$. The logs are base 2. The arithmetic for arguments of log and D is done modulo 19. The value for log(0) doesn't matter: it's always multiplied by $\log(1) = 0$. Any multiple of D also works, with K adjusted appropriately. $E()$ is another solution for the functional equations, with $K = -40$. $3E = 192D \pmod{360}$. Any linear combination $dD + eE$ works, with $K = 20d - 40e$. There are 1080 solutions: 360 multiples of D, plus $0, 1, 2$ times E. $2D + E$ gives a "Modular Rogers Dilogarithm" without log terms ($K = 0$).

The Modular Dilogarithm satisfies the same functional equations that the regular dilogarithm does, with a few adjustments. Log terms are interpreted as discrete logs. Since discrete logs are mod $P-1$, the log terms must be multiplied by $P+1$ to fit into functional equations mod P^2-1. The base of the logs is unspecified, so the log terms may need an additional scaling factor. We replace $\pi^2/6$ with $D(1)$, and allow the possibility that $D(0)$ is nonzero. $D(\infty)$ is left undefined. All the functional equations of dilogs are simple sums of dilog terms, so any multiple of dilog will also satisfy the equations if other terms are scaled to match.

Some properties of the solution space: $D(x^2)$ is divisible by 2, so quadratic residues have even dilogs. For $P = 19$, $D(x)$ is odd exactly when both x and $1-x$ are non-residues. Dilogs of cubic residues are divisible by 3.

9.3 Computing Modular Dilogarithms

Individual values of Modular Dilogarithms don't have meaning; a particular value is not right or wrong. We must compute the entire function – the value at each residue mod P – to have something to check.

Our approach is to assign an independent unknown variable to represent each function value. We substitute numbers into the functional equations, and learn relationships between the function values.

For example, suppose $P = 7$. We would use 7 variables, $D0, D1, ..., D6$. The functional equation $2[D(x) + D(-x)] = D(x^2)$ would be specialized with $x = 0...6$. $X = 2$ would give the relationship

$$2D2 + 2D5 = D4.$$

Since all the functional equations are linear in the dilog terms, we use matrices to keep track of our work. We create a non-square matrix, with P columns and many rows. Column J corresponds to the variable $D(J)$. The matrix entries are integers, representing coefficients of $D(J)$. The rows are initialized from substitutions into the functional equations. Each row represents one relationship between the D values, saying "These D values, with these coefficients, sum to 0." Most of the coefficients in the row are 0, since there are only a few terms in the functional equations. If the functional equation has fractions, we clear them, to avoid non-integer entries in the matrix. When substitution into a functional equation gives an infinity in some argument, we discard the instance, and don't make a matrix row. When a functional equation has numerical or log terms, we use an extra column of the matrix to hold the values. Log terms are evaluated as discrete logs (using a fixed primitive root mod P), with values defined mod $P - 1$. Our matrix solution method avoids combining entries in different columns.

Most of our work uses the co-dilogarithm, which has a nice 6-term functional equation with no log terms or constants. This allows agnosticism on the base of the logarithms.

An example of the matrix for $P = 7$, using codilogs. Column J is $cD(J) = D(1 - J)$.

$$\begin{array}{ccccccc}
1 & 0 & 2 & 0 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 2 & -1 & 0 & 2 \\
0 & 0 & 0 & 0 & 2 & 2 & -1 \\
0 & 0 & 4 & 2 & -1 & 0 & -2 \\
0 & 0 & 1 & -1 & 0 & 1 & 2 \\
0 & 0 & -1 & 4 & 2 & -2 & 0 \\
0 & 0 & -1 & -2 & 4 & 0 & 2 \\
0 & 0 & 0 & 0 & -3 & 6 & 0
\end{array}$$

The first row is $cD(0) + 2cD(2) = 0$. This corresponds to $x = 0, y = 0, z = 2$ in the 6-term functional equation:

$$2[cD(0) + cD(0) + cD(2)] = cD(0) + cD(0) + cD(0)$$

For larger P, the number of rows is about $P^2/6$. (Most of the rows turn out to be redundant and could be dropped or never generated. Probably any set of somewhat more than P rows is adequate.)

The matrix entries are regarded as integers, rather than modular residues. This postpones the choice of modulus. The matrix is "solved" by elementary row operations, adding or subtracting multiples of one row to or from another row. The algorithm diagonalizes the matrix as best it can, although there are some remaining non-diagonal elements. Redundant rows are simplified to 0, and dropped. We avoid division of a row in solving the matrix, to avoid losing potential modular information.

After the row-reduction, the example matrix reduces to

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 11 \\ 0 & 0 & 0 & 1 & 0 & -1 & 9 \\ 0 & 0 & 0 & 0 & 1 & -2 & -8 \\ 0 & 0 & 0 & 0 & 0 & 6 & -9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 24 \end{matrix}$$

Some all zero rows have been dropped. The bottom two rows correspond to the equations $6cD(5) - 9cD(6) = 0$ and $24cD(6) = 0$. We want integer solutions, so the modulus for the dilogarithms must be a multiple of 48. The other rows are all compatible with 48. Using the bottom row, we assign $cD(6) = 2t$. The next row, $6cD(5) - 9cD(6) = 0$, gives $cD(5) = 3t + 8u$. The other cD values can be computed from the matrix rows. $cD(1)$ requires one new parameter; $cD(1) = 16v$. Dilog values are then computed from $D(x) = cD(1 - x)$. The co-dilog functional equation has weight 3, so each matrix row also has weight 3. The weights of rows in the solution matrix are multiples of 3. The other dilogarithm functional equations may impose additional conditions on the variables.

We have solved coDilog matrices for primes 5...23. Attempts to use several single variable functional equations instead of the two-variable co-dilog equations have not worked out.

Computational Complexity
Computing a dilog table for $P = 1009$ would be a major effort with present methods; in contrast, the discrete log table can be computed in a millisecond.

Our current matrix approach requires $O(P^4)$ arithmetic operations to solve the dilog matrix. This would drop to $O(P^3)$ if we only used $O(P)$ rows. A successful basis approach, with a basis size of perhaps $\sqrt{P}$ values, would bring the complexity down to $O(P^{1.5})$. Checking that a proposed dilog table satisfies a two-variable functional equation is $O(P^2)$, unless we develop new theorems that allow checking only a subset of the combinations. These are all much larger than the corresponding effort for discrete logs, for which naive algorithms are $O(P)$, and good algorithms begin at $O(\sqrt{P})$ and drop to $O(P^\epsilon)$.

Possibilities for Extension to Larger P

Ladders
The matrix method is limited to modest P values, but we have some ideas for larger P. There are dozens of "polylogarithm ladders" known, which give relationships between Li values of various algebraic numbers. The relationships involve both dilogarithms and higher polylogs. A simple

example is

$$Li_2(\phi^{-6}) = 4Li_2(\phi^{-3}) + 3Li_2(\phi^{-2}) - 6Li_2(\phi^{-1}) + 7\pi^2/30$$

with $\phi = 1.618...$, the golden ratio. If P is a prime ending in 1 or 9, then 5 is a square (mod P), and there is a residue corresponding to ϕ, a root of the equation $\phi^2 = \phi + 1$. The example ladder above may translate to a "free" modular dilog relation.

A large part of [20] is devoted to ladder relationships. These ladders could map to residues mod P, and determine relationships among modular dilogs. The most extensive example known [1] uses a root W of the Lehmer polynomial, $W^{10} + W^9 - W^7 - W^6 - W^5 - W^4 - W^3 + W + 1$, and goes up to $Li17$, involves a smattering of powers up to W^{630}, and coefficients with hundreds of digits.

Symbolic Values

Another possible approach is related to the matrix approach, with "opportunism". The idea is to assume a few symbolic values for particular dilogs, and use the functional equations to determine related dilogs. When all possible dilogs have been derived, another symbolic value is assumed. Eventually, every dilog is assigned a value based on the assumed symbolic values. Then the functional equations can be enumerated systematically, and the symbolic values checked. Occasionally a new relationship among symbolic values will be learned, and a symbolic value is eliminated from the system.

Example: Mod 101, starting with symbolic values for $D5$ and $D8$. We can use the following "rules of inference". If we know $D(x)$ we can express $D(y)$:

$$\begin{aligned} x &\rightarrow 1/x \text{ and } 1-x \text{ and the rest of the 6-ring} \\ x \text{ and } -x &\rightarrow x^2 \\ x \text{ and } x^2 &\rightarrow -x \end{aligned}$$

We begin with $D(5) = D5$ and $D(8) = D8$. We also assume $D(0) = D0$ and $D(1) = D1$ are known, and we assume a multiplier K and base 2 for the discrete logs which appear. 2 is a primitive root mod 101. (We could also use one of the dilog variants which doesn't need log terms in its functional equations.) Each residue is in a ring of 6 values. We proceed according to the following plan:

$$\begin{aligned} 5 &\rightarrow -20, -4, 25, -24, 21 \\ 5, 25 &\rightarrow -5 \\ -5 &\rightarrow 20, 6, 17, -16, -19 \\ 8 &\rightarrow 38, -7, -29, 30, -37 \\ 8, -37 &\rightarrow -8 \\ -8 &\rightarrow -38, 9, 39, 45, -44 \\ 9, -20 &\rightarrow -9 \\ &etc. \end{aligned}$$

Using the functional equation for $D(5) + D(1/5)$, we find

$$\begin{aligned} D(-20) &= D(1/5) \\ &= (-2D1 + K(log(-5))^2)/2 - D5 \end{aligned}$$

$$
\begin{aligned}
&= -D1 - D5 - (K74^2)/2 + 5100u \\
&= -D1 - D5 - 2738K + 5100u
\end{aligned}
$$

The $5100u$ term arises from dividing an equation mod 10200 by 2. We continue by calculating expressions for $D(-4)$ and so on, until we have every residue mod 101. Whether this approach actually will work is speculative.

9.4 Extensions: Other Moduli, Other Fields, Other Functions

Prime Powers
We've done experiments to see how far the Modular Dilogarithm idea can be extended. A co-dilog matrix was created and solved for Mod 25, as an example of a prime power. The dilogs seem to be defined mod 600. This is analogous to the path from mod 5 to mod 25 for discrete logs, where the solution range (mod 4) is multiplied by 5 to get (mod 20). The $P = 5$ dilog range (mod 24) is multiplied by 25 to get (mod 600).

Composite Moduli
We haven't tried experiments with composite moduli divisible by different primes. The results might be simply the cross product of the separate prime power solutions, or there might be new phenomena.

Finite Fields
Another experiment was with $GF(5^2)$, the finite field of order 25. Here the dilogs appear to be mod 624, which is $5^4 - 1$. There's also a relationship between the dilog of a field element and the dilog of its conjugate. Curiously, the intermediate partially-reduced matrices included integers of more than $100K$ bits. The matrix took $\sim 50K$ row-reduction steps to process, much more than the mod 25 case or the mod 19 case. The final matrix only contained 3 digit numbers, with no sign that the intermediates were huge.

Modular Trilogarithms
We've made some limited progress with Modular Trilogarithms. The single variable functional equations don't give enough relationships. A 22-term trivariate functional equation due to Goncharov [20], p. 375 always produced zero solutions. (Our conversion to modular form might be wrong.) We had some success with a two-variable 10-term equation, eqn 6.93 in [19] p. 174. Lewin derives this from an 1809 result of Spence. Let $u = s + t - st$.

$$
\begin{aligned}
T(u) = 2T(s) \quad &+ \quad 2T(t) + 2T(s/u) + 2T(t/u) + 2T(-st/u) + T(-s/t) + T(-t/s) \\
&- \quad T(s^2/u) - T(t^2/u) - 2T(1) \\
&+ \quad (\pi^2/6 + \log(s/t)^2/2)\log(u^2/st) - (\log(u/s)^3 + \log(u/t)^3)/3
\end{aligned}
$$

The $T(1)$ and π and log terms aren't counted in the name "10-term". We multiplied the non-trilog terms by 6 to clear fractions, and used $\pi^2 => -(p-1)^2/4$. This approach yielded matrices with solutions mod $7(P^3 - 1)$ for $P = 5...17$.

9.5 Miscellaneous Musings

Toward a Formula for Modular Dilogs
Starting with a functional equation from [19] p. 9.

$$\begin{aligned} D(ab) = D(a) &+ D(b) + D((ab-a)/(1-a)) + D((ab-b)/(1-b)) \\ &+ K * 1/2(\log((1-a)/(1-b)))^2 \end{aligned}$$

Fix $a \neq 0, 1$. Sum over $b \neq 1 (mod P)$. Let $S = D(0) + D(1) + ... + D(P-1)$.

$$\begin{aligned} S - D(a) = (P-1)D(a) &+ (S - D(1)) + (S - D(0)) + (S - D(1-a)) \\ &+ K * 1/2 \sum_{i=0}^{p-2} i^2) \end{aligned}$$

$$-PD(a) = 2S - D(0) - D(1) - D(1-a) + K/12(P-2)(P-1)(2P-3)$$

Assume $D(a) + D(1-a) = D(0) + D(1) - K * \log(a) * \log(1-a)$

$$\begin{aligned} -(P+1)Da = 2S &- 2(D(0) + D(1)) + K * \log(a) * \log(1-a) \\ &+ K/12(P-2)(P-1)(2P-3) \end{aligned}$$

If we assume that log terms are multiplied by $P+1$, then this determines $D(a)$ up to approximately mod $P-1$.

Why the Modulus for Modular Dilogarithms is $P^2 - 1$
Consider the squaring formula, $2(D(x) + D(-x)) = D(x^2)$. If we are working over a finite field $GF(2^N)$, then $x = -x$, so $4D(x) = D(x^2)$. N squarings gets us back to $x : x^{2^N} = x$, so $4^N D(x) = D(x^{2^N}) = D(x)$. Whence $(4^N - 1)D(x) = 0$. So we expect dilogs in $GF(2^N)$ to be mod $4^N - 1$. Similarly, over a general finite field $GF(P^E)$, we use the P-tuplication formula

$$P(D(x) + D(wx) + D(w^2x) + D(w^3x) + ... + D(w^{P-1}x)) = D(x^P)$$

with $w^P = 1$. But in a field over (mod P), the only P-th root of 1 is 1, so $w = 1$. Thus $P^2 D(x) = D(x^P)$. Taking P-th powers E times gives $P^{2E} D(x) = D(x^{P^E}) = D(x)$, since $x^{P^E} = x$in a finite field. Thus we expect $(P^{2E} - 1)D(x) = 0$ for all x in the field, and the dilogs "exist" mod $P^{2E} - 1$.

The argument extends to trilogarithms: $P^2(T(x) + T(wx) + ... + T(w^{P-1}x)) = T(x^P)$, with $w^P = 1$, so $P^3 T(x) = T(x^P)$, and then $P^{3E} T(x) = T(x^{P^E}) = T(x)$, and trilogs "exist" mod $P^{3E} - 1$. The argument also works in the other direction, to "explain" why discrete logs are mod $P^E - 1$.

This also explains the conjugation mapping, $P^2 D(x) = D(x^P)$. Taking P-th powers of an element cycles through the conjugates; in the simple case $E = 2, x^P = x \sim$. This explains the structure in the $GF(5^2)$ example; and why the elements in the ground field, which are their own conjugates, satisfy $x^P = x$ and therefore $P^2 D(x) = D(x)$, so they are multiples of $(P^4 - 1)/(P^2 - 1) = P^2 + 1$.

This isn't rigorous, but it might be the beginnings of a proof.

9.6 Prospects

Dilogarithms and polylogarithms are an active area of research. A Net search turned up several different threads, although our modular idea seems to be new.

The most important unanswered question is: **Why Do Modular Dilogarithms Exist?**

All simpler modular operations and functions, up through discrete logs, can be derived from analogies with the integer and real number versions of the functions, and can ultimately be traced back to counting operations. For example, the familiar "Laws of Exponents" are derived by counting the number of times various factors are multiplied together. The Dilogarithm concept has no such grounding: the definition depends on limit operations – integrals and infinite series – which have no modular counterpart. That there are modular solutions of the functional equations can only be regarded as magic at this point.

There are many other questions: Give a proof that Modular Dilogarithms exist for all primes P, and perhaps for prime powers and finite fields. Give a formula to compute them. Describe the structure of the solution space. Is there a useful inverse function, a Modular Diexponential? Are there special primes for which computing dilogs is easy? How far can the concept of Modular Functions be extended?

10 Recurrence Sequences

We contracted with Bill Gosper to explore (the idea of using formulas in recurrence sequences to develop) another idea for faster operations with "points" on systems that are closely related to elliptic and hyperelliptic curves. He developed a number of "point addition" formulas for these systems. His work confirms that these systems could be used as alternatives to elliptic curves in, for example, key exchange. More work is needed to make his discoveries efficient. As it stands, his formulas need more arithmetic than the usual elliptic curve methods, but they may be competitive with hyperelliptic curves (which have much more complex point addition formulas). His formulas may also be speeded up by judicious specialization, careful choices of some system parameter values as 0 or 1 to eliminate some terms from the formulas. Another idea to examine is to use $GF(2^N)$ fields, where squaring is essentially a free operation and terms with even coefficients drop out.

We explored another such system, closely related to classic elliptic function theory over the complex numbers. This scheme also seems not-quite-competitive with the usual elliptic function methods. More details for all of this are given in his final report [11]. Contact the authors for more information.

11 Other Groups

There have been a number of other groups suggested for use in cryptography. For example hyperelliptic curves or braid groups. Neither of these are competitive with elliptic curves, but a new idea using torus groups [30] is competitive and in fact offer shorter signatures.

12 References.

References

[1] D. Bailey and D. Broadhurst, "A Seventeenth-Order Polylogarithm Ladder", http://www.nersc.gov/∼ dhbailey/dhbpapers/ladder.pdf, Oct. 12, 1999.

[2] D. Boneh and M. Franklin, "Identity Based Encryption from the Weil Pairing", SIAM J. of Computing, Vol. 32, No. 3, p. 586-615, 2003.

[3] E. Brickell, D. Gordon, K. McCurley, and D. Wilson, "Fast Exponentiation with Precomputation", Advances in Cryptology – Eurocrypt '92, Springer lNCS 658 p. 400-407, 1993.

[4] H. Cohen, A. Miyaji, T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates", Advances in Cryptology – Asiacrypt '98, Springer LNCS 1514, p. 51-59, 1999.

[5] J.-M. Couveignes, "Computing ℓ-isogenies using the p-torsion", in Algorithmic Number Theory - ANTS-II, H. Cohen (Ed), Springer LNCS 1122, p. 59-65, 1006.

[6] K. Eisentrager, K. Lauter and P. Montgomery, "Fast Elliptic Curve Arithmetic and Improved Weil Pairing Evalutation", CT-RSA 2003, Springer LNCS 2612, p. 343-354.

[7] M. Fouquet, P. Gaudry, and R. Harley, "An extension of Satoh's algorithm and its implementation", Journal of the Ramanujan Mathematical Society, vol. 15, p. 281-318. 2000.

[8] M. Fouquet, P. Gaudry and R. Harley, "Finding Secure Curves with the Satoh-FGH Algorithm and an Early-Abort Strategy", Advances in Cryptology – Eurocrypt 2001, Springer LNCS 2045, p. 14-29.

[9] R. Gallant, R. Lambert, and S. Vanstone, "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms", Advances in Cryptology – Crypto 2001, Springer LNCS 2139, p. 190-200.

[10] P. Gaudry, F. Hess and N. Smart, *Constructive and Destructive Facets of Weil Descent on Elliptic Curves*, in Journal of Cryptology, 15(1), 19-46, Jan. 2002.

[11] W. Gosper, "Sequence Addition Formulae and Related Results", Sandia Internal Report, August 2003.

[12] D. Hankerson, J. Hernandez, A. Menezes, "Software Implementation of Elliptic Curve Cryptography Over Binary Fields", CHES 2000 workshop notes, p.1-24, 2000.

[13] IEEE P1363, Standard Specifications for Public Key Cryptography. Appendix A, 1997.

[14] E. Knudsen, "Elliptic Scalar Multiplication Using Point Halving", Advances in Cryptology – Asiacrypt '99, Springer LNCS 1716, 1999, p. 135-149.

[15] N. Koblitz, "Elliptic Curve Cryptosystems", Math. Comp. (48) p. 203-209, 1987.

[16] N. Koblitz, "CM-curves with good cryptographic properties", Advances in Cryptology – Crypto '91, p. 279-287, 1992.

[17] R. Lercier and D. Lubicz, "Counting Points on Elliptic Curves over Finite Fields of Small Characteristic in Quasi Quadratic Time", Advances in Cryptology – Eurocrypt 2003, Springer LNCS 2656, p. 360-373.

[18] L. Lewin, Dilogarithms and Associated Functions, Macdonald, 1958.

[19] L. Lewin, Polylogarithms and Associated Functions, Elsevier North Holland, 1981.

[20] L. Lewin ed., Structural Properties of Polylogarithms, AMS Surveys and Monograms (37), 1991.

[21] J. Lopez and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2^n)", SAC '98, Springer LNCS 1556, p.201-212.

[22] C. Lim and P. Lee, "More flexible exponentiation with precomputation", Advances in Cryptology – Crypto '94, Springer LNCS 839, p. 95-107.

[23] A. Menezes, T. Okamoto and S. Vanstone, *Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field*, in STOC '91, p. 80-89.

[24] V. Miller, "Use of Elliptic Curves in Cryptography", Advances in Cryptology – Crypto '85, Springer LNCS 218, p. 217-426.

[25] P. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization", Math. Comp. (48), p. 243-264, 1987.

[26] F. Morain, "Calcul du nombre de points sur une courbe elliptique dnas un corps fini: aspects algorithmiques", J. Théor. Nombres Bordeaux (7), p. 255-282, 1995.

[27] F. Morain and J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtaction chains", Informatique Théorique et Applications (24) p. 531-544, 1990.

[28] V. Müller, "Fast multiplication in elliptic curves over small fields of characteristic two", Journal of Cryptology (1), p. 219-234, 1998.

[29] M. Rosing, Implementing Elliptic Curve Cryptography, Manning Publications, 1999.

[30] K. Rubin and A. Silverberg, "Torus-based Cryptography", Advances in Cryptology – Crypto 2003, Springer LNCS 2729, p. 349-365.

[31] T. Satoh, "The Canonical Lift of an Ordinary Elliptic Curve over a Finite Field and its Point Counting", in J. Ramanujan Math. Soc., 15 (4), 2000, pp. 247-270.

[32] T. Satoh, B. Skjernaa and Y. Taguchi, "Fast Computation of Canonical Lifts of Elliptic Curves and its Application to Point Counting", Finite Fields and their Applications (9) p. 89-101, 2003.

[33] R. Schoof, "Elliptic Curves over Finite Fields and the Computation of Square Roots mod p", in Math. Comp. Vol. 44, no. 170, April 1985, pp. 483-494.

[34] R. Schoof, "Counting points on elliptic curves over finite fields" J. Théor. Nombres Bordeaux (7), p. 219-254, 1995.

[35] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast Key Exchange with Elliptic Curve Systems", in Advances in Cryptology – Crypto '95, Springer LNCS 963, 1995, p. 43-56.

[36] R. Schroeppel, "Faster Elliptic Calculations in $GF(2^N)$, preprint March 1998.

[37] R. Schroeppel, "Elliptic Curves – Twice as Fast", Midwest Algebraic Geometry Conference, Urbana, IL, November 2000.

[38] R. Schroeppel, "Circuits for Solving a Quadratic Equation in $GF(2^N)$", in preparation, 2003.

[39] R. Schroeppel, C. Beaver, R. Gonzales, R. Miller and T. Draelos, "A Low Power Design for an Elliptic Curve Digital Signature Chip", CHES 2002, Springer LNCS 2523, p. 366-380.

[40] R. Schroeppel and C. Beaver, "Faster Elliptic Curve Computations using Montgomery's Reciprocal Sharing Trick", patent filed Sept. 2003.

[41] J. Silverman, The Arithmetic of Elliptic Curves. Springer-Verlag, 1986.

[42] J. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves", Advances in Cryptology – Crypto '97, Springer LNCS 1294, p. 357-371.

[43] J. Solinas, "Generalized Mersenne Numbers", Technical Report CORR 99-39, University of Waterloo, 1999.

[44] J. Solinas, "Efficient arithmetic on Koblitz curves", Designs, Codes and Cryptography (19), p. 195-249, 2000.

[45] N. Smart, "Elliptic curve cryptosystems over small fields of odd characteristic", Journal of Cryptology (12), p. 141-151, 1999.

[46] N. Smart, "How Secure Are Elliptic Curves over Composite Extension Fields?", in Eurocrypt 2001, LNCS 2045, May 2001, p. 30-39.

[47] M. Torgerson, "Fast DL Secret Exponentiation with Pre-Computation", patent application filed Sept. 2003.

[48] F. Vercauteren, B. Preneel and J. Vandewalle, "A Memory Efficient Version of Satoh's Algorithm", Advances in Cryptography – Eurocrypt 2001, Springer LNCS2045, p. 1-13.

[49] D. Zagier, Special Values and Functional Equations of Polylogarithms, Appendix A in [20], p. 377-400.

DISTRIBUTION:

5 MS 0785
C. L. Beaver, 6514

5 MS 0785
R. C. Schroeppel, 6514

1 MS 0785
T. S. McDonald, 6514

1 MS 0451
S. G. Varnado, 6500

1 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 9616