Towards Minimizing Memory Requirement for Implementation of Hyperelliptic Curve Crytosystems

Pradeep Kumar Mishra[1], Pinakpani Pal[2] and Palash Sarkar[2]
[1] Centre for Information Security and Cryptography
University of Calgary
Calgary (CANADA)
E-mail: pradeep@math.ucalgary.ca
[2] Cryptology Research Group
Indian Statistical Institute
Kolkata (INDIA)
E-mail: {pinak, palash}@isical.ac.in

Abstract

Elliptic (ECC) and hyperelliptic curve cryptosystems (HECC) have emerged as cryptosystems of choice for small handheld and mobile devices. A lot of research has been devoted to the secure and efficient implementation on these devices. As such devices come with a very low amount of resources, efficient memory management is an important issue in all such implementations. HECC arithmetic is now generally performed using so called explicit formulae. In literature, there is no result which focuses on the exact memory requirement for implementing these formulae. This is the first work to report such minimal memory requirement. Also, in the work we have provided a general methodology for realization of explicit formulae with minimal number of registers. Applying such methodology this work settles the issue for some important explicit formulae available in the literature. This is an attempt to experimentally solve a particular instance based on HECC explicit formulae of the so called "Register Sufficiency Problem", which is an NP-complete problem.

Key Words: Elliptic and hyperelliptic curve cryptosystems, memory, explicit formula, divisor addition, divisor doubling, scalar multiplication.

1 Introduction

For about one and half a decade, elliptic and hyperelliptic curve cryptosystems have occupied the centerstage of public key cryptographic research. The main reason behind it is their versatility. These are the most ideal cryptosystems to be implemented on small mobile devices with low computing power. There is no known subexponential algorithm to solve elliptic or hyperelliptic curve discrete logarithm problem for carefully chosen curves. This ensures a high level of security for smaller key length and makes these cryptosystems suitable for such small devices.

In these cryptosystems, the most dominant operation is the computation of so called scalar multiplication. Unless otherwise stated, in the current work, by a *point* we will generally mean a point on an elliptic curve or a point on the Jacobian of a hyperelliptic curve. Note that the points on the Jacobian are represented by divisor classes of degree zero. Let P be a point and let m be a positive integer. The operation of computing mP is called the scalar multiplication. It is generally computed by a series of point doublings and additions. A lot of effort has been put by the researchers to compute the scalar multiplication efficiently and securely.

The efficiency of scalar multiplication is intimately connected to the efficiency of point addition and doubling algorithms. The efficiency of these algorithms, on the other hand, depends upon the point representation. In affine coordinates, both these operations involve inversion of field elements which is considered a very costly operation. To avoid inversions, various other coordinate systems like projective, Jacobian, modified Jacobian, Lopez-Dahab coordinate systems have been proposed for elliptic curves.

For hyperelliptic curves, Koblitz in his pioneering work [11], had proposed Cantor's algorithm to be used for divisor addition and doubling. Later, it was felt that the computation can be speeded up by fixing the genus of the curves and computing the parameters of the resultant divisor explicitly. Such an algorithm is called an explicit formula. Many proposals of explicit formula have come up in literature and the ones proposed by Lange in [13, 14, 15] are the currently known most efficient ones for general curves of genus 2.

ECC and HECC are considered to be the ideal cryptosystem for mobile devices. Mobile devices are generally equipped with very little computing power. Before trying to implement a cryptosystem on these devices one has to be sure that the resources, particularly memory, available on these devices are sufficient for the implementation. For ECC, the formulae are smaller involving 7 to 15 multiplications and squarings in the underlying field in non-affine arithmetic. So it is simple to calculate the numbers of registers required to store the inputs, outputs and intermediate variables. In many works reported in literature on ECC, the authors have provided the number of registers required for the computation. These figures are obtained by checking manually. However, to our knowledge, there is no result stating that a particular ECC algorithm cannot be implemented in less than a certain amount of memory.

There has been no study of exact memory requirement for an implementation of HECC. In [2] the authors have briefly touched the topic. Besides that there has been no mention of memory requirement in any work on HECC so far. The point addition and doubling algorithm for ECC can be found in many papers as a sequence of three address codes, like, $R_i = R_j \ op \ R_k$, where R_i, R_j, R_k are register names or constants and *op* is an arithmetic operation. In the current work we will refer to this format as *Explicit Register Specified Format (ERSF)*. Looking at a formula in ERSF, one can know exactly how many registers will be required for its implementation. Unfortunately, no HECC explicit formulae occuring in the literature has been described in ERSF. All are described as a set of mathematical equations. We will refer to this format of representing a formula as *raw* format.

Probably, the reason behind all HECC formulae appearing in raw format only is the fact that HECC formulae are relatively complex ones compared to those of ECC. An HECC (genus 2) formula involves around 25 to 50 multiplication with or without an inversion. The first step for expressing such a formulae in ERSF is to know how many registers will be required. For a long formula it is difficult to manually find out how many registers will suffice. It is nearly impossible to say what is the minimal requirement.

In fact, finding the minimum number of intermediate variables required for the execution of a formula is an NP-complete problem. This is called *Register Sufficiency Problem* and has been studied earlier in a very general framework. However, the results obtained earlier do not apply straight away in the scenario we are in nowand, furthermore, we are dealing with clearly describrd algorithms.Hence a brute force approach is possible but is too time consuming. In the current work, we try to provide an experimental solution to this particular instance of the problem. We believe that our methodology can be applied to many similar situations.

The question is: *Given an explicit formula what is the minimum number of registers required to compute it sequentially or in parallel*? In the current work, we provide a methodology (in Section 4) to address this issue. We used this methodology to compute the minimum memory requirement for some of the well known and widely used formulae. For elliptic curves, we checked for the general addition formula in Jacobian coordinates, which is mostly used in implementations and found that it can not be executed with less than 7 registers. It is known that the elliptic curve addition can be done in 7 registers. Our finding ensures that it can not be done in less.

We have used our methodology to find the minimum register requirement for many formulae in HECC. The formulae proposed by Pelzl et al [20] for a special class of curves are very efficient ones. Their doubling formulae uses 10 registers and the addition uses 15. Similar formulae are proposed by Lange in [13]. These set of formulae use 11 and 15 registers for doubling and addition respetively. The doubling formula in [13] requires 6 curve constants to be stored. Thus, (baring the storage required for curve constants) for computing the scalar multiplication both set of formulae require 15 registers. Thus, although the formulae proposed in [20] are cheaper in number of operations, they use the same amount of memory as the ones in [13]. More recently Lange and Stevens [17] have come out with more efficient doubling formulae for all isomorphism classes of curves over binary fields. These formulae not only require very few field operations per group operation but also are very memory efficient. All our findings have been described in Section 6.

The remainder of the paper is organised as follows. In Section 2, we briefly describe the background of the work. In Section 3, we briefly describe the theoretical status of the register sufficiency problem. In Section 4, we describe our methodology. In Section 6, we provide the results we found. Section 7 concludes the paper. The detailed description of the formulae we consider are stated in ERSF in the appendices.

2 Background

Hyperelliptic curve cryptosystems were proposed by Koblitz [11] in 1987. In this section we provide a brief overview of hyperelliptic curves. For details, readers can refer to [18]. Let K be a field and let $\overline{K}$ be the algebraic closure of K. A *hyperelliptic curve* C of genus g over K is an equation of the form $C : y^2 + h(x)y = f(x)$ where $h(x)$ in $K[x]$ is a polynomial of degree at most $g, f(x)$ in $K[x]$ is a monic polynomial of degree $2g + 1$, and there are no singular points (x, y) in $\overline{K} \times \overline{K}$.

Elliptic curves are hyperelliptic curves of genus 1.

The elliptic curve group law does not apply to hyperelliptic curves. The groups used in hyperelliptic curve cryptosystems are the divisor class group, each group element represented by a special kind of divisor called *reduced divisor*. The beauty of the hyperelliptic curves is that the group of divisor classes is isomorphic to the group of ideal classes. That leads to a nice cannonical representation for each group element. Each group element can be represented by a pair of polynomials of small degree, $(u(x), v(x))$, where $deg(v) < deg(u) \leq g$ and u divides $v^2 - hv + f$. Koblitz in his pioneering work suggested to perform the group operation using Cantor's algorithm [3].

Cantor's algorithm for divisor class addition and doubling were quite complex for an efficient implementation. Later it was realised that the efficiency of group law algorithms can be enhanced by fixing the genus of the curve and computing the coefficients of the polynomials representing the resultant divisor directly from those of the input divisor(s). Thus the group law algrithms become a sequence of field operation. Such an algorithm is called an explicit formula. Spallek [22] made the first attempt to compute divisor addition by explicit formula for genus 2 curves over fields of odd characteristic. Gaudry and Harley [7] observed that one can derive different explicit formula for divisor operations depending upon the weight of the divisors. Harley [8] improved the running time of the algorithm in [22] by distinguishing between the different weights of the input divisors and between addition and doubling. Later many researchers came out with various explicit formula for various genera of hyperelliptic curves. An overview of most proposals can be found e. g. in [19].

In the current work we concentrate on curves of genus 2. For most general curves of genus 2, the explicit formulae proposed by Lange are the currently known most efficient ones. In [13], Lange's addition (HCADD) and doubling (HCDBL) involve inversion. Taking the lead from the different projective coordinates in ECC, Lange in [14], [15] has proposed explicit formulae in various coordinate

Table 1: Complexity of Explicit Formulae

Name/Proposed in	Characteristic	Cost(HCADD)	Cost(HCDBL)	Cost (mHCADD)
Lange [13]	All	$1[i] + 22[m] + 3[s]$	$1[i] + 22[m] + 5[s]$	-
Lange [15]	Odd	$47[m] + 7[s]$	$34[m] + 7[s]$	$36[m] + 5[s]$
Lange [15]	Even $h_2 \neq 0$	$46[m] + 4[s]$	$35[m] + 6[s]$	$35[m] + 6[s]$
Lange [15]	Even $h_2 = 0$	$44[m] + 6[s]$	$29[m] + 6[s]$	$34[m] + 6[s]$
Pelzl et al [20]	Even	$1[i] + 9[m] + 6[s]$	$1[i] + 21[m] + 3[s]$	-
Lange et al [17]	Even	$1[i] + 5[m] + 6[s]$	-	-

systems. In [14] she has proposed formulae in "projective" coordinates. Introducing a new variable, a field element in the structure of a divisor the inversion can be avoided in HCADD and HCDBL as in ECC. Again taking the lead from Chudonovski Jacobian coordinates in ECC, Lange has proposed her "new coordinates" in [15], a representation using weighted coordinates. This lead to faster HCADD and especially HCDBL. The latest version all these formulae with an extensive comparison of coordinate systems is available in [16]. In the current work we use formulae presented in [13, 15].

More recently, Pelzl et al. [20] and Lange and Stevens [17] have proposed divisor addition and doubling algorithms for special classes of curves, in which doublings are much cheaper. In Table 1, we provide the complexity of various formulae we investigated in the current work.

3 Theoretical Status of the Register Sufficiency Problem

The problem of minimizing the number of intermediate variables required for executing a set of arithmetic formulae has been studied earlier. The problem is called the register sufficiency problem and the decision version is known to be NP-complete [21]. See [6, page 272] for further details.

The minimization version has also been studied in the literature. According to the compendium of NP-optimization problems [4], there is an $O(\log^2 n)$ (where n is the number of operations) approximation algorithm for this problem [10]. This result is obtained in [10] using general results on flow problems and does not lead to a practical algorithm to solve the problem. Another issue is that one does not obtain any idea about the constant in the $O(\log^2 n)$ expression. Moreover, for the cases in which we are interested n is at most around 200. For such n, a performance guarantee of $O(\log^2 n)$ is not really useful.

In fact, in our implementation, in many cases we are able to obtain the minimum number of registers and in other cases we are able to show that the minimum is at most one or two less than the result we obtain. Thus, on the one hand, the theoretical status of the problem is not really useful for obtaining a practical algorithm and on the other hand, for the concrete situations in which we are interested, we obtain better performance guarantee than the known theoretical bound.

4 Our Methodology

Our primary aim in this work is to answer the question:
Problem: Given an explicit formula $\mathcal{F}$, what is the minimum number of intermediate variables required to be stored to execute $\mathcal{F}$?

Let $\mathcal{F}$ be an explicit formula. Let $p_1, \ldots, p_k$ be the inputs to $\mathcal{F}$. We can look at $\mathcal{F}$ as a sequence of arithmetic operations, each having a unique id, like; $Id_i : p_i = q_i \ op_i \ r_i, k \leq i \leq n$, where op_i is one of the binary operations $\{+,-,*,/\}$ and q_i, r_i are among the p_j's $j < i$. In fact, explicit formula in literature generally occur in raw format. We can convert them into this form by a simple parser program.

We will call a sequence $S = \{Id_{i_1}, Id_{i_2}, \ldots, Id_{i_{n-k+1}}\}$ or simply $S = \{i_1, i_2, \ldots, i_{n-k+1}\}$ of operation id's of $\mathcal{F}$ a *valid sequence* if $\mathcal{F}$ can be computed by executing its operations in the order as dictated by the sequence S. For example if $\mathcal{F} = \{Id_1, Id_2, Id_3, Id_4\}$, where $Id_1 : p_4 = x * y$, $Id_2 : p_5 = p_4 * z$, $Id_3 : p_6 = y * z$ and $Id_4 : p_7 = p_5 * p_6$, then there are only three valid sequences, namely, $\{1,2,3,4\}$, $\{1,3,2,4\}$ and $\{3,1,2,4\}$. $\mathcal{F}$ can not be executed in any other order.

Further, one may be interested in knowing which valid sequence needs the minimum number of intermediate variables for executing the explicit formula $\mathcal{F}$.

Let $\mathcal{F}$ be an explicit formula and let $\mathcal{A}_0$ be the set of inputs to it. In $\mathcal{F}$, there are certain computations which can be computed from the set $\mathcal{A}_0$ of inputs to $\mathcal{F}$. After one or more of them are executed we get some intermediate values which can trigger some more operations of $\mathcal{F}$. Let V_0 be the set of computations in $\mathcal{F}$, which can be computed directly from the set $\mathcal{A}_0$ of inputs to $\mathcal{F}$. Let $|V_0| = \alpha_0$ be the size of the set V_0. So one can begin the execution of $\mathcal{F}$ starting from any one of these α_0 operations. Suppose we choose the operation

$$Id_{i_1} : p_{i_1} = q_{i_1} \ op_{i_1} \ r_{i_1}$$

in V_0 to be executed first. After this operation we have the value of p_{i_1} available to us. So an operation involving p_{i_1} and some other known value in A_0 in its right hand side can also now be executed. Let $\mathcal{A}_{i_1} = \mathcal{A}_0 \bigcup \{p_{i_1}\}$. Let $V_1^{i_1}$ be the set of operations in $\mathcal{F}_1 = \mathcal{F} - \{Id_{i_1}\}$, which can be executed from the the values available in $\mathcal{A}_{i_1}$. Let $|V_1^{i_1}| = \alpha_{i_1}$. Note that the set $V_1^{i_1}$ and the value of α_{i_1} depend upon the choice of Id_{i_1}. Thus, we have α_{i_1} options for executing the next operation of $\mathcal{F}$. Suppose we choose

$$Id_{i_2} : p_{i_2} = q_{i_2} \ op_{i_2} \ r_{i_2}$$

We update the set of available values as $\mathcal{A}_{i_1,i_2} = \mathcal{A}_{i_1} \bigcup \{p_{i_2}\}$ and look for the set of operations in $\mathcal{F}_2 = \mathcal{F}_1 - \{Id_{i_2}\}$ which can be computed from the set of values available in $\mathcal{A}_{i_1,i_2}$ and proceed like this. In general, if k operations $Id_{i_1}, \ldots, Id_{i_k}$ are already computed and p_{i_k} is the output of the last computation, then we update the set of available values as $\mathcal{A}_{i_1,i_2,\ldots,i_{k+1}} = \mathcal{A}_{i_1,i_2,\ldots,i_k} \bigcup \{p_{i_{k+1}}\}$ and set $V_{k+1}^{i_1,\ldots,i_{k+1}}$ to be the set of computations in $\mathcal{F}_{k+1} = \mathcal{F}_k - \{Id_{i_k}\}$, which can be computed from $\mathcal{A}_{k+1}$. Note that Id_{i-k} is the Id of the last operation. Let $|V_k^{i_0,\ldots,i_k}| = \alpha_{i_0,\ldots,i_k}$. We choose one of the operations $\alpha_{i_0,\ldots,i_k}$ operations in $V_k^{i_0,\ldots,i_k}$ to continue.

The sets A_j contain the set of live variables at each step. To minimise the number of intermediate variables we can not afford to keep redundant values in this set. At each step before inserting a new value into $\mathcal{A}_{i_0,\ldots,i_j}$, we check if it contains any value which is not required in further computations in $\mathcal{F}_j$. All such redundant values are discarded from $\mathcal{A}_{i_0,\ldots,i_k}$.

We stop the procedure when for some $\bar{k}$, $V_{\bar{k}}^{i_0,\ldots,i_{\bar{k}}}$ becomes empty. If $\overline{k}$ is $n-k+1$, i.e. the total number of operations in $\mathcal{F}$, then the sequence of operations $\{Id_{i_1}, \ldots, Id_{\bar{k}}\}$, is a valid sequence for $\mathcal{F}$.

After choosing a valid sequence, we check the sets

$$\mathcal{A}_{i_1,\ldots,i_j} - \mathcal{A}_0, \qquad 0 \le j \le n-k+1$$

If $\max_{0 \le i \le n-k+1} |\mathcal{A}_i - \mathcal{A}_0| = \beta$, then for executing $\mathcal{F}$ by the obtained valid sequence the storage for β intermediate variables is necessary and also sufficient.

After obtaining one valid sequence, we backtrack to the last step where we had more choices for the next operation than the ones already undertaken. We choose a different operation from the corresponding set V_i there and proceed in a new path of computation. Following this method we obtain all valid sequences for $\mathcal{F}$ and find the one which requires the minimum number of intermediate variables. That valid sequence gives us the execution sequence of $\mathcal{F}$, which is the most memory efficient one.

Obviously, the method described above is an exhaustive search type. It looks for all possible paths from a possible starting point (which may not be unique) for execution of $\mathcal{F}$ to the end. To find the minimum number of intermediate variables it looks for all possible paths from the beginning to end of $\mathcal{F}$. To bring down the running time we adopt the following four strategies:

1. **Neglecting the paths which requires same number of intermediate variables as the known one:** We use early abort strategy for improving the running time of the algorithm. As we get the first valid sequence we count the number of intermediate variables required to be stored to execute $\mathcal{F}$ by that sequence and store it in a variable, say β. While looking for another path by backtracking, we check the size of the set of intermediate variables after each step. If the current size is equal to the value stored in β, then we need not proceed along this path further. It is not going to yield a more economical path. So we abandon this path and look for another one. If a particular valid sequence needs less than β intermediate values, then we replace the value of β by this new value.

2. **Avoiding the count of the number of intermediate variables at each step:** Counting the number of variables at each step of the algorithm is a time consuming operation. Suppose the value stored currently in β is t. While looking for a new path, we save time by not counting the number of minimum variables till the path is t operations long. Because, if less than t operations have been executed then the number of intermediate variables can not be more than t.

3. **Backtracking several steps at a time:** After finding a valid sequence, instead of backtracking one step (to the last step) at a time, we can go back until a step b such that $\max_{0\leq i\leq b}|\mathcal{A}_i - \mathcal{A}_0| = \beta - 1$. This will reduce the task of going to each level and hence will aid to efficiency. Note that this does not affect the optimality of the final result. That is because the paths which we are skipping need at least β intermediate variables.

4. **Using ordered sets in place of V_j's:** Before starting the first step, we scan the explicit formula and make a frequency table of all the inputs and intermediate variables. Against name of each variable it contains the number of times it has been used in the formula, i.e the number of times it appears in the right hand side of some equation in $\mathcal{F}_j$. We treat the sets V_j's as ordered sets and ordered according as priorities assignined to these equations. The highest priority (see below) is assigned to those, in which the inputs variables have lower frequency. Each time we choose an equation for computation, we update the frequency table by reducing the frequencies of the involved input variables by 1.

   Here we describe how we assign priorities to the operations in V_j. Observe that any equation $r = p\ op\ q$ takes two inputs and computes an intermediate value. If the variables p and q are used again later i.e. their frequencies are greater than 1 before this operation, then we have to store all three variables. But if frequencies of p and q are one then we are required to store the result only. Thus, we can reduce the width of the set of live variables by 1. We assign highest priorities to such computations and put them at the beginning of the sets V_j's. If the frequency of one of p and q is 1, then we need not save that variable. We are required to store the other variable and the computed value and the thus the number of live variables does not increase. So we assign second highest priority to such equations and put them next in the set V_j. So are the equations which have a constant and a variable of frequency 1. Thirdly, squarings and doublings involve one variable only and produce a new variable. So a squaring or a doubling of a variable of frequency 1 also does not increase the number of live variables. We keep such type of operations next. Other equations of $\mathcal{F}_j$ are kept in such an order that the ones involving a variable of minimum frequency preceeds the others.

These optimisation techniques for running time of the algorithm have paid high dividends. It is observed that an implementation using these techniques runs much faster than one without them. These techniques however, do not guarantee that an implementation of this search strategy will terminate in reasonable time for any large explicit formula. An explicit formula may contain a huge number of equations. In that case the program may run for a considerable duration of time and the last value of β may be accepted as the good minimal value. The actual minimum may be lesser.

Let us put the above discussion in the form of an algorithm. In fact, we present two algorithms below, the first one only initialises and calls the second one. The second is a recursive one implementing the techniques described above.

Algorithm MinVar.
Input : The number of inputs to $\mathcal{F}$ and all the operations in $\mathcal{F}$.
Output : Minimum number of intermediate variables required to be stored if $\mathcal{F}$ is executed sequentially.

```
1. (Initialisation) Read the number of inputs $k$ to $\mathcal{F}$;
2. Read all computations in $\mathcal{F}$ and store them in an array indexed by their Id's. Let their number be $N$;
3. Let $k = 0$, minvar = the number of equations in $\mathcal{F}$;
4. Call Algorithm ProcVar($k$, minvar);
5. Output minVar;
```

Algorithm ProcVar(k, minvar).
Input : The number of computations (k) of already carried out.
Output : Recursively computes the number of intermediate variables and outputs their minimum.

```
1. Let Count = $N - k$;
2. If $k >$ minVar
    2.1 Count the number of intermediate variables in the current path;
    2.2 If the number of variables in the current path = minVar then return; /* There is no need to consider this path */
3. If Count $\neq 0$
    3.1 Find the corresponding ordered set $V_k$ and $\alpha_k$;
    3.2 for $i = 1$ to $\alpha_k$;
        3.2.1 Process the $i$th computation in $V_k$;
        3.2.2 Update the data structures for $\mathcal{A}_{k+1}$;
        3.2.3 Call Algorithm ProcVar($k + 1$);
4. minvar = the number of variables in the current path;
5. end
```

Here is an important observation about running of the algorithm.

- **Observation** At any point of time during the execution of the algorithm in the search of a valid sequence, suppose the operations $S_1 = \{i_1, i_2, \ldots, i_k\}$ have been chosen. Also suppose that at the end of this last step the number of live variables is t. Let $S_2 = \{j_1, j_2, \ldots, j_k\}$ be any permutation of $\{i_1, i_2, \ldots, i_k\}$. If S_2 is a valid order in which the operations in S_1 can also be performed, then at the end of last step of S_2 one will have t number of intermediate variables as well. An important application of this observation is that if we have two valid sequences of an explicit formula, which have the same k operations at the beginning, maybe in a different order and if we know that the first requires t intermediate variables till the kth step, then we need not count the number of intermediate variables for first k steps of the second. It is t at the end of kth step.

4.1 The Forward and Reverse Programs

As said earlier, each explicit formula in raw format was modified to be a sequence of binary operations. This is the preprocessing done to each of the raw formulae under consideration. This preprocessed formula was given as input to a program embodying the methodlogy described in the last section, which calculated the minimum number of intermediate variables required for sequential execution of the explicit formula. When the program terminates it outputs exactly how many intermediate variables are required for an execution of the formula and the corresponding valid sequence. As our methodlogy is a kind of exhaustive search with some running time optimization measures, for a long input file it may take substantial amount of time to run. In order to get the results within a reasonable time, another program was employed. We will refer to this later program as the *reverse* program and the former as *forward* program. The *reverse* program initially takes $k = 1$ and using the same logic as *forward* checks if the explicit formula can be executed with k temporary locations. If not it reports this fact and tries again with k replaced by $k+1$. When it gets an affirmative answer it outputs the corresponding value of k and the corresponding valid sequence. After obtaining a path requiring l intermediate variables, the *forward* program looks for path needing less than l intermediate locations. If during the search process it comes across a path which also requires l locations, then *forward* abandons this path and look for a newer one. The *reverse* program uses the same logic, taking $k = 1, 2 \ldots.$ We run both the programs with the same input file on two different machines. If either of *forward* or *reverse* terminates we get the result. Otherwise, if at some point of time *forward* reports the formulae can be executed with k variables and at the same time *reverse* reports it can not be done with less than $k - 1$ intermediate locations, then also the conclusion follows.

The employment of *reverse* helped us to get to the conclusions quite early. Some of the explicit formula under consideration have more that 100 lines of three address codes (i.e. total number of arithmetic operations is more than 100). In spite of the speed-up measures described above, there is no guarantee that *forward* will terminate. In fact, we ran *forward* program without any speed up measure on some longer inputs and found that it did not terminate in a week. Even after the optimisation methods described above are employed, for some of the formulae given in [15] it did not terminate for three days, though it ran much faster. Surprisingly (because they have quite a small number of operations), for the doubling formulae in new coordinates in even characteristic, in both cases $h_2 \neq 0$ and $h_2 = 0$, *forward* did not terminate. So, we took help of the *reverse* program to derive our conclusions. With the help of *reverse* we could get conclusion on any formula in less than two days.

5 Pre and Post Processing

We implemented the methodology described above and found out the minimum number of registers required for a select set of explicit formula. Each formula was preprocessed in the manner described below:

5.1 Preprocessing

We call the different explicit formulae which appear in research papers to be in raw form. Our first step is to preprocess such raw formulae and convert into a form suitable for input to our register minimization program. This conversion of explicit formulae from one form to another is done using a preprocessing program. We verify the correctness of the two forms using Mathematica, a symbolic computation software. We first describe this verification method and then describe the preprocessing algorithm.

Each explicit formula consists of a set of input and output variables and a sequence of arithmetic operations involving temporary intermediate variables. The arithmetic operations are addition, subtraction and multiplication. Thus each output variable can be written as a multivariate polynomial in the input variables. This polynomial can be computed by successively eliminating the intermediate variables. We say that two explicit formulae with the same set of input and output variables are equivalent if the multivariate polynomials corresponding to the output variables are same for both the formulae. We use Mathematica to perform this verification. Being a symbolic computation software, Mathematica can efficiently construct the multivariate polynomial corresponding to the output variable of an explicit formula. There is, however, a problem in this method. If an explicit formula reuses a variable, then Mathematica falls into an infinite loop and fails to construct the multivariate polynomial. There is another pitfall when variables are reused. According to Garey-Johnson [6, page 272], the decision version of the problem when variables can be reused, is known to be NP-hard but not known to be in NP. Thus our first step is to eliminate variable reuse from raw formulae.

We say that a variable is reused if it occurs more than once on the left hand side.There are three different kinds of variable – input, output and temporary. If an input variable is reused, then we want the first occurrence to have the identifier of the input variable, while subsequent ones will have new identifiers. On the other hand, if an output variable is reused, the last occurrence must have the identifier of the output variable while the others will have new identifiers. Reuse of intermediate variables can be tackled either as an input or as an output variable – we tackle them as input variables. While eliminating reuse of input variables, we have to scan the formula from the start to the end and while eliminating reuse of output variables, we have to scan the formula from the end to the start. We do not provide further details, as they are fairly routine and technical in nature.

The second step after elimination of variable reuse is to convert the raw formula into a sequence of binary operations. For this we initially convert the formula to postfix form and then use a stack to obtain the sequence of binary operations. This step is also quite routine and is given in most data structure textbooks. However, we note that we attempt no optimisation at this step. Compiler construction based techniques of using directed acyclic graphs (DAG's) to identify common minimal subexpressions could be applied at this point. Thus the minima we report are really minimum after conversion to three address codes. The actual minimum could be lesser, though we think this to be unlikely.

The third step is to perform a series of checks: no variable is reused, each output variable occurs once on the left side, each intermediate variable should occur exactly once on the left side and at least once on the right side. Finally, we also check if each input variable is used at least once. We do not consider the failure of this check to be a serious error, since some of the curve parameters are sometimes not used in some formulae. However, we do report this failure and if any of the other checks fail, we do not proceed with further processing of the formulae. The output of the third step is verified to be equivalent to the raw formula after eliminating variable reuse. Once this verification succeeds, we provide this as input to our register minimization algorithm.

5.2 Post-processing

As mentioned in the last section, the register minimization programs both determine the minimum number of intermediate variables required in the implementation of the explicit formulae and also output the corresponding valid sequence. If the number of inputs to a formula is α, minimum number of intermediate variables required is β and γ is the number of outputs of the formula, then it can surely be implemented with $\alpha+\beta+\gamma$ locations in memory. For optimal usage of memory, the registers occupied by input variables can be reused after last usage of the input. Also, in some situations one may not like reusing the locations storing the input variables as they may be required later. For example in scalar multiplication algorithms, whenever a HCADD is called one argument is the base point. So, if the locations storing the parameters of the base point are reused, one has to load them again into these registers when HCADD is invoked again by the scalar multiplication algorithm. Thus there may be some inputs such that the register containing them can not be reused. This leads to two cases,

1. All registers are reused.

2. Some selected registers containing some vital inputs are not reallocated.

In our investigation, we allowed reuse of all registers for HCDBL. For HCADD, the inputs are the parameters of two points (divisors) to be added. In our implementation we allowed reuse of the registers containing the parameters of the first divisor. We assumed that the second divisor is the base divisor and its contents should not be destroyed by reusing these registers. For mixed addition algorithm, the divisor which is in affine coordinates is the base point. We did not allow the reuse of the registers containing the point in affine coordinates. Also, the registers containing the curve parameters are never reused nor counted. These are the registers containing the 'vital' inputs.

For sake of generality, we also ran our programs allowing reuse of all the registers (except the ones containing the curve parameters) for all addition formulae. We found that the register usage level can be brought down by reusing all the registers. If in a device, the base point is stored in some permanent memory like some kind of ROM and transfering data from ROM to registers is fast enough, then reuse of all registers is preferable. However, if such data transfer takes significant time, then the time for an addition may go up significantly relative to doubling and the implementation risks being prone to timing attacks.

A register allocation program was implemented which converted the binary instructions of the valid sequences obtained from the minimum intermediate variables programs into the explicit register specified format (ERSF). A register containing a non-vital input is reallocated as soon as the variable it is containing is not required any more (is not a 'live' register).

The output of this program for an explicit formula is in the ready-to-implement form. We applied this methodology to several important formuale in ECC and HECC. For HECC, these outputs are the first explicit formula in ERSF. Hopefully, these formulae will be of importance for an implementation in software or in hardware.

6 Results

The addition (ECADD) and doubling (ECDBL) formulae in ECC have received much attention from the research community and the formulae are quite simpler in comparison to those in HECC. It has been reported by many researchers that the ECADD in mixed Jacobian coordinates for most general curves over fields of odd characterestic needs 7 registers for implementation. To our knowledge, there is no result stating that it can not be executed in less than seven registers. This aroused our curiosity for testing these formulae in our methodology. We experimented with ECADD and ECDBL in Jacobian coordinates and found that ECADD and ECDBL can be implemented with no less than 7 and 6 registers respectively. Both *forward* and *reverse* program reported this fact. As we intended this work to focus on HECC, we did not pay attention to other ECC algorithms.

We applied our methodology to many formula in HECC. First of all, we applied our methodology to Lange's formulae in new coordinates [15]. These formulae are the most efficient ones for general hyperelliptic curves of genus 2. All these formulae are inversion-free. However, the cost of avoiding the inversions is more than an inversion in binary fields. Hence for an implementation over binary fields, affine arithmetic still looks quite attractive. So we used our methodology to calculate the minimum number of registers required for implementing HCDBL and HCADD in affine coordinates also. We used the formulae presented in [13] which are the most efficient ones in affine coordinates for general curves of genus 2. Pelzl et al. [20], have proposed a very efficient HCDBL formula for a special class of curves. We investigated the memory requirement of the HCADD and HCDBL formulae presented in [20] also.

Recently, many efficient doubling formulae have been presented in [17]. Many situations, (like $deg(h) = 1$ or 2, $h_0 = 0$ or $\neq 0$, h_1 is small etc.) have been considered. If a particular variable is small then multiplication by that variable can be effected by some additions. The number of additions will depend upon the value of the small value. Hence we have not inquired these situations. When $deg(h) = 1$ and h_1^2 and h_1^{-1} are precomputed, the doubling formula proposed in [17] is very efficeint. To compute a doubling $(1[i] + 9[m] + 5[s])$ one needs only 7 registers. However to compute the scalar multiplication one has to couple it with an addition formula which requires 15 registers.

Note that HCADD and HCDBL for genus 2 curves have many special cases. The most general and also the most frequent case is the one in which the divisor(s) are of full weight, i.e. if $D = (u, v)$ is the divisor, then $deg(u) = 2, deg(v) = 1$. In the current work we concentrate on the most general and the frequent case only. The same methodology can be applied to other special cases as well. Also, the cost of various operations we have given in the Table 2 below does not corroborate with the costs provided in the corresponding papers. That is because authors generally avoid counting the multiplication and squaring of/with curve constants. In some formulae such operations occur in significant numbers. For example in even characteristic doubling formula ($h_2 \neq 0$), there are 21 such multiplications/squarings. Many of the curve constants can be made 0 or 1. For sake of generality, we have accounted for these multiplications and squarings as well.

We use the following naming convention for the name of various algorithms. The formulae presented in [13] and in [20] are in affine coordinates. We use a superscript $\mathcal{A}$ for them, e.g. HCADD$^{\mathcal{A}}$ and HCDBL$^{\mathcal{A}}$. The formulae in [15] are in Lange's new coordinates. For the formulae in these new coordinates over fields of even characteristic we will use the superscript $\mathcal{N}e$ and for those over fields of odd characteristic we will use superscript $\mathcal{N}o$. Divisor addition algorithms in mixed coordinates will be denoted by a suffix 'm' e.g mHCADD$^{\mathcal{N}o}$.

We summarise our findings in Table 2. In the appendix we present all these formulae in Explicit Register Specified Format.

Observing Table 2, one can conclude that the formulae presented in [17] are best for an implementation over binary fields. However, these are based on a classification of curves into isomorphism classes and if one wants to use curve randomization not all curve parameters can be chosen in this optimal way. An implementation of the formulae in [13], which are more general in nature, needs only one more register in doubling. In the scalar multiplication algorithm, sets of explicit formulae will require 15 registers each. The former has no curve parameter which is not zero or 1. The later requires storing of atmost 6 curve parameters. The computation can be made secure against simple power attacks by using Coron's dummy addition method without any extra registers. Two popular countermeasures against DPA are Coron's point randomisation [5] and Joye-Tymen's Curve randomisation [9]. Both have been extended to hyperelliptic curves by Avanzi [1]. In affine representation point randomisation countermeasure can not be implemented. As there is no curve constant involved in the explicit formulae in [20], curve randomisation can not be applied. Hence for a secure implementation the formulae of [13] are suitable. It will require at most 21 registers (at most 6 registers for curve parameters).

For an implementation over fields of odd characteristic, clearly the formulae in [15] are the most suitable. In this representation mixed addition requires 20 registers and doubling requires 16. Besides 2 curve parameters are to be stored. So the scalar multiplication can be computed in 22 registers. For a secure implementation, Coron's dummy addition method and point randomisation can be used without increasing the number of registers.

For addition formulae we do not reuse the registers containing the parameters of the base point. As stated above we also experimented reusing all registers. We provide our results in Table 3. It can be

Table 2: Register Requirement for Various Explicit Formulae

Algorithm	Proposed in	Characteristic	Cost	Registers Required
HCADD$^{\mathcal{A}}$	[13]	All	$1[i] + 22[m] + 3[s] + 44[a]$	15
HCDBL$^{\mathcal{A}}$	[13]	All	$1[i] + 22[m] + 5[s] + 56[a]$	11
HCADD$^{\mathcal{N}o}$	[15]	Odd	$49[m] + 7[s] + 34[a]$	23
mHCADD$^{\mathcal{N}o}$	[15]	Odd	$36[m] + 5[s] + 35[a]$	16
HCDBL$^{\mathcal{N}o}$	[15]	Odd	$36[m] + 7[s] + 41[a]$	20
HCADD$^{\mathcal{N}e}_{h_2\neq 0}$	[15]	Even	$52[m] + 4[s] + 35[a]$	27
mHCADD$^{\mathcal{N}e}_{h_2\neq 0}$	[15]	Even	$42[m] + 5[s] + 34[a]$	17
HCDBL$^{\mathcal{N}e}_{h_2\neq 0}$	[15]	Even	$54[m] + 8[s] + 29[a]$	20
HCADD$^{\mathcal{N}e}_{h_2=0}$	[15]	Even	$47[m] + 6[s] + 37[a]$	27
mHCADD$^{\mathcal{N}e}_{h_2=0}$	[15]	Even	$37[m] + 6[s] + 30[a]$	22
HCDBL$^{\mathcal{N}e}_{h_2=0}$	[15]	Even	$40[m] + 6[s] + 27[a]$	16
HCADD$^{\mathcal{A}}$	[20]	Even	$1[i] + 21[m] + 3[s] + 30[a]$	15
HCDBL$^{\mathcal{A}}$	[20]	Even	$1[i] + 9[m] + 6[s] + 24[a]$	10
HCDBL$^{\mathcal{A}}_{deg(h)=1}$	[17]	Even	$1[i] + 5[m] + 9[s] + 10[a]$	7
HCDBL$^{\mathcal{A}}_{deg(h)=1,h_1=1}$	[17]	Even	$1[i] + 5[m] + 9[s] + 7[a]$	6
HCDBL$^{\mathcal{A}}_{deg(h)=2,h_0=0}$	[17]	Even	$1[i] + 17[m] + 5[s] + 31[a]$	10

seen that number of registers goes down sinificantly if all registers are reused.

7 Possible Improvements and Conclusion

Although our register minimisation technique produces minimum number of registers required for any explicit formula, its output depends upon the nature of the input file. The input file is generally a sequence of three address codes. There is a vast literature in compiler construction studies on efficient methods for converting an arithmetic formula into three address codes. Our parsing program which converted the explicit formulae into three address codes may not be the optimal one. Therefore there is still some scope for improvement. Besides, the explicit formulae used for finding the minimum register

Table 3: Register Requirement: Register Reuse vs No Reuse

Algorithm	Proposed in	#Registers (all reused)	#Registers (selective reuse)
HCADD$^{\mathcal{A}}$	[13]	13	15
HCADD$^{\mathcal{N}o}$	[15]	19	23
mHCADD$^{\mathcal{N}o}$	[15]	19	20
HCADD$^{\mathcal{N}e}_{h_2\neq 0}$	[15]	23	27
mHCADD$^{\mathcal{N}e}_{h_2\neq 0}$	[15]	18	20
HCADD$^{\mathcal{N}e}_{h_2=0}$	[15]	23	27
mHCADD$^{\mathcal{N}e}_{h_2=0}$	[15]	19	22
HCADD$^{\mathcal{A}}$	[20]	14	15

requirements are best known algorithms. In future, researchers may come out with more efficient formulae. Thus the minimum register requirements reported in the current work may not be the best for hyperelliptic curve cryptosystems.

In a memory constrained small device, we may sacrifice a small amount efficiency for efficient memory usage. That is, instead of keeping a memory location occupied with a computed value which will be required much later, we can free the corresponding location to store other intermediate values and recompute the earlier value once again exactly when it is required. For example, suppose in an algorithm at step k a value $x = y \ op \ z$ is computed and used at Steps $k+1$ and $k+k_1$, where k_1 is not small. Also, suppose that at Step $k+k_1$, both y and z are alive. Then if memory is a concern, instead of storing the value of x for k_1 steps, one may prefer to free that memory at Step $k+2$ and recompute x just before the Step $k+k_1$. Thus one saves a memory location for some steps by recomputing one operation. This may be worthwhile if it is a cheap operation like field addition or negation. In this way one can trade-off memory for some extra operations. In the current work, we have not gone for such optimizations. In an implementation on a small device, this kind of optimization can lead to better utilization of memory.

References

[1] R. M. Avanzi. Countermeasures Against Differential Power Analysis for Hyperelliptic Curve Cryptosystems. In *Proceedings of CHES 2003*, LNCS 2779, pages 366- 381, Springer-Verlag, 2003.

[2] G. Bertoni, L. Breveglieri, T. Wollinger and C. Paar. Finding Optimum Parallel Coprocessor Design for Genus 2 Hyperelliptic Curve Cryptosystems. Cryptology ePrint Archive, Report 2004/29, 2004.

[3] D. G. Cantor. Computing in the Jacobian of a Hyperelliptic curve. In *Mathematics of Computation*, volume 48, pages 95-101, 1987.

[4] A compendium of NP-optimization problems.

`http://www.nada.kth.se/~viggo/problemlist/compendium.html`

[5] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. *Proceedings of CHES 1999*, pp 292-302, 1999.

[6] M. R. Garey and D.S. Johnson. Computers and Intractibility: A Guide to the Theory of NP-completeness. W.H. Freeman, San Francisco, 1979.

[7] P. Gaudry and R. Harley Counting Points on Hyperelliptic Curves over Finite Fields. In *ANTS IV*, volume 1838 of LNCS; pp 297-312, Berlin, 2000, Springer-Verlag.

[8] R. Harley. Fast Arithmetic on Genus 2 Curves. *Avaiable at http://cristal.inria.fr/~harley/hyper,2000.*

[9] M. Joye and C. Tymen Protection against differential attacks for elliptic curve cryptography. *CHES 2001*, LNCS 2162, pp 377-390, Springer-Verlag.

[10] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, 726–737, 1990.

[11] N. Koblitz. Hyperelliptic Cryptosystems. In *Journal of Cryptology*, 1: pages 139–150, 1989.

[12] T. Lange. Efficient Arithmetic on Hyperelliptic Curves. PhD thesis, Universität Gesamthochsschule Essen, 2001.

[13] T. Lange. Efficient Arithmetic on Genus 2 Curves over Finite Fields via Explicit Formulae. Cryptology ePrint Archive, Report 2002/121, 2002. *http://eprint.iacr.org/*.

[14] T. Lange. Inversion-free Arithmetic on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/147, 2002. *http://eprint.iacr.org/*.

[15] T. Lange. Weighted coordinates on Genus 2 Hyperelliptic Curves. Cryptology ePrint Archive, Report 2002/153, 2002. `http://eprint.iacr.org/`.

[16] T. Lange. Formulae for Arithmetic on Genus 2 Hyperelliptic Curves `http://www.itsc.ruhr-uni-bochum.de/tanja/preprints.html`, 2003(To appear in J. AAECC).

[17] T. Lange and M. Stevens. Efficeint Doubling on Genus 2 Curves over Binary Fields. In *Selected Areas in Cryptography, 2004*, LNCS 3061, Springer Verlag 2004.

[18] A. Menezes, Y. Wu, R. Zuccherato. An Elementary Introduction to Hyperelliptic Curves. Technical Report CORR 96-19, University of Waterloo(1996), Canada. Available at http://www.cacr.math.uwaterloo.ca.

[19] P. K. Mishra and P. Sarkar *Parallelizing Explicit Formula for Arithmetic in the Jacobian of Hyperelliptic Curves (Extended Abstract)* In Asiacrypt 2003, LNCS , pp, Springer-Verlag, 2003. Full version available at Cryptology ePrint Archive, Report 2003/180, 2003. `http://eprint.iacr.org/`

[20] J. Pelzl and T. Wollinger and C. Paar. High Performance Arithmetic for Hyperelliptic Curve Cryptosystems of Genus Two Cryptology ePrint Archive, Report 2003/097, 2003. `http://eprint.iacr.org/`.

[21] R. Sethi. Complete register allocation problems. *SIAM Journal of Computing*, 4, 226–248, 1975.

[22] A. M. Spallek. Kurven vom Geschletch 2 und ihre Anwendung in Public-Key-Kryptosystemen. PhD Thesis, Universität Gesamthochschule, Essen, 1994.

A HECC Formulae in ERSF with Selective Register Reuse

Algorithm HCADD$^{\mathcal{N}o}$ of [15]
Curve Constants Used: *None*
Input Variables: $U_{11}, U_{10}, V_{11}, V_{10}, Z_{11}, Z_{12}, z_{11}$,
$U_{21}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2$

1. $R_1 := U11$	2. $R_2 := U10$	3. $R_3 := V11$	4. $R_4 := V10$
5. $R_5 := Z11$	6. $R_6 := Z12$	7. $R_7 := z11$	8. $R_8 := U21$
9. $R_9 := U20$	10. $R_{10} := V21$	11. $R_{11} := V20$	12. $R_{12} := Z21$
13. $R_{13} := Z22$	14. $R_{14} := z21$		
15. $R_{15} := R_1 * R_{14}$	16. $R_{16} := R_2 * R_{14}$	17. $R_{17} := R_5 * R_6$	18. $R_{17} := R_7 * R_{17}$
19. $R_{18} := R_{12} * R_{13}$	20. $R_{14} := R_{14} * R_{18}$	21. $R_{18} := R_3 * R_{14}$	22. $R_{14} := R_4 * R_{14}$
23. $R_{19} := R_7 + R_1$	24. $R_{10} := R_{10} * R_{17}$	25. $R_{11} := R_{11} * R_{17}$	26. $R_{17} := R_{18} - R_{10}$
27. $R_{14} := R_{14} - R_{11}$	28. $R_{18} := R_{14} + R_{17}$	29. $R_9 := R_9 * R_7$	30. $R_{16} := R_9 - R_{16}$
31. $R_{20} := R_{16} * R_7$	32. $R_8 := R_8 * R_7$	33. $R_{15} := R_{15} - R_8$	34. $R_{17} := R_{15} * R_{17}$
35. $R_{19} := R_{17} * R_{19}$	36. $R_{17} := R_2 * R_{17}$	37. $R_{21} := R_1 * R_{15}$	38. $R_{20} := R_{21} + R_{20}$
39. $R_{14} := R_{20} * R_{14}$	40. $R_{17} := R_{14} - R_{17}$	41. $R_{21} := R_7 * R_{15}$	42. $R_{21} := R_{20} + R_{21}$
43. $R_{18} := R_{21} * R_{18}$	44. $R_{14} := R_{18} - R_{14}$	45. $R_{14} := R_{14} - R_{19}$	46. $R_{18} := R_{16} * R_{20}$
47. $R_{13} := R_6 * R_{13}$	48. $R_{19} := R_{15} * R_{15}$	49. $R_{19} := R_{19} * R_2$	50. $R_{18} := R_{18} + R_{19}$
51. $R_{12} := R_5 * R_{12}$	52. $R_{19} := R_{12} * R_{12}$	53. $R_{13} := R_{13} * R_{19}$	54. $R_{13} := R_{13} * R_{18}$
55. $R_{20} := R_{14} * R_{19}$	56. $R_{19} := R_{17} * R_{19}$	57. $R_{18} := R_{18} * R_{20}$	58. $R_{17} := R_{17} * R_{20}$
59. $R_{10} := R_{18} * R_{10}$	60. $R_{11} := R_{18} * R_{11}$	61. $R_{18} := R_{14} * R_{14}$	62. $R_{14} := R_{14} * R_{20}$
63. $R_{16} := R_{16} * R_{14}$	64. $R_{21} := R_{12} * R_{13}$	65. $R_{12} := R_{12} * R_{12}$	66. $R_{13} := R_{13} * R_{13}$
67. $R_{22} := R_{15} + R_8$	68. $R_{18} := R_{18} * R_{22}$	69. $R_{22} := R_{17} + R_{17}$	70. $R_{18} := R_{18} - R_{22}$
71. $R_{18} := R_{15} * R_{18}$	72. $R_{22} := R_{17} + R_{14}$	73. $R_{17} := R_{17} * R_9$	74. $R_9 := R_9 + R_8$
75. $R_9 := R_{22} * R_9$	76. $R_9 := R_9 - R_{17}$	77. $R_{11} := R_{17} + R_{11}$	78. $R_{17} := R_{20} * R_{19}$
79. $R_{22} := R_{20} * R_{20}$	80. $R_{11} := R_{22} * R_{11}$	81. $R_{23} := R_{17} + R_{17}$	82. $R_{19} := R_{19} * R_{19}$
83. $R_{18} := R_{19} + R_{18}$	84. $R_{16} := R_{18} + R_{16}$	85. $R_{18} := R_{14} * R_8$	86. $R_8 := R_8 + R_8$
87. $R_9 := R_9 - R_{18}$	88. $R_{17} := R_{18} + R_{17}$	89. $R_8 := R_8 + R_{15}$	90. $R_8 := R_8 * R_{13}$
91. $R_{13} := R_{15} * R_{14}$	92. $R_{13} := R_{23} - R_{13}$	93. $R_{13} := R_{13} - R_{12}$	94. $R_{14} := R_{17} - R_{13}$
95. $R_{15} := R_{14} * R_{13}$	96. $R_9 := R_9 + R_{10}$	97. $R_{10} := R_{10} + R_{10}$	98. $R_{10} := R_{16} + R_{10}$
99. $R_8 := R_{10} + R_8$	100. $R_{10} := R_{14} * R_8$	101. $R_{10} := R_{10} - R_{11}$	102. $R_9 := R_9 - R_8$
103. $R_9 := R_{22} * R_9$	104. $R_9 := R_{15} - R_9$		
$Up_1 := R_{13}$	$Up_0 := R_8$	$Vp_1 := R_9$	$Vp_0 := R_{10}$
$Zp_1 := R_{20}$	$Zp_2 := R_{21}$	$zp_1 := R_{22}$	$zp_2 := R_{12}$

Number of registers used $= 23$

Algorithm mHCADD$^{\mathcal{N}o}$ of [15]
Curve Constants Used: *None*
Input Variables: $U_{10}, U_{11}, V_{10}, V_{11}, U_{20}, U_{21}, V_{20}, V_{21}, Z_{21}, Z_{22}, z_{21}, z_{22}$
Output Variables: $Up_0, Up_1, Vp_0, Vp_1, Zp_1, Zp_2, zp_1, zp_2$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := U_{21}$	6. $R_6 := U_{20}$	7. $R_7 := V_{21}$	8. $R_8 := V_{20}$
9. $R_9 := Z_{21}$	10. $R_{10} := Z_{22}$	11. $R_{11} := z_{21}$	12. $R_{12} := z_{22}$
13. $R_{10} := R_9 * R_{10}$	14. $R_{13} := R_{11} * R_{10}$	15. $R_{14} := R_1 * R_{11}$	16. $R_{14} := R_{14} - R_5$
17. $R_{15} := R_2 * R_{11}$	18. $R_{15} := R_6 - R_{15}$	19. $R_{16} := R_1 * R_{14}$	20. $R_{16} := R_{16} + R_{15}$
21. $R_{15} := R_{15} * R_{16}$	22. $R_{17} := R_{14} * R_{14}$	23. $R_{17} := R_{17} * R_2$	24. $R_{15} := R_{15} + R_{17}$
25. $R_{10} := R_{15} * R_{10}$	26. $R_{17} := R_{10} * R_9$	27. $R_{10} := R_{10} * R_{10}$	28. $R_{18} := R_{17} * R_{17}$
29. $R_4 := R_4 * R_{13}$	30. $R_4 := R_4 - R_8$	31. $R_3 := R_3 * R_{13}$	32. $R_3 := R_3 - R_7$
33. $R_{13} := R_{16} + R_{14}$	34. $R_{16} := R_{16} * R_4$	35. $R_4 := R_4 + R_3$	36. $R_4 := R_{13} * R_4$
37. $R_3 := R_{14} * R_3$	38. $R_4 := R_4 - R_{16}$	39. $R_{13} := 1 + R_1$	40. $R_{13} := R_3 * R_{13}$
41. $R_3 := R_2 * R_3$	42. $R_3 := R_{16} - R_3$	43. $R_4 := R_4 - R_{13}$	44. $R_{13} := R_{15} * R_4$
45. $R_{15} := R_3 * R_{11}$	46. $R_9 := R_4 * R_9$	47. $R_{16} := R_9 * R_9$	48. $R_{19} := R_3 * R_4$
49. $R_8 := R_{13} * R_8$	50. $R_7 := R_{13} * R_7$	51. $R_{11} := R_{19} * R_{11}$	52. $R_{13} := R_4 * R_4$
53. $R_{20} := R_{13} + R_{19}$	54. $R_{19} := R_{19} * R_6$	55. $R_8 := R_{19} + R_8$	56. $R_8 := R_{16} * R_8$
57. $R_1 := R_1 * R_4$	58. $R_1 := R_3 - R_1$	59. $R_3 := R_{14} * R_4$	60. $R_3 := R_{15} - R_3$
61. $R_1 := R_1 * R_3$	62. $R_3 := R_{11} + R_{11}$	63. $R_4 := R_{13} * R_5$	64. $R_{11} := R_4 + R_{11}$
65. $R_{13} := R_{14} * R_{13}$	66. $R_3 := R_3 - R_{13}$	67. $R_3 := R_3 - R_{18}$	68. $R_{11} := R_{11} - R_3$
69. $R_{13} := R_{11} * R_3$	70. $R_6 := R_5 + R_6$	71. $R_6 := R_{20} * R_6$	72. $R_6 := R_6 - R_{19}$
73. $R_4 := R_6 - R_4$	74. $R_1 := R_1 + R_4$	75. $R_4 := R_4 + R_7$	76. $R_6 := R_7 + R_7$
77. $R_2 := R_2 * R_{16}$	78. $R_1 := R_1 - R_2$	79. $R_1 := R_1 + R_6$	80. $R_2 := R_5 + R_5$
81. $R_2 := R_2 + R_{14}$	82. $R_2 := R_2 * R_{10}$	83. $R_1 := R_1 + R_2$	84. $R_2 := R_{11} * R_1$
85. $R_2 := R_2 - R_8$	86. $R_4 := R_4 - R_1$	87. $R_4 := R_{16} * R_4$	88. $R_4 := R_{13} - R_4$
$Up_1 := R_3$	$Up_0 := R_1$	$Vp_1 := R_4$	$Vp_0 := R_2$
$Zp_1 := R_9$	$Zp_2 := R_{17}$	$zp_1 := R_{16}$	$zp_2 := R_{18}$

Number of registers used $= 20$

Algorithm HCDBL$^{\mathcal{N}o}$ of [15]
Curve Constants Used: f_3, f_2
Input Variables: $U_1, U_0, V_1, V_0, Z_1, Z_2, z_1, z_2$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2$

1. $R_1 := U1$	2. $R_2 := U0$	3. $R_3 := V1$	4. $R_4 := V0$
5. $R_5 := Z1$	6. $R_6 := Z2$	7. $R_7 := z1$	8. $R_8 := z2$
9. $R_9 := R_4 * R_7$	10. $R_{10} := R_3 * R_3$	11. $R_{11} := R_2 * R_7$	12. $R_{12} := R_1 * R_1$
13. $R_{13} := R_{12} - R_{11}$	14. $R_{14} := R_1 * R_3$	15. $R_9 := R_9 - R_{14}$	16. $R_9 := R_4 * R_9$
17. $R_{14} := R_{10} * R_2$	18. $R_9 := R_{14} + R_9$	19. $R_6 := R_6 * R_9$	20. $R_6 := R_6 * R_7$
21. $R_5 := R_6 * R_5$	22. $R_5 := R_5 + R_5$	23. $R_{14} := R_5 * R_5$	24. $R_6 := R_6 * R_6$
25. $R_6 := R_6 + R_6$	26. $R_6 := R_6 * R_1$	27. $R_{15} := R_7 * R_7$	28. $R_{16} := f_3 * R_{15}$
29. $R_{16} := R_{16} + R_{12}$	30. $R_{12} := R_{12} - R_{11}$	31. $R_{12} := R_{13} + R_{12}$	32. $R_{12} := R_{12} + R_{16}$
33. $R_{12} := R_8 * R_{12}$	34. $R_{13} := R_{15} * R_7$	35. $R_{13} := R_{13} * f_2$	36. $R_{15} := R_{11} + R_{11}$
37. $R_{15} := R_{15} + R_{11}$	38. $R_{15} := R_{15} + R_{11}$	39. $R_{15} := R_{15} - R_{16}$	40. $R_{15} := R_1 * R_{15}$
41. $R_{13} := R_{15} + R_{13}$	42. $R_8 := R_8 * R_{13}$	43. $R_8 := R_8 - R_{10}$	44. $R_{10} := R_{16} + R_3$
45. $R_{13} := R_8 * R_{16}$	46. $R_3 := R_{12} * R_3$	47. $R_8 := R_8 + R_{12}$	48. $R_8 := R_{10} * R_8$
49. $R_8 := R_8 - R_{13}$	50. $R_{10} := R_3 * R_{11}$	51. $R_{10} := R_{13} - R_{10}$	52. $R_{11} := 1 + R_1$
53. $R_3 := R_3 * R_{11}$	54. $R_3 := R_8 - R_3$	55. $R_7 := R_3 * R_7$	56. $R_8 := R_9 * R_7$
57. $R_3 := R_8 * R_3$	58. $R_4 := R_8 * R_4$	59. $R_4 := R_4 + R_4$	60. $R_6 := R_3 + R_6$
61. $R_3 := R_3 + R_3$	62. $R_8 := R_{10} * R_{10}$	63. $R_8 := R_8 + R_6$	64. $R_8 := R_8 + R_6$
65. $R_8 := R_8 + R_6$	66. $R_6 := R_8 + R_6$	67. $R_8 := R_{10} * R_7$	68. $R_9 := R_{10} * R_3$
69. $R_3 := R_7 * R_3$	70. $R_{10} := R_7 * R_7$	71. $R_{11} := R_9 + R_3$	72. $R_3 := R_3 * R_1$
73. $R_9 := R_9 * R_2$	74. $R_1 := R_2 + R_1$	75. $R_1 := R_{11} * R_1$	76. $R_1 := R_1 - R_9$
77. $R_2 := R_9 + R_4$	78. $R_1 := R_1 - R_3$	79. $R_1 := R_1 + R_3$	80. $R_3 := R_3 + R_8$
81. $R_1 := R_1 - R_6$	82. $R_1 := R_{10} * R_1$	83. $R_2 := R_{10} * R_2$	84. $R_4 := R_8 + R_8$
85. $R_4 := R_4 - R_{14}$	86. $R_3 := R_3 - R_4$	87. $R_8 := R_3 * R_6$	88. $R_2 := R_8 - R_2$
89. $R_3 := R_3 * R_4$	90. $R_1 := R_3 - R_1$		
$Up1 := R_4$	$Up0 := R_6$	$Vp1 := R_1$	$Vp0 := R_2$
$Zp1 := R_7$	$Zp2 := R_5$	$zp1 := R_{10}$	$zp2 := R_{14}$

Number of registers used $= 16$

Algorithm HCADD$^{\mathcal{N}e}_{h_2 \neq 0}$
Curve Constants Used: h_2, h_1, h_0
Input Variables: $U_{11}, U_{10}, V_{11}, V_{10}, Z_{11}, Z_{12}, z_{11}, z_{12}, z_{13}, z_{14}$,
$U_{21}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := Z_{11}$	6. $R_6 := Z_{12}$	7. $R_7 := z_{11}$	8. $R_8 := z_{12}$
9. $R_9 := z_{13}$	10. $R_{10} := z_{14}$	11. $R_{11} := U_{21}$	12. $R_{12} := U_{20}$
13. $R_{13} := V_{21}$	14. $R_{14} := V_{20}$	15. $R_{15} := Z_{21}$	16. $R_{16} := Z_{22}$
17. $R_{17} := z_{21}$	18. $R_{18} := z_{22}$	19. $R_{19} := z_{23}$	20. $R_{20} := z_{24}$
21. $R_{21} := R_3 * R_{20}$	22. $R_{20} := R_4 * R_{20}$	23. $R_{22} := R_2 * R_{17}$	24. $R_{23} := R_1 * R_{17}$
25. $R_{24} := R_7 + R_1$	26. $R_{13} := R_{13} * R_{10}$	27. $R_{21} := R_{21} + R_{13}$	28. $R_{14} := R_{14} * R_{10}$
29. $R_{20} := R_{20} + R_{14}$	30. $R_{25} := R_{20} + R_{21}$	31. $R_{12} := R_{12} * R_7$	32. $R_{22} := R_{22} + R_{12}$
33. $R_{26} := R_{22} * R_7$	34. $R_{11} := R_{11} * R_7$	35. $R_{23} := R_{23} + R_{11}$	36. $R_{21} := R_{23} * R_{21}$
37. $R_{24} := R_{21} * R_{24}$	38. $R_{21} := R_{21} * R_2$	39. $R_{27} := R_1 * R_{23}$	40. $R_{26} := R_{27} + R_{26}$
41. $R_{20} := R_{26} * R_{20}$	42. $R_{21} := R_{20} + R_{21}$	43. $R_{27} := R_{23} * R_7$	44. $R_{27} := R_{26} + R_{27}$
45. $R_{25} := R_{27} * R_{25}$	46. $R_{20} := R_{25} + R_{20}$	47. $R_{20} := R_{20} + R_{24}$	48. $R_{24} := R_{22} * R_{26}$
49. $R_{25} := R_{12} + R_{11}$	50. $R_{19} := R_9 * R_{19}$	51. $R_{26} := R_{23} * R_{23}$	52. $R_{26} := R_{26} * R_2$
53. $R_{24} := R_{24} + R_{26}$	54. $R_{19} := R_{24} * R_{19}$	55. $R_{17} := R_7 * R_{17}$	56. $R_{26} := R_{20} * R_{17}$
57. $R_{24} := R_{24} * R_{26}$	58. $R_{13} := R_{24} * R_{13}$	59. $R_{14} := R_{24} * R_{14}$	60. $R_{24} := R_{21} * R_{17}$
61. $R_{21} := R_{21} * R_{26}$	62. $R_{12} := R_{21} * R_{12}$	63. $R_{14} := R_{12} + R_{14}$	64. $R_{17} := R_{19} * R_{17}$
65. $R_{19} := R_{23} * R_{19}$	66. $R_{27} := R_{23} + R_{11}$	67. $R_{27} := R_{23} * R_{27}$	68. $R_{23} := R_{23} * R_{20}$
69. $R_{20} := R_{20} * R_{26}$	70. $R_{11} := R_{20} * R_{11}$	71. $R_{21} := R_{21} + R_{20}$	72. $R_{21} := R_{21} * R_{25}$
73. $R_{12} := R_{21} + R_{12}$	74. $R_{20} := R_{22} * R_{20}$	75. $R_{12} := R_{12} + R_{11}$	76. $R_{12} := R_{12} + R_{13}$
77. $R_{13} := R_{24} * R_{24}$	78. $R_{21} := R_{27} * R_{23}$	79. $R_{13} := R_{13} + R_{21}$	80. $R_{13} := R_{13} + R_{20}$
81. $R_{20} := R_{24} + R_{27}$	82. $R_{20} := h_2 * R_{20}$	83. $R_{21} := R_{26} * R_{24}$	84. $R_{11} := R_{11} + R_{21}$
85. $R_{21} := h_1 * R_{26}$	86. $R_{20} := R_{20} + R_{21}$	87. $R_{19} := R_{20} + R_{19}$	88. $R_{19} := R_{17} * R_{19}$
89. $R_{13} := R_{13} + R_{19}$	90. $R_{12} := R_{12} + R_{13}$	91. $R_{19} := R_{23} * R_{26}$	92. $R_{20} := R_{26} * R_{26}$
93. $R_{12} := R_{20} * R_{12}$	94. $R_{14} := R_{20} * R_{14}$	95. $R_{21} := R_{26} * R_{17}$	96. $R_{22} := R_{17} * R_{17}$
97. $R_{24} := R_{20} * R_{21}$	98. $R_{25} := h_2 * R_{21}$	99. $R_{27} := h_2 * R_{21}$	100. $R_{11} := R_{11} + R_{27}$
101. $R_{19} := R_{19} + R_{25}$	102. $R_{19} := R_{19} + R_{22}$	103. $R_{11} := R_{11} + R_{23}$	104. $R_{25} := R_{11} * R_{13}$
105. $R_{14} := R_{25} + R_{14}$	106. $R_{11} := R_{11} * R_{23}$	107. $R_{11} := R_{11} + R_{12}$	108. $R_{12} := h_1 * R_{24}$
109. $R_{11} := R_{11} + R_{12}$	110. $R_{12} := h_0 * R_{24}$	111. $R_{12} := R_{14} + R_{12}$	
$U_p1 := R_{19}$	$Up_0 := R_{13}$	$Vp_1 := R_{11}$	$Vp0 := R_{12}$
$Zp1 := R_{26}$	$Zp2 := R_{17}$	$zp1 := R_{20}$	$zp2 := R_{22}$
$zp3 := R_{21}$	$zp4 := R_{24}$		

Number of registers used $= 27$

Algorithm mHCADD$^{\mathcal{N}e}_{h_2 \neq 0}$
Curve Constants Used: h_2, h_1, h_0.
Input Variables: $U_{11}, U_{10}, V_{11}, V_{10}, U_{21}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := U_{21}$	6. $R_6 := U_{20}$	7. $R_7 := V_{21}$	8. $R_8 := V_{20}$
9. $R_9 := Z_{21}$	10. $R_{10} := Z_{22}$	11. $R_{11} := z_{21}$	12. $R_{12} := z_{22}$
13. $R_{13} := z_{23}$	14. $R_{14} := z_{24}$		
15. $R_{15} := R_1 * R_{11}$	16. $R_{15} := R_{15} + R_5$	17. $R_{11} := R_2 * R_{11}$	18. $R_{11} := R_6 + R_{11}$
19. $R_{16} := R_1 * R_{15}$	20. $R_{16} := R_{16} + R_{11}$	21. $R_{17} := R_{11} * R_{16}$	22. $R_{18} := R_{15} * R_{15}$
23. $R_{18} := R_{18} * R_2$	24. $R_{17} := R_{17} + R_{18}$	25. $R_{13} := R_{17} * R_{13}$	26. $R_{18} := R_4 * R_{14}$
27. $R_{18} := R_{18} + R_8$	28. $R_{14} := R_3 * R_{14}$	29. $R_{14} := R_{14} + R_7$	30. $R_{19} := R_{16} + R_{15}$
31. $R_{16} := R_{16} * R_{18}$	32. $R_{18} := R_{18} + R_{14}$	33. $R_{18} := R_{19} * R_{18}$	34. $R_{14} := R_{15} * R_{14}$
35. $R_{18} := R_{18} - R_{16}$	36. $R_{19} := 1 + R_1$	37. $R_{19} := R_{14} * R_{19}$	38. $R_{14} := R_2 * R_{14}$
39. $R_{14} := R_{16} + R_{14}$	40. $R_{16} := R_{18} - R_{19}$	41. $R_{17} := R_{17} * R_{16}$	42. $R_{18} := R_{14} * R_9$
43. $R_{14} := R_{14} * R_{16}$	44. $R_8 := R_{17} * R_8$	45. $R_7 := R_{17} * R_7$	46. $R_{17} := R_{16} * R_{16}$
47. $R_{16} := R_{16} * R_9$	48. $R_{11} := R_{17} * R_{11}$	49. $R_{19} := R_{17} + R_{14}$	50. $R_{14} := R_{14} * R_6$
51. $R_8 := R_{14} + R_8$	52. $R_{20} := R_{17} * R_5$	53. $R_{17} := R_{15} * R_{17}$	54. $R_9 := R_{13} * R_9$
55. $R_{13} := R_{13} * R_{13}$	56. $R_{13} := R_{15} * R_{13}$	57. $R_5 := R_5 + R_6$	58. $R_5 := R_{19} * R_5$
59. $R_5 := R_5 - R_{14}$	60. $R_5 := R_5 - R_{20}$	61. $R_5 := R_5 + R_7$	62. $R_6 := h_2 * R_9$
63. $R_6 := R_{18} + R_6$	64. $R_6 := R_{18} * R_6$	65. $R_7 := R_{16} * R_{18}$	66. $R_7 := R_{20} + R_7$
67. $R_{14} := R_{16} * R_{16}$	68. $R_8 := R_{14} * R_8$	69. $R_{15} := R_{16} * R_9$	70. $R_{18} := R_9 * R_9$
71. $R_{19} := R_{17} + R_{18}$	72. $R_{20} := h_2 * R_{15}$	73. $R_{20} := R_{20} + R_{17}$	74. $R_{20} := R_1 * R_{20}$
75. $R_6 := R_6 + R_{20}$	76. $R_6 := R_6 + R_{11}$	77. $R_{11} := h_1 * R_{15}$	78. $R_6 := R_6 + R_{11}$
79. $R_6 := R_6 + R_{13}$	80. $R_5 := R_5 + R_6$	81. $R_5 := R_{14} * R_5$	82. $R_{11} := R_{14} * R_{15}$
83. $R_{13} := h_2 * R_{15}$	84. $R_{13} := R_{19} + R_{13}$	85. $R_{19} := h_2 * R_{15}$	86. $R_7 := R_7 + R_{19}$
87. $R_7 := R_7 + R_{17}$	88. $R_{19} := R_7 * R_6$	89. $R_8 := R_{19} + R_8$	90. $R_7 := R_7 * R_{17}$
91. $R_5 := R_7 + R_5$	92. $R_7 := h_1 * R_{11}$	93. $R_5 := R_5 + R_7$	94. $R_7 := h_0 * R_{11}$
95. $R_7 := R_8 + R_7$			
$Up_1 := R_{13}$	$Up_0 := R_6$	$Vp_1 := R_5$	$Vp_0 := R_7$
$Zp_1 := R_{16}$	$Zp_2 := R_9$	$zp_1 := R_{14}$	$zp_2 := R_{18}$
$zp_3 := R_{15}$	$zp_4 := R_{11}$		

Number of registers used $= 20$

Algorithm $\mathbf{HCDBL}^{\mathcal{N}e}_{h_2 \neq 0}$
Curve Constants Used: h_2, h_1, h_0, f_3, f_2.
Input Variables: $U_1, U_0, V_1, V_0, Z_1, Z_2, z_1, z_2, z_3, z_4$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_1$	2. $R_2 := U_0$	3. $R_3 := V_1$	4. $R_4 := V_0$
5. $R_5 := Z_1$	6. $R_6 := Z_2$	7. $R_7 := z_1$	8. $R_8 := z_2$
9. $R_9 := z_3$	10. $R_{10} := z_4$		
11. $R_{11} := h_1 * R_7$	12. $R_{12} := h_1 * h_1$	13. $R_{13} := h_2 * h_2$	14. $R_{14} := h_1 * R_1$
15. $R_{15} := R_3 * h_1$	16. $R_{16} := R_7 * R_7$	17. $R_{12} := R_{12} * R_{16}$	18. $R_{16} := f_3 * R_{16}$
19. $R_{17} := h_2 * R_2$	20. $R_{14} := R_{14} + R_{17}$	21. $R_{17} := h_0 * R_7$	22. $R_{14} := R_{14} + R_{17}$
23. $R_{14} := R_7 * R_{14}$	24. $R_{17} := R_4 * h_2$	25. $R_{15} := R_{15} + R_{17}$	26. $R_{17} := f_2 * R_{10}$
27. $R_{15} := R_{15} + R_{17}$	28. $R_{15} := R_{10} * R_{15}$	29. $R_{17} := h_2 * R_1$	30. $R_{11} := R_{11} + R_{17}$
31. $R_{17} := R_1 * R_1$	32. $R_{13} := R_{13} * R_{17}$	33. $R_{12} := R_{12} + R_{13}$	34. $R_{12} := R_{12} * R_2$
35. $R_{13} := R_{16} + R_{17}$	36. $R_{16} := h_2 * R_{17}$	37. $R_{14} := R_{14} + R_{16}$	38. $R_8 := R_{13} * R_8$
39. $R_{16} := R_3 * h_2$	40. $R_{16} := R_{16} * R_9$	41. $R_8 := R_8 + R_{16}$	42. $R_{16} := h_0 * R_2$
43. $R_{14} := R_{16} * R_{14}$	44. $R_{12} := R_{12} + R_{14}$	45. $R_9 := R_9 * R_{12}$	46. $R_{10} := R_9 * R_{10}$
47. $R_{12} := R_3 * R_3$	48. $R_{14} := R_1 * R_8$	49. $R_{12} := R_{14} + R_{12}$	50. $R_{12} := R_{12} + R_{15}$
51. $R_{14} := R_{13} + R_{11}$	52. $R_{13} := R_{12} * R_{13}$	53. $R_{11} := R_8 * R_{11}$	54. $R_8 := R_{12} + R_8$
55. $R_8 := R_{14} * R_8$	56. $R_8 := R_8 + R_{13}$	57. $R_{12} := h_2 * R_1$	58. $R_{14} := 1 + R_1$
59. $R_{14} := R_{11} * R_{14}$	60. $R_{11} := R_2 * R_{11}$	61. $R_8 := R_8 + R_{14}$	62. $R_{11} := R_{11} * R_7$
63. $R_{11} := R_{13} + R_{11}$	64. $R_{13} := h_2 * R_{11}$	65. $R_{14} := h_1 * R_7$	66. $R_{12} := R_{12} + R_{14}$
67. $R_{12} := R_8 * R_{12}$	68. $R_{12} := R_{13} + R_{12}$	69. $R_{12} := R_{10} * R_{12}$	70. $R_7 := R_8 * R_7$
71. $R_9 := R_9 * R_7$	72. $R_3 := R_9 * R_3$	73. $R_4 := R_9 * R_4$	74. $R_9 := R_{11} * R_{11}$
75. $R_9 := R_9 + R_{12}$	76. $R_{12} := R_{11} * R_7$	77. $R_{11} := R_{11} * R_8$	78. $R_8 := R_7 * R_8$
79. $R_{13} := R_8 + R_{11}$	80. $R_8 := R_8 * R_1$	81. $R_{11} := R_{11} * R_2$	82. $R_{12} := R_8 + R_{12}$
83. $R_4 := R_{11} + R_4$	84. $R_1 := R_1 + R_2$	85. $R_1 := R_{13} * R_1$	86. $R_1 := R_1 - R_{11}$
87. $R_1 := R_1 - R_8$	88. $R_1 := R_1 + R_3$	89. $R_1 := R_1 + R_9$	90. $R_2 := R_7 * R_7$
91. $R_1 := R_2 * R_1$	92. $R_3 := R_2 * R_4$	93. $R_4 := R_7 * R_{10}$	94. $R_8 := R_{10} * R_{10}$
95. $R_{11} := R_2 * R_4$	96. $R_{13} := h_2 * R_4$	97. $R_{14} := h_2 * R_4$	98. $R_{14} := R_8 + R_{14}$
99. $R_{12} := R_{12} + R_{13}$	100. $R_{12} := R_{12} + R_{14}$	101. $R_{13} := R_{12} * R_9$	102. $R_3 := R_{13} - R_3$
103. $R_{12} := R_{12} * R_{14}$	104. $R_1 := R_{12} - R_1$	105. $R_{12} := R_{11} * h_1$	106. $R_1 := R_1 + R_{12}$
107. $R_{12} := R_{11} * h_0$	108. $R_3 := R_3 + R_{12}$		
$Up_1 := R_{14}$	$Up_0 := R_9$	$Vp_1 := R_1$	$Vp_0 := R_3$
$Zp_1 := R_7$	$Zp_2 := R_{10}$	$zp_1 := R_2$	$zp_2 := R_8$
$zp_3 := R_4$	$zp_4 := R_{11}$		

Number of registers used $= 17$

Algorithm HCADD$^{\mathcal{N}e}_{h_2=0}$
Curve Constants Used: h_0.
Input Variables: $U_{21}, U_{11}, U_{10}, V_{11}, V_{10}, Z_{11}, Z_{12}, z_{11}, z_{12}, z_{13}, z_{14}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{21}$	2. $R_2 := U_{11}$	3. $R_3 := U_{10}$	4. $R_4 := V_{11}$
5. $R_5 := V_{10}$	6. $R_6 := Z_{11}$	7. $R_7 := Z_{12}$	8. $R_8 := z_{11}$
9. $R_9 := z_{12}$	10. $R_{10} := z_{13}$	11. $R_{11} := z_{14}$	12. $R_{12} := U_{20}$
13. $R_{13} := V_{21}$	14. $R_{14} := V_{20}$	15. $R_{15} := Z_{21}$	16. $R_{16} := Z_{22}$
17. $R_{17} := z_{21}$	18. $R_{18} := z_{22}$	19. $R_{19} := z_{23}$	20. $R_{20} := z_{24}$
21. $R_{21} := R_4 * R_{20}$	22. $R_{20} := R_5 * R_{20}$	23. $R_{22} := R_3 * R_{17}$	24. $R_{23} := R_2 * R_{17}$
25. $R_{24} := R_8 + R_2$	26. $R_{13} := R_{13} * R_{11}$	27. $R_{21} := R_{21} + R_{13}$	28. $R_{11} := R_{14} * R_{11}$
29. $R_{14} := R_{20} + R_{11}$	30. $R_{20} := R_{14} + R_{21}$	31. $R_{12} := R_{12} * R_8$	32. $R_{22} := R_{22} + R_{12}$
33. $R_{25} := R_{22} * R_8$	34. $R_{26} := R_1 * R_8$	35. $R_{23} := R_{23} + R_{26}$	36. $R_{21} := R_{23} * R_{21}$
37. $R_{24} := R_{21} * R_{24}$	38. $R_{21} := R_{21} * R_3$	39. $R_{27} := R_2 * R_{23}$	40. $R_{25} := R_{27} + R_{25}$
41. $R_{14} := R_{25} * R_{14}$	42. $R_{21} := R_{14} + R_{21}$	43. $R_{27} := R_{23} * R_8$	44. $R_{27} := R_{25} + R_{27}$
45. $R_{20} := R_{27} * R_{20}$	46. $R_{14} := R_{20} + R_{14}$	47. $R_{14} := R_{14} + R_{24}$	48. $R_{20} := R_{22} * R_{25}$
49. $R_{24} := R_{23} + R_{26}$	50. $R_{19} := R_{10} * R_{19}$	51. $R_{25} := R_{23} * R_{23}$	52. $R_{25} := R_{25} * R_3$
53. $R_{20} := R_{20} + R_{25}$	54. $R_{19} := R_{20} * R_{19}$	55. $R_{25} := R_{14} * R_{14}$	56. $R_{24} := R_{25} * R_{24}$
57. $R_{17} := R_8 * R_{17}$	58. $R_{25} := R_{14} * R_{17}$	59. $R_{20} := R_{20} * R_{25}$	60. $R_{14} := R_{14} * R_{25}$
61. $R_{22} := R_{22} * R_{14}$	62. $R_{13} := R_{20} * R_{13}$	63. $R_{11} := R_{20} * R_{11}$	64. $R_{20} := R_{19} * R_{19}$
65. $R_{20} := R_{20} * R_{17}$	66. $R_{20} := R_{24} + R_{20}$	67. $R_{20} := R_{23} * R_{20}$	68. $R_{23} := R_{23} * R_{14}$
69. $R_{19} := R_{19} * R_{17}$	70. $R_{17} := R_{21} * R_{17}$	71. $R_{21} := R_{21} * R_{25}$	72. $R_{24} := R_{17} * R_{25}$
73. $R_{27} := R_{21} + R_{14}$	74. $R_{14} := R_{14} * R_{26}$	75. $R_{21} := R_{21} * R_{12}$	76. $R_{12} := R_{12} + R_{26}$
77. $R_{12} := R_{27} * R_{12}$	78. $R_{12} := R_{12} + R_{14}$	79. $R_{14} := R_{14} + R_{24}$	80. $R_{12} := R_{12} + R_{21}$
81. $R_{12} := R_{12} + R_{13}$	82. $R_{11} := R_{21} + R_{11}$	83. $R_{13} := R_{17} * R_{17}$	84. $R_{13} := R_{13} + R_{20}$
85. $R_{13} := R_{13} + R_{22}$	86. $R_{17} := R_{25} * R_{25}$	87. $R_{11} := R_{17} * R_{11}$	88. $R_{20} := R_{25} * R_{19}$
89. $R_{21} := R_{19} * R_{19}$	90. $R_{22} := R_{23} + R_{21}$	91. $R_{14} := R_{14} + R_{22}$	92. $R_{23} := R_{14} * R_{22}$
93. $R_{24} := 1 * R_{20}$	94. $R_{13} := R_{13} + R_{24}$	95. $R_{14} := R_{14} * R_{13}$	96. $R_{11} := R_{14} + R_{11}$
97. $R_{12} := R_{12} + R_{13}$	98. $R_{12} := R_{17} * R_{12}$	99. $R_{14} := R_{17} * R_{20}$	100. $R_{12} := R_{23} + R_{12}$
101. $R_{23} := R_{14} * 1$	102. $R_{12} := R_{12} + R_{23}$	103. $R_{23} := R_{14} * h_0$	104. $R_{11} := R_{11} + R_{23}$
$Up_1 := R_{22}$	$Up_0 := R_{13}$	$Vp_1 := R_{12}$	$Vp_0 := R_{11}$
$Zp_1 := R_{25}$	$Zp_2 := R_{19}$	$zp_1 := R_{17}$	$zp_2 := R_{21}$
$zp_3 := R_{20}$	$zp_4 := R_{14}$		

Number of registers used $= 27$

Algorithm mHCADD$^{\mathcal{N}e}_{h_2=0}$
Curve Constants Used: h_1, h_0.
Input Variables: $U_{21}, U_{11}, U_{10}, V_{11}, V_{10}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{21}$	2. $R_2 := U_{11}$	3. $R_3 := U_{10}$	4. $R_4 := V_{11}$
5. $R_5 := V_{10}$	6. $R_6 := U_{20}$	7. $R_7 := V_{21}$	8. $R_8 := V_{20}$
9. $R_9 := Z_{21}$	10. $R_{10} := Z_{22}$	11. $R_{11} := z_{21}$	12. $R_{12} := z_{22}$
13. $R_{13} := z_{23}$	14. $R_{14} := z_{24}$		
15. $R_{15} := R_2 * R_{11}$	16. $R_{15} := R_{15} + R_1$	17. $R_{11} := R_3 * R_{11}$	18. $R_{11} := R_6 + R_{11}$
19. $R_{16} := R_2 * R_{15}$	20. $R_{16} := R_{16} + R_{11}$	21. $R_{17} := R_{11} * R_{16}$	22. $R_{18} := R_{15} * R_{15}$
23. $R_{18} := R_{18} * R_3$	24. $R_{17} := R_{17} + R_{18}$	25. $R_{13} := R_{17} * R_{13}$	26. $R_{18} := R_{13} * R_{13}$
27. $R_{13} := R_{13} * R_9$	28. $R_5 := R_5 * R_{14}$	29. $R_5 := R_5 + R_8$	30. $R_{14} := R_4 * R_{14}$
31. $R_{14} := R_{14} + R_7$	32. $R_{19} := R_{16} + R_{15}$	33. $R_{16} := R_{16} * R_5$	34. $R_5 := R_5 + R_{14}$
35. $R_5 := R_{19} * R_5$	36. $R_{14} := R_{15} * R_{14}$	37. $R_5 := R_5 - R_{16}$	38. $R_{19} := 1 + R_2$
39. $R_{19} := R_{14} * R_{19}$	40. $R_{14} := R_3 * R_{14}$	41. $R_{14} := R_{16} + R_{14}$	42. $R_5 := R_5 - R_{19}$
43. $R_{16} := R_{17} * R_5$	44. $R_{17} := R_{14} * R_9$	45. $R_{14} := R_{14} * R_5$	46. $R_8 := R_{16} * R_8$
47. $R_7 := R_{16} * R_7$	48. $R_{16} := R_{17} * R_{17}$	49. $R_{19} := R_5 * R_5$	50. $R_5 := R_5 * R_9$
51. $R_9 := R_{17} * R_5$	52. $R_{11} := R_{11} * R_{19}$	53. $R_{17} := R_{19} + R_{14}$	54. $R_{14} := R_{14} * R_6$
55. $R_8 := R_{14} + R_8$	56. $R_{20} := R_{13} * R_{13}$	57. $R_{21} := R_5 * R_{13}$	58. $R_{22} := R_5 * R_5$
59. $R_6 := R_1 + R_6$	60. $R_6 := R_{17} * R_6$	61. $R_6 := R_6 - R_{14}$	62. $R_{14} := R_{19} * R_1$
63. $R_6 := R_6 - R_{14}$	64. $R_6 := R_6 + R_7$	65. $R_7 := R_{14} + R_9$	66. $R_9 := R_{19} * R_2$
67. $R_9 := R_9 + R_{18}$	68. $R_9 := R_{15} * R_9$	69. $R_{14} := R_{15} * R_{19}$	70. $R_{14} := R_{14} + R_{20}$
71. $R_9 := R_{16} + R_9$	72. $R_9 := R_9 + R_{11}$	73. $R_7 := R_7 + R_{14}$	74. $R_{11} := R_7 * R_{14}$
75. $R_{15} := h_1 * R_{21}$	76. $R_9 := R_9 + R_{15}$	77. $R_7 := R_7 * R_9$	78. $R_{15} := h_1 * R_{21}$
79. $R_6 := R_6 + R_{15}$	80. $R_6 := R_6 + R_9$	81. $R_6 := R_{22} * R_6$	82. $R_6 := R_{11} + R_6$
83. $R_{11} := h_0 * R_{21}$	84. $R_8 := R_8 + R_{11}$	85. $R_8 := R_{22} * R_8$	86. $R_{11} := R_{22} * R_{21}$
87. $R_7 := R_7 + R_8$			
$Up_1 := R_{14}$	$Up_0 := R_9$	$Vp_1 := R_6$	$Vp_0 := R_7$
$Zp_1 := R_5$	$Zp_2 := R_{13}$	$zp_1 := R_{22}$	$zp_2 := R_{20}$
$zp_3 := R_{21}$	$zp_4 := R_{11}$		

Number of registers used $= 22$

Algorithm $\mathbf{HCDBL}^{\mathcal{N}e}_{h_2=0}$ in [15]
Curve Constants: h_1, h_0, f_3, f_2.
Input Variables: $U_1, U_0, V_1, V_0, Z_1, Z_2, z_1, z_2, z_3, z_4$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_1$	2. $R_2 := U_0$	3. $R_3 := V_1$	4. $R_4 := V_0$
5. $R_5 := Z_1$	6. $R_6 := Z_2$	7. $R_7 := z_1$	8. $R_8 := z_2$
9. $R_9 := z_3$	10. $R_{10} := z_4$		
11. $R_{11} := h_1 * R_2$	12. $R_{12} := h_1 * R_1$	13. $R_{13} := f_2 * R_{10}$	14. $R_{14} := R_1 * h_0$
15. $R_{11} := R_{11} + R_{14}$	16. $R_{11} := h_1 * R_{11}$	17. $R_{14} := h_0 * h_0$	18. $R_{14} := R_{14} * R_7$
19. $R_{11} := R_{11} + R_{14}$	20. $R_{11} := R_{11} * R_{10}$	21. $R_{14} := h_0 * R_7$	22. $R_{12} := R_{12} + R_{14}$
23. $R_{14} := R_3 * R_3$	24. $R_{15} := R_1 * R_1$	25. $R_{16} := R_7 * R_7$	26. $R_{16} := f_3 * R_{16}$
27. $R_{15} := R_{16} + R_{15}$	28. $R_8 := R_8 * R_{15}$	29. $R_{15} := 1 + R_1$	30. $R_{16} := R_1 * R_8$
31. $R_{14} := R_{16} + R_{14}$	32. $R_{16} := R_3 * h_1$	33. $R_{13} := R_{13} + R_{16}$	34. $R_{13} := R_{10} * R_{13}$
35. $R_{13} := R_{14} + R_{13}$	36. $R_{14} := R_{12} + h_1$	37. $R_{12} := R_{13} * R_{12}$	38. $R_{13} := R_{13} + R_8$
39. $R_{13} := R_{14} * R_{13}$	40. $R_8 := R_8 * h_1$	41. $R_{13} := R_{13} + R_{12}$	42. $R_{14} := R_{15} * R_8$
43. $R_8 := R_2 * R_8$	44. $R_8 := R_8 * R_7$	45. $R_8 := R_{12} + R_8$	46. $R_{12} := R_{13} + R_{14}$
47. $R_{13} := R_8 * R_8$	48. $R_{14} := R_8 * R_{12}$	49. $R_{15} := R_{14} * R_2$	50. $R_2 := R_1 + R_2$
51. $R_7 := R_{12} * R_7$	52. $R_8 := R_8 * R_7$	53. $R_{12} := R_7 * R_{12}$	54. $R_{14} := R_{12} + R_{14}$
55. $R_2 := R_{14} * R_2$	56. $R_1 := R_{12} * R_1$	57. $R_2 := R_2 - R_{15}$	58. $R_2 := R_2 - R_1$
59. $R_1 := R_1 + R_8$	60. $R_8 := R_{11} * R_7$	61. $R_{10} := R_{11} * R_{10}$	62. $R_3 := R_8 * R_3$
63. $R_4 := R_8 * R_4$	64. $R_4 := R_{15} + R_4$	65. $R_2 := R_2 + R_3$	66. $R_3 := R_7 * R_4$
67. $R_4 := R_7 * R_7$	68. $R_8 := R_7 * R_{10}$	69. $R_{11} := R_{10} * R_{10}$	70. $R_1 := R_1 - R_{11}$
71. $R_{12} := R_1 * R_{11}$	72. $R_{14} := h_1 * R_8$	73. $R_{13} := R_{13} + R_{14}$	74. $R_1 := R_1 * R_{13}$
75. $R_1 := R_1 + R_3$	76. $R_2 := R_2 + R_{13}$	77. $R_2 := R_4 * R_2$	78. $R_2 := R_{12} + R_2$
79. $R_3 := R_4 * R_8$	80. $R_{12} := R_8 * h_1$	81. $R_2 := R_2 + R_{12}$	82. $R_{12} := R_8 * h_0$
83. $R_1 := R_1 + R_{12}$			
$Up_1 := R_{11}$	$Up_0 := R_{13}$	$Vp_1 := R_2$	$Vp_0 := R_1$
$Zp_1 := R_7$	$Zp_2 := R_{10}$	$zp1 := R_4$	$zp_2 := R_{11}$
$zp_3 := R_8$	$zp_4 := R_3$		

Number of registers used $= 16$

Algorithm HCADD$^{\mathcal{A}}$ of [13]
Curve Constants Used: h_2, h_1, h_0, f_4.
Input Variables: $u_{10}, u_{11}, v_{10}, v_{11}, u_{20}, u_{21}, v_{20}, v_{21}$
Output Variables: up_0, up_1, vp_0, vp_1

1. $R_1 := u_{10}$	2. $R_2 := u_{11}$	3. $R_3 := v_{10}$	4. $R_4 := v_{11}$
5. $R_5 := u_{20}$	6. $R_6 := u_{21}$	7. $R_7 := v_{20}$	8. $R_8 := v_{21}$
9. $R_9 := R_5 - R_1$	10. $R_{10} := R_3 - R_7$	11. $R_{11} := R_4 - R_8$	12. $R_{12} := R_{10} + R_{11}$
13. $R_{13} := R_2 - R_6$	14. $R_{14} := R_2 * R_{13}$	15. $R_{14} := R_{14} + R_9$	16. $R_9 := R_9 * R_{14}$
17. $R_{10} := R_{14} * R_{10}$	18. $R_{15} := R_{13} * R_{13}$	19. $R_{15} := R_{15} * R_1$	20. $R_9 := R_9 + R_{15}$
21. $R_{11} := R_{13} * R_{11}$	22. $R_{13} := R_{14} + R_{13}$	23. $R_{12} := R_{13} * R_{12}$	24. $R_{12} := R_{12} - R_{10}$
25. $R_{13} := 1 + R_2$	26. $R_{13} := R_{11} * R_{13}$	27. $R_{11} := R_1 * R_{11}$	28. $R_{10} := R_{10} - R_{11}$
29. $R_{11} := R_{12} - R_{13}$	30. $R_{12} := R_9 * R_{11}$	31. $R_{11} := R_{11} * R_{11}$	32. $R_{12} := 1/R_{12}$
33. $R_{11} := R_{11} * R_{12}$	34. $R_{12} := R_9 * R_{12}$	35. $R_9 := R_9 * R_{12}$	36. $R_{10} := R_{10} * R_{12}$
37. $R_{12} := R_{10} - R_{13}$	38. $R_{13} := R_{10} - R_2$	39. $R_{14} := h2 * R_9$	40. $R_{12} := R_{12} + R_{14}$
41. $R_{12} := R_{13} * R_{12}$	42. $R_{12} := R_{12} - R_1$	43. $R_{13} := R_6 + R_6$	44. $R_{13} := R_{13} + R_{13}$
45. $R_{13} := R_{13} - f4$	46. $R_{14} := R_{10} + R_{10}$	47. $R_{13} := R_{14} - R_{13}$	48. $R_{14} := h2 * R_9$
49. $R_{13} := R_{13} + R_{14}$	50. $R_{14} := R_9 * R_9$	51. $R_{13} := R_{13} * R_{14}$	52. $R_{13} := R_{13} - R_{14}$
53. $R_{14} := R_5 * R_{10}$	54. $R_{15} := R_8 + R_8$	55. $R_{15} := h1 + R_{15}$	56. $R_9 := R_{15} * R_9$
57. $R_{15} := R_6 + R_{10}$	58. $R_6 := R_6 * R_{10}$	59. $R_5 := R_6 + R_5$	60. $R_6 := R_{12} + R_5$
61. $R_6 := R_6 + R_9$	62. $R_6 := R_6 + R_{13}$	63. $R_9 := R_{15} - R_{13}$	64. $R_{10} := R_{13} * R_9$
65. $R_{12} := h2 * R_{13}$	66. $R_{10} := R_{10} + R_6$	67. $R_5 := R_{10} - R_5$	68. $R_5 := R_5 * R_{11}$
69. $R_5 := R_5 - R_8$	70. $R_5 := R_5 - h1$	71. $R_5 := R_5 + R_{12}$	72. $R_8 := R_6 * R_9$
73. $R_9 := h2 * R_6$	74. $R_8 := R_8 - R_{14}$	75. $R_8 := R_8 * R_{11}$	76. $R_7 := R_8 - R_7$
77. $R_7 := R_7 - h0$	78. $R_7 := R_7 + R_9$		
$up_0 := R_6$	$up_1 := R_{13}$	$vp_0 := R_7$	$vp_1 := R_5$

Number of registers used $= 15$

Algorithm HCDBL$^{\mathcal{A}}$ of [13]
Curve Constants: $h_2, h_1, h_0, f4, f_3, f_2$
Input Variables: u_1, u_0, v_1, v_0
Output Variables: up_0, up_1, vp_0, vp_1

1. $R_1 := u_1$	2. $R_2 := u_0$	3. $R_3 := v_1$	4. $R_4 := v_0$
5. $R_5 := f4 * R_1$	6. $R_6 := 2 * R_3$	7. $R_6 := h_1 + R_6$	8. $R_7 := h_2 * R_1$
9. $R_6 := R_6 - R_7$	10. $R_7 := 2 * R_4$	11. $R_7 := h_0 + R_7$	12. $R_8 := h_2 * R_2$
13. $R_7 := R_7 - R_8$	14. $R_8 := R_3 * R_3$	15. $R_9 := R_1 * R_1$	16. $R_5 := R_9 - R_5$
17. $R_9 := f_3 + R_9$	18. $R_5 := 2 * R_5$	19. $R_5 := R_5 + R_9$	20. $R_{10} := 2 * R_2$
21. $R_5 := R_5 - R_{10}$	22. $R_{10} := 2 * R_{10}$	23. $R_9 := R_{10} - R_9$	24. $R_{10} := R_3 * h_2$
25. $R_5 := R_5 - R_{10}$	26. $R_{10} := f4 * R_1$	27. $R_9 := R_9 + R_{10}$	28. $R_{10} := R_3 * h_2$
29. $R_9 := R_9 + R_{10}$	30. $R_9 := R_1 * R_9$	31. $R_9 := R_9 + f_2$	32. $R_8 := R_9 - R_8$
33. $R_9 := R_1 * R_6$	34. $R_{10} := R_7 - R_9$	35. $R_{10} := R_7 * R_{10}$	36. $R_7 := R_7 - R_9$
37. $R_9 := R_6 * R_6$	38. $R_9 := R_2 * R_9$	39. $R_9 := R_9 + R_{10}$	40. $R_{10} := 2 * f4$
41. $R_{10} := R_{10} * R_2$	42. $R_8 := R_8 - R_{10}$	43. $R_{10} := R_3 * h_1$	44. $R_8 := R_8 - R_{10}$
45. $R_{10} := R_4 * h_2$	46. $R_8 := R_8 - R_{10}$	47. $R_{10} := R_7 + R_6$	48. $R_7 := R_8 * R_7$
49. $R_6 := R_5 * R_6$	50. $R_5 := R_8 + R_5$	51. $R_5 := R_{10} * R_5$	52. $R_5 := R_5 - R_7$
53. $R_8 := 1 + R_1$	54. $R_8 := R_6 * R_8$	55. $R_5 := R_5 - R_8$	56. $R_6 := R_2 * R_6$
57. $R_6 := R_7 - R_6$	58. $R_7 := R_9 * R_5$	59. $R_5 := R_5 * R_5$	60. $R_7 := 1/R_7$
61. $R_5 := R_5 * R_7$	62. $R_7 := R_9 * R_7$	63. $R_8 := R_9 * R_7$	64. $R_6 := R_6 * R_7$
65. $R_7 := 2 * R_6$	66. $R_9 := R_8 * R_8$	67. $R_{10} := R_8 * h_2$	68. $R_7 := R_7 + R_{10}$
69. $R_7 := R_7 - R_9$	70. $R_{10} := R_6 - R_1$	71. $R_{10} := h_2 * R_{10}$	72. $R_{11} := 2 * R_3$
73. $R_{10} := R_{10} + R_{11}$	74. $R_{10} := R_{10} + h_1$	75. $R_8 := R_8 * R_{10}$	76. $R_{10} := R_1 + R_6$
77. $R_{10} := R_{10} - R_7$	78. $R_{11} := R_6 * R_6$	79. $R_8 := R_{11} + R_8$	80. $R_{11} := 2 * R_1$
81. $R_{11} := R_{11} - f4$	82. $R_9 := R_9 * R_{11}$	83. $R_8 := R_8 + R_9$	84. $R_1 := R_1 * R_6$
85. $R_1 := R_1 + R_2$	86. $R_2 := R_2 * R_6$	87. $R_6 := R_7 * R_{10}$	88. $R_9 := R_7 * h_2$
89. $R_6 := R_6 + R_8$	90. $R_1 := R_6 - R_1$	91. $R_1 := R_1 * R_5$	92. $R_1 := R_1 - R_3$
93. $R_1 := R_1 - h_1$	94. $R_1 := R_1 + R_9$	95. $R_3 := R_8 * R_{10}$	96. $R_6 := h_2 * R_8$
97. $R_2 := R_3 - R_2$	98. $R_2 := R_2 * R_5$	99. $R_2 := R_2 - R_4$	100. $R_2 := R_2 - h_0$
101. $R_2 := R_2 + R_6$			
$up_0 := R_8$	$up_1 := R_7$	$vp_0 := R_2$	$vp_1 := R_1$

Number of registers used $= 11$

Algorithm HCADD$^{\mathcal{A}}$ of [20]
Input Variables: $u_{10}, u_{11}, v_{10}, v_{11}, u_{20}, u_{21}, v_{20}, v_{21}$
Output Variables: up_0, up_1, vp_0, vp_1

1. $R_1 := u10$	2. $R_2 := u11$	3. $R_3 := v10$	4. $R_4 := v11$
5. $R_5 := u20$	6. $R_6 := u21$	7. $R_7 := v20$	8. $R_8 := v21$
9. $R_9 := R_2 + R_6$	10. $R_{10} := R_5 - R_1$	11. $R_{11} := R_3 - R_7$	12. $R_{12} := R_4 - R_8$
13. $R_{13} := R_{11} + R_{12}$	14. $R_{14} := R_2 - R_6$	15. $R_{12} := R_{14} * R_{12}$	16. $R_{15} := R_2 * R_{14}$
17. $R_{15} := R_{15} + R_{10}$	18. $R_{10} := R_{10} * R_{15}$	19. $R_{11} := R_{15} * R_{11}$	20. $R_{15} := R_{15} + R_{14}$
21. $R_{13} := R_{15} * R_{13}$	22. $R_{13} := R_{13} - R_{11}$	23. $R_{14} := R_{14} * R_{14}$	24. $R_{14} := R_{14} * R_1$
25. $R_{10} := R_{10} + R_{14}$	26. $R_{14} := 1 + R_2$	27. $R_{14} := R_{12} * R_{14}$	28. $R_{12} := R_1 * R_{12}$
29. $R_{11} := R_{11} - R_{12}$	30. $R_{12} := R_{13} - R_{14}$	31. $R_{13} := R_{10} * R_{12}$	32. $R_{12} := R_{12} * R_{12}$
33. $R_{13} := 1/R_{13}$	34. $R_{12} := R_{12} * R_{13}$	35. $R_{13} := R_{10} * R_{13}$	36. $R_{10} := R_{10} * R_{13}$
37. $R_{11} := R_{11} * R_{13}$	38. $R_{13} := R_6 + R_{11}$	39. $R_{14} := R_{11} - R_2$	40. $R_{15} := R_{13} - R_2$
41. $R_{14} := R_{14} * R_{15}$	42. $R_{14} := R_{14} - R_1$	43. $R_6 := R_6 * R_{11}$	44. $R_6 := R_6 + R_5$
45. $R_{14} := R_{14} + R_6$	46. $R_{14} := R_{14} + R_{10}$	47. $R_{10} := R_{10} * R_{10}$	48. $R_9 := R_{10} * R_9$
49. $R_9 := R_{14} + R_9$	50. $R_5 := R_5 * R_{11}$	51. $R_{11} := R_{11} + R_{13}$	52. $R_{11} := R_{11} - R_2$
53. $R_{10} := R_{11} - R_{10}$	54. $R_{11} := R_{13} - R_{10}$	55. $R_{13} := R_{10} * R_{11}$	56. $R_{13} := R_{13} + R_9$
57. $R_{11} := R_9 * R_{11}$	58. $R_5 := R_{11} - R_5$	59. $R_6 := R_{13} - R_6$	60. $R_6 := R_6 * R_{12}$
61. $R_5 := R_5 * R_{12}$	62. $R_6 := R_6 - R_8$	63. $R_5 := R_5 - R_7$	64. $R_6 := R_6 - 1$
$up_0 := R_9$	$up_1 := R_{10}$	$vp_0 := R_5$	$vp_1 := R_6$

Number of registers used $= 15$

Algorithm HCDBL$^{\mathcal{A}}$ of [20]
Input Variables: u_0, u_1, v_0, v_1
Output Variables: up_0, up_1, vp_0, vp_1

1. $R_1 := u0$	2. $R_2 := u1$	3. $R_3 := v0$	4. $R_4 := v1$
5. $R_5 := R_4 * R_4$	6. $R_6 := R_1 * R_1$	7. $R_7 := R_1 + R_2$	8. $R_8 := R_2 * R_2$
9. $R_2 := R_2 * R_8$	10. $R_5 := R_2 + R_5$	11. $R_5 := R_5 + R_4$	12. $R_1 := R_1 * R_5$
13. $R_9 := R_5 + R_8$	14. $R_7 := R_7 * R_9$	15. $R_7 := R_7 + R_2$	16. $R_7 := R_7 + R_1$
17. $R_1 := 1/R_1$	18. $R_6 := R_6 * R_1$	19. $R_1 := R_5 * R_1$	20. $R_9 := R_8 * R_1$
21. $R_2 := R_2 + R_9$	22. $R_2 := R_2 + R_5$	23. $R_5 := R_6 + R_8$	24. $R_1 := R_1 + R_5$
25. $R_9 := R_6 * R_6$	26. $R_{10} := R_9 * R_8$	27. $R_{10} := R_{10} * R_8$	28. $R_8 := R_{10} + R_8$
29. $R_8 := R_8 + R_6$	30. $R_5 := R_5 * R_8$	31. $R_{10} := R_8 + R_9$	32. $R_1 := R_1 * R_{10}$
33. $R_1 := R_1 + R_6$	34. $R_1 := R_1 + R_5$	35. $R_5 := R_5 + R_7$	36. $R_1 := R_1 + 1$
37. $R_3 := R_5 + R_3$	38. $R_1 := R_1 + R_2$	39. $R_1 := R_1 + R_4$	
$up_0 := R_8$	$up_1 := R_9$	$vp_0 := R_3$	$vp_1 := R_1$

Number of registers used $= 10$

Algorithm HCDBL$^{\mathcal{A}}_{deg(h)=1}$ of [17]
Curve Constants: $h0, h1, h1i, h12, f0, f1, f2, f3$
Input Variables: $u0, u1, v0, v1$
Output Variables: $u0p, u1p, v0p, v1p$

1. $R_1 := u0$	2. $R_2 := u1$	3. $R_3 := v0$	4. $R_4 := v1$
5. $R_1 := R_1 * R_1$	6. $R_5 := R_2 * R_2$	7. $R_5 := R_5 + f3$	8. $R_3 := R_3 * R_3$
9. $R_3 := f0 + R_3$	10. $R_3 := 1/R_3$	11. $R_3 := R_3 * R_1$	12. $R_6 := R_5 * R_3$
13. $R_2 := R_6 + R_2$	14. $R_2 := R_2 * R_2$	15. $R_7 := h12 * R_3$	16. $R_5 := R_7 + R_5$
17. $R_6 := R_5 * R_6$	18. $R_2 := R_2 + R_7$	19. $R_5 := R_5 * R_2$	20. $R_5 := R_5 + f1$
21. $R_1 := R_5 + R_1$	22. $R_1 := h1i * R_1$	23. $R_3 := R_7 * R_3$	24. $R_5 := R_7 * R_3$
25. $R_5 := R_6 + R_5$	26. $R_5 := R_5 + f2$	27. $R_4 := R_4 * R_4$	28. $R_4 := R_5 + R_4$
29. $R_4 := h1i * R_4$			
30. $u0p := R_2$	31. $u1p := R_3$	32. $v0p := R_1$	33. $v1p := R_4$

Number of registers used $= 7$

Algorithm HCDBL$^{\mathcal{A}}_{deg(h)=1,h_1=1}$ of [17]
Curve Constants: $f0, f1, f2, f3$
Input Variables: $u0, u1, v0, v1$
Output Variables: $u0p, u1p, v0p, v1p$

1. $R_1 := u0$	2. $R_2 := u1$	3. $R_3 := v0$	4. $R_4 := v1$
5. $R_1 := R_1 * R_1$	6. $R_5 := R_2 * R_2$	7. $R_5 := R_5 + f3$	8. $R_3 := R_3 * R_3$
9. $R_3 := f0 + R_3$	10. $R_3 := 1/R_3$	11. $R_3 := R_3 * R_1$	12. $R_6 := R_5 * R_3$
13. $R_5 := R_3 + R_5$	14. $R_2 := R_6 + R_2$	15. $R_6 := R_5 * R_6$	16. $R_2 := R_2 * R_2$
17. $R_2 := R_2 + R_3$	18. $R_5 := R_5 * R_2$	19. $R_5 := R_5 + f1$	20. $R_1 := R_5 + R_1$
21. $R_4 := R_4 * R_4$	22. $R_5 := R_3 * R_3$	23. $R_3 := R_3 * R_5$	24. $R_3 := R_6 + R_3$
25. $R_3 := R_3 + f2$	26. $R_3 := R_3 + R_4$		
27. $u0p := R_2$	28. $u1p := R_5$	29. $v0p := R_1$	30. $v1p := R_3$

Number of registers used $= 6$

Algorithm HCDBL$^{\mathcal{A}}_{deg(h)=2}$ of [17]
Curve Constants: $h1, h12, f1$
Input Variables: $f4, u0, u1, v0, v1$
Output Variables: $u0p, u1p, v0p, v1p$

1. $R_1 := f4$ 5. $R_5 := v1$	2. $R_2 := u0$	3. $R_3 := u1$	4. $R_4 := v0$
6. $R_6 := R_2 + h12$	7. $R_7 := R_2 * R_2$	8. $R_7 := f1 + R_7$	9. $R_8 := R_3 * R_3$
10. $R_9 := h1 + R_5$	11. $R_9 := R_5 * R_9$	12. $R_{10} := R_8 + R_5$	13. $R_{11} := h1 * R_5$
14. $R_6 := R_6 + R_{11}$	15. $R_{11} := R_4 + R_{11}$	16. $R_8 := R_{11} + R_8$	17. $R_{11} := R_1 * R_3$
18. $R_{12} := R_{10} + R_{11}$	19. $R_{12} := R_3 * R_{12}$	20. $R_9 := R_{12} + R_9$	21. $R_{12} := h1 * R_{10}$
22. $R_4 := R_4 + R_{12}$	23. $R_4 := R_9 + R_4$	24. $R_9 := h1 * R_9$	25. $R_7 := R_7 + R_9$
26. $R_9 := 1/R_7$	27. $R_2 := R_2 * R_9$	28. $R_6 := R_6 * R_2$	29. $R_2 := R_4 * R_2$
30. $R_2 := R_3 + R_2$	31. $R_{12} := R_4 * R_4$	32. $R_4 := h1 * R_4$	33. $R_4 := R_7 + R_4$
34. $R_4 := R_{10} * R_4$	35. $R_4 := R_{12} + R_4$	36. $R_4 := R_9 * R_4$	37. $R_7 := R_2 + h1$
38. $R_9 := R_6 * R_6$	39. $R_9 := R_6 + R_9$	40. $R_{10} := R_9 + R_2$	41. $R_{10} := R_{10} + R_1$
42. $R_{10} := R_{10} + R_3$	43. $R_{10} := R_6 * R_{10}$	44. $R_5 := R_5 + R_{10}$	45. $R_5 := R_5 + R_4$
46. $R_1 := R_1 * R_6$	47. $R_7 := R_7 + R_1$	48. $R_4 := R_7 + R_4$	49. $R_4 := R_2 * R_4$
50. $R_7 := R_2 * R_2$	51. $R_2 := R_2 + h1$	52. $R_2 := R_2 + R_3$	53. $R_1 := R_2 + R_1$
54. $R_1 := R_6 * R_1$	55. $R_1 := R_7 + R_1$	56. $R_2 := R_1 + R_{11}$	57. $R_2 := R_6 * R_2$
58. $R_2 := R_8 + R_2$	59. $R_2 := R_2 + R_4$		
60. $u0p := R_1$	61. $u1p := R_9$	62. $v0p := R_2$	63. $v1p := R_5$

Number of registers used $= 12$

B HECC Addition Formulae in ERSF with Full Register Reuse

Algorithm HCADD$^{\mathcal{N}o}$ of [15]
Curve Constants Used: $None$
Input Variables: $U11, U10, V11, V10, Z11, Z12, z11, U21, U20, V21, V20, Z21, Z22, z21$
Output Variables: $Up1, Up0, Vp1, Vp0, Zp1, Zp2, zp1, zp2$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := Z_{11}$	6. $R_6 := Z_{12}$	7. $R_7 := z_{11}$	8. $R_8 := U_{21}$
9. $R_9 := U_{20}$	10. $R_{10} := V_{21}$	11. $R_{11} := V_{20}$	12. $R_{12} := Z_{21}$
13. $R_{13} := Z_{22}$	14. $R_{14} := z_{21}$		
15. $R_{15} := R_1 * R_{14}$	16. $R_{16} := R_2 * R_{14}$	17. $R_{17} := R_5 * R_6$	18. $R_{17} := R_7 * R_{17}$
19. $R_{18} := R_{12} * R_{13}$	20. $R_{14} := R_{14} * R_{18}$	21. $R_3 := R_3 * R_{14}$	22. $R_4 := R_4 * R_{14}$
23. $R_{14} := R_7 + R_1$	24. $R_6 := R_6 * R_{13}$	25. $R_5 := R_5 * R_{12}$	26. $R_{12} := R_5 * R_5$
27. $R_{13} := R_5 * R_5$	28. $R_6 := R_6 * R_{13}$	29. $R_{11} := R_{11} * R_{17}$	30. $R_{10} := R_{10} * R_{17}$
31. $R_3 := R_3 - R_{10}$	32. $R_4 := R_4 - R_{11}$	33. $R_{17} := R_4 + R_3$	34. $R_9 := R_9 * R_7$
35. $R_{16} := R_9 - R_{16}$	36. $R_{18} := R_{16} * R_7$	37. $R_8 := R_8 * R_7$	38. $R_{15} := R_{15} - R_8$
39. $R_1 := R_1 * R_{15}$	40. $R_1 := R_1 + R_{18}$	41. $R_{18} := R_{16} * R_1$	42. $R_4 := R_1 * R_4$
43. $R_{19} := R_{15} * R_{15}$	44. $R_{19} := R_{19} * R_2$	45. $R_{18} := R_{18} + R_{19}$	46. $R_6 := R_6 * R_{18}$
47. $R_5 := R_5 * R_6$	48. $R_6 := R_6 * R_6$	49. $R_3 := R_{15} * R_3$	50. $R_7 := R_7 * R_{15}$
51. $R_1 := R_1 + R_7$	52. $R_1 := R_1 * R_{17}$	53. $R_1 := R_1 - R_4$	54. $R_7 := R_3 * R_{14}$
55. $R_2 := R_2 * R_3$	56. $R_2 := R_4 - R_2$	57. $R_1 := R_1 - R_7$	58. $R_3 := R_1 * R_1$
59. $R_4 := R_2 * R_{13}$	60. $R_7 := R_1 * R_{13}$	61. $R_{13} := R_{18} * R_7$	62. $R_2 := R_2 * R_7$
63. $R_1 := R_1 * R_7$	64. $R_{10} := R_{13} * R_{10}$	65. $R_{11} := R_{13} * R_{11}$	66. $R_{13} := R_{16} * R_1$
67. $R_{14} := R_2 + R_1$	68. $R_{15} := R_7 * R_4$	69. $R_{16} := R_7 * R_7$	70. $R_{17} := R_{15} + R_{15}$
71. $R_4 := R_4 * R_4$	72. $R_{18} := R_1 * R_8$	73. $R_{15} := R_{18} + R_{15}$	74. $R_1 := R_{15} * R_1$
75. $R_1 := R_{17} - R_1$	76. $R_1 := R_1 - R_{12}$	77. $R_{15} := R_{15} - R_1$	78. $R_{17} := R_{15} * R_1$
79. $R_{19} := R_2 * R_9$	80. $R_2 := R_2 + R_2$	81. $R_9 := R_9 + R_8$	82. $R_9 := R_{14} * R_9$
83. $R_9 := R_9 - R_{19}$	84. $R_{11} := R_{19} + R_{11}$	85. $R_9 := R_9 - R_{18}$	86. $R_9 := R_9 + R_{10}$
87. $R_{10} := R_{10} + R_{10}$	88. $R_{11} := R_{16} * R_{11}$	89. $R_{14} := R_{15} + R_8$	90. $R_8 := R_8 + R_8$
91. $R_3 := R_3 * R_{14}$	92. $R_2 := R_3 - R_2$	93. $R_2 := R_{15} * R_2$	94. $R_3 := R_8 + R_{15}$
95. $R_3 := R_3 * R_6$	96. $R_2 := R_4 + R_2$	97. $R_2 := R_2 + R_{13}$	98. $R_2 := R_2 + R_{10}$
99. $R_2 := R_2 + R_3$	100. $R_3 := R_{15} * R_2$	101. $R_3 := R_3 - R_{11}$	102. $R_4 := R_9 - R_2$
103. $R_4 := R_{16} * R_4$	104. $R_4 := R_{17} - R_4$		
$Up_1 := R_1$	$Up_0 := R_2$	$Vp_1 := R_4$	$Vp_0 := R_3$
$Zp_1 := R_7$	$Zp_2 := R_5$	$zp_1 := R_{16}$	$zp_2 := R_{12}$

Number of registers used $= 19$

Algorithm mHCADD$^{\mathcal{N}o}$ of [15]
Curve Constants Used: *None*
Input Variables: $U_{10}, U_{11}, V_{10}, V_{11}, U_{20}, U_{21}, V_{20}, V_{21}, Z_{21}, Z_{22}, z_{21}, z_{22}$
Output Variables: $Up_0, Up_1, Vp_0, Vp_1, Zp_1, Zp_2, zp_1, zp_2$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := U_{21}$	6. $R_6 := U_{20}$	7. $R_7 := V_{21}$	8. $R_8 := V_{20}$
9. $R_9 := Z_{21}$	10. $R_{10} := Z_{22}$	11. $R_{11} := z_{21}$	12. $R_{12} := z_{22}$
13. $R_{10} := R_9 * R_{10}$	14. $R_{13} := R_{11} * R_{10}$	15. $R_{14} := R_1 * R_{11}$	16. $R_{14} := R_{14} - R_5$
17. $R_{15} := R_2 * R_{11}$	18. $R_{15} := R_6 - R_{15}$	19. $R_{16} := R_1 * R_{14}$	20. $R_{16} := R_{16} + R_{15}$
21. $R_{15} := R_{15} * R_{16}$	22. $R_{17} := R_{14} * R_{14}$	23. $R_{17} := R_{17} * R_2$	24. $R_{15} := R_{15} + R_{17}$
25. $R_{10} := R_{15} * R_{10}$	26. $R_{17} := R_{10} * R_9$	27. $R_{10} := R_{10} * R_{10}$	28. $R_{18} := R_{17} * R_{17}$
29. $R_4 := R_4 * R_{13}$	30. $R_4 := R_4 - R_8$	31. $R_3 := R_3 * R_{13}$	32. $R_3 := R_3 - R_7$
33. $R_{13} := R_{16} + R_{14}$	34. $R_{16} := R_{16} * R_4$	35. $R_{14} := R_{14} * R_3$	36. $R_3 := R_4 + R_3$
37. $R_3 := R_{13} * R_3$	38. $R_3 := R_3 - R_{16}$	39. $R_4 := 1 + R_1$	40. $R_4 := R_{14} * R_4$
41. $R_{13} := R_2 * R_{14}$	42. $R_{13} := R_{16} - R_{13}$	43. $R_3 := R_3 - R_4$	44. $R_4 := R_{15} * R_3$
45. $R_{14} := R_{13} * R_{11}$	46. $R_9 := R_3 * R_9$	47. $R_{15} := R_9 * R_9$	48. $R_{16} := R_{13} * R_3$
49. $R_8 := R_4 * R_8$	50. $R_4 := R_4 * R_7$	51. $R_7 := R_{16} * R_{11}$	52. $R_{11} := R_3 * R_3$
53. $R_{19} := R_{11} + R_{16}$	54. $R_{16} := R_{16} * R_6$	55. $R_8 := R_{16} + R_8$	56. $R_8 := R_{15} * R_8$
57. $R_1 := R_1 * R_3$	58. $R_1 := R_{13} - R_1$	59. $R_3 := R_{14} * R_3$	60. $R_3 := R_{14} - R_3$
61. $R_1 := R_1 * R_3$	62. $R_3 := R_7 + R_7$	63. $R_{13} := R_{11} * R_5$	64. $R_7 := R_{13} + R_7$
65. $R_{11} := R_{14} * R_{11}$	66. $R_3 := R_3 - R_{11}$	67. $R_3 := R_3 - R_{18}$	68. $R_7 := R_7 - R_3$
69. $R_{11} := R_7 * R_3$	70. $R_6 := R_5 + R_6$	71. $R_6 := R_{19} * R_6$	72. $R_6 := R_6 - R_{16}$
73. $R_6 := R_6 - R_{13}$	74. $R_1 := R_1 + R_6$	75. $R_6 := R_6 + R_4$	76. $R_4 := R_4 + R_4$
77. $R_2 := R_2 * R_{15}$	78. $R_1 := R_1 - R_2$	79. $R_1 := R_1 + R_4$	80. $R_2 := R_5 + R_5$
81. $R_2 := R_2 + R_{14}$	82. $R_2 := R_2 * R_{10}$	83. $R_1 := R_1 + R_2$	84. $R_2 := R_7 * R_1$
85. $R_2 := R_2 - R_8$	86. $R_4 := R_6 - R_1$	87. $R_4 := R_{15} * R_4$	88. $R_4 := R_{11} - R_4$
$Up_1 := R_3$	$Up_0 := R_1$	$Vp_1 := R_4$	$Vp_0 := R_2$
$Zp_1 := R_9$	$Zp_2 := R_{17}$	$zp_1 := R_{15}$	$zp_2 := R_{18}$

Number of registers used $= 19$

Algorithm HCADD$^{\mathcal{N}e}_{h_2 \neq 0}$ of [15]
Curve Constants Used: h_2, h_1, h_0
Input Variables: $U_{11}, U_{10}, V_{11}, V_{10}, Z_{11}, Z_{12}, z_{11}, z_{12}, z_{13}, z_{14}$,
$U_{21}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := Z_{11}$	6. $R_6 := Z_{12}$	7. $R_7 := z_{11}$	8. $R_8 := z_{12}$
9. $R_9 := z_{13}$	10. $R_{10} := z_{14}$	11. $R_{11} := U_{21}$	12. $R_{12} := U_{20}$
13. $R_{13} := V_{21}$	14. $R_{14} := V_{20}$	15. $R_{15} := Z_{21}$	16. $R_{16} := Z_{22}$
17. $R_{17} := z_{21}$	18. $R_{18} := z_{22}$	19. $R_{19} := z_{23}$	20. $R_{20} := z_{24}$
21. $R_3 := R_3 * R_{20}$	22. $R_4 := R_4 * R_{20}$	23. $R_{20} := R_2 * R_{17}$	24. $R_{21} := R_1 * R_{17}$
25. $R_9 := R_9 * R_{19}$	26. $R_{13} := R_{13} * R_{10}$	27. $R_3 := R_3 + R_{13}$	28. $R_{10} := R_{14} * R_{10}$
29. $R_4 := R_4 + R_{10}$	30. $R_{14} := R_4 + R_3$	31. $R_{12} := R_{12} * R_7$	32. $R_{19} := R_{20} + R_{12}$
33. $R_{20} := R_{19} * R_7$	34. $R_{11} := R_{11} * R_7$	35. $R_{21} := R_{21} + R_{11}$	36. $R_{22} := R_1 * R_{21}$
37. $R_{20} := R_{22} + R_{20}$	38. $R_{22} := R_{19} * R_{20}$	39. $R_4 := R_{20} * R_4$	40. $R_{23} := R_{21} * R_{21}$
41. $R_{23} := R_{23} * R_2$	42. $R_{22} := R_{22} + R_{23}$	43. $R_9 := R_{22} * R_9$	44. $R_3 := R_{21} * R_3$
45. $R_{21} := R_{21} * R_7$	46. $R_{20} := R_{20} + R_{21}$	47. $R_{14} := R_{20} * R_{14}$	48. $R_{14} := R_{14} + R_4$
49. $R_{20} := R_{21} * R_9$	50. $R_1 := R_7 + R_1$	51. $R_1 := R_3 * R_1$	52. $R_1 := R_{14} + R_1$
53. $R_2 := R_3 * R_2$	54. $R_2 := R_4 + R_2$	55. $R_3 := R_{12} + R_{11}$	56. $R_4 := R_7 * R_{17}$
57. $R_7 := R_9 * R_4$	58. $R_9 := R_2 * R_4$	59. $R_4 := R_1 * R_4$	60. $R_{14} := R_{22} * R_4$
61. $R_2 := R_2 * R_4$	62. $R_{12} := R_2 * R_{12}$	63. $R_{13} := R_{14} * R_{13}$	64. $R_{10} := R_{14} * R_{10}$
65. $R_{10} := R_{12} + R_{10}$	66. $R_{14} := R_9 * R_9$	67. $R_{17} := R_{21} + R_{11}$	68. $R_{17} := R_{21} * R_{17}$
69. $R_{21} := R_{21} * R_1$	70. $R_1 := R_1 * R_4$	71. $R_{11} := R_1 * R_{11}$	72. $R_2 := R_2 + R_1$
73. $R_1 := R_{19} * R_1$	74. $R_2 := R_2 * R_3$	75. $R_2 := R_2 + R_{12}$	76. $R_2 := R_2 + R_{11}$
77. $R_2 := R_2 + R_{13}$	78. $R_3 := R_{17} * R_{21}$	79. $R_3 := R_{14} + R_3$	80. $R_1 := R_3 + R_1$
81. $R_3 := R_9 + R_{17}$	82. $R_3 := h_2 * R_3$	83. $R_9 := R_4 * R_9$	84. $R_9 := R_{11} + R_9$
85. $R_{11} := h_1 * R_4$	86. $R_3 := R_3 + R_{11}$	87. $R_3 := R_3 + R_{20}$	88. $R_3 := R_7 * R_3$
89. $R_1 := R_1 + R_3$	90. $R_2 := R_2 + R_1$	91. $R_3 := R_{21} * R_4$	92. $R_{11} := R_4 * R_4$
93. $R_2 := R_{11} * R_2$	94. $R_{10} := R_{11} * R_{10}$	95. $R_{12} := R_4 * R_7$	96. $R_{13} := R_7 * R_7$
97. $R_{14} := R_{11} * R_{12}$	98. $R_{17} := h_2 * R_{12}$	99. $R_{19} := h_2 * R_{12}$	100. $R_9 := R_9 + R_{19}$
101. $R_3 := R_3 + R_{17}$	102. $R_3 := R_3 + R_{13}$	103. $R_9 := R_9 + R_{21}$	104. $R_{17} := R_9 * R_1$
105. $R_{10} := R_{17} + R_{10}$	106. $R_9 := R_9 * R_{21}$	107. $R_2 := R_9 + R_2$	108. $R_9 := h_1 * R_{14}$
109. $R_2 := R_2 + R_9$	110. $R_9 := h_0 * R_{14}$	111. $R_9 := R_{10} + R_9$	
$Up_1 := R_3$	$Up_0 := R_1$	$Vp_1 := R_2$	$Vp_0 := R_9$
$Zp_1 := R_4$	$Zp_2 := R_7$	$zp_1 := R_{11}$	$zp_2 := R_{13}$
$zp_3 := R_{12}$	$zp_4 := R_{14}$		

Number of registers used $= 23$

Algorithm mHCADD$^{\mathcal{N}e}_{h_2\neq 0}$ of [15]
Curve Constants Used: h_2, h_1, h_0.
Input Variables: $U_{11}, U_{10}, V_{11}, V_{10}, U_{21}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := U_{21}$	6. $R_6 := U_{20}$	7. $R_7 := V_{21}$	8. $R_8 := V_{20}$
9. $R_9 := Z_{21}$	10. $R_{10} := Z_{22}$	11. $R_{11} := z_{21}$	12. $R_{12} := z_{22}$
13. $R_{13} := z_{23}$	14. $R_{14} := z_{24}$		
15. $R_{15} := R_1 * R_{11}$	16. $R_{15} := R_{15} + R_5$	17. $R_{11} := R_2 * R_{11}$	18. $R_{11} := R_6 + R_{11}$
19. $R_{16} := R_1 * R_{15}$	20. $R_{16} := R_{16} + R_{11}$	21. $R_{17} := R_{11} * R_{16}$	22. $R_{18} := R_{15} * R_{15}$
23. $R_{18} := R_{18} * R_2$	24. $R_{17} := R_{17} + R_{18}$	25. $R_{13} := R_{17} * R_{13}$	26. $R_4 := R_4 * R_{14}$
27. $R_4 := R_4 + R_8$	28. $R_3 := R_3 * R_{14}$	29. $R_3 := R_3 + R_7$	30. $R_{14} := R_{16} + R_{15}$
31. $R_{16} := R_{16} * R_4$	32. $R_{15} := R_{15} * R_3$	33. $R_3 := R_4 + R_3$	34. $R_3 := R_{14} * R_3$
35. $R_3 := R_3 - R_{16}$	36. $R_4 := 1 + R_1$	37. $R_4 := R_{15} * R_4$	38. $R_2 := R_2 * R_{15}$
39. $R_2 := R_{16} + R_2$	40. $R_3 := R_3 - R_4$	41. $R_4 := R_{17} * R_3$	42. $R_{14} := R_2 * R_9$
43. $R_2 := R_2 * R_3$	44. $R_8 := R_4 * R_8$	45. $R_4 := R_4 * R_7$	46. $R_7 := R_3 * R_3$
47. $R_3 := R_3 * R_9$	48. $R_{11} := R_7 * R_{11}$	49. $R_{15} := R_7 + R_2$	50. $R_2 := R_2 * R_6$
51. $R_8 := R_2 + R_8$	52. $R_{16} := R_3 * R_{14}$	53. $R_{17} := R_{13} * R_{13}$	54. $R_9 := R_{13} * R_9$
55. $R_{13} := R_{15} * R_{17}$	56. $R_{15} := R_{15} * R_7$	57. $R_7 := R_7 * R_5$	58. $R_{16} := R_7 + R_{16}$
59. $R_5 := R_5 + R_6$	60. $R_5 := R_{15} * R_5$	61. $R_2 := R_5 - R_2$	62. $R_2 := R_2 - R_7$
63. $R_2 := R_2 + R_4$	64. $R_4 := h_2 * R_9$	65. $R_4 := R_{14} + R_4$	66. $R_4 := R_{14} * R_4$
67. $R_5 := R_3 * R_3$	68. $R_6 := R_5 * R_8$	69. $R_7 := R_3 * R_9$	70. $R_8 := R_9 * R_9$
71. $R_{14} := R_{15} + R_8$	72. $R_{15} := h_2 * R_7$	73. $R_{15} := R_{15} + R_{15}$	74. $R_1 := R_1 * R_{15}$
75. $R_1 := R_4 + R_1$	76. $R_1 := R_1 + R_{11}$	77. $R_4 := h_1 * R_7$	78. $R_1 := R_1 + R_4$
79. $R_1 := R_1 + R_{13}$	80. $R_2 := R_2 + R_1$	81. $R_2 := R_5 * R_2$	82. $R_4 := R_5 * R_7$
83. $R_{11} := h_2 * R_7$	84. $R_{11} := R_{14} + R_{11}$	85. $R_{13} := h_2 * R_7$	86. $R_{13} := R_{16} + R_{13}$
87. $R_{13} := R_{13} + R_{15}$	88. $R_{14} := R_{13} * R_1$	89. $R_6 := R_{14} + R_6$	90. $R_{13} := R_{13} * R_{15}$
91. $R_2 := R_{13} + R_2$	92. $R_{13} := h_1 * R_4$	93. $R_2 := R_2 + R_{13}$	94. $R_{13} := h_0 * R_4$
95. $R_6 := R_6 + R_{13}$			
$Up_1 := R_{11}$	$Up_0 := R_1$	$Vp_1 := R_2$	$Vp_0 := R_6$
$Zp_1 := R_3$	$Zp_2 := R_9$	$zp_1 := R_5$	$zp_2 := R_8$
$zp_3 := R_7$	$zp_4 := R_4$		

Number of registers used $= 18$

Algorithm HCADD$^{\mathcal{N}e}_{h_2=0}$ of [15]
Curve Constants Used: h_0, h_1.
Input Variables: $U_{21}, U_{11}, U_{10}, V_{11}, V_{10}, Z_{11}, Z_{12}, z_{11}, z_{12}, z_{13}, z_{14}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{11}$	2. $R_2 := U_{10}$	3. $R_3 := V_{11}$	4. $R_4 := V_{10}$
5. $R_5 := Z_{11}$	6. $R_6 := Z_{12}$	7. $R_7 := z_{11}$	8. $R_8 := z_{12}$
9. $R_9 := z_{13}$	10. $R_{10} := z_{14}$	11. $R_{11} := U_{21}$	12. $R_{12} := U_{20}$
13. $R_{13} := V_{21}$	14. $R_{14} := V_{20}$	15. $R_{15} := Z_{21}$	16. $R_{16} := Z_{22}$
17. $R_{17} := z_{21}$	18. $R_{18} := z_{22}$	19. $R_{19} := z_{23}$	20. $R_{20} := z_{24}$
21. $R_3 := R_3 * R_{20}$	22. $R_4 := R_4 * R_{20}$	23. $R_{20} := R_2 * R_{17}$	24. $R_{21} := R_1 * R_{17}$
25. $R_9 := R_9 * R_{19}$	26. $R_{13} := R_{13} * R_{10}$	27. $R_3 := R_3 + R_{13}$	28. $R_{10} := R_{14} * R_{10}$
29. $R_4 := R_4 + R_{10}$	30. $R_{14} := R_4 + R_3$	31. $R_{12} := R_{12} * R_7$	32. $R_{19} := R_{20} + R_{12}$
33. $R_{20} := R_{19} * R_7$	34. $R_{11} := R_{11} * R_7$	35. $R_{21} := R_{21} + R_{11}$	36. $R_{22} := R_1 * R_{21}$
37. $R_{20} := R_{22} + R_{20}$	38. $R_{22} := R_{19} * R_{20}$	39. $R_4 := R_{20} * R_4$	40. $R_{23} := R_{21} * R_{21}$
41. $R_{23} := R_{23} * R_2$	42. $R_{22} := R_{22} + R_{23}$	43. $R_9 := R_{22} * R_9$	44. $R_3 := R_{21} * R_3$
45. $R_{21} := R_{21} * R_7$	46. $R_{20} := R_{20} + R_{21}$	47. $R_{14} := R_{20} * R_{14}$	48. $R_{14} := R_{14} + R_4$
49. $R_{20} := R_9 * R_9$	50. $R_1 := R_7 + R_1$	51. $R_1 := R_3 * R_1$	52. $R_1 := R_{14} + R_1$
53. $R_2 := R_3 * R_2$	54. $R_2 := R_4 + R_2$	55. $R_3 := R_{21} + R_{11}$	56. $R_4 := R_1 * R_1$
57. $R_3 := R_4 * R_3$	58. $R_4 := R_7 * R_{17}$	59. $R_7 := R_{20} * R_4$	60. $R_3 := R_3 + R_7$
61. $R_3 := R_{21} * R_3$	62. $R_7 := R_9 * R_4$	63. $R_9 := R_2 * R_4$	64. $R_4 := R_1 * R_4$
65. $R_{14} := R_{22} * R_4$	66. $R_1 := R_1 * R_4$	67. $R_2 := R_2 * R_4$	68. $R_{17} := R_{19} * R_1$
69. $R_{19} := R_{21} * R_1$	70. $R_{13} := R_{14} * R_{13}$	71. $R_{10} := R_{14} * R_{10}$	72. $R_{14} := R_9 * R_4$
73. $R_{20} := R_2 + R_1$	74. $R_1 := R_1 * R_{11}$	75. $R_2 := R_2 * R_{12}$	76. $R_{11} := R_{12} + R_{11}$
77. $R_{11} := R_{20} * R_{11}$	78. $R_{11} := R_{11} + R_1$	79. $R_1 := R_1 + R_{14}$	80. $R_{11} := R_{11} + R_2$
81. $R_{11} := R_{11} + R_{13}$	82. $R_2 := R_2 + R_{10}$	83. $R_9 := R_9 * R_9$	84. $R_3 := R_9 + R_3$
85. $R_3 := R_3 + R_{17}$	86. $R_9 := R_4 * R_4$	87. $R_2 := R_9 * R_2$	88. $R_{10} := R_4 * R_7$
89. $R_{12} := R_7 * R_7$	90. $R_{13} := R_{19} + R_{12}$	91. $R_1 := R_1 + R_{13}$	92. $R_{14} := R_1 * R_{13}$
93. $R_{17} := h_1 * R_{10}$	94. $R_3 := R_3 + R_{17}$	95. $R_1 := R_1 * R_3$	96. $R_1 := R_1 + R_2$
97. $R_2 := R_{11} + R_3$	98. $R_2 := R_9 * R_2$	99. $R_{11} := R_9 * R_{10}$	100. $R_2 := R_{14} + R_2$
101. $R_{14} := R_{11} * h_1$	102. $R_2 := R_2 + R_{14}$	103. $R_{14} := R_{11} * h_0$	104. $R_1 := R_1 + R_{14}$
$Up_1 := R_{13}$	$Up_0 := R_3$	$Vp_1 := R_2$	$Vp_0 := R_1$
$Zp_1 := R_4$	$Zp_2 := R_7$	$zp_1 := R_9$	$zp_2 := R_{12}$
$zp_3 := R_{10}$	$zp_4 := R_{11}$		

Number of registers used $= 23$

Algorithm mHCADD$^{\mathcal{N}e}_{h_2=0}$ of [15]
Curve Constants Used: h_1, h_0.
Input Variables: $U_{21}, U_{11}, U_{10}, V_{11}, V_{10}, U_{20}, V_{21}, V_{20}, Z_{21}, Z_{22}, z_{21}, z_{22}, z_{23}, z_{24}$
Output Variables: $Up_1, Up_0, Vp_1, Vp_0, Zp_1, Zp_2, zp_1, zp_2, zp_3, zp_4$

1. $R_1 := U_{21}$ 5. $R_5 := Z_{21}$ 9. $R_9 := z_{23}$	2. $R_2 := U_{20}$ 6. $R_6 := Z_{22}$ 10. $R_{10} := z_{24}$	3. $R_3 := V_{21}$ 7. $R_7 := z_{21}$	4. $R_4 := V_{20}$ 8. $R_8 := z_{22}$
11. $R_{11} := U11 * R_7$ 15. $R_{12} := U11 * R_{11}$ 19. $R_{14} := R_{14} + R_4$ 23. $R_{12} := R_{12} * R_{14}$ 27. $R_{10} := R_{10} - R_{12}$ 31. $R_9 := R_{13} * R_9$ 35. $R_{15} := R_{11} * R_{15}$ 39. $R_{12} := R_{13} * R_{10}$ 43. $R_5 := R_{10} * R_5$ 47. $R_{10} := R_{10} + R_{16}$ 51. $R_{11} := R_{11} * R_{14}$ 55. $R_{13} := R_{13} * R_{13}$ 59. $R_3 := R_{12} * R_3$ 63. $R_2 := R_{11} * R_2$ 67. $R_3 := R_4 + R_3$ 71. $R_{11} := h_1 * R_{17}$ 75. $R_3 := R_3 + R_{11}$ 79. $R_4 := h_0 * R_{17}$ 83. $R_1 := R_1 + R_2$	12. $R_{11} := R_{11} + R_1$ 16. $R_{12} := R_{12} + R_7$ 20. $R_{10} := V11 * R_{10}$ 24. $R_{11} := R_{11} * R_{10}$ 28. $R_{14} := R_{11} * R_{11}$ 32. $R_{14} := R_9 * R_9$ 36. $R_{11} := U10 * R_{11}$ 40. $R_{13} := R_{11} * R_5$ 44. $R_7 := R_7 * R_{15}$ 48. $R_{17} := R_5 * R_9$ 52. $R_{14} := R_{13} * R_5$ 56. $R_{11} := R_{13} + R_{11}$ 60. $R_1 := R_{15} * R_1$ 64. $R_{11} := R_{11} - R_2$ 68. $R_1 := R_1 + R_{14}$ 72. $R_7 := R_7 + R_{11}$ 76. $R_3 := R_3 + R_7$ 80. $R_2 := R_2 + R_4$	13. $R_7 := U10 * R_7$ 17. $R_{13} := R_7 * R_{12}$ 21. $R_{10} := R_{10} + R_3$ 25. $R_{10} := R_{14} + R_{10}$ 29. $R_{14} := R_{14} * U10$ 33. $R_9 := R_9 * R_5$ 37. $R_{11} := R_{12} + R_{11}$ 41. $R_{11} := R_{11} * R_{10}$ 45. $R_{10} := R_{11} * R_{15}$ 49. $R_{18} := R_{15} * U11$ 53. $R_{18} := R_5 * R_5$ 57. $R_7 := R_{11} + R_7$ 61. $R_{11} := R_{15} + R_{11}$ 65. $R_2 := R_2 + R_4$ 69. $R_1 := R_1 + R_{10}$ 73. $R_1 := R_1 * R_7$ 77. $R_3 := R_{18} * R_3$ 81. $R_2 := R_{18} * R_2$	14. $R_7 := R_2 + R_7$ 18. $R_{14} := V10 * R_{10}$ 22. $R_{15} := R_{12} + R_{11}$ 26. $R_{10} := R_{15} * R_{10}$ 30. $R_{13} := R_{13} + R_{14}$ 34. $R_{15} := 1 + U11$ 38. $R_{10} := R_{10} - R_{15}$ 42. $R_{15} := R_{10} * R_{10}$ 46. $R_{16} := R_9 * R_9$ 50. $R_{14} := R_{18} + R_{14}$ 54. $R_{19} := R_1 + R_2$ 58. $R_4 := R_{12} * R_4$ 62. $R_{11} := R_{11} * R_{19}$ 66. $R_4 := R_{11} - R_1$ 70. $R_4 := R_1 * R_{10}$ 74. $R_{11} := h_1 * R_{17}$ 78. $R_3 := R_4 + R_3$ 82. $R_4 := R_{18} * R_{17}$
$Up_1 := R_{10}$ $Zp_1 := R_5$ $zp_3 := R_{17}$	$Up_0 := R_7$ $Zp_2 := R_9$ $zp_4 := R_4$	$Vp_1 := R_3$ $zp_1 := R_{18}$	$Vp_0 := R_1$ $zp_2 := R_{16}$

Number of registers used $= 19$

Algorithm HCADD$^{\mathcal{A}}$ of [13]
Curve Constants Used: h_2, h_1, h_0, f_4.
Input Variables: $u_{10}, u_{11}, v_{10}, v_{11}, u_{20}, u_{21}, v_{20}, v_{21}$
Output Variables: up_0, up_1, vp_0, vp_1

1. $R_1 := u_{10}$	2. $R_2 := u_{11}$	3. $R_3 := v_{10}$	4. $R_4 := v_{11}$
5. $R_5 := u_{20}$	6. $R_6 := u_{21}$	7. $R_7 := v_{20}$	8. $R_8 := v_{21}$
9. $R_9 := R_5 - R_1$	10. $R_3 := R_3 - R_7$	11. $R_4 := R_4 - R_8$	12. $R_{10} := R_3 + R_4$
13. $R_{11} := R_2 - R_6$	14. $R_{12} := R_2 * R_{11}$	15. $R_{12} := R_{12} + R_9$	16. $R_9 := R_9 * R_{12}$
17. $R_3 := R_{12} * R_3$	18. $R_{13} := R_{11} * R_{11}$	19. $R_{13} := R_{13} * R_1$	20. $R_9 := R_9 + R_{13}$
21. $R_4 := R_{11} * R_4$	22. $R_{11} := R_{12} + R_{11}$	23. $R_{10} := R_{11} * R_{10}$	24. $R_{10} := R_{10} - R_3$
25. $R_{11} := 1 + R_2$	26. $R_{11} := R_4 * R_{11}$	27. $R_4 := R_1 * R_4$	28. $R_3 := R_3 - R_4$
29. $R_4 := R_{10} - R_{11}$	30. $R_{10} := R_9 * R_4$	31. $R_4 := R_4 * R_4$	32. $R_{10} := 1/R_{10}$
33. $R_4 := R_4 * R_{10}$	34. $R_{10} := R_9 * R_{10}$	35. $R_9 := R_9 * R_{10}$	36. $R_3 := R_3 * R_{10}$
37. $R_{10} := R_3 - R_{11}$	38. $R_2 := R_3 - R_2$	39. $R_{11} := h_2 * R_9$	40. $R_{10} := R_{10} + R_{11}$
41. $R_2 := R_2 * R_{10}$	42. $R_1 := R_2 - R_1$	43. $R_2 := 2 * R_6$	44. $R_2 := R_2 + R_{11}$
45. $R_2 := R_2 - f_4$	46. $R_{10} := 2 * R_3$	47. $R_{10} := R_{10} - R_{11}$	48. $R_{11} := h_2 * R_9$
49. $R_{10} := R_{10} + R_{11}$	50. $R_{11} := R_9 * R_9$	51. $R_2 := R_2 * R_{11}$	52. $R_{10} := R_{10} - R_{11}$
53. $R_{11} := R_5 * R_3$	54. $R_{12} := 2 * R_8$	55. $R_{12} := h_1 + R_{12}$	56. $R_9 := R_{12} * R_9$
57. $R_{12} := R_6 + R_3$	58. $R_3 := R_6 * R_3$	59. $R_3 := R_3 + R_5$	60. $R_1 := R_1 + R_3$
61. $R_1 := R_1 + R_9$	62. $R_1 := R_1 + R_2$	63. $R_2 := R_{12} - R_{10}$	64. $R_5 := R_{10} * R_2$
65. $R_6 := h_2 * R_{10}$	66. $R_5 := R_5 + R_1$	67. $R_3 := R_5 - R_3$	68. $R_3 := R_3 * R_4$
69. $R_3 := R_3 - R_8$	70. $R_3 := R_3 - h_1$	71. $R_3 := R_3 + R_6$	72. $R_2 := R_1 * R_2$
73. $R_5 := h_2 * R_1$	74. $R_2 := R_2 - R_{11}$	75. $R_2 := R_2 * R_4$	76. $R_2 := R_2 - R_7$
77. $R_2 := R_2 - h_0$	78. $R_2 := R_2 + R_5$		
$up_0 := R_1$	$up_1 := R_{10}$	$vp_0 := R_2$	$vp_1 := R_3$

Number of registers used $= 13$

Algorithm HCADD$^{\mathcal{A}}of$ [20]
Input Variables: $u_{10}, u_{11}, v_{10}, v_{11}, u_{20}, u_{21}, v_{20}, v_{21}$
Output Variables: up_0, up_1, vp_0, vp_1

1. $R_1 := u_{10}$	2. $R_2 := u_{11}$	3. $R_3 := v_{10}$	4. $R_4 := v_{11}$
5. $R_5 := u_{20}$	6. $R_6 := u_{21}$	7. $R_7 := v_{20}$	8. $R_8 := v_{21}$
9. $R_9 := R_2 + R_6$	10. $R_{10} := 1 + R_2$	11. $R_4 := R_4 - R_8$	12. $R_3 := R_3 - R_7$
13. $R_{11} := R_3 + R_4$	14. $R_{12} := R_5 - R_1$	15. $R_{13} := R_2 - R_6$	16. $R_4 := R_{13} * R_4$
17. $R_{10} := R_4 * R_{10}$	18. $R_4 := R_1 * R_4$	19. $R_{14} := R_2 * R_{13}$	20. $R_{14} := R_{14} + R_{12}$
21. $R_{12} := R_{12} * R_{14}$	22. $R_3 := R_{14} * R_3$	23. $R_{14} := R_{14} + R_{13}$	24. $R_{11} := R_{14} * R_{11}$
25. $R_{11} := R_{11} - R_3$	26. $R_{10} := R_{11} - R_{10}$	27. $R_3 := R_3 - R_4$	28. $R_4 := R_{13} * R_{13}$
29. $R_4 := R_4 * R_1$	30. $R_4 := R_{12} + R_4$	31. $R_{11} := R_4 * R_{10}$	32. $R_{10} := R_{10} * R_{10}$
33. $R_{11} := 1/R_{11}$	34. $R_{10} := R_{10} * R_{11}$	35. $R_{11} := R_4 * R_{11}$	36. $R_4 := R_4 * R_{11}$
37. $R_3 := R_3 * R_{11}$	38. $R_{11} := R_4 * R_4$	39. $R_9 := R_{11} * R_9$	40. $R_{12} := R_6 + R_3$
41. $R_{13} := R_3 - R_2$	42. $R_6 := R_6 * R_3$	43. $R_6 := R_6 + R_5$	44. $R_5 := R_5 * R_3$
45. $R_3 := R_3 + R_{12}$	46. $R_3 := R_3 - R_2$	47. $R_3 := R_3 - R_{11}$	48. $R_2 := R_{12} - R_2$
49. $R_2 := R_{13} * R_2$	50. $R_1 := R_2 - R_1$	51. $R_1 := R_1 + R_6$	52. $R_1 := R_1 + R_4$
53. $R_1 := R_1 + R_9$	54. $R_2 := R_{12} - R_3$	55. $R_4 := R_3 * R_2$	56. $R_4 := R_4 + R_1$
57. $R_2 := R_1 * R_2$	58. $R_2 := R_2 - R_5$	59. $R_4 := R_4 - R_6$	60. $R_4 := R_4 * R_{10}$
61. $R_2 := R_2 * R_{10}$	62. $R_4 := R_4 - R_8$	63. $R_2 := R_2 - R_7$	64. $R_4 := R_4 - 1$
$up_0 := R_1$	$up_1 := R_3$	$vp_0 := R_2$	$vp1 := R_4$

Number of registers used $= 14$