Efficient Key Agreement and Signature Schemes Using Compact Representations in $GF(p^{10})$

Kenneth J. Giuliani[1] and Guang Gong[2]

[1] Dept. of Combinatorics and Optimization
University of Waterloo
Waterloo, ON, Canada, N2L 3G1
kjgiulia@cacr.math.uwaterloo.ca

[2] Dept. of Electrical and Computer Engineering
University of Waterloo
Waterloo, ON, Canada, N2L 3G1
ggong@calliope.uwaterloo.ca

Abstract. We propose a new cryptographic system and signature scheme based on the finite field $GF(p^{10})$. A compact representation is given allowing keys to be stored with only 40% as many bits as in the canonical representation. We examine the efficiency of several algorithms for cryptographic computation. In particular, we introduce a new algorithm based on fifth-order characteristic sequences which is faster than any previously known algorithm for computing with compact representations. An extensive analysis is performed to quantify the the amount of computation needed in these algorithms. We also introduce the *Trace-DLP* problem on which the signature scheme is based.

Keywords: Finite fields, compact representations, LUC, GH, XTR, fifth-order characteristic sequences

1 Introduction

Key size and bandwidth are both important issues to consider when dealing with data and cryptographic transmission. As a result, there have been several proposals for finite field based cryptographic systems which reduce the number of bits needed for public keys. These cryptosystems include LUC [9], GH [4], and XTR [6] which use the fields $GF(p^2)$, $GF(p^3)$, and $GF(p^6)$ respectively.

This paper proposes using a system using the finite field field $GF(p^{10})$. A compact representation is given which allows representation to be accomplished using only two elements from $GF(p^2)$, a reduction to 40% of the canonical representation. We show how to perform a Diffie-Hellman key agreement and give an ElGamal-like signature scheme this representation. Also, an elegant new algorithm for basic cryptographic computation is given based on fifth-order characteristic sequences which is faster than the naive algorithms for such computation. An extensive and precise analysis is performed on all of the algorithms presented.

It should be noted that Rubin and Silverberg [8] proposed an alternative explicit representation for the fields $GF(p^2)$, $GF(p^3)$, and $GF(p^6)$ using algebraic tori which achieves the same representation savings as LUC, GH, and XTR respectively. However, their representation has additional functionality, such as being able to perform finite field multiplications. It appears that their results can be extended to work in the field $GF(p^{10})$. Indeed, the existence of such an extension is based on a conjecture of Voskresenskii [11] which was proven true for the algebraic torus T_{10}. However, an explicit formulation has yet to be determined. In addition, the computationally efficiency of their representation has yet to be studied.

This paper is organized as follows. Section 2 discusses the representation of elements and cryptographic operations in $GF(p^{10})$. Section 3 discusses the key agreement and signature schemes along with their efficiency and security. Section 4 presents the building block finite field representations and an analysis of their operations. Section 5 gives a background on the theory of fifth-order characteristic sequences. Section 6 lists several elementary algorithms, including one for an important cryptographic operation, and analyses their computational cost. Section 7 presents and analyses the algorithms to perform the other critical cryptographic operation. Section 8 discusses the relationship between this field and supersingular elliptic curves while a summary and list of future work is given in section 9.

2 Cryptographic Aspects of $GF(p^{10})$

2.1 Representation of $GF(p^{10})$ Elements

Let us first describe a method to represent elements in $GF(p^{10})$ with fewer bits. See the work of Bosm, Hutton, and Verheul [1] for the background theory and proofs.

$GF(p^{10})^*$ has a unique subgroup of order $p^4 - p^3 + p^2 - p + 1$. Let Q be a divisor of this number and let $\gamma \in GF(p^{10})$ be an element of order Q. Observe that no power of γ besides 1 is contained in a proper subfield of $GF(p^{10})$.

For any $0 \leq k < Q$, the minimal polynomial of γ^k over $GF(p^2)$ is

$$f_{\gamma^k}(x) = x^5 - a_k x^4 + b_k x^3 - b_k^p x^2 + a_k^p x - 1$$

where

$$a_k = Tr(\gamma^k) = \sum_{i=0}^{4} \gamma^{kp^{2i}}$$

and

$$b_k = Tr(\gamma^{k(1+p)} + \gamma^{k(1+p^2)}) = \sum_{0 \leq i < j \leq 4} \gamma^{kp^{2i} + kp^{2j}}$$

where Tr is the trace function from $GF(p^{10})$ to $GF(p^2)$.

Thus, we can represent γ^k by the pair (a_k, b_k) of elements in $GF(p^2)$. We shall say that (a_k, b_k) represents γ^k throughout the rest of this paper. We remark

here that (a_k, b_k) also represents the other four conjugates of γ^k, namely $\gamma^{kp^{2i}}$ for $i = 1, 2, 3, 4$. However, this is not a large concern for implementation purposes.

Observe that this representation is $\frac{2}{5}$ the size of the canonical representation in $GF(p^{10})$. It should be noted that the ratio of $\frac{2}{5}$ is better than the ratios of $\frac{1}{2}$ by LUC [9], and $\frac{2}{3}$ by GH [4], but not as compact as $\frac{1}{3}$ by XTR [6].

2.2 Cryptographic Operations and Analysis

The key agreement and signature schemes proposed make use of the following two cryptographic operations which we shall call *Type 1* and *Type 2* operations respectively.

1. Calculate a_{k+l} from the sets $\{a_{k-2}, a_{k-1}, a_k, a_{k+1}, a_{k+2}\}$ and $\{a_{l-2}, a_{l-1}, a_l, a_{l+1}, a_{l+2}\}$
2. Calculate (a_{kl}, b_{kl}) from (a_k, b_k) and l where $0 \leq k, l < Q$

Thus, determining the efficiency of these operations is paramount to determining the efficiency of the schemes.

We will conduct an extensive analysis of these operations using the cryptographic algorithms presented in section? The measurement of efficency for these operations is the number of operations in $GF(p)$ which they require.

The $GF(p)$ operations we shall be concerned with are addition, multiplication, and scalar multiplication. Scalar multiplication differs from regular multiplication in that one of the operands is fixed. The cases two are separated since, in some implementations, scalar multiplications in $GF(p)$ can be made faster than regular multiplications. We also shall count squarings in $GF(p)$ in our analysis, but these will eventually be converted to multiplications under the convention used in [2, 6] that a squaring in $GF(p)$ requires 80% of the cost of a multiplication.

We shall regard subtraction the same as addition and shall not count negation. Finally, we shall take the convention that multiplication by 2 and 3 counts as 1 and 2 additions respectively.

3 Cryptographic Schemes

We now present the key agreement and signature schemes.

3.1 Diffie-Hellman Key Agreement

Domain Parameters: $p, Q, (a_1, b_1)$

Alice:

1. Chooses random private key l, $0 \leq l < Q$
2. Computes public key (a_l, b_l) and transmits this to Bob
3. Receives Bob's public key (a_k, b_k)
4. Computes the shared secret (a_{kl}, b_{kl})

Bob performs the symmetric operation. Since all of the above elements are simply compact representations of finite field elements, they inherit the difficulty of the discrete log and Diffie-Hellman problems of the finite field $GF(p^{10})$, which are believed computationally infeasible to solve.

Observe that Alice and Bob transmit only 40% as many bits as they would have to if canonical representations were used.

3.2 ElGamal-Like Signature Scheme

Domain parameters: $p, Q, (a_1, b_1)$, Q prime
Private Key: w, $0 \leq w < Q$
Public Key: $(a_{w-2}, a_{w-1}, a_w, a_{w+1}, a_{w+2})$

Signature Generation:

1. Hash the message M to obtain $h(M)$
2. Choose random k and compute (a_k, b_k)
3. Let r be the first coefficient of $a_k + b_k$ modulo Q
4. Calculate $s \equiv k^{-1}(h(M) + wr) \pmod{Q}$
5. The signature is (a_k, b_k, s).

Signature Verification:

1. Compute r and $l = h(M)r^{-1} \pmod{Q}$
2. Compute first $(a_{l-2}, \ldots, a_{l+2})$ and then compute $u = a_{w+l}$
3. Compute $(a_{ksr^{-1}}, b_{ksr^{-1}})$ and let $v = a_{ksr^{-1}}$
4. If $u = v$ accept, otherwise reject.

3.3 Efficiency of the Schemes

The Diffie-Hellman key agreement scheme requires transmission of only $4 \log p$ bits, only 40% the bandwidth of a canonical element in $GF(p^{10})$. Also, Alice and Bob must each perform only one Type 2 operation each.

For the signature scheme, the public key is the same size as an element in the canonical representation in $GF(p^{10})$. However, the this can be reduced somewhat at the cost of some simple precomputation by doing the following?

The signatures themselves, however, only involve the reduced representations, resulting in significant bandwidth savings. In particular, the signature requires only $4 \log p + \log Q$ bits.

Efficiency-wise, in addition to hashing and some simple modulo Q arithmetic, signature generation requires only one Type 2 cryptographic operation. Signature verification, on the other hand requires one Type 1 and two Type 2 operations.

3.4 Security of the Schemes

As for security, in [3] it is shown that the Diffie-Hellman problem in this representation is equivalent to the Diffie-Hellman problem in the canonical representation of $GF(p^{10})$.

For the signature scheme, it is clear that if we can solve discrete logs in $GF(p^{10})^*$, then forged signatures can be created. However, forgeries can also be created if we can solve the following problem.

Definition 1. *Given $\gamma \in GF(p^{10})$ and an element $s \in GF(p^2)$, the problem of determining an index l with $0 \leq l < Q$ such that $Tr(\gamma^l) = s$ is called* ***The Trace Discrete Logarithm Problem (DLP)****.*

The difficulty of the Trace-DLP will be investigated in the future.

4 Representation and Computation of Finite Fields

4.1 Representation and Computation of Elements in $GF(p^2)$

The following representation was originally proposed for fast arithmetic of elements in $GF(p^2)$ by Lenstra and Verheul [6]. Let $p \equiv 2 \pmod 3$ be prime. Then $x^2 + x + 1$ is irreducible modulo p. Let α be a root of this polynomial. The pair $\{\alpha, \alpha^2\}$ is a basis for $GF(p^2)$ over $GF(p)$ and we may represent an element β as $\beta = c_1\alpha + c_2\alpha^2$. Note that an element $c \in GF(p)$ is represented as $(-c)\alpha + (-c)\alpha^2$ in this basis.

Since $p \equiv 2 \pmod 3$, $\alpha^p = \alpha^2$ and so $\beta^p = c_1\alpha^p + c_2\alpha^{2p} = c_2\alpha + c_1\alpha^2$. Hence, $\{\alpha, \alpha^2\}$ is a normal basis and so taking a p-th power requires no computation.

Let $\delta = d_1\alpha + d_2\alpha^2$ and $\zeta = e_1\alpha + e_2\alpha^2$, and $d \in GF(p)$. Table 1 lists the basic operations in $GF(p^2)$, the way they are calculated using $GF(p)$ operations, and the number of $GF(p)$ operations needed. For scalar multiplications, β is fixed. Note the special operation called joint multiplication.

Operation	Type	Representation	# Add	# Mult	# Sc Mult
Addition	$\beta + \delta$	$(c_1 + d_1)\alpha + (c_2 + d_2)\alpha^2$	2	-	-
Squaring	β^2	$c_2(c_2 - 2c_1)\alpha + c_1(c_1 - 2c_2)\alpha^2$	2	2	-
Multiplication	$\beta\delta$	$[(c_2 - c_1)(d_2 - d_1) - c_1d_1]\alpha$	4	3	-
Scalar Mult		$+[(c_2 - c_1)(d_2 - d_1) - c_2d_2]\alpha^2$	3	-	3
$GF(p)$ Mult	$d\beta$	$(dc_1)\alpha + (dc_2)\alpha^2$	-	2	-
Norm	$\beta\beta^p$	$[-(b_1 - b_2)^2 - b_1b_2]\alpha + [-(b_1 - b_2)^2 - b_1b_2]\alpha^2$	2	1.8	-
Joint Mult	$\beta\delta + \beta^p\zeta$	$[b_1(c_2 - d_2 - c_1) + b_2(d_2 - d_1 - c_2)]\alpha$	10	4	-
Joint Scal Mult		$+[b_1(d_1 - d_2 - c_1) + b_2(c_1 - d_1 - c_2)]\alpha^2$	10	-	4

Table 1. Cost of Operations in $GF(p^2)$ over $GF(p)$

4.2 Operations in $GF(q^5)$

We now analyze the amount of $GF(q)$ operations needed to add, square, and mutliply elements in $GF(q^5)$ when represented in a polynomial basis over $GF(q)$, where q is a prime power. Multiplication and squaring is a variant of Karatsuba multiplication [5]. We remark that we do not account for polynomial reduction. This is accounted for in the next subsection.

We shall represent elements β and δ in $GF(q^5)$ as $\beta = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$ and $\delta = d_0 + d_1x + d_2x^2 + d_3x^3 + d_4x^4$.

Addition is straightforward and requires 5 $GF(q)$ additions. For multiplication, define $u_m = c_m d_m$ for all $m = 0, 1, 2, 3, 4$ and

$$v_{i,j} = c_i d_j + c_j d_i = (b_i + b_j)(d_i + d_j) - u_i - u_j, \forall 0 \leq i < j \leq 4$$

Multiplication then becomes

$$\beta\delta = u_0 + v_{0,1}x + (u_1 + v_{0,2})x^2 + (v_{0,3} + v_{1,2})x^3 + \cdots + u_4x^8$$

For squaring, we just let $\beta = \delta$. Table 2 lists the operations involved and their cost in terms of $GF(q)$ operations.

Operation	# Add	# Sq	# Mult
Addition	5	-	-
Squaring	36	15	-
Multiplication	46	-	15

Table 2. Cost of Operations in $GF(q^5)$ over $GF(q)$

Example 1. Take $q = p^2$ with $p \equiv 2 \pmod 3$, and use the representation of $GF(p^2)$ from section 4.1. Table 3 lists $GF(p^{10})$ operations with the number of $GF(p)$ operations required.

Operation	# Add	# Sq	# Mult
Addition	10	-	-
Squaring	102	-	30
Multiplication	152	-	45

Table 3. Cost of Operations in $GF(p^{10})$ over $GF(p^2)$

4.3 Polynomial Reduction

After multiplication or squaring is performed in $GF(q^5)$, it is necessary to reduce the result back to the form $c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4$. If the field has been derived using the irreducible polynomial $x^5 + e_4x^4 + e_3x^3 + e_2x^2 + e_1x + e_0$, then this is done by using the relation $x^n = -e_4x^4 - e_3x^3 - e_2x^2 - e_1x - e_0$ a total of 4 times. Each step requires a total of 5 scalar multiplications and 5 additions in $GF(q)$. Hence, the total for a reduction is 20 scalar multiplications and 20 additions.

However, if we specialize the polynomial, we can reduce the cost. In particular, if a polynomial of the form $x^5 - x^k - 1$ is used for some $1 \leq k < 5$, the computational cost amounts to two additions per step, hence a total of 10 additions for each reduction.

4.4 Exponentiation in a Finite Field

For a finite field element γ^k and an integer l, we shall calculate γ^{kl} using the standard square-and-multiply algorithm based on the binary representation of l. For general l, we shall assume its binary representation consists of equal zeroes and ones. Hence the square-and-multiply algorithm will require $\log k$ squarings and $\frac{1}{2}\log l$ multiplications where logarithms are taken to the base 2.

Using table 3, we see that calculating γ^{kl} from γ^k shall take $52.5 \log l$ multiplications and $178 \log l$ additions in $GF(p)$ (not counting reduction).

4.5 Computing (a_k, b_k) from γ^k

Suppose we have γ^k and wish to compute the pair (a_k, b_k). This essentially amounts to computing the conjugates of γ^k, namely $\gamma^{kp^2}, \gamma^{kp^4}, \gamma^{kp^6}, \gamma^{kp^8}$. We can do this by first calculating the values $p^2, p^4, p^6, p^8 \bmod Q$ and then exponentiate γ^k to these powers. This may be useful when Q is significantly less than p^2, which it will be in practice. Assuming these are of the same order as Q and that their binary representations have the same number of zeroes as ones, it would take approximately $205 \log Q$ multiplications and $712 \log Q$ additions in $GF(p)$, not counting reduction.

5 The Fifth-Order Characteristic Sequences over $GF(p^2)$

The new algorithm we present for computing in the compact representation of $GF(p^{10})$ requires the theory of fifth-order characteristic sequences. This section will introduce the necessary theory. A more thorough exposition is given in [3].

5.1 Definition of Characteristic Sequences

Consider the sequence $\{s_l\}$ of elements in $GF(p^2)$ generated from the linear recurrence

$$s_{l+5} = as_{l+4} - bs_{l+3} + b^p s_{l+2} - a^p s_{l+1} + s_l \tag{1}$$

where $a, b \in GF(p^2)$. If $f(x) = x^5 - ax^4 + bx^3 - b^p x^2 + a^p x - 1$ is irreducible over $GF(p^2)$, then the sequence generated with initial conditions $s_0 = 5$, $s_1 = a$, $s_2 = a^2 - 2b$, $s_3 = a^3 - 3ab + 3b^p$, $s_4 = s_2^2 - 2b^2 - 4a^p + 4b^p$ is called the **fifth-order characteristic sequence of f over $GF(p^2)$**.

This sequence has the property that if γ is a root of f in $GF(p^{10})$, then

$$s_l = \gamma_0^l + \gamma_1^l + \gamma_2^l + \gamma_3^l + \gamma_4^l$$

for all l where $\gamma_i = \gamma^{p^{2i}}$ for $i = 0, 1, 2, 3, 4$. The period of $\{s_l\}$ is equal to the order of γ. By s_{-l} we mean s_{P-l} where P is the period of the sequence. It is also true that $s_{-l} = s_l^p$ for all l.

There is a secondary sequence $\{t_l\}$ defined as

$$t_l = \sum_{0 \le i < j \le 4} \gamma_i^l \gamma_j^l$$

5.2 Properties of Characteristic Sequences

The sequences $\{s_l\}$ and $\{t_l\}$ have the following properties.

Lemma 1. *For all n, m,*

1. $s_{2n} = s_n^2 - 2t_n$
2. $t_{2n} = t_n^2 + 2s_n^p - 2s_n t_n^p$
3. $s_{3n} = s_n^3 - 3s_n t_n + 3t_n^p$
4. $t_{3n} = t_n^3 - 3s_n^p t_n - 3s_n t_n t_n^p + 3s_n^2 s_n^p + 3t_n^{2p} - 3s_n$
5. $s_{n+m} = s_n s_m - s_{n-m} t_m + s_{n-2m} t_m^p - s_{n-3m} s_m^p + s_{n-4m}$
6. $t_n t_m - s_m^p t_{n-m} + 3t_{n+m} = s_n s_m s_{n+m} - s_{n-2m} s_{n-m} + s_{2n-3m} - s_{n+2m} s_n - s_{2n+m} s_m + s_{n+m}^2$

Proof.

$$s_n^2 = \sum_{i,j=0}^{4} \gamma_i^n \gamma_j^n = \sum_{i=0}^{4} \gamma_i^{2n} + 2 \sum_{0 \le i < j \le 4} \gamma_i^n \gamma_j^n = s_{2n} - 2t_n$$

which proves 1. The rest can similarly be proven. □

5.3 Calculating the k-th Term of a Sequence

We now present an algorithm for calculating the k-th term of a sequence. We use an analogue to square-and-multiply which will be referred to as *triple-and-add/subtract*. It works on the same premise as square-and-multiply except that we are tripling instead of doubling. Let us first make a few observations.

Suppose we are given the pair of 5-tuples $(s_{l-2}, s_{l-1}, s_l, s_{l+1}, s_{l+2})$ and $(t_{l-2}, t_{l-1}, t_l, t_{l+1}, t_{l+2})$. Then from these elements, we can compute the sequence terms $s_{3l-3}, s_{3l-2}, \ldots, s_{3l+3}$ and $t_{3l-3}, t_{3l-2}, \ldots, t_{3l+3}$. Explicit formulae will be given in section ? when these computations are analyzed. These formulae are derived simply by substituting appropriate values for n and m in Lemma 1.

To compute using triple-and-add/subtract, we look at the ternary expansion of k with the indices $\{-1, 0, 1\}$. That is, we find an expansion $c_n c_{n-1} \cdots c_0$ where $c_i \in \{-1, 0, 1\}$ and $k = \sum_{i=0}^{n} c_i 3^i$.

We combine these steps into the following algorithm.

Algorithm 1: Computing the k-th Term of a Sequence

1. Set $l = 1$ and $s = (s_{-1}, s_0, s_1, s_2, s_3)$ and $t = (t_{-1}, t_0, t_1, t_2, t_3)$
2. For $i = 0, \ldots n$
   (a) If $c_i = -1$, then $s = (s_{3l-3}, s_{3l-2}, s_{3l-1}, s_{3l}, s_{3l+1})$, and $t = (t_{3l-3}, t_{3l-2}, t_{3l-1}, t_{3l}, t_{3l+1})$
   (b) If $c_i = 0$, then $s = (s_{3l-2}, s_{3l-1}, s_{3l}, s_{3l+1}, s_{3l+2})$, and $t = (t_{3l-2}, t_{3l-1}, t_{3l}, t_{3l+1}, t_{3l+2})$
   (c) If $c_i = 1$, then $s = (s_{3l-1}, s_{3l}, s_{3l+1}, s_{3l+2}, s_{3l+3})$, and $t = (t_{3l-1}, t_{3l}, t_{3l+1}, t_{3l+2}, t_{3l+3})$
   (d) $l = 3l + c_i$
3. Output $(a_k, b_k) = (s_k, t_k)$.

Steps 2(a), 2(b), and 2(c) will be referred to as triple-and-subtract, triple, and triple-and-add respectively.

6 Elementary Algorithms

6.1 The Type 1 Cryptographic Operation

Recall that the Type 1 cryptographic operation was defined as calculating a_{k+l} from the sets $\{a_{k-2}, a_{k-1}, a_k, a_{k+1}, a_{k+2}\}$ and $\{a_{l-2}, a_{l-1}, a_l, a_{l+1}, a_{l+2}\}$. We may use the theory of fifth-order characteristic sequences presented in the previous section to aid us.

Using the a_i's in the place of s_i's in the relation (1), we can derive that

$$\begin{bmatrix} a_{u-1} & a_u & a_{u+1} & a_{u+2} & a_{u+3} \end{bmatrix} = \begin{bmatrix} a_{u-2} & a_{u-1} & a_u & a_{u+1} & a_{u+2} \end{bmatrix} A$$

for all u where A is the 5×5 matrix given by

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & -a_1^p \\ 0 & 1 & 0 & 0 & b_1^p \\ 0 & 0 & 1 & 0 & -b_1 \\ 0 & 0 & 0 & 1 & -a_1 \end{bmatrix}$$

We can extend this to the relation

$$\begin{bmatrix} a_{u+v-2} & a_{u+v-1} & a_{u+v} & a_{u+v+1} & a_{u+v+2} \end{bmatrix} = \begin{bmatrix} a_{u-2} & a_{u-1} & a_u & a_{u+1} & a_{u+2} \end{bmatrix} A^v$$

for all u and v.

Now let the 5×5 matrix M_u be defined as

$$M_u = \begin{bmatrix} a_{u-4} & a_{u-3} & a_{u-2} & a_{u-1} & a_u \\ a_{u-3} & a_{u-2} & a_{u-1} & a_u & a_{u+1} \\ a_{u-2} & a_{u-1} & a_u & a_{u+1} & a_{u+2} \\ a_{u-1} & a_u & a_{u+1} & a_{u+2} & a_{u+3} \\ a_u & a_{u+1} & a_{u+2} & a_{u+3} & a_{u+4} \end{bmatrix}$$

Then, we see that $M_u = M_0 A^u$, whence $A^u = M_0^{-1} M_u$ provided that M_0 is invertible which it almost certainly will be. Hence,

$$a_{u+v} = \begin{bmatrix} a_{u-2} & a_{u-1} & a_u & a_{u+1} & a_{u+2} \end{bmatrix} C_3(M_0^{-1} M_v) \tag{2}$$

where C_3 means the third column of the matrix.

Algorithm 2: Type 1 Cryptographic Operation

Precomputation:

1. Compute the terms $a_{-4}, a_{-3}, \ldots, a_4$ from a_1
2. Form the matrix M_0
3. Compute M_0^{-1}

Computation:

1. Form the matrix M_k using $a_{k-2}, \ldots, a_{k+2}$
2. Compute a_{k+l} using equation (2)

Analysis: The precomputation stage depends only on the domain parameter a_1, and thus needs only to be done once for all computations. Observe that $a_0 = 5$ and $a_{-i} = a_i^p$. Hence, we only need compute a_2, a_3, and a_4 which can be done by using Lemma 1. The precomputation stage, thus, needs only a small constant number of operations done only once, so it will be omitted in the analysis.

In equation (2), to compute the third column of $M_0^{-1} M_k$, we only need multiply M_0^{-1} to the third column of M_k. This requires a total of 25 scalar multiplications and 20 additions in $GF(p^2)$. The vector multiplication from equation (2) requires a total of 5 multiplications and 4 additions, bringing the total to 5 multiplications, 25 scalar multiplications, and 24 additions in $GF(p^2)$. From Rable 1, this translates to 15 multiplications, 75 scalar multiplications, and 119 additions in $GF(p)$.

6.2 Root Finding

We describe in detail the root finding algorithm due to Rabin [7]. Suppose we wish to find a root γ^k, represented in $GF(p^{10})$ of $f_{\gamma^k}(x) = x^5 - a_k x^4 + b_k x^3 - b_k^p x^2 + a_k^p x - 1$ over $GF(p^2)$. All conjugates $\gamma_i^k = \gamma^{k p^{2i}}$ of f_{γ^k} satisfy $(\gamma_i^k)^{p^{10}-1} = 1$, and so they are roots of $x^{p^{10}-1} - 1 = (x^{\frac{p^{10}-1}{2}} - 1)(x^{\frac{p^{10}-1}{2}} + 1)$. We shall assume

that if we choose a random element $\zeta \in GF(p^{10})$, then the elements $\gamma_i^k + \zeta$ are randomly distributed in $GF(p^{10})$. Hence, some would be roots of $x^{\frac{q-1}{2}} + 1$ and other would be roots of $x^{\frac{q-1}{2}} - 1$.

Thus, calculating $g(x) = \gcd((x+\zeta)^{\frac{p^{10}-1}{2}} - 1, f_{\gamma^k})$, should yield a non-trivial factor of f_{γ^k}. If this were the case, either g or f_{γ^k}/g would be a polynomial of degree 1 or 2. A polynomial of degree 1 immediately gives a root γ^k while γ^k can be found from a polynomial of degree 2 by using the quadratic formula. If, on the other hand, $g(x) = 1$ or $g(x) = f_{\gamma^k}(x)$, then the algorithm fails and we should try again with a new ζ.

We can summarize this discussion by the following algorithm:

Algorithm 3: Root-Finding

1. Choose a random element $\zeta \in GF(p^{10})$
2. Compute $g(x) = \gcd((x+\zeta)^{\frac{p^{10}-1}{2}} - 1, f_{\gamma^k})$
3. If $g(x) = 1$ or $g(x) = f_{\gamma^k}(x)$, go to step 1
4. If $\deg(g) = 3, 4$, then $g(x) = f_{\gamma^k}(x)/g(x)$
5. Output a root of g, using quadratic formula if necessary

Analysis: If the roots are randomly distributed, then an iteration of this algorithm fails to produce a non-trivial factor of f_{γ^k} with probability 1/16. For simplicity, we shall assume that the algorithm always succeeds. Note also that we cannot choose $\zeta \in GF(p^2)$ and hope to get a nontrivial factor g. For if we did, g would be a polynomial over $GF(p^2)$ of degree less than 5 with γ as a root, a contradiction since f_{γ^k} is the minimal polynomial of γ^k.

Let us now determine how many $GF(p)$ operations this root finding algorithm takes. We shall assume that we are using a specialized polynomial for reduction in $GF(p^{10})$ as described in section 4.3. To calculate g, we first compute $h(x) = (x+\zeta)^{\frac{p^{10}-1}{2}} - 1$ modulo f_{γ^k}. We then compute f_{γ^k} modulo h and so on until we obtain the gcd.

Note f_{γ^k} has degree 5 and h has degree less than 5. Thus, from the point we derive h and on, only a constant number of operations is required. Hence, we shall disregard this computation and focus on the first modular exponentiation.

We may calculate $(x+\zeta)^{\frac{p^{10}-1}{2}} - 1$ modulo f_{γ^k} by a square-and-multiply algorithm reducing modulo f_{γ^k} at each step. Using the table 2, squaring takes 30 multiplications and 102 additions in $GF(p^{10})$. This translates to 1350 multiplications and 6180 additions in $GF(p)$ including reduction in $GF(p^{10})$.

A multiplication however is only by the element $x + \zeta$ and hence requires 5 multiplications and 4 additions in $GF(p^{10})$. This translates to 225 mutliplications and 820 additions in $GF(p)$.

A reduction from a squaring deals with 16 scalar multiplications by elements from $GF(p^2)$ on elements in $GF(p^{10})$. Each such scalar multiplication costs 5 scalar multiplications in $GF(p^2)$. Thus this requires a total of 240 scalar multiplications and additions in $GF(p)$. Reduction also carries an additional cost of

20 $GF(p^{10})$ additions which brings the total to 240 scalar multiplications and 440 additions in $GF(p)$.

When a multiplication occurs, the highest term before reduction is x^5. Hence, only one reduction is needed. This amounts to 4 scalar multiplications of a $GF(p^2)$ element on a $GF(p^{10})$ element. Thus, only 75 scalar multiplications and additions in $GF(p)$. An additional 5 $GF(p^{10})$ additions brings the total to 75 scalar multiplications and 125 additions in $GF(p)$.

Thus, the total cost to find a root would be $14625 \log p$ multiplications, $2775 \log p$ scalar multiplications, and $70925 \log p$ additions in $GF(p)$.

7 The Type 2 Cryptographic Operation

Recall that the Type 2 cryptographic operation was defined as calculating (a_{kl}, b_{kl}) from (a_k, b_k) and l. We now consider algorithms to perform this operation.

7.1 The Type 2 Root-Finding Algorithm

Algorithm 4: Type 2 Algorithm using Root-Finding

1. Find a root $\delta = \gamma^k$ of $x^5 - a_k x^4 + b_k x^3 - b_k^p x^2 + a_k^p x - 1$
2. Exponentiate to get δ^l
3. Calculate (a_{kl}, b_{kl}) from $\delta^l = \gamma^{kl}$ as in Section 4.5.

Analysis: Step 1 requires $14625 \log p$ multiplications, $2775 \log p$ scalar multiplications, and $70925 \log p$ additions in $GF(p)$.

Step 2 requires $52.5 \log l$ multiplications and $184 \log l$ additions in $GF(p)$.

Step 3 using the special reduction polynomial costs $205 \log Q$ multiplications and $832 \log Q$ additions in $GF(p)$

Thus the entire total cost of the first naive algorithm is $14625 \log p + 257.5 \log l$ multiplications, 2775 scalar multiplications, and $70925 \log p + 1016 \log l$ additions.

7.2 The Type 2 Polynomial Field Extension Algorithm

Algorithm 5: Type 2 Algorithm using Polynomial Field Extension

1. Use $f_{\gamma^k}(x) = x^5 - a_k x^4 + b_k x^3 - b_k^p x^2 + a_k^p x - 1$ as the reduction polynomial
2. Exponentiate γ^k to get γ^{kl}
3. Calculate (a_{kl}, b_{kl}) from γ^{kl} as in Section 4.5.

We know that f is irreducible over $GF(p^2)$ and hence can be used to extend to $GF(p^{10})$. Then we can easily represent γ^k in this polynomial basis since γ^k is a part of the basis.

Analysis: Step 1 requires no computation. For step 2, we must take into account polynomial reduction. This involves 16 scalar multiplications and 20 additions in $GF(p^2)$. Hence, step 2 costs is $52.5 \log l$ multiplications, $24 \log l$ scalar multiplications, and $208 \log l$ additions in $GF(p)$.

The total cost of step 3 is $205 \log Q$ multiplications, $96 \log Q$ scalar mutliplications, and $832 \log Q$ additions.

Since $0 \leq l < Q$, we shall assume that $\log l$ is roughly the same as $\log Q$. Thus, the total cost of this algorithm is $257.5 \log l$ multiplications, $120 \log l$ scalar multiplications, and $1040 \log l$ additions.

7.3 The Type 2 Sequence Algorithm

We now propose an algorithm using sequences to calculate the term (a_{kl}, b_{kl}) from (a_k, b_k) and l. This algorithm does not need to extend to $GF(p^{10})$.

Algorithm 6: Type 2 Algorithm using Fifth-Order Characteristic Sequences

Let $s_i = a_{ki}$ and $t_i = b_{ki}$ for all i.

1. Let $l = \sum_{i=0}^{n} c_i 3^i$ where $c_i \in \{-1, 0, 1\}$
2. Set $m = 1$ and $u = (s_{-1}, s_0, s_1, s_2, s_3)$ and $v = (t_{-1}, t_0, t_1, t_2, t_3)$
3. For $i = 0, \ldots n$
   (a) Compute $u = (s_{3m-2+c_i}, \ldots, s_{3m+2+c_i})$ and $v = (t_{3m-2+c_i}, \ldots, t_{3m+2+c_i})$
   (b) $m = 3m + c_i$
4. Output $(a_{kl}, b_{kl}) = (s_m, t_m)$.

Analysis: All computation is performed in Step 3. The key part of the analysis is determining the cost of each sequence term in a particular iteration. We will separate the analysis into three stages: precomputation, calculation of s terms, and calculation of t terms. Terms will be grouped together where formulae are similar. The symbol (sc) appearing in tables shall refer to scalar operations.

Precomputation: The precomputation stage calculates terms which are pervasive in the computation of the s and t terms.

Table 4 shows the equations used to compute various terms in precomputation.

Term	Formula
s_{l+3}	$s_{l+3} = as_{l+2} - bs_{l+1} + b^p s_l - a^p s_{l-1} + s_{l-2}$
s_{2l}	$s_{2l} = s_l^2 - 2t_l$
t_{2l}	$t_{2l} = t_l t_l - 2t_l^p s_l + 2s_l^p$
s_{4l+2}	$s_{4l+2} = s_{3l+2}s_l - s_{2l+2}t_l + s_{l+2}t_l^p - s_2 s_l^p + s_{l-2}$

Table 4. Formulae for Precomputation Terms

Table 5 lists all of the terms grouped by likeness and the costs involved in computing them.

Terms	$GF(p^2)$				$GF(p)$	
	# Add	# Sq	# Mult	# Jt Mult	# Add	# Mult
s_{l+3}, s_{3l-3}	2	-	-	2 (sc)	24	8 (sc)
$s_{2l}, s_{2l+2}, s_{2l-2}, s_{2l+4}$ $s_{2l-4}, s_{4l}, s_{4l+4}, s_{4l-4}$	2	1	-	-	6	2
$t_{2l}, t_{2l+2}, t_{2l-2}$	3	-	-	1	16	4
s_{4l+2}, s_{4l-2}	2	-	-	2	24	8

Table 5. Cost of Precomputation Terms

Computation of s terms: We now explicitly represent the s terms as shown in table 6 and examine its cost in table 7.

Term	Formula
s_{3l}	$s_{3l} = s_l(s_{2l} - t_l) + 3t_l^p$
s_{3l+1}	$s_{3l+1} = s_{2l}s_{l+1} - s_{l-1}t_{l+1} + s_2^p t_{l+1}^p - s_{l+3}^p s_{l+1}^p - s_{2l+4}$
s_{3l+2}	$s_{3l+2} = s_{2l+2}s_l - s_{l+2}t_l + s_2 t_l^p - s_{l-2}^p s_l^p - s_{2l-2}^p$

Table 6. Formulae for $\{s\}$ Sequence Terms

Terms	$GF(p^2)$				$GF(p)$	
	# Add	# Sq	# Mult	# Jt Mult	# Add	# Mult
$s_{3l}, s_{3l+3}, s_{3l-3}$	4	-	1	-	12	3
s_{3l+1}, s_{3l-1}	2	-	-	2	24	8
s_{3l+2}, s_{3l-2}	2	-	-	2	24	8

Table 7. Cost of $\{s\}$ Sequence Terms

Computation of t terms: We now explicitly represent the t terms as shown in table 8 and examine its cost in table 9.

Total Cost: Let us now examine the total number of operations required to calculate (a_{kl}, b_{kl}) from (a_k, b_k) and l using this algorithm.

All precomputation terms must be computed. This totals 44 multiplications, 8 scalar multiplications, and 192 additions in $GF(p)$.

Term	Formula
t_{3l}	$t_{3l} = t_l(t_{2l} + s_l^p) + s_l(3s_l s_l^p - t_l t_l^p - 9) + 3t_{2l}^p$
t_{3l+1}	$t_{3l+1} = [s_{l+1}(s_{2l}s_{3l+1} - s_{4l}s_{l+1} - s_{l-3}s_{l+1}^p + s_{3l-1}t_{l+1} - s_{2l-2}t_{l+1}^p - s_4^p) + s_{l+1}^p t_{l-1}$ $-s_2^p s_{l-1} + s_{l-3} - s_{4l+2}s_{2l} + s_{3l+1}^2 - t_{2l}t_{l+1}]/3$
t_{3l+2}	$t_{3l+2} = [s_{l+2}(s_{2l}s_{3l+2} - s_{4l+2}s_l - s_{l+2}s_l^p + s_{3l+2}t_l - s_{2l+2}t_l^p - s_2) - s_{3l-2}^p s_{l-2}^p$ $-s_{4l+4}s_{2l} + t_{l-2}^p s_{2l}^p + s_{4l-4}^p + s_{3l+2}^2 - t_{2l}t_{l+2}]/3$

Table 8. Formulae for $\{t\}$ Sequence Terms

Terms	$GF(p^2)$						$GF(p)$		
	# Add	# Sq	# Mult	# Jt Mult	# $GF(p)$ Mult	# Norm	# Add	# Sq	# Mult
$t_{3l}, t_{3l+3}, t_{3l-3}$	9	-	1	-	1	2	26	2	7
t_{3l+1}, t_{3l-1}	8	1	3 + 1 (sc)	3	1 (sc)	-	63	-	23 + 5 (sc)
t_{3l+2}, t_{3l-2}	7	1	4	3	1 (sc)	-	62	-	26 + 2 (sc)

Table 9. Cost of $\{t\}$ Sequence Terms

The s and t terms required for a given step depend upon which triple method is used. In the case of "triple" all but those with indices $3l-3$ and $3l+3$ are needed. This makes for a total of 35 multiplications, 108 additions for the s terms and 105 multiplications, 2 squarings, 14 scalar multiplications, and 276 additions for the t terms.

In the case of "triple-and-add" all but those with indices $3l-3$ and $3l-2$ are needed. This makes for a total of 30 multiplications, 96 additions for the s terms and 86 multiplications, 4 squarings, 12 scalar multiplications, and 240 additions for the t terms.

In the case of "triple-and-subtract", all but those with indices $3l+3$ and $3l+2$ are needed. This case gives identical cost to triple-and-add.

In the course of computation, each triple step will be called, on average, one third of the time, a total of $\frac{1}{3}\log_3 l$ times each. Thus, the average computation cost is $169.3\log_3 l$ multiplications, $3.3\log_3 l$ squarings, $20.7\log_3 l$, scalar multiplications, and $444\log_3 l$ additions in $GF(p)$.

Using the fact that $\log_2 l = 1.585\log_3 l$ and using the fact a squaring is 80% of a multiplication, this becomes approximately $108.5\log l$ multiplications, $13\log l$ scalar multiplications, and $280.1\log l$ additions.

The computational results of this algorithm and the two naive algorithm are shown in table 10.

Algoithm	# Add	# Mult	# Sc Mult
Naive 1	$70925\log p + 1016\log l$	$14625\log p + 257.5\log l$	$2775\log p$
Naive 2	$1040\log l$	$257.5\log l$	$120\log l$
Fifth-Order	$280.1\log l$	$108.5\log l$	$13\log l$

Table 10. Total Average Operational Cost of the Three Algorithms

8 Supersingular Elliptic Curves

In [10], it was shown that a supersingular elliptic curve can be embedded into the XTR group by using the Weil Pairing. In our case, we should notice that our group order $p^4 - p^3 + p^2 - p + 1$ is outside the Hasse interval for an elliptic curve over $GF(p^4)$. However, there does exist a supersingular elliptic curve over $GF(p^5)$ whose order the multiple $p^5 + 1$. It is easily seen that the curve

$$Y^2 = X^3 + 1$$

has order $p^5 + 1$ when p and hence $p^5 \equiv 2 \pmod 3$. Using α which is a root of $\alpha^2 + \alpha + 1$, if $P = (x, y)$ is a point on the curve of order Q, then $\hat{P} = (\alpha x, y)$ is as well. For a point P of order Q, we can then define the map $\psi(P) = e_Q(P, \hat{P})$ where e_Q is the Weil pairing. This is an isomorphism from $\langle P \rangle$ into $\langle \gamma \rangle$.

9 Summary and Future Work

We have presented a new finite field-based discrete log cryptosystem in $GF(p^{10})$ which represents with only $\frac{2}{5}$ as much storage as the canonical representation. We have also given a new algorithm which computes k-th powers in this representation faster than any previously known algorithm. We have also performed a very explicit analysis describing the exact number of operations needed to exponentiate.

Rubin and Silverberg [8] presented an alternative method for representing elements in finite fields with fewer bits. They used the theory of algebraic tori to explicitly give representations for elements in $GF(p^2)$ and $GF(p^6)$. It appears that their technique may also be applied to $GF(p^{10})$, although this has not yet been done. In addition, the computational efficiency of their algorithms are still unclear. Both of these questions are currently being studied.

References

1. W. Bosma, J. Hutton, and E. Verheul. Looking beyond xtr. In *Advances in Cryptology – Asiacrypt '2002*, pages 46–63, 2002.
2. H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curves exponentiation using mixed coordinates. In *Advances in Cryptology – Asiacrypt '98*, pages 51–65, 1998.
3. K. Giuliani and G. Gong. Analogues to the gong-harn and xtr cryptosystems. Combinatorics and Optimization Research Report to appear, University of Waterloo.
4. G. Gong and L. Harn. Public-key cryptosystems based on cubic finite field extensions. *IEEE Transactions on Information Theory*, 45:2601–2605, 1999.
5. A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Physics-Daklady*, 7:595–596, 1963.
6. A. Lenstra and E. Verheul. The xtr public key system. In *Advances in Cryptology – Crypto '2000*, pages 1–19, 2000.
7. M. Rabin. Probabilistic algorithms in finite fields. *SIAM Journal of Computing*, 9:273–280, 1980.

8. K. Rubin and A. Silverberg. Torus-based cryptography. In *Advances in Cryptology – Crypto '2003*, pages 349–365, 2003.
9. P. Smith and C. Skinner. A public-key cryptosystem and a digital signature system based on the lucas function analogue to discrete logarithms. In *Advances in Cryptology – Asiacrypt '1994*, pages 357–364, 1994.
10. E. Verheul. Evidence that xtr is more secure than supersingular elliptic curve cryptosystems. In *Advances in Cryptology – Eurocrypto '2001*, pages 195–210, 2001.
11. V. E. Voskresenskii. *Algebraic Groups and their Birational Invariants*. Number 179 in Translations of Mathematical Monographs. American Mathematical Society, Providence, 1998.