FINITE FIELDS AND THEIR APPLICATIONS **2,** 348–368 (1996)
ARTICLE NO. 0022

Binary Representations of Finite Fields and Their Application to Complexity Theory*

JÜRG GANZ

Union Bank of Switzerland, CH-8021 Zürich, Switzerland

Received October 31, 1994; revised November 28, 1995

Binary representations of finite fields are defined as an injective mapping from a finite field to l-tuples with components in $\{0, 1\}$ where 0 and 1 are elements of the field itself. This permits one to study the algebraic complexity of a particular binary representation, i.e., the minimum number of additions and multiplications in the field needed to compute the binary representation. The two-way complexity of a binary representation is defined as the sum of the algebraic complexities of the binary representation and of its inverse mapping. Two particular binary representations are studied: the standard representation and the logarithmic representation. A method of surrogate computation is developed and used to deduce relationships between the algebraic complexities of certain functions. The standard representation of a finite field is shown to be among the two-way easiest representations of this field. In particular, the standard representation of a finite field with characteristic p is two-way easy whenever $p - 1$ has only small prime factors. For any finite field having a two-way easy binary representation, the algebraic complexity in this field is shown to be essentially equivalent to Boolean circuit complexity. For any finite field, the Boolean circuit complexity of Zech's (or Jacobi's) logarithm is shown to be closely related to the Boolean circuit complexity of the discrete logarithm problem that is used in public-key cryptography. © 1996 Academic Press, Inc.

1. INTRODUCTION

This paper studies properties of binary representations of general finite fields GF(q), where $q = p^n$ for some prime p and some positive integer n. As commonly understood, a "binary representation" assigns to each element in

* This work has been performed at the Signal and Information Processing Laboratory of the Swiss Federal Institute of Technology.

GF(q) a unique element in $\{0, 1\}^l$. Although such "binary representations" are often used in practice, we are not aware of any previous systematic study of them. It is crucial to our development that we consider the "0" and "1" of the binary representation to be the 0 and 1 elements of GF(q). This allows us to raise such fundamental questions as how many GF(q) operations are needed to compute a particular binary representation of GF(q).

Formally, we define a *binary representation* of GF(q) to be an injective (or "one-to-one") mapping ϕ: GF(q) $\to \{0, 1\}^l \subset \mathrm{GF}(q)^l$. We will often refer to $\phi(x)$ for $x \in \mathrm{GF}(q)$ as the binary representation of x. The *inverse mapping* τ: $\mathrm{Im}(\phi) \to \mathrm{GF}(q)$, where $\mathrm{Im}(\phi)$ denotes the image (or "range") of ϕ, is defined by $\tau(\phi(x)) = x$ for $x \in \mathrm{GF}(q)$ and will play an important role in our study. Addition and multiplication of two elements of GF(q) correspond to the operations of Add and Mult on their binary representations, where Add and Mult are functions from $\mathrm{Im}(\phi)^2$ to $\mathrm{Im}(\phi)$ defined by

$$\mathrm{Add}(\phi(x), \phi(y)) = \phi(x + y) \tag{1}$$

and

$$\mathrm{Mult}(\phi(x), \phi(y)) = \phi(x \cdot y), \tag{2}$$

respectively, for $x, y \in \mathrm{GF}(q)$.

The requirement that ϕ be injective, i.e., that each element of GF(q) have a unique binary l-tuple as its representation, implies that $l \geq \log_2 q$. One is generally interested in practice in the case where l is not much greater than $\log_2 q$, so we will usually require that $l = O(\log q)$. Of course, a given finite field has many different binary representations, which we distinguish when necessary by capital letters as subscripts, e.g., the binary representation ϕ_A and the corresponding τ_A, Add_A, Mult_A, and l_A.

In the next section, the algebraic complexity measure and some basic relations among complexity measures are introduced. Section 3 treats the standard (or "polynomial") representation ϕ_S of GF(q). This is the representation most often used in implementations of GF(q) = GF(p^n) in which every field element is represented by binary digits that specify the coefficients in GF(p) of a polynomial of degree less than n. The most interesting property of the standard representation ϕ_S is that both Add_S and Mult_S are easy to compute. The inverse mapping τ_S is always easy to compute and ϕ_S itself is known to be easy to compute for finite fields with small characteristic. In Section 3 these facts are made precise in terms of algebraic complexity, and the method of surrogate computation is introduced.

The logarithmic representation ϕ_L of GF(q) is introduced in Section 4. The element β^z in GF(q), where β is a primitive element of GF(q) and

$0 \leq z \leq q - 2$, is represented as the l_L-tuple with components in $\{0, 1\} \subset \mathrm{GF}(q)$ that is formally the same as the usual length $l_L = \lceil \log_2 q \rceil$ binary form of the integer z. The mapping ϕ_L can be viewed as the logarithm function from $\mathrm{GF}(q)$ to $\{0, 1\}^{l_L} \subset \mathrm{GF}(q)^{l_L}$. Section 4 gives also the formal definition of the logarithm function Log used in public-key cryptography [1] that maps a vector of GF(2) elements into another such vector. Further, Zech's logarithm (see [2]) is introduced and its close relation to the addition function Add_L for the logarithmic representation ϕ_L is formulated. None of these results are entirely new, but it is important for the following that they be formulated within the framework of binary representations and algebraic complexity.

The standard and the logarithmic representations of $\mathrm{GF}(q)$ are the binary representations most often used in practice. They both have the property that the inverse mapping τ is easy to compute but that no easy computation of ϕ itself is known in general. One might wonder whether these are general properties. But extending the techniques of this paper, we have shown in [3] that there is a binary representation of $\mathrm{GF}(q)$ with exactly the opposite properties, i.e., ϕ is easy to compute but no easy computation of τ is known in general. Recently, this representation has also been used in [4] to devise an efficient factoring algorithm for polynomials over every finite field.

For many applications it is important to have a binary representation ϕ of $\mathrm{GF}(q)$ such that ϕ and the inverse mapping τ are both easy to compute. A binary representation that has this property will be called *two-way* easy. The method of surrogate computation of Section 3 is applied in Section 5 to demonstrate that a binary representation ϕ is among the two-way easiest representations of $\mathrm{GF}(q)$ if both Add and Mult are easy for this representation. It follows that, for any finite field $\mathrm{GF}(q)$, the standard representation ϕ_S is among the two-way easiest representations. It must be stressed that it is not known whether or not there exist fields for which even the two-way easiest representation has at least one difficult mapping (ϕ or τ). However, the standard representation ϕ_S is two-way easy for every $q = p^n$, where $p - 1$ has only small prime factors. These results can also be used in a reverse manner: If the standard representation ϕ_S of $\mathrm{GF}(q)$ is easy, then, for any binary representation ϕ of this field, the more difficult mapping (ϕ or τ) is about as difficult to compute as the more difficult operation (Add or Mult).

In Section 6 the method of surrogate computation is used to obtain bounds on the complexity of mappings between two different binary representations of $\mathrm{GF}(q)$. It is shown how the complexity of the mapping between the standard representation ϕ_S and an arbitrary binary representation ϕ are related to the more difficult operation (Add or Mult) in ϕ. Applying this result to the logarithmic representation ϕ_L shows that, for every field $\mathrm{GF}(q)$, the complexity of the logarithm function Log used in public-key cryptography is closely related to the complexity of Add_L and therefore also to the complexity of Zech's logarithm. If the standard representation ϕ_S is easy, it follows further that ϕ_L, Log, Zech's logarithm, and the function Add_L all have about the same complexity. In particular, this holds for fields $\text{GF}(p^n)$ where $p - 1$ has only small prime factors.

Finally, in Section 7, the previous results are used to analyze the relations among various complexity measures. Lempel *et al.* [5] showed that algebraic complexity is essentially equivalent to circuit complexity when the characteristic p of $\text{GF}(q)$ is small. Indeed, it has been thought that this condition is necessary for a close relation of these complexity measures (see, e.g., [6] or [7]). We extend the result of Lempel *et al.* by showing the essential equivalence of these two complexity measures whenever there exists a two-way easy binary representation ϕ of $\text{GF}(q)$. It follows that algebraic complexity is essentially equivalent to circuit complexity whenever $p - 1$ has only small prime factors, which includes the case of some finite fields with large characteristic and even some large prime fields.

2. Some Basic Complexity Measures and Relations

To study the complexity of binary representations and other functions, we will primarily rely on the well-known concept of a straight-line algorithm (SLA) and its related complexity measure (see [5], or see [8] where an SLA is called an "Ω-Berechnug"). An SLA in $\text{GF}(q)$ of length k $(0 \leq k)$ is defined as follows:

The *input* consists of the elements $x_1, \ldots, x_a$, which can assume any value in $\text{GF}(q)$. If $k > 0$, then at each *step* i, for $i = 1, 2, \ldots, k$, the algorithm performs one specified operation on two specified operands. The operation must be either addition or multiplication in $\text{GF}(q)$. Each operand at step i must be an input element, a result of a previous step j for some $1 \leq j < i$ or a specified element of $\text{GF}(q)$. Each component of the *output* is the result of some specified step i, $1 \leq i \leq k$, a specified input element or a specified element of $\text{GF}(q)$.

Note that the choice of operands at each step in an SLA is independent of the values of the input elements and of the values computed in previous steps. We say that an SLA *computes* a function f: $S \to \text{GF}(q)^b$ where $S \subset \text{GF}(q)^a$ if, for every input $\underline{x} = [x_1, \ldots, x_a]$ in S, the output is equal to $f(\underline{x})$. Note that any constant function $f(\underline{x}) = \underline{c} \in \text{GF}(q)^b$ and the identity function $f(\underline{x}) = \underline{x}$ can be computed by an SLA of length $k = 0$.

The *algebraic complexity* $C_q(f)$ of a function f from a subset of $\text{GF}(q)^a$ to $\text{GF}(q)^b$ is the smallest length k of all the SLAs in $\text{GF}(q)$ that compute f. But any SLA in $\text{GF}(q)$ of length k that computes f can be considered as defining a $\text{GF}(q)$-*gate circuit* (i.e., a gate circuit in which only gates that perform addition or multiplication in $\text{GF}(q)$ are allowed) for computing

f having k gates that are indexed from 1 to k. One sees immediately that $C_2(f)$ is the same as *circuit complexity with respect to the basis* {AND, XOR} (see [9, p. 9] or [10, p. 24], where circuit complexity is called "combinational complexity"). In addition, it is well known that $C_2(f)$ differs by at most a factor 3 from the *unrestricted circuit complexity*, where every 2-input 1-output boolean function is in the basis. We will refer to $C_2(f)$ as the *circuit complexity* of the function f.

From counting arguments, it can be shown that almost every function f: GF(q) $\rightarrow$ GF(q) has algebraic complexity $C_q(f) \approx q$ (see [5]). Such functions are considered to be *difficult*. A function f: GF$(q)^a \rightarrow$ GF$(q)^b$ will be called *easy* if $C_q(f) = O(\log q)^i$ for some small integer i (e.g., $i = 2$). We will call a binary representation ϕ of GF(q) *two-way easy* if ϕ and τ are easy.

We will frequently be considering functions f from a subset of $\{0, 1\}^a \subset$ GF$(q)^a$ to $\{0, 1\}^b \subset$ GF$(q)^b$. For instance, the function Add of (1) is a mapping from a subset of $\{0, 1\}^{2l} \subset$ GF$(q)^{2l}$ to $\{0, 1\}^l \subset$ GF$(q)^l$, as also is the function Mult of (2). Because most hardware realizations of such functions use binary logic, it is natural to identify such a function f with its *binary equivalent* $\tilde{f}$ that maps a subset of $\{0, 1\}^a \subset$ GF$(2)^a$ to $\{0, 1\}^b \subset$ GF$(2)^b$ in the "natural" manner that, for all x in the domain of f,

$$\theta(f(\underline{x})) = \tilde{f}(\theta(\underline{x})),$$

where θ, when applied to a single element in GF(q), is the function that maps 0 and 1 of GF(q) to 0 and 1 of GF(2), respectively, and more generally when applied to an m-tuple over GF(q), for any $m \geq 2$, performs this natural mapping on each of its components. In practice, when one would want to implement f in hardware, one would in fact implement $\tilde{f}$ with binary logic. Thus, $C_2(\tilde{f})$, the circuit complexity of $\tilde{f}$, is the complexity of true practical interest, not $C_q(f)$, the algebraic complexity of f. Fortunately, $C_q(f)$ bounds $C_2(\tilde{f})$ from below in the following manner (and also bounds $C_2(\tilde{f})$ from above as we will show in Theorem 2).

LEMMA 1. *For any function f from a subset of* $\{0, 1\}^a \subset$ GF$(q)^a$ *to* $\{0, 1\}^b \subset$ GF$(q)^b$ *and its binary equivalent $\tilde{f}$, it holds that* $C_q(f) \leq 3 \cdot C_2(\tilde{f})$.

Proof. We first note that $\tilde{f}_1(x, y) = x$ AND y, a mapping from GF$(2)^2$ to GF(2) is the binary equivalent of $f_1(x, y) = x \cdot y$, a mapping from $\{0, 1\}^2 \subset$ GF$(q)^2$ to $\{0, 1\} \subset$ GF(q) for every q so that $C_2(\tilde{f}_1) = C_q(f_1) = 1$. Similarly, $\tilde{f}_2(x, y) = x$ XOR y is the binary equivalent of $f_2(x, y) = (x + (-1) \cdot y)^2$ so that $C_2(\tilde{f}_2) = 1$ but $C_q(f_2) \leq 3$. It follows that an SLA in GF(2) computing any $\tilde{f}$ can be converted to an SLA in GF(q) computing f with at most three times as many steps. ■

In several relations, the maximum complexity of Add and Mult will be important. Therefore, it is convenient to use

$$M \triangleq \max\{C_q(\text{Add}), C_q(\text{Mult})\},$$
$$\tilde{M} \triangleq \max\{C_2(\widetilde{\text{Add}}), C_2(\widetilde{\text{Mult}})\}$$

and, if necessary, the subscripts of the corresponding binary representation, e.g., M_A and $\tilde{M}_A$ for the binary representation ϕ_A. As a first application of Lemma 1, it follows that $M \leq 3 \cdot \tilde{M}$.

3. Standard Representation

For a prime field GF(p), the standard representation is the function ϕ_S: GF$(p) \rightarrow \{0, 1\}^l \subset$ GF$(p)^l$, where $l = \lceil \log_2 p \rceil$ and $\phi_S(y) = [y_{l-1}, \ldots, y_1, y_0]$ is formally the same as the usual binary form of the integer formally equal to y. For instance, for GF(7), $\phi(6) = [1, 1, 0] \in$ GF$(7)^3$.

Every element x of an extension field GF(q) = GF(p^n), $n > 1$, can be written as $x = x_0 + x_1 \cdot \alpha + \cdots + x_{n-1} \cdot \alpha^{n-1}$ with coefficients x_i in GF(p) and with α in GF(q) defining a basis $\{1, \alpha, \ldots, \alpha^{n-1}\}$ of GF(q) over GF(p). The standard representation ϕ_S: GF$(q) \rightarrow \{0, 1\}^{l_S} \subset$ GF$(q)^{l_S}$ with $l_S = n \cdot l = n \cdot \lceil \log_2 p \rceil$ is defined as

$$\begin{aligned}\phi_S(x) &= \phi_S(x_0 + x_1 \cdot \alpha + \cdots + x_{n-1} \cdot \alpha^{n-1})\\ &= [\phi_S^{(p)}(x_0), \phi_S^{(p)}(x_1), \ldots, \phi_S^{(p)}(x_{n-1})]\\ &= [[x_{0,l-1}, \ldots, x_{0,0}], \ldots, [x_{n-1,l-1}, \ldots, x_{n-1,0}]],\end{aligned}$$

where $\phi_S^{(p)}$ is the standard representation of the ground field GF(p). Note that the standard representation ϕ_S, which is defined by a polynomial basis, can easily be converted to a binary representation defined by an arbitrary basis of GF(q) over GF(p) and vice versa (for details see [3]).

The above definition of the standard representation ϕ_S does not give an explicit way to compute $\phi_S(x)$ by operations on x in GF(q). Rather, the definition shows that the inverse mapping τ_S: Im$(\phi_S) \rightarrow$ GF(q) can be computed as

$$\tau_S(\underline{x}) = \sum_{i=0}^{n-1} \alpha^i \cdot \left(\sum_{j=0}^{l-1} x_{i,j} \cdot 2^j \right),$$

where $\underline{x} = [[x_{0,l-1}, \ldots, x_{0,0}], \ldots, [x_{n-1,l-1}, \ldots, x_{n-1,0}]]$. Observe that, for fields with characteristic $p = 2$, this computation reduces to $\tau_S(\underline{x}) = \tau_S([x_0, x_1, \ldots, x_n]) = x_0 + x_1 \cdot \alpha + \cdots + x_{n-1} \cdot \alpha^{n-1}$. In all cases, the function τ_S is easy; more precisely,

$$C_q(\tau_S) \le n \cdot 2(l-1) + 2(n-1) = O(\log q). \tag{3}$$

It is well known that there are simple binary-logic circuits to compute the functions $\widetilde{\mathrm{Add}}_S$ and $\widetilde{\mathrm{Mult}}_S$ for the standard representation ϕ_S of $\mathrm{GF}(p^n)$ with $n \cdot O(\log p)$ and $n^2 \cdot O(\log p)^2$ gates, respectively, so that $C_2(\widetilde{\mathrm{Add}}_S) = O(\log q)$ and $C_2(\widetilde{\mathrm{Mult}}_S) = O(\log q)^2$. Applying Lemma 1 shows that also the functions Add_S and Mult_S are easy, i.e., $C_q(\mathrm{Add}_S) \le 3 \cdot C_2(\widetilde{\mathrm{Add}}_S) = O(\log q)$ and $C_q(\mathrm{Mult}_S) \le 3 \cdot C_2(\widetilde{\mathrm{Mult}}_S) = O(\log q)^2$. In terms of the maximum complexity of Add_S and Mult_S, this yields

$$M_S \le 3 \cdot \tilde{M}_S = O(\log q)^2. \tag{4}$$

Lempel *et al.* showed in [5] that, given any basis of $\mathrm{GF}(q)$ over $\mathrm{GF}(p)$, the coefficients in the basis expansion of an element in $\mathrm{GF}(q)$ can easily be computed by using only field operations. First, they showed that, for any basis $\{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ of $\mathrm{GF}(q)$ over $\mathrm{GF}(p)$, there is a trace-dual basis $\{\beta_0, \beta_1, \ldots, \beta_{n-1}\}$ such that

$$\mathrm{Tr}(\beta_i \cdot \alpha_j) = \begin{cases} 0, & i \ne j \\ 1, & i = j \end{cases},$$

where $\mathrm{Tr}(x) = x + x^p + x^{p^2} + \cdots + x^{p^{n-1}}$ (see [11, pp. 54–57] for properties of the trace function). Thus, the components x_i of the basis expansion $x = x_0 \cdot \alpha_0 + x_1 \cdot \alpha_1 + \cdots + x_{n-1} \cdot \alpha_{n-1}$ can be computed as $x_i = \mathrm{Tr}(\beta_i \cdot x)$ for $0 \le i \le n-1$. Applying this method of computing x_i to the standard representation ϕ_S, where the basis $\{\alpha_0, \alpha_1, \ldots, \alpha_{n-1}\}$ is in fact the standard basis $\{1, \alpha, \ldots, \alpha^{n-1}\}$, gives

$$C_q(\phi_S) \le n \cdot C_p(\phi_S^{(p)}) + 2 \cdot \lceil \log_2 p \rceil \cdot n^2. \tag{5}$$

This shows that if the standard representation $\phi_S^{(p)}$ of the ground field $\mathrm{GF}(p)$ is easy then so is ϕ_S for every extension $\mathrm{GF}(q) = \mathrm{GF}(p^n)$. For $p = 2$, $C_2(\phi_S^{(2)}) = 0$ and therefore $C_{2^n}(\phi_S) \le 2 \cdot n^2$. Because τ_S is always easy, it follows that the standard representation ϕ_S of $\mathrm{GF}(2^n)$ is two-way easy.

Given a binary representation ϕ_A of $\mathrm{GF}(q)$, we associate with the function f mapping a subset of $\mathrm{GF}(q)^a$ to $\mathrm{GF}(q)^b$ its ϕ_A-*associate* f_A that maps

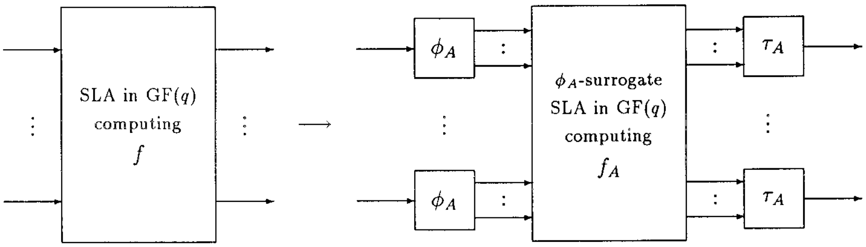
SLA in GF(q) computing f
ϕ_A
ϕ_A-surrogate SLA in GF(q) computing f_A
τ_A

FIG. 1. The structure of an SLA computing f in GF(q) and its corresponding imbedded-ϕ_A-surrogate SLA computing f in GF(q).

a subset of $\{0, 1\}^{a \cdot l_A} \subset \mathrm{GF}(q)^{a \cdot l_A}$ to $\{0, 1\}^{b \cdot l_A} \subset \mathrm{GF}(q)^{b \cdot l_A}$ in the manner that, for all $\underline{x}$ in the domain of f,

$$f_A(\underline{\phi}_A(\underline{x})) = \underline{\phi}_A(f(\underline{x}))$$

or, equivalently,

$$f(\underline{x}) = \underline{\tau}_A(f_A(\underline{\phi}_A(\underline{x}))), \tag{6}$$

where $\underline{\phi}_A$ and $\underline{\tau}_A$ when applied to an m-tuple over GF(q) performs ϕ_A and τ_A, respectively, on each of its components. Assume an SLA computing a function f in GF(q) is given. The corresponding *ϕ_A-surrogate SLA computing f_A* in GF(q) will now be defined as follows:

The inputs $x_1, \ldots, x_a$ of the original SLA are replaced by the inputs $\phi_A(x_1), \ldots, \phi_A(x_a)$ in the surrogate SLA. An addition step computing $s_i := s_j + s_k$ in the original SLA is replaced in the surrogate SLA by the sequence of steps in a specified one of the minimal-length SLAs that computes $\phi_A(s_i) := \mathrm{Add}_A(\phi_A(s_j), \phi_A(s_k))$. (Note that $\phi_A(s_j)$ and $\phi_A(s_k)$ are available as inputs, constants, or results of previous steps in the new SLA.) Similarly, a multiplicative step computing $s_i := s_j \cdot s_k$ in the original SLA is replaced in the surrogate SLA by the sequence of steps in a specified one of the minimal-length SLAs that computes $\phi_A(s_i) := \mathrm{Mult}_A(\phi_A(s_j), \phi_A(s_k))$. Finally, an output s_i in the original SLA is replaced in the surrogate SLA by the output $\phi_A(s_i)$.

Because M_A is the maximum of the algebraic complexities of Add_A and Mult_A, it follows immediately that the number of steps in the ϕ_A-surrogate algorithm is at most M_A times the number of steps in the original SLA. Thus, $C_q(f_A) \leq C_q(f) \cdot M_A$. The definition (6) of the ϕ_A-associate f_A of ϕ shows that if f can be computed by some SLA in GF(q), then f can also be computed by applying an SLA that computes $\underline{\phi}_A$ to the input, performing the ϕ_A-surrogate SLA computing f_A and applying an SLA that computes $\underline{\tau}_A$ to the output (cf. Fig. 1, where the SLAs are depicted as GF(q)-gate circuits). We call such a new SLA the *imbedded-ϕ_A-surrogate SLA computing f* in GF(q). Of course, such an imbedded surrogate SLA is a complicated way of computing f, but its consideration makes it possible to derive interesting results on the complexity of related functions.

If the SLAs computing Add_A and Mult_A in the ϕ_A-surrogate SLA computing f_A in GF(q) are replaced by minimal-length SLAs computing $\widetilde{\text{Add}}_A$ and $\widetilde{\text{Mult}}_A$, respectively, then one obtains the corresponding *ϕ_A-surrogate SLA computing the binary equivalent $\tilde{f}_A$ of f_A* in GF(2). Because $C_q(f_A) \le C_q(f) \cdot M_A$, it follows that $C_2(\tilde{f}_A) \le C_q(f) \cdot \tilde{M}_A$. We are now ready to prove the following basic result.

THEOREM 2. *For any function f mapping a subset of* $\{0, 1\}^a \subset \text{GF}(q)^a$ *to* $\{0, 1\}^b \subset \text{GF}(q)^b$ *and its binary equivalent $\tilde{f}$, it holds that*

$$\tfrac{1}{3} \cdot C_q(f) \le C_2(\tilde{f}) = O(\log q)^2 \cdot C_q(f).$$

Proof. We first define the mapping h_1 from $\{0, 1\} \subset \text{GF}(q)$ to $\{0, 1\}^{l_S} \subset \text{GF}(q)^{l_S}$ as the restriction of ϕ_S to the inputs 0 and 1 in GF(q). The definition of the standard representation ϕ_S shows that all the components of $\phi_S(0)$ and $\phi_S(1)$ are 0, except for the component $x_{0,0}$ of $\phi_S(1)$, which is 1. It follows that h_1 can be computed by an SLA of length 0 that assigns the input x to the output component $x_{0,0}$ and 0 to all the other output components. Therefore, $C_q(h_1) = 0$ and $C_2(\tilde{h}_1) = 0$, where $\tilde{h}_1$ is the binary equivalent of h_1. Similarly, the inverse mapping h_2 from $\{\phi_S(0), \phi_S(1)\} \subset \text{Im}(\phi_S)$ to $\{0, 1\} \subset \text{GF}(q)$ that is the corresponding restriction of τ_S fulfills $C_q(h_2) = 0$ and $C_2(\tilde{h}_2) = 0$.

The above considerations of the ϕ_S-surrogate SLA computing $\tilde{f}_S$ in GF(2) together with inequality (4) show that $C_2(\tilde{f}_S) \le C_q(f) \cdot \tilde{M}_S = C_q(f) \cdot O(\log q)^2$. Further, $\tilde{f}$ can be computed by applying to each component of the input the mapping $\tilde{h}_1$, performing the ϕ_S-surrogate SLA computing $\tilde{f}_S$ and applying $\tilde{h}_2$ at the output to produce the needed number of outputs. Because the computation $\tilde{h}_1$ and $\tilde{h}_2$ is free, $C_2(\tilde{f}) = C_2(\tilde{f}_S)$ and the upper bound of the theorem follows. The lower bound is a direct consequence of Lemma 1. ■

Theorem 2 shows in particular that the algebraic complexity of the functions Add and Mult for an arbitrary representation ϕ of GF(q) are closely related to the circuit complexity of their binary equivalents $\widetilde{\text{Add}}$ and $\widetilde{\text{Mult}}$, respectively. In terms of the complexities of the more complex function, it follows that

$$\tfrac{1}{3} \cdot M \le \tilde{M} = O(\log q)^2 \cdot M \tag{7}$$

for every binary representation ϕ of GF(q).

4. Logarithmic Representation

Every element x in $\mathrm{GF}(q)\backslash\{0\}$ can be written as $x = \beta^z$, where β is a primitive element of $\mathrm{GF}(q)$ and z is an integer between 0 and $q - 2$. The logarithmic representation is the function ϕ_L: $\mathrm{GF}(q) \to \{0, 1\}^{l_L} \subset \mathrm{GF}(q)^{l_L}$, where $l_L = \lceil \log_2 q \rceil$, $\phi_L(0) = [1, 1, \ldots, 1]$ and, for $x \neq 0$, $\phi_L(x) = \phi_L(\beta^z) = [z_{l_L-1}, \ldots, z_1, z_0] = \underline{z}$ is formally the same as the usual binary form of the integer z. Although this representation depends on β, its choice is not crucial for our purposes.

The above definition of the logarithmic representation ϕ_L describes a way of computing its inverse τ_L. Because $\tau_L(\underline{z}) = \beta^z$ for $\underline{z} \neq [1, \ldots, 1]$, it follows that the inverse mapping τ_L: $\mathrm{Im}(\phi_L) \to \mathrm{GF}(q)$ can be computed as

$$\begin{aligned}\tau_L(\underline{z}) &= \tau_L([z_{l_L-1}, \ldots, z_1, z_0]) \\ &= \prod_{i=0}^{l_L-1} ((\beta^{2^i} - 1) \cdot z_i + 1) + (-\beta^{2^{l_L-1}}) \cdot \prod_{i=0}^{l_L-1} z_i,\end{aligned}$$

where the second product is included to ensure that $\tau_L([1, \ldots, 1]) = 0$. From the obvious SLA realizing this sum of two products, it follows that

$$C_q(\tau_L) \leq 4 \cdot l_L = O(\log q), \tag{8}$$

so that the inverse mapping τ_L is easy. The definition of the logarithmic representation ϕ_L shows also that the multiplication of two non-zero field elements given in this representation is formally an addition of two integers given in the usual binary form modulo $q - 1$. Because integer addition modulo $q - 1$ can be realized by binary-logic circuits of size $O(\log q)$, we have

$$C_2(\widetilde{\mathrm{Mult}}_L) = O(\log q). \tag{9}$$

The function ϕ_L maps a non-zero field element $x = \beta^z$ into the l_L-tuple $[z_{l_L-1}, \ldots, z_1, z_0] \in \{0, 1\}^{l_L} \subset \mathrm{GF}(q)^{l_L}$ that is formally the same as the usual binary form of the integer z. Thus, ϕ_L is closely related to the logarithm function that maps the same field element x to the integer z, $0 \leq z < q - 1$, which is referred to as the *discrete logarithm*. Pohlig and Hellman [12] proposed an algorithm that computes the discrete logarithm. They showed that computing the discrete logarithm with respect to an element of order $q - 1 = \prod_{i=1}^{k} q_i$ can be reduced to computing k discrete logarithms, one with respect to an element of order q_i for $i = 1, 2, \ldots, k$. The main computations of this reduction are k exponentiations β^z in $\mathrm{GF}(q)$ and the reduction therefore requires about $k \cdot O(\log q)$ field operations. The Pohlig–Hellman reduction can be performed with an SLA over GF(q) to simplify in the same way the computation of ϕ_L. However, the remainder of the Pohlig–Hellman algorithm uses conditional steps that cannot be realized with an SLA. Instead, we solve the "Pohlig–Hellman subproblems" that arise in the computation of ϕ_L by "testing," via the use of field operations, each of the q_i possible values to see which is the correct value. All the subproblems can then be solved with $\sum_{i=1}^{k} q_i \cdot O(\log q)$ field operations. Because $k \leq \log_2 q$ it follows by using the Pohlig–Hellman reduction to the case where $q_1, q_2, \ldots, q_k$ are all primes that

$$C_q(\phi_L) = O(\log q)^2 \cdot q_m, \tag{10}$$

where $q_m = \max\{q_1, q_2, \ldots, q_k\}$ is the largest prime dividing $q - 1$.

It is well known that difficult functions can be used to design public-key cryptosystems (cf. [1]). Indeed, the discrete logarithm is used in public-key cryptosystems because it is widely believed to be "difficult" except for finite fields GF(q) with special structure (e.g., those where $q - 1$ has only small prime factors). Note that we did not define the complexity of a function like the discrete logarithm but mapping finite field elements to integers. The complexity of the closely related function ϕ_L is well defined but, in fact, an intruder in such a cryptosystem is faced neither with computing ϕ_L nor with computing the discrete logarithm itself. This follows by observing that, in order to transmit or store the ciphertext, the field elements composing the ciphertext have to be represented in binary form. The representations used in practice are closely related to our standard representation ϕ_S, because a representation is used where it is easy both to add and to multiply and because using a logarithmic representation would simply cancel the difficulty of the discrete logarithm. Therefore, the mapping from the standard to the logarithmic representation is important, i.e., the mapping $\phi_S to \phi_L(\cdot) = \phi_L(\tau_S(\cdot))$ from Im(ϕ_S) to Im(ϕ_L). Because this function maps a subset of $\{0,1\}^{l_S} \subset \mathrm{GF}(q)^{l_S}$ to $\{0,1\}^{l_L} \subset \mathrm{GF}(q)^{l_L}$, we are especially interested in its binary equivalent

$$\mathrm{Log} \triangleq \widetilde{\phi_S to \phi_L}.$$

Log, which is the binary equivalent of the mapping from the standard to the logarithmic representation, is then the relevant logarithm function for public-key cryptography. Sometimes the logarithm problem is considered more generally with respect to "some" representation ϕ. In our framework, this corresponds to the mapping $\phi to \phi_L(.) = \phi_L(\tau(.))$. We do not handle this generalization because we do not know any representation beside ϕ_S that yields more interesting results.

Any SLA computing the function ϕ_L can be easily adapted to compute the function Log as follows by first observing that $C_q(\phi_S to \phi_L) \leq C_q(\phi_L) + C_q(\tau_S)$. Then applying the upper bound (3) on τ_S and Theorem 2 to relate the algebraic complexity of $\phi_S to \phi_L$ to the circuit complexity of its binary equivalent Log, one gets

$$C_2(\text{Log}) = O(\log q)^2 \cdot (C_q(\phi_L) + O(\log q)) = O(\log q)^2 \cdot C_q(\phi_L).$$

For cryptographic purposes, however, one has to consider that Log could be much easier than ϕ_L, because having the standard representation of a field element as input could be advantageous compared to having only the field element as input. Note that, for the same reason, the cryptographically important function Log could also be much easier than the discrete logarithm itself, no matter how the complexity of the discrete logarithm mapping a field element to an integer is defined. The only cases where we are able to prove that the complexities of the logarithm functions ϕ_L and Log are equivalent follow from the fact that $\phi_L(.) = \phi_S to \phi_L(\phi_S(.))$, for, using Theorem 2 relating the complexities of $\phi_S to \phi_L$ and Log, we can deduce a second relation between the complexities of ϕ_L and Log and it follows that

$$\tfrac{1}{3} \cdot (C_q(\phi_L) - C_q(\phi_S)) \leq C_2(\text{Log}) = O(\log q)^2 \cdot C_q(\phi_L). \tag{11}$$

Therefore, both ϕ_L and Log have essentially the same complexity if the standard representation ϕ_S is easy. Equation (5) shows that this is the case for fields with small characteristics. Moreover, Corollary 5 will show that ϕ_S is easy for every $q = p^n$, where $p - 1$ has only small prime factors.

In the literature, one finds more related logarithm functions. One of these is a polynomial that maps $x = \beta^z$ in GF(q) into another element of GF(q) corresponding to the integer z. Such a polynomial has been explicitly determined in [13] and [14]. Analyzing this polynomial within our framework, we can show that the complexity of evaluating this polynomial lies in a sense between the complexities of Log and ϕ_L (for details see [3]).

Finally, we consider a logarithm function whose complexity does not seem to be related with the complexities of Log and ϕ_L, but Section 6 will show the contrary. This function is Zech's logarithm (see [2] or [11, p. 368], where it is called Jacobi's logarithm) that was introduced to simplify calculations in the logarithmic representation. With the help of the function $\widetilde{\text{Add}}_L$, Zech's logarithm can be defined as the function $\underline{\text{Zech}}$ from a subset of $\text{GF}(2)^{l_L}$ to $\text{GF}(2)^{l_L}$ such that

$$\underline{\text{Zech}}(\underline{x}) \triangleq \widetilde{\text{Add}}_L(\underline{x}, \underline{0}),$$

where $\underline{0} = \phi_L(1)$. This means that $\underline{\text{Zech}}$ maps the usual binary form $\underline{x}$ of

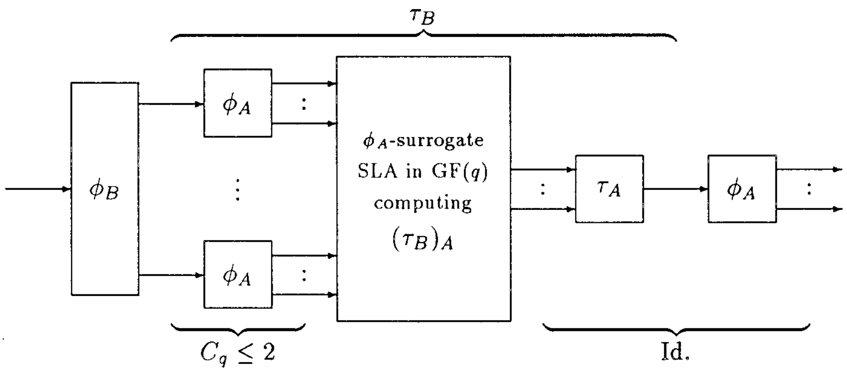

FIG. 2. Realization of ϕ_A using the imbedded-ϕ_A-surrogate SLA computing τ_B.

the integer x into the usual binary form $\underline{y}$ of y, where $\beta^y = \beta^x + 1$. Obviously, any upper bound on the complexity of $\widetilde{\mathrm{Add}}_L$ is also an upper bound on the complexity of $\underline{\mathrm{Zech}}$. But because $\beta^x + \beta^y = \beta^y \cdot (\beta^{x-y} + 1)$ for all x, y with $0 \leq y \leq x \leq q - 2$, it follows that the function $\widetilde{\mathrm{Add}}_L$ can be performed by subtracting an integer in usual binary form from another integer in this form, applying the function $\underline{\mathrm{Zech}}$ and then adding two integers in usual binary form. Together with the fact that addition and subtraction of integers in usual binary form can both be done with $O(\log q)$ binary-logic gates, one gets

$$C_2(\underline{\mathrm{Zech}}) \leq C_2(\widetilde{\mathrm{Add}}_L) = C_2(\underline{\mathrm{Zech}}) + O(\log q). \tag{12}$$

Therefore, the complexity of Zech's logarithm is essentially the complexity of adding two elements in their logarithmic representation.

5. STANDARD REPRESENTATION IS THE TWO-WAY EASIEST REPRESENTATION

Consider two different representations ϕ_A and ϕ_B of the finite field GF(q). Obviously, we have $\phi_A(x) = \phi_A(\tau_B(\phi_B(x)))$ for every $x \in \mathrm{GF}(q)$. The method of surrogate computation as introduced in Section 3 can be used to replace an SLA computing τ_B by its imbedded-ϕ_A-surrogate SLA. This yields an SLA computing ϕ_A as depicted in Fig. 2. (Note that the SLAs are in fact depicted as their corresponding GF(q)-gate circuits. Observe also that $(\tau_B)_A$ denotes the ϕ_A-associate of τ_B as defined in (6).) It is obvious that the cascade of τ_A and ϕ_A at the output can be omitted. Further, we know that each output component of ϕ_B can assume only the values 0 or

1 and, therefore, each following ϕ_A can be replaced by its restriction to the domain $\{0, 1\} \subset \mathrm{GF}(p)$. Every component function of this restriction of ϕ_A must either be constant, be x or be $1 + (-1) \cdot x$, and this shows that the algebraic complexity of the restriction of ϕ_A is at most 2. This yields the following bound:

$$C_q(\phi_A) \leq C_q(\phi_B) + M_A \cdot C_q(\tau_B) + 2 \cdot l_B. \tag{13}$$

Similar arguments can be applied for the realization of $\tau_A(.) = \tau_B(\phi_B(\tau_A(.)))$ using the imbedded-ϕ_A-surrogate SLA computing ϕ_B (for details see [3]) and one gets

$$C_q(\tau_A) \leq M_A \cdot C_q(\phi_B) + C_q(\tau_B) + 2 \cdot l_B. \tag{14}$$

Inequalities (13) and (14) together show that if the functions Add and Mult for a binary representation of GF(q) are both easy, then this representation is among the two-way easiest representations of GF(q) in the following precise sense:

THEOREM 3. *Given two binary representations ϕ_A and ϕ_B of* GF(q), *then*

$$C_q(\phi_A) + C_q(\tau_A) \leq (M_A + 1) \cdot (C_q(\phi_B) + C_q(\tau_B)) + 4 \cdot l_B.$$

A direct consequence of Theorem 3 is that the standard representation ϕ_S is one of the two-way easiest representations of GF(q).

COROLLARY 4. *Given the standard representation ϕ_S and an arbitrary binary representation ϕ of* GF(q) *with $l = O(\log q)$, then*

$$C_q(\phi_S) + C_q(\tau_S) = O(\log q)^2 \cdot (C_q(\phi) + C_q(\tau)).$$

Proof. Applying the bound of Theorem 3 with $\phi_A = \sigma_S$ and $\phi_B = \phi$ and using the upper bound $O(\log q)^2$ of (4) on M_S together with the hypothesis $l = O(\log q)$ gives the bound of the corollary. ■

To get an efficient realization of the function ϕ_S, it is often convenient to use the method of computation that led to inequality (5) showing how ϕ_S can be computed by computing the corresponding function in the ground field. We now apply Theorem 3 in the ground field to obtain the following corollary, which shows that the standard representation ϕ_S of GF(q) is two-way easy if there is a two-way easy binary representation of the ground field. In particular, by taking the arbitrary binary representation of the ground field to be the logarithmic representation ϕ_L, the following corollary shows that the standard representation ϕ_S is two-way easy for every $q = p^n$, where $p - 1$ has only small prime factors.

COROLLARY 5. *Given the standard representation ϕ_S of* GF(q) *and an arbitrary binary representation $\phi^{(p)}$ of* GF(p) *with $l = O(\log p)$ and $q = p^n$, then*

$$C_q(\phi_S) + C_q(\tau_S) \le n \cdot C_p(\phi^{(p)}) + n \cdot O(\log p)^2 \cdot C_p(\tau^{(p)}) + n^2 \cdot O(\log p).$$

In particular,

$$C_q(\phi_S) + C_q(\tau_S) = O(\log q)^2 \cdot p_m + O(\log q)^3,$$

where p_m is the largest prime dividing $p - 1$.

Proof. Using (3) and (5), we see that $C_q(\phi_S) + C_q(\tau_S)$ can be bounded from above by $n \cdot C_p(\phi_S^{(p)}) + n^2 \cdot O(\log p) + O(\log q)$. Applying inequality (13) with $\phi_A = \phi_S$ and $\phi_B = \phi$ to bound $C_p(\phi_S^{(p)})$ and using the upper bound (4) on $\underline{M}_S$ together with the hypothesis $l = O(\log p)$ gives the first inequality of the corollary.

To bound $C_q(\phi_S) + C_q(\tau_S)$ with $\phi^{(p)} = \phi_L$ for $p - 1$ having only small prime factors, we apply bounds (8) and (10) for the logarithmic representation ϕ_L of GF(p). It follows directly that $C_q(\phi_S) + C_q(\tau_S) = n \cdot O(\log p)^2 \cdot p_m + n \cdot O(\log p)^3 + n^2 \cdot O(\log p)$. ■

Suppose that there is a two-way easy binary representation of GF(q). For this case, the following corollary shows that the two-way complexity of any binary representation ϕ of GF(q) is closely related to the maximum complexity M of the functions Add and Mult for this representation ϕ. Note that this maximum complexity M of Add and Mult is defined as an algebraic complexity but, because of the close relation (7) between M and the circuit complexity $\tilde{M}$, the following corollary could also be formulated using $\tilde{M}$ in place of M.

COROLLARY 6. *If the standard representation ϕ_S of* GF(q) *fulfills $C_q(\phi_S) = O(\log q)^i$, then it follows for any binary representation ϕ of* GF(q) *with $l = O(\log q)$ that*

$$\tfrac{1}{2} \cdot M \le C_q(\phi) + C_q(\tau) = O(\log q)^i \cdot M.$$

Proof. First observe that Add($\underline{x}$, $\underline{y}$) $= \phi(\tau(\underline{x} + \tau(\underline{y}))$ and $C_q(\text{Add}) \le C_q(\phi) + 2 \cdot C_q(\tau) + 1$. Because similar relations hold for Mult, the left inequality follows. The right inequality is a direct consequence of Theorem 3, the upper bound of (3) on $C_q(\tau_S)$, the fact that $l_S = O(\log q)$, and the hypothesis $C_q(\phi_S) = O(\log q)^i$. ■

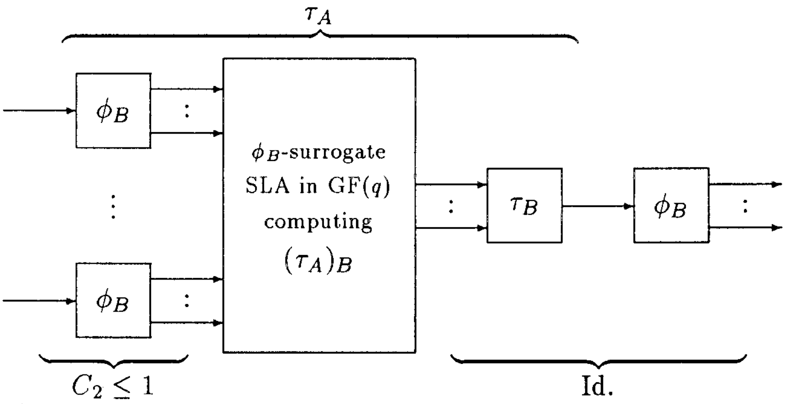

FIG. 3. Realization of $\phi_A to \phi_B$ using the imbedded-ϕ_B-surrogate SLA computing τ_A.

6. CLOSE RELATION BETWEEN ZECH'S AND DISCRETE LOGARITHM

Consider the function $\phi_A to \phi_B$: $\mathrm{Im}(\phi_A) \rightarrow \mathrm{Im}(\phi_B)$ defined by $\phi_A to \phi_B(.) = \phi_B(\tau_A(.))$ that maps $\phi_A(x)$ into $\phi_B(x)$ for every x in GF(q). In Fig. 3, $\phi_A to \phi_B$ is realized by using the imbedded-ϕ_B-surrogate SLA computing τ_A in GF(q). Obviously, the cascade of τ_B and ϕ_B at the output can be eliminated. The ϕ_B's at the input can be replaced by their restrictions to the inputs 0 and 1 in GF(q) as in Section 5. Now, all remaining functions needed in the computations of Fig. 3 can be replaced by their binary equivalents. For the restrictions of ϕ_B, their binary equivalents can be computed with at most one operation in GF(2). Further, using the surrogate SLA in GF(2) yields the bound

$$C_2(\widetilde{\phi_A to \phi_B}) \leq \tilde{M}_B \cdot C_q(\tau_A) + l_A. \tag{15}$$

It follows from Theorem 2 that this upper bound on $C_2(\widetilde{\phi_A to \phi_B})$ also gives an upper bound on $C_q(\phi_A to \phi_B)$, namely $C_q(\phi_A to \phi_B) \leq 3 \cdot C_2(\widetilde{\phi_A to \phi_B})$. Note that the close relation (7) between $\tilde{M}$ and M allows us to state the results in terms of M instead of $\tilde{M}$.

The mappings between representations are of special interest when one of them is the standard representation ϕ_S (cf. Section 3). If the inverse mapping τ of the second representation ϕ is easy, then inequality (15) can be used to show that $\phi to \phi_S$ is also easy and that $\phi_S to \phi$ is about as difficult as the most difficult function Add or Mult for the representation ϕ. The following theorem states this more precisely.

THEOREM 7. *Given the standard representation ϕ_S and an arbitrary bi-*

nary representation ϕ of GF(q) *with* $l = O(\log q)$, *then the following inequalities hold:*

$$C_2(\widetilde{\phi to \phi_S}) = O(\log q)^2 \cdot C_q(\tau)$$
$$\tilde{M} - O(\log q)^2 \cdot C_q(\tau) \leq C_2(\widetilde{\phi_S to \phi}) = O(\log q) \cdot \tilde{M}.$$

Proof. The first inequality is a direct consequence of (15), the upper bound (4) on $\tilde{M}_S$, and the hypothesis $l = O(\log q)$. From the obvious relations

$$C_2(\widetilde{\mathrm{Add}}) \leq 2 \cdot C_2(\widetilde{\phi to \phi_S}) + C_2(\widetilde{\mathrm{Add}}_S) + C_2(\widetilde{\phi_S to \phi})$$
$$C_2(\widetilde{\mathrm{Mult}}) \leq 2 \cdot C_2(\widetilde{\phi to \phi_S}) + C_2(\widetilde{\mathrm{Mult}}_S) + C_2(\widetilde{\phi_S to \phi}),$$

it follows that

$$\tilde{M} \leq 2 \cdot C_2(\widetilde{\phi to \phi_S}) + \tilde{M}_S + C_2(\widetilde{\phi_S to \phi}). \tag{16}$$

Using the first inequality of the theorem and the upper bound (4) on $\tilde{M}_S$, we obtain

$$\tilde{M} - O(\log q)^2 \cdot C_q(\tau) \leq C_2(\widetilde{\phi_S to \phi}).$$

Using inequality (15) together with the upper bound (3) on $C_q(\tau_S)$ and the fact that $l_S = O(\log q)$, we obtain the final inequality of the theorem. ■

This theorem can be applied in various ways (see [3]). In what follows, we restrict ourselves to showing interesting relations between the functions introduced in Section 4. The following corollary asserts that, for any finite field, the logarithm function used in public-key cryptography has about the same complexity as has adding of two field elements given in the logarithmic representation. The complexity of addition here is measured as circuit complexity, but Theorem 2 also allows one to obtain similar results for the algebraic complexity of addition.

COROLLARY 8. *For the mappings between the standard and the logarithmic representations of GF(q), i.e., between ϕ_S and ϕ_L, the following bounds hold:*

$$C_2(\widetilde{\phi_L to \phi_S}) = O(\log q)^3$$
$$C_2(\widetilde{\mathrm{Add}}_L) - O(\log q)^3 \leq C_2(\mathrm{Log}) = O(\log q) \cdot C_2(\widetilde{\mathrm{Add}}_L).$$

If $C_q(\phi_S) = O(\log q)^i$, then

$$C_2(\widetilde{\mathrm{Add}}_L)/O(\log q)^2 = C_q(\phi_L) = O(\log q) \cdot C_2(\widetilde{\mathrm{Add}}_L) + O(\log q)^i.$$

Proof. The first two inequalities follow from Theorem 7 and the upper bound (8). One notes that, because $\widetilde{\mathrm{Add}}_L$ depends on $2 \cdot l_L = 2 \cdot \lceil \log_2 q \rceil$ inputs, the complexity of $\widetilde{\mathrm{Add}}_L$ must be at least $2 \cdot \lceil \log_2 q \rceil - 1$ and therefore (9) implies that $\widetilde{\mathrm{Mult}}_L$ is never substantially more complex. Therefore $\tilde{M}_L$ is essentially equal to the complexity of $\widetilde{\mathrm{Add}}_L$. To obtain the last inequality, one uses inequality (11) together with the fact that $C_q(\phi_L) \geq \lceil \log_2 q \rceil$. ■

Because of the close relation (12) between the complexity of Zech's logarithm and that of the function $\widetilde{\mathrm{Add}}_L$, it follows from (17) that computing Zech's logarithm $\underline{\mathrm{Zech}}$ is about as difficult as computing the logarithm function Log used in public-key cryptography. Because this holds for every finite field, we know either that Zech's logarithm cannot be used to simplify calculations made in the logarithmic representation of GF(q) or else that the logarithm function used in public-key cryptography is much easier than the discrete logarithm itself.

The second part of Corollary 8 shows that the functions Log, ϕ_L, and $\underline{\mathrm{Zech}}$ all have similar complexities when the standard representation ϕ_S is two-way easy. It follows in particular that using Zech's logarithm does not simplify calculations in the logarithmic representation of a finite field GF(p^n) where $p - 1$ has only small prime factors (e.g., GF(2^n)).

7. Relations between Algebraic and Circuit Complexity

This section extends Theorem 2, which shows a close relation between algebraic and circuit complexity for certain functions. We restrict ourselves to functions f: GF(q) $\rightarrow$ GF(q), but all the results of this section can be extended to functions mapping GF$(q)^a$ to GF$(q)^b$.

It is shown in [3] that, given a lower bound on the algebraic complexity of f, there is a specific Boolean function having almost the same lower bound on its circuit complexity. The result is not entirely satisfying because the choice of the associated Boolean function is quite arbitrary and its complexity could be much larger than that of f. It would be much more natural to have a close complexity relation when the Boolean function is defined with the help of the standard representation ϕ_S. The following theorem relates the complexities of an arbitrary function f and its ϕ_S-associate $f_S(.) = \phi_S(f(\tau_S(.)))$ as defined in Section 3. The result is not stated in terms of the algebraic complexity of f_S, but rather in terms of the closely related and practically relevant circuit complexity of its binary equivalent $\tilde{f}_S$.

THEOREM 9. *Given the standard representation ϕ_S of* GF(q), *for any function f*: GF$(q) \to$ GF(q), *the binary equivalent $\tilde{f}_S$ of $f_S(.) = \phi_S(f(\tau_S(.)))$*: Im$(\phi_S) \to$ Im(ϕ_S) *satisfies*

$$\tfrac{1}{3} \cdot (C_q(f) - C_q(\phi_S) - O(\log q)) \leq C_2(\tilde{f}_S) = O(\log q)^2 \cdot C_q(f). \quad (18)$$

Any binary representation ϕ of GF(q) *with inverse mapping τ and $l = O(\log q)$ satisfies*

$$C_q(\phi_S) \leq C_q(\phi) + O(\log q)^2 \cdot C_q(\tau). \quad (19)$$

Proof. The definition of f_S implies that $f(.) = \tau_S(f_S(\phi_S(.)))$ and, therefore, that

$$C_q(f) \leq C_q(f_S) + C_q(\phi_S) + C_q(\tau_S).$$

Equation (3) shows that τ_S is easy and Theorem 2 gives a lower bound on $C_2(\tilde{f}_S)$ in terms of $C_q(f_S)$. This yields the first inequality in (18). The second inequality in (18) follows from using the ϕ_S-surrogate SLA to compute $\tilde{f}_S$ in GF(2), which implies

$$C_2(\tilde{f}_S) \leq \tilde{M}_S \cdot C_q(f),$$

and from the upper bound (4) on $\tilde{M}_S$. Inequality (19) is a consequence of (13) applied to ϕ_S, together with the bound (4) on $\tilde{M}_S$ and the hypothesis that $l = O(\log q)$. ■

Lempel *et al.* [5] showed for fields GF(q) with small characteristics p that algebraic complexity C_q is essentially circuit complexity. Indeed, the condition that GF(q) has small characteristic p has been considered to be necessary for this equivalence (e.g., see [6]). But the condition that $p - 1$ has only small prime factors is sufficient.

COROLLARY 10. *Let ϕ_S be the standard representation ϕ_S of GF(q) with $q = p^n$ and let p_m be the largest prime dividing $p - 1$. If $p_m = O(\log q)$, then it holds for any function f: GF$(q) \to$* GF(q) *and the binary equivalent $\tilde{f}_S$ of its ϕ_S-associate $f_S(.) = \phi_S(f(\tau_S(.)))$ that*

$$\tfrac{1}{3} \cdot C_q(f) - O(\log q)^3 \leq C_2(\tilde{f}_S) = O(\log q)^2 \cdot C_q(f).$$

Proof. For $p - 1$ having only small prime factors, Corollary 5 gives an upper bound on $C_q(\phi_S)$ that, together with the bound of Theorem 9, yields the bound of this corollary. ■

8. Concluding Remarks

A specific notion of binary representations of finite fields was introduced. This permits one to study binary representations in terms of algebraic complexity. Several new insights and relations concerning the standard and the logarithmic representation were deduced. In particular, it was shown that the circuit complexity of Zech's logarithm is closely related to the circuit complexity of the discrete logarithm used in public-key cryptography. The following interesting question could be solved only partially in this paper.

Open Problem. *Is there a two-way easy binary representation for every finite field?*

Actually, it remains only to solve the problem for prime fields where the field size minus 1 has a large prime factor. If the problem is solved, then the standard representation is also two-way easy for every finite field and this has several further consequences. In particular, it would follow that the algebraic complexity and the circuit complexity are equivalent for every finite field.

We have seen that the notion of binary representations of finite fields provides new insight into the algebraic complexity of functions. Recently, we exploited binary representations to device an efficient algorithm for factoring polynomials in an arbitrary finite field (see [4]). We expect that the notion of binary representations will prove useful in other applications as well.

Acknowledgment

The author is grateful to J. L. Massey, who significantly improved the presentation of this paper.

References

1. W. Diffie and M. E. Hellman, New directions in cryptography, *IEEE Trans. Inform. Theory* **IT-22,** no. 6 (1976), 644–654.

2. K. Huber, Some comments on Zech's logarithms, *IEEE Trans. Inform. Theory* **IT-36,** no. 4 (1990), 946–950.
3. J. Ganz, "Algebraic Complexity in Finite Fields," Ph.D. thesis no. 10867, Swiss Federal Institute of Technology, 1994.
4. J. Ganz, Factoring polynomials using binary representations of finite fields, *IEEE Trans. Inform. Theory,* to appear.
5. A. Lempel, G. Seroussi, and J. Ziv, On the power of straight-line computations in finite fields, *IEEE Trans. Inform. Theory* **IT-28,** no. 6 (1982), 875–880.
6. J. von zur Gathen and G. Seroussi, Boolean circuits versus arithmetic circuits, *Inform. Comput.* **91** (1991), 142–154.
7. L. G. Valiant, Why is boolean complexity theory difficult? *in* "Boolean Function Complexity," London Math. Soc. Lecture Note Series, Vol. 169, pp. 84–94, Cambridge Univ. Press, Cambridge, New York, 1992.
8. V. Strassen, Berechnung und Programm, I, *Acta Informatica* **1** (1972), 320–335.
9. I. Wegener, "The Complexity of Boolean Functions," Wiley, New York, 1987.
10. J. E. Savage, "The Complexity of Computing," Wiley, New York, 1976. [Reprint with corrections by Krieger, Malabar, FL, 1987]
11. R. Lidl and H. Niederreiter, "Finite Fields," Addison–Wesley, Reading, MA, 1983.
12. S. C. Pohlig and M. E. Hellman, An improved algorithm for computing logarithms over GF(p) and its cryptographic significance, *IEEE Trans. Inform. Theory* **IT-24,** no. 1 (1978), 106–110.
13. G. L. Mullen and D. White, A polynomial representation for logarithms in GF(q), *Acta Arithmetica* **47** (1986), 255–261.
14. H. Niederreiter, A short proof for explicit formulas for discrete logarithms in finite fields, *Appl. Algebra Engrg. Comm. Comput.* **1** (1990), 55–57.