COMPUTING SPECIAL POWERS
INFINITE FIELDS

JOACHIM VON ZUR GATHEN AND MICHAEL NÖCKER

Abstract. We study exponentiation in nonprime finite fields with very special exponents such as they occur, for example, in inversion, primitivity tests, and polynomial factorization. Our algorithmic approach improves the corresponding exponentiation problem from about quadratic to about linear time.

1. Introduction

Exponentiation in finite fields \( \mathbb{F}_{q^n} \) has many applications, several cryptosystems among them, e.g., [12] and [14]. In those situations, one has arbitrary (or random) exponents. There is a substantial body of literature on this topic; see the references given in [23]. The fastest algorithms in \( \mathbb{F}_{q^n} \)—for different basis representations of \( \mathbb{F}_{q^n} \)—use \( O(n^2 \log \log n \log q) \) operations in \( \mathbb{F}_q \); see [17]. In this paper we deal with a different problem: very special exponents, e.g., repunits \( (q^n - 1)/(q - 1) \) with all 1’s in their \( q \)-ary representation. Such exponents occur in inversion and in primitivity tests, and we can employ our methods in polynomial factorization.

We start in Section 2 with a recapitulation of what we need about addition chains and a variant which is important for our problem: \( q \)-addition chains, where multiplication by some fixed integer \( q \) is free. We use this for exponentiation in extension fields of \( \mathbb{F}_q \). Section 3 summarizes the basic algorithmic tool which is an adaption of Brauer’s [6] method, namely a \( q \)-addition chain for the repunit \( e = (q^n - 1)/(q - 1) \) with about \( \log n \) non-\( q \)-steps, which is only logarithmic in the length \( n \log q \) of generic numbers of the same magnitude. The known efficient algorithms for general exponentiation are reviewed in Section 4.

This approach improves the corresponding exponentiation problem from quadratic to about linear time. We discuss five applications: inversion in Section 5, primitivity testing in Section 7, and three tasks connected to polynomial factorization in Section 8; these last two sections use an exponentiation algorithm developed in Section 6. Experiments show that our method often yields better results than other well-known algorithms. For example, the number of multiplications to test an element \( \mathbb{F}_{q^n}^* \) for primitivity can be reduced to less than 50% on average (see Table 2) with addition chains for special exponents.

From a high-level point of view, we have the following picture for exponentiation in \( \mathbb{F}_{q^n} \). The number of operations are in the “\( O \)”-sense. Some of the algorithms assume an optimal normal basis as data structure, where a \( q \)-th power in \( \mathbb{F}_{q^n} \) is free, or a sparse irreducible polynomial with a constant number of terms.

An extended abstract of this paper has been published; see [24].

Received by the editor July 28, 2002 and, in revised form, December 9, 2002.

2000 Mathematics Subject Classification. Primary 68Q40; Secondary 11Y16.
2. ADDITION CHAINS

The standard reference on this topic is [39], Section 4.6.3. An addition chain is a sequence \( \gamma \) of pairs \( ((j(1), k(1)), \ldots, (j(l), k(l))) \) of nonnegative integers with \( 0 \leq k(i) \leq j(i) < i \) for all \( 1 \leq i \leq l \). The number \( l \) of pairs is the length \( L(\gamma) \) of \( \gamma \). The semantics of \( \gamma \) is the set \( S(\gamma) = \{ a_0, \ldots, a_l \} \) of integers such that \( a_0 = 1 \) and \( a_i = a_{j(i)} + a_{k(i)}, \) for \( 1 \leq i \leq l \). For our purpose we may assume \( 1 = a_0 < a_1 < \cdots < a_l \), and we use this assumption tacitly throughout the paper. We say that \( \gamma \) computes \( e \) if \( e \in S(\gamma) \).

The main purpose in life of an addition chain is to generate an exponentiation algorithm: if \( \gamma \) is an addition chain computing \( e \) as above, then for \( \beta \in \mathbb{F}_q^* \) we can compute \( \beta^e \) by computing \( \beta^{a_i} = \beta^{a_{j(i)}} \cdot \beta^{a_{k(i)}} \) for all \( 1 \leq i \leq l \).

In the literature it is common to identify the semantics with the addition chain itself. But different addition chains may have the same semantics. As remarked by Knuth [39] an addition chain \( \gamma \) corresponds in a natural way to a directed graph \( \Gamma \). The set of nodes of \( \Gamma \) is just \( S(\gamma) \), and edges point from \( a_{j(i)} \) and \( a_{k(i)} \) to \( a_i \), for all \( 1 \leq i \leq l \). If \( j(i) = k(i) \), then we call step \( i \) a doubling. If \( i - 1 = j(i) > k(i) \), then step \( i \) is a star step. A star chain consists only of doublings and star steps.

Example 1. The graphs of two addition chains computing \( e = 22 \) are given below. Both have the same semantics \( S = \{ 1, 2, 3, 5, 6, 11, 22 \} \). The first one is \( ((0, 0), (1, 0), (2, 1), (3, 0), (4, 3), (5, 5)) \). Both addition chains have \( \ell = 6 \) steps. The

First addition chain:

```
1 --> 2 --> 3 --> 5 --> 6 --> 11 --> 22
```

Second addition chain:

```
1 --> 2 --> 3 --> 5 --> 6 --> 11 --> 22
```

first one is a star chain with 2 doublings and 4 additions, and the second one is not a star chain and has 3 doublings and 3 additions. The edges from \( a_{j(i)} \) to \( a_i \) are drawn in bold.
For our algorithmic purposes it is useful to generalize the notion of addition chains in the following way; see [13]. Besides adding two previous values, we also allow multiplying a previous value by a fixed number $q$.

**Definition 2.** Let $q \in \mathbb{N}_{\geq 2}$. A multiple addition chain with multiplication by $q$, or $q$-addition chain for short, is a sequence $\gamma = ((j(1), k(1)), \ldots, (j(l), k(l)))$ of pairs of integers with $0 \leq j(i) < i$ and $k(i) = -q$ or $0 \leq k(i) \leq j(i)$ for all $1 \leq i \leq l$. We let $A(\gamma) = \{i \leq l : k(i) \geq 0\}$ be the number of additions and $Q(\gamma) = \{i \leq l : k(i) = -q\}$ be the number of $q$-steps, and the number $l$ of pairs is the length $L(\gamma) = A(\gamma) + Q(\gamma)$ of $\gamma$. We define the semantics $S(\gamma) = \{a_0, \ldots , a_l\}$ by $a_0 = 1$ and

$$a_i = \begin{cases} a_{j(i)} + a_{k(i)} & \text{if } k(i) \neq -q, \\ q \cdot a_{j(i)} & \text{if } k(i) = -q, \end{cases}$$

for all $1 \leq i \leq l$. Then $\gamma$ computes any element of $S(\gamma)$.

Again, we may assume that $1 = a_0 < a_1 < \cdots < a_l$. These $q$-addition chains are useful for exponentiation in finite fields when a $q$th power is essentially free (see Section 4). Every $q$-addition chain can be rewritten as an addition chain by expanding the $q$-step $a_i = q \cdot a_{j(i)}$ using an addition chain computing $q$. A $2$-addition chain is just an addition chain, $2$-steps are doublings, and an addition chain is a $q$-addition chain for any $q \geq 2$.

**Example 3.** The 5-addition chain $\gamma = ((0,0), (1,1), (2, -5), (3,1))$ computes 22.

$$1 \rightarrow 2 \rightarrow 4 \xrightarrow{5} 20 \rightarrow 22$$

We can expand the 5-step $(2, -5)$ using an addition chain $\delta$ with semantics $S(\delta) = \{1, 2, 4, 5\}$; this includes two doublings and one star step. We connect $\delta$ to the node $a_2 = 4$ and insert the steps $8 = 4 + 4, 16 = 8 + 8, 20 = 16 + 4$ in $\gamma$.

We define

$$\ell_q(e) = \min \{L(\gamma) : \gamma \text{ is a } q\text{-addition chain computing } e\}$$

as the length of a shortest $q$-addition chain for $e$, and we set $\ell_q(1) = 0$. Then $\ell_2(e)$ corresponds to the usual addition chains and is sometimes called the additive complexity $\ell(e)$ of $e$.

**Operations on addition chains.** In order to state our constructions succinctly, the following terminology is useful. Let $q \in \mathbb{N}_{\geq 2}$ and $\gamma$ be as in Section 2 and let $0 \leq t \leq l$. We define the truncation of $\gamma$ at $a_t$ as the $q$-addition chain $\gamma|_{a_t} = ((j(1), k(1)), \ldots, (j(t), k(t)))$ with $S(\gamma|_{a_t}) = \{a_0, \ldots , a_t\}$. This is well defined since $1 = a_0 < a_1 < \cdots < a_l$. Thus $\gamma = \gamma|_{a_t}$ and $\gamma|_{a_t}$ is the empty chain with $S(\gamma|_{a_t}) = \{1\}$. Furthermore,

$$\gamma + a_t = ((j(1), k(1)), \ldots, (j(l), k(l)), (l, t))$$

is a $q$-addition chain computing $a_t + a_t$. Obviously $A(\gamma + a_t) = A(\gamma) + 1, Q(\gamma + a_t) = Q(\gamma)$, and $L(\gamma + a_t) = L(\gamma) + 1$. 
Let \( \delta = ((j'(1), k'(1)), \ldots, (j'(t), k'(t))) \) be another \( q \)-addition chain with semantics \( S(\delta) = \{b_0, \ldots, b_t\} \). The product chain
\[ \gamma \circ \delta = ((j(1), k(1)), \ldots, (j(l), k(l)), (l + j'(1), k'(1)), \ldots, (l + j'(t), k'(t))) \]
is a \( q \)-addition chain for \( a_t \cdot b_t \), where
\[ k''(i) = \begin{cases} l + k'(i) & \text{if } 0 \leq k'(i), \\ -q & \text{if } k'(i) = -q. \end{cases} \]
We have \( A(\gamma \circ \delta) = A(\gamma) + A(\delta) \), \( Q(\gamma \circ \delta) = Q(\gamma) + Q(\delta) \) and the semantics are \( S(\gamma \circ \delta) = S(\gamma) \cup \{a_1, b_1, \ldots, a_t, b_t\} \). Thus \( \ell_q(e \cdot f) \leq \ell_q(e) + \ell_q(f) \) for all \( e, f \in \mathbb{N} \), as already remarked by A. Brauer [6]. Bergeron et al. ([4], [3]) use continued fraction approximations and product chains to describe efficient addition chains.

**Example 4.** The following addition chain \( \gamma \) for \( e = 7 \) has shortest length \( \ell_2(7) = 4 \):

\[
\begin{array}{c}
1 \\
\longrightarrow
\end{array}
\begin{array}{c}
2 \\
\longrightarrow
\end{array}
\begin{array}{c}
3 \\
\longrightarrow
\end{array}
\begin{array}{c}
5 \\
\longrightarrow
\end{array}
\begin{array}{c}
7
\end{array}
\]

We obtain an addition chain for \( e^2 = 49 \) of length \( L(\gamma \circ \gamma) = 2\ell_2(7) = 8 \):

\[
\begin{array}{c}
1 \\
\longrightarrow
\begin{array}{c}
2 \\
\longrightarrow
\end{array}
\begin{array}{c}
3 \\
\longrightarrow
\end{array}
\begin{array}{c}
5 \\
\longrightarrow
\end{array}
\begin{array}{c}
7 \\
\longrightarrow
\end{array}
\begin{array}{c}
14 \\
\longrightarrow
\end{array}
\begin{array}{c}
21 \\
\longrightarrow
\end{array}
\begin{array}{c}
35 \\
\longrightarrow
\end{array}
\begin{array}{c}
49
\end{array}
\]

This method does not necessarily compute a \( q \)-addition chain for \( e \cdot f \) of shortest length even if \( \gamma \) and \( \delta \) are minimal. For example \( \ell_2(49) = 7 < 8 = 2 \cdot \ell_2(7) \) (see [39], Section 4.6.3, Figure 14: \( \{1, 2, 4, 8, 16, 32, 33, 49\} \)).

We let \( \gamma \sqcup \delta \) be the concatenation of \( \gamma \) and \( \delta \) with values occurring twice being removed once and the result sequence being sorted. By \( \gamma \circ q^r \) with \( r \in \mathbb{N}_{\geq 1} \) we denote the \( q \)-addition chain \( ((j(1), k(1)), \ldots, (j(l), k(l)), (l, -q), \ldots, (l + r - 1, -q)) \) computing \( q^r \cdot a_t \).

**Upper bounds.** Let \( e, q \in \mathbb{N}_{\geq 1} \) with \( q \geq 2 \) in what follows. The \( q \)-ary representation of \( e \) is \( (e)_q = (e_{\lambda - 1}, \ldots, e_0) \) with \( e_0, \ldots, e_{\lambda - 1} \in \{0, \ldots, q - 1\} \) uniquely determined such that \( \sum_{0 \leq i < \lambda} e_i q^i = e \), and length \( \lambda = \lambda_q(e) = |\log_q e| + 1 \). The \( q \)-ary Hamming weight of \( e \) is \( \nu_q(e) = \# \{i: 0 \leq i < \lambda, e_i \neq 0\} \leq \lambda_q(e) \).

A \( q \)-addition chain \( \gamma \) is called a star \( q \)-addition chain if there are only \( q \)-steps and star steps, so that \( j(i) = i - 1 \) for all \( i \leq l \). We write \( \ell^*_q(e) \) for the length of a shortest star \( q \)-addition chain for \( e \) and define \( \ell^*_q(1) = 0 \). Of course \( \ell_q(e) \leq \ell^*_q(e) \) for all \( e \in \mathbb{N}_{\geq 1} \). Knuth in [39], Section 4.6.3, page 477, reports that sometimes inequality holds: \( \ell_2(12509) < \ell^*_2(12509) \).

**Lemma 5.** Let \( \gamma \) be an addition chain with \( S(\gamma) = \{a_0, \ldots, a_l\} \). Then
\begin{enumerate}
\item \( \sum_{1 \leq i \leq l} a_k(i) \geq a_l - 1 \).
\item \( \sum_{1 \leq i \leq l} a_k(i) = a_l - 1 \) if and only if \( \gamma \) is a star addition chain.
\end{enumerate}
Proof. For (i), we prove by induction that \( \sum_{1 \leq i \leq h} a_{k(i)} \geq a_h - 1 \) for all \( 1 \leq h \leq l \).
The case \( h = 1 \) is trivial, and for the induction step we have
\[
\sum_{1 \leq i \leq h} a_{k(i)} = \sum_{1 \leq i < h} a_{k(i)} + a_{k(h)} \\
\geq a_{h-1} - 1 + a_{k(h)} \geq a_j(h) - 1 + a_{k(h)} \\
= a_h - 1,
\]
since \( j(h) \leq h - 1 \) and \( a_0 < a_1 < \cdots < a_l \) by assumption. For a star addition chain, the same induction works with \("=\) instead of \("\leq\) since \( h - 1 = j(h) \) for \( 0 \leq h \leq l \). Now assume that \( \gamma \) is not a star addition chain. Let \( 1 < h \leq l \) be the smallest index of a nonstar step, so that \( a_j(h) + a_{k(h)} = a_h, k(h) \leq j(h) < h - 1 \), and \( \gamma|_{a_{h-1}} \) is a star addition chain. Then
\[
\sum_{0 \leq i \leq h} a_{k(i)} = \sum_{0 \leq i < h} a_{k(i)} + a_{k(h)} \\
= a_{h-1} - 1 + a_{k(h)} > a_j(h) - 1 + a_{k(h)} = a_h - 1,
\]
since \( a_0 < a_1 < \cdots < a_l \) by assumption. Proceeding as above, we also find strict inequality in (i) for \( \gamma \).

In this paper, we will present various addition chains. Besides the notion of "computing" given above, we also say that we "compute" these chains, which are really algorithms to compute numbers. Thus we present algorithms that compute algorithms that compute numbers; maybe "compile" would be a better word for the former.

We consider the \( q^r \)-ary representation of \( e \) with a parameter \( r \in \mathbb{N}_{\geq 1} \). Brauer [6] gives the result below for \( q = 2 \). A more detailed result is proven in [18]; see also [25]. We refer to the corresponding addition chain as Brauer's addition chain.

**Theorem 6** (Brauer [6]). Let \( q, e \in \mathbb{N}_{\geq 2} \) and \( s = \lambda_q(e) \geq 0 \). Then there exists a \( q \)-addition chain \( \gamma \) for \( e \) with
\[
A(\gamma) \leq \frac{s}{\log q s} \cdot \left( 1 + \frac{2 \log \log_q s}{\log_s - 2 \log \log_q s} + \frac{q}{\log_q s} \right) - 2 \\
= \frac{s}{\log q s} \cdot (1 + o(1)) \text{ additions,}
\]
\[
Q(\gamma) \leq s \text{ many } q\text{-steps.}
\]
This yields
\[
\ell_q(e) \leq L(\gamma) \leq s + \frac{s}{\log_q s} (1 + o(1)).
\]

This result is obtained by choosing \( r \) near \( \log_q \lambda_q(e) - 2 \log_q \log_q \lambda_q(e) \) in Brauer's method. In practice, it is probably best to take \( r \) as the closest integer to this value and then to modify the adjacent integers until one has a value \( r \) whose Brauer chain is shorter than those for \( r \pm 1 \). In each case, the precomputed elements less than \( q^r \) that are not needed should be discarded. Brauer's approach can be seen as a generalization of the well-known repeated squaring algorithm. Here only non-\( q \)-steps are used. We refer to this as the binary addition chain; it is a star addition.
Theorem 11. Let $e$ be a positive integer. The length of a binary addition chain yields a well-known upper bound on the additive complexity of $e$:

$$\ell(e) \leq \ell^*(e) \leq \lambda_2(e) + \nu_2(e) - 2 \leq 2 \lfloor \log_2 e \rfloor.$$  

A trivial lower bound is $\ell_0(e) \geq \log_2 e$. A. Schönhage [47] proved

$$\ell(e) \geq \log_2 e + \log_2 \nu_2(e) - 2.13$$

as a lower bound on the additive complexity for any $e \in \mathbb{N}_{\geq 2}$. Downey et al. [13] proved that the problem of deciding for a set of positive integers $E = \{e_1, \ldots, e_m\}$ and an integer $L$ whether there exists an addition chain for $E$ of length at most $L$ is $\mathcal{NP}$-complete. Knuth [39], page 698, remarks: “It is unknown whether or not the problem of computing $\ell_2(n)$ is $\mathcal{NP}$-complete.” In view of this, it does not seem to be a promising approach to try to calculate an addition chain of shortest length for $E = \{e\}$; rather we look for one with reasonably short length.

3. $q$-addition chains for repunits

Let $q,n \in \mathbb{N}_{\geq 2}$, and let $e = (q^n - 1)/(q - 1)$. The $q$-ary representation of $e$ consists only of ones, and $e$ is called a repunit; see [2]. We can improve the result of Theorem 6 for repunits because of their special form. For an integer $a$, we let $w_a = (q^a - 1)/(q - 1) = \sum_{0 \leq i < a} q^i$. The simple equation valid for all $a,b \in \mathbb{N}_{\geq 1}$

$$w_{a+b} = \sum_{0 \leq i < (a+b)} q^i = \left( \sum_{0 \leq i < a} q^i \right) \cdot q^b + \sum_{0 \leq i < b} q^i = w_a \cdot q^b + w_b$$

indicates how to compose two $q$-addition chains for the right-hand sums with $b$ many $q$-steps and one addition to get a chain for the left-hand sum. This reduces the problem of finding a $q$-addition chain for $e$ to that of obtaining an (ordinary) addition chain for $n$. We get the following method for a repunit, which is in [10] for $q = 2$.

Algorithm 10 ($q$-addition chain for repunits).
Input: Integers $n,q \in \mathbb{N}_{\geq 2}$ and an addition chain $\varepsilon = ((j(1), k(1)), \ldots, (j(l), k(l)))$ for $n$ with $S(\varepsilon) = \{a_0, \ldots, a_l\}$.
Output: A $q$-addition chain $\gamma$ for $e = (q^n - 1)/(q - 1)$.
1. Set $\gamma$ equal to the empty addition chain with $S(\gamma) = \{1\}$.
2. For $1 \leq i \leq l$ do compute $\gamma \leftarrow \gamma \sqcup (\gamma|_{w_{j(i)}(x)} \odot q^{a_k(i)}) \oplus w_{a_k(i)}$. [We will show that the quantities used have been computed.]
3. Return $\gamma$.

Theorem 11. Let $n, q, \varepsilon, \gamma,$ and $e$ be as in Algorithm 17. Then $\gamma$ computes $e$, and $A(\gamma) = L(\varepsilon)$ and $n - 1 \leq Q(\gamma) \leq \sum_{1 \leq i \leq L(\varepsilon)} a_k(i)$.

Proof. Using induction along the algorithm, we see that $\gamma$ initially computes $1 = (q^a - 1)/(q - 1) = w_a$, and computes $w_i$ for all $i \leq L(\varepsilon)$, by (10). In particular, the two values $w_{a(i)}$ and $w_{a_k(i)}$ used in step (2) of the algorithm are actually computed by the previous version of $\gamma$, and correctness is clear. We have $A(\gamma) = L(\varepsilon)$ and also $Q(\gamma) \leq \sum_{1 \leq i \leq L(\varepsilon)} a_k(i)$. To show the lower bound on $Q(\gamma)$, we prove by induction on $i$ that $Q(\gamma|_{w_i}) \geq a_i - 1$. For $i = 0$ we have $Q(\gamma|_{w_0}) = 0 \geq 1 - 1$. From the induction hypothesis for $j(i) < i$, we have $Q(\gamma|_{w_{j(i)}}) \geq Q(\gamma|_{w_{j(i)}}) + a_k(i) \geq a_j(i) - 1 + a_k(i) = a_i - 1$. In particular, $Q(\gamma) = Q(\gamma|_{w_{L(\varepsilon)}}) \geq a_{L(\varepsilon)} = n - 1$. \qed
Example 12. Let $q = 2$, let $n = 22$ and let $\varepsilon$ be the first addition chain for 22 in Example 11. Algorithm 10 yields the following addition chain $\gamma$ for $e = 2^{22} - 1$. An edge from $a_j(i)$ to $a_i$ labeled with “2” abbreviates the intermediate doublings $2 \cdot a_j(i), \ldots, 2^m \cdot a_j(i)$.

Here $w_{a(i)} = 2^{a(i)} - 1$ for $0 \leq i \leq 6$: 1, 3, 7, 31, 63, 2047, 4194303. There are exactly $A(\gamma) = L(\varepsilon) = 6$ additions and $Q(\gamma) = n - 1 = 21$ doublings. The new graph is obtained by multiplying the bold edges in the top graph of Example 11 with $2^{w(i)}$.

The number $Q(\gamma)$ of $q$-steps is not necessarily equal to $\sum_{0 \leq i \leq L(\varepsilon)} a_k(i)$. An example is given by the second addition chain for 22 in Example 11. Here $\sum_{0 \leq i \leq 6} a_k(i) = 23$ but the addition chain $\gamma$ for $(q^{22} - 1)/(q - 1)$ contains only $Q(\gamma) = 21$ different $q$-steps. The computation of $w_6 = w_3 \cdot q + w_3$ can profit from the previous computation of $w_5 = w_3 \cdot q^2 + w_2$ since the element $w_3 \cdot q^2$ is already in $\gamma$. Thus only one further $q$-step has to be performed for its first summand. For a star addition chain, equality always holds by Lemma 5.

Corollary 13. Let $n, q \in \mathbb{N}_{\geq 2}$, let $e = (q^n - 1)/(q - 1)$, and let $\varepsilon$ be a star addition chain for $n$. Then the $q$-addition chain $\gamma$ for $e$ uses $L(\varepsilon)$ additions and $n - 1$ many $q$-steps. In particular,

$$\ell_q(e) \leq L^*(n) + n - 1.$$  

The case $q = 2$ was proven by Brauer [6]:

$$\ell_2(2^n - 1) \leq \ell_2^*(2^n - 1) \leq \ell_2^*(n) + n - 1.$$  

Scholz [15] and Brauer [6] conjectured that $\ell_2(2^n - 1) \leq \ell_2(n) + n - 1$. This Scholz-Brauer conjecture is the most prominent open problem in the theory of addition chains. Corollary 13 means that we can compute $e = (q^n - 1)/(q - 1)$ using only $O(\log n)$ non-$q$-steps instead of $O(n/\log n)$ with Brauer’s addition chain (Theorem 3). This is an exponential improvement on the number of non-$q$-steps. Applied to ordinary addition chains, that is, 2-addition chains, it says that there always exist reasonably short chains almost all of whose operations are doublings.

Corollary 14. Let $n \in \mathbb{N}$, $q = 2$, and $e = 2^n - 1$. Then Algorithm 11 computes an addition chain for $e$ which is at most $\lfloor \log_2 \log_2 (e+1) \rfloor + 2.13$ longer than Schönhage’s lower bound 5.

Proof. We have $\nu_2(e) = n$, and $\log_2 e < n$. Then $\ell^*_2(n) \leq \lambda_2(n) + \nu_2(n) - 2 \leq \log_2 n + \nu_2(n) - 1$. Hence for the length $L(\gamma) = \ell^*_2(n) + n - 1$ of the addition chain $\gamma$ from Algorithm 11 this yields

$$L(\gamma) - (\log_2 e + \log_2 \nu_2(e) - 2.13) = \ell^*_2(n) + n - 1 - \log_2 e - \log_2 \nu_2(e) + 2.13 \leq \log_2 n + \nu_2(n) - 1 + n - 1 - (n - 1) - \log_2 n + 2.13 = \nu_2(n) + 1.13 \leq \lambda_2(n) + 1.13 = \lfloor \log_2 \log_2 (e+1) \rfloor + 2.13.$$  

□
Downey et al. [43] show that if for computing $2^n - 1$, one insists on doing the doubling steps first, so that $2, 2^2, 2^3, \ldots, 2^{n-1}$ are computed, then one has to use $\sqrt{n}$ further steps rather than just $O(\log n)$.

4. Addition chains with weighted length

Starting in this section, we will see applications where $q$-steps are much cheaper than other steps when applied to the exponentiation problem. In order to model this, we consider as our cost measure the weighted length $L_{(c_A, c_Q)}(\gamma) = c_A \cdot A(\gamma) + c_Q \cdot Q(\gamma)$ of $\gamma$, for a pair $(c_A, c_Q) \in \mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 1}$ of nonnegative constants. The unweighted (usual) length equals $L_{(1,1)}$. Let $q$ be a prime power. We can regard $\mathbb{F}_{q^n}$ as a vector space of dimension $n$ over $\mathbb{F}_q$, and we consider two types of bases, which illustrate the use of this measure.

Let $f \in \mathbb{F}_q[x]$ be an irreducible polynomial of degree $n$. Then we have $\mathbb{F}_{q^n} \cong \mathbb{F}_q[x]/(f)$, the $\alpha_i = x^i$ mod $f$ with $0 \leq i < n$ form a basis, and any element of $\mathbb{F}_{q^n}$ can be represented by a polynomial of degree at most $n - 1$. Within this polynomial basis representation we use fast polynomial arithmetic. We call a function $M : \mathbb{N}_{\geq 0} \rightarrow \mathbb{R}_{>0}$ a multiplication time for $\mathbb{F}_q[x]$ if polynomials in $\mathbb{F}_q[x]$ of degree less than $n$ can be multiplied using at most $M(n)$ operations in $\mathbb{F}_q$. Classical polynomial multiplication yields $M(n) \leq 2n^2$. We can take $M(n) \in O(n \log n \log \log n)$ according to Schönhage and Strassen [49] and Schönhage [48]. Counting the operations in $\mathbb{F}_q$, we should thus use $c_A = M(n)$ and $c_Q = M(n)\ell_2(q) \leq 2M(n) \log_2 q$.

Another representation of $\mathbb{F}_{q^n}$ uses a normal basis $\mathcal{N} = (\alpha_0, \ldots, \alpha_{n-1})$ with $\alpha_i = \alpha_0^q^i$ for $1 \leq i < n$; surveys on this topic can be found for example in [36] and [41]. Then $\alpha_0 \in \mathbb{F}_{q^n}$ is called a normal element over $\mathbb{F}_q$. Let $\beta \in \mathbb{F}_{q^n}$ be given in this normal basis representation as $\beta = \sum_{0 \leq i < n} b_i \alpha_i$ with all $b_i \in \mathbb{F}_q$. Then $\beta^q = (\sum_{0 \leq i < n} b_i \alpha_i)^q = \sum_{0 \leq i < n} b_i \alpha_i^q = \sum_{0 \leq i < n} b_{i-1} \alpha_i$ with index arithmetic modulo $n$. Hence raising to the $q$th power is just a cyclic shift of the coefficients and is therefore essentially free in this representation. We model this by setting $c_Q = 0$ for a normal basis representation.

Experiments in [23] show that multiplication for an arbitrary normal basis representation in $\mathbb{F}_{2^n}$ is significantly slower than for a polynomial basis representation if implemented in software. But Gao et al. [17] provide a way to connect fast multiplication (using the polynomial basis representation in a larger ring) and free raising to the $q$th power in $\mathbb{F}_{q^n}$ (using normal basis representation). Their idea is based on a special normal basis for $\mathbb{F}_{q^n}$ generated by Gauß periods.

Definition 15. Let $n, k \in \mathbb{N}_{\geq 1}$ be such that $r = nk + 1$ is prime. Then $\mathcal{K} \subseteq \mathbb{Z}_r^*$ be the unique subgroup of $\mathbb{Z}_r^*$ of order $k$, and let $\xi$ be a primitive $r$th root of unity in $\mathbb{F}_{q^k}$. Then $\alpha = \sum_{\alpha \in \mathcal{K}} \xi^\alpha$ is called a Gauß period of type $(n, k)$ over $\mathbb{F}_q$.

A Gauß period of type $(n, k)$ generates a normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ if and only if $\gcd(e, n) = 1$, where $e$ is the index of $q$ modulo $r$; see [10], [32], and [17].

Fact 16 (Gao et al. [17]). Let $\alpha \in \mathbb{F}_{q^n}$ be a normal Gauß period of type $(n, k)$. Then two elements in $\mathbb{F}_{q^n}$ given in the normal basis representation generated by $\alpha$ can be multiplied with $M(kn) + 2kn - 1$ operations in $\mathbb{F}_q$.

We can model this situation by setting
\begin{equation}
(17)
c_A = M(kn) + 2kn - 1, \quad c_Q = 0.
\end{equation}
If \( \gamma \) is a \( q \)-addition chain for \( e < q^n \), then the weighted length \( L_{(M(kn)+2kn-1,0)}(\gamma) \) counts the number of operations in \( \mathbb{F}_q \) for calculating \( \beta^e \in \mathbb{F}_{q^n} \) for given \( \beta \in \mathbb{F}_q \).

5. INVERSION IN \( \mathbb{F}_{q^n} \)

We use addition chains for repunits and combine them with normal bases generated by Gauß periods. With these tools we compute the inverse of an element in \( \mathbb{F}_{q^n}^\times \) in the same asymptotic time as via the Extended Euclidean Algorithm (EEA). Our experimental running times for \( q = 2 \) are, in favorable circumstances, about 72\% of that of the EEA (for example, for \( n = 51282 \)).

This approach via an addition chain for \( n \) can also be found in the papers of Wang et al. [52], Itoh and Tsujii [35], Asano et al. [1], and Xu [54]. In all papers, preselected addition chains are used to compute \( n - 1 \). Itoh and Tsujii [35] employ the binary addition chain; a recursive version can be found in Itoh and Tsujii [34]. In a later paper, Asano et al. [1] use a variant of the factor method; see [39], [38], for a presentation of the factor method. Our approach allows an arbitrary star addition chain for \( n \) as an input, giving an average speed-up of about 1.13 for the fields \( \mathbb{F}_{2^n} \) displayed in Figure 1.

**Inversion using Fermat.** Fermat’s Little Theorem says that \( \beta^{-1} = \beta^{q^n-2} \) for \( \beta \in \mathbb{F}_{q^n}^\times \). Setting \( e = (q^n-1)/(q-1) \), we have

\[
(18) \quad q^n - 2 = e \cdot (q-1)q + (q-2).
\]

We use the methods of Section 3 to obtain a \( q \)-addition chain for \( q^n - 2 \).

**Algorithm 19** (\( q \)-addition chain for \( q^n - 2 \)).

**Input:** \( n, q \in \mathbb{N}_{\geq 2} \), an addition chain \( \varepsilon \) for \( n - 1 \), and an addition chain \( \delta \) for \( q - 2 \).

**Output:** A \( q \)-addition chain \( \gamma \) for \( q^n - 2 \).

1. Set \( \gamma \leftarrow \delta \oplus 1 \).
2. Compute a \( q \)-addition chain \( \eta \) for \( e = (q^n-1)/(q-1) \) using Algorithm 14 with input \( n - 1, q, \) and \( \varepsilon \). Compute \( \gamma \leftarrow \gamma \circ \eta \).
3. Compute \( \gamma \leftarrow \gamma \oplus q \). Set \( \gamma \leftarrow \gamma \oplus (q-2) \).
4. Return \( \gamma \).

**Lemma 20.** Let \( \varepsilon \) be a star addition chain. Then Algorithm 17 computes a \( q \)-addition chain \( \gamma \) for \( q^n - 2 \) with

1. \( A(\gamma) = L(\varepsilon) + L(\delta) + 2 \) additions, \( Q(\gamma) = n - 1 \) many \( q \)-steps, and \( L(\gamma) = L(\varepsilon) + L(\delta) + n + 1 \) if \( q > 2 \).
2. \( A(\gamma) = L(\varepsilon) \) additions, \( Q(\gamma) = n - 1 \) doublings, and \( L(\gamma) = L(\varepsilon) + n - 1 \) if \( q = 2 \).

**Proof.** The correctness of Algorithm 17 follows directly from 15.

If \( q > 2 \), then we have \( L(\delta) + 2 \) additions in steps (1) and (3) of the algorithm (since a chain for \( q - 2 < q \) has no \( q \)-step) and one \( q \)-step. According to Theorem 11 for a star addition chain \( \varepsilon \), the \( q \)-addition chain \( \eta \) for \( e \) contains \( L(\varepsilon) \) additions and \((n - 1) - 1\) many \( q \)-steps.

For \( q = 2 \), step (1) can be skipped. Step (3) contains only one doubling because \( q - 2 = 0 \). Therefore we have \( L(\varepsilon) \) additions and \( n - 2 + 1 = n - 1 \) doublings. \( \square \)

If \( \varepsilon \) is not a star chain, then \( Q(\gamma) \leq \sum_{1 \leq i \leq L(\varepsilon)} a_k(i) \), where \( S(\varepsilon) = \{a_0, \ldots, a_{L(\varepsilon)}\} \).
Example 21. Let $q = 2$ and $n = 23$. We can compute $\beta^{-1}$ for $\beta \in \mathbb{F}_{23}^{n}$ using the star addition chain $\varepsilon$ of Example 12 for $n - 1 = 22$. This leads to the values $\beta^1, \beta^3, \beta^7, \beta^{31}, \beta^{63}, \beta^{2047}, \beta^{4194303} = \beta^{222 - 1}$, where 21 doubling steps are not shown. Finally we compute $(\beta^{4194303})^2 = \beta^{8388606} = \beta^{223 - 2} = \beta^{-1}$. Hence $\beta^{-1}$ can be computed using 6 multiplications and 22 squarings in $\mathbb{F}_{23}^{n}$.

Theorem 22. The inverse of an element of $\mathbb{F}_{q^n}$ can be computed with at most

1. $\ell'_2(n - 1) + \ell_2(q - 2) + 2$ multiplications and $n - 1$ many $q$th powers in $\mathbb{F}_{q^n}$ if $q > 2$.
2. $\ell'_2(n - 1)$ multiplications and $n - 1$ squarings if $q = 2$.

If we use Brauer’s addition chain (Theorem 9), we have

$$\ell'_2(n - 1) \leq \lambda_2(n - 1) + \frac{\lambda_2(n - 1)}{\log_2 \lambda_2(n - 1)}(1 + o(1))$$

and

$$\ell_2(q - 2) \leq \lambda_2(q - 2) + \frac{\lambda_2(q - 2)}{\log_2 \lambda_2(q - 2)}(1 + o(1)).$$

Combining this with Fact 16 we get the following result.

Corollary 23. If we have a normal basis of type $(n, k)$ for $\mathbb{F}_{q^n}$ as in Fact 16 then we may use (17) and can invert in $\mathbb{F}_{q^n}^{*}$ using

1. $c_{A} \cdot \left( \lambda_2(n - 1) + \frac{\lambda_2(n - 1)}{\log_2 \lambda_2(n - 1)}(1 + o(1)) + \lambda_2(q - 2) + \frac{\lambda_2(q - 2)}{\log_2 \lambda_2(q - 2)}(1 + o(1)) \right) \in O(M(kn) \log(nq))$ operations in $\mathbb{F}_{q}$ if $q > 2$,
2. $c_{A} \cdot \left( \lambda_2(n - 1) + \frac{\lambda_2(n - 1)}{\log_2 \lambda_2(n - 1)}(1 + o(1)) \right) \in O(M(kn) \log n)$ operations in $\mathbb{F}_{2}$ if $q = 2$.

For small $k$ (we choose $k \in \{1, 2\}$ for our experiments) we get $O(M(n) \log n)$ if $q$ is much smaller than $n$. Gauss periods of type $(n, 1)$ or $(n, 2)$ do not exist for all $q$ and $n$, but they seem to exist for a reasonably dense set of values of $n$, e.g., for 23% of all $n \leq 1200$ if $q = 2$; see [12]. The percentage of fields $\mathbb{F}_{q^n}$ for which optimal normal bases do exist for some small primes $q$ and $n < 10000$ is given below.

<table>
<thead>
<tr>
<th>$q$</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>11</th>
<th>13</th>
<th>17</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>17.07*</td>
<td>4.76</td>
<td>4.92</td>
<td>4.65</td>
<td>4.43</td>
<td>4.57</td>
<td>4.50</td>
<td>4.72</td>
</tr>
</tbody>
</table>

*For $q = 2$, we have two different types of optimal normal bases: the first one appears in 4.70% of the field extensions over $\mathbb{F}_{2}$, and the second one exists in 12.37%.

See [28] for general results concerning the density of Gauss periods. Feisel et al. [15] have extended the notion of Gauss periods, and von zur Gathen and Nöcker [20] provide fast algorithms also for this generalization.

Inversion using the Extended Euclidean Algorithm. Let $\mathbb{F}_{q^n}$ be given by a polynomial basis representation $\mathbb{F}_{q}[x]/(f)$ with $f$ irreducible and of degree $n$. The canonical representative of $\beta \in \mathbb{F}_{q^n}$ is the unique polynomial $g \in \mathbb{F}_{q}[x]$ of degree less than $n$ such that $(g \mod f) = \beta$. The inverse of $\beta$, if nonzero, can be computed with the Extended Euclidean Algorithm. Lehmer [10], Knuth [39], Schönhage [46],
and Strassen [51] introduced fast versions of the Euclidean Algorithm based on the divide-and-conquer technique; see [20], Section 11, for a presentation.

**Fact 24.** The inverse of an element of \( \mathbb{F}_{q^n}^\times \) given in a polynomial basis representation can be calculated with \( O(M(n) \log n) \) operations in \( \mathbb{F}_q \).

Gao et al. [17] have shown how to combine the fast Extended Euclidean Algorithm with normal bases.

**Fact 25 (Gao et al. [17]).** The inverse of an element of \( \mathbb{F}_{q^n}^\times \) given in a normal basis representation generated by a Gauß period of type \((n,k)\) can be calculated with \( O(M(kn) \log (kn)) \) operations in \( \mathbb{F}_q \).

Therefore all three ways to compute the inverse of an element in \( \mathbb{F}_{q^n} \) use \( O(M(n) \log n) \) operations in \( \mathbb{F}_q \), provided we have a Gauß period of type \((n,k)\) with small \( k \). Since the theory cannot distinguish between their costs, we revert to experiment.

**Experimental results.** We have implemented all three inversion algorithms on a LINUX-PC with two pentium II-processors, rated at 500 MHz. The software is written in C++. The coefficient lists of both the polynomial and the normal basis representation are represented as arrays of 32-bit unsigned integers, and 32 consecutive coefficients are packed into one machine word. For polynomial arithmetic we use the C++-library BiPolAr that is described in [19], [21]; see also [20], Section 9, and [5] for the factorization of a polynomial with degree more than one million. This library offers fast polynomial arithmetic over \( \mathbb{F}_2 \) including several algorithms for polynomial multiplication over \( \mathbb{F}_2 \): the classical method, the algorithm of Karatsuba in [37], and the method introduced by Cantor [10]. We only deal with field extensions of \( \mathbb{F}_2 \) of degree \( n \) for which a so-called optimal normal basis exists, that is, a normal Gauß period of type \((n,k)\) with \( k \in \{1, 2\} \), and we show the results in Figures 1 and 2. In the first of these, we have small degrees \( n \approx 200i \) for \( 1 \leq i \leq 50 \), and in the second one, some large degrees. Each figure displays the timings for four algorithms, averaged over 100 random inputs. In the first three, the extension \( \mathbb{F}_{2^n} \) of \( \mathbb{F}_2 \) is represented by a normal Gauß period, and in the last one, by a polynomial basis. The algorithms for inversion are Fermat’s formula for the first two, with the binary and an optimal addition chain, respectively. The optimal chains come from Knuth’s [30] power tree. For the last two inversion methods, we use Euclid’s algorithm.

With a normal Gauß period, the multiplication cost depends (essentially linearly) on the parameter \( k \), as stated in Fact 15. This is clearly visible in the figures as the two curves for one algorithm, one corresponding to \( k = 1 \) and the other to \( k = 2 \).

In Figure 2, we have chosen pairs of values for \( n \) which are close to each other and where there exist Gauß periods with \( k = 1 \) for one value and with \( k = 2 \) for the other value.

In a polynomial basis of \( \mathbb{F}_{2^n} \), modulo a random irreducible polynomial, an inverse is computed by the Extended Euclidean Algorithm (labelled polynomial: Euclid). In contrast to the normal basis representation (labelled normal Gauß periods: Euclid), the problem size depends no longer on a blow-up factor \( k \), and the times are close to the EEA for normal Gauß periods of type \((n,1)\). At \( n = 61716 \), the latter takes less than 80% of the time of the polynomial Euclidean algorithm.
The upshot of our experiments is: for small degrees, polynomial Euclid is best, and for large degrees, say over 16000, Fermat with Gauss periods of type \((n,1)\) is fastest (if such a period exists).

6. **Addition chains for special sets**

In this section, we describe an efficient method for exponents which divide \(q^n - 1\). This will be applied to primitivity testing in \(\mathbb{F}_{q^n}\) in the next section.

**Addition chains for \((q^n - 1)/t\).** Let \(n, q, t \in \mathbb{N}_{\geq 1}\) with \(t \geq 1\) dividing \(q^n - 1\). Then \(e = (q^n - 1)/t \in \mathbb{N}_{\geq 1}\) and \((e)_q\) has a regular structure. To see why, we consider the \(q\)-ary representation of \(1/t = \sum_{i \leq -1} t_i q^i\) with \(0 \leq t_i < q\) for all \(i\). Then \((1/t)_q = (t_{-1}, t_{-2}, \ldots)\) is unique if \(t_i \neq q - 1\) for infinitely many \(i\).

\((1/t)_q\) is called periodic if there exist \(v, w \in \mathbb{N}_{\geq 0}\) with \(t_{-(w+j)} = t_{-j}\) for all \(j \geq v\), and the minimal such \(w\) is the length of the period. The sequence \(t_{-1}, \ldots, t_{-v}\), for minimal \(v\), is called the preperiod of length \(v\). Because \(t\) divides \(q^n - 1\), we have \(\gcd(t, q) = 1\). The following lemma determines the length of the period; see [30], article 313, or [31], Satz 5.

**Lemma 26.** Let \(t, q \in \mathbb{N}_{\geq 1}\) with \(\gcd(t, q) = 1\). Then \(w = \text{ord}_t(q) = \min\{j \in \mathbb{N}_{>0} : q^j \equiv 1 \pmod{t}\}\) is the length of the period of \((1/t)_q\), and \(w\) divides \(n\). The preperiod has length zero.

Let \(s\) be the period of \((1/t)_q\) with length \(\lambda_q(s) = w = \text{ord}_t(q)\), and \(1/t = \sum_{i \leq -1} s q^{wi} = s \cdot \sum_{1 \leq i < \infty} (1/q^w)^i = s \cdot (q^w/(q^w - 1) - 1) = s/(q^w - 1)\). Therefore
Inversion in extensions of the binary field

Figure 2. Results for large degrees.

$s = (q^w - 1)/t$. Because $q^n \equiv 1 \mod t$, $w$ divides $n$ and $m = n/w$ is the number of repetitions of $s$ in

$$(e)_q = ((q^n - 1)/t)_q = ((q^w - 1)/t \cdot \sum_{0 \leq i < m} q^{wi})_q = (s \cdot \sum_{0 \leq i < m} q^{wi}) = (s, \ldots, s).$$

We call such integers $q^w$-ary repdigits in what follows. Now we derive $q$-addition chains for repdigits from $q$-addition chains for repunits.

**Algorithm 27** ($q$-addition chain for repdigits).

Input: $n, q, w, t \in \mathbb{N}_{\geq 1}$ with $q \geq 2$, $w$ dividing $n$, and $t$ dividing $q^w - 1$, an addition chain $\gamma$ for $n/w$, and a $q$-addition chain $\delta$ for $s = (q^w - 1)/t$.

Output: A $q$-addition chain $\varepsilon$ for $e = (q^n - 1)/t$.

1. Using Algorithm 10 with input $n/w$, $q^w$, and $\gamma$, compute a $q^w$-addition chain $\eta$ for $e/s = (q^w n/w - 1)/(q^w - 1) = \sum_{0 \leq i < n/w} q^{wi}$.
2. Transform $\eta$ into a $q$-addition chain $\eta'$ by substituting $w$ single $q$-steps for each $q^w$-step.
3. Return $\varepsilon \leftarrow \delta \odot \eta'$.

**Theorem 28.** Let $n, q, t \in \mathbb{N}_{\geq 1}$ with $q \geq 2$ and $t$ dividing $q^n - 1$. Let $w = \text{ord}_4(q)$, $s = (q^w - 1)/t$, let $\gamma$ and $\delta$ be the input chains for Algorithm 27, and assume that $\gamma$ is a star addition chain. Then the algorithm furnishes a $q$-addition chain $\varepsilon$ for $e = (q^n - 1)/t$ with at most $L(\gamma) + A(\delta)$ additions and $n - w + Q(\delta)$ $q$-steps. In particular,

$$\ell_q(e) \leq \ell^*_2(n/w) + (n - w) + \ell_q(s).$$

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
Proof. Since \( s \cdot e/s = e \), correctness is clear. Concerning the length of \( \varepsilon \), Corollary 13 says that \( \eta \) has \( A(\eta) = L(\gamma) \) additions and \( n/w - 1 \) many \( q^w \)-steps. Then \( Q(\eta') = w \cdot (n/w - 1) \). The product chain \( \varepsilon \) has \( A(\varepsilon) = A(\delta) + A(\eta') \) additions and \( Q(\varepsilon) = Q(\delta) + Q(\eta') \) many \( q \)-steps.

The last claim follows by choosing optimal chains \( \gamma \) and \( \delta \).

This method is useful when \( t \) is small; then also \( w \) and \( s \) are fairly small.

Example 29. Let \( q = 2 \) and \( n = 22 \) again, and let \( t = 3 \). Then 3 divides \( (2^{22} - 1) = 4194303 \), and \( e = (2^{22} - 1)/3 = 1398101 \). We have \( w = \text{ord}_3(2) = 2 \) and \( m = 22/2 = 11 \), \( s = (2^2 - 1)/3 = 1 \), and \( (s)_4 = (1) \). The 4-ary representation of \( e \) illustrates this:

\[
(2^{22} - 1)/3_4 = (1111111111111_{m=11}).
\]

The addition chain for \( s \) has length 0. Thus we can compute an \( e \)-th power of an element in \( \mathbb{F}_2^{22} \) (or in any ring) with \( \ell_2(11) = 5 \) multiplications, using the first addition chain of Example 1 restricted to 11, plus \((11 - 1) \cdot 2 = 20 \) squarings.

Exponent sets. Let \( \mathcal{E} \subset \mathbb{N}_{>1} \) be a finite set. A \( q \)-addition chain \( \varepsilon \) computes \( \mathcal{E} \) if \( \mathcal{E} \subseteq S(\varepsilon) \). This is a natural generalization of the previous definition for \( \mathcal{E} = \{e\} \).

We set \( \ell_q(\mathcal{E}) = \min \{ L(\varepsilon) : \varepsilon \text{ computes } \mathcal{E} \} \), and then we have

\[
\max \{ \ell_q(e) : e \in \mathcal{E} \} \leq \ell_q(\mathcal{E}) \leq \sum_{e \in \mathcal{E}} \ell_q(e).
\]

We can modify the algorithm used for Theorem 6 to compute a \( q \)-addition chain \( \varepsilon \) for \( \mathcal{E} \). We precompute \( \{1, \ldots, q^r - 1\} \) once, using \( q^r - 1 \) many \( q \)-steps and \( q^r - q^{r-1} - 1 \) further steps. The number of steps left for each element \( e \in \mathcal{E} \) is \( \nu_q(e) - 1 \) additions and at most \( r \cdot (\lambda_{q^r}(e) - 1) \leq \lambda_q(e) \) many \( q \)-steps. Setting \( d = \#\mathcal{E}, \nu = \max \{ \nu_q(e) : e \in \mathcal{E} \} \), and \( m = \max \mathcal{E} \), we get the following bounds on the cost:

\[
\begin{align*}
A(\varepsilon) & \leq q^r - q^{r-1} - 1 + d \cdot (\nu - 1), \\
Q(\varepsilon) & \leq q^{r-1} - 1 + dr \cdot (\lambda_{q^r}(m) - 1), \\
\ell_q(\mathcal{E}) & \leq q^r - 2 + d \cdot (\nu + \lambda_q(m) - 1).
\end{align*}
\]

Yao 55 gives a better upper bound for \( q = 2 \). Further results on this problem are in 13 and 8, 32 gives an overview. We can adapt this result to \( q \)-addition chains.

Fact 30 (Yao 55). Let \( q \in \mathbb{N}_{\geq 2}, \mathcal{E} \subset \mathbb{N}_{\geq 1} \) be finite, \( m = \max \mathcal{E} \), and \( d = \#\mathcal{E} \).

Then there exists a \( q \)-addition chain \( \varepsilon \) for \( \mathcal{E} \) with at most \( \sum_{e \in \mathcal{E}} \frac{\lambda_q(e)}{\log_q \lambda_q(e)}(1 + o(1)) \leq d \cdot \frac{\lambda_q(m)}{\log_q \lambda_q(m)}(1 + o(1)) \) additions and at most \( \lambda_q(m) \) many \( q \)-steps.

A good systematic way we have for computing a set \( \mathcal{E} \) is to take separate Brauer chains for each \( e \in \mathcal{E} \), with the same value of \( r \), and to remove doubles.

7. Testing primitivity in \( \mathbb{F}_q^n \)

We use the periodic form of the \( q \)-ary representation of \( (q^n - 1)/p \) to apply our short addition chains for repdigits to the problem of testing for primitivity. In our experiments we compare these chains with the general addition chain algorithm of Brauer (Theorem 6). This method reduces the number of multiplications by a
factor up to 7.96 (for \( n = 841 \)). Using a normal basis generated by Gauß periods this speeds up the running time in the same manner. On average our addition chains contain about half as many multiplications as the general chains (Table 2).

A test for primitivity. When one wants to find a primitive element by choosing random elements and testing them for primitivity, one expects to need about \((q^n - 1)/\varphi(q^n - 1)\) choices, where \(\varphi\) is Euler’s totient function. If this number is fairly large—which happens when \(q^n - 1\) has many different small prime factors—it may pay to invest in designing a good addition chain for this computation. The order

\[
\text{ord}_{\mathbb{F}_q}(\beta) = \min\{w \in \mathbb{N}: w \geq 1, \beta^w = 1\}
\]

of \(\beta \in \mathbb{F}_{q^n}^\times\) is a divisor of \(q^n - 1\), and \(\beta\) is primitive if and only if \(\text{ord}_{\mathbb{F}_q}(\beta) = q^n - 1\). Thus \(\beta\) is primitive if and only if \(\beta^{(q^n - 1)/p} \neq 1\) for all primes \(p\) dividing \(q^n - 1\). See [22] for the average order in \(\mathbb{F}_{q^n}^\times\) and [7] for computing large primitive trinomials over \(\mathbb{F}_2\).

The corresponding algorithm requires the set \(\mathcal{P}\) of all prime factors of \(q^n - 1\) as input. This is the true bottleneck for any primitivity-testing algorithm known so far. Finding \(\mathcal{P}\) is difficult for moderate \(n\) and practically impossible for huge \(n\). For \(2 \leq q \leq 12\), tables of factorizations of \(q^n - 1\) are published by the Cunningham Project serviced by Paul Leyland (see the information on ftp://sable.ox.ac.uk/pub/math/cunningham/); a historical overview of this project is given in [9]. We use these tables for our experimental results below. It is well known that the number \(\omega(k)\) of prime divisors of \(k\) is at most \(\ln k / \ln \ln k\) (and roughly this large if \(k\) is the product of the first primes), and \(\ln \ln x + B_1 + o(1)\) for the \(k \leq x\) on average, with \(B_1 = C + \sum_{p \leq x} (\ln(1 - 1/p) + 1/p)\), where \(\prod_{p \leq x} (1 - 1/p) \approx e^{-C} / \ln x\), and Euler’s constant \(C = \lim_{n \to \infty} (1 + \frac{1}{2} + \cdots + \frac{1}{n} - \ln n) \approx 0.57722\); see [33], §22.10. The averages reported in Table 2 for \(k = 2^n - 1\) are somewhat higher than \(\ln \ln k + B_1\).

We connect Theorem [28] and Fact [30] to compute a \(q\)-addition chain for the set \(\mathcal{E} = \{(q^n - 1)/p: p \in \mathcal{P}\}\), where \(\mathcal{P}\) is the set of prime divisors of \(q^n - 1\). The idea is as follows: For each \(p \in \mathcal{P}\) we set \(w(p) = \text{ord}_{\mathbb{F}_q}(q)\), \(s(p) = (q^{e(p)} - 1)/p\), and \(e(p) = (q^n - 1)/p\). We start in a first step by generating a \(q\)-addition chain \(\delta\) for the set \(\mathcal{S} = \{s(p): p \in \mathcal{P}\}\) using the algorithm behind Fact [30]. This \(\delta\) has at most

\[
A(\delta) \leq \sum_{p \in \mathcal{P}} \lambda_q(s(p)) / \log_q \lambda_q(s(p)) \cdot (1 + o(1)) \text{ additions and}
\]

\[
Q(\delta) \leq \lambda_q(m) \text{ many } q\text{-steps}
\]

where \(m = \max \mathcal{S}\). Furthermore we assume that for each \(p \in \mathcal{P}\) we have a star addition chain \(\gamma(p)\) computing \(n/w(p)\). In the second step we apply for every \(p \in \mathcal{P}\) Algorithm [27] to the input that consists of the integers \(n, q, w(p), p\) and the addition chains \(\gamma(p)\) for \(n/w(p)\) and \(\delta(p) = \delta_{s(p)}\) for \(s(p)\). Let the resulting \(q\)-addition chain computing \(e(p) = (q^n - 1)/p\) be \(\varepsilon(p)\). This chain has at most

\[
A(\varepsilon(p)) \leq L(\gamma(p)) + A(\delta_{s(p)}) \text{ non-}q\text{-steps and}
\]

\[
Q(\varepsilon(p)) \leq n \cdot w(p) + Q(\delta_{s(p)}) \text{ many } q\text{-steps}
\]

by Theorem [28]. Then the \(q\)-addition chain computing \(\varepsilon\) is the concatenation \(\varepsilon = \bigcup_{p \in \mathcal{P}} \varepsilon(p)\) with \(L(\varepsilon) \leq \sum_{p \in \mathcal{P}} L(\varepsilon(p))\).
Corollary 31. Let \( n, q \in \mathbb{N}_{\geq 2} \), let \( \mathcal{P} \) be the set of prime divisors of \( q^n - 1 \), let \( \mathcal{E} = \{(q^n - 1)/p: p \in \mathcal{P}\} \), \( d = \# \mathcal{P} \), and let \( \delta \) be a \( q \)-addition chain computing \( \{(q^n - 1)/p: w = \text{ord}_p(q), p \in \mathcal{P}\} \). Then there exists a \( q \)-addition chain \( \varepsilon \) for \( \mathcal{E} \) with

\[
A(\varepsilon) \leq A(\delta) + \sum_{p \in \mathcal{P}} \ell_2(n/\text{ord}_p(q)),
\]

\[
Q(\varepsilon) \leq Q(\delta) + \sum_{p \in \mathcal{P}} (n - \text{ord}_p(q)) = Q(\delta) + dn - \sum_{p \in \mathcal{P}} \text{ord}_p(q).
\]

Examples are given below.

Corollary 32. Let \( n, q \in \mathbb{N}_{\geq 2} \), let \( \mathcal{P} \) be the set of prime divisors of \( q^n - 1 \) as above, let \( d = \# \mathcal{P} \), and let \( s = \max\{q^{\text{ord}_p(q)} - 1/p: p \in \mathcal{P}\} \). We can test an element \( \beta \in \mathbb{F}_{q^n} \) for primitivity using at most

\[
d \cdot \left( \frac{\log_q(s)}{\log_q \log_q s} \cdot (1 + o(1)) + 2 \log_2 n \right)
\]

multiplications in \( \mathbb{F}_{q^n} \), plus \([\log_q s] + dn\) many \( q \)th powers.

Proof. The proof follows from Corollary 31. We set \( w(p) = \text{ord}_p(q) \) for all \( p \in \mathcal{P} \) and \( S = \{(q^{w(p)} - 1)/p: p \in \mathcal{P}\} \). We have \( s = \max S < q^n - 1 \). By Fact 31 there is a \( q \)-addition chain \( \delta \) for \( S \) with \( Q(\delta) \leq \lambda_q(s) \) and \( A(\delta) \leq d \frac{\lambda_q(s)}{\log_q \lambda_q(s)} \cdot (1 + o(1)) \).

Corollary 31 says there exists a \( q \)-addition chain \( \varepsilon \) computing \( \mathcal{E} = \{(q^n - 1)/p: p \in \mathcal{P}\} \) with \( A(\varepsilon) \leq A(\delta) + \sum_{p \in \mathcal{P}} \ell_2(n/w(p)) \) and \( Q(\varepsilon) \leq Q(\delta) + dn - \sum_{p \in \mathcal{P}} w(p) \).

We can estimate \( \ell_2(n/w(p)) \leq \lambda_2(n/w(p)) + \nu_2(n/w(p)) - 2 \leq 2 \log_2 n \) with (7), since the binary addition chain is a star addition chain. Inserting this and the estimates on \( A(\delta) \) and \( Q(\delta) \) yields

\[
A(\varepsilon) \leq d \frac{\lambda_q(s)}{\log_q \lambda_q(s)} \cdot (1 + o(1)) + 2d \log_2 n,
\]

\[
Q(\varepsilon) \leq [\log_q s] + 1 + dn - \sum_{p \in \mathcal{P}} w(p) \leq \log_2 s + dn. \quad \square
\]

Example 33. Let \( q = 2 \) and \( n = 22 \). From \( 2^{22} - 1 = 3 \cdot 23 \cdot 89 \cdot 683 \) we have \( \mathcal{P} = \{3, 23, 89, 683\} \) and \( \# \mathcal{P} = 4 \). We use \( w_i \) for the order of 2 modulo \( p_i \), and \( e_i = (2^{22} - 1)/p_i \) for a given prime divisor \( p_i \) of \( 2^{22} - 1 \).

<table>
<thead>
<tr>
<th>( i )</th>
<th>( p_i )</th>
<th>( e_i )</th>
<th>( (e_i)_{2} )</th>
<th>( s_i )</th>
<th>( w_i )</th>
<th>( n/w_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1398101</td>
<td>(10101101101010101)</td>
<td>1</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>23</td>
<td>182361</td>
<td>(1011110001011101)</td>
<td>89</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>89</td>
<td>47127</td>
<td>(1010110000010111)</td>
<td>23</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>683</td>
<td>6141</td>
<td>(10111111111101)</td>
<td>6141</td>
<td>22</td>
<td>1</td>
</tr>
</tbody>
</table>

Hence we only have to find a 2-addition chain for \( S = \{1, 23, 89, 6141\} \) and addition chains for 11 and 2.

(i) A 2-addition chain \( \gamma \) for \( S \) can be generated with Brauer addition chains (Theorem 37) for each element of \( S \), using \( r = 4 \), and then merging them.

The choice for the parameter \( r \) is usually determined by the largest element of \( S \).
This 2-addition chain contains $A(\gamma) = 9$ additions, $Q(\gamma) = 12$ doublings (written in italics) and a total length of $L(\gamma) = 21$.

For various values of $r$, we find the following cost:

<table>
<thead>
<tr>
<th>$r$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>11</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>$Q$</td>
<td>14</td>
<td>12</td>
<td>14</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

(ii) An addition chain $δ_1$ for 11 of length 5 is given by the left graph, and the addition chain $δ_2$ for 2 has length 1 (right graph):

Addition chain $δ_1$ for 11 and $δ_2$ for 2

(iii) In the final step we combine the chains. We only write $2^j$ to mark that $j$ many doublings are left out, and $W_i(j) = ((2^{w_i})^j - 1)/(2^{w_i} - 1)$ for short.

Thus, we can test an element in $\mathbb{F}_{2^{22}}$ for primitivity using $5 + 2 \cdot 1 + 1 + 9 = 17$ multiplications and $10 \cdot 2 + 1 \cdot 11 + 1 \cdot 11 + 0 + 12 = 54$ squarings in $\mathbb{F}_{2^{22}}$.

If we compute separate addition chains for $\{(2^{22} - 1)/p; p \in \mathcal{P}\} = \{e_1, e_2, e_3, e_4\}$ directly (Theorem 3) and merge them, we get the following chain $η$ (doublings are printed in italics again).
precomputed elements | common elements | other elements
---|---|---
1, 2, 4, 5 | 10, 20, 40, 44, 88 | 176, 352, 356, 712, 1424, 2848, 2849, 5698, 11396, 22792, 22795, 45590, 91180, 182360, 182361
1, 2, 4 | 8, 11, 22, 44, 88 | 92, 184, 368, 736, 1472, 2944, 5888, 5890, 11780, 23560, 47120, 47127
1, 2, 4, 5 | 10, 20, 40 | 42, 84, 168, 336, 341, 682, 1364, 2728, 2730, 5460, 10920, 21840, 21845, 43690, 87380, 174760, 174762, 349524, 699048, 1398096, 1398101
1, 2, 3, 4, 7 | 8, 11, 22, 44, 88 | 95, 190, 380, 760, 767, 1534, 3068, 6136, 6141

The resulting addition chain contains $A(\eta) = 20$ additions and $Q(\eta) = 50$ doublings. Here our special addition chain reduces the number of (expensive) nondoublings by 15%. On the other hand, the number of (cheap) doublings is expanded by 8%. We note that our chain in (iii) is not a handcrafted optimization, but it is obtained by the concatenation of systematic procedures.

**Experiments.** We report on our computation of the cost for various primitivity tests in $\mathbb{F}_{2^n}$ for some values of $n$ with $2 \leq n \leq 948$. For 848 of these values the set $\mathcal{P}_n$ of all prime factors of $2^n - 1$ is known; for 99 values the factorization is not complete. These factorizations can be found in the Cunningham tables. We counted the number of squarings ($Q$) and of multiplications ($A$) in $\mathbb{F}_{2^n}$. Table I gives the results for $725 \leq n \leq 750$; these are reasonably representative. The number of prime factors is $d = #\mathcal{P}_n$. We proceeded as illustrated in Example 33 and compared our approach with *general addition chains* that ignore the special structure of the exponents. Namely, we first created both binary and Brauer addition chains (Theorem 6) for each $(2^n - 1)/p$ for $p \in \mathcal{P}_n$, and we merged them (see columns 3 to 6 of Table I) labelled *general addition chains*.

For the second set of results we applied our approach as described by Algorithm 27. We separated each exponent $e = (2^n - 1)/p$ into a regular part $e/s$ and a repeated part $s$ as described in Algorithm 24. We applied Algorithm 10 to the regular part $e/s$ to profit from the regular structure of the exponents. As illustrated in Example 33 we additionally have to create an addition chain for the set $\mathcal{S} = \{(2^{ord_p(n)} - 1)/p; p \in \mathcal{P}\}$. For each element of $\mathcal{S}$ we computed the binary addition chain and Brauer’s addition chain; see Theorem 6. For both algorithms we merged the single chains to create a chain for $\mathcal{S}$. The labels *binary* and *Brauer* in columns 7–10 of Table I indicate which addition chain has been used to generate $\mathcal{S}$. If $2^n - 1$ is a Mersenne prime, no computation is necessary because every element of $\mathbb{F}_{2^n}$ except 1 is primitive. If the factorization for the integer $2^n - 1$ is not known—this is the case for $n = 727$ which is marked by “*” in the corresponding row in Table I—then no computation is done either. In the last column, $u$ is the quotient of the number of multiplications for Brauer’s addition chain without and with Algorithm 27 (columns 5 and 9 in Table I). Thus $u = 865/350 \approx 2.5$ in the first row. In a representation of $\mathbb{F}_{2^n}$ where squarings are essentially free, $u$ represents the improvement of special over general addition chains.
Table 1. The number of multiplications \((A)\) and squarings \((Q)\) for primitivity testing in \(F_{2n}\) using different addition chains in the range between 725 and 750. Here \(d = \omega(2^n - 1)\) is the number of different prime divisors of \(2^n - 1\), and \(u\) is the quotient of the number of multiplications in columns 5 and 9. The factorization of \(2^{727} - 1\) is not complete yet; thus no computation is done.

| \(n\) | \(d\) | \begin{tabular}{l} General addition chains \\ \begin{tabular}{llll}
Binary & Brauer & Binary & Brauer
\end{tabular}
\end{tabular} | \begin{tabular}{l} With Algorithm 27 \\ \begin{tabular}{llll}
Binary & Brauer & Binary & Brauer
\end{tabular}
\end{tabular} | \(u\) |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>725</td>
<td>10</td>
<td>2424 6493 865 6521</td>
<td>920 6497 350 6528</td>
<td>2.5</td>
</tr>
<tr>
<td>726</td>
<td>19</td>
<td>5947 12980 1724 12978</td>
<td>2173 12989 665 12995</td>
<td>2.6</td>
</tr>
<tr>
<td>727</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>728</td>
<td>28</td>
<td>8911 18799 2515 18788</td>
<td>2146 19189 676 19186</td>
<td>3.7</td>
</tr>
<tr>
<td>729</td>
<td>14</td>
<td>4170 9422 1280 9431</td>
<td>1843 9428 585 9447</td>
<td>2.2</td>
</tr>
<tr>
<td>730</td>
<td>15</td>
<td>4572 10154 1370 10180</td>
<td>1916 10161 597 10185</td>
<td>2.3</td>
</tr>
<tr>
<td>731</td>
<td>6</td>
<td>1197 3639 463 3676</td>
<td>441 3643 205 3684</td>
<td>2.3</td>
</tr>
<tr>
<td>732</td>
<td>20</td>
<td>6596 13816 1820 13799</td>
<td>2936 13834 859 13850</td>
<td>2.1</td>
</tr>
<tr>
<td>733</td>
<td>2</td>
<td>367 731 165 783</td>
<td>367 731 165 783</td>
<td>1.0</td>
</tr>
<tr>
<td>734</td>
<td>12</td>
<td>4039 8031 1159 8050</td>
<td>2944 8037 857 8058</td>
<td>1.4</td>
</tr>
<tr>
<td>735</td>
<td>20</td>
<td>5302 13886 1757 13891</td>
<td>2461 13903 817 13912</td>
<td>2.2</td>
</tr>
<tr>
<td>736</td>
<td>19</td>
<td>6651 13160 1747 13174</td>
<td>2021 13172 593 13191</td>
<td>2.9</td>
</tr>
<tr>
<td>737</td>
<td>9</td>
<td>2391 5859 821 5887</td>
<td>1559 5863 524 5895</td>
<td>1.6</td>
</tr>
<tr>
<td>738</td>
<td>18</td>
<td>6052 13204 1740 13213</td>
<td>2369 13219 711 13235</td>
<td>2.4</td>
</tr>
<tr>
<td>739</td>
<td>2</td>
<td>343 737 166 789</td>
<td>343 737 166 789</td>
<td>1.0</td>
</tr>
<tr>
<td>740</td>
<td>24</td>
<td>7912 16901 2291 16899</td>
<td>3302 16915 996 16917</td>
<td>2.3</td>
</tr>
<tr>
<td>741</td>
<td>14</td>
<td>3604 9562 1197 9580</td>
<td>1468 9585 497 9614</td>
<td>2.4</td>
</tr>
<tr>
<td>742</td>
<td>17</td>
<td>5303 11799 1613 11808</td>
<td>2440 11810 749 11819</td>
<td>2.2</td>
</tr>
<tr>
<td>743</td>
<td>6</td>
<td>1805 3700 588 3737</td>
<td>1805 3700 588 3737</td>
<td>1.0</td>
</tr>
<tr>
<td>744</td>
<td>25</td>
<td>8646 17727 2317 17724</td>
<td>2980 17764 868 17758</td>
<td>2.7</td>
</tr>
<tr>
<td>745</td>
<td>4</td>
<td>688 2227 302 2272</td>
<td>239 2230 89 2236</td>
<td>3.4</td>
</tr>
<tr>
<td>746</td>
<td>6</td>
<td>1881 3711 539 3751</td>
<td>1325 3717 391 3761</td>
<td>1.4</td>
</tr>
<tr>
<td>747</td>
<td>8</td>
<td>2013 5196 682 5237</td>
<td>839 5202 292 5249</td>
<td>2.3</td>
</tr>
<tr>
<td>748</td>
<td>22</td>
<td>7346 15596 2063 15608</td>
<td>3132 15616 928 15633</td>
<td>2.2</td>
</tr>
<tr>
<td>749</td>
<td>6</td>
<td>1303 3727 507 3769</td>
<td>1204 3737 408 3782</td>
<td>1.2</td>
</tr>
<tr>
<td>750</td>
<td>22</td>
<td>6910 15643 2066 15646</td>
<td>2035 15660 646 15666</td>
<td>3.2</td>
</tr>
</tbody>
</table>

The average timings in Table 2 give a statistical précis of our experiments. We have divided the values of \(n\) into groups of about 50 each. The values given are the arithmetic mean over all factored values of the interval (column 1).

These averages show a somewhat superlinear increase with the field degree \(n\), but the close-up look of Table 1 reveals a rather large variation, correlated with the number \(d\) of prime factors of \(2^n - 1\). Figure 3 describes the gain factor (called \(u\) in Table 1) of our method over general chains in dependence on \(d\). We observe a tendency towards higher improvement rates as \(d\) increases. The average gain in our method is large when there are many prime factors \(p\) of \(2^n - 1\) with small \(\text{ord}_p(2)\); this usually corresponds to small \(p\) and to large \(d = \omega(2^n - 1)\).
8. Polynomial factorization

In many algorithms for factoring a polynomial \( f \in \mathbb{F}_q[x] \), exponentiation modulo \( f \) accounts for the bulk of the computing time. We now apply our addition chain technology to three particular subproblems: equal-degree factorization, trace computation, and irreducibility testing. There does not seem to be any fancy data structure like normal bases available, and so we will only gain a constant factor in the cost. See [27] for a survey and Chapter 14 of [20] for the algorithms.

In _equal-degree factorization_, we know that \( f \) is a product of distinct irreducible factors of degree \( d \). In the algorithm of [11] for odd \( q \), the most costly part is computing a \((q^d - 1)/2\)th power of a polynomial modulo \( f \). The binary addition chain takes at most \( 2d \log_q q \) multiplications modulo \( f \). Brauer’s method turns the factor 2 into \( 1 + o(1) \), and Algorithm [27] yields the same cost, possibly with a simpler algorithm.

**Corollary 34.** Applying Algorithm [27], we can compute a \( q \)-addition chain \( \gamma \) for \((q^d - 1)/(q - 1)\) with

\[
A(\gamma) \leq c_2(d) \text{ additions}
\]

and

\[
Q(\gamma) \leq d - 1 \text{ many } q\text{-steps}
\]

or a (classical) addition chain of total length at most \( d \log_q q \cdot (1 + o(1)) \).
Primitivity testing using general vs. special addition chains

Figure 3. The gain $u$ in our method in dependence on the number of prime divisors of $2^n - 1$, for $n \leq 948$. The graphic shows the minimal, average, and maximal improvement.

Proof. With input $n = d, q = w = 1$, we have $s = 1$ in Theorem 28 and a $q$-addition chain $\gamma$ for $(q^d - 1)/(q - 1)$ with $A(\gamma) \leq \ell_2^* (d) \leq \log_2 d \cdot (1 + o(1))$ and $Q(\gamma) \leq d - 1$. To turn this into a (classical) addition chain, we expand each $q$-step in $\gamma$ into $\log_2 q \cdot (1 + o(1))$ additions.

A faster algorithm was introduced in [29], reducing the time from $O^~(n^2 \log q)$ to $O^~(n^2 + n \log q)$ operations in $F_q$, where we use $d \leq n$, and the “soft Oh” $O^~$ hides factors $\log n$. It is based on the polynomial representation of the Frobenius. We write $\xi = x \mod f \in R = F_q[x]/(f)$, and for any $a = \sum_{0 \leq i < n} a_i \xi^i \in R$ with all $a_i \in F_q$, we let $\tilde{a} = \sum_{0 \leq i < n} a_i x^i \in F_q[x]$ be the canonical representative of $a$. The crucial property is that for any positive integer $m$

$$\tilde{a}(\xi^m) = \sum_{0 \leq i < m} a_i \xi^{qm} = (\sum_{0 \leq i < n} a_i \xi^i)^m = a^m. \quad \text{(35)}$$

We have the following adaptation of Algorithm 5.2 from [29] for computing trace maps.

Algorithm 36 (Trace map via addition chain).

Input: $f, a, b, m$ and $\gamma$, where $f \in F_q[x]$ has degree $n$, $a$ and $b$ are elements of $R$ with $b = \xi^t$ for some power $t$ of $q$, and $\gamma = ((j(1), k(1)), \ldots, (j(l), k(l)))$ is an addition chain of length $l$ for the positive integer $m$.

Output: The elements $a^m$ and $\sum_{1 \leq u \leq m} a_i^m$ in $R$.

1. Compute $\tau_0 \leftarrow \tilde{a}(b), \mu_0 \leftarrow b$. 

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
For \( i = 1, \ldots, l \) do 3–4.

3. \( \tau_i \leftarrow \tau_j(i) + \bar{\tau}_{k(i)}(\mu_j(i)) \).

4. \( \mu_i \leftarrow \bar{\mu}_{k(i)}(\mu_j(i)) \).

5. Return \( \mu_l \) and \( \tau_l \).

**Theorem 37.** Algorithm 37 works correctly as specified and uses \( O(l\ln M(n)) \) operations in \( \mathbb{F}_q \).

**Proof.** Let \( S(\gamma) = \{c_0, \ldots, c_l\} \) be the semantics of \( \gamma \), with \( c_i < c_{i+1} \) for all \( i \), as usual. We prove by induction on \( i \) that

\[
\tau_i = \sum_{1 \leq u \leq c_i} a^u, \quad \mu_i = \xi^{c_i},
\]

for \( 0 \leq i \leq l \). Since \( c_l = m \), correctness then follows. Applying (35) with \( q^m = t \), we have \( \tau_0 = a^l \), and the claim follows for \( i = 0 \). For \( i \geq 1 \), we have

\[
\tau_i = \tau_j(i) + \bar{\tau}_{k(i)}(\mu_j(i)) = \tau_j(i) + (\tau_{k(i)})^{\mu_j(i)}
\]

\[
= \tau_j(i) + \left( \sum_{1 \leq u \leq c_{k(i)}} a^u \right)^{\mu_j(i)} = \tau_j(i) + \sum_{1 \leq u \leq c_{k(i)}} a^{\mu_j(i) + u}
\]

\[
= \sum_{1 \leq u \leq c_j(i)} a^u + \sum_{c_j(i) < u \leq c_j(i) + c_{k(i)}} a^u = \sum_{1 \leq u \leq c_i} a^u,
\]

since \( c_i = c_j(i) + c_{k(i)} \). Similarly,

\[
\mu_i = \bar{\mu}_{k(i)}(\mu_j(i)) = (\mu_{k(i)})^{\mu_j(i)} = (\xi^{c_{k(i)}})^{\mu_j(i)} = \xi^{c_{j(i)} + c_{k(i)}} = \xi^{c_i},
\]

The cost of the algorithm is \( l \) additions and \( 2l \) modular compositions. The cost of the latter is discussed in Fact 5.1 of [29]; this gives our estimate. \( \square \)

Even better bounds are given in the cited paper, based on fast matrix multiplication. Of course, our algorithm gives no asymptotic improvement, but at best the factor of at most 2 corresponding to the length ratio between the binary addition chain (which, when used for \( \gamma \), essentially gives the older algorithm) and shorter chains. Also, the presentation of our algorithm is somewhat simpler.

A further application of our methodology is to Rabin’s [11] irreducibility test. The bottleneck there is to compute \( x^q^t \) modulo \( f \) for \( t = 1 \) and each prime divisor \( t \) of \( n \). We can now take an addition chain \( \gamma \) for this set of exponents (Section 6) and run Algorithm 37 using \( \gamma \) as part of the input.

**9. Conclusion**

We have presented addition chains for \( e \in \mathbb{N}_{\geq 1} \) that benefit from a given regularity of the \( q \)-ary representation of \( e \). A basic tool is the generalization of addition chains to \( q \)-addition chains. For several applications of addition chains we have to take into account the properties of different representations of finite fields, which lead to different cost measures for \( q \)-steps and additions in our \( q \)-addition chains. We have applied these ideas for addition chains to five computational problems in finite fields: inversion, primitivity testing, and three tasks connected to polynomial factorization.
References

AAECC-12, Toulouse, France, Teo Mora & Harold Mattson, editors, number 1255 in Lecture Notes in Computer Science, 88–113. Springer-Verlag, ISSN 0302-9743. MR 99c:08125


47. A. Schönhage (1975) A lower bound for the length of addition chains. Theoretical Computer Science 1, 1–12. MR 57:18220


Fakultät für Elektrotechnik, Informatik, Mathematik, Universität Paderborn, D-33095 Paderborn, Germany

*E-mail address*: gathen@upb.de

Bückeburger Str. 12, D-59174 Kamen, Germany

*E-mail address*: noecker@upb.de