Joachim von zur Gathen & Michael Nöcker (1997). Exponentiation in Finite Fields : Theory and Practice. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes : AAECC-12*, Toulouse, France, Teo Mora & Harold Mattson, editors, number 1255 in Lecture Notes in Computer Science, 88–113. Springer-Verlag. ISSN 0302-9743. URL http://dx.doi.org/10.1007/3-540-63163-1_8.

Proc. 12th Symposium Applied Algebra, Algebraic Algorithms and
Error-Correcting Codes, AAECC-12
Toulouse, France, 1997

Springer lecture notes in computer science **1255**, pp. 88-133

Exponentiation in finite fields: theory and practice

Joachim von zur Gathen and Michael Nöcker

Fachbereich 17 Mathematik-Informatik
Universität-GH Paderborn
D-33095 Paderborn, Germany
{gathen,noecker}@uni-paderborn.de

1 Introduction

Several cryptographical methods use exponentiation as their basic operation: e.g., the Diffie–Hellman method for key–exchange (Diffie & Hellman 1976), ElGamal's algorithm for digital signature (ElGamal 1985) or the RSA–scheme of Rivest *et al.* (1978). In some of these public key cryptosystems, one uses large exponents in finite fields for securely encoded transmission. Therefore it is of interest to develop fast exponentiation methods, and as we will see in the sequel, also fast multiplication algorithms.

The goal of this article is twofold: first, to present and analyze in a unified framework five addition chain algorithms from the literature, plus a new one. This allows their theoretical comparison in Section 2. The second goal is to achieve a similarly clear comparison in Section 8 of three known ways of using addition chains for exponentiation in finite fields which are presented in Sections 6 through 8. We also compare these theoretical results to experiments; this is reported in Chapters 4 and 9.

At the core of any exponentiation algorithm lies a method for multiplying two elements. Our basic result is that the best exponentiation method is one that combines fast addition chains with fast multiplication algorithms. For these issues, we study the polynomial and normal basis representations of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.

2 Algorithms on addition chains

2.1 Addition chains and its generalizations

Although exponentiation deals with multiplication, the problem can be easily reduced to addition, since the exponents are additive. Therefore, we first concentrate on addition chains. Much of this material can be found in Knuth (1981), 4.6.3. We recall the following:

Notation 1. Given integers $m \in \mathbb{N}$ and $q \geq 2$, the *q-ary representation* of m is $(m_{\lambda-1}, \ldots, m_0)$, with $\sum_{0 \leq i < \lambda} m_i q^i = m$, $\lambda = \lfloor \log_q m \rfloor + 1$ and $m_0, \ldots, m_{\lambda-1} \in \{0, \ldots, q-1\}$. We write $(m_{\lambda-1}, \ldots, m_0) = (m)_q$. The *Hamming weight* of $(m)_q$ is defined as $\nu_q(m) = \#\{i : 0 \leq i < \lambda, m_i \neq 0\}$.

Definition 2. An addition chain for $m \in \mathbb{N}$ is a sequence of pairs $(j_1, k_1), \ldots, (j_L, k_L)$ with $0 \leq j_i, k_i < i$ for all $1 \leq i \leq L$ and if we define $a_0, \ldots, a_L \in \mathbb{N}$ by $a_0 = 1$ and $a_i = a_{j_i} + a_{k_i}$, for $1 \leq i \leq L$, then $a_L = m$. The length of the addition chain is the integer L. The smallest L for which there exists an addition chain of length L for m is denoted by $l(m)$.

It is common in the literature to only consider the semantics of an addition chain by identifing the sequence of integers $a_0, \ldots, a_L$ with the addition chain. We often concentrate on the semantics to avoid technical details when the syntax is clear.

Fact 3. (Schönhage 1975) *Let $m \in \mathbb{N}$ and $\nu_2(m)$ its binary Hamming weight. Then $l(m) \geq \log_2 m + \log_2 \nu_2(m) - 2.13$.*

By definition we have, for $1 \leq i \leq L$, $a_i = a_{j_i} + a_{k_i}$ for some $0 \leq j_i, k_i < i$. If $j_i = k_i \leq i-1$ then we call step i a *doubling step*. If $j_i \neq k_i$ and $j_i = i-1$ or $k_i = i-1$, then step i is a *star step*.

Fact 4. (Downey *et al.* 1981) *Let m and k be positive integers. The problem whether there exists an addition chain for m with length $L \leq k$ is NP–complete.*

Therefore, it would not be a promising approach to try and calculate an addition chain of shortest length; rather we look for one with reasonably short length. For our algorithmic purposes it is useful to generalize the notion of addition chains in the following way (von zur Gathen 1991):

Definition 5. Let $q, m \in \mathbb{N}$. A *q-addition chain* for m is a sequence of pairs $(j_1, k_1), \ldots, (j_L, k_L)$ with $0 \leq k_i < i$ and $j_i = -q$ or $0 \leq j_i < i$ for all $1 \leq i \leq L$ and, if we set $a_0 = 1$ and $a_i = a_{j_i} + a_{k_i}$ (if $j_i \neq -q$) or $a_i = q \cdot a_{k_i}$ (if $j_i = -q$) for all i, then $a_L = m$. If $j_i = -q$ we call step i a *q-step*.

We denote the number of doublings by D, the number of q-steps by Q and the number of remaining addition steps by A. Then we have $L = D + Q + A$ for a q-addition chain of length L. Every q-addition chain can be rewritten as an addition chain by expanding $a_i = q \cdot a_{k_i}$ using at most $\lfloor \log_2 q \rfloor$ doublings and at most $\lfloor \log_2 q \rfloor$ star steps. An addition chain is just a 2-addition chain.

2.2 Word chains

An alphabet $\mathcal{A}$ is a finite set; if $q = \#\mathcal{A}$, it is a q-letter alphabet. We may assume without loss of generality that $\mathcal{A} = \{0, \ldots, q-1\}$. We define $w_{i-(q-1)} = i$ for $0 \leq i \leq q-1$. Following standard terminology, as e.g. in Lothaire (1983), a *word* over $\mathcal{A}$ is a finite sequence of elements of $\mathcal{A}$, and $\mathcal{A}^*$ is *set of all words* over $\mathcal{A}$. Concatenation $\circ$ makes $\mathcal{A}^*$ into a monoid. We write $\varepsilon \in \mathcal{A}^*$ for the *empty word*. A word $v \in \mathcal{A}^*$ is a *left factor* of $w \in \mathcal{A}^*$ if there exists a word $z \in \mathcal{A}^*$ such that $w = v \circ z$. This gives an order on $\mathcal{A}^*$ which will be denoted by $v \leq w$. Let $v, v' \in \mathcal{A}^*$ be left factors of w. Then we have either $v \leq v'$ or $v' \leq v$.

Definition 6. A *word chain* for a word $w \in \mathcal{A}^*$ over a q-letter alphabet $\mathcal{A}$ is a sequence of pairs $(j_1, k_1), \ldots, (j_L, k_L)$ with $1 - q \leq j_i, k_i < i$ for all $1 \leq i \leq L$ and, if $\{w_{1-q}, \ldots, w_0\} = \mathcal{A}$ and $w_i = w_{j_i} \circ w_{k_i}$ for all i, then $w_L = w$. The length of the word chain is the integer L. The shortest length L for which there exists a word chain for w is denoted by $l_{\mathcal{A}}(w)$.

We concentrate on the semantics when the syntax is clear. Addition chains correspond bijectively to word chains over a one–letter alphabet, and therefore word chains are a generalization of addition chains. Word chains provide a short notation for shifts and concatenations of the q-ary representations. We can simulate a word chain $w_{1-q}, \ldots, w_0, w_1, \ldots, w_L$ over the q-letter alphabet $\mathcal{A} = \{0, \ldots, q-1\}$ by q-addition chains for integers represented in q-ary notation by words from $\mathcal{A}^*$: Set $a_0 = w_{1-q+1} = 1, \ldots, a_{q-2} = w_0 = q-1$. For $w_{j_i} = (a)_q$ and $w_{k_i} = (b)_q$ we have $w_i = w_{j_i} \circ w_{k_i} = (a \cdot q^{\# w_{k_i}} + b)_q$. Therefore step i of a word chain can be simulated by a q-addition chain using $\# w_{k_i}$ many q-steps plus one star step.

Proposition 7. *A word chain over the q-letter alphabet $\mathcal{A} = \{0, \ldots, q-1\}$ of length L can be simulated by a q-addition chain of length $L' = A' + Q' \leq L + q - 2 + 2^L - 1$ using $A' \leq L + q - 2$ star steps and $Q' \leq \sum_{1 \leq i \leq L} \# w_{k_i} \leq 2^L - 1$ q-steps.*

2.3 A survey on algorithms

We concentrate on algorithms for word chains to avoid shift operations in the sequel. Because of the results given above we easily can transfer them to addition chain algorithms. Most of the algorithms to create word chains for w given in the literature can be described in two steps: in the first step a set $\mathcal{D}$ of words is chosen for each of which a word chain is created. The second step uses this set and expands the corresponding word chain to a word chain for w.

Definition 8. Let $w \in \mathcal{A}^*$ and $\mathcal{D} \subset \mathcal{A}^*$. Then $v \in \mathcal{D}$ is called *a maximal left factor* of w in $\mathcal{D}$ if $v \leq w$ and $z \leq v$ for all $z \in \mathcal{D}$ with $z \leq w$.

A general algorithm can be described as follows:

Algorithm 9 `word chain`**.** Input: $w \in \mathcal{A}^*$. Output: A word chain $\mathcal{W}$ for w.

A. Determine a set $\mathcal{A} \subseteq \mathcal{D} \subset \mathcal{A}^*$ and a word chain $\mathcal{W}'$ for $\mathcal{D}$.

B. Compute a word chain $\mathcal{W}$ with prefix $\mathcal{W}'$ for w as follows:

   1. Let $x = \varepsilon$ and $\mathcal{W} = \mathcal{W}'$.
   2. While $(w \neq \varepsilon)$ do repeat
      3. Let $v \in \mathcal{D}$ be the maximal left factor of w in $\mathcal{D}$ with $w = v \circ z$.
      4. Append $x \circ v$ to $\mathcal{W}$.
      5. Set $w \leftarrow z$ and $x \leftarrow x \circ v$.

C. Return $\mathcal{W}$.

We describe five algorithms to compute word chains: Brauer (1939), Yacobi (1991), Bocharova & Kudryashov (1995), a new one, and Brickell *et al.* (1993). The well-known `binary` method (see e.g. Knuth 1981) is just a special case of Brauer's algorithm. Because of Algorithm 9 it is sufficient to describe how $\mathcal{D}$ and $\mathcal{W}'$ are determined.

Brauer. For the algorithm of Brauer (1939) Step A is as follows.

A. Determine a set $\mathcal{A} \subseteq \mathcal{D} \subset \mathcal{A}^*$ and a word chain $\mathcal{W}'$ for $\mathcal{D}$ according to `brauer`*:*

   1. Choose a parameter $r \in \mathbb{N}$.
   2. Set $\mathcal{D} = \{w \in \mathcal{A}^* : \#w = r\}$. The word chain $\mathcal{W}' = (1-q, 1-q), \ldots, (1-q, 0), (1-q+1, 1-q), \ldots, (q^r - q, 0)$ is a word chain for $\mathcal{D}$.

Brauer's algorithm is referred to as the q-ary or q^r-ary method. Brauer described the algorithm for addition chains.

Lemma 10. *Let $\mathcal{A}$ be a q-letter alphabet and $r \in \mathbb{N}$. Using Algorithm* `brauer` *a word chain for $w \in \mathcal{A}$ can be computed with at most $q^r - q + \lceil \frac{\omega}{r} \rceil$ concatenations, where $\omega = \#w$.*

Corollary 11. *An addition chain algorithm according to* `brauer` *generates a 2^r-addition chain for m with $A = \nu_{2^r}(m) + 2^r - 3$ star steps and $Q = \lfloor \log_{2^r} m \rfloor$ 2^r-steps.*

Corollary 12. (Brauer 1939) *Let $l(m)$ be the shortest length of addition chains for m, and $\mu = \log_2 m$. Then*

$$l(m) \leq \mu(1 + \frac{2}{\log_2 \mu} + \frac{2}{\sqrt{\mu}}) \leq \mu + 2\frac{\mu}{\log_2 \mu}(1 + o(1)).$$

Corollary 13. *The* `binary` *method generates addition chains for $m \in \mathbb{N}$ of length $\lfloor \log_2 m \rfloor + \nu_2(m) - 1$.*

Yacobi. The determination of $\mathcal{D}$ according to `brauer` has two properties: The elements are determined without considering the structure of word w and all elements have the same length. Yacobi (1991) does not impose these restrictions, and uses the data compression algorithm of Ziv & Lempel (1978) to determine $\mathcal{D}$. The letter $0 \in \mathcal{A}$ plays a special role.

A. Determine a set $\mathcal{A} \subseteq \mathcal{D} \subset \mathcal{A}^*$ and a word chain $\mathcal{W}'$ for $\mathcal{D}$ according to `yacobi`*:*
  1. *Set $\mathcal{D} = \mathcal{A}$ and $\mathcal{W}' = \emptyset$. Set $w' = w$.*
  2. *while $(w' \neq \varepsilon)$ do repeat*
     3. *Let $v \in \mathcal{D}$ be the maximal left factor of w' in $\mathcal{D}$ with $w' = v \circ z'$.*
     4. *if $(v =$ '0'$)$ then set $w' \leftarrow z'$*
     5. *else let $x \in \mathcal{D}$ be the maximal left factor of z' in $\mathcal{A}$ with $z' = x \circ z$. Add $v \circ x$ to $\mathcal{D}$ and $\mathcal{W}'$. Set $w' \leftarrow z$.*

For $q = 2$, Yacobi obtains the following results on uniformly chosen random input words w of length ω.

Lemma 14. *Let $\mathcal{A}$ be the 2-letter alphabet $\mathcal{A} = \{0, 1\}$. On the average Algorithm* `yacobi` *computes a word chain for $w \in \mathcal{A}^*$ with $3\frac{\omega}{\log_2 \omega}(1+o(1))$ concatenations, where $\omega = \#w$.*

Corollary 15. (Yacobi 1991) *Let $\mu \in \mathbb{N}$. Then* `yacobi` *yields an addition chains with $D_{ave} = \lfloor \mu \rfloor + \frac{\mu}{\log_2 \mu}(1 + o(1))$ doublings and $A_{ave} = \frac{3}{2}\frac{\mu}{\log_2 \mu}(1 + o(1))$ star steps on the average for a randomly chosen $m \in \mathbb{N}$ with $\lfloor \log_2 m \rfloor = \mu$.*

Bocharova. The algorithm given by Bocharova & Kudryashov (1995) repeats Step A r times where $r \in \mathbb{N}$ is a selectable parameter. New words are added to $\mathcal{D}$ after each loop using an idea of Tunstall (1968).

A. Determine a set $\mathcal{A} \subseteq \mathcal{D} \subset \mathcal{A}^*$ and a word chain $\mathcal{W}'$ for $\mathcal{D}$ according to `bocharova`*:*
  1. *Set $\mathcal{D} = \mathcal{A}$ and $\mathcal{W}' = \emptyset$. Determine a parameter $r \in \mathbb{N}$.*
  2. *Repeat $r - 1$ times*
     3. *Let $w = v_1 \circ \cdots \circ v_k$ with $v_i \in \mathcal{D}$ the maximal left factor of $v_i \circ \cdots \circ v_k$ in $\mathcal{D}$ for $1 \leq i \leq k$.*
     4. *Let $v \in \mathcal{D} - \{$'0'$\}$ be a word appearing most often in $v_1, \ldots, v_k$. Add $v \circ w_j$ to $\mathcal{D}$ and append it to $\mathcal{W}'$ for all $w_j \in \mathcal{A}$.*

Lemma 16. *Let $\mathcal{A} = \{0, 1\}$ and $r \in \mathbb{N}$. On the average Algorithm* `bocharova` *computes a word chain for $w \in \mathcal{A}^*$ with $A = 2(\frac{\omega}{1+\log_2 \omega} + r)$ concatenations, where $\omega = \#w$.*

Corollary 17. *Let $\mu \in \mathbb{N}$. Algorithm* `bocharova` *computes an addition chain with $D_{ave} < \lfloor \mu \rfloor + \frac{\mu}{(\log_2 \mu)^2}$ doubling steps and*

$$A_{ave} = \frac{\mu}{\log_2 \mu}(1 + \frac{\log_2 \log_2 \mu}{\log_2 \mu - 2\log_2 \log_2 \mu} + \frac{1}{\log_2 \mu}) = \frac{\mu}{\log_2 \mu}(1 + o(1))$$

star steps on the average for a randomly chosen $m \in \mathbb{N}$ with $\lfloor \log_2 m \rfloor = \mu$.

2.4 A new algorithm based on data compression

Both algorithms `yacobi` and `bocharova` concatenate a word already in $\mathcal{D}$ with elements of $\mathcal{A}$ to generate new words for $\mathcal{D}$. Our new algorithm allows also to concatenate words of $\mathcal{D}$ with each other to compute longer words in Step A if useful. Just as the two last algorithms, it does not fix the length of the words in $\mathcal{D}$, and $\mathcal{D}$ depends on w and uses ideas similar to data compression techniques. When our algorithm is transferred to addition chains, it tries to reduce the number of star steps by using more doublings. The basic ideas of the algorithm are:

- Create $\mathcal{D}$ by splitting the given word similarly as in `yacobi` but adding words of $\mathcal{D} \times \mathcal{D}$ to $\mathcal{D}$. This is realized by storing the concatenation of the last left factor found with its predecessor.
- Divide Step A of Algorithm 9 into two main substeps: first create a set of words that is possibly used in Step B (*set of candidates*). Then reduce this set to such words that are really used (*set of used words*).

The last idea can also be used within the other algorithms invented so far. The idea is especially helpful when words of a special type or with repeating sequences of letters have been given. Practical tests of our algorithm show that splitting Step A reduces the number of elements of $\mathcal{D}$ up to 50%. The corresponding word chain $\mathcal{W}'$ can be reduced by 25%.

A. Determine a set $\mathcal{A} \subseteq \mathcal{D} \subset \mathcal{A}^*$ and a word chain $\mathcal{W}'$ for $\mathcal{D}$ according to `lookback`*:*
- *A1. Find a set of candidates:*
  - *1. Set $w' \leftarrow w$. Set $\mathcal{D}' = \mathcal{A}$ and $\mathcal{W}' = \emptyset$.*
  - *2. Let y be the maximal left factor of w' in $\mathcal{D}'$ with $w' = y \circ z$. Set $w' \leftarrow z$.*
  - *3. While $(w' \neq \varepsilon)$ do repeat*
    - *4. Let v be the maximal left factor of w' in $\mathcal{D}'$ with $w' = v \circ z$. Append $y \circ v$ to $\mathcal{W}'$.*
    - *5. If $(v = \text{‘0‘})$ then set $y \leftarrow y \circ v$*
    - *6. else append $y \circ v$ to $\mathcal{D}'$ and set $y \leftarrow v$.*
    - *7. Set $w' \leftarrow z$.*
- *A2. Find the set of used words:*
  - *8. Set $w' \leftarrow w$ and $\mathcal{D} = \mathcal{A}$.*
  - *9. While $(w' \neq \varepsilon)$ do repeat*
    - *10. Let v be a maximal left factor of w' in $\mathcal{D}'$ with $w' = v \circ z$. Append v to $\mathcal{D}$.*
  - *11. Find a prefix of $\mathcal{W}'$ (also denoted by $\mathcal{W}'$) which is a word chain for $\mathcal{D}$.*

Lemma 18. *Algorithm* `lookback` *computes a word chain for $w \in \{0,1\}^*$ with at most $\#w - 1$ concatenations of words.*

Corollary 19. *Algorithm* `lookback` *computes a addition chain for $m \in \mathbb{N}$ with $A \leq \nu_2(m)$ star steps and $D \leq \frac{1}{2}\mu(\mu-1)$ doublings where $\mu = \lfloor \log_2 m \rfloor$.*

These upper bounds are not very good. In practice `lookback` seems to work better than these bounds predict. We did not analyze the average case in detail so far. In our experiments, on the average Algorithm `lookback` computed addition chains with $A \approx 1.7\frac{\mu}{\log_2 \mu}$ star steps and $D \approx \mu + 2.2\frac{\mu}{\log_2 \mu}$ doublings when $m \in \mathbb{N}$ with $\mu = \lfloor \log_2 m \rfloor \in \{1024, 2048, 4096, 8192\}$ was tested. 1000 numbers of each length were tested. If the experiments can be extrapolated, then it seems that Algorithm `lookback` computes on average an addition chain for $m \in \mathbb{N}$ with $O(\frac{\mu}{\log_2 \mu})$ star steps and $\mu + O(\frac{\mu}{\log_2 \mu})$ doublings, where $\mu = \log_2 m$.

2.5 A further algorithm

To complete our survey on addition chain algorithms, we cite the algorithm `bgmw` of Brickell *et al.* (1993) which cannot be formulated in $(\mathcal{A}, \circ)$. We give the main results for q-addition chains.

Lemma 20. *Let $m \in \mathbb{N}$. Then a q-addition chain for m can be computed according to Algorithm* `bgmw` *in at most $Q = r\lfloor \log_{q^r} m \rfloor$ q-steps and $A = q^r + \lfloor \log_{q^r} m \rfloor - 2$ further addition steps. We therefore get the length L of the q-addition chain as $L \leq A + Q = q^r + (r+1)\lfloor \log_{q^r} m \rfloor - 2$.*

Corollary 21. (Brickell *et al.* 1993) *Let $m, q \in \mathbb{N}$, $q \geq 2$, and $\mu = \log_q m$. There is a q-addition chain for m of length at most*

$$l(m) \leq \mu + \frac{\mu}{\log_q \mu}\left(1 + \frac{q}{\log_q \mu} + \frac{2\log_q \log_q \mu}{\log_q \mu - 2\log_q \log_q \mu}\right) \leq \mu + \frac{\mu}{\log_q \mu}(1 + o(1)).$$

2.6 Summarizing survey

The following tables show the results given before for addition chains. $m \in \mathbb{N}$ is the integer for which an addition chain has to be computed. We only consider the case $q = 2$ to facilitate comparison of all algorithms. Hence, the corresponding exponentiation algorithms need A multiplications and D squarings.

3 Addition chains for special values

3.1 Repeating sequences

So far we found some algorithms to compute short additon chains for arbitrary $m \in \mathbb{N}$. Now we concentrate on the following

Problem 22. Let $q, r, k, m, n \in \mathbb{N}$ with $0 < kr \leq n$ and $(s)_q = (s_{r-1}, \ldots, s_0)$ with $0 \leq s_i < q$ for all $0 \leq i < r$. Let $(m)_q = (\underbrace{(s)_q, (s)_q, \ldots, (s)_q}_{k})$. Find a good q-addition chain for m.

First, we reduce Problem 22 to a simpler one in two steps:

Algorithm	`binary`	`brauer`	`bgmw`
#steps L	$\lfloor \mu \rfloor + \nu_2(m) - 1$	$\nu_{2^r}(m) + 2^r - 3$ $+r\lfloor \frac{\mu}{r} \rfloor - (r - \lfloor \log_2 \lfloor \frac{m}{2^{r\lfloor \frac{\mu}{r} \rfloor}} \rfloor \rfloor)$	$(r+1)\lfloor \frac{\mu}{r} \rfloor + 2^r - 2$
#doublings D	$\lfloor \mu \rfloor$	$r\lfloor \frac{\mu}{r} \rfloor - (r - \lfloor \log_2 \lfloor \frac{m}{2^{r\lfloor \frac{\mu}{r} \rfloor}} \rfloor \rfloor)$	$r\lfloor \frac{\mu}{r} \rfloor$
#further steps A	$\nu_2(m) - 1$	$\nu_{2^r}(m) + 2^r - 3$	$\lfloor \frac{\mu}{r} \rfloor + 2^r - 2$
Upper bounds			
Parameter r		$\lfloor \frac{1}{2} \log_2 \mu \rfloor + 1$	$\lfloor \log_2 \mu - 2 \log_2 \log_2 \mu \rfloor + 1$
L_{worst}	$\leq 2\mu$	$\leq \mu + 2\frac{\mu}{\log_2 \mu}(1 + o(1))$	$\leq \mu + \frac{\mu}{\log_2 \mu}(1 + o(1))$
D_{worst}	$= \lfloor \mu \rfloor$	$\leq \mu$	$\leq \mu$
A_{worst}	$= \lceil \mu \rceil - 1$	$\leq 2\frac{\mu}{\log_2 \mu}(1 + \frac{\log_2 \mu}{\sqrt{\mu}})$	$< \frac{\mu}{\log_2 \mu}(1$ $+2\frac{\log_2 \log_2 \mu}{\log_2 \mu - 2 \log_2 \log_2 \mu} + \frac{2}{\log_2 \mu})$
$\#\mathcal{D}$	1	$2\sqrt{\mu}$	$\frac{\mu}{\log_2 \mu}(1 + o(1))$

Description: $\mu = \log_2 m$

Table 1. Theoretical comparison between the classical addition chain algorithms.

Algorithm	`yacobi`	`bocharova`	`lookahead`
#steps L	$\lfloor \mu \rfloor + 2S + R$	$\lfloor \mu \rfloor + 2r - S - s_1 - 2$	
#doublings D	$\lfloor \mu \rfloor + S$	$r + \lfloor \mu \rfloor - s_1$	
#further steps A	$R + S$	$r + S - 2$	
Average case			
Parameter r		$\lfloor \frac{\mu}{(\log_2 \mu)^2} \rfloor$	
$L_{ave} <$	$\lfloor \mu \rfloor + \frac{5}{2}\frac{\mu}{\log_2 \mu}(1 + o(1))$	$\lfloor \mu \rfloor + \frac{\mu}{(\log_2 \mu)}(1 + o(1))$	
$D_{ave} <$	$\lfloor \mu \rfloor + \frac{\mu}{\log_2 \mu}(1 + o(1))$	$\lfloor \mu \rfloor + \frac{\mu}{(\log_2 \mu)^2}$	
$A_{ave} <$	$\frac{3}{2}\frac{\mu}{\log_2 \mu}(1 + o(1))$	$\frac{\mu}{\log_2 \mu}(1 + o(1))$	
Upper bounds			
Parameter r		$\lfloor \frac{\mu}{(\log_2 \mu)^2} \rfloor$	
$L_{worst} <$	$\mu + \frac{5}{2}\frac{\mu \log_2 \log_2 \mu}{\log_2 \mu - \log_2 \log_2 \mu}(1 + o(1))$	$\mu + 2\frac{\mu \log_2 \log_2 \mu}{\log_2 \mu}(1 + o(1))$	$\frac{1}{2}\mu(\mu - 1) + \nu_2(m)$
$D_{worst} \leq$	$\mu + \frac{\mu \log_2 \log_2 \mu}{\log_2 \mu - \log_2 \log_2 \mu}(1 + o(1))$	$\mu + \frac{\mu}{(\log_2 \mu)^2}$	$\frac{1}{2}\mu(\mu - 1)$
$A_{worst} <$	$\frac{3}{2}\frac{\mu \log_2 \log_2 \mu}{\log_2 \mu - \log_2 \log_2 \mu}(1 + o(1))$	$2\frac{\mu \log_2 \log_2 \mu}{\log_2 \mu}(1 + o(1))$	$\nu_2(m)$
$\#\mathcal{D}$	S	$2r - 1$	

Description: S: #sequences, R: #sequences with last bit '1', s_1: length of first sequence, $\mu = \log_2 m$, A', D': #star steps/doublings in Part A

Table 2. Theoretical comparision between addition chain algorithms based on data compression.

1. Since $0 < s < q^r$, we may concentrate on $(m)_{q^r} = (\underbrace{s, \ldots, s}_{k})$ by changing from the q-ary to the q^r-ary representation of m. Hence, we concentrate on $(m)_q = (\underbrace{s, \ldots, s}_{k})$ with $0 < s < q$.
2. We have $m = \sum_{0 \le i < k} sq^i = s \cdot \sum_{0 \le i < k} q^i$. If we find an addition chain for $0 < s < q$ and a q-addition chain for $\sum_{0 \le i < k} q^i = \frac{q^k - 1}{q-1}$ we can easily build a q-addition chain for m by concatenating them according to the following remark.

Remark 23. Let $a, b \in \mathbb{N}$. Let $1 = a_0, \ldots, a_{L_a} = a$ an addition chain for a and $1 = b_0, \ldots, b_{L_b} = b$ be an addition chain for b. Then $1 = a_0, \ldots, a_{L_a} = a_{L_a} \cdot b_0, a_{L_a} \cdot b_1, \ldots, a_{L_a} \cdot b_{L_b} = a \cdot b$ is an addition chain for $a \cdot b$ of length $L_a + L_b$.

According to this we concentrate on q-addition chains for $(m)_q = (\underbrace{1, \ldots, 1}_{k})$.

We use word chains for simplicity in the sequel and transform the result to q-addition chains.

Algorithm 24 `only ones`. Input: $\mathcal{A} = \{0, \ldots, q-1\}$ a q-letter alphabet, $q, k \in \mathbb{N}$ and an addition chain $1 = a_0, \ldots, a_L = k$ for k of length L given by pairs of integers $(j_i, k_i) \in \mathbb{N}^2$ for $0 < i \le L$.

Output: A word chain for $(1, \ldots, 1) \in \mathcal{A}^k$ over $\mathcal{A}$.

1. Set $w[a_0] = (1)_q$.
2. For $1 \le i \le L$ compute
   $w[a_i] = w[a_{j_i}] \circ w[a_{k_i}]$. [Comment: the following invariant holds: $w[a_i] = (\sum_{0 \le j < a_i} q^j)_q$.]
3. Return $(1)_q = w[a_0], \ldots, w[a_L] = (m)_q$.

Lemma 25. *Let $q \in \mathbb{N}$, and $\mathcal{A} = \{0, \ldots, q-1\}$ be a q-letter alphabet. Then a word chain over $\mathcal{A}$ for $(1, \ldots, 1) \in \mathcal{A}^k$ with $l(k)$ concatenations $\mathcal{A}$ exists.*

Theorem 26. *Let $q \in \mathbb{N}$. Let an addition chain $1 = a_0, \ldots, a_L = k$ of length L for k be given. Then we can compute a q-addition chain for $m = \frac{q^k - 1}{q-1}$ containing L star steps and $\sum_{1 \le i \le L} a_{k_i}$ many q-steps.*

Remark. If $a_0, \ldots, a_L$ is an addition chain containing only star steps or doublings — which means that $j_i = i - 1$ for (j_i, k_i) for all $1 \le i \le L$ — then we have $\sum_{1 \le i \le L} a_{k_i} = a_L - 1$.

Let s, k, q, m as in Problem 22. We use Algorithm `bgmw` to create a q-addition chain for s of length L_s and Algorithm `brauer` to compute an addition chain for k of length $L_{m/s}$. Then we have a q-addition chain for m according to Remark 23 of length $L_s + L_{m/s}$. Inserting the results of Corollary 21 for s and Corollary 12 for k we get the following result as a simple consequence of Theorem 26 noting that Algorithm `brauer` generates an addition chain that satisfies Remark 3.1:

Result 27. Let $q, k, m, n, r \in \mathbb{N}$ as in Problem 22. Let $\sigma = \log_q s$ and $\kappa = \log_2 k$. Then a q-addition chain for $m = \frac{q^k-1}{q-1}$ can be computed with at most

$$A \leq \kappa + (\frac{\sigma}{\log_q \sigma} + 2\frac{\kappa}{\log_2 \kappa})(1 + o(1)) \text{ star steps and}$$
$$Q \leq \sigma + (k-1)r \text{ } q\text{-steps.}$$

3.2 Inversion in finite fields

From Fermat's Little Theorem we have $\alpha^{q^n-1} = 1$ in $\mathbb{F}_{q^n}$ for a prime power q, $n \in \mathbb{N}$ and $\alpha \in \mathbb{F}_{q^n} \setminus \{0\} = \mathbb{F}_{q^n}^{\times}$. We therefore can calculate the inverse of $\alpha \in \mathbb{F}_{q^n}^{\times}$ as $\alpha^{-1} = 1 \cdot \alpha^{-1} = \alpha^{q^n-1} \cdot \alpha^{-1} = \alpha^{q^n-2}$. But we have

$$q^n - 2 = q^n - q + q - 2 = (q^{n-1} - 1)q + (q-2) \quad (1)$$

and $(q^{n-1} - 1)_q = (q-1, \ldots, q-1)$ is of the type we have already mentioned.

Algorithm 28 `inverse`. Input: $\alpha \in \mathbb{F}_{q^n}^{\times}$ with a prime power q, $n \in \mathbb{N}$ and two addition chains: $1 = b_0, \ldots, b_{L_1} = n-1$ for $n-1$ of length L_1 and $1 = a_0, \ldots, a_{L_2} = q-2$ for $q-2$ of length L_2.
Output: $\alpha^{-1} \in \mathbb{F}_{q^n}$.

1. Calculate $y = \alpha^{q-2}$ using the addition chain for $q-2$.
2. Calculate $z = y \cdot \alpha = \alpha^{q-1}$.
3. Calculate $x = z^{\frac{q^{n-1}-1}{q-1}}$ using Algorithm 24 with input $k = n-1$, q and $b_0, \ldots, b_{L_1}$.
4. Return $x^q \cdot y$.

Theorem 29. *Let $\alpha \in \mathbb{F}_{q^n}^{\times}$, $q \in \mathbb{N}$ prime, and an addition chain for $n-1$ of length L_1 and, if $q > 2$ an addition chain for $q-2$ of length L_2 be given. Then we can evaluate $\alpha^{-1} \in \mathbb{F}_{q^n}$ with*

1. *$L_1 + L_2 + 2$ multiplications in $\mathbb{F}_{q^n}$ if $q > 2$, and*
2. *L_1 multiplications in $\mathbb{F}_{2^n}$ if $q = 2$.*

Let $b_{j_i} + b_{k_i} = b_i$ for $0 \leq j_i \leq k_i < i$ according to the first addition chain. Then we have to compute $1 + \sum_{i=1}^{L_1} b_{j_i}$ qth powers in $\mathbb{F}_{q^n}$.

Corollary 30. *Let $\alpha \in \mathbb{F}_{q^n}^{\times}$, $q \geq 2$ prime. Then the inverse of α in $\mathbb{F}_{q^n}$ can be computed using*

1. *$\log_2(n-1)(2 + \frac{2}{\log_2\log_2(n-1)} + \frac{2}{\sqrt{\log_2(n-1)}}) = \log_2(n-1)(2 + o(1))$ multiplications in $\mathbb{F}_{2^n}$ if $q = 2$, or*
2. *$\log_2(n-1)(2 + \frac{2}{\log_2\log_2(n-1)} + \frac{2}{\sqrt{\log_2(n-1)}}) + \log_2(q-2)(2 + \frac{2}{\log_2\log_2(q-2)} + \frac{2}{\sqrt{\log_2(q-2)}}) + 2 = (\log_2(n-1)(q-2))(2 + o(1))$ multiplications in $\mathbb{F}_{q^n}$ if $q \neq 2$.*

The computation needs $n-1$ further qth powers.

Remark. When using the binary method to generate an addition chain for $n-1$ we get Theorem 2 in Itoh & Tsujii (1988) as a special case of our result.

3.3 Comparison with inversion by Euclid

Another method to compute the inverse in $\mathbb{F}_{q^n} \cong \mathbb{F}_q[x]/(f)$, where $f \in \mathbb{F}_q[x]$ is irreducible of degree n, is via the Extended Euclidean Algorithm.

Definition 31. Let R be a ring. A function $\mathsf{M}\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ is called a *multiplication time* for $R[x]$ if polynomials in $R[x]$ of degree less than n can be multiplied using $O(\mathsf{M}(n))$ operations in R. It is assumed that $\mathsf{M}(n) \geq n$ and $\mathsf{M}(2n) \geq 2\mathsf{M}(n)$.

We can choose $\mathsf{M}(n) = n \log n \log\log n$ according to Schönhage & Strassen (1971). The fast Euclidean Algorithm due to Lehmer (1938), Knuth (1981), Schönhage (1971), Strassen (1983) yields the following.

Theorem 32. *1. The* gcd *of two univariate polynomials over a finite field $\mathbb{F}_{q^n}$ can be computed in $O(\mathsf{M}(n) \log n)$ operations in $\mathbb{F}_q$.*

2. For given $\alpha \in \mathbb{F}_{q^n}^\times$ the inverse $\alpha^{-1} \in \mathbb{F}_{q^n}^\times$ can be calculated with $O(\mathsf{M}(n) \log n)$ operations in $\mathbb{F}_q$.

The method based on Fermat needs $O(\mathsf{M}(n)) \log(n)(1 + o(1))$ operations in $\mathbb{F}_q$ if raising to the qth power is for free. This assumption can be made using a normal basis representation of $\mathbb{F}_{q^n}$ (see Section 7). Euclid's algorithm uses $O(\mathsf{M}(n) \log(n))$ operations in $\mathbb{F}_q$ as well and works on a power basis representation of $\mathbb{F}_{q^n}$. (We deal with the topic of representation of finite fields in Section 5.)

4 Practical results for addition chain heuristics

4.1 Numerical results in the literature

Several authors give numerical results for some of the addition chain algorithms. They concentrate on average and worst case for inputs of length $\lambda = 160$ bits and $\lambda = 512$ bits. A survey is given in Table 3.

4.2 Our experiment

We concentrate on addition chains for $q = 2$, and vary the number of bits between $\lambda = 160, 512$, and 1024. We also distinguish between different Hamming weights $\nu_2 \approx \frac{\lambda}{4}$, $\nu_2 \approx \frac{\lambda}{2}$, and $\nu_2 \approx \frac{3\lambda}{4}$. All of these 9 combinations are tested for 1000 randomly chosen input values. The parameters are chosen based on the theoretical results and optimized by practical trials. We use for `brauer` $r = \frac{1}{2}\log_2 \lambda + 1$, for `bgmw` $r = \log_2 \lambda - 2\log_2\log_2 \lambda + 3$, and for `bocharova` $r = \frac{\lambda}{(\log_2 \lambda)^2} + 4$. The results are presented in Table 4, 5 and 6 giving the average, the minimal, and the maximal values for each test series.

input λ	algorithm	reference	param. r	#steps aver	#steps max	#non-doub. aver	#non-doub. max	storage aver	storage max
160	`binary`	Brickell *et al.* (1993)	*	237	318				
	`bgmw`	Brickell *et al.* (1993)	$\log_2 12$			50.25	54	45	45
			$\log_2 19$			43.00	45	76	76
		de Rooij (1995)	?			50		45	47
	`brauer`	de Rooij (1995)	?	197				9	9
512	`binary`	Brickell *et al.* (1993)	*	765	1022				
	`bgmw`	Brickell *et al.* (1993)	$\log_2 26$			127.81	132	109	109
			$\log_2 45$			111.91	114	188	188
		de Rooij (1995)	?			128		109	111
	`brauer`	de Rooij (1995)	?	611				17	17
		Bocharova *et al.* (1995)	?			111		62	62
	`bocharova`	Bocharova *et al.* (1995)	?			102		16	16

Description: '?' no parameter is specified., '*' no parameter used

Table 3. Some numerical results on addition chain algorithms in the literature

ν_2	algorithm	#steps min	#steps aver	#steps max	#doublings min	#doublings aver	#doublings max	#non-doubs min	#non-doubs aver	#non-doubs max	storage min	storage aver	storage max
$\frac{\lambda}{4}$	`binary`	183	198	216		159		24	39	57		**1**	
	`brauer`	187	196	206		**156**		31	40	50		15	
	`bgmw`	187	196	206		**156**		31	40	50		40	
	`yacobi`	190	202	215	169	173	179	19	28	39	15	20	26
	`bocharova`	176	**187**	96	159	160	163	17	**26**	34		11	
	`lookback`	196	220	262	171	193	236	17	**26**	35	34	58	95
$\frac{\lambda}{2}$	`binary`	221	238	261		159		62	79	102		**1**	
	`brauer`	201	206	209		**156**		45	50	53		15	
	`bgmw`	201	206	209		**156**		45	50	53		40	
	`yacobi`	213	224	233	176	180	185	35	43	51	23	27	32
	`bocharova`	194	**200**	206	160	161	163	34	39	44		11	
	`lookback`	209	236	258	176	198	218	31	**37**	44	46	68	90
$\frac{3\lambda}{4}$	`binary`	262	278	299		159		103	119	140		**1**	
	`brauer`	206	208	209		**156**		50	52	53		15	
	`bgmw`	206	208	209		**156**		50	52	53		40	
	`yacobi`	219	231	241	177	182	186	40	49	55	23	27	31
	`bocharova`	192	**201**	209	159	160	163	33	40	46		11	
	`lookback`	221	245	266	186	207	230	30	**37**	45	51	78	102

Table 4. Number of steps for $\lambda = 160$ bit

ν_2	algorithm	#steps			#doublings			#non-doubs			storage		
		min	aver	max	min	aver	max	min	aver	max	min	aver	max
$\frac{\lambda}{4}$	`binary`	603	638	673		511		92	127	162		**1**	
	`brauer`	600	614	628		**507**		93	107	121		31	
	`bgmw`	602	616	629		510		92	106	119		103	
	`yacobi`	607	630	652	543	551	559	61	78	94	45	54	64
	`bocharova`	570	**588**	601	511	515	519	58	**72**	85		19	
	`lookback`	642	689	741	568	617	671	59	**72**	85	110	157	225
$\frac{\lambda}{2}$	`binary`	726	766	804		511		215	255	293		**1**	
	`brauer`	629	635	639		**507**		122	128	132		31	
	`bgmw`	630	637	641		510		120	127	131		103	
	`yacobi`	668	684	706	560	567	575	104	116	132	68	72	78
	`bocharova`	613	**621**	631	515	516	518	96	104	1114		19	
	`lookback`	691	730	773	596	630	673	91	**99**	109	150	184	227
$\frac{3\lambda}{4}$	`binary`	863	894	925		511		352	383	414		**1**	
	`brauer`	636	638	639		**507**		129	131	132		31	
	`bgmw`	639	640	641		510		129	130	131		103	
	`yacobi`	678	696	712	564	571	577	113	125	136	63	70	76
	`bocharova`	608	**621**	632	512	516	519	94	105	114		19	
	`lookback`	710	752	804	616	656	703	84	**95**	108	169	211	264

Table 5. Number of steps for $\lambda = 512$ bit

ν_2	algorithm	#steps			#doublings			#non-doubs			storage		
		min	aver	max	min	aver	max	min	aver	max	min	aver	max
$\frac{\lambda}{4}$	`binary`	1221	1279	1324		1023		198	256	301		**1**	
	`brauer`	1202	1219	1236		**1018**		184	201	218		63	
	`bgmw`	1200	1220	1236		1020		180	200	216		171	
	`yacobi`	1208	1239	1264	1085	1096	1107	121	143	162	85	99	110
	`bocharova`	1136	**1163**	1178	1026	1031	1034	110	**132**	146		27	
	`lookback`	1291	1354	1434	1165	1221	1300	113	**132**	147	222	283	372
$\frac{\lambda}{2}$	`binary`	1481	1534	1585		1023		458	511	562		**1**	
	`brauer`	1240	1247	1250		**1018**		222	229	232		63	
	`bgmw`	1240	1248	1251		1020		220	228	231		171	
	`yacobi`	1314	1334	1356	1115	1125	1134	195	209	224	123	130	140
	`bocharova`	1211	**1221**	1233	1031	1032	1033	178	188	200		27	
	`lookback`	1374	1424	1492	1193	1245	1315	169	**179**	192	281	332	402
$\frac{3\lambda}{4}$	`binary`	1750	1790	1837		1023		727	767	814		**1**	
	`brauer`	1248	1249	1250		**1018**		230	231	232		63	
	`bgmw`	1249	1250	1251		1020		229	230	231		171	
	`yacobi`	1328	1351	1376	1120	1130	1140	207	221	237	117	124	132
	`bocharova`	1202	**1218**	1230	1027	1031	1034	172	186	196		27	
	`lookback`	1397	1466	1526	1233	1297	1365	155	**168**	184	311	382	452

Table 6. Number of steps for $\lambda = 1024$ bit

4.3 Results

We can divide the algorithms in two groups: the first one computes $\mathcal{D}$ only depending on λ: `binary`, `brauer`, and `bgmw`. The other three algorithms pay also attention to the binary form of the input: `yacobi`, `bocharova`, and `lookback`.

`binary` has only one advantage: it needs least storage. The number of doublings is roughly the same as for `brauer` and `bgmw`, but the number of non–doublings is higher, depending on ν_2. Therefore, `binary` should only be used if $\nu < \frac{\lambda}{4}$. `brauer` and `bgmw` show no real difference in the number of doublings and non–doublings. `brauer` uses 2 to 3 times less storage, but `bgmw` stores only powers of $q = 2$; hence, no storage is needed if the cost of computing doublings are negligible.

The second group is inhomogenous. `bocharova` generates the shortest addition chains of all given algorithms on average and needs least storage (not counting `binary`). The number of non–doublings is very low without increasing the number of doublings very much in comparison to the first group. The increase of the number of doublings causes the large number of steps for `yacobi` and `lookback`. For exponentiation in finite fields of characteristic 2, the number of non–doublings is the crucial parameter (Section 9); `lookback` wins with respect to this. The storage requirement can be reduced to $\approx \frac{1}{3}$ if doublings can be computed with very low cost. The number of steps for all algorithms of the second group scatters in a wide range. But this is clear because these algorithms are based on data compression techniques. They should be preferred if the number of non–doublings is most important.

4.4 Theory vs. practice

Comparing practical and theoretical results we recall that the theoretical bounds are asymptotical. But the practical experiments use relatively short inputs with $\lambda \leq 1024$ bit. This may explain one of the two discrepancies between theory and practice: `yacobi` needs fewer non–doubling steps than predicted compared to `bgmw`. The assumptions used for the theoretical analysis of the average case for `yacobi` may not be entirely correct in this practical situation. The other surprise is that `brauer` and `bgmw` show no discrepancy in practice. This points out that the worst case estimates given in the literature for `brauer` are not sharp enough. Altogether the experiments confirm the theoretical results for the five known algorithms. The new algorithm `lookback` looks promising for exponentiation in $\mathbb{F}_{2^n}$; see the following sections.

5 Finite fields

The second point to deal with when discussing exponentiation in the finite field $\mathbb{F}_{q^n}$, is to speed up the time needed for a single multiplication or raising to a determined power, respectively. We will continue the separation between multiplication and raising to the qth power when discussing how to speed up basic arithmetic operations in $\mathbb{F}_{q^n}$.

5.1 Normal bases

We recall some definitions and facts about finite fields that are needed in the sequel. Further details can be found in Lidl & Niederreiter (1983).

Let $\mathbb{E}$ be the splitting field of $x^r - 1$ over $\mathbb{F}_q$ and $\gcd(r, q) = 1$. Then the roots $\zeta_1, \ldots, \zeta_r$ of $x^r - 1$ are called the *rth roots of unity* over $\mathbb{F}_q$. The set of all rth roots of unity over $\mathbb{F}_q$ is a cyclic subgroup of the splitting field of $x^r - 1$ over $\mathbb{F}_q$ with respect to multiplication. Let ζ be an rth root of unity over $\mathbb{F}_q$. If ζ generates a multiplicative subgroup of order r in the splitting field of $x^r - 1 \in \mathbb{F}_q[x]$ then ζ is called *primitive.* The polynomial

$$\Phi_r(x) = \prod_{\substack{1 \le i \le r \\ \gcd(i,r)=1}} (x - \zeta^i) \in \mathbb{F}_q[x]$$

is the rth *cyclotomic polynomial* over $\mathbb{F}_q$.

Definition 33. A *normal basis* $\mathcal{N} = (\alpha_0, \ldots, \alpha_{n-1})$ of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ is a basis with $\alpha_0, \alpha_1 = \alpha_0^q, \ldots, \alpha_{n-1} = \alpha_0^{q^{n-1}}$. In this case, $\alpha_0 \in \mathbb{F}_{q^n}$ is called a *normal basis generator* or a *normal element* over $\mathbb{F}_q$.

5.2 The representation of finite fields

A crucial point is the representation of the elements of a finite field $\mathbb{F}_{q^n}$.

We can regard $\mathbb{F}_{q^n}$ as a vector space of dimension n over $\mathbb{F}_q$. Thus $\mathbb{F}_{q^n}$ can be identified with $\mathbb{F}_q^n$. If $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{F}_{q^n}$ form a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, $\alpha \in \mathbb{F}_{q^n}$ can be uniquely written as $\alpha = \sum_{0 \le i < n} a_i \alpha_i$ with $a_0, \ldots, a_{n-1} \in \mathbb{F}_q$. We concentrate on two different ways to represent the elements of $\mathbb{F}_{q^n}$ in the sequel.

1. Let f be an irreducible polynomial in $\mathbb{F}_q[x]$ of degree n. Because $\mathbb{F}_{q^n}$ is the splitting field of f over $\mathbb{F}_q$, we have $\mathbb{F}_{q^n} \cong \mathbb{F}_q[x]/(f)$ and any $\alpha \in \mathbb{F}_{q^n}$ can be represented by a polynomial of degree at most $n - 1$ over $\mathbb{F}_q$. So arithmetic here means polynomial arithmetic in $\mathbb{F}_q[x]$ modulo f. We call this a *polynomial representation* of $\mathbb{F}_{q^n}$. If $\alpha = x \bmod f$ in $\mathbb{F}_q[x]/(f)$, then $\mathcal{B} = (1, \alpha, \ldots, \alpha^{n-1})$ is a basis for $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$.
2. A normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}$ gives a *normal basis representation* of $\mathbb{F}_{q^n}$.
3. Let $\zeta \in \mathbb{F}_{q^n}$ be primitive. Then we can represent $\alpha \in \mathbb{F}_{q^n} \setminus \{0\}$ by $\log_\zeta \alpha \in \mathbb{N}$, with $0 \le \log_\zeta \alpha \le q^n - 2$. This can be used to implement arithmetic efficiently in small finite fields, with the help of exp– and log–tables stored in main memory. This is the *primitive element representation* of $\mathbb{F}_{q^n}$. It is useful only for small fields.

6 Polynomial representation

6.1 Modular composition

Definition 34. Let R be a ring. A real number $\omega \in \mathbb{R}_{>0}$ is called a *feasible matrix multiplication exponent* if matrices in $R^{n \times n}$ can be multiplied using $O(n^\omega)$ operations in R.

Theorem 35. (Strassen 1969) *Two square matrices* $A, B \in \mathbb{F}_q^{m\times m}$ *with* $m = 2^t$ *and* $t \in \mathbb{N}$ *can be multiplied with* $O(m^\omega)$ *operations in* $\mathbb{F}_q$*, where* $\omega = \log_2 7 \approx 2.80735492$.

Strassen's result for ω has been improved since. The current world record is $\omega < 2.376$ (Coppersmith & Winograd 1990).

A basic tool in the algorithm of Shoup (1994) is the calculation of modular compositions as introduced in the *iterated Frobenius* algorithm of von zur Gathen & Shoup (1992). Let $f, g, h \in \mathbb{F}_q[x]$ with $\deg f = n$ and $\deg g, \deg h < n$. The *modular composition* of g and h is given by $g(h) \bmod f$.

Fact 36. *Let* $f, g, h \in \mathbb{F}_q[x]$ *and* $r \in \mathbb{N}$ *with* $h = x^{q^r} \operatorname{rem} f$*. Then* $g^{q^r} \equiv g(h) \bmod f$.

Hence we can use modular composition to raise to the q^rth power in $\mathbb{F}_q[x]/(f)$ for any $r \in \mathbb{N}$.

Theorem 37. (Brent & Kung 1978) *We can compute* $g(h) \operatorname{rem} f$ *using* $O(n^{1/2}\mathsf{M}(n) + n^{(\omega+1)/2})$ *operations in* $\mathbb{F}_q$.

Remark. Modular composition can be done with

1. $O(n^{5/2})$ operations using classical arithmetic, i.e. $\mathsf{M}(n) = O(n^2)$ and $\omega = 3$.
2. $O(n^{1/2}(n^{\log_2 3} + n^{\log_2 7/2})) = O(n^{1/2+\log_2 3}) = O(n^{2.085})$ operations using the algorithms of Karatsuba & Ofman and Strassen, i.e. $\mathsf{M}(n) = O(n^{\log_2 3})$ and $\omega = \log_2 7$.
3. $O(n^{3/2}\log n \log\log n + n^{(\omega+1)/2}) = O(n^{1.668})$ operations with $\omega < 2.376$ using the results of Schönhage & Strassen (1971), Schönhage (1977) and Cantor & Kaltofen (1991) for $\mathsf{M}(n)$ and Coppersmith & Winograd (1990) for ω, i.e. $\mathsf{M}(n) = O(n \log n \log\log n)$ and $\omega < 2.376$.

6.2 A more detailed model for counting operations

We can estimate the cost for one multiplication of two polynomials modulo a fixed polynomial f of degree n by $3\mathsf{M}(n) + n$ ignoring the precomputation of the inverse of the reverse of f modulo x^n. A cyclic shift of coefficients is assumed to be free.

Corollary 38. *Modular composition can be done using at most*

$$9\sqrt{n}\mathsf{M}(n) + 4n^{3/2} + \lceil \sqrt{n}\, \rceil O(\sqrt{n}^{\;\omega})$$

operations in $\mathbb{F}_q$*. If classical matrix multiplication is used, we have* $\omega = 3$ *and* $\lceil \sqrt{n}\, \rceil O(n^{3/2}) = 2n^2(1 + o(1))$.

6.3 Shoup's algorithm

Algorithm 39 `exponentiation with composition`. Input: $f, b \in \mathbb{F}_q[x]$ with $\deg b < \deg f = n$, $e \in \mathbb{N}$ with $0 < e < q^n$ and a parameter $r \in \mathbb{N}$.

Output: $y = b^e \operatorname{rem} f$.

1. Let $(e)_{q^r} = (e_{\lambda-1}, \ldots, e_0)$ be the q^r-ary representation of e with $0 \le e_i < q^r$ for all $0 \le i < \lambda$ where $\lambda = \lfloor \log_{q^r} e \rfloor + 1$.
2. (Pre)Compute and store all values $b^{e_i} \operatorname{rem} f$ for $0 \le i < \lambda$.
3. Compute $h = x^{q^r} \operatorname{rem} f$.
4. Let $y = b^{e_{\lambda-1}} \operatorname{rem} f$. For $i = \lambda - 2$ downto 0 do
    5. Compute $y = y(h) \operatorname{rem} f$ by modular composition according to Brent & Kung (1978).
    6. Compute $y = yb^{e_i} \operatorname{rem} f$ using precomputed values.
7. Return y.

Theorem 40. (Shoup 1994) *Let $b \in \mathbb{F}_{q^n}$ and $0 < e < q^n$. Then b^e can be evaluated with $O(\mathsf{M}(n)\frac{n}{\log n} + \sqrt{n}^{\,\omega+1} \log n)$ operations in $\mathbb{F}_q$. Using fast polynomial arithmetic we have $O(n^2 \log\log n)$ operations in $\mathbb{F}_q$ and storage for $O(\frac{n}{(\log n)^2})$ elements of $\mathbb{F}_{q^n}$.*

Corollary 41. *Algorithm* `exponentiation with composition` *computes $b^e \in \mathbb{F}_{q^n}$ for $b \in \mathbb{F}_{q^n}$ and $e \in \mathbb{N}$ with $e < q^n$ using at most*

$$(9(\log_2 q)^2 \frac{n}{\log_2 n} + \frac{27}{\log_2 q} n^{\frac{1}{2}} \log_2 n + \frac{3}{\log_2 q} \log_2 n)\mathsf{M}(n)(1 + o(1))$$
$$+ (3(\log_2 q)^2 \frac{n}{\log_2 n} + \frac{2n + 4n^{\frac{1}{2}} + 1}{\log_2 q} \log_2 n)n(1 + o(1))$$

operations in $\mathbb{F}_q$.

6.4 Number of operations

We summarize the results of this section in the following theorem:

Theorem 42. *Let $q, n \in \mathbb{N}$. Then the following holds in the polynomial representation for $\mathbb{F}_{q^n}$:*

1. *Addition of two elements can be done with n additions in $\mathbb{F}_q$.*
2. *Multiplication of two elements can be done with $O(n \log(n) \log\log(n))$ operations in $\mathbb{F}_q$.*
3. *Exponentiation of an element can be done with $O(n^2 \log\log n)$ operations in $\mathbb{F}_q$ using storage for $O(n/(\log n)^2)$ elements of $\mathbb{F}_{q^n}$.*

7 Normal bases

We examine a representation by a normal basis $\mathcal{N} = (\alpha_0, \ldots, \alpha_{n-1})$ of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, as in Definition 33. By the Normal Basis Theorem, $\mathbb{F}_{q^n}$ has always a normal basis over $\mathbb{F}_q$. We know that the Frobenius automorphism $\sigma\colon \mathbb{F}_{q^n} \to \mathbb{F}_{q^n} : \alpha \mapsto \alpha^q$ is a linear operator on $\mathbb{F}_{q^n}$, as a $\mathbb{F}_q$-vector space. Therefore we have for an arbitrary $\beta = \sum_{0\le i<n} b_i\alpha_i \in \mathbb{F}_{q^n}$ with $(\beta)_{\mathcal{N}} = (b_0, \ldots, b_{n-1})$ that $\beta^q = \sigma(\beta) = \sigma(\sum_{0\le i<n} b_i\alpha_i) = \sum_{0\le i<n} b_i\sigma(\alpha_i) = \sum_{0\le i<n} b_i\alpha_{i+1}$. Thus $(\beta^q)_{\mathcal{N}} = (b_{n-1}, b_0, \ldots, b_{n-2})$ is just a cyclic shift of the coordinates of β. It is therefore customary to neglect the cost of raising to the qth power (cf. Agnew *et al.* 1988, Stinson 1990, von zur Gathen 1991, Jungnickel 1993) because no arithmetic operation in $\mathbb{F}_q$ has to be done. However, our notion of q-addition chains is designed to also keep track of these operations.

Unfortunately, multiplication is more difficult and expensive. To illustrate this (see Mullin *et al.* 1989, Menezes *et al.* 1993, Chapter 5) let $(\delta)_{\mathcal{N}} = (\beta\cdot\gamma)_{\mathcal{N}} \in \mathbb{F}_{q^n}$. Then, expressing the d_k's in terms of b_i's and c_j's, we have $\delta = \sum_{0\le k<n} d_k\alpha_k = (\sum_{0\le i<n} b_i\alpha_i)(\sum_{0\le j<n} c_j\alpha_j) = \sum_{0\le i,j<n} b_ic_j\alpha_i\alpha_j$. We define the *multiplication tensor* to consist of the n matrices $T_k = (t_{ij}^{(k)})_{0\le i,j<n} \in \mathbb{F}_q^{n\times n}$ with $\alpha_i\alpha_j = \sum_{0\le k<n} t_{ij}^{(k)}\alpha_k$. Then we get

$$\sum_{0\le i,j<n} b_ic_jt_{ij}^{(k)} = d_k = \beta\cdot T_k\cdot\gamma^T \text{ for all } 0\le k<n. \tag{2}$$

In a normal basis $\mathcal{N}$, we can find a single matrix $T_{\mathcal{N}} = (t_{ij})_{0\le i,j<n} \in \mathbb{F}_q^{n\times n}$, so that $t_{ij}^{(k)} = t_{i-j,0}^{(k-j)} = t_{i-j,k-j}$ for all $0\le i,j,k<n$. We call $\mathcal{T}_{\mathcal{N}}$ the *multiplication table* of the normal basis $\mathcal{N}$. Equation (2) leads directly to a multiplication algorithm in $\mathbb{F}_{q^n}$. The number of multiplications in $\mathbb{F}_q$ depends on the number of non–zero entries in $\mathcal{T}_{\mathcal{N}}$, which is called the *density* $c_{\mathcal{N}}$ of $\mathcal{N}$ in the sequel.

Lemma 43. *Multiplying two elements of $\mathbb{F}_{q^n}$ given in a normal basis representation can be done with $2nc_{\mathcal{N}}$ multiplications in $\mathbb{F}_q$ and storage for $c_{\mathcal{N}}$ elements of $\mathbb{F}_q$.*

Theorem 44. (Mullin *et al.* 1989) *If $\mathcal{N}$ is a normal basis for $\mathbb{F}_{q^n}$, then $c_{\mathcal{N}} \ge 2n-1$.*

Mullin *et al.* (1989) call *optimal* a normal basis $\mathcal{N}$ with minimal density $c_{\mathcal{N}} = 2n-1$, and show how to construct optimal normal bases over $\mathbb{F}_2$ for certain $\mathbb{F}_{2^n}$.

To construct a normal basis $\mathcal{N}$ for $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ with low density $c_{\mathcal{N}}$, we introduce Gauß periods.

Definition 45. Let $n, k \in \mathbb{N}$ such that $r = nk+1$ is prime. Let $\mathcal{K} < \mathbb{Z}_r^\times$ be the unique subgroup of $\mathbb{Z}_r^\times$ of order k, and let ζ be a primitive rth root of unity in $\mathbb{F}_{q^{nk}}$. Then $\alpha = \sum_{a\in\mathcal{K}} \zeta^a$ is called a *Gauß period* of type (n,k) over $\mathbb{F}_q$.

Theorem 46. *In the above notation, the Gauß period* $\alpha = \sum_{a\in\mathcal{K}} \zeta^a$ *generates a normal basis* $\mathcal{N} = (\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}})$ *of* $\mathbb{F}_{q^n}$ *over* $\mathbb{F}_q$ *if and only if* $\gcd(e, n) = 1$, *where* e *is the index of* $q \bmod r$ *in* $\mathbb{Z}_r^\times$.

A proof can be found in Gao *et al.* (1995); see also Gao & Lenstra (1992), and Wassermann (1993). Gao & Lenstra (1992) showed the following; see also Menezes *et al.* (1993).

Theorem 47. (Optimal normal basis theorem) *Any optimal normal basis of* $\mathbb{F}_{q^n}$ *over* $\mathbb{F}_q$ *is generated by a Gauß period of type* (n, k), *where* $r = kn + 1$ *is prime and*

1. q *is a prime power,* $k = 1$, *and* $\mathbb{Z}_{n+1}^\times = \langle q \rangle$, *or*
2. $q = 2$, $k = 2$, *and either* $\langle 2 \rangle = \mathbb{Z}_{2n+1}^\times$, *or* $2n + 1 \equiv 3 \bmod 4$ *and* $\langle 2 \rangle = \{a \in \mathbb{Z}_{2n+1} \colon \exists x \in \mathbb{Z}_{2n+1} : x^2 \equiv a \bmod 2n + 1\}$.

Therefore, there are finite fields for which no optimal normal basis exists. We concentrate on normal bases generated by Gauss periods because of the following:

Fact 48. *Let* $\mathcal{N}$ *be a normal basis constructed according to Theorem 46. Then*

$$c_{\mathcal{N}} \leq (n-1)k + n.$$

For a proof, see Geiselmann (1994) or Menezes *et al.* (1993).

Hence, we have a new parameter k in the estimation of the density $c_{\mathcal{N}}$. To construct 'good' normal bases we therefore have to examine if there exists a small k for given $q, n \in \mathbb{N}$. This leads to the following definition (von zur Gathen & Schlink 1996):

$$\kappa'_q(n) = \begin{cases} \inf k & : \quad (n,k) \text{ Gauß period of type } (n,k) \text{ over } \mathbb{F}_q, \text{ if any exist,} \\ \infty & : \quad \text{if no such Gauß periods exist.} \end{cases}$$

Fact 49. (Wassermann 1993, Satz 3.3.4) *Let* $q = p^t$, p *a prime,* $t \in \mathbb{N}$ *with the notations above. Then* $\kappa'_q(n) < \infty$ *if and only if the following conditions hold*

1. $\gcd(n, t) = 1$ *and*
2. *either* $2p \nmid n$ *and* $p \equiv 1 \bmod 4$, *or* $4p \nmid n$.

Theorem 50. *Let* $q, n, k \in \mathbb{N}$ *satisfy the conditions of Theorem 46. Then using the normal basis representation for* $\mathbb{F}_{q^n}$ *the following hold:*

1. *The addition of two elements in* $\mathbb{F}_{q^n}$ *can be done with* n *additions in* $\mathbb{F}_q$.
2. *The multiplication of two elements in* $\mathbb{F}_{q^n}$ *can be done with* $O(n^2k)$ *operations in* $\mathbb{F}_q$.
3. *The exponentiation of an element in* $\mathbb{F}_{q^n}^\times$ *can be done with* $2nc_{\mathcal{N}}\frac{n}{\log_q n}(1 + o(1)) \leq 2\log_2 q \frac{n^3 k}{\log_2 n}(1 + o(1))$ *operations in* $\mathbb{F}_q$. $O(\frac{n}{\log n})$ *elements of* $\mathbb{F}_{q^n}$ *and* $c_{\mathcal{N}}$ *elements of* $\mathbb{F}_q$ *have to be stored.*

Corollary 51. *Exponentiation of an element in* $\mathbb{F}_{q^n}^\times$ *can be done with* $2\log_2 q \frac{n^2 c_{\mathcal{N}} + n^3}{\log_2 n}(1 + o(1))$ *operations in* $\mathbb{F}_q$.

8 Using fast multiplication within normal basis representation

Gao *et al.* (1995) provide a way to connect fast multiplication (using polynomial basis representation) and free raising to the qth power in $\mathbb{F}_{q^n}$ (using normal basis representation). They have the following results.

Theorem 52. *Let $q, n, k \in \mathbb{N}$ satisfy the conditions of Theorem 46. Then the following holds for the normal basis representation of elements of $\mathbb{F}_{q^n}$:*

1. *Addition of two elements can be done with n additions in $\mathbb{F}_q$.*
2. *Multiplication of two elements can be done with $O(nk\log(nk)\log\log(nk))$ operations in $\mathbb{F}_q$.*
3. *Exponentiation of an element uses at most $O(\frac{n^2k}{\log n}\log(nk)\log\log(nk))$ operations in $\mathbb{F}_q$. The algorithm needs to store $O(\frac{n}{\log n})$ elements of $\mathbb{F}_{q^n}$.*

Before we introduce the results of our implementations we give a theoretical comparison of the three exponentiation algorithms for $\mathbb{F}_{q^n}$ we have analyzed. We restrict to the case $q = 2$ and $k \leq 2$, i.e. the following Table 7 is only valid for field extensions over $\mathbb{F}_2$ for which a optimal normal basis exists.

We use the following short names:

- `onb`: Algorithm `bgmw` in connection with normal basis representation for $\mathbb{F}_{2^n}$ using the multiplication table for multiplication.
- `shoup`: Abbreviation for Algorithm 39 `exponentiation with composition` in the polynomial representation of $\mathbb{F}_{2^n}$.
- `ggp`: Algorithm `bgmw` in connection with fast polynomial multiplication and normal basis representation for $\mathbb{F}_{2^n}$.

Algorithm	onb	ggp	shoup
total operations	$O(\frac{n^3}{\log n})$	$O(n^2 \log\log n)$	$O(n^2 \log\log n)$
block operations			
$c_{\mathsf{M}} \cdot \mathsf{M}(n)(1+o(1))$	$c_{\mathsf{M}} = 0$	$c_{\mathsf{M}} \leq k^2 \frac{n}{\log_2 n}$	$c_{\mathsf{M}} = 9\frac{n}{\log_2 n} + 27n^{\frac{1}{2}}\log_2 n$ $+3\log_2 n$
$+$			
$c_{\mathsf{S}} \cdot n(1+o(1))$	$c_{\mathsf{S}} = 6\frac{n^2}{\log_2 n}$	$c_{\mathsf{S}} = 2k\frac{n}{\log_2 n}$	$c_{\mathsf{S}} = 2n\log_2 n + 3\frac{n}{\log_2 n}$ $+4n^{\frac{1}{2}}\log_2 n + \log_2 n$
$(\omega = 3)$			
storage	$O(\frac{n}{\log n})$	$O(\frac{n}{\log n})$	$O(\frac{n}{(\log n)^2})$

Table 7. Theoretical comparison between three exponentiation algorithms over $\mathbb{F}_{2^n}$ with a Gauß period of type (n, k).

9 Practical comparison of exponentiation algorithms

We implemented the three algorithms `onb`, `ggp` and `shoup` on a Sun Sparc Ultra 1 computer, rated at 143 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 used the software library written in `C++` by Jürgen Gerhard that is described in von zur Gathen & Gerhard (1996), Section 10. This library offers fast polynomial arithmetic over $\mathbb{F}_2$ including several algorithms for polynomial multiplication over $\mathbb{F}_2$: the classical method, Karatsuba & Ofman's algorithm and the method introduced by Cantor (1989). We use the library's implementation of modular composition according to Brent & Kung (1978), based on classical matrix multiplication.

We only consider field extensions over $\mathbb{F}_2$ of degree n for which an optimal normal basis exists, i.e., the normal basis corresponds to a Gauß period of type (n, k) with $k \in \{1, 2\}$. We use two different series of values for n: We choose $n \in \mathbb{N}, n \approx 200 \cdot i, 1 \leq i \leq 50$ as *test series 1* to examine in detail practical aspects of the three exponentiation algorithms. In cryptography values for n between 512 and 1024 have been used for cryptosystems (cf. the remarks in Brickell *et al.* 1993 and Odlyzko 1985). *Test series 2* consists of $n \in \mathbb{N}, n \approx 2^i, 10 \leq i \leq 16$ and some intermediate values. Using this input we want to give an idea of the asymptotic behaviour of the three exponentiation algorithms. The exponents are randomly chosen and uniformly distributed in $\{1, \ldots, 2^n - 1\}$.

The results of our practical comparison for $\mathbb{F}_{2^n}, n \leq 10000$ are clear with respect to normal basis representation (cf. Figure 2): using a multiplication matrix — even with low density — for multiplication is too slow. Software based implementation of the *Massey–Omura multiplier* is only useful for small field extensions of $\mathbb{F}_2$. This corresponds to our theoretical results: `onb` uses $O(\frac{n^3}{\log n})$ operations in $\mathbb{F}_2$ (Theorem 50), but `ggp` and `shoup` both use about $O(\frac{n^{2.6}}{\log n})$ operations because polynomial multiplication for degrees $n \leq 10000$ is implemented with Karatsuba & Ofman's algorithm, so that $\mathsf{M}(n) = O(n^{\log_2 3})$. In theory both algorithms need about $O(\frac{n^{2.6}}{\log n})$ operations. But a closer look at the hidden constants shows that in `ggp` for $k = 2$ we have $c_{\mathsf{M}} \leq k^{\log_2 3} \frac{n}{\log_2 n} = 3 \frac{n}{\log_2 n}$ and for `shoup` we have $c_{\mathsf{M}} = 9 \frac{n}{\log_2 n}$ (Corollary 41). In the experiments, the quotient grew from about 2 to almost 5 (cf. Tables 8 and 9).

The advantage of `shoup` is that it can be used for all $n \in \mathbb{N}$ even when no Gauß period of type (n, k) with small k exists.

10 Conclusion

Finally we want to outline the main properties for a fast software exponentiation algorithm in $\mathbb{F}_{2^n}$ for large $n \in \mathbb{N}$:

1. The algorithm should use fast polynomial multiplication. Neither multiplication by multiplication tensors nor classical polynomial arithmetic is fast

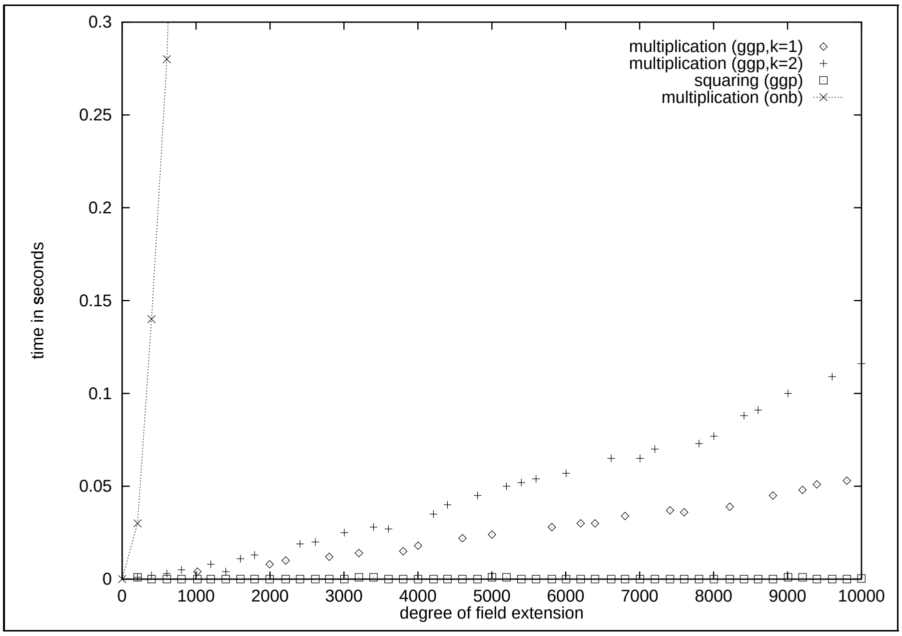

Fig. 1. The time for squaring and the dependence of the multiplication time in ggp on k compared to multiplication time in onb.

enough.

2. The algorithm should be based upon an addition chain for the exponent e with a small number of non–doubling steps.
3. The algorithm should offer a cheap way to compute $\alpha^{2^m} \in \mathbb{F}_{2^n}$ for $m \in \mathbb{N}$ and $\alpha \in \mathbb{F}_{2^n}$. Both Shoup's and Gao *et al.*'s algorithm achieve this.

References

G. B. Agnew, R. C. Mullin, and S. A. Vanstone, Fast exponentiation in $GF(2^n)$. In *Advances in Cryptology—EUROCRYPT '88*, ed. C. G. Günther, vol. 330 of *Lecture Notes in Computer Science*, 251–255. Springer, Berlin, 1988.

I. Bocharova and B. Kudryashov, Fast exponentiation in cryptography. In *Proceedings Applied algebra, algebraic algorithms and error correcting codes: 11th International Symposium AAECC*, ed. G. Cohen, Lecture notes in computer science **948**, Berlin, 1995, Springer, 146–157.

A. Brauer, On addition chains. *Bull. Amer. Math. Soc.* **45** (1939), 736–739.

R. P. Brent and H. T. Kung, Fast algorithms for manipulating formal power series. *J. Assoc. Comput. Mach.* **25** (1978), 581–595.

E. Brickell, D. Gordon, K. McCurley, and D. Wilson, Fast exponentiation with precomputation. In *Advances in cryptology: Proceedings EUROCRYPT '92*, ed. R. Rueppel, Lecture notes in computer science **658**, Berlin, 1993, Springer, 200–207.

D. G. Cantor, On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A* **50** (1989), 285–300.

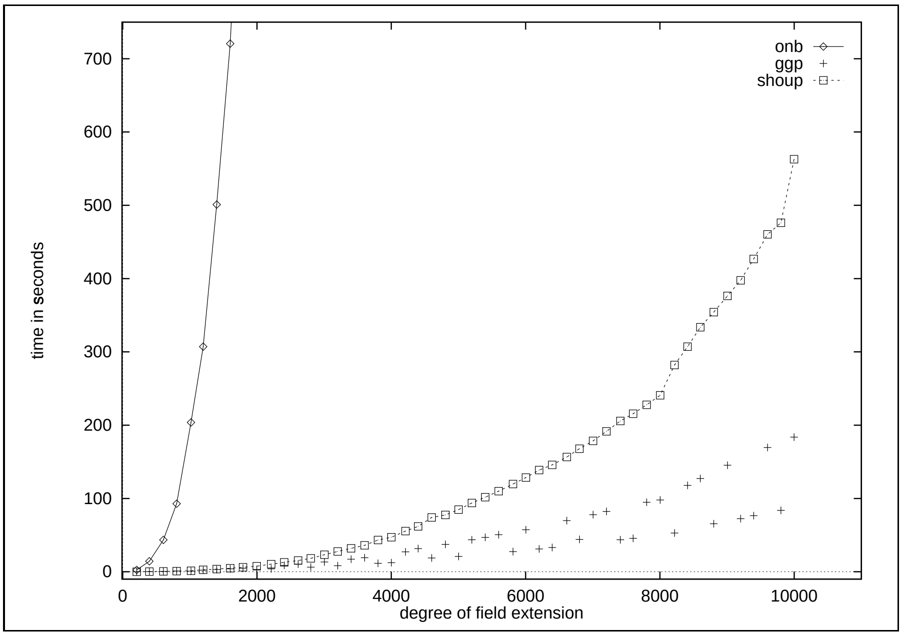

Fig. 2. Comparison of the three exponentiation algorithms for $n \leq 10000$

D. G. Cantor and E. Kaltofen, On fast multiplication of polynomials over arbitrary algebras. *Acta. Inform.* **28** (1991), 693–701.

D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions. *J. Symb. Comp.* **9** (1990), 251–280.

W. Diffie and M. E. Hellman, New directions in cryptography. *IEEE Trans. Inform. Theory* **22** (1976), 644–654.

P. Downey, B. Leong, and R. Sethi, Computing sequences with addition chains. *SIAM J. Comput.* **10**(3) (1981), 638–646.

T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on information theory* **IT-31**(4) (1985), 469–472.

S. Gao and H. W. Lenstra, Jr., Optimal normal bases. *Designs, Codes, and Cryptography* **2** (1992), 315–323.

S. Gao, J. von zur Gathen, and D. Panario, Gauss periods and fast exponentiation in finite fields. In *Proc. Latin '95, Valparaiso, Chile*, Springer Lecture Notes in Computer Science **911**, 1995, 311–322.

J. von zur Gathen, Efficient and optimal exponentiation in finite fields. *Comput complexity* **1** (1991), 360–394.

J. von zur Gathen and J. Gerhard, Arithmetic and factorization of polynomials over $\mathbb{F}_2$. In *Proc. ISSAC '96, Zürich, Switzerland.* ACM press, 1996, 1–9.

Joachim von zur Gathen and Sandra Schlink, Normal bases via general Gauss periods. Reihe Informatik tr-ri-96-177, Universität-Gesamthochschule Paderborn, 1996.

J. von zur Gathen and V. Shoup, Computing Frobenius maps and factoring polynomials. *Computational complexity* **2** (1992), 187–224.

		onb	ggp	shoup			onb	ggp	shoup
n	k	t/sec	t/sec	t/sec	n	k	t/sec	t/sec	t/sec
209	2	2.44	0.09	0.04	5199	2	20449.90	43.70	93.80
398	2	14.30	0.26	0.19	5399	2	21961.90	46.92	101.82
606	2	43.47	0.60	0.46	5598	2	24424.30	50.62	110.02
803	2	92.86	1.00	0.82	5812	1	27082.60	27.56	119.65
1018	1	203.79	0.90	1.32	6005	2	30688.90	57.39	128.65
1199	2	307.07	2.18	2.65	6202	1		31.13	138.82
1401	2	500.96	3.08	3.67	6396	1		33.18	145.90
1601	2	720.60	3.95	4.89	6614	2		69.76	156.61
1791	2	1049.14	4.75	6.21	6802	1		44.12	167.96
1996	1	1251.76	3.19	7.66	7005	2		77.96	178.61
2212	1	1738.70	4.04	10.46	7205	2		82.39	191.55
2406	2	2256.20	8.81	12.88	7410	1		43.63	205.82
2613	2	2921.65	10.45	15.35	7602	1		45.60	215.70
2802	1	3332.23	6.28	18.28	7803	2		94.78	227.76
3005	2	4138.09	13.41	23.28	8003	2		97.88	240.81
3202	1	5037.51	8.28	27.74	8218	1		52.80	282.19
3401	2	6088.73	17.23	32.04	8411	2		117.90	307.10
3603	2	7314.72	19.18	36.14	8601	2		127.33	333.69
3802	1	8296.18	11.54	43.38	8802	1		65.49	354.22
4002	1	9513.86	12.39	47.13	9006	2		145.43	376.45
4211	2	11348.90	27.27	55.49	9202	1		72.45	397.68
4401	2	13025.20	31.61	61.87	9396	1		76.56	426.80
4602	1	15209.50	18.78	74.03	9603	2		169.51	460.41
4806	2	16138.80	37.40	77.60	9802	1		83.83	476.15
5002	1	17545.40	20.93	84.78	9998	2		183.65	562.80

Table 8. Running times for test series 1

W. GEISELMANN, *Algebraische Algorithmenentwicklung am Beispiel der Arithmetik in endlichen Körpern.* Dissertation, Universität Karlsruhe, Aachen, 1994.

T. ITOH AND S. TSUJII, A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases. *Information and Computation* **78** (1988), 171–177.

D. JUNGNICKEL, *Finite Fields: Structure and Arithmetics.* BI Wissenschaftsverlag, Mannheim, 1993.

A. KARATSUBA AND Y. OFMAN, Умножение многозначных чисел на автоматах. *Dokl. Akad. Nauk USSR* **145** (1962), 293–294. Multiplication of multidigit numbers on automata, Soviet Physics–Doklady **7** (1963), 595-596.

D. E. KNUTH, *The Art of Computer Programming, Vol.2, Seminumerical Algorithms.* Addison-Wesley, Reading MA, 2 edition, 1981.

D. H. LEHMER, Euclid's algorithm for large numbers. *American Mathematical Monthly* **45** (1938), 227–233.

R. LIDL AND H. NIEDERREITER, *Finite Fields.* Encyclopedia of Mathematics and its Applications **20**. Addison-Wesley, Reading MA, 1983.

M. LOTHAIRE, *Combinatorics on Words.* Addison–Wesley Reading, MA, 1983.

n	k	onb t/sec	ggp t/sec	shoup t/sec
1034	2	205.36	1.63	1.67
2141	2	1595.74	7.28	9.47
4098	1	10401.90	14.5	51.98
8325	2	78019.00	127.76	302.86
16679	2		565.89	1759.61
23903	2		1064.7	4489.31
32075	2		1856.83	7545.09
43371	2		3593.04	15530.10
51251	2		4990.81	22039.70
61709	2		6973.74	34297.50

Table 9. Running times for test series 2

Alfred J. Menezes, Ian F. Blake, XuHong Gao, Ronald C. Mullin, Scott A. Vanstone, and Tomik Yaghoobian, *Applications of finite fields.* Kluwer Academic Publishers, Norwell MA, 1993.

R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, Optimal normal bases in GF(p^n). *Discrete Applied Math.* **22** (1989), 149–161.

A. Odlyzko, Discrete logarithms and their cryptographic significance. In *Advances in Cryptology, Proceedings of Eurocrypt 1984.* Springer-Verlag, 1985, 224–314.

R. L. Rivest, A. Shamir, and L. M. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* **21** (1978), 120–126.

P. de Rooij, Efficient exponentiation using precomputation and vector addition chains. In *Advances in cryptology: Proceedings EUROCRYPT '94*, ed. A. DeSantis, Lecture notes in computer science **950**, Berlin, 1995, Springer, 389–399.

A. Schönhage, Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica* **1** (1971), 139–144.

A. Schönhage, A lower bound for the length of addition chains. *Theor. Computer Science* **1** (1975), 1–12.

A. Schönhage, Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Inf.* **7** (1977), 395–398.

A. Schönhage and V. Strassen, Schnelle Multiplikation großer Zahlen. *Computing* **7** (1971), 281–292.

V. Shoup, Exponentiation in $GF(2^n)$ using fewer polynomial multiplications. Preprint, 1994.

D. R. Stinson, Some observations on parallel algorithms for fast exponentiation in $GF(2^n)$. *SIAM J. Comput.* **19** (1990), 711–717.

V. Strassen, Gaussian elimination is not optimal. *Numer. Mathematik* **13** (1969), 354–356.

V. Strassen, The computational complexity of continued fractions. *SIAM J. Comput.* **12** (1983), 1–27.

B. P. Tunstall, *Synthesis of noiseless compression codes.* Ph.D. dissertation, Georgia Inst. Technol., 1968.

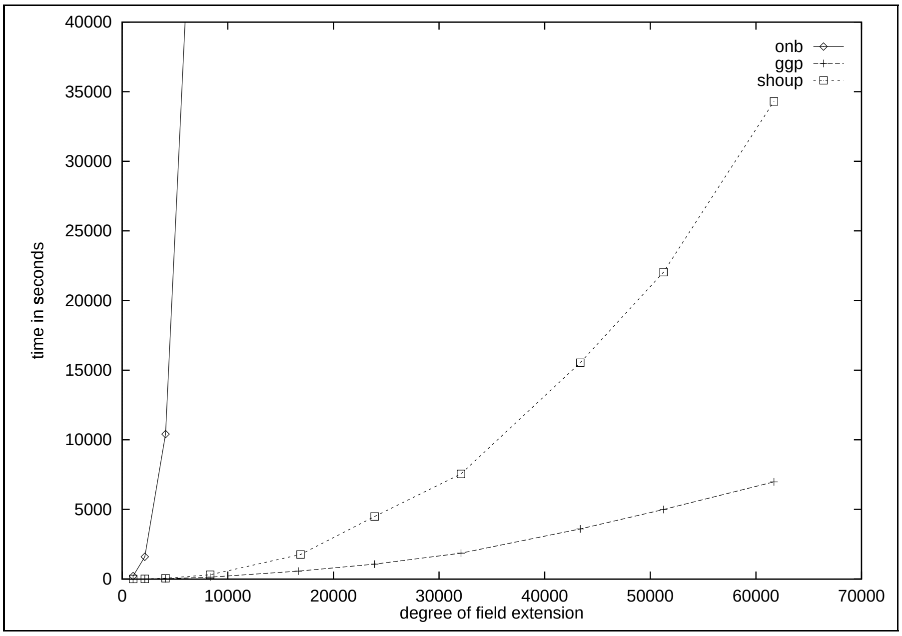

Fig. 3. Comparison of the three exponentiation algorithms for $n \approx 2^i$, $10 \le i \le 16$ and k as in Table 9.

A. Wassermann, Zur Arithmetik in endlichen Körpern. *Bayreuther Math. Schriften* **44** (1993), 147–251.

Y. Yacobi, Exponentiating faster with addition chains. In *Advances in cryptology: Proceedings EUROCRYPT '90*, ed. I. Damgard, Lecture notes in computer science **473**, Berlin, 1991, Springer, 222–229.

J. Ziv and A. Lempel, Compression of individual sequences via variable-rate coding. *IEEE Trans. Inform. Theory* **IT-24**(5) (1978), 530–536.

This article was processed using the LaTeX macro package with LLNCS style