A Generalization of Addition Chains and Fast Inversions in Binary Fields

Kimmo Järvinen, Vassil Dimitrov, and Reza Azarderakhsh

Abstract—In this paper, we study a generalization of addition chains where k previous values are summed together on each step instead of only two values as in traditional addition chains. Such chains are called k-chains and we show that they have applications in finding efficient parallelizations in problems that are known to be difficult to parallelize. In particular, 3-chains improve computations of inversions in finite fields using hybrid-double multipliers. Recently, it was shown that this operation can be efficiently computed using a ternary algorithm but we show that 3-chains provide a significantly more efficient solution.

Index Terms—Addition chain, finite field, binary extension field, normal basis, hybrid-double multiplier, inversion, exponentiation, pseudo-random number generation, cryptography, codes, ASIC

1 Introduction

An addition chain is a sequence of integers from one to n such that each value is a sum of two previous values. Addition chains have many applications in the field of computing. Several essential computational tasks required, for example, in public-key cryptosystems and computational number theory can be modelled as addition chains including exponentiations, scalar multiplications on elliptic curves, primitivity tests, and inversions in finite fields. For instance, the optimal addition-chain method computes an exponentiation with minimal number of multiplications. As a consequence of this importance, additions chains have been thoroughly studied for decades. The reader is referred, e.g., to [1] for a thorough survey on addition chains.

Addition chains can be extended or generalized into many directions and several proposals are available in the literature including addition-subtraction chains [2], [3], addition-multiplication chains [4], addition sequences [5], q-addition chains [6], [7], [8], and Lucas chains [9]. These generalizations typically offer improvements over general addition chains in the context of certain applications by exploiting features specific to those applications; e.g., addition-subtraction chains offer significant speedups compared to addition chains for elliptic curve cryptography because additions and subtractions are equally fast in additive groups formed by points on elliptic curves but they are useless when applied to multiplicative groups where divisions are significantly more expensive than multiplications.

In this paper, we study a generalization of addition chains where each value in the sequence is a sum of at most k previous values from the sequence. We call these sequences k-chains. To the best of our knowledge, k-chains have been discussed previously only by Hayata and Wagatuma in an unpublished manuscript [10] from 2006. They generalized many of the properties of addition chains to the properties of k-chains including an algorithm for searching the shortest possible k-chains efficiently. However, they did not provide any applications for k-chains.

In this paper, we deepen the knowledge on k-chains in a number of ways

- We provide a new algorithm for finding short (but not necessarily optimal) k-chains for an arbitrary n;
- We tighten the bounds for the length of k-chains and provide results on the relations of addition chains and k-chains with different values of k;
- We introduce a new parallel version of k-chains that allows efficient parallelization of problems utilizing k-chains;
- Most importantly, we propose applications for k-chains. We show that (parallel) k-chains lead to the fastest known algorithms and implementations for computing inversions in $GF(2^m)$ and, hence, k-chains have applications in cryptography and codes that require fast finite field arithmetic. Certain other applications are also shortly surveyed; and
- We also present practical contributions by showing how these findings can be utilized in designing an efficient hardware architecture for inversions in a binary field $GF(2^m)$. We provide implementation results on 65-nm complementary metal-oxide-semiconductor (CMOS) application-specific integrated circuit (ASIC) for inversions in the binary fields recommended by National Institute of Standards and Technology (NIST) in [11].

- *K. Järvinen is with the Department of Information and Computer Science, Aalto University, Konemiehentie 2, FI-02150 Espoo, Finland. E-mail: kimmo.jarvinen@aalto.fi.*
- *V. Dimitrov is with the Department of Electrical and Computer Engineering, University of Calgary, 2500 University Dr. NW, Calgary AB T2N 1N4, Canada. E-mail: vdimitro@ucalgary.ca.*
- *R. Azarderakhsh is with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, New York, NY. E-mail: rxaeec@rit.edu.*

Manuscript received 27 Mar. 2014; revised 29 Oct. 2014; accepted 14 Nov. 2014. Date of publication 25 Nov. 2014; date of current version 12 Aug. 2015. Recommended for acceptance by A. Nannarelli.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2014.2375182

The paper is structured as follows: Section 2 reviews basic definitions and properties of addition chains. In Section 3, we discuss k-chains and their properties by reviewing certain relevant aspects of [10], by deriving algorithms for finding k-chains, and by providing new insight about the properties of k-chains. Section 4 presents the link between k-chains and inversions in $GF(2^m)$ and introduces new faster inversion algorithms. We introduce parallel k-chains and show their applicability for inversions in Section 5. We present proof-of-concept implementations and their results in Section 6. We discuss certain other applications where k-chains can be useful in Section 7 and, finally, we conclude the paper in Section 8.

2 Addition Chains

We begin with several formal definitions that are relevant for this paper. The reader is referred, e.g., to [1] for more background about addition chains.

Definition 1 (Addition chain). *An addition chain for a positive integer n is a sequence of natural numbers v*

$$v = (v(0), \dots, v(s)), \tag{1}$$

with $v(0) = 1$, $v(s) = n$, and $v(i) = v(i_0) + v(i_1)$ for all $0 < i \le s$ so that $0 \le i_0, i_1 < i$; i.e., each term in v is the sum of two previous terms. The number s is called the length of the addition chain.

Definition 2 (Optimality). *An addition chain v for a positive integer n is called optimal if its length s is the smallest among all possible addition chains for n.*

Optimal addition chains for n can be found, e.g., with an efficient backtracking algorithm that was introduced by Thurber in [12].

Definition 3 (Brauer-type addition chain [13]). *An addition chain is a Brauer-type addition chain if the previous value is always used for the next one; i.e., $i_0 = i - 1$ or $i_1 = i - 1$ for all $0 < i \le s$. Respectively, an addition chain is a non-Brauer-type addition chain if there exists $v(i) = v(i_0) + v(i_1)$ for which $i_0 \ne i - 1$ and $i_1 \ne i - 1$.*

Proposition 1 (Bounds for the length of an optimal addition chain [14]). *The length of an optimal addition chain for an integer n satisfies the following bounds:*

$$\log_2 n + \log_2 h_2(n) - 2.13 \le s \le \lfloor \log_2 n \rfloor + h_2(n) - 1, \tag{2}$$

where $h_2(n)$ is the Hamming weigth of n; i.e., the number of ones in the binary representation of n.

The lower bound of Proposition 1 is from [14] and the upper bound follows trivially from the binary method that is explained next. The left-to-right binary method is an easy way to find a relatively short Brauer-type addition chain for an arbitrary n. The binary expansion of n is $n = \sum_{j=0}^{\lfloor \log_2 n \rfloor} n_j 2^j$. Then, starting from $n_{\lfloor \log_2 n \rfloor - 1}$ down to n_0, the addition chain is built so that, if $n_j = 0$, then $v(i) = 2v(i-1)$ is added to chain. If $n_j = 1$, then one adds two terms: $v(i) = 2v(i-1)$ and $v(i+1) = v(i) + v(0)$.

Example 1. Let $n = 19 = \langle 1, 0, 0, 1, 1 \rangle$. Then, $v(1) = 2v(0) = 2$ because $n_{4-1} = n_3 = 0$, $v(2) = 2v(1) = 4$ because $n_2 = 0$. Because $n_1 = 1$, we add $v(3) = 2v(2) = 8$ and $v(4) = v(3) + v(0) = 9$. Similarly, we have $v(5) = 2v(4) = 18$ and $v(6) = v(5) + v(0) = 19$. Hence, the binary addition chain for 19 is $(1, 2, 4, 8, 9, 18, 19)$. It is one of the 33 optimal addition chains[1] for 19.

3 k-Chains and Their Properties

Definition 4 (k-chain). *A k-chain for a positive integer n is a sequence of natural numbers v*

$$v = (v(0), \dots, v(s))_k, \tag{3}$$

with $v(0) = 1$, $v(s) = n$, and $v(i) = \sum_{h=0}^{k-1} v(i_h)$ for all $0 < i \le s$ and $-1 \le i_h < i$ with $v(-1) = 0$; i.e., each term in v is the sum of k previous terms (including zero). The number s is called the length of the k-chain.

There are many connections between addition chains and k-chains. Many definitions and results for addition chains have straightforward generalizations to k-chains [10]. These include optimality, Brauer-type, etc. The most obvious connections are the following:

Remark 1. 2-chain is a synonym for addition chain.

Remark 2. A 2-chain (addition chain) can be derived from each k-chain with $k > 2$. Hence, a k-chain with $k > 2$ can be seen as a compression of an addition chain (2-chain). These addition chains are not necessarily optimal.

Example 2. The only optimal 3-chain for 7 is $(1, 3, 7)_3$, where $3 = 1 + 1 + 1$ and $7 = 1 + 3 + 3$. The corresponding 2-chains (addition chains) are $(1, 2, 3, 4, 7)_2$ and $(1, 2, 3, 6, 7)_2$ which are optimal.

3.1 The Length of an Optimal k-Chain

Optimal k-chains can be found efficiently with the backtracking algorithm originally proposed for addition chains in [12] and optimized for k-chains in [10]. Fig. 1 plots the lengths of optimal k-chains for $2 \le k \le 5$ and $1 \le n \le 256$. This clearly shows the reduction of lengths when k increases. Hence, if an application using addition chains can exploit k-chains instead of addition chains, then results (typically, computation latency) are likely to improve significantly. In the following, we discuss certain aspects related to the lengths of k-chains.

Proposition 2. *Let v_0 be an optimal k_0-chain of length s_0 for an integer n. If $k_1 > k_0$, the length s_1 of an optimal k_1-chain v_1 for n satisfies $s_1 \le s_0$.*

Proof. An optimal k_1-chain cannot be longer than an optimal k_0-chain because each k_0-chain is also a k_1-chain. The k_1-chain v_1 can be constructed by setting $v_1(i) = \sum_{h=0}^{k_0-1} v_0(j_h) + \sum_{h'=k_0}^{k_1-1} v_1(-1)$; i.e., by adding $k_1 - k_0$ zeros to all $v_0(i)$. □

1. The optimality can be checked, e.g., from http://wwwhomes.uni-bielefeld.de/achim/addition_chain.html (Oct. 27, 2014).

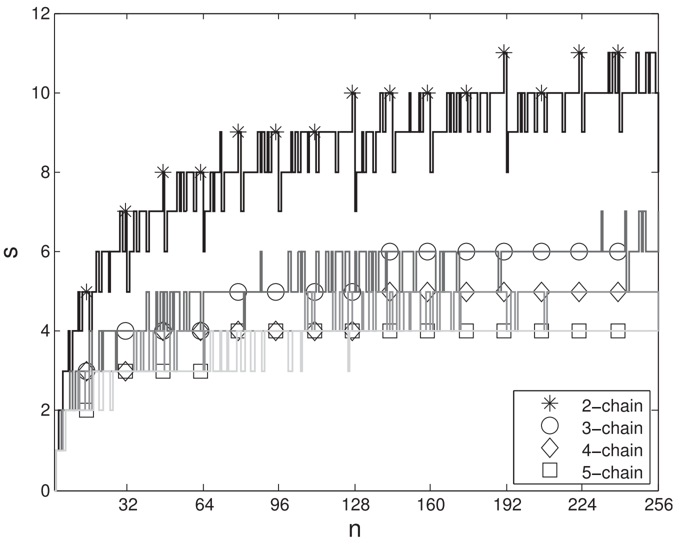


Fig. 1. The lengths of optimal k-chains with $2 \leq k \leq 5$ for positive integers $1 \leq n \leq 256$.

Consequently, an optimal k_0-chain is not necessarily an optimal k_1-chain but if an optimal k_1-chain is also a k_0-chain, then it is always an optimal k_0-chain.

Example 3. There is only one optimal 2-chain for the number 8: $(1,2,4,8)_2$. Its length is 3. There are eight optimal 3-chains: $(1,2,3,8)_3$, $(1,2,4,8)_3$, $(1,2,5,8)_3$, $(1,2,6,8)_3$, $(1,3,4,8)_3$, $(1,3,5,8)_3$, $(1,3,6,8)_3$, and $(1,3,7,8)_3$. Their length is also 3. The optimal 2-chain is also an optimal 3-chain. The length reduces to 2 for the three optimal 4-chains: $(1,2,8)_4$, $(1,3,8)_4$, and $(1,4,8)_4$.

Proposition 3 (Bounds for s [10]). *The length s of an optimal k-chain for a positive integer n satisfies the following bounds*

$$\lceil \log_k n \rceil \leq s \leq \lfloor \log_k n \rfloor + h_k(n) - \delta_k(n), \tag{4}$$

where $h_k(n)$ is the k-ary Hamming weight, the number of nonzeros in the k-ary representation of n, and $\delta_k(n) = 1$ if $n_{\lfloor \log_k n \rfloor} = \lfloor n/k^{\lfloor \log_k n \rfloor} \rfloor = 1$; otherwise, $\delta_k(n) = 0$.

The lower bound follows trivially from the fact that the largest value reachable with h steps is k^h. The upper bound comes from a k-ary generalization of the binary method [10]. Although [10] did not explicitly present the k-ary algorithm that was used for deriving the upper bound of (4), the algorithm given in Algorithm 1 is equivalent with that algorithm in terms of the length of the k-chains. Algorithm 1 constructs different k-chains when n_i are processed in different orders in the for loop, but they all have the same length.

Algorithm 1. A k-ary algorithm for finding a k-chain for an integer n

```
Input: $k$, $n = \sum_{i=0}^{\ell} n_i k^i = \langle n_0, n_1, \dots, n_\ell \rangle_k$ where $\ell = \lfloor \log_k n \rfloor$
Output: A $k$-chain $v$ for $n$
1  $v(i) \leftarrow k^i$ for $0 \leq i \leq \ell$
2  $j \leftarrow 1$;
3  for all $x_i > 0$ in $x = \{n_0, n_1, \dots, n_{\ell-1}, n_\ell - 1\}$ do
4      $v(\ell + j) \leftarrow v(\ell + j - 1) + x_i v(i)$
5      $j \leftarrow j + 1$
6  return $v$
```

While addition chains derived with all variations of the binary method (e.g., left-to-right or right-to-left) have the same length, it is possible to derive different variations of the k-ary algorithm so that the lengths of the resulting k-chains differ. In the following, we propose a variation which constructs k-chains that are at least as short as the ones provided by Algorithm 1 and the variation considered in [10]. As a consequence, we can also use this new algorithm for improving the upper bound of Proposition 3.

3.2 An Algorithm for Finding Short k-Chains

Even though backtracking algorithm allows finding optimal k-chains relatively efficiently, finding optimal k-chains for an arbitrary large n becomes impractical, especially, if it needs to be done on-the-fly. Hence, we propose a simple procedure for finding short, but not necessarily optimal, k-chains. It is another k-ary generalization of the well-known binary method and it is captured in Algorithm 2. The algorithm constructs a temporary vector t which contains the indexes of $v(i) = k^i$ which are to be summed. Then, t is processed so that k values are always summed in calculating each $v(i)$ (excluding the last one if the number of summands is not a multiple of k).

Algorithm 2. Another k-ary algorithm for finding a k-chain for a positive integer n

```
Input: $k$, $n = \sum_{i=0}^{\ell} n_i k^i = \langle n_0, n_1, \dots, n_\ell \rangle_k$ where $\ell = \lfloor \log_k n \rfloor$
Output: A $k$-chain $v$ for $n$
1  $v(i) \leftarrow k^i$ for $0 \leq i \leq \ell$
2  $j \leftarrow 0$
3  for all $x_i > 0$ in $x = \{n_0, n_1, \dots, n_{\ell-1}, n_\ell - 1\}$ do
4      $(t_j, t_{j+1}, \dots, t_{j+x_i-1}) \leftarrow (i, i, \dots, i)$
5      $j \leftarrow j + x_i$
6  $(t_j, t_{j+1}, \dots, t_{\lceil j/(k-1) \rceil (k-1) - 1}) \leftarrow (-1, -1, \dots, -1)$
7  for $i = 0$ to $\lfloor j/(k-1) \rfloor$ do
8      $v(\ell + i + 1) \leftarrow v(\ell + i) + v(t_{i(k-1)}) + v(t_{i(k-1)+1}) + \cdots +$
           $v(t_{i(k-1)+k-2})$
9  return $v$
```

Example 4. Let $n = 426$ and $k = 5$. The quinary representation of n is $426 = \langle 3, 2, 0, 1 \rangle_5$. Hence, we first calculate the powers up to $v(3) = 5^3 = 125$. Then, it remains to compute $n = 3 \times 125 + 2 \times 25 + 1$ which requires $\lceil (3 + 2 + 1 - 1)/(5 - 1) \rceil = 2$ extra steps. In Algorithm 2, this results in $t = (0, 2, 2, 3, 3, -1, -1, -1)$ which gives

$$\begin{aligned} v(4) &= v(3) + v(0) + v(2) + v(2) + v(3) \\ &= 125 + 1 + 2 \times 25 + 125 = 301 \quad \text{and} \\ v(5) &= v(4) + v(3) + v(-1) + v(-1) + v(-1) \\ &= 301 + 125 + 3 \times 0 = 426. \end{aligned}$$

Hence, we get the following 5-chain: $(1, 5, 25, 125, 301, 426)_5$ and $s = 5$. By chance, this 5-chain happens to be one of the 1,627 optimal 5-chains for 426.

Proposition 4. *The length s of a k-chain for a positive integer n obtained with Algorithm 2 is*

$$s = \lfloor \log_k n \rfloor + \lceil (d_k(n) - 1)/(k - 1) \rceil, \tag{5}$$

where $d_k(n) = \sum_{i=0}^{\lfloor \log_k n \rfloor} n_i$, the sum of digits of the k-ary representation of $n = \sum_{i=0}^{\lfloor \log_k n \rfloor} n_i k^i$.

Proof. The first part of the length, $\lfloor \log_k n \rfloor$, follows directly from computing the powers of k up to $k^{\lfloor \log_k n \rfloor}$. After this, the powers of k are summed according to the digits of the k-ary representation. The sum $n = \sum_{i=0}^{\lfloor \log_k n \rfloor} n_i k^i$ is a sum of $d_k(n)$ powers of k (each k^i appears n_i times in the sum). It can be computed with a k-chain with length $\lceil (d_k(n) - 1)/(k - 1) \rceil$ because all k^i are already available which results in the bound of (5). □

Although Algorithms 1 and 2 are both k-ary generalizations of the binary algorithm, they are, indeed, different algorithms and they return different k-chains. The difference is that Algorithm 1 processes each digit n_i independently whereas Algorithm 2 groups processing of different digits whenever possible. If $k = 2$, these algorithms are equivalent. The ternary Itoh-Tsujii (TIT) algorithm used in [15] is essentially the same algorithm as Algorithm 1 applied to inversions in $GF(2^m)$. Indeed, Algorithm 2 improves upon them as shown in the following proposition:

Proposition 5. *Algorithm 2 produces a k-chain which is at least as short as a k-chain from Algorithm 1 for all positive integers n.*

Proof. The length of a k-chain from Algorithm 2 is given by (5) in Proposition 4 and the length of a k-chain from Algorithm 1 is given by the upper bound of (4) in Proposition 3. Hence, we need to show that the following inequality holds for all n:

$$\lfloor \log_k n \rfloor + \left\lceil \frac{d_k(n) - 1}{k - 1} \right\rceil \leq \lfloor \log_k n \rfloor + h_k(n) - \delta_k(n). \quad (6)$$

The following holds for $d_k(n) = \sum_{i=0}^{\lfloor \log_k n \rfloor} n_i$ because $0 \leq n_i \leq k - 1$:

$$d_k(n) \leq (k - 1)(h_k(n) - 1) + n_{\lfloor \log_k n \rfloor}.$$

Hence,

$$\begin{aligned} \left\lceil \frac{d_k(n) - 1}{k - 1} \right\rceil &\leq \left\lceil \frac{(k - 1)(h_k(n) - 1) + n_{\lfloor \log_k n \rfloor} - 1}{k - 1} \right\rceil \\ &= h_k(n) - 1 + \left\lceil \frac{n_{\lfloor \log_k n \rfloor} - 1}{k - 1} \right\rceil \\ &= \begin{cases} h_k(n) - 1 & \text{if } n_{\lfloor \log_k n \rfloor} = 1 \\ h_k(n) & \text{if } n_{\lfloor \log_k n \rfloor} > 1 \end{cases} \\ &= h_k(n) - \delta_k(n). \end{aligned}$$

□

Example 5. Let $n = 607$ and $k = 3$. The ternary representation of 607 is $\langle 2, 1, 1, 1, 1, 1 \rangle_3$ from which we see that $\lfloor \log_3 (607) \rfloor = 5$, $h_3(607) = 6$, $\delta_3(607) = 0$, and $d_3(607) = 7$. Hence, the length from (5) (Algorithm 2) is $\lfloor \log_3 (607) \rfloor + \lceil (d_3(607) - 1)/(3 - 1) \rceil = 5 + 3 = 8$. The upper bound of (4) (Algorithm 1) is $\lfloor \log_3(607) \rfloor + h_3(607) - \delta_3(607) = 5 + 6 - 0 = 11$, which is overly pessimistic. For the optimal 3-chains for 607, $s = 7$.

As a consequence of Propositions 4 and 5, we have a new upper bound for the lengths of optimal k-chains and we can revise Proposition 3 as follows:

Corollary 1 (New bounds for s). *The length s of an optimal k-chain for a positive integer n satisfies the following bounds:*

$$\lceil \log_k n \rceil \leq s \leq \lfloor \log_k n \rfloor + \lceil (d_k(n) - 1)/(k - 1) \rceil. \quad (7)$$

Although it is easy to see that k-chains with $k > 2$ obtained with Algorithm 2 are usually shorter than binary addition chains or even optimal addition chains, there could be cases where they are longer. For instance, a binary addition chain could be shorter if the binary representation is sparse and the k-ary representation is dense and contains many large digits. The following proposition sheds light on this issue.

Proposition 6. *If $k \geq 4$, then a k-chain given by Algorithm 2 is at least as short as an optimal addition chain for all positive integers n.*

Proof The length of an optimal addition chain is at least $\lceil \log_2 n \rceil$ because the largest value reachable with h steps is 2^h. Hence, we need to show that the following inequality holds for all n:

$$\lfloor \log_k n \rfloor + \left\lceil \frac{d_k(n) - 1}{k - 1} \right\rceil \leq \lceil \log_2 n \rceil. \quad (8)$$

The right-hand side of (8) is bounded from below as follows:

$$\lceil \log_2 n \rceil \geq \log_2 n. \quad (9)$$

Respectively, the left-hand side of (8) is bounded from above as follows:

$$\lfloor \log_k n \rfloor + \left\lceil \frac{d_k(n) - 1}{k - 1} \right\rceil < 2 \log_k(2) \log_2(n). \quad (10)$$

Consequently, the left-hand side is smaller or equal to the right-hand side if

$$2 \log_k(2) \leq 1. \quad (11)$$

Hence, (8) always holds when $k \geq 4$. □

Many interesting applications of k-chains use $k = 3$ that is not covered by Proposition 6. Indeed, even binary addition chains (not necessarily optimal) are shorter than the 3-chains given by Algorithm 2 in some extremely rare occasions. For example, all such cases in the interval $n \in [1, 2^{24} = 16{,}777{,}216]$ are 32,768, 32,776, 32,800, 32,804, 65,600, 98,304, 98,306, 294,920, 526,336, 1,052,672, 2,097,152, 2,105,344, 2,105,348, 6,291,968, and 13,631,488. These cases are slightly more common for Algorithm 1: there are in total 831 such values for $n \in [1, 2^{24}]$.

4 3-Chains and Inversions in $GF(2^m)$

Fermat's Little Theorem provides means for computing inversions in finite fields via exponentiations with a constant exponent. In $GF(2^m)$, the inverse element of $A \neq 0 \in GF(2^m)$ is received through the following exponentiation:

$$A^{-1} = A^{2^m-2} = B^{2^{m-1}-1}, \text{ where } B = A^2. \qquad (12)$$

Because $2^{m-1} - 1 = 1 + 2 + 2^2 + \cdots + 2^{m-2}$, the traditional binary method for exponentiations performs badly and requires $m-2$ multiplications and $m-1$ squarings [16]. In [17], Itoh and Tsujii (IT) proposed a clever technique that reduces the number of multiplications to $\lfloor \log_2(m-1) \rfloor + h_2(m-1) - 1$. In [18], Takagi et al. presented that improvements can be achieved in cases where $m-1$ has a 'nice' factorization. Recently, Dimitrov and Järvinen [19] proposed algorithms based on double-base and triple-base representations of $m-1$ that slightly improve the number of multiplications and provide certain implementation specific advantages over IT. It is also known that an inversion algorithm for $GF(2^m)$ can be derived from any addition chain for $m-1$ and using an optimal addition chain leads to an optimal number of multiplications.

A hybrid-double (HD) multiplier [20] is a multiplier that computes the product of three elements, $\alpha \times \beta \times \Gamma$, where $\alpha, \beta, \Gamma \in GF(2^m)$, with a latency comparable to normal multipliers computing the product of two elements $\alpha \times \beta$. This is achieved by interleaving parallel-in-serial-out (PISO) and serial-in-parallel-out (SIPO) digit-serial multipliers so that the PISO multiplier computes $\Delta = \alpha \times \beta$ and the SIPO multiplier begins computing $\Gamma \times \Delta = \alpha \times \beta \times \Gamma$ immediately when the PISO multiplier starts returning the first digits of the result Δ [20]. None of the above mentioned inversion algorithms allows efficient exploitation of the potential offered by HD multipliers in computing inversions. For instance, using double multiplications instead of normal multiplications in the case of IT improves the latency at most by a latency of one multiplication [15].

In [15], Azarderakhsh et al. solved this problem and showed how inversions in $GF(2^m)$ can be computed faster with HD multipliers. They used a variation of the IT inversion algorithm where they utilized the ternary representation of $m-1$ instead of the typical binary representation. This variation called TIT is essentially Algorithm 1 applied to inversions in binary fields. Using TIT allowed them to utilize the potential of the HD multipliers; e.g., TIT allows computing an inversion in $GF(2^{163})$, which is the smallest binary field included in [11], with only five double multiplications when IT would require nine (the number of normal multiplications would also be nine in this case). Although TIT typically improves latency significantly compared to the traditional IT, the results of TIT are suboptimal and can, in fact, perform even worse than IT in some rare cases; e.g., IT requires nine double multiplications for computing an inversion in $GF(2^{393})$ whereas TIT requires 10.

In the following, we show how we can utilize k-chains and the findings from previous sections in inversions in $GF(2^m)$. The essential link between k-chains and algorithms for computing inversions in $GF(2^m)$ is described in the following proposition:

Proposition 7. *Let $v(i) = \sum_{j=0}^{k-1} v(i_j)$ and $V_{i_j} = B^{2^{v(i_j)}-1}$. Then,*

$$V_i = V_{i_0} \prod_{h=1}^{k-1} V_{i_h}^{2^{\sum_{j=0}^{h-1} v(i_j)}} = B^{2^{v(i)}-1}. \qquad (13)$$

Proof.

$$\begin{aligned} V_i &= V_{i_0} \prod_{h=1}^{k-1} V_{i_h}^{2^{\sum_{j=0}^{h-1} v(i_j)}} \\ &= B^{2^{v(i_0)}-1} \prod_{h=1}^{k-1} \left(B^{2^{v(i_h)}-1}\right)^{2^{\sum_{j=0}^{h-1} v(i_j)}} \\ &= B^{2^{v(i_0)}-1} \prod_{h=1}^{k-1} B^{2^{\sum_{j=0}^{h-1} v(i_j)}\left(2^{v(i_h)}-1\right)} \\ &= B^{2^{v(i_0)}-1} \prod_{h=1}^{k-1} B^{2^{\sum_{j=0}^{h} v(i_j)} - 2^{\sum_{j=0}^{h-1} v(i_j)}} \\ &= B^{2^{v(i_0)}-1} B^{\sum_{h=1}^{k-1} \left(2^{\sum_{j=0}^{h} v(i_j)} - 2^{\sum_{j=0}^{h-1} v(i_j)}\right)} \\ &= B^{2^{v(i_0)}-1} B^{2^{\sum_{j=0}^{k-1} v(i_j)} - 2^{v(i_0)}} \\ &= B^{2^{\sum_{j=0}^{k-1} v(i_j)}-1} = B^{2^{v(i)}-1}. \end{aligned}$$

□

When $k = 3$, Proposition 7 gets the form shown in the following corollary:

Corollary 2. *Let $v(i) = v(i_0) + v(i_1) + v(i_2)$ and $V_{i_j} = B^{2^{v(i_j)}-1}$. Then,*

$$V_i = V_{i_0} \times V_{i_1}^{2^{v(i_0)}} \times V_{i_2}^{2^{v(i_0)+v(i_1)}} = B^{2^{v(i)}-1} \quad . \qquad (14)$$

We now have a direct way to use k-chains for computing inversions in $GF(2^m)$ by computing the V_i iteratively. Especially, Corollary 2 gives a straightforward method to construct inversion algorithms for $GF(2^m)$ from 3-chains for $m-1$. These algorithms require s double multiplications and, hence, selecting a short 3-chain (preferably optimal if possible) leads to an inversion algorithm that efficiently exploits the potential of HD multipliers.

Example 6. Using Fermat's Little Theorem, we have that inversions in $GF(2^8)$ are given by $A^{-1} = A^{2^8-2} = A^{2(2^7-1)} = B^{2^7-1}$, where $B = A^2$, for all $A \neq 0 \in GF(2^8)$. The only optimal 3-chain for the integer 7 is $(1, 3, 7)_3$. Set $V_0 = B$. Because $v(1) = v(0) + v(0) + v(0) = 1+ 1 + 1 = 3$, Corollary 2 gives that $V_1 = V_0 \times V_0^2 \times V_0^{2^{1+1}} = B^{2^3-1}$. Because $v(2) = v(0) + v(1) + v(1) = 1 + 3 + 3 = 7$, we get that $V_2 = V_0 \times V_1^2 \times V_1^{2^{1+3}} = B^{2^7-1} = A^{-1}$. Hence, inversions in $GF(2^8)$ can be computed with two double multiplications. Notice that both IT and TIT require three double multiplications. Inversions in $GF(2^8)$ are used in various applications including, e.g., the S-box of the Advanced Encryption Standard (AES) [21].

Example 7. Both IT and TIT require nine double multiplications for inversion in $GF(2^{233})$. Using any of the 3,603 optimal 3-chains for 232 allows computing an inversion with just seven double multiplications. Consider the arbitrarily chosen optimal 3-chain $(1, 2, 4, 6, 18, 38, 114, 232)_3$, where $2 = 1 + 1 + 0$, $4 = 2 + 2 + 0$, $6 = 2 + 4 + 0$, $18 = 6 + 6 + 6$, $38 = 2 + 18 + 18$, $114 = 38 + 38 + 38$, and $232 = 4 + 114 + 114$. Applying Corollary 2 results in the inversion algorithm given in Algorithm 3.

Algorithm 3. Inversion in $GF(2^{233})$ with an optimal 3-chain and double multiplications

```
    Input: $A \in GF(2^{233}), A \neq 0$
    Output: $B = A^{-1}$
1   $B \leftarrow A \gg 1$
2   $B, R_1 \leftarrow B \times (B \gg 1) \times 1$
3   $B, R_2 \leftarrow B \times (B \gg 2) \times 1$
4   $B \leftarrow R_1 \times (B \gg 2) \times 1$
5   $B \leftarrow B \times (B \gg 6) \times (B \gg 12)$
6   $B \leftarrow R_1 \times (B \gg 2) \times (B \gg 20)$
7   $B \leftarrow B \times (B \gg 38) \times (B \gg 76)$
8   $B \leftarrow R_2 \times (B \gg 4) \times (B \gg 118)$
9   return $B = A^{-1}$
```

Although all the techniques based on the use of k-chains can be used regardless of the basis of the field, all inversion algorithms in this paper and the inverter architecture in Section 6 use a normal basis notation for the squarings which are simply rotations of the bit vector: $B^{2^e} = B \gg e$. We use this notation because the HD multipliers introduced in [20] use Gaussian normal bases (GNB). HD multipliers can be constructed also for polynomial bases by employing the PISO multiplier structure introduced by Reyhani-Masoleh in [22] together with any of the multiple SIPO multipliers available in the literature. Hence, all findings of this paper can be generalized for polynomial bases in a straightforward manner.

Fig. 2 shows the number of double multiplications required to compute inversions in $GF(2^m)$ with the IT, TIT, 3-chains from Algorithm 2, and optimal 3-chains. Notice that TIT equals the upper bound given by (4) for $m-1$ and Algorithm 2 reflects the new bound given by (7) for $m-1$. Table 1 highlights the costs for the five binary fields recommended by NIST in [11].

Out of the 256 cases considered in Fig. 2, IT is optimal only in 11 cases ($GF(2^{49})$ being the largest of them) and even TIT is optimal only in 57 cases. In particular, TIT provides optimal results only for two out of the five NIST fields: $GF(2^{163})$ and $GF(2^{409})$. The smallest field for which TIT produces sub-optimal results is $GF(2^8)$ which is discussed above in Example 6. Algorithm 2 performs a bit better and returns optimal results for 122 cases in the interval $1 \leq m \leq 256$ and for three NIST fields. Even for the two NIST fields, for which the results are sub-optimal, the results are better than those of TIT. The smallest sub-optimal case for Algorithm 2 is $GF(2^{15})$.

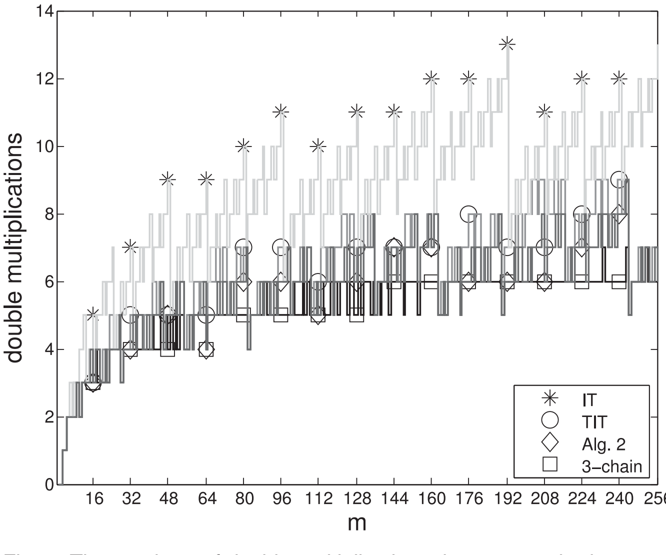


Fig. 2. The numbers of double multiplications that are required to compute inversions in $GF(2^m)$ with $1 \leq m \leq 256$ by using either IT, TIT, Algorithm 2, or 3-chain algorithms.

TABLE 1
The Number of Double Multiplications Required for Inversions Over NIST Binary Fields

Field	$GF(2^{163})$	$GF(2^{233})$	$GF(2^{283})$	$GF(2^{409})$	$GF(2^{571})$
IT	9	9	11	10	13
TIT/Algorithm 1	5	9	8	7	8
Algorithm 2	5	8	7	7	7
Optimal 3-chain	5	7	6	7	7

Conjecture 1. *If v is an optimal 3-chain for a positive integer $m-1$, then its length s is a strict lower bound for the number of double multiplications required for the exponentiation A^{2^m-2}.*

An algorithm requiring exactly s double multiplications can be constructed by using Proposition 7 and if Conjecture 1 holds, then an algorithm derived from an optimal 3-chain computes an inversion with the lowest possible number of double multiplications.

5 Parallel k-Chains and Inversions in $GF(2^m)$

Parallel computation of inversions in $GF(2^m)$ is known to be difficult. Traditional techniques, such as IT, are entirely sequential processes. In [7], Nöcker showed how a binary addition chain can be translated into a parallel version with a depth of $s = \lceil \log_2 n \rceil$. This enabled computing inversions in $GF(2^m)$ with a critical path of $\lceil \log_2(m-1) \rceil$ multiplications with two parallel multipliers [7]. Parallel k-chains presented below generalize Nöcker's idea for k-chains.

Definition 5 (Parallel k-chain). *A parallel k-chain for a positive integer n is the following array v of two sequences of natural number v_0 and v_1:*

$$v = \begin{pmatrix} v_0(0), & v_0(1), & \dots, & v_0(s_1) & \\ & v_1(1), & \dots, & v_1(s_2-1), & v_1(s_2) \end{pmatrix}_k, \quad (15)$$

where $v_0(i)$ and $v_1(i)$ are set by using the k-ary representation $n = \sum_{i=0}^{\lfloor \log_k n \rfloor} n_i k^i = \langle n_{\lfloor \log_k n \rfloor}, n_{\lfloor \log_k n \rfloor - 1}, \dots n_1, n_0 \rangle_k$ as follows:

- *$v_0(i) = k^i$ for $0 \leq i \leq \lfloor \log_k n \rfloor$*
- *If $n \neq k^{\lfloor \log_k n \rfloor}$, then $v_1(i) = v_1(i-1) + n_{i-1} v_0(i-1)$ for $1 \leq i \leq \lceil \log_k n \rceil$ and $v(0)^{(1)} = 0$.*

If $n = k^{\lfloor \log_k n \rfloor}$, then $v(s_1) = n$; otherwise, $v(s_2) = n$.

It follows directly from Definition 5 that the length s of a parallel k-chain is:

$$s = \lceil \log_k n \rceil. \quad (16)$$

TABLE 2
The Lengths of the Critical Paths (Multiplications) for the Techniques for Parallel Inversions Over the NIST Fields

Field	$GF(2^{163})$	$GF(2^{233})$	$GF(2^{283})$	$GF(2^{409})$	$GF(2^{571})$	Note
Itoh-Tsujii [17]	9	10	11	11	13	One multiplier
Opt. addition chain	9	10	11	10	12	One multiplier
Nöcker [7]	8	8	9	9	10	Two multipliers
Optimal 3-chain	5	7	6	7	7	One HD multiplier
Parallel 3-chain	5	5	6	6	6	Two HD multipliers

Example 8. Because $232 = \langle 2,2,1,2,1 \rangle_3$, an inversion in $GF(2^{233})$ can be computed with the following parallel 3-chain:

$$\begin{pmatrix} 1, & 3, & 9, & 27, & 81 & \\ & 1, & 7, & 16, & 70, & 232 \end{pmatrix}_3,$$

where $v_0(i) = 3^i$ for $0 \le i \le 4$ and $v_1(1) = v_0(0)$, $v_1(2) = v_1(1) + 2v_0(1)$, $v_1(3) = v_1(2) + v_0(2)$, $v_1(4) = v_1(3) + 2v_0(3)$, and $v_1(5) = v_1(4) + 2v_0(4)$. The length is $s = s_2 = 5$. This translates into the algorithm given in Algorithm 4 that computes inversions with a critical path of only five double multiplications.

Remark 3. A parallel k-chain for n improves the critical path if and only if the length of optimal k-chains for n is longer than $\lceil \log_k n \rceil$. In particular, if $d_k(n) \le k$, then it follows directly from Corollary 1 that optimal k-chains for n have the same length with the parallel k-chain and that one of such optimal k-chains can be obtained with Algorithm 2. Hence, using parallel k-chains for such n is guaranteed not to offer any improvements.

Algorithm 4. Inversion in $GF(2^{233})$ using two HD multipliers

```
   Input: $A \in GF(2^{233}), A \neq 0$
   Output: $T_2 = A^{-1}$
1    $T_1 \leftarrow A \gg 1$
2    $T_1 \leftarrow T_1 \times (T_1 \gg 1) \times (T_1 \gg 2) \mid T_2 \leftarrow T_1$
3    $T_1 \leftarrow T_1 \times (T_1 \gg 3) \times (T_1 \gg 6) \mid$
     $T_2 \leftarrow T_2 \times (T_1 \gg 1) \times (T_1 \gg 4)$
4    $T_1 \leftarrow T_1 \times (T_1 \gg 9) \times (T_1 \gg 18) \mid T_2 \leftarrow T_2 \times (T_1 \gg 7) \times 1$
5    $T_1 \leftarrow T_1 \times (T_1 \gg 27) \times (T_1 \gg 54) \mid$
     $T_2 \leftarrow T_2 \times (T_1 \gg 16) \times (T_1 \gg 43)$
6    $T_2 \leftarrow T_2 \times (T_1 \gg 70) \times (T_1 \gg 151)$
7    return $T_2 = A^{-1}$
```

Table 2 collects the costs of inversions over the NIST fields with different parallelization schemes. Sequential inversions using IT and optimal addition chains are included for convenience. As can be seen in Table 2, (parallel) 3-chains offer significant improvements over both the traditional sequential techniques as well as the parallelization technique presented by Nöcker [7]. Keeping in mind that an HD multiplier with a latency comparable to a normal multiplier occupies approximately twice the area of the normal multiplier [20], we see that optimal 3-chains offer a significant reduction in latency with approximately the same area requirements compared to Nöcker [7]. Naturally, the requirement for a special type of a multiplier may restrict the applicability of the technique in some applications. The use of parallel 3-chains offers further improvements in latency, especially for $GF(2^{233})$, but not without a hefty penalty in area requirements. Hence, parallel 3-chains are likely to be a feasible solution only in applications requiring extremely fast inversions with loose area constraints.

Remark 4. The fact that critical paths achieved with parallel k-chains are at least as short as the critical paths with optimal k-chains does not violate Conjecture 1. This is because the total number of double multiplications computed by using a parallel k-chain is the same or larger than that computed with an optimal k-chain but some of them can be computed in parallel.

A different approach to parallel computation of inversions in $GF(2^m)$ was introduced by Rodríguez-Henríquez et al. in [23]. Instead of aiming to compute multiplications in parallel, they decomposed the inversion algorithm so that one essentially runs two algorithms in parallel: one based on squarings in $GF(2^m)$ and the other on square roots in $GF(2^m)$. Thanks to this decomposition, the squarings and square roots can be computed faster, but the number of multiplications on the critical path remains the same and equals the length of an optimal addition chain for $m-1$. Multiplications are typically significantly slower than squarings or square roots. Consequently, our approaches (using either optimal 3-chains or parallel 3-chains) are likely to offer faster inversions unless very fast multipliers are available; e.g., the results in [23] were obtained using a bit-parallel Karatsuba-Ofman multiplier that has a latency of one clock cycle but occupies a very large area and operates at a low clock frequency.

6 Implementations and Results

6.1 Architecture

An architecture for computing inversions using the TIT algorithm was recently introduced in [15] and it can be used directly with algorithms derived from 3-chains. This architecture employs the HD multiplier presented in [20]. Fig. 3 depicts the architecture and the following example shows how the architecture is used in computing inversions; the reader is referred to [15] for further details.

Example 9. The architecture of Fig. 3 computes an inversion in $GF(2^{233})$ using Algorithm 3 as follows. In order to compute the algorithm, the set of exponents used in the multiplexer should be $e = \{1,2,4,6,18,38,114\}$. Line 1 of Algorithm 3 is given directly with the rotation used at the input. Line 2 is computed by setting $r_0 = r_1 = r_2 = r_3 = 1$

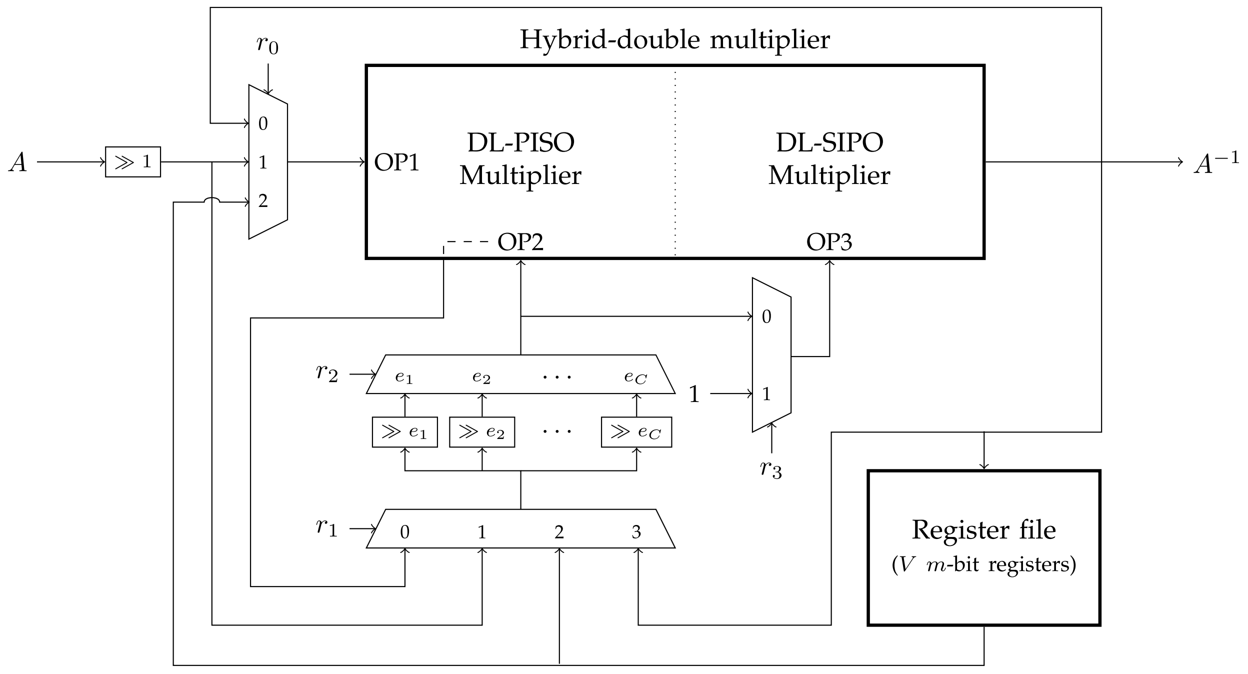


Fig. 3. The inverter architecture introduced in [15] using an HD multiplier from [20].

and line 3 is computed with the selections $r_0 = 0$, $r_1 = 3$, $r_2 = 2$, and $r_3 = 1$. The results of these computations are stored in the register file. Whenever R_1 or R_2 from the register file are used as the first operand (lines 4, 6, and 8), they are selected with $r_0 = 2$, otherwise $r_0 = 0$. Whenever the third operand is used, the rotation is computed from the already rotated second operand. For example, on line 5, one first computes the second operand $(B \gg 6)$ by selecting $r_1 = 3$ and $r_2 = 6$. This result is then used to compute $(B \gg 12)$ by selecting it with $r_1 = 0$ and rotating it again by 6 with the selection $r_2 = 6$. This is possible because the DL-SIPO multiplier in the HD multiplier starts its operation one clock cycle later than the DL-PISO multiplier [15]. When all lines of Algorithm 3 have been processed, A^{-1} is available at the output.

6.2 ASIC Implementations Results

In this section, we provide synthesis results for the architecture from Section 6.1. In order to have a fair comparison to the previous works available in the literature, we have selected 65-nm CMOS library for the synthesis on ASIC technology. The architectures were synthesized using Synopsys Design Compiler tools. The architecture was modelled in VHDL and synthesized for different d, the digit sizes of the HD multiplier (the same digit size was used in both PISO and SIPO multipliers).

We chose fields based on the NIST recommendations [11]. We implemented the architecture for the three fields where using optimal 3-chains improves the results presented in [15], i.e., $m \in \{233, 283, 571\}$. For the remaining two fields, i.e., $m \in \{163, 409\}$, the results would be identical with the results from [15]. In the implementations, we used the following 3-chains that were selected from the sets of optimal 3-chains:

- $m = 233$: $(1, 2, 4, 6, 18, 38, 114, 232)_3$ (Algorithm 3)
- $m = 283$: $(1, 3, 9, 19, 47, 94, 282)_3$
- $m = 571$: $(1, 2, 4, 10, 30, 90, 270, 570)_3$.

The implementation results with any of the optimal 3-chains are similar and, therefore, the 3-chain can be chosen arbitrarily as long as it is optimal.

We investigated the performance of the proposed scheme by varying the digit-size d. The digit-sizes were chosen so that we found the minimum time required for an inversion with every scheme. The syntheses results are reported in Tables 3, 4, and 5, for $m = 233$, $m = 283$, and $m = 571$, respectively. The latency of a single multiplication is $q = \lceil m/d \rceil$ and the latency of a double multiplication is $q + 1$. The tables include minimum critical path delays (CPD) given by the syntheses. The inversion times were computed by assuming that the circuitry is clocked at the maximum clock frequencies determined by the CPDs.

The results show that 3-chains with HD multipliers lead to the fastest inverters available in the literature. We stress that our main contributions are based on employing 3-chains for reducing the latency of inversions and it is natural that the results come with the expense of larger area requirements than traditional schemes, such as IT. However, 3-chains also allow area-time trade-offs and provide faster inversion than IT or TIT with the same area requirements. Fig. 4 demonstrates this by showing the area-time plots of the results given in Tables 3, 4, and 5. Fig. 4 shows that, as expected, using 3-chains leads to faster inversions than TIT when the area requirements of the two implementations are similar. As shown by Fig. 4, the 3-chain scheme also achieves better timing with smaller area than IT for all three fields when the digit-size d gets large enough; i.e., 3-chains achieve faster inversion times with approximately the same area as IT and TIT.

When designing inverters for extremely high-performance applications, one can increase d further and get even

TABLE 3
ASIC (65-nm CMOS Standard Technology) Synthesis Results for the Inverter Using 3-Chains and its Comparison to Inverters Using Optimal Addition Chains (Opt. AC) or IT, Which Has the Same Number of Multiplications, and TIT for Type 2 GNB over $GF(2^{233})$ for Different Digit Sizes

d	q	$q+1$	Opt. AC / IT (DL-PIPO Multiplier)				3-chain (HD Multiplier)			
			Latency	CPD [ns]	Time [ns]	Area [μm^2]	Latency	CPD [ns]	Time [ns]	Area [μm^2]
8	30	31	312	0.98	305.76	30,776	226	1.55	350.30	57,215
10	24	25	252	1.18	297.36	36,059	184	1.68	309.12	68,197
12	20	21	212	1.42	301.04	41,259	156	1.82	283.92	80,711
14	17	18	182	1.57	285.74	46,272	135	2.01	271.35	92,084
16	15	16	162	1.71	277.02	51,844	121	2.18	263.78	103,492
20	12	13	132	2.10	277.20	62,469	100	2.48	248.00	126,936
24	10	11	112	2.45	274.40	74,076	86	2.66	228.76	150,291
26	9	10	102	2.69	274.38	78,729	79	2.97	234.63	161,962
30	8	9	92	2.95	271.40	90,490	72	3.37	242.64	184,882
34	7	8	82	3.19	261.58	101,245	65	3.59	233.35	207,650
39	6	7	72	3.63	261.36	114,231	58	4.17	241.86	236,791
47	5	6	62	4.34	269.08	136,594	51	4.94	251.94	282,507
59	4	5	52	5.43	282.36	169,434	44	6.30	277.20	351,667

TABLE 4
ASIC (65-nm CMOS Standard Technology) Synthesis Results for the Inverter Using 3-Chains and its Comparison to Inverters Using Optimal Addition Chains (Opt. AC) or IT, Which Has the Same Number of Multiplications, and TIT for Type 6 GNB over $GF(2^{283})$ for Different Digit Sizes

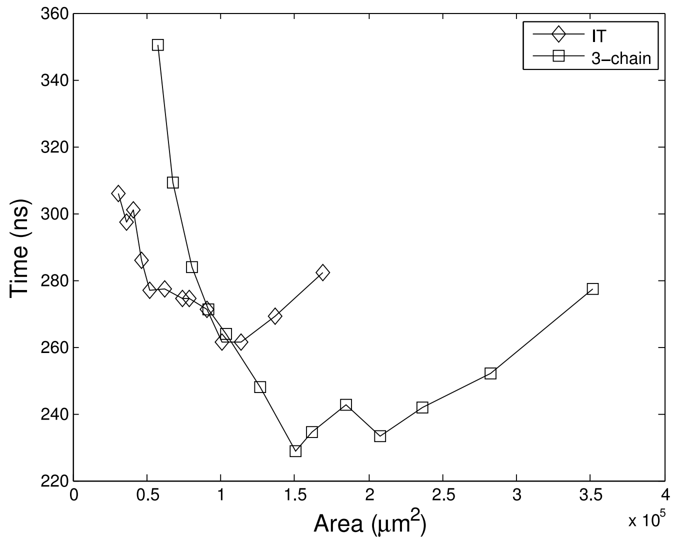
d	q	$q+1$	Opt. AC / IT (DL-PIPO Multiplier)				TIT (HD Multiplier) [14]				3-chain (HD Multiplier)			
			Latency	CPD [ns]	Time [ns]	Area [μm^2]	Latency	CPD [ns]	Time [ns]	Area [μm^2]	Latency	CPD [ns]	Time [ns]	Area [μm^2]
3	95	96	1,058	0.98	1,036.84	26,930	778	1.55	1,205.90	53,377	584	1.58	922.72	53,458
4	71	72	794	1.02	809.88	31,573	586	1.60	937.60	65,092	440	1.62	712.80	65,152
7	41	42	464	1.39	644.96	46,482	346	1.95	674.70	102,332	260	1.98	514.80	102,419
15	19	20	222	2.26	501.72	85,943	170	2.71	460.70	199,123	128	2.77	354.56	199,357
29	10	11	123	3.84	472.32	156,141	98	4.12	403.76	370,651	74	4.16	307.84	370,753
36	8	9	101	4.60	464.60	190,573	82	5.08	416.56	455,238	62	5.10	316.20	455,538
41	7	8	90	5.10	459.00	215,265	74	5.57	412.18	516,789	56	5.59	313.04	516,903
48	6	7	79	5.91	466.89	250,518	66	6.43	424.38	615,322	50	6.45	322.50	601,863

TABLE 5
ASIC (65-nm CMOS Standard Technology) Synthesis Results for the Inverter Using 3-Chains and its Comparison to Inverters Using Optimal Addition Chains (Opt. AC) and TIT for Type 10 GNB Over $GF(2^{571})$ for Different Digit Sizes

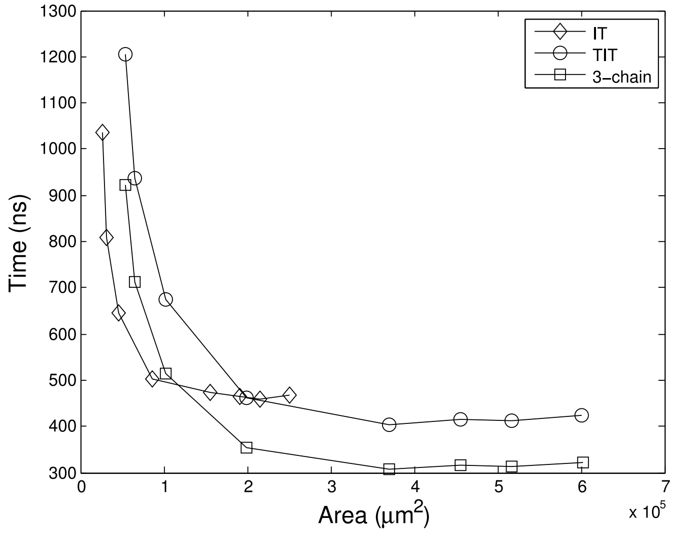
d	q	$q+1$	Opt. AC (DL-PIPO Multiplier)				TIT (HD Multiplier) [14]				3-chain (HD Multiplier)			
			Latency	CPD [ns]	Time [ns]	Area [μm^2]	Latency	CPD [ns]	Time [ns]	Area	Latency	CPD [ns]	Time [ns]	Area [μm^2]
11	52	53	638	2.03	1,295.14	164,328	434	2.81	1,219.54	433,405	380	2.85	1,083.00	433,495
13	44	45	542	2.30	1,246.60	190,443	370	3.95	1,461.50	506,675	324	3.09	1,001.16	506,689
22	26	27	326	3.57	1,163.82	307,559	226	4.01	906.26	836,351	198	4.07	805.86	836,491
26	22	23	278	4.09	1,137.02	361,435	194	4.48	869.12	984,782	170	4.52	768.40	984,980
28	21	22	266	4.20	1,117.20	392,682	186	4.70	874.20	1,058,140	163	4.76	775.88	1,058,230
29	20	21	254	4.37	1,109.98	403,217	178	4.89	870.42	1,094,805	156	4.91	765.96	1,094,937
31	19	20	242	4.71	1,139.82	433,000	170	5.10	867.00	1,167,750	149	5.12	762.88	1,167,844
34	17	18	218	4.93	1,074.74	468,577	154	5.40	831.60	1,277,456	135	5.42	731.70	1,277,504
36	16	17	206	5.25	1,081.50	493,956	146	5.78	843.88	1,350,567	128	5.79	741.12	1,350,704
39	15	16	194	5.60	1,086.40	536,060	138	6.03	832.14	1,461,921	121	6.10	738.10	1,462,033
41	14	15	182	5.76	1,048.32	559,529	130	6.29	817.70	1,535,087	114	6.32	720.48	1,535,168
44	13	14	170	6.29	1,069.30	597,902	122	6.85	835.70	1,644,678	107	6.91	739.37	1,644,759

faster results by occupying more area. This is not possible to this extent with IT because synthesizing inverters with large d becomes very difficult and leads to poor CPDs and, consequently, longer inversion times.

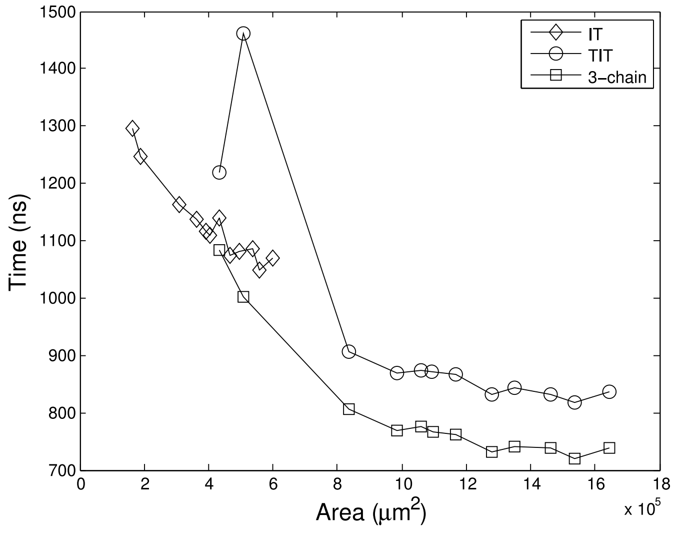
If the clock frequency used by the system is fixed and the system cannot be clocked at the maximum frequency determined by the CPDs (e.g., if the maximum CPD of other parts of the system is larger than that of the inverter), then



(a)



(b)



(c)

Fig. 4. Area-time plots of the ASIC implementations for (a) $m = 233$, (b) $m = 283$, and (c) $m = 571$.

TABLE 6
The Number of Double Multiplications with IT and Optimal 3-Chains for the Fields with Optimal Normal Bases Proposed to Be Used for PRNG in [24]

m	IT	Opt. 3-chain	Reduction
30	6	4	−33%
33	5	4	−20%
35	6	4	−33%
36	7	4	−43%
58	7	4	−43%
60	8	5	−38%
65	6	5	−17%
66	7	5	−29%
119	9	6	−33%
130	8	5	−38%
131	8	6	−25%
251	11	7	−36%
254	12	6	−50%
508	14	7	−50%
509	13	7	−46%

using 3-chains provides even more significant speedups compared to IT because the speedup is determined by the latency columns of the tables, and not by the time columns.

7 Other Applications for k-Chains

It is widely known that addition chains can be used for computing general exponentiations A^n. Similarly, 3-chains and HD multipliers can be used for computing general exponentiations A^n, where $A \in GF(2^m)$ and n is an arbitrary positive integer. The special structure of n in the case of inversions, where $n = 2^m - 2$, allowed one to find an efficient algorithm directly for an entire n because it was enough to find an optimal k-chain for $m - 1$ which is small. Unfortunately, n is typically very large in practical applications which often prevents one from finding an optimal 3-chain for an entire arbitrary n. If n is predefined, it is possible to derive an optimal, or at least piecewise optimal, result offline and use it for efficient exponentiation by hardcoding it. Another option is to find a short, but probably sub-optimal, 3-chain using Algorithm 2. The use of even these suboptimal 3-chains instead of regular binary addition chains results in significant speedups for general exponentations A^n if an HD multiplier is available: the exponentiation requires $\lfloor \log_3 n \rfloor + \lceil (d_3(n) - 1)/2 \rceil$ double multiplications instead of $\lfloor \log_2 n \rfloor + h_2(n)$ multiplications. General exponentiations can be accelerated also by using parallel 3-chains and, in that case, the latency reduces to only $\lceil \log_3 n \rceil$ double multiplications.

The proposed techniques based on 3-chains can have importance in basically all applications that require exponentiations (or specifically inversions) in binary fields. Speeding up pseudo-random number generation (PRNG) using inversions in finite fields is an example of this. In [24], Eichenauer-Herrmann and Niederreiter proposed a PRNG scheme where a sequence of pseudo-random numbers is retrieved from $\gamma_i \in GF(q)$ defined by the recursion

$$\gamma_i = \alpha\gamma_{i-1}^{-1} + \beta, \quad i \geq 1. \tag{17}$$

Clearly, the computational cost of the scheme is dominated by the inversion: $\gamma_{i-1}^{-1} = \gamma_{i-1}^{2^m-2}$.

Eichenauer-Herrmann and Niederreiter proposed using $GF(2^m)$ with an optimal normal basis such that m is reasonably close to a power of two [24]. The suggested m are collected in Table 6 together with the costs of inversions using IT that was suggested in [24] and an optimal 3-chain. Clearly, using optimal 3-chains and HD multipliers would allow significant reductions (between 17-50 percent) in the latency of the PRNG scheme introduced in [24].

8 Conclusions

In this paper, we discussed a generalization of addition chains called k-chains. Especially, we showed how 3-chains can lead to very fast computation of inversions in $GF(2^m)$ if one can employ an HD multiplier. Using an optimal 3-chain leads to significant speedups over both traditional sequential solutions such as [17], [18], [19] and parallel solutions such as [7], [23]. It is notable that 3-chains offer speed improvements over Nöcker's parallel approach from [7] with approximately the same area requirements because the area of an HD multiplier is approximately twice the area of the single multiplier having the same latency. We also introduced parallel k-chains which can lead to even more efficient parallelizations, especially, if n is very large. Practical consequences of parallel k-chains will be studied more thoroughly in the future.

The findings of this paper can be utilized also in the context of general exponentiations and can have importance in various applications using finite field arithmetic. An example of such an application is the PRNG via inversions in binary fields. As future research, we plan to study the use of Algorithm 2 for general exponentiations in practice. It is also possible that (parallel) k-chains have independent interest in applications beyond finite field arithmetic with HD multipliers and, hence, we encourage future research on applications of k-chains.

Acknowledgments

The authors express their deepest graditude to Dr. Colin Walter for providing insightful comments and suggestions to an early version of this paper. The work of K. Järvinen was done when he was with Aalto University, Department of Information and Computer Science, and it was funded by the Academy of Finland under the project #138358. Dr. Kimmo Järvinen is the corresponding author.

References

[1] D. E. Knuth, *The Art of Computer Programming. Vol. 2, Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.
[2] H. Volger, "Some results on addition/subtraction chains," *Inf. Process. Lett.*, vol. 20, no. 3, pp. 155–160, Apr. 1985.
[3] F. Morain and J. Olivos, "Speeding up the computations of an elliptic curve using addition-subtraction chains," *RAIRO Theoretical Inf. Appl.*, vol. 24, no. 6, pp. 531–543, 1990.
[4] H. M. Bahig, "On a generalization of addition chains: Addition-multiplication chains," *Discr. Math.*, vol. 308, no. 4, pp. 611–616, 2008.
[5] A. C.-C. Yao, "On the evaluation of powers," *SIAM J. Comput.*, vol. 5, no. 1, pp. 100–103, 1976.
[6] M. Nöcker, "Some remarks on parallel exponentiation (extended abstract)," in *Proc. Int. Symp. Symbolic Algebraic Comput.*, 2000, pp. 250–257.
[7] M. Nöcker, "Data structures for parallel exponentiation in finite fields," Ph.D. dissertation, Faculty Mathematics Comput. Sci., Univ. Paderborn, Paderborn, Germany, 2001.
[8] J. von zur Gathen and M. Nöcker, "Computing special powers in finite fields," *Math. Comput.*, vol. 73, no. 247, pp. 1499–1523, 2004.
[9] P. L. Montgomery. (1983). Evaluating recurrences of form $x_{m+n} = f(x_m, x_n, x_{m-n})$ via Lucas chains, revised 1992, unpublished, [Online]. Available: http://cr.yp.to/bib/1992/montgomery-lucas.ps
[10] T. Hayata and M. Wagatuma. (2006). On ternary addition chains, unpublished, [Online]. Available: http://tnt.math.se.tmu.ac.jp/pub/ac05/Hayata/hayata.pdf
[11] National Institute of Standards and Technology (NIST), Digital signature standard (DSS) Federal Information Processing Standard, FIPS PUB 186-4, Jul. 2013.
[12] E. G. Thurber, "Efficient generation of minimal length addition chains," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1247–1263, 1999.
[13] A. Brauer, "On addition chains," *Bull. Amer. Math. Soc.*, vol. 45, pp. 736–739, 1939.
[14] A. Schönhage, "A lower bound for the length of addition chains," *Theoretical Comput. Sci.*, vol. 1, no. 1, pp. 1–12, 1975.
[15] R. Azarderakhsh, K. Järvinen, and V. Dimitrov, "Fast inversion in $GF(2^m)$ with normal basis using hybrid-double multipliers," *IEEE Trans. Comput.*, vol. 63, no. 4, pp. 1041–1047, Apr. 2014.
[16] C. C. Wang, T. K. Troung, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI architectures for computing multiplications and inverses in $GF(2^m)$," *IEEE Trans. Comput.*, vol. C-34, no. 8, pp. 709–717, Aug. 1985.
[17] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases," *Inf. Comput.*, vol. 78, no. 3, pp. 171–177, Sep. 1988.
[18] N. Takagi, J. Yoshiki, and K. Takagi, "A fast algorithm for multiplicative inversion in $GF(2^m)$ using normal basis," *IEEE Trans. Comput.*, vol. 50, no. 5, pp. 394–398, May 2001.
[19] V. Dimitrov and K. Järvinen, "Another look at inversions over binary fields," in *Proc. 21st IEEE Int. Symp. Comput. Arithmetic*, 2013, pp. 211–218.
[20] R. Azarderakhsh and A. Reyhani-Masoleh, "Low-complexity multiplier architectures for single and hybrid-double multiplications in Gaussian normal bases," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 744–757, Apr. 2013.
[21] National Institute of Standards and Technology (NIST), "*Advanced encryption standard (AES)*," Federal Information Processing Standard, FIPS PUB 197, Nov. 2001.
[22] A. Reyhani-Masoleh, "A new bit-serial architecture for field multiplication using polynomial bases," in *Proc. 10th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2008, vol. 5154, p. 300–314.
[23] F. Rodríguez-Henríquez, G. Morales-Luna, N. A. Saqib, and N. Cruz-Cortés, "Parallel Itoh-Tsujii multiplicative inversion algorithm for a special class of trinomials," *Designs, Codes Cryptography*, vol. 45, no. 1, pp. 19–37, Oct. 2007.
[24] J. Eichenauer-Herrmann and H. Niederreiter, "Digital inversive pseudorandom numbers," *ACM Trans. Model. Comput. Simul.*, vol. 4, no. 4, pp. 339–349, 1994.

Kimmo Järvinen received the MSc (Tech) degree in 2003 and the DSc (Tech) degree in 2008, both in electrical engineering from the Helsinki University of Technology (TKK), Espoo, Finland. He was with the Signal Processing Laboratory at TKK from 2002 to 2008. From 2008 to 2014, he was in the Cryptology Group, Department of Information and Computer Science, Aalto University, Espoo, Finland. He had a three-year postdoctoral researcher's project funded by the Academy of Finland from 2011 to 2014. In 2014, he joined the COSIC group of KU Leuven ESAT, Leuven, Belgium, where he is currently a FWO Pegasus Marie Curie fellow. His research interests include efficient and secure realization of cryptosystems, general computer arithmetic, and FPGAs.

Vassil Dimitrov received the PhD degree in mathematics from the Mathematical Institute of the Bulgarian Academy of Sciences in 1995. He has had research scientist positions at the University of Windsor, Canada, Helsinki University of Technology, Finland, and Nanyang Technological University, Singapore. He has been an associate professor at the Department of Electrical and Computer Engineering, University of Calgary, Canada since 2001. His research interests include implementation of cryptographic protocols, computational complexity problems, the use of number theoretic techniques in digital signal processing, image compression and related topics.

Reza Azarderakhsh received the BSc degree in electrical and electronic engineering in 2002, the MSc degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2005, and the PhD degree in electrical and computer engineering from the University of Western Ontario, London, ON, Canada, in 2011. He joined the Department of Electrical and Computer Engineering, University of Western Ontario, as a limited duties instructor, in September 2011. He received the Natural Sciences and Engineering Research Council of Canada (NSERC) Post-Doctoral Research Fellowship in 2012. He has been a post-doctoral research fellow with the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON, Canada. He is currently a faculty member with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY. His current research interests include finite field and its application, elliptic curve cryptography, and pairing based cryptography.