# A Survey of Addition Chain Algorithms

Thomas Schibler
tschibler@gmail.com
Department of Computer Science
University of California Santa Barbara

June 13, 2018

**Abstract**

Modular exponentiation appears as a necessary computational tool in many cryptographic systems. Specifically, given modulus $n$, base $b$ and exponent $e$, exponentiation computes $b^e \mod n$ through repeated multiplications. To minimize the number of such multiplications, an *addition chain* of exponents is computed, such that each exponent in the chain can be expressed as the sum of two previous exponents. While the problem of finding a minimal addition chain for a given exponent $e$ is widely thought to be intractable, many algorithms exist to find short chains. We survey a variety of addition chain algorithms, focusing on those which partition the bit representation of the target exponent into manageable sized blocks.

## 1 Introduction

Addition Chains are an important tool in cryptographic settings, particularly for public key cryptography algorithms like RSA, in which fast modular exponentiation is a must. For background, modular exponentiation involves computing the value $b^e \mod n$ for some base $b$, exponent $e$, and modulus $n$. While in general the value of $b^e$ can grow unbounded particularly as $e$ becomes large, $b^e \mod n$ can never exceed the value of $n$. Naturally, we want to compute the modular result without having to deal with such large numbers. Indeed, in crypto settings the raw value $b^e$ would often exceed any feasible memory limit. Fortunately, we have the identity,

$$c \mod n = (a \mod n) \cdot (b \mod n)$$

if $a = b \cdot c$. Therefore for modular exponentiation, decomposing the exponent $e$, such that $e = e_1 + e_2$, yields

$$b^e \mod n = (b^{e_1} \mod n) \cdot (b^{e_2} \mod n),$$

allowing us to condense the result of intermediate multiplications to the size of the modulus $n$. However, this result also indicates that we need to break apart the

exponent into pieces that sum together, leading us to addition chains. Specifically, an addition chain is a sequence of values

$$A = (a_1, a_2, \cdots, a_m),$$

such that $a_1 = 1$, and $a_i = a_j + a_k$ with $j, k < i$ for all $i > 1$. That is to say, each value in the chain (other than the starting value of 1) can be computed as a sum of two previous values. If the final value of the chain $a_m = e$ for our exponent $e$, then each step of the chain implies a multiplication step in computing $b^e \mod n$. As modular exponentiation is of the more costly operations in algorithms like RSA, an efficient method is appealing, corresponding to few multiplications, or, a short addition chain.

# 2 Minimum Length Addition Chains

The key to fast modular exponentiation lies in efficient computation of short addition chains. Specifically, the length of an addition chain $A = (a_1, a_2, \cdots, a_m)$ is defined as $l(A) = m - 1$, as $a_1 = 1$ will always be the first value, leaving $m - 1$ remaining steps. Let us also specify $L(n)$ as the length of the shortest chain terminating with $n$. Unfortunately, [1], [] et. al. show that problem of finding a minimum length addition chain containing every value in some set $X = \{x_1, x_2, \cdots x_n\}$ is NP-Hard. However, [2] and [] both point out that this result does not indicate a hardness result for the case of single target value, which is often assumed erroneously. Therefore, the problem of finding a minimum length chain given target $n$ is still open. That said, numerous observations support the conjecture that the problem is indeed hard.

For example, Figure 2 depicts the shortest chains for small values of $n$ as a tree. Already, we can see that the minimum length addition chain is not monotone in $n$, and demonstrates little regularity in the relation between $n$ and $n + 1$. Surprisingly, [3] even shows that there exists $n$ such that $L(2n) < L(n)$. Upon careful inspection, one might notice that chains are always extended from $a_{i-1}$ to $a_i$ by adding some value $a_j, j < i$ to $a_{i-1}$. Such chains that always use the most recent value $a_{i-1}$ are called *star* chains (or *Brauer* chains as a result of [brauer]). In general the minimum length chain may not be a star chain (proof from Hansen in 1958), with 12,509 as the smallest $n$ for which this occurs [geneticalg]. Furthermore, Brauer proved that for such chains, which he called "special chains," denoted $L^*$,

$$L(2^{n+1} - 1) \leq m + L^*(n + 1),$$

which only differs in the use of a Brauer chain from the longstanding Scholz conjecture

$$L(2^{n+1} - 1) \leq n + L(n + 1).$$

Clift has also proven equality of the Scholz conjecture for values of $n$ up to $2^32$ in [clift]. Brauer's formula (and the Scholz conjecture should it be proven, especially with equality) bound the length of a chain for $2^n - 1$ in terms of the length of a chain for the exponent $L(n)$, and the minimum number of doublings $\lfloor \log(2^n - 1) \rfloor = n - 1$.

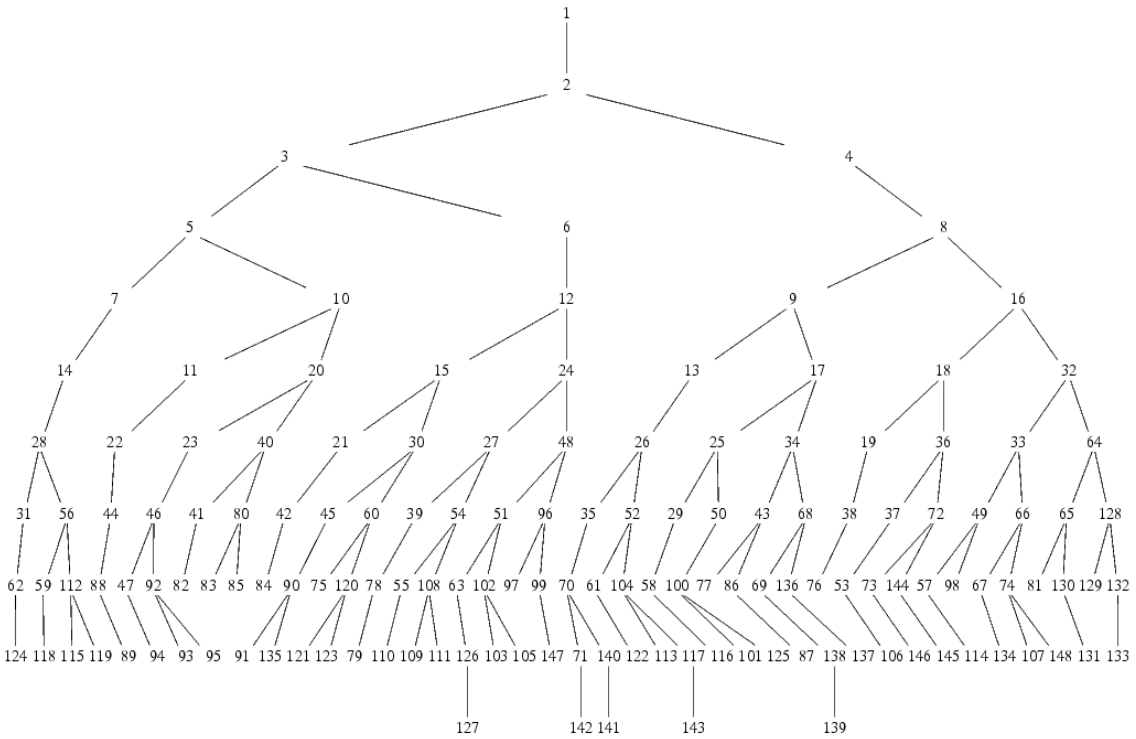


Figure 1: Minimum length addition chains for target $n \leq 148$. Reproduced from [4]

However, as [2] points out, even if the problem of finding a minimum length chain for target $n$ is hard, finding efficient approximations may not be. Many heuristics exist for the shortest chain problem, some quite complex like the evolutionary techniques described in [5]. Picek et. al. generate a variety of valid chains for target $n$, then apply a genetic algorithm that repeatedly combines such chains and "repairs" them to optimize for length. Among so many algorithms, we focus primarily on those which compute a chain based on the bit representation of $n$, which give provable bounds on the produced chain length. Fortunately it is easy to show a lower bound on the chain length $L(n)$ for any $n$, yielding a base for comparing approximations. [6] show that minimum chain length is $log_2 n + log_2 s(n) - 2.13$, where $s(n)$ is the *Hamming weight* of $n$, the sum of bits in its binary expansion. We instead include the intuitive inductive proof of the bound of $\log_2 n$.

**Lemma 2.1.** *An addition chain $A$ containing $n$ must have length $L(A) \geq \lfloor \log_2 n \rfloor$.*

*Proof.* This follows from the fact that $a_i$ appears in chain $A = (a_1, \cdots, a_m)$ only if $a_i = a_j + a_k$ for $j, k < i$. If $j = k = i - 1$, then the chain value can at most double with each step. By induction, it takes at least $\lfloor \log_2 n \rfloor$ doublings to reach $n$. □

Additionally, it is easy to show an upper bound, such that for all $n$, $L(n) \leq 2\lceil \log_2 n \rceil$, which is the subject of the next section.

# 3 Binary Method

The binary method is one of the simplest, efficient methods for computing a short addition chain. Given a target $n$ in its bit representation, $n = (b_{m-1}, \cdots, b_0)_2$, the binary method computes a chain for $n$ by processing the $m$ bits in either direction. To process right to left, we simply compute each power of 2 through repeated doubling, while summing all those that correspond to non-zero bits of $n$. To process left to right, we double with each bit, additionally adding 1 on each non-zero bit.

Both of these variants compute $\lfloor \log_2(n) \rfloor$ doublings, together with a single addition for each non-zero bit. In the worst case, this is roughly only twice the previously described lower bound, and only 1.5 times in the expected case. Interestingly the worst case are the all 1 strings, which are precisely those described in the Scholz conjecture. For the $m$-bit all 1 string $(2^m - 1)$, the Scholz conjecture would indicate that the length of the chain should be $m - 1 + L(m)$. This is identical to binary method in terms of the number of doublings, but differs in the additions. While the binary method would compute an addition for each bit, the Scholz conjecture predicts that this can be done with a shortest chain for the *number of bits*. In other words, the binary method could theoretically be improved from $2(m-1)$ to $m - 1 + O(logm)$, or equivalently from a factor 2 to $1 + O(\log(m)/m)$. In the next section, the Brauer Method comes close to accomplishing this goal.

# 4 Brauer's Method

In [brauer], Brauer also described an algorithm for addition chains, which [pippenger] notes is now called Brauer's method, or the $2^k$-ary method. Similar to the binary method, this method expresses $n$ in its bit representation. However, instead of processing individual bits, it processes them $k$ bits at a time, still preforming doublings and additions. It preforms $k$ doublings with each step, still one per bit as expected based on the Scholz conjecture (recall that the number of doublings indicated by the conjecture remains unchanged). The method gets the name $2^k$-ary however, because each addition is based on $k$ bits, or values in $[1, 2^k)$. The Brauer method therefore precomputes all values in this range with the full chain $(1, 2, \cdots, 2^k)$, for use later in the chain. For reasonably sized $k$, this can cut the number of additions by a factor $k$, without requiring too severe preprocessing. In fact, as Bernstein summarizes in [2], the Brauer method achieves chains of length

$$\log n + (1 + o(1)) \frac{\log n}{\log \log n}.$$

Furthermore, [2] notes that with a lower bound of Erdős, Brauer's method always finds a chain within a factor

$$\frac{1 + o(1)}{\log \log n}$$

of optimal. This is much closer to the desired goal based on Scholz conjecture. In the remaining portion, we include a few adjustments pointed out in [2]. That said,

Brauer's method already comes very close to achieving optimal chains, and further improvement is likely of lower priority than other cryptographic concerns.

### 4.1 Minor Remarks

In [2], Bernstein also highlights a few small improvements to Brauer's algorithm from several sources; we briefly describe a few below.

1. Of these improvements, Thurber's appears most interesting. Instead of always doubling then adding some $r \in [1, 2^k)$, one can instead add $r/2$, and then proceed to double. This change effectively means that even values do not need to be precomputed, as we have a choice of how to proceed with each block.

2. Knuth points out that not all values in $[1, 2^k)$ will be used. Especially for larger $k$, we can instead compute a shortest chain targeting a value only it appears in some $k$-bit block. Of course, this problem must still be approximated as it is precisely the problem shown to be NP-Hard in [1].

The remaining points in [2] address cases of computing products or sequences of values, rather than a single target. While interesting extensions to the addition chain problem, we leave them unturned for the curious reader.

## 5 Concluding Remarks

It is worth noting that addition chains are useful not only in modular exponentiation, which we cited as the main motivation, but for all problems involving repeated operations that can be similarly decomposed, such as point addition for elliptic curves. Additionally, the addition chain algorithms and theoretical bounds discussed only address the case of short addition chains, where they come very close to optimal. While interesting theory questions remain, such as whether the single target case is in fact NP-Hard like the sequence problem of [1], modern cryptography has other pressing concerns. For many crypto settings, resistance from side channel attacks is an important consideration, which these algorithms do not take into concern. Additionally, some settings may allow for chains that include other operations. That said, the algorithms addressed in this paper form an important foundation for understanding some of the core engineering components of cryptography.

# References


[1] P. Downey, B. Leong, and R. Sethi. Computing sequences with addition chains. *SIAM Journal on Computing*, 10(3):638–646, 1981.

[2] Daniel J. Bernstein. Pippenger's exponentiation algorithm. *Mathematics Subject Classification*, 1991.

[3] Neill Michael Clift. Calculating optimal addition chains. *Computing*, 91:265–284, 2011.

[4] Shortest addition chains.

[5] Stjepan Picek et. al. Evolutionary algorithms for finding short addition chains: Going the distance.

[6] Arnold Schönhage. A lower bound for the length of addition chains. *Theoretical Computer Science*, 1(1):1 – 12, 1975.