Evaluating Bernstein-Rabin-Winograd Polynomials

Sebati Ghosh and Palash Sarkar
Indian Statistical Institute
203, B.T.Road, Kolkata, India - 700108.
{sebati_r,palash}@isical.ac.in

November 2, 2017

Abstract

We describe an algorithm which can efficiently evaluate Bernstein-Rabin-Winograd (BRW) polynomials. The presently best known complexity of evaluating a BRW polynomial on $m \geq 3$ field elements is $\lfloor m/2 \rfloor$ field multiplications. Typically, a field multiplication consists of a basic multiplication followed by a reduction. The new algorithm requires $\lfloor m/2 \rfloor$ basic multiplications and $1 + \lfloor m/4 \rfloor$ reductions. Based on the new algorithm for evaluating BRW polynomials, we propose two new hash functions BRW128 and BRW256 with digest sizes 128 bits and 256 bits respectively. The practicability of these hash functions is demonstrated by implementing them using instructions available on modern Intel processors. Timing results obtained from the implementations suggest that BRW based hashing compares favourably to the highly optimised implementation by Gueron of Horner's rule based hash function.

Keywords: almost universal hash function, BRW polynomials.

1 Introduction

In [1], Bernstein built upon a previous work due to Rabin and Winograd [7] to propose a family of polynomials which has been called the BRW polynomials in [8]. The importance of such polynomials for constructing almost universal hash functions with low collision and differential probabilities has been discussed in [1]. Such hash functions have many applications in cryptography including constructions of message authentication code (MAC), authenticated encryption and disk encryption schemes among others.

A BRW polynomial is constructed from $m \geq 0$ field elements. For a fixed value of m, hardware implementation of BRW polynomials has been reported in [3]. Also, a recent work [2] reports software implementations of BRW polynomials for $m \leq 31$ over the fields $GF(2^{128})$ and $GF(2^{256})$.

The definition of BRW polynomials is recursive. A recursive implementation is possible, but, will not be efficient. To the best of our knowledge, till date no algorithm has been proposed for efficiently evaluating BRW polynomials where m can vary.

Our Contributions

In this work, we present an efficient non-recursive algorithm for evaluating BRW polynomials constructed from m field elements without any restriction on m. The algorithm processes its input in a left-to-right fashion and maintains a set of partial results computed from the elements that have been processed. For a fixed $t \geq 2$, the algorithm reads the next 2^t elements and updates the partial results. The subtlety of the algorithm is in the manner in which the partial results are maintained and updated.

For $m \geq 3$, Bernstein [1] showed that a BRW polynomial defined using m field elements can be evaluated using $\lfloor m/2 \rfloor$ field multiplications. Typically, a field multiplication consists of a basic multiplication followed by a reduction operation. We show that a BRW polynomial defined using $m \geq 3$ field elements can be evaluated using $\lfloor m/2 \rfloor$ basic multiplications and $1 + \lfloor m/4 \rfloor$ reduction operations. This is a significant reduction in the number of operations.

As a practical contribution, we propose two new hash functions, namely BRW128 and BRW256 which are based on BRW polynomials over the fields $GF(2^{128})$ and $GF(2^{256})$ respectively. These hash functions have been implemented using the Intel intrinsics instruction set available on modern Intel processors. Timing results for BRW128 compare favourably to those of a highly optimised implementation by Gueron [5] of Horner's rule based hash function.

2 Preliminaries

Throughout the paper, n is a positive integer and p is a prime.

Finite field: Let $\mathbb{F} = GF(p^n)$ be the finite field of p^n elements. The addition operation over $\mathbb{F}$ will be denoted by $+$; for $X, Y \in \mathbb{F}$, the product will be denoted as XY.

For $X, Y \in \mathbb{F}$, let $\mathsf{mult}(X, Y)$ denote the operation used to compute the product XY. The operation $\mathsf{mult}(X, Y)$ consists of two distinct steps. The first step consists of a basic or an unreduced multiplication $\mathsf{unreducedMult}(X, Y)$ which returns a value Z and the second step consists of an operation $\mathsf{reduce}(Z)$, i.e.,

$$\mathsf{mult}(X, Y) \quad = \quad \mathsf{reduce}(\mathsf{unreducedMult}(X, Y)). \tag{1}$$

Depending on the value of n, there are two scenarios.

1. Case $n = 1$. In this case, elements of $\mathbb{F}$ are represented by the integers $0, \ldots, p-1$. Given two integers X and Y in $\{0, \ldots, p-1\}$, the operation $\mathsf{unreducedMult}(X, Y)$ performs the integer multiplication of the integers X and Y. The result Z is then at most $(p-1)^2$ and the operation $\mathsf{reduce}(Z)$ returns the element $W \in \{0, \ldots, p-1\}$ such that $W \equiv Z \bmod p$.

2. Case $n > 1$. In this case, using a fixed irreducible polynomial $f(x)$ of degree n over $GF(p)$, the elements of $\mathbb{F}$ can be identified with the polynomials over $GF(p)$ of degrees less than n. Given two such polynomials $X = X(x)$ and $Y = Y(x)$, the operation $\mathsf{unreducedMult}(X, Y)$ performs the polynomial multiplication of $X(x)$ and $Y(x)$ and returns the result $Z = Z(x)$ which is a polynomial of degree at most $2(n-1)$ over $GF(p)$. The operation $\mathsf{reduce}(Z)$ returns $W = W(x)$ such that the degree of $W(x)$ is less than n and $W(x) \equiv Z(x) \bmod f(x)$.

BRW polynomials: For $m \geq 0$, let $\mathsf{BRW} : \mathbb{F} \times \mathbb{F}^m \to \mathbb{F}$ be the function defined below, where, we write $\mathsf{BRW}_\tau(\cdots)$ to denote $\mathsf{BRW}(\tau, \cdots)$.

- $\mathsf{BRW}_\tau() = 0$;
- $\mathsf{BRW}_\tau(M_1) = M_1$;
- $\mathsf{BRW}_\tau(M_1, M_2) = M_1\tau + M_2$;
- $\mathsf{BRW}_\tau(M_1, M_2, M_3) = (\tau + M_1)(\tau^2 + M_2) + M_3$;
- $\mathsf{BRW}_\tau(M_1, M_2, \ldots, M_m) = \mathsf{BRW}_\tau(M_1, \ldots, M_{k-1})(\tau^k + M_k) + \mathsf{BRW}_\tau(M_{k+1}, \ldots, M_m)$;
  if $k \in \{4, 8, 16, 32, \ldots\}$ and $k \leq m < 2k$.

$\mathsf{BRW}_\tau(M_1, M_2, \ldots, M_m)$ is a polynomial in τ whose coefficients are defined from $M_1, \ldots, M_m$. We consider the following problem:

$$\text{Given } \tau, M_1, \ldots, M_m \in \mathbb{F} \text{ compute } \mathsf{BRW}_\tau(M_1, M_2, \ldots, M_m).$$

Informally, we will say that the evaluation of $\mathsf{BRW}_\tau(M_1, M_2, \ldots, M_m)$ is an *m-block* BRW *computation.*

The following facts about BRW polynomials have been proved in [1].

1. For $m \geq 3$, $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$ can be computed using $\lfloor m/2 \rfloor$ field multiplications and $\lfloor \lg m \rfloor$ additional field squarings to compute $\tau^2, \tau^4, \ldots$.
2. *Let* $\mathfrak{d}(m)$ *denote the degree of* $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$. For $m \geq 3$, $\mathfrak{d}(m) = 2^{\lfloor \lg m \rfloor + 1} - 1$ and so $\mathfrak{d}(m) \leq 2m - 1$; the bound is achieved if and only if $m = 2^a$; and $\mathfrak{d}(m) = m$ if and only if $m = 2^a - 1$; for some integer $a \geq 2$.
3. The map from $\mathbb{F}^m$ to $\mathbb{F}[\tau]$ given by $(M_1, \ldots, M_m) \longmapsto \mathsf{BRW}_\tau(M_1, \ldots, M_m)$ is injective. Consequently, for $(M_1, \ldots, M_m), (M'_1, \ldots, M'_m) \in \mathbb{F}^m$, $(M_1, \ldots, M_m) \neq (M'_1, \ldots, M'_m)$, and a uniform random τ from $\mathbb{F}$,

$$\Pr[\mathsf{BRW}_\tau(M_1, \ldots, M_m) = \mathsf{BRW}_\tau(M'_1, \ldots, M'_m)] \leq \frac{\mathfrak{d}(m)}{\#\mathbb{F}} \leq \frac{2m-1}{\#\mathbb{F}}. \tag{2}$$

3 Algorithm

The definition of BRW polynomials is recursive. It is easy to write a recursive program which takes as inputs τ and $M_1, \ldots, M_m$, $m \geq 0$ and produces as output $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$. The function will make two calls to itself on inputs of smaller sizes leading to a binary recursion tree. Such a recursive program, however, will have substantial overhead of stack maintenance and will not lead to a fast implementation. Due to this reason, we do not consider a recursive implementation of BRW.

Suppose m is a fixed integer. For $m = 1, 2$ or 3, the evaluation of $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$ is given by a simple formula. For $m = 4, 5, 6, 7$ and 8, the definitions of $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$ are the following:

$$\left.\begin{array}{rcl}
\mathsf{BRW}_\tau(M_1, \ldots, M_4) &=& \mathsf{BRW}_\tau(M_1, M_2, M_3)(\tau^4 + M_4); \\
\mathsf{BRW}_\tau(M_1, \ldots, M_5) &=& \mathsf{BRW}_\tau(M_1, M_2, M_3)(\tau^4 + M_4) + M_5; \\
\mathsf{BRW}_\tau(M_1, \ldots, M_6) &=& \mathsf{BRW}_\tau(M_1, M_2, M_3)(\tau^4 + M_4) + M_5\tau + M_6; \\
\mathsf{BRW}_\tau(M_1, \ldots, M_7) &=& \mathsf{BRW}_\tau(M_1, M_2, M_3)(\tau^4 + M_4) + (\tau + M_5)(\tau^2 + M_6) + M_7; \\
\mathsf{BRW}_\tau(M_1, \ldots, M_8) &=& \mathsf{BRW}_\tau(M_1, \ldots, M_7)(\tau^8 + M_8).
\end{array}\right\} \tag{3}$$

Again, it is easy to write a sequence of field operations (additions and multiplications) to evaluate $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$ in each of the above cases. Continuing, this process can be carried out for any m. This process, however, is not general as separate code is required for each value of m. In [2], implementations of this approach over $GF(2^{128})$ and $GF(2^{256})$ have been reported for $m = 1, \ldots, 31$.

The goal is to obtain a non-recursive algorithm to evaluate $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$ which works for any m. Suppose the input blocks are $M_1, \ldots, M_m$. From a practical point of view as well as for an efficient implementation, it is desirable to process the blocks in a left-to-right manner. A typical left-to-right algorithm would maintain a partial result X obtained by processing blocks $M_1, \ldots, M_i$ and would read the next block M_{i+1} and update X. We discuss the difficulties faced when trying to use this approach to design an algorithm to evaluate BRW.

Suppose that i blocks have been processed and $i \equiv 0 \bmod 4$ and the partial result computed so far is X. If there are exactly $i+1$ blocks to be processed, then the final result is $X + M_{i+1}$; if there are exactly $i+2$ blocks to be processed, then the final result is $X + M_{i+1}\tau + M_{i+2}$; and if there are exactly $i+3$ blocks to be processed, then the final result is $X + (\tau + M_{i+1})(\tau^2 + M_{i+2}) + M_{i+3}$. This shows that reading only the next unprocessed block, i.e., one block look-ahead is not sufficient. So, the question arises as to how many look-ahead blocks are needed?

There is another difficulty. Suppose, there are exactly $i+4$ blocks to be processed. Since i is a multiple of 4, so is $i+4$. If $i+4$ is a power of two, then the final result is $(X + (M_{i+1} + \tau)(M_{i+2} + \tau^2) + M_{i+3})(M_{i+4} + \tau^{i+4})$. On the other hand, if $i+4$ is not a power of two, then to obtain the final result, $(M_{i+4} + \tau^{i+4})$ is not to be multiplied with $(X + (M_{i+1} + \tau)(M_{i+2} + \tau^2) + M_{i+3})$; instead, it has to be multiplied with the result Y corresponding to the last $(i+4-k)$ blocks where $k \in \{4, 8, 16, 32, \cdots\}$ and $k \leq i+4 < 2k$. Since only X is kept as the partial result of processing the blocks $M_1, \ldots, M_i$, Y is not available. This indicates that more than one partial results have to be maintained. The corresponding question is how many partial results need to be maintained and how are these to be updated?

We develop a non-recursive algorithm to compute $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$ for any $m \geq 1$. The number of look-ahead blocks can be 2^t for any $t \geq 2$. Relevant partial results are stored in an array. The subtlety of the algorithm arises from the maintenance and updation of the partial results. In particular, we note that the number of partial results to be stored is not monotonic increasing with m.

Evaluation of BRW polynomials requires field multiplications. As discussed in Section 2, a field multiplication is a composition of an unreducedMult operation followed by a reduce operation. The number of unreducedMult operations required in evaluating BRW is necessarily equal to the number of field multiplications. On the other hand, it is possible to reduce the number of reduce operations. To be able to do this, we define a modification of BRW polynomials where the final result is not reduced.

- $\mathsf{unreducedBRW}_\tau() = 0$;
- $\mathsf{unreducedBRW}_\tau(M_1) = M_1$;
- $\mathsf{unreducedBRW}_\tau(M_1, M_2) = \mathsf{unreducedMult}(M_1, \tau) + M_2$;
- $\mathsf{unreducedBRW}_\tau(M_1, M_2, M_3) = \mathsf{unreducedMult}((\tau + M_1), (\tau^2 + M_2)) + M_3$;
- $\mathsf{unreducedBRW}_\tau(M_1, M_2, \ldots, M_k)$
  $= \mathsf{unreducedMult}(\mathsf{reduce}(\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_{k-1})), (\tau^k + M_k))$,
  if $k \in \{4, 8, 16, 32, \ldots\}$;
- $\mathsf{unreducedBRW}_\tau(M_1, M_2, \ldots, M_m)$
  $= \mathsf{unreducedBRW}_\tau(M_1, \ldots, M_k) + \mathsf{unreducedBRW}_\tau(M_{k+1}, \ldots, M_m)$,
  if $k \in \{4, 8, 16, 32, \ldots\}$ and $k < m < 2k$.

EvalBRW given in Algorithm 1 shows how to compute $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$ for any $m \geq 1$. The following parameters and data structures are used in the algorithm.

t: an integer ≥ 2 which is a parameter to the algorithm;
isDef$[0, \ldots]$: a bit array;
res$[0, \ldots]$: an array where partial results are stored;
keyPow$[0, \ldots]$: the j-th location stores τ^{2^j}.

The interpretation of the two arrays isDef and res is as follows: $\mathsf{isDef}[j] = 1$ if and only if $\mathsf{res}[j]$ holds a valid partial result.

Algorithm 1 Evaluation of $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$, $m \geq 1$.

```
1: function EvalBRW($\tau, M_1, \ldots, M_m$)
2:     keyPow[0] $\leftarrow \tau$;
3:     if $m > 2$ then
4:         for $j = 1$ to $\lfloor \lg m \rfloor$ do
5:             keyPow[$j$] $\leftarrow$ keyPow[$j-1$]$^2$;
6:         end for;
7:     end if;
8:     isDef[0] $\leftarrow 0$;
9:     if $m \geq 2^t$ then
10:        for $j = 1$ to $\lfloor \lg m \rfloor - t + 1$ do
11:            isDef[$j$] $\leftarrow 0$;
12:        end for;
13:    end if;
14:    for $i = 1$ to $\lfloor m/2^t \rfloor$ do
15:        res[0] $\leftarrow$ unreducedBRW$_\tau(M_{2^t \cdot i-(2^t-1)}, \ldots, M_{2^t \cdot i-1})$;
16:        $j \leftarrow 1$; tmp $\leftarrow$ res[0];
17:        while (isDef[$j$] $= 1$) do
18:            tmp $\leftarrow$ tmp + res[$j$];
19:            $j \leftarrow j + 1$;
20:        end while;
21:        res[$j$] $\leftarrow$ unreducedMult(reduce(tmp), $M_{2^t \cdot i}$ + keyPow[$j + t - 1$]);
22:        isDef[$j$] $\leftarrow 1$;
23:        for $k = 0$ to $j - 1$ do
24:            isDef[$k$] $\leftarrow 0$;
25:        end for;
26:    end for;
27:    $r = m \bmod 2^t$;
28:    tmp $\leftarrow$ unreducedBRW$_\tau(M_{m-r+1}, \ldots, M_m)$;
29:    for $j = 1$ to $\lfloor \lg m \rfloor - t + 1$ do
30:        if isDef[$j$] $= 1$ then
31:            tmp $\leftarrow$ tmp + res[$j$];
32:        end if;
33:    end for;
34:    return reduce(tmp);
35: end function.
```

The array isDef can be implemented using a b-bit unsigned integer: the initialisation in Steps 8 to 13 can be done simply as $\mathsf{isDef} \leftarrow 0$; the value of the j-th position can be obtained as $((\mathsf{isDef} \gg j)$ `and` $1)$ (required in Steps 17 and 30); the value of the j-th bit can be set to 1 using $\mathsf{isDef} \leftarrow (\mathsf{isDef}$ `or` $(1 \ll j))$ (required in Step 22); the j least significant bits of isDef can be set to 0 using $\mathsf{isDef} \leftarrow (\mathsf{isDef}$ `and` $(1^b \ll j))$ (required in Steps 23 to 25).

At Step 15, EvalBRW calls unreducedBRW on $2^t - 1$ blocks while at Step 28, EvalBRW calls unreducedBRW on η blocks where $0 \leq \eta < 2^t$. The algorithm assumes that there is a separate subroutine which returns the evaluation of $\mathsf{unreducedBRW}$ on η blocks for $0 \leq \eta < 2^t$. Since t is a fixed parameter of the algorithm, the computation of $\mathsf{unreducedBRW}$ on η blocks can be done by a fixed sequence of field operations without any loop or branch statement (essentially as a straight line program). For $\eta = 1, 2, 3$, the definition of $\mathsf{unreducedBRW}_\tau(X_1, \ldots, X_\eta)$ is simple and the code to directly evaluate the expression is also quite simple. If $t = 2$, then the code to compute $\mathsf{unreducedBRW}$ on η blocks for $\eta = 0, 1, 2$ and 3 is sufficient. For $t \geq 3$, similar code for direct computation of $\mathsf{unreducedBRW}$ on η blocks can be worked out from the definition. For $GF(2^n)$ with $n = 128$ or $n = 256$, such implementations have been reported in [2] for $t = 5$ and correspondingly $1 \leq \eta \leq 31$.

In $\mathsf{EvalBRW}$, the number of blocks m is assumed to be known. The value of m is used to determine the maximum value of the loop counter at Step 14, to compute the value of r at Step 27 and in the computation of $\mathsf{unreducedBRW}$ at Step 28. It is possible to modify the algorithm to work in the case where the number of blocks is not known at the beginning. The idea is the following. While the buffer is not empty, attempt to read the next 2^t blocks from the buffer. If 2^t blocks are indeed retrieved, then these blocks are processed in the same manner as in $\mathsf{EvalBRW}$; if less than 2^t blocks are retrieved, then these are the last blocks in the buffer and the "wrapping up" procedure is executed in a manner also similar to that of $\mathsf{EvalBRW}$. With this idea, it is straightforward to write out the details of an algorithm which does not require to know the value of m at the outset. Hence, we do not provide an explicit description of such an algorithm.

4 Correctness and Complexity

The material in this section is divided into three parts. In the first part, we prove some results on the structure of $\mathsf{unreducedBRW}$. These results are required in the proofs of correctness and complexity of $\mathsf{EvalBRW}$. The second part proves the correctness of $\mathsf{EvalBRW}$ while the third part proves the complexity of $\mathsf{EvalBRW}$.

4.1 Structural Properties of $\mathsf{unreducedBRW}$

We start with the following simple result.

Lemma 1. *For $m \geq 0$, $\mathsf{BRW}_\tau(M_1, \ldots, M_m) = \mathsf{reduce}(\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_m))$.*

Proof. From the definition of BRW, we have the following two observations.

1. For $k \in \{4, 8, 16, 32, \ldots\}$, $\mathsf{BRW}_\tau(M_1, \ldots, M_k) = \mathsf{BRW}_\tau(M_1, \ldots, M_{k-1})(\tau^k + M_k)$.
2. For $k \in \{4, 8, 16, 32, \ldots\}$ and $k < m < 2k$, $\mathsf{BRW}_\tau(M_1, \ldots, M_m) = \mathsf{BRW}_\tau(M_1, \ldots, M_k) + \mathsf{BRW}_\tau(M_{k+1}, \ldots, M_m)$.

Using these two observations and (1), the result follows from the definition of $\mathsf{unreducedBRW}$ by induction on m. □

The next result is more complicated and forms the intuition behind the correctness of $\mathsf{EvalBRW}$.

Lemma 2. *Let $t \geq 2$ be an integer. For any $m \geq 2^t$, write*

$$\left\lfloor \frac{m}{2^t} \right\rfloor = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_s}, \quad (4)$$

where $k_1, \ldots, k_s$ are integers such that $k_1 > k_2 > \cdots > k_s \geq 0$. Let $K_0 = 0$, $K_1 = 2^{t+k_1}$, $K_2 = 2^{t+k_1} + 2^{t+k_2}, \ldots, K_s = 2^{t+k_1} + \cdots + 2^{t+k_s}$. Then

$$\begin{aligned}
&\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_m) \\
&= \mathsf{unreducedBRW}_\tau(M_{K_0+1}, \ldots, M_{K_1}) + \mathsf{unreducedBRW}_\tau(M_{K_1+1}, \ldots, M_{K_2}) \\
&\quad + \cdots + \mathsf{unreducedBRW}_\tau(M_{K_s+1}, \ldots, M_m). \qquad (5)
\end{aligned}$$

Proof. Let $r = m \bmod 2^t$. For $1 \leq i \leq s$, $K_i - K_{i-1} = 2^{t+k_i}$ and $m - K_s = m - 2^t(2^{k_1} + 2^{k_2} + \cdots + 2^{k_s}) = r$. So, the first s terms on the right hand side of (5) consist of $\mathsf{unreducedBRW}$ on $2^{t+k_1}, 2^{t+k_2}, \ldots, 2^{t+k_s}$ blocks respectively while the last term on the right hand side of (5) consists of $\mathsf{unreducedBRW}$ on r blocks. If $r = 0$, then the last term is not present. As a result, the number of terms on the right hand side of (5) equals s or $s+1$ according as 2^t divides m or not.

The proof of (5) is by induction on $s \geq 1$.

Base step: For $s = 1$, $\lfloor m/2^t \rfloor = 2^{k_1}$ and $K_1 = 2^{t+k_1}$. So, $2^{t+k_1} \leq m < 2^{t+k_1} + 2^t \leq 2^{t+k_1+1}$ and we can write

$$\begin{aligned}
&\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_m) \\
&= \mathsf{unreducedBRW}_\tau(M_1, \ldots, M_{2^{t+k_1}}) + \mathsf{unreducedBRW}_\tau(M_{2^{t+k_1}+1}, \ldots, M_m) \\
&= \mathsf{unreducedBRW}_\tau(M_{K_0+1}, \ldots, M_{K_1}) + \mathsf{unreducedBRW}_\tau(M_{K_1+1}, \ldots, M_m).
\end{aligned}$$

This proves the base case.

Induction step: Fix $s > 1$ and suppose that (5) holds for all $m \geq 2^t$ such that $\lfloor m/2^t \rfloor$ is the sum of $s - 1$ powers of two.

Now consider m such that $\lfloor m/2^t \rfloor = 2^{k_1} + \cdots + 2^{k_s}$ with $k_1 > k_2 > \cdots > k_s \geq 0$. Since $s > 1$, $m > 2^{t+k_1} = K_1$. From $k_1 > k_2 > \cdots > k_s \geq 0$, it follows that $k_i \leq k_1 - i + 1$ for $i \geq 2$; and $k_1 \geq s - 1$. So,

$$\begin{aligned}
m &= 2^t(2^{k_1} + 2^{k_2} + \cdots + 2^{k_s}) + r \\
&< 2^{t+k_1} + 2^{t+k_2} + 2^{t+k_3} + \cdots + 2^{t+k_s} + 2^t \qquad (\text{as } r = m \bmod 2^t) \\
&\leq 2^{t+k_1} + 2^{t+k_1-1} + 2^{t+k_1-2} + \cdots + 2^{t+k_1-(s-1)} + 2^t \\
&= 2^{t+k_1} + 2^{t+k_1-s+1}(2^{s-2} + 2^{s-3} + \cdots + 2^1 + 1) + 2^t \\
&= 2^{t+k_1} + 2^{t+k_1-s+1}(2^{s-1} - 1) + 2^t \\
&\leq 2^{t+k_1} + 2^{t+k_1} - 2^t + 2^t \qquad (\text{as } k_1 - s + 1 \geq 0) \\
&= 2^{t+k_1+1}.
\end{aligned}$$

So, we have $K_1 = 2^{t+k_1} < m < 2^{t+k_1+1} = 2K_1$. Since $t \geq 2$ and $m \geq 2^t \geq 4$, it follows from the definition of $\mathsf{unreducedBRW}$ that

$$\begin{aligned}
&\mathsf{unreducedBRW}_\tau(M_1, M_2, \ldots, M_m) \\
&= \mathsf{unreducedBRW}_\tau(M_1, \ldots, M_{K_1}) + \mathsf{unreducedBRW}_\tau(M_{K_1+1}, \ldots, M_m). \qquad (6)
\end{aligned}$$

Let $m' = m - K_1 = 2^{t+k_2} + \cdots + 2^{t+k_s} + r$ and note that $(M_{K_1+1}, \ldots, M_m)$ consists of m' blocks. Also, $\lfloor m'/2^t \rfloor = 2^{k_2} + \cdots + 2^{k_s}$, i.e., $\lfloor m'/2^t \rfloor$ can be written as sum of $s - 1$ powers of two. Since $s > 1$, we have $s - 1 \geq 1$ which implies that $m' \geq 2^t$. So, we can apply the induction hypothesis to $\mathsf{unreducedBRW}_\tau(M_{K_1+1}, \ldots, M_m)$ to obtain

$$\begin{aligned}
&\mathsf{unreducedBRW}_\tau(M_{K_1+1}, \ldots, M_m) \\
&= \mathsf{unreducedBRW}_\tau(M_{K_1+1}, \ldots, M_{K_2}) + \cdots + \mathsf{unreducedBRW}_\tau(M_{K_s+1}, \ldots, M_m). \qquad (7)
\end{aligned}$$

Combining (7) with (6) gives the desired result. $\square$

The next result determines the number of $\mathsf{unreducedMult}$ and reduce operations required in the evaluation of $\mathsf{unreducedBRW}$. These counts are independent of $\mathsf{EvalBRW}$ and are obtained from the recursive definition of $\mathsf{unreducedBRW}$.

Lemma 3. *For $m \geq 1$, evaluating $\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_m)$ requires $\lfloor m/2 \rfloor$ $\mathsf{unreducedMult}$ operations and $\lfloor m/4 \rfloor$ reduce operations. Additionally, for $m > 2$, $\lfloor \lg m \rfloor$ squarings are required to compute the relevant powers of τ.*

Proof. From the definition of $\mathsf{unreducedBRW}$, the statement is clearly true for $m = 1, 2, 3$.

For $m \geq 4$, the proof follows by induction on m.

If $m = 2^\ell$, then from the definition of $\mathsf{unreducedBRW}$ the number of $\mathsf{unreducedMult}$ (resp. reduce) operations is $1 + \lfloor (2^\ell - 1)/2 \rfloor = 2^{\ell-1}$ (resp. $1 + \lfloor (2^\ell - 1)/4 \rfloor = 2^{\ell-2}$).

If m is not a power of two, then m can be written as $m = 2^\ell + m_1$ with $m_1 < 2^\ell$. In this case, from the definition of $\mathsf{unreducedBRW}$ the number of $\mathsf{unreducedMult}$ (resp. reduce) operations is $2^{\ell-1} + \lfloor m_1/2 \rfloor = \lfloor m/2 \rfloor$ (resp. $2^{\ell-2} + \lfloor m_1/4 \rfloor = \lfloor m/4 \rfloor$). $\square$

4.2 Correctness of EvalBRW

For the correctness proof of $\mathsf{EvalBRW}$ some preliminary results are required.

Lemma 4. *For $m > 2$, Steps 3 to 7 of $\mathsf{EvalBRW}$ ensure that $\mathsf{keyPow}[j] = \tau^{2^j}$ for $j = 1, \ldots, \lfloor \lg m \rfloor$.*

Lemma 5. *Let $t \geq 2$ and $m \geq 2^t$. Let the loop counter $i \in \{1, \ldots, i_{\max}\}$, with $i_{\max} = \lfloor m/2^t \rfloor$, in Step 14 of $\mathsf{EvalBRW}$ be written as*

$$i \quad = \quad 2^{k_{i,1}} + 2^{k_{i,2}} + \cdots + 2^{k_{i,s_i}} \tag{8}$$

where $k_{i,1} > k_{i,2} > \cdots > k_{i,s_i} \geq 0$. Let $K_{i,0} = 0$, $K_{i,1} = 2^{t+k_{i,1}}, K_{i,2} = 2^{t+k_{i,1}} + 2^{t+k_{i,2}}, \ldots, K_{i,s_i} = 2^{t+k_{i,1}} + \cdots + 2^{t+k_{i,s_i}}$. After i iterations of the loop given by Steps 14 to 26, the following properties hold. For $l \in \{1, \ldots, s_i\}$,

$$\begin{aligned} \mathsf{isDef}[j] &= \begin{cases} 1 & \text{if } j = 1 + k_{i,l}; \\ 0 & \text{otherwise.} \end{cases} \\ \mathsf{res}[1 + k_{i,l}] &= \mathsf{unreducedBRW}_\tau(M_{K_{i,l-1}+1}, \ldots, M_{K_{i,l}}). \end{aligned}$$

Proof. The proof is by induction on $i \geq 1$.

Base step: For $i = 1$, $s_i = 1$, $k_{i,1} = 0$ and $K_{i,1} = 2^t$. The entries of the array isDef are set to 0 before Step 14. So, the **while** loop given by Steps 17 to 20 is not executed and in Step 22, j has the value it was assigned in Step 16 which is 1. As a result, $\mathsf{isDef}[1] = \mathsf{isDef}[1 + k_{i,1}]$ is set to 1 in Step 22 and all other entries of isDef remain 0.

In Step 15, $\mathsf{res}[0]$ is set to $\mathsf{unreducedBRW}_\tau(M_1, \cdots, M_{2^t-1})$ and in Step 16, tmp is set to $\mathsf{res}[0]$. In Step 21, the value of j is 1 and $\mathsf{res}[1] = \mathsf{res}[1+k_{i,1}]$ is set to the value $\mathsf{unreducedMult}(\mathsf{reduce}(\mathsf{tmp}), M_{2^t} + \mathsf{keyPow}[t])$. The correctness of this value is seen from the following simple computation.

$$\begin{aligned} &\mathsf{unreducedMult}(\mathsf{reduce}(\mathsf{tmp}), M_{2^t} + \mathsf{keyPow}[t]) \\ &\quad = \mathsf{unreducedMult}(\mathsf{reduce}(\mathsf{res}[0]), M_{2^t} + \tau^{2^t}) \quad \text{(from Lemma 4)} \\ &\quad = \mathsf{unreducedMult}(\mathsf{reduce}(\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_{2^t-1})), M_{2^t} + \tau^{2^t}) \\ &\quad = \mathsf{unreducedBRW}_\tau(M_1, \ldots, M_{2^t}) \quad \text{(from the definition of } \mathsf{unreducedBRW}\text{)} \\ &\quad = \mathsf{unreducedBRW}_\tau(M_{K_{i,0}+1}, \ldots, M_{K_{i,1}}). \end{aligned}$$

Inductive step: Suppose that the lemma holds for $i = 2^{k_{i,1}} + 2^{k_{i,2}} + \cdots + 2^{k_{i,s_i}} \geq 1$. We have to show that it holds for $i + 1$. Note that

$$\begin{aligned} i+1 &= 2^{k_{i,1}} + 2^{k_{i,2}} + \cdots + 2^{k_{i,s_i}} + 1 \\ &= 2^{k_{i+1,1}} + 2^{k_{i+1,2}} + \cdots + 2^{k_{i+1,s_{i+1}}}. \end{aligned}$$

Below we derive the expressions for $k_{i+1,1}, \ldots, k_{i+1,s_{i+1}}$ in terms of $k_{i,1}, \ldots, k_{i,s_i}$.

By the induction hypothesis, after i iterations, for $l = 1, \ldots, s_i$, $\mathsf{isDef}[j] = 1$ if and only if $j = 1 + k_{i,l}$ and $\mathsf{res}[1 + k_{i,l}] = \mathsf{unreducedBRW}_\tau(M_{K_{i,l-1}+1}, \ldots, M_{K_{i,l}})$.

There are two cases.

Case i is even: In this case $k_{i,s_i} > 0$ and so $s_{i+1} = s_i + 1$, $k_{i+1,1} = k_{i,1}, \ldots, k_{i+1,s_i} = k_{i,s_i}$ and $k_{i+1,s_{i+1}} = 0$ resulting in $K_{i+1,l} = K_{i,l}$ for $l = 1, \ldots, s_i$ and $K_{i+1,s_{i+1}} = K_{i,s_i} + 2^t$.

Since $k_{i,s_i} > 0$, at the end of the i-th iteration, $\mathsf{isDef}[1] = 0$ and so in the $(i+1)$-st iteration, the **while** loop in Steps 17 to 20 is not executed. As a result, in Step 22, $\mathsf{isDef}[1] = \mathsf{isDef}[1+k_{i+1,s_{i+1}}]$ is set to 1. No other value of isDef is changed. So, the stated conditions on isDef after $i+1$ iterations hold.

By a similar reasoning, at the end of the $(i+1)$st iteration, $\mathsf{res}[1] = \mathsf{res}[1 + k_{i+1,s_{i+1}}]$ gets set to $\mathsf{unreducedBRW}_\tau(M_{K_{i+1,s_i}+1}, \ldots, M_{K_{i+1,s_{i+1}}})$. No other value of res changes and so the stated conditions on res after $i + 1$ iterations hold.

Case i is odd: In this case $k_{i,s_i} = 0$. Let $\beta \in \{1, \ldots, s_i\}$ be the minimum value such that $k_{i,\beta} = s_i - \beta$. Since $k_{i,\beta} > k_{i,\beta+1} > \cdots > k_{i,s_i}$, it follows that for $l = \beta, \ldots, s_i - 1$, $k_{i,l} = s_i - l$. Further, if $\beta > 1$, then $k_{i,\beta-1} > k_{i,\beta} + 1 = s_i - \beta + 1$. So,

$$i = 2^{k_{i,1}} + \cdots + 2^{k_{i,\beta-1}} + 2^{s_i-\beta+1} - 1 \quad \text{and} \quad i+1 = 2^{k_{i,1}} + \cdots + 2^{k_{i,\beta-1}} + 2^{s_i-\beta+1}.$$

Consequently, $s_{i+1} = \beta$ and $k_{i+1,l} = k_{i,l}$ for $l = 1, \ldots, \beta - 1$ and $k_{i+1,\beta} = s_i - \beta + 1$.

From the induction hypothesis, after the ith iteration, $\mathsf{isDef}[1 + k_{i,l}] = 1$ for $l = 1, \ldots, s_i$ and 0 elsewhere. This in particular means that $\mathsf{isDef}[1] = \mathsf{isDef}[2] = \cdots = \mathsf{isDef}[1 + s_i - \beta] = 1$ and $\mathsf{isDef}[2 + s_i - \beta] = 0$ after the ith iteration. So, during the $(i + 1)$st iteration, at the end of the **while** loop given by Steps 17 to 20, the value of j is $2 + s_i - \beta$. This results in setting $\mathsf{isDef}[1 + k_{i+1,\beta}] = \mathsf{isDef}[2 + s_i - \beta] = 1$. The **for** loop given by Steps 23 to 25 results in setting the values of $\mathsf{isDef}[0], \ldots, \mathsf{isDef}[1+s_i-\beta]$ to 0. No other value of isDef is changed. As a result, at the end of the $(i+1)$st iteration, $\mathsf{isDef}[1+k_{i+1,l}] = \mathsf{isDef}[1+k_{i,l}] = 1$ for $l = 1, \ldots, \beta-1$; $\mathsf{isDef}[1+k_{i+1,\beta}] = 1$ and all other positions of isDef contain 0. This shows that the stated conditions on isDef after $i+1$ iterations hold.

From the values of $k_{i+1,1}$ to $k_{i+1,\beta}$, it follows that $K_{i+1,0} = K_{i,0} = 0, K_{i+1,1} = K_{i,1}, \ldots, K_{i+1,\beta-1} = K_{i,\beta-1}$ and $K_{i+1,\beta} = K_{i+1,\beta-1} + 2^{s_i-\beta+t+1}$. As a result, for $l = 1, \ldots, \beta - 1$,

$$\begin{aligned} \mathsf{res}[1 + k_{i+1,l}] &= \mathsf{res}[1 + k_{i,l}] \\ &= \mathsf{unreducedBRW}_\tau(M_{K_{i,l-1}+1}, \ldots, M_{K_{i,l}}) \\ &= \mathsf{unreducedBRW}_\tau(M_{K_{i+1,l-1}+1}, \ldots, M_{K_{i+1,l}}). \end{aligned}$$

So, we only need to argue that $\mathsf{res}[1+k_{i+1,\beta}]$ contains $\mathsf{unreducedBRW}_\tau(M_{K_{i+1,\beta-1}+1}, \ldots, M_{K_{i+1,s_{i+1}}})$ which is an application of $\mathsf{unreducedBRW}$ on $2^{s_i-\beta+t+1}$ blocks.

Since the total number of blocks processed up to and including the ith iteration is $i \cdot 2^t$, we have $K_{i,s_i} = i \cdot 2^t$ and similarly, $K_{i+1,s_{i+1}} = (i + 1)2^t$. After Step 15, in $(i + 1)$st iteration,

$$\begin{aligned} \mathsf{res}[0] &= \mathsf{unreducedBRW}_\tau(M_{2^t \cdot i+1}, \ldots, M_{2^t(i+1)-1}) \\ &= \mathsf{unreducedBRW}_\tau(M_{K_{i,s_i}+1}, \ldots, M_{K_{i+1,s_{i+1}}-1}). \end{aligned}$$

Note that

$$\begin{array}{rcl} K_{i,\beta} & = & K_{i,\beta-1} + 2^{t+k_{i,\beta}} = K_{i,\beta-1} + 2^{t+s_i-\beta} \\ K_{i,\beta+1} & = & K_{i,\beta} + 2^{t+k_{i,\beta+1}} = K_{i,\beta} + 2^{t+s_i-\beta-1} \\ \cdots & . & \cdots \\ K_{i,s_i} & = & K_{i,s_i-1} + 2^{t+k_{i,s_i}} = K_{i,s_i-1} + 2^t. \end{array}$$

From the induction hypothesis, at the end of the ith step

$$\begin{array}{rcl} \mathsf{res}[1+s_i-\beta] & = & \mathsf{res}[1+k_{i,\beta}] = \mathsf{unreducedBRW}_\tau(M_{K_{i,\beta-1}+1}, \ldots, M_{K_{i,\beta}}) \\ \mathsf{res}[1+s_i-\beta-1] & = & \mathsf{res}[1+k_{i,\beta+1}] = \mathsf{unreducedBRW}_\tau(M_{K_{i,\beta}+1}, \ldots, M_{K_{i,\beta+1}}) \\ \cdots & . & \cdots \\ \mathsf{res}[1+s_i-s_i] & = & \mathsf{res}[1+k_{i,s_i}] = \mathsf{unreducedBRW}_\tau(M_{K_{i,s_i-1}+1}, \ldots, M_{K_{i,s_i}}). \end{array}$$

In the $(i+1)$st iteration, at the end of the **while** loop given by Steps 17 to 20, the variable tmp contains the sum $\mathsf{res}[0]+\cdots+\mathsf{res}[1+s_i-\beta]$; in Step 21, $\mathsf{reduce}(\mathsf{tmp})$ is multiplied (without reduction) to $(M_{2^t(i+1)} + \tau^{2^{1+s_i-\beta+t}})$ and the value is assigned to $\mathsf{res}[2+s_i-\beta]$. We have

$$\begin{array}{rl} & \mathsf{res}[0] + \mathsf{res}[1] + \cdots + \mathsf{res}[1+s_i-\beta] \\ = & \mathsf{unreducedBRW}_\tau(M_{K_{i,s_i}+1}, \ldots, M_{K_{i+1,s_{i+1}}-1}) \\ & +\mathsf{unreducedBRW}_\tau(M_{K_{i,s_i-1}+1}, \ldots, M_{K_{i,s_i}}) \\ & + \cdots + \mathsf{unreducedBRW}_\tau(M_{K_{i,\beta-1}+1}, \ldots, M_{K_{i,\beta}}) \\ = & \mathsf{unreducedBRW}_\tau(M_{K_{i,\beta-1}+1}, \ldots, M_{K_{i+1,s_{i+1}}-1}) \quad \text{(from Lemma 2).} \end{array}$$

So, at the end of $(i+1)$st iteration,

$$\begin{array}{rcl} \mathsf{res}[1+k_{i+1,\beta}] & = & \mathsf{res}[2+s_i-\beta] \\ & = & \mathsf{unreducedMult}(\mathsf{reduce}(\mathsf{res}[0] + \cdots + \mathsf{res}[1+s_i-\beta]), M_{2^t(i+1)} + \tau^{2^{1+s_i-\beta+t}}) \\ & = & \mathsf{unreducedMult}(\mathsf{reduce}(\mathsf{unreducedBRW}_\tau(M_{K_{i,\beta-1}+1}, \ldots, M_{K_{i+1,s_{i+1}}-1})), \\ & & \qquad M_{K_{i+1,s_{i+1}}} + \tau^{2^{1+s_i-\beta+t}}) \\ & \overset{(a)}{=} & \mathsf{unreducedBRW}_\tau(M_{K_{i,\beta-1}+1}, \ldots, M_{K_{i+1,s_{i+1}}}) \\ & = & \mathsf{unreducedBRW}_\tau(M_{K_{i+1,\beta-1}+1}, \ldots, M_{K_{i+1,\beta}}). \end{array}$$

The equality in Step (a) above follows from the definition of $\mathsf{unreducedBRW}$.

This completes the induction step and the proof. □

Next we prove the correctness of $\mathsf{EvalBRW}$.

Theorem 1. *For any $t \geq 2$ and any $m \geq 1$, $\mathsf{EvalBRW}(\tau, M_1, \ldots, M_m)$ correctly computes $\mathsf{BRW}_\tau(M_1, \cdots, M_m)$.*

Proof. In Step 34, $\mathsf{EvalBRW}$ returns $\mathsf{reduce}(\mathsf{tmp})$. From Lemma 1, it follows that it is sufficient to show that tmp in Step 34 is equal to $\mathsf{unreducedBRW}_\tau(M_1, \cdots, M_m)$.

If $m < 2^t$, then the **for** loop from Step 14 to 26 is not executed. In Step 28, tmp gets assigned to $\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_m)$ which remains unchanged till Step 34. This proves the result for $m < 2^t$.

So, suppose $m \geq 2^t$ and as in Lemma 2, let $\lfloor m/2^t \rfloor$ be written as $\lfloor m/2^t \rfloor = 2^{k_1} + \cdots + 2^{k_s}$ and $K_0 = 0$, $K_1 = 2^{t+k_1}$, $K_2 = 2^{t+k_1} + 2^{t+k_2}, \ldots, K_s = 2^{t+k_1} + \cdots + 2^{t+k_s}$. Let $r = m \bmod 2^t$ so that $m = 2^t(2^{k_1} + \cdots + 2^{k_s}) + r$ implying $K_s = m - r$. From Lemma 2, we can write

$$\begin{aligned}
&\mathsf{unreducedBRW}_\tau(M_1, \ldots, M_m) \\
&\quad= \mathsf{unreducedBRW}_\tau(M_{K_0+1}, \ldots, M_{K_1}) + \cdots + \mathsf{unreducedBRW}_\tau(M_{K_{s-1}+1}, \ldots, M_{K_s}) \\
&\qquad+\mathsf{unreducedBRW}_\tau(M_{K_s+1}, \ldots, M_m) \\
&\quad= \mathsf{unreducedBRW}_\tau(M_{K_0+1}, \ldots, M_{K_1}) + \cdots + \mathsf{unreducedBRW}_\tau(M_{K_{s-1}+1}, \ldots, M_{K_s}) \\
&\qquad+\mathsf{unreducedBRW}_\tau(M_{m-r+1}, \ldots, M_m). \qquad (9)
\end{aligned}$$

The loop counter i of the **for** loop in Step 14 runs from 1 to $i_{\max} = \lfloor m/2^t \rfloor$. Applying Lemma 5 to $i_{\max}$, we obtain that after $i_{\max}$ iterations, for $l \in \{1, \ldots, s\}$,

$$\begin{aligned}
\mathsf{isDef}[j] &= \begin{cases} 1 & \text{if } j = 1 + k_l; \\ 0 & \text{otherwise.} \end{cases} \\
\mathsf{res}[1 + k_l] &= \mathsf{unreducedBRW}_\tau(M_{K_{l-1}+1}, \ldots, M_{K_l}).
\end{aligned}$$

From $m = 2^t(2^{k_1}+\cdots+2^{k_s})+r$ and $0 \leq r < 2^t$, we have $2^t(2^{k_1}+\cdots+2^{k_s}) \leq m < 2^t(2^{k_1}+\cdots+2^{k_s}+1)$. Since $k_1 > k_2 > \cdots > k_s$, we obtain $2^{t+k_1} \leq m < 2^{t+k_1+1}$ and so $k_1 = \lfloor \lg m \rfloor - t$. As a result, we get that the maximum positions of isDef and res that are accessed by the algorithm are both equal to $1 + \lfloor \lg m \rfloor - t$. This justifies the upper bound on the loop counter of the **for** loops in Steps 10 and 29.

In Step 28, tmp is initialised to $\mathsf{unreducedBRW}_\tau(M_{m-r+1}, \ldots, M_m)$. For $j = 1, \ldots, 1+\lfloor \lg m \rfloor - t$, the **for** loop in Steps 29 to 33 adds $\mathsf{res}[j]$ to tmp if and only if $\mathsf{isDef}[j] = 1$. As a result, after this **for** loop

$$\begin{aligned}
\mathsf{tmp} &= \mathsf{unreducedBRW}_\tau(M_{K_0+1}, \ldots, M_{K_1}) + \cdots + \mathsf{unreducedBRW}_\tau(M_{K_{s-1}+1}, \ldots, M_{K_s}) \\
&\quad+\mathsf{unreducedBRW}_\tau(M_{m-r+1}, \ldots, M_m).
\end{aligned}$$

From (9), we have that in Step 34, $\mathsf{tmp} = \mathsf{unreducedBRW}_\tau(M_1, \ldots, M_m)$ as desired. □

4.3 Complexity of EvalBRW

The space complexity of EvalBRW is determined by the maximum sizes of the arrays res, isDef and keyPow (plus a constant number of variables). The sizes of these arrays are determined in the proof of Theorem 1 and we record these in the following result.

Proposition 1. *Let $t \geq 2$. For correctly computing $\mathsf{BRW}_\tau(M_1, \ldots, M_m)$, it is sufficient for the arrays* isDef *and* res *to have length $\lfloor \lg m \rfloor - t + 2$. Further, for $m > 2$,* keyPow *stores $1 + \lfloor \lg m \rfloor$ elements of $\mathbb{F}$.*

Proof. The proof of Theorem 1 shows that the maximum positions of isDef and res that are accessed are both equal to $\lfloor \lg m \rfloor - t + 1$. Since both the arrays start from position 0, the length is $\lfloor \lg m \rfloor - t + 2$. □

We now consider the time complexity of $\mathsf{EvalBRW}(\tau, M_1, \ldots, M_m)$. For this, we separately count the number of unreducedMult and reduce operations that are required.

Theorem 2. *For $m \geq 2$,* $\mathsf{EvalBRW}(\tau, M_1, \ldots, M_m)$ *requires* $\lfloor m/2 \rfloor$ unreducedMult *operations and* $1 + \lfloor m/4 \rfloor$ reduce *operations. Additionally, for $m > 2$, $\lfloor \lg m \rfloor$ squarings are required to compute the relevant powers of τ.*

Proof. The final output returned by $\mathsf{EvalBRW}(\tau, M_1, \ldots, M_m)$ at Step 34 is reduce(tmp). So, it is sufficient to show that computing tmp up to Step 34 requires $\lfloor m/2 \rfloor$ unreducedMult operations and $\lfloor m/4 \rfloor$ reduce operations.

The call to unreducedBRW in Step 15 is on $2^t - 1$ blocks while the call to unreducedBRW in Step 28 is on r blocks. From Lemma 3, we have that for any fixed $t \geq 2$, a straight line code to compute $\mathsf{unreducedBRW}_\tau$ in Step 15 requires $2^{t-1} - 1$ unreducedMult operations and $2^{t-2} - 1$ reduce operations. Similarly, the call to $\mathsf{unreducedBRW}_\tau$ in Step 28 requires $\lfloor r/2 \rfloor$ unreducedMult operations and $\lfloor r/4 \rfloor$ reduce operations.

Let $i_{\max} = \lfloor m/2^t \rfloor$. The counter of the **for** loop in Step 14 runs up to $i_{\max}$. Then $m = i_{\max} \cdot 2^t + r$, where $r = m \bmod 2^t$ is computed in Step 27. Since $t \geq 2$, we have $(m - r)/2 = i_{\max} \cdot 2^{t-1}$ and $(m - r)/4 = i_{\max} \cdot 2^{t-2}$ to be both integers.

First consider the number of unreducedMult operations required for computing tmp at Step 34. Each iteration of the **for** loop given by Steps 14 to 26 makes one call to unreducedBRW on $2^t - 1$ blocks and one unreducedMult operation. The call to unreducedBRW operations requires $2^{t-1} - 1$ unreducedMult operations. So, the number of unreducedMult operations required during the entire **for** loop given by Steps 14 to 26 is $i_{\max}(1 + 2^{t-1} - 1)$. Additionally, $\lfloor r/2 \rfloor$ unreducedMult operations are required by the call to unreducedBRW in Step 28. As a result, the total number of unreducedMult operations required to compute tmp required at Step 34 is

$$i_{\max} \cdot 2^{t-1} + \lfloor r/2 \rfloor \quad = \quad (m - r)/2 + \lfloor r/2 \rfloor = \lfloor m/2 \rfloor.$$

Since $(m - r)/2$ is an integer, m and r are either both even or both odd. If both are even, then the last equality is immediate while if they are both odd, then writing $(m - r)/2 = (m - 1)/2 - (r - 1)/2$ shows the last equality.

For the number of reduce operations required for computing tmp at Step 34, a reasoning similar to above shows that the required number is

$$i_{\max} \cdot 2^{t-2} + \lfloor r/4 \rfloor \quad = \quad (m - r)/4 + \lfloor r/4 \rfloor = \lfloor m/4 \rfloor.$$

In this case, for the last equality, we have to use the fact that $(m - r)/4$ is an integer implies that $m \equiv r \bmod 4$ so that if $m \bmod 4 = a = r \bmod 4$, then $(m - r)/4 + \lfloor r/4 \rfloor = (m - a)/4 - (r - a)/4 + (r - a)/4 = (m - a)/4 = \lfloor m/4 \rfloor$. □

The role of the parameter t: Note that from Theorem 2, the number of operations required by EvalBRW does not depend on t. The parameter t determines the number of blocks in the unreducedBRW call at Step 15. As mentioned earlier, it is assumed that there is a subroutine for this computation and the subroutine performs this computation as a straight line code without any loop. In other words, the value of t determines the extent of loop unrolling. To some extent, using a greater amount of loop unrolling leads to improved efficiency as indicated by the results of our implementations.

5 Design of Hash Function

We propose a hash function based on BRW polynomials. For convenience of the description, we define the following terminology.

$\mathsf{len}(X)$: For a binary string X, its length will be denoted as $\mathsf{len}(X)$.

$\mathsf{bin}_n(i)$: For an integer i with $0 \leq i < 2^n$, $\mathsf{bin}_n(i)$ denotes the n-bit binary representation of i.

$\mathsf{pad}_n(X)$: For a binary string X and $n > 0$, if X is the empty string, then $\mathsf{pad}_n(X)$ will denote the string 0^n; while if X is non-empty, then $\mathsf{pad}_n(X)$ will denote $X||0^i$, where $i \geq 0$ is the minimum integer such that n divides $\mathsf{len}(X||0^i)$.

$\mathsf{format}_n(X)$: For a positive integer n and a non-empty binary string X whose length is a multiple of n, $\mathsf{format}_n(X)$ denotes $(X_1, X_2, \ldots, X_m)$ where $X = X_1||X_2||\cdots||X_m$, $m = \mathsf{len}(X)/n$, $\mathsf{len}(X_i) = n$ for $1 \leq i \leq m$. In other words, $\mathsf{format}_n(X)$ divides the string X into m n-bit blocks $X_1, \ldots, X_m$.

The hash function that we define uses BRW computation over $GF(2^n)$. Using a fixed irreducible polynomial over $GF(2)$ of degree n, it is possible to identify the elements of $GF(2^n)$ with the elements of $\{0,1\}^n$. In the following, we will implicitly assume this identification.

For a positive integer n, let

$$\mathcal{D} = \bigcup_{i=0}^{2^n-1} \{0,1\}^i. \tag{10}$$

We define the hash function $\mathsf{BRW}n$ in the following manner.

$$\mathsf{BRW}n : \{0,1\}^n \times \mathcal{D} \to \{0,1\}^n. \tag{11}$$

The computation of $\mathsf{BRW}n$ is shown in Algorithm 2.

Algorithm 2 Computation of BRW based hash function.

```
1: function $\mathsf{BRW}n(\tau, X)$
2:     $(X_1, \ldots, X_m) \leftarrow \mathsf{format}_n(\mathsf{pad}_n(X))$;
3:     $Y \leftarrow \mathsf{EvalBRW}(\tau, X_1, \ldots, X_m)$;
4:     $Z \leftarrow \tau(\tau Y \oplus \mathsf{bin}_n(\mathsf{len}(X)))$;
5:     return $Z$;
6: end function.
```

We will write $\mathsf{BRW}n_\tau(\cdot)$ to denote $\mathsf{BRW}n(\tau, \cdot)$.

The following result shows that the differential probability of $\mathsf{BRW}n$ is small.

Proposition 2. *Let $X, X' \in \mathcal{D}$, $X \neq X'$ and $\alpha \in GF(2^n)$. Then for a uniform random $\tau \in GF(2^n)$,*

$$\Pr[\mathsf{BRW}n_\tau(X) \oplus \mathsf{BRW}n_\tau(X') = \alpha] \leq \frac{2\max(m, m') + 1}{2^n} \tag{12}$$

where m (resp. m') is the number of blocks in the output of $\mathsf{format}_n(\mathsf{pad}_n(X))$ (resp. $\mathsf{format}_n(\mathsf{pad}_n(X'))$).

Proof. Let $(X_1, \ldots, X_m) = \mathsf{format}_n(\mathsf{pad}_n(X))$, $Y = \mathsf{EvalBRW}(\tau, X_1, \ldots, X_m)$ and $\mathsf{BRW}n_\tau(X) = Z = \tau(\tau Y \oplus \mathsf{bin}_n(\mathsf{len}(X)))$. Similarly, let $(X'_1, \ldots, X'_{m'}) = \mathsf{format}_n(\mathsf{pad}_n(X'))$, $Y' = \mathsf{EvalBRW}(\tau, X'_1, \ldots, X'_{m'})$ and $\mathsf{BRW}n_\tau(X') = Z' = \tau(\tau Y' \oplus \mathsf{bin}_n(\mathsf{len}(X')))$.

Y (resp. Y') is a polynomial in τ of degree $\mathfrak{d}(m)$ (resp. $\mathfrak{d}(m')$). Consequently, Z (resp. Z') is a polynomial in τ of degree $\mathfrak{d}(m)+2$ (resp. $\mathfrak{d}(m')+2$). Assume without loss of generality $m \geq m'$, so that $\max(m, m') = m$.

Suppose that X and X' are of different lengths. Then the coefficients of τ in Z and Z' are different. Consequently $Z \oplus Z' \oplus \alpha$ is a non-zero polynomial of degree $\mathfrak{d}(m)+2$.

So, suppose that X and X' have the same length. Then $m = m'$. Since $X \neq X'$, it follows that $(X_1, \ldots, X_m) \neq (X'_1, \ldots, X'_m)$. By the injectivity of BRW (see Section 2), it follows that the polynomials Y and Y' are distinct. Consequently, $Z \oplus Z' \oplus \alpha$ is a non-zero polynomial of degree at most $\mathfrak{d}(m)+2$.

In both cases, we have $Z \oplus Z' \oplus \alpha$ to be a non-zero polynomial of degree at most $\mathfrak{d}(m)+2$. The probability that a uniform random τ from $GF(2^n)$ is a root of this polynomial is at most $(\mathfrak{d}(m)+2)/2^n \leq (2m+1)/2^n$. The last inequality follows from the fact that $\mathfrak{d}(m) \leq 2m-1$ (see Section 2). $\square$

6 Implementation

We report implementations of BRW128 and BRW256. These require the implementations of EvalBRW over $GF(2^n)$ in the two cases of $n = 128$ and $n = 256$. $GF(2^{128})$ was represented using the irreducible polynomial $\sigma(x) = x^{128} \oplus x^7 \oplus x^2 \oplus x \oplus 1$ and $GF(2^{256})$ was represented using the irreducible polynomial $\sigma(x) = x^{256} \oplus x^{10} \oplus x^5 \oplus x^2 \oplus 1$.

Our target platform for the implementation was the Intel Skylake processor, which supports the Intel Intrinsics instruction set. The instruction of particular interest for our implementation was `pclmulqdq`, which takes as input two polynomials over $GF(2)$ of degrees at most 63 each and returns their product which is a polynomial over $GF(2)$ of degree at most 126. The two input polynomials fit into 64-bit words while the output polynomial fits into a 128-bit word.

A field multiplication in $GF(2^n)$ consists of a unreducedMult followed by a reduce operation.

1. For each unreducedMult over $GF(2^{128})$, we need to compute the polynomial multiplication of two polynomials of degrees at most 127 each. Using the schoolbook method this requires 4 `pclmulqdq` instructions while using Karatsuba's algorithm it requires 3 `pclmulqdq` instructions and some additional XOR instructions. It has been reported in [5] that on the Skylake processor, the schoolbook method is faster and so we have used this method.

2. For each unreducedMult over $GF(2^{256})$, we need to compute the polynomial multiplication of two polymomials of degrees at most 255 each. For this, the schoolbook and Karatsuba's methods require 16 and 9 `pclmulqdq` instructions respectively. In this case, Karatsuba's method gives better performance [2] and so we have used this method.

For the reduce operation over $GF(2^{128})$, following the procedure described in [4] the reduction modulo $\sigma(x) = x^{128} \oplus x^7 \oplus x^2 \oplus x \oplus 1$ can be done using 2 `pclmulqdq` instructions along with some XORs and shifts. An extension of this procedure [2] for $GF(2^{256})$ shows that the reduction modulo $\sigma(x) = x^{256} \oplus x^{10} \oplus x^5 \oplus x^2 \oplus 1$ requires 4 `pclmulqdq` instructions along with some XORs and shifts.

EvalBRW has the parameter t. For the implementation, we have considered $t = 2, 3, 4$ and 5. This requires implementation of unreducedBRW as a straight line program for m blocks with $1 \leq m \leq 31$. Implementations of BRW for 1 to 31 blocks have been reported in [2]. Here we use these implementations with the modification that the final reduction is not applied so that unreducedBRW is computed instead of BRW.

Table 1: Timing results for BRW128 and POLYVAL.

	512	1024	4096	8192	16384	30000
$t = 2$	0.819	0.611	0.425	0.388	0.368	0.356
$t = 3$	0.826	0.623	0.444	0.407	0.389	0.379
$t = 4$	0.787	0.583	0.401	0.364	0.344	0.336
$t = 5$	0.776	0.552	0.348	0.309	0.287	0.278
POLYVAL	0.786	0.549	0.376	0.347	0.333	0.328

Table 2: Timing results for BRW256.

	512	1024	4096	8192	16384	30000
$t = 2$	1.162	0.909	0.675	0.628	0.603	0.587
$t = 3$	1.118	0.864	0.629	0.581	0.559	0.539
$t = 4$	1.099	0.841	0.607	0.558	0.533	0.519
$t = 5$	1.095	0.862	0.619	0.569	0.544	0.529

The code for our implementations of BRW128 and BRW256 is publicly available[1].

6.1 Timings

The timing measurements were taken on a single core of a machine with Intel Core i7-6500U Skylake @ 2.5GHz. The operating system was 64-bit Ubuntu-14.04-LTS and the C codes were complied using GCC version 4.8.4. For measuring time, we followed the strategy of [6].

The corresponding timing results that were obtained are shown in Tables 1 and 2. The column headers of the first row provide the message size in bytes. From the second row onwards, the rows are for different values of t. The entries in these rows are in cycles per byte.

For $n = 128$, we provide the timings of POLYVAL [5] for the purpose of comparison. The code for POLYVAL is essentially a highly optimised implementation of Horner's rule based polynomial hash. In particular, it performs a single reduction for every eight polynomial multiplications and the ordering of the instructions seems to have been done very carefully so as to minimise the cycle counts.

In contrast, we would like to clarify that we do not claim to have provided the best possible implementations of BRW128 and BRW256. We have considered the possible algorithmic improvements and the corresponding implementation in Intel intrinsics. For concrete speed-ups one also needs to consider details of instruction level pipelining issues and also possibly carry out an implementation in assembly. Since the main goal of our implementation was to show the practicability of Algorithm EvalBRW, we have not tried to aggresively optimise the code. Future implementation efforts may attempt such work.

7 Conclusion

In this work we have described an efficient non-recursive algorithm to evaluate BRW polynomials which works for any number of blocks. This algorithm has been used to define two concrete hash

[1]`https://github.com/sebatighosh/BRW`

functions. Implementations of the hash functions using instructions available on modern Intel processors show promising timing results making the hash functions worthy candidates for actual deployment.

Acknowledgement

We acknowledge with thanks several helpful discussions with Debrup Chakraborty.

References

[1] Daniel J. Bernstein. Polynomial evaluation and message authentication, 2007. `http://cr.yp.to/papers.html#pema`.

[2] Debrup Chakraborty, Sebati Ghosh, and Palash Sarkar. A fast single-key two-level universal hash function. *IACR Trans. Symmetric Cryptol.*, 2017(1):106–128, 2017.

[3] Debrup Chakraborty, Cuauhtemoc Mancillas-López, Francisco Rodríguez-Henríquez, and Palash Sarkar. Efficient hardware implementations of BRW polynomials and tweakable enciphering schemes. *IEEE Trans. Computers*, 62(2):279–294, 2013.

[4] Shay Gueron and Michael E. Kounavis. Efficient implementation of the Galois Counter Mode using a carry-less multiplier and a fast reduction algorithm. *Inf. Process. Lett.*, 110(14-15):549–553, 2010.

[5] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: specification and analysis. *IACR Cryptology ePrint Archive*, 2017:168, 2017.

[6] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[7] Michael O. Rabin and Shmuel Winograd. Fast evaluation of polynomials by rational preparation. *Communications on Pure and Applied Mathematics*, 25:433–458, 1972.

[8] Palash Sarkar. Efficient tweakable enciphering schemes from (block-wise) universal hash functions. *IEEE Trans. Information Theory*, 55(10):4749–4760, 2009.