# New Parallel Domain Extenders for UOWHF

Wonil Lee[1], Donghoon Chang[1], Sangjin Lee[1],
Soohak Sung[2], and Mridul Nandi[3]

[1] Center for Information and Security Technologies
Korea University, Seoul, Korea
{wonil,dhchang,sangjin}@cist.korea.ac.kr
[2] Applied Math. Department, Paichai University, Daejeon, Korea
sungsh@mail.paichai.ac.kr
[3] Applied Statistics Unit, Indian Statistical Institute, Kolkata, India
hi_mridul@yahoo.com

**Abstract.** We present two new parallel algorithms for extending the domain of a UOWHF. The first algorithm is complete binary tree based construction and has less key length expansion than Sarkar's construction which is the previously best known complete binary tree based construction. But only disadvantage is that here we need more key length expansion than that of Shoup's sequential algorithm. But it is not too large as in all practical situations we need just two more masks than Shoup's. Our second algorithm is based on non-complete $l$-ary tree and has the same optimal key length expansion as Shoup's which has the most efficient key length expansion known so far. Using the recent result [9], we can also prove that the key length expansion of this algorithm and Shoup's sequential algorithm are the minimum possible for any algorithms in a large class of "natural" domain extending algorithms. But its parallelizability performance is less efficient than complete tree based constructions. However if $l$ is getting larger, then the parallelizability of the construction is also getting near to that of complete tree based constructions. We also give a sufficient condition for valid domain extension in sequential domain extension.

**Keywords:** UOWHF, hash function, masking assignment, sequential construciton, parallel construction, tree based construction.

## 1 Introduction

Naor and Yung [7] introduced the notion of universal one-way hash function (UOWHF) to prove that secure digital signatures can be based on any 1-1 one-way function. A UOWHF is a family of functions $\{h_k\}_{k\in\mathcal{K}}$ for which the following task of the adversary is computationally infeasible. The adversary has to choose a $x$ from the domain, and then given a random $k \in \mathcal{K}$, he has to find a $y$ such that $x \neq y$ but $h_k(x) = h_k(y)$. Intuitively, a UOWHF is a weaker primitive than a collision resistant hash function (CRHF), since the task of the adversary is more difficult, i.e., the adversary has to commit to the string $x$ before knowing the actual hash function $h_k$ for which the collision has to be found. Furthermore, Simon [11] had shown that there is an oracle relative to which UOWHFs exist but not CRHFs.

A UOWHF is an attractive alternative to a CRHF because it seems that building an efficient and secure UOWHF is easier than building an efficient and secure CRHF, and in many applications, most importantly for building digital signature schemes, a UOWHF is sufficient. In addition, as mentioned in [1], the birthday attack does not apply to UOWHFs. Hence the size of the message digest can be significantly shorter.

A reasonable approach to designing a UOWHF that hashes messages of arbitrary and variable length is to first design a compression function, that is, a UOWHF that hashes fixed-length messages, and then design a method for composing these compression functions so as to hash arbitrary and variable messages. The present paper deals with the second problem, that of composing compression functions. We will call the composite method *construction* or *domain extender* for the most part in this paper. The main technical problem in designing such domain extender is to keep the key length of the domain extender from getting too large.

The rest of this paper is organized as follows. Motivation and our contributions are given in Section 2. Some detailed history of UOWHF is also provided in Section 2 in order to precisely explain our contributions. Preliminaries are given in Section 3. We will generalize Shoup's sequential construction in Section 4. In this section we also provide a sufficient condition for valid sequential domain extension. Then we will present our new complete binary tree based parallel domain extender and will give a proof of validness of the extension in Section 5. we will present our second new parallel domain extender which is based on non-complete $l$-ary tree and the proof of security in Section 6. In Section 7, we specifically compare the known constructions with our two constructions. This paper concludes with Section 8.

## 2 Motivation and Our Contribution

Most practical signature schemes follow "hash-and-sign" paradigm. They take a message $M$ of an arbitrary length and hash it to obtain a constant length string, which is then fed into a signing algorithm. Many schemes use CRHFs to hash a message $x$, but as it was first pointed out in [1] a UOWHF suffices for that purpose. Indeed, if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF, then to sign a message $x$, the signer chooses a random key $k$, and produces the signature $(k, \sigma(k, h_k(x)))$, where $\sigma$ is the underlying signing function for short messages.

Note that the key length varies with the length of input message for UOWHFs. Therefore, in many cases, the size of $(k, h_k(x))$ can be larger than the input size of $\sigma$. However, in these cases, we can solve the problem by applying the signing algorithm $\sigma$ to $(h_{K'}(k), h_k(x))$, where $K'$ is part of the signer's public key. Here the signature becomes $(k, \sigma(h_{K'}(k), h_k(x)))$. And note that the function $h_{K'}$ can be replaced by any second-preimage resistant function, because its input is random and chosen by the signer. Since messages can be very long, hashing speed is a crucial factor. On the other hand, a closer look at the signature scheme reveals that the key $k$ must be part of the signature so the receiver can recompute the hash. Therefore the shorter the key better the signature scheme.

These facts lead us to think we should consider two aspects.

1. Minimizing the key length expansion: This is certainly a very important aspect of any domain extending algorithm.
2. Parallel implementation: From an implementation point of view parallelizability is also an important aspect of any domain extending algorithm.

Bellare and Rogaway [1] suggested the XOR tree hash (XTH) construction in order to reduce the key length expansion. Since XTH is based on the complete (or full) $l$-ary tree($l \geq 2$), it has also an efficiency regarding the parallelizability (the processing speed). XTH had been the most efficient construction not only regarding the key length expansion but also regarding the parallelizability before Shoup's construction was presented in [10]. Shoup's construction is more efficient than XTH with regard to the key length expansion. Furthermore, Mironov [4] had shown that the key length expansion needed in Shoup's construction is the minimum possible for any sequential algorithm. In other words, there is no sequential algorithm which has more efficient key length expansion than Shoup's. But his construction is not more efficient than XTH with regard to the parallelizability since it is based on the uniary tree. In the following, 'B < A' means that A is more efficient than B regarding the key length expansion or parallelizability.

| | |
|---|---|
| Key length expansion: | XTH < Shoup |
| Parallelizability: | Shoup < XTH |

Sarkar's work [8] was an attempt to propose a parallel algorithm which has the following properties:

– The algorithm's key length expansion is as good as possible.
– The algorithm's parallelizable efficiency is the same as XTH.

Therefore, he also chose the complete tree to obtain the same parallelizable performance as XTH and chose binary structure to adopt both of the mask assignment methods of Shoup's and XTH algorithm so that the key length expansion can be reduced as much as possible. As a result, Sarkar's construction has the same parallalizable performance as XTH. However, his construction does not have the same key length expansion as Shoup's one.

| | |
|---|---|
| Key length expansion: | XTH < Sarkar < Shoup |
| Parallelizability: | Shoup < XTH = Sarkar |

In this paper we will first present a tree based domain extension whose key length expansion is significantly less than Sarkar's construction. Furthermore, its parallelizable efficiency is the same as Sarkar's since it is also based on the complete binary tree. It will be called Improved Binary Tree based Construction

(IBTC). In fact, we have got a lot of evidences in [6] that IBTC will be optimal in the class of complete binary tree based algorithm. But only disadvantage is that here we need more masks (part of the key) than sequential construction. But it is not too large as in all practical situations we need just two more masks than Shoup's construction.

However, note that all the previously proposed parallel algorithms, including our first new construction, took more key length expansion than that of Shoup's sequential algorithm. So an important question is whether this is true in general of any parallel algorithm. Our second new construction shows that this is not the case.

The following is our motivation to design the second new parallel algorithm. At the present stage, it seems that the parallel constructions based on the complete $l$-ary tree have the most efficient parallelizability. But we think it is difficult to construct the parallel domain extender which has the same key length expansion as Shoup's sequential domain extender if we can only use the complete $l$-ary tree. Therefore, we decide to take somewhat different approach as follows without the assumption that we can use only the complete $l$-ary tree: In contrast to [8] and our first construction, this work is an attempt to propose a parallel algorithm which has the following properties:

– The algorithm has the same key length expansion as Shoup's.
– The algorithm's parallelizable efficiency is as good as possible.

As a result, the second new construction has the same key length expansion as Shoup's one. But the construction does not have the same parallalizable performance with our first new construction. The construction will be called $l$-DIMensional construction ($l$-DIM, $l \geq 2$).

| | |
|---|---|
| Key length expansion: | XTH < Sarkar < **IBTC** < Shoup = **$l$-DIM** |
| Parallelizability: | Shoup < **$l$-DIM** < XTH = Sarkar = **IBTC** |

The results may be summarized as shown in Table 1 (Here, 'seq' means 'sequential' and 'par' means 'parallel'). A more detailed comparison is presented in Table 2 in Section 7.

**Table 1.** Comparison of domain extenders for UOWHF

| Method | Used Tree | Ranking of Key expansion | Ranking of Parallelizability | Seq /Par |
|---|---|---|---|---|
| BLH [1] | Unary | 7 | 3 | seq |
| XLH [1] | Unary | 6 | 3 | seq |
| Shoup [10] | Unary | 1 | 3 | seq |
| BTH [1] | Complete $l$-ary ($l \geq 2$) | 5 | 1 | par |
| XTH [1] | Complete $l$-ary ($l \geq 2$) | 4 | 1 | par |
| Sarkar [8] | Complete Binary | 3 | 1 | par |
| **IBTC**(this paper) | Complete Binary | 2 | 1 | par |
| **$l$-DIM**(this paper) | Non-Complete $l$-ary ($l \geq 2$) | 1 | 2 | par |

We think it is difficult to say that which one is more important than the other between the key length expansion and the parallel implementation. Of course, it would be very nice to have a regular parallel structure something like the complete tree which also minimizes the key length expansion. But at this point, we do not have any such algorithm and IBTC is the best known construction among the complete binary tree based constructions. Hence, in our opinion, we should separately consider both the above-mentioned two points of view with the same importance. And the present works are important in regarding the former and the latter point of views, respectively. Particularly, the $l$-DIM and Shoup's one are the only two known algorithms which minimize the key length expansion. In addition that, the reason why the $l$-DIM has more meaning is that it is a parallel algorithm which has the same key length expansion as Shoup's sequential algorithm and this is the very first trial in designing the parallel algorithms.

Using the recent result [9], we can also prove that the key length expansion of our new parallel construction and Shoup's sequential construction are the minimum possible for any constructions in a large class of "natural" domain extenders including all the previously proposed methods.

We also give a sufficient condition for valid domain extension for sequential construction and it is likely that the condition is necessary. So, that will characterize the valid domain extension for sequential construction. In [6] M. Nandi has also shown that the same condition becomes sufficient for general tree based domain extension.

**Related Work:** Note that all of the above described parallel constructions are based on the assumption that the number of processors grows with the length of the message. In [9], Sarkar has first suggested a parallel domain extending algorithm which can be implemented with finitely many processors. But it does not have the same key length expansion as Shoup's. Here, it should be noted that his work mainly focuses on the parallel implementation with finite processors, on the contrary, the present work focuses on the parallel implementation with optimal key length expansion. And it seems that using the technique of [9], our new parallel constructions can be modified to the constructions which can work with finite processors.

## 3 Preliminaries

The following notations are used in this paper.

1. $[a, b] = \{a, a+1, ..., b\}$ where $a$ and $b$ are integers.
2. Suppose $A$ is a finite set. By $a \in_R A$ we mean that $a$ is a uniform discrete random variable taking values from $A$.
3. $\nu_2(i) = j$ if $2^j | i$ and $2^{j+1} \not| i$.
4. For $t > 1$, $\log_2^m(t)$ means that the function $\log_2$ applies $m$ many times on $t$. $\log_2^*(t) = m$ if $\log_2^m(t) \leq 1$ but $\log_2^{m-1}(t) > 1$.

5. In the complete binary tree based construction, $T_t = (V_t, E_t)$ means the complete binary tree where $V_t = \{1, 2, ..., 2^t - 1\}$ is a node set and $E_t = \{e_i : 2 \leq i \leq 2^t - 1\}$ is a directed edge set where $e_i = (i, \lfloor i/2 \rfloor)$. Here $e_i = (v, w)$ denotes a directed edge, i.e., $v$ is the initial node and $w$ the terminal node. $ht_t(i) = j$ means that $2^{t-j} \leq i \leq 2^{t+1-j} - 1$. So, the root node has height $t$ and all leaves have height 1. For any node $i$, define $T_t[i]$ by the complete binary sub-tree rooted at $i$.
6. In the 4-dimensional construction, for integer $t$, $g(t) = (a, b, c, d)$, where $a = \lfloor t/4 \rfloor + \lfloor ((t \bmod 4) + 3)/4 \rfloor$, $b = \lfloor t/4 \rfloor + \lfloor ((t \bmod 4) + 2)/4 \rfloor$, $c = \lfloor t/4 \rfloor + \lfloor ((t \bmod 4) + 1)/4 \rfloor$, and $d = \lfloor t/4 \rfloor$. Here $t \bmod 4 = t - \lfloor t/4 \rfloor n$.
7. In the 4-dimensional construction let $T_t = (V_t, E_t)$ be a non-complete 4-ary tree, where $V_t = \{1, 2..., 2^t\}$ and $E_t = \{e_i : 2 \leq i \leq 2^t\}$ where $e_i = (i, i-1)$ for $2 \leq i \leq 2^a$, $e_i = (i, i - 2^a)$ for $2^a < i \leq 2^{a+b}$, $e_i = (i, i - 2^{a+b})$ for $2^{a+b} < i \leq 2^{a+b+c}$, and $e_i = (i, i - 2^{a+b+c})$ for $2^{a+b+c} < i \leq 2^t$. Here $a, b, c$, and $d$ are such that $g(t) = (a, b, c, d)$.

Let $\{h_k\}_{k \in \mathcal{K}}$ be a keyed family of hash functions, where each $h_k : \{0,1\}^n \rightarrow \{0,1\}^m, n > m$. Consider the following adversarial game.

1. Adversary chooses an $x \in \{0,1\}^n$.
2. Adversary is given a $k$ which is chosen uniformly at random from $\mathcal{K}$.
3. Adversary has to find $x'$ such that $x \neq x'$ but $h_k(x) = h_k(x')$.

A strategy $\mathcal{A}$ for the adversary runs in two stages. In the first stage $\mathcal{A}^{\text{guess}}$, the adversary finds the $x$ to which he has to commit in Step 1. It also produces some auxiliary state information $\sigma$. In the second stage $\mathcal{A}^{\text{find}}(k, x, \sigma)$, the adversary either finds a $x' \neq x$ such that $h_k(x) = h_k(x')$ or reports failure. Both $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(k, x, \sigma)$ are probabilistic algorithms. The success probability of the strategy is measured over the random choices made by $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(k, x, \sigma)$ and the random choice of $k$ in Step 2 of the game.

We say that $\mathcal{A}$ is an $(\varepsilon, \eta)$-strategy for $\{h_k\}_{k \in \mathcal{K}}$ if the success probability of $\mathcal{A}$ is at least $\varepsilon$ and it invokes the hash function $h_k$ at most $\eta$ times. In this case we say that the adversary has an $(\varepsilon, \eta)$-strategy for $\{h_k\}_{k \in \mathcal{K}}$. Note that we do not include time as an explicit parameter though it would be easy to do so. Informally, we say that $\{h_k\}_{k \in \mathcal{K}}$ is a UOWHF if the adversary has a negligible probability of success with respect to any probabilistic polynomial time strategy. Here, the security parameter is length of the message i.e., the length of the input.

In this paper we are interested in extending the domain of a UOWHF. More specifically, given a UOWHF $\{h_k\}_{k \in \mathcal{K}}$, $h_k : \{0,1\}^n \rightarrow \{0,1\}^m$, $n > m$ , we would like to construct another extended UOWHF $\{H_p\}_{p \in \mathcal{P}}$ with $H_p : \{0,1\}^N \rightarrow \{0,1\}^m$, where $n < N$.

We say that $\mathcal{B}$ is an $(\varepsilon, \eta)$-extended strategy for $\{H_p\}_{p \in \mathcal{P}}$ if the success probability of $\mathcal{B}$ is at least $\varepsilon$ and it invokes the hash function $h_k$ at most $\eta$ times. In this case we say that the adversary has an $(\varepsilon, \eta)$-extended strategy for $\{H_p\}_{p \in \mathcal{P}}$. Note that $H_p$ is built using $h_k$ and hence while studying strategies for $H_p$ we are interested in the number of invocations of the hash function $h_k$.

The correctness of our construction will essentially be a Turing reduction. We will show that if there is an $(\varepsilon, \eta)$-extended strategy $\mathcal{B}$ for $\{H_p\}_{p \in \mathcal{P}}$, then there is an $(\varepsilon', \eta')$-strategy $\mathcal{A}$ for $\{h_k\}_{k\in\mathcal{K}}$, where $\varepsilon'$ is not significantly lesser than $\varepsilon$ and $\eta'$ is not much larger than $\eta$. This shows that if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF, then so is $\{H_p\}_{p\in\mathcal{P}}$. In this case, we say that the domain extension is valid.

The key length for the base hash family $\{h_k\}_{k\in\mathcal{K}}$ is $\lceil \log_2|\mathcal{K}|\rceil$. On the other hand, the key length for the extended hash family $\{H_p\}_{p\in\mathcal{P}}$ is $\lceil \log_2|\mathcal{P}|\rceil$. Thus increasing the size of the input from $n$ bits to $N$ bits results in an increase of the key size by an amount $\lceil \log_2|\mathcal{P}|\rceil - \lceil \log_2|\mathcal{K}|\rceil$. From a practical point of view it is very important to minimize this increase in the key length.

For the remainder of this paper we assume the following conventions.

1. $\{h_k\}_{k\in\mathcal{K}}$ is always the base hash family, where $\mathcal{K} = \{0,1\}^K$ and $h_k : \{0,1\}^n \to \{0,1\}^m$. In case of sequential construction $n > m$, in case of full binary tree based construction $n > 2m$, and in case of 4-dimensional construction $n > 4m$.
2. We will construct $\{H_p\}_{p\in\mathcal{P}}, H_p : \{0,1\}^N \to \{0,1\}^m$ using the base hash family $\{h_k\}_{k\in\mathcal{K}}$, where $p = k||\mu_1||\mu_2||\cdots||\mu_l$ for some $l$ and each $\mu_i$ is $m$-bit binary string called *mask* and $|k| = K$. Here, in case of sequential algorithm $N = n(r+1)-mr$, in case of tree based construction $N = n(2^t-1)-m(2^t-2)$ and in case of 4-dimensional construction $N = n2^t - m(2^t-1)$. Let us define $\mu[i,j] = \mu_i||\ldots||\mu_j$, where $1 \le i \le j \le l$. We will use $\mu[j]$ instead of $\mu[1,j]$ for $j \ge 1$ and define $\mu[0]$ to be empty string.
3. In sequential construction input of $H_p$ is written as $y = y_0||y_1||\cdots||y_r$ where $|y_0| = n$ and $|y_i| = n-m$ for $1 \le i \le r$. In case of tree based construction input of $H_p$ is written as $x = x_1||\cdots||x_{2^t-1}$ where $|x_i| = n-2m$ for $1 \le i < 2^{t-1}$ and $|x_i| = n$ for $2^{t-1} \le i \le 2^t - 1$. In 4-dimensional construction input of $H_p$ is written as $x = x_1||\cdots||x_{2^t}$ where $|x_i| = n-4m$ for $1 \le i < 2^a$, $|x_i| = n-3m$ for $2^a \le i \le 2^a(2^b-1)$, $|x_i| = n-2m$ for $2^a(2^b-1) < i \le 2^{a+b}(2^c-1)$, $|x_i| = n-m$ for $2^{a+b}(2^c-1) < i \le 2^{a+b+c}(2^d-1)$ , and $|x_i| = n$ for $2^{a+b+c}(2^d-1)+1 \le i \le 2^t$. Here $a, b, c$, and $d$ are such that $g(t) = (a,b,c,d)$.
4. In tree based construction let $i \in V_t$ and $x$ be a message of length $N$. We define $x(i) = x_i||x_{2i}||x_{2i+1}||x_{4i}||x_{4i+1}||\ldots$ i.e. concatenating all $x_j$ in ascending order of $j$ where $j$ runs in $T_t[i]$. In other words the part of the message used in the complete binary sub-tree rooted at $i$.

## 4 Sequential Construction

The best known sequential algorithm is given by Shoup [10]. We will generalize the idea of the construction. We also give the sufficient condition for valid sequential construction. Let $\psi : [1,r] \to [1,l]$ be any function called a **masking assignment**. Fix a masking assignment $\psi$, $H_p(y)$, the extended hash function, is computed by the following algorithm.

1. **Input:** $y = y_0||y_1||\ldots||y_r$ and $p = k||\mu_1||\mu_2||\ldots||\mu_l$.
2. $z_0 = h_k(y_0)$.
3. For $1 \le i \le r$, define $s_i = z_{i-1} \oplus \mu_{\psi(i)}$ and $z_i = h_k(s_i||y_i)$.
4. **Output:** $z_r$.

We say that the sequential construction is based on the masking assignment $\psi$. In Shoup's algorithm $\psi = \nu_2 + 1$ and $l = 1 + \lfloor \log_2 r \rfloor$ (in his paper $\nu_2$ is masking assignment but that makes no difference). We will write $s(i, y, k, \mu)$, $z(i, y, k, \mu)$ for $s_i$ and $z_i$ respectively (in the algorithm with input $(y, p)$, where $p = k||\mu$). Now we will define some terms related with masking assignment and domain extension.

**Definition 1.** *We say that $\psi$ is **correct** if for all $1 \leq i \leq r$, $C \in \{0,1\}^m$, $y \in \{0,1\}^N$ and for any hash function $h_k$ there is an algorithm called $\mathsf{Mdef}_{seq}(i, y, k, C, \psi)$ which outputs $\mu = \mu_1||\mu_2||\ldots||\mu_l$ such that $s(i, y, k, \mu) = C$. $\mathsf{Mdef}_{seq}(i, y, k, C, \psi)$ is called a mask defining algorithm. A sequential construction based on a correct masking assignment is called a correct domain extension. A masking assignment is **totally correct** if there is a mask defining algorithm $\mathsf{Mdef}_{seq}(i, y, k, C, \psi) = \mu = \mu_1||\mu_2||\ldots||\mu_l$ for any $i, y, k, C$ as above such that $s(i, y, k, \mu) = C$ holds and $\mu$ is a random string whenever $C$ is a random string and other inputs are fixed.*

**Definition 2.** *We say that a domain extension algorithm is **valid** if $\{H_p\}_{p \in \mathcal{P}}$ is a UOWHF whenever $\{h_k\}_{k \in \mathcal{K}}$ is a UOWHF. In case of sequential construction if valid domain extension algorithm is based on a masking assignment $\psi$ then we say that the masking assignment is **valid.***

**Definition 3.** *A masking assignment $\psi : [1, r] \to [1, l]$ is **strongly even-free** (or **even-free**) if for each $[a, b] \subseteq [1, r]$ there exists $c \in [a, b]$ such that $\psi(c)$ occurs exactly once (respectively, odd times) in the sequence $\psi(a), \psi(a+1), \ldots, \psi(b)$. Call this $c$ (also the mask $\psi(c)$) a **single-man** for the interval $[a, b]$.*

Now we will try to characterize all valid masking assignments. From Mironov's paper [4] we have seen that every valid masking assignment is even-free. He also showed that, every even-free masking assignment requires at least $1 + \lfloor \log_2 r \rfloor$ many masks and the minimum attains if we consider the masking assignment $\psi = \nu_2 + 1$ which is used in Shoup's algorithm. Now we will prove that, in case of sequential construction, every strongly even-free masking assignmentis valid. The same masking assignment i.e. $\nu_2 + 1$ is in fact a strongly even-free masking assignment.

To provide the sufficient condition for valid sequential extension, we will first prove that strongly even-free implies totally correct. The proof of totally correct implies valid is a basic idea of proving an extension is valid. In all known papers the same idea is used for proving validness of extension. So, one can see this any one of these papers [8,10,4]. We will give a proof in case of complete binary tree based domain extension in Section 5.

**Lemma 1.** *If $\psi$ is strongly even-free then $\psi$ is totally correct.*

**Proof.** We will define the mask defining algorithm $\mathsf{Mdef}_{seq}(i, y, k, C, \psi)$.

1. If $i = 1$ then define $\mu_{\psi(1)} = C \oplus h_k(y_0)$ and define all yet undefined masks randomly and quit.

2. If $i > 1$ then choose any $c$ which is a single-man for the interval $[1, i]$. Compute $j \leftarrow i - c$, If $j = 0$ then goto step 4.
3. Let $\psi' : [1, j] \to [1, l]$ be a masking assignment such that $\psi'(n) = \psi(n + c)$ where $n \in [1, j]$. Take a random string $D$ and then define, $y' = y'_0||\ldots||y'_j$ where, $y'_n = y_{n+c}$ when $n \geq 1$ and $y'_0 = D||y_c$. Run $\mathsf{Mdef}_{seq}(j, y', k, C, \psi')$.
4. Define all yet undefined masks except $\mu_{\psi(c)}$ (i.e. after running $\mathsf{Mdef}_{seq}$ some masks may not be defined as $\psi'$ may not be onto or $j$ can be 0) randomly. Compute $\mu_{\psi(c)} = z(c - 1, y, k, \mu) \oplus D$ and quit.

Note that to compute $z(c - 1, y, k, \mu)$ we do not need the mask $\mu_{\psi(c)}$ as $c$ is a single-man and the above recursive algorithm will always stop as $j < i$. The masking assignment $\psi'$ is nothing but $\psi$ restricted at $[c, i]$. So, if $s(c, y, k, \mu) = D$ then by induction $s(i, y, k, \mu) = C$. But, $s(c, y, k, \mu) = D$ is true by definition of $\mu_{\psi(c)}$. It proves the correctness of $\psi$. If $C$ is a random string then all masks $\mu$ is a random string as they are randomly defined (in step-4) or they are obtained by XOR-ing with a random string (in step-1). So, it is totally correct. ■

**Theorem 1.** ***(Sufficient Condition for Valid Sequential Extension)*** *If a sequential domain extension is based on a strongly even-free masking assignment $\psi$ then the domain extension is valid.*

**Proof.** By the above lemma $\psi$ is totally correct. The proof of totally correct implies valid is given in case of complete binary tree domain extension in Section 5. The same idea will carry through in case of sequential construction. So, we omit this proof.

**Remark:** Strongly even-free is sufficient condition for correct masking assignment. For example $\nu_2 + 1$. One can feel that the condition may be necessary. So, we may conjecture that, if a masking assignment is correct for any arbitrary hash function then it should be strongly even-free.

## 5 Complete Binary Tree Based Construction

In the previous section we study about sequential construction. Now, we will first define the generic algorithm based on complete binary tree of height $t$. Let $T_t = (V_t, E_t)$ be the full binary tree where $V_t = \{1, 2, \ldots, 2^t - 1\}$ and $E_t = \{e_i; 2 \leq i \leq 2^t - 1\}$, $e_i = (i, \lfloor i/2 \rfloor)$. Let any function $\psi_t : E_t \to [1, l]$ be a masking assignment. (Note that we use $E_t$ for domain of $\psi_t$.) Let $x = x_1||\ldots||x_{2^t-1}$ be the input message of length $N$. Given $\psi_t, x$, and $p = k||\mu$, $H_p(x)$ is computed by the following algorithm.

1. **Input:** $x = x_1||x_2||\ldots||x_{2^t-1}$ and $p = k||\mu_1||\mu_2||\ldots||\mu_l$.
2. If $2^{t-1} \leq i \leq 2^t - 1$ then $z_i = h_k(x_i)$ else if $1 \leq i < 2^{t-1}$ then $z_i = h_k(s_{2i}||s_{2i+1}||x_i)$, where $s_i = z_i \oplus \mu_{\psi_t(e_i)}$.
3. **Output:** $z_1$.

Note that the input of $i^{th}$ node is $s_{2i}||s_{2i+1}||x_i$ and output of node $i$ $z_i$. We say that the above complete binary tree based domain extension is based on the masking assignment $\psi_t$. We will write $s(i, x, k, \mu, t)$ and $z(i, x, k, \mu, t)$ for $s_i$ and $z_i$ in the above algorithm, respectively. Like sequential algorithm we say that $\psi_t$ is **correct** if for each $1 \leq i < 2^{t-1}$, there is a mask defining algorithm $\mathsf{Mdef}_{tree}(i, x, k, t, r_0, r_1, \psi_t)$ where $|r_0| = |r_1| = m$ which outputs $\mu = \mu_1 || \dots || \mu_l$ such that $s(2i, x, k, \mu, t) = r_0$ and $s(2i+1, x, k, \mu, t) = r_1$. $\psi_t$ is **totally correct** if the output $\mu$ of the mask defining algorithm is random string provided $r_0$, $r_1$ are random strings and other inputs are fixed.

**Definition 4.** *A masking assignment $\psi_t : E_t \to [1, l]$ is a* ***level uniform*** *masking assignment if there are two functions $\alpha_t, \beta_t : [2, t] \to [1, l]$ such that $\psi_t(e_i) = \alpha_t(j)$ if $i$ is odd and $\psi_t(e_i) = \beta_t(j)$ if $i$ is even, where $j = ht_t(i) + 1$.*

We will first briefly state some standard binary tree based constructions all of which are based on level-uniform masking assignment.

1. **Bellare-Rogaway** [1]: $\alpha_t(i) = i - 1$ and $\beta_t(i) = t + i - 2$. In [1] it was shown that $\psi_t$ is valid. Here, we need $2(t-1)$ masks.
2. **Sarkar** [8]: $\alpha_t(i) = i - 1$ and $\beta_t(i) = t + \nu_2(i-1)$. In [8] it was shown that $\psi_t$ is valid. Here, we need $t + \lceil \log_2 t \rceil - 1$ masks.

Now, we will propose our binary tree based construction which needs lesser number of masks than Sarkar's. Like above examples our domain extension is also based on level uniform masking assignment. So, it is enough to define these two functions $\alpha_t$ and $\beta_t$. This construction can be found more detail in [5].

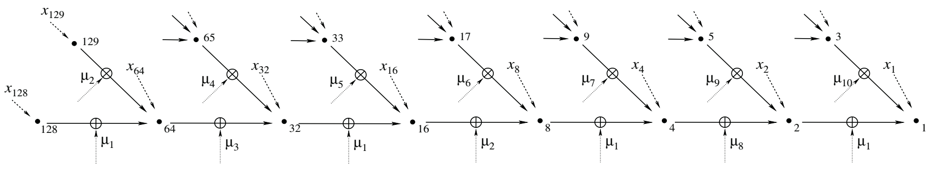

**Fig. 1.** The right most part of the complete binary tree when you place the root of the tree (i.e. vertex 1) in top. ($t = 8$ and $|x_1| = \cdots = |x_{127}| = n - 2m$, and $|x_{128}| = \cdots = |x_{255}| = n$. $\bullet$ means $h_k(\cdot)$.)

## 5.1 Improved Binary Tree Based Construction

Define two sequences $\{l_k\}_{k \geq 0}$ and $\{m_t\}_{t \geq 2}$ as follow: $l_{k+1} = 2^{l_k + k} + l_k$ where, $l_0 = 2$ and $m_2 = 2$ and if $k \geq 1$, $m_t = t + k$ for all $t \in [l_{k-1} + 1, l_k]$. Note that, both $l_k$ and $m_t$ are strictly increasing sequences and if $t = l_k$ for some $k$ then $m_{t+1} = m_t + 2$ and if for some $k$, $l_k < t < l_{k+1}$ then $m_{t+1} = m_t + 1$. Later, we will see that $m_t$ is the number of masks of our algorithm for binary tree of height $t$ and $m_t \leq t + k$ till $t \leq l_k$. Intuitively, $k = O(\log_2^*(l_k))$ so, $m_t = t + O(\log_2^* t)$. The level uniform masking assignment $\psi_t$ is based on the functions $\alpha_t$, $\beta_t : [2, t] \to [1, m_t]$, $t \geq 2$ where they are defined as follow (See Figure 1 where a right most part of the tree is drawn which will completely determine the functions $\alpha_t$ and $\beta_t$.):

1. $\alpha_2(2) = 2$ and $\beta_2(2) = 1$.
2. For $t \geq 3$, $\alpha_t(i) = \alpha_{t-1}(i)$ and $\beta_t(i) = \beta_{t-1}(i)$ whenever $2 \leq i \leq t-1$.
3. If $t \geq 3$ and $t - 1 = l_k$ for some $k$ then $\alpha_t(t) = \alpha_{t-1}(t-1) + 2$ and $\beta_t(t) = \alpha_{t-1}(t-1) + 1$ and if $l_k < t - 1 < l_{k+1}$ then $\alpha_t(t) = \alpha_{t-1}(t-1) + 1$ and $\beta_t(t) = \nu_2(t - 1 - l_k) + 1$.

**Theorem 2.** *For $t \geq 2$, $\alpha_t$ and $\beta_t$ map into $[1, m_t]$. Moreover, $\alpha_t(t) = m_t$ and $\alpha_t([2,t]) \cup \beta_t([2,t]) = [1, m_t]$. So, we need $m_t$ many masks.*

**Proof.** For $2 \leq i \leq t$, $\alpha_t(i) = m_i$ can be easily proved by induction. Also note that, when $i = l_k + 1$ for some $k$, then $\beta_t(i) = m_{i-1} + 1 < m_i$ and when $l_k + 1 < i \leq l_{k+1}$, $\beta_t(i) = \nu_2(i - l_k) + 1 \leq l_k + k = m_{l_k} < m_i$. So, it proves that $\alpha_t$ and $\beta_t$ map into $[1, m_t]$. To prove the last part let $1 \leq j \leq m_t$. So we have some $i$ so that $j = m_i$ or $j = m_{l_k} + 1$. If $j = m_i$ then $\alpha_t(i) = m_i = j$ otherwise $\beta_t(l_k + 1) = m_{l_k} + 1 = j$. So, we have that $\alpha_t([2,t]) \cup \beta_t([2,t]) = [1, m_t]$. ∎

Now, we will prove that the above $\psi_t$ is totally correct for all $t \geq 2$. For this we need to define $\mathsf{Mdef}_{tree}(i, x, k, t, r_0, r_1, \psi_t)$. We will define the mask defining algorithm for $i = 1$ otherwise we can consider the complete binary tree rooted at $i$ (i.e. $T_t[i]$) and define $\mathsf{Mdef}_{tree}(i, x, k, t, r_0, r_1, \psi_t)$ by $\mathsf{Mdef}_{tree}(1, x', k, t', r_0, r_1, \psi')$ where, $t' = ht_t(i)$, $x' = x(i)$ i.e. the part of the message involved in the subtree $T_t[i]$ and $\psi'$ is $\psi_t$ restricted at $T_t[i]$ which is same as $\psi_{t'}$ (it can be checked easily as $\psi_t$ is level uniform). So, we can assume that $i = 1$.

1. If $t = l_k + 1$ for some $k$ then
   (a) Define $\mu$ by random string.
   (b) Compute $\mu_{m_t - 1} = z(2, x, k, \mu, t) \oplus r_0$ and $\mu_{m_t} = z(3, x, k, \mu, t) \oplus r_1$. To compute $z(2, x, k, \mu, t)$ and $z(3, x, k, \mu, t)$ we actually need only $\mu[m_t - 2]$ as $\mu_{m_t - 1}$ and $\mu_{m_t}$ appear only on edges $e_2$ and $e_3$.
2. If $l_k + 1 < t \leq l_{k+1}$ for some $k$ then
   (a) Let the set $A = \{2^{i+1} + 1 : 0 \leq i \leq r\} \cup \{2^{r+1}\}$ where, $r = t - (l_k + 1)$.
   (b) Choose $b_i$ randomly for all $i \in A - \{3\}$ such that, $|b_i| = m$ and $b_3 = r_1$.
   (c) Let $y = y_0 || y_1 || \ldots || y_r$ where, $y_0 = b_{2^{r+1}} || b_{2^{r+1}+1} || x_{2^r}$ and $y_j = b_{2^{r+1-j}+1} || x_{2^{r-j}}$ for $1 \leq j \leq r$.
   (d) Run $\mathsf{Mdef}_{seq}(r, y, k, r_0, \psi') = \mu[l']$ where, $l' = \lfloor \log_2 r \rfloor + 1 \leq l_k + k = m_{l_k}$ and $\psi'$ is same as $\psi$ restricted at the path $e_{2^r}, e_{2^{r-1}}, \ldots, e_2$. More precisely, $\psi'(i) = \psi(e_{2^{r+1-i}})$. So, if $\mu$ is computed such a way that, $s(i, x, k, \mu, t) = b_i$ for all $i \in A$ then by definition of $\mathsf{Mdef}_{seq}$ we will have $s(2, x, k, \mu, t) = r_0$.
   (e) Define remaining masks randomly and for $i \in A$ (in descending order) compute $\mu_i = z(i, x, k, \mu, t) \oplus b_i$.

When $t = l_k + 1$ the correctness of $\psi_t$ easily follows from the definition of $\mu_{m_t - 1}$ and $\mu_{m_t}$. Note that, to compute $z(j, x, k, \mu, t)$ for $j \in A$ in step-2(e), we do not need the masks $\mu_{\psi(e_{j'})}$ for all $j' \in A$, $j' > j$. So, $s(i, x, k, \mu, t) = b_i$ for all $i \in A$ and hence $\mathsf{Mdef}_{tree}$ is correct. If $r_0$ and $r_1$ are random strings then so is the output $\mu$ and hence $\psi_t$ is totally correct for all $t$. So, we have the following theorem:

**Theorem 3.** *The masking assignment $\psi_t$ based on two functions $\alpha_t$ and $\beta_t$ as above is totally correct.*

Now we will prove the statement *totally correct implies valid* for binary tree based masking assignment. The same idea will carry through for the other constructions.

**Theorem 4.** ***(Validness of domain extension)*** *In case of binary tree based domain extension a totally correct masking assignment is always valid. More precisely, we have that, if there is an $(\varepsilon, \eta)$ winning strategy $\mathcal{A}$ for $\{H_p\}_{p\in\mathcal{P}}$ then there is also an $(\frac{\varepsilon}{2^t-1}, \eta + 2(2^t-1))$-strategy $\mathcal{B}$ for $\{h_k\}_{k\in\mathcal{K}}$ whenever $\{H_p\}_{p\in\mathcal{P}}$ is based on totally correct masking assignment.*

**Proof.** We describe the two stages of the strategy $\mathcal{B}$ as follows.

**Algorithm $\mathcal{B}^{\textbf{guess}}$**$= (y, s)$**:**
Run $\mathcal{A}^{\text{guess}}$ to obtain $x \in \{0,1\}^N$ and state information $s'$. Choose an $i \in_R \{1, \ldots, 2^t - 1\}$. If $2^{t-1} \leq i \leq 2^t - 1$, set $y = x_i$; $r_0, r_1$ to be the empty string and $s = (s', i, r_0, r_1, x)$. Output $(y, s)$ and stop. If $1 \leq i \leq 2^{t-1} - 1$, then choose two strings $r_0$ and $r_1$ uniformly at random from the set $\{0,1\}^m$. Set $y = r_0||r_1||x_i$ and $s = (s', i, r_0, r_1, x)$. Output $(y, s)$ and stop. At this point the adversary is given a $k$ which is chosen uniformly at random from the set $\mathcal{K} = \{0,1\}^K$. The adversary then runs $\mathcal{B}^{\text{find}}$ which is described below.

**Algorithm $\mathcal{B}^{\textbf{find}}(y, k, s) = y'$:** (Note $s = (s', i, r_0, r_1, x)$.)
Define the masks $\mu_1, \ldots, \mu_{m_t}$ by executing algorithm $\mathsf{Mdef}_{tree}(i, x, k, t, r_0, r_1)$. This defines the key $p = k||\mu$ for the function $H_p$. Run $\mathcal{A}^{\text{find}}(x, p, s')$ to obtain $x'$. Let $y'$ be the input of $i^{th}$ node corresponding to the string $x'$. Output $y'$.

We now lower bound the probability of success. By totally correctness $p$ is a randomly chosen key from the set $\mathcal{P}$. Suppose $x$ and $x'$ $(x \neq x')$ collide for the function $H_p$. Then there must be a $j$ in the range $1 \leq j \leq 2^t - 1$ such that at vertex $j$ there is a collision for the function $h_k$. (Otherwise it is possible to prove by a backward induction that $x = x'$.) The probability that $j = i$ is $\frac{1}{2^t-1}$ where $i$ is a random number lying between 1 and $2^t - 1$. Hence if the success probability of $\mathcal{A}$ is at least $\epsilon$, then the success probability of $\mathcal{B}$ is at least $\frac{\epsilon}{2^t-1}$. Also the number of invocations of $h_k$ by $\mathcal{B}$ is equal to the number of invocations of $h_k$ by $\mathcal{A}$ plus at most $2(2^t - 1)$. This completes the proof. ■

**Theorem 5.** *The speed-up of our algorithm over the sequential algorithm in Section 4 is by a factor of $\frac{2^t-1}{t}$.*

**Proof.** This algorithm hashes a message of length $n(2^t - 1) - m(2^t - 2)$ into a digest of length $m$ using $t$ parallel rounds. The time taken by a single parallel round is proportional to the time required by a single invocation of the hash function $h_k$. The sequential construction require $2^t - 1$ invocations of the hash function $h_k$ on a message of length $n(2^t - 1) - m(2^t - 2)$. Hence, the speed-up of the binary tree algorithm over the sequential algorithm is by a factor of $\frac{2^t-1}{t}$. ■

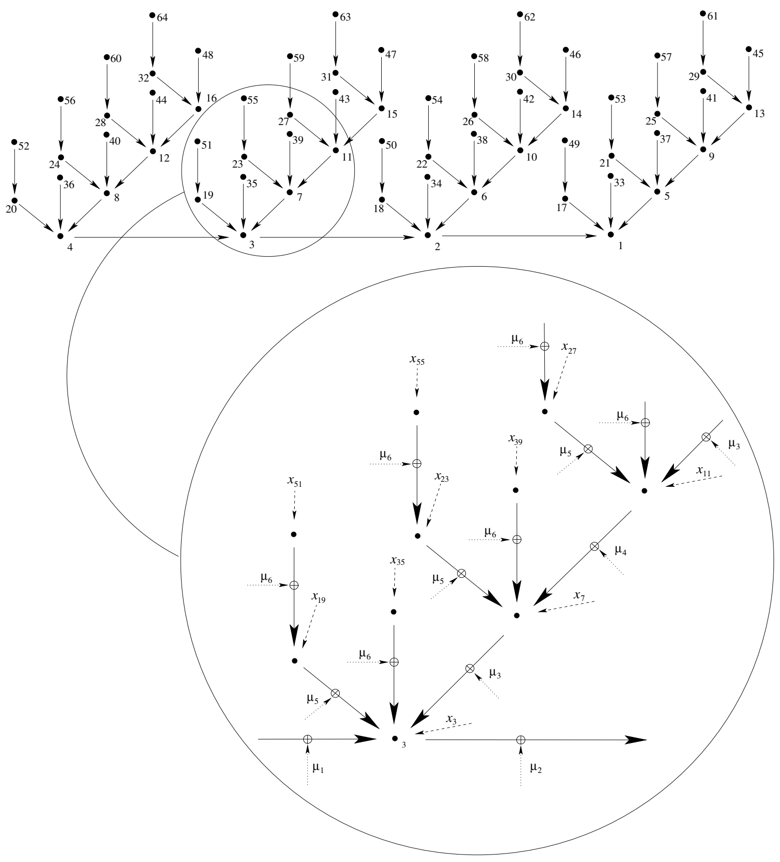

**Fig. 2.** 4-dimensional parallel algorithm ($t = 6$ and $x = x_1 || \dots || x_{2^6}$. Note that $g(6) = (2, 2, 1, 1)$ and $|x_1| = \cdots = |x_3| = n - 4m$, $|x_4| = \cdots = |x_{12}| = n - 3m$, $|x_{13}| = \cdots = |x_{16}| = n - 2m$, $|x_{17}| = \cdots = |x_{32}| = n - m$, and $|x_{33}| = \cdots = |x_{2^6}| = n$. $\cdot$ means $h_k(\cdot)$.)

**Remark:** The speed-up achieved by our algorithm is substantial even for moderate values of $t$. Such speed-up will prove to be advantageous for hashing long messages.

**Theorem 6.** *The number of masks for this algorithm is* $t + O(log_2^* t)$.

**Proof.** From the recurrence relation it is clear that $2^{2^{l_k}} > l_{k+1} > 2^{l_k}$. So, $\log_2^*(l_k) + 1 \leq \log_2^*(l_{k+1}) \leq \log_2^*(l_k) + 2$ and hence $\log_2^*(l_k) = \theta(k)$ i.e. $\log_2^*(l_k)$ and $k$ are of same order. So, for all $l_k \leq t < l_{k+1}$, $m_t - t = k = O(\log_2^* t)$. ■

## 6 Non-complete *l*-Ary Tree Based Construction

Our first new construction IBTC is based on the complete binary tree. In this section we present a new parallel construction for a UOWHF based on a 4-ary directed tree which is not complete.

We will first define the generic algorithm based on the 4-ary directed tree $T_t = (V_t, E_t)$ for $t \geq 4$ (See this notation in Section 3). For $t = 2$ and $t = 3$, we can define the algorithm based on the binary and 3-ary tree based construction (See Section 6.3), respectively.

Like previous constructions, any function $\psi_t : E_t \to [1, l]$ is a masking assignment. Let $x = x_1||x_2||\ldots||x_{2^t}$ be the input message of length $N$. Given $\psi_t, x$, and $p = k||\mu$, $H_p(x)$ is computed by the following algorithm. This is depicted in Figure 2. In this section $a, b, c$ and $d$ denote the output of $g(t)$.

```
1. **Input:** $x = x_1||x_2|| \cdots ||x_{2^t}$ and $p = k||\mu_1||\mu_2||\ldots||\mu_l$.
2. If $2^{a+b+c}(2^d - 1) < i \leq 2^t$ then $z_i = h_k(x_i)$.
3. If $d = 1$ then goto step 4.
   (a) For $j = 2^d - 2$ down to 1 do
       For $j2^{a+b+c} < i \leq (j+1)2^{a+b+c}$, $z_i = h_k(s_{i+2^{a+b+c}}||x_i)$ where $s_i = z_i \oplus \mu_{\psi_t(e_i)}$ (This notation is also same in the following procedure).
4. For $2^{a+b}(2^c - 1) < i \leq 2^{a+b+c}$, $z_i = h_k(s_{i+2^{a+b+c}}||x_i)$.
5. If $c = 1$ go to step 6.
   (a) For $j = 2^c - 2$ down to 1 do
       For $j2^{a+b} < i \leq (j+1)2^{a+b}$, $z_i = h_k(s_{i+2^{a+b}}||s_{i+2^{a+b+c}}||x_i)$.
6. For $2^a(2^b - 1) < i \leq 2^{a+b}$, $z_i = h_k(s_{i+2^{a+b}}||s_{i+2^{a+b+c}}||x_i)$.
7. If $b = 1$ go to step 8.
   (a) For $j = 2^b - 2$ down to 1 do
       For $j2^a < i \leq (j+1)2^a$, $z_i = h_k(s_{i+2^a}||s_{i+2^{a+b}}||s_{i+2^{a+b+c}}||x_i)$.
8. For $i = 2^a$, $z_i = h_k(s_{i+2^a}||s_{i+2^{a+b}}||s_{i+2^{a+b+c}}||x_i)$.
9. For $i = 2^a - 1$ down to 1, $z_i = h_k(s_{i+1}||s_{i+2^a}||s_{i+2^{a+b}}||s_{i+2^{a+b+c}}||x_i)$.
10. **Output:** $z_1$.
```

We say that, the above non-complete 4-ary tree based construction is based on the masking assignment $\psi_t$. Here, we need some definitions in order to consider the correctness of $\psi_t$.

1. We will write $s(i, x, k, \mu, t)$, $z(i, x, k, \mu, t)$ for $s_i$ and $z_i$, respectively.
2. $\epsilon$ means the empty string.
3. For each node $1 \leq i \leq 2^{a+b+c}(2^d - 1)$,
   (a) Define $s^0(i, x, k, \mu, t)$ as $s(i + 1, x, k, \mu, t)$ for $1 \leq i < 2^a$ and as $\epsilon$ for $2^a \leq i \leq 2^{a+b+c}(2^d - 1)$.
   (b) Define $s^1(i, x, k, \mu, t)$ as $s(i + 2^a, x, k, \mu, t)$ for $1 \leq i \leq 2^a(2^b - 1)$ and as $\epsilon$ for $2^a(2^b - 1) < i \leq 2^{a+b+c}(2^d - 1)$.
   (c) Define $s^2(i, x, k, \mu, t)$ as $s(i + 2^{a+b}, x, k, \mu, t)$ for $1 \leq i \leq 2^{a+b}(2^c - 1)$ and as $\epsilon$ for $2^{a+b}(2^c - 1) < i \leq 2^{a+b+c}(2^d - 1)$.
   (d) Define $s^3(i, x, k, \mu, t)$ as $s(i + 2^{a+b+c}, x, k, \mu, t)$ for $1 \leq i \leq 2^{a+b+c}(2^d - 1)$.

Therefore the input of $i^{th}$ node can be represented by $s^0(i, x, k, \mu, t)\ ||s^1(i, x, k, \mu, t)||s^2(i, x, k, \mu, t)||s^3(i, x, k, \mu, t)||x_i$ for $1 \leq i \leq 2^{a+b+c}(2^d - 1)$.

We will say that $\psi_t$ is **correct** if, for each $1 \leq i \leq 2^{a+b+c}(2^d - 1)$, there is an algorithm $\mathsf{Mdef}_{4dim}(i, x, k, t, r_0, r_1, r_2, r_3, \psi_t)$, where $r_0$ is a $m$-bit string if $1 \leq i < 2^a$ and $\epsilon$ if $2^a \leq i \leq 2^a(2^b - 1)$, $r_1$ is a $m$-bit string if $1 \leq i \leq 2^a(2^b - 1)$

and $\epsilon$ if $2^a(2^b-1) < i \le 2^{a+b}(2^c-1)$, $r_2$ is a $m$-bit string if $1 \le i \le 2^{a+b}(2^c-1)$ and $\epsilon$ if $2^{a+b}(2^c-1) < i \le 2^{a+b+c}(2^d-1)$, and $r_3$ is a $m$-bit string for $1 \le i \le 2^{a+b+c}(2^d-1)$ which outputs $\mu = \mu_1 || \cdots || \mu_l$ such that $s^j(i,x,k,\mu,t) = r_j$ for $0 \le j \le 3$. $\psi_t$ is **totally correct** if the output $\mu$ of the mask defining algorithm is random string provided $r_0$, $r_1, r_2$ and $r_3$ are random strings and other inputs are fixed.

## 6.1 4-Dimensional Domain Extender

Our second new parallel construction uses the following masking assignment $\psi_t : E_t \to [1,t]$. The map represents the assignment of masks to the directed edges. Here we present our definition of $\psi_t$ which needs $t$ masks for 4-dimensional construction. Intuitively, the map $\psi_t$ is made from expanding the mask assigning method of Shoup's sequential construction into four directions. At first, we define four functions $\alpha_t$, $\beta_t$, $\gamma_t$, and $\delta_t$ as follows.

1. $\alpha_t : [1, 2^a - 1] \to [1, a]$ is defined by $\alpha_t(i) = 1 + \nu_2(2^a - i)$.
2. $\beta_t : [1, 2^b - 1] \to [a+1, a+b]$ is defined by $\beta_t(i) = a + 1 + \nu_2(2^b - i)$.
3. $\gamma_t : [1, 2^c - 1] \to [a+b+1, a+b+c]$ is defined by $\gamma_t(i) = a+b+1+\nu_2(2^c - i)$.
4. $\delta_t : [1, 2^d - 1] \to [a+b+c+1, t]$ is defined by $\delta_t(i) = a+b+c+1+\nu_2(2^d - i)$.

Our masking assignment $\psi_t(e_i)$ is defined as follow:

1. $\psi_t(e_i) = \alpha_t(j)$ if $2 \le i \le 2^a$ and $j = i - 1$.
2. $\psi_t(e_i) = \beta_t(j)$ if $2^a < i \le 2^{a+b}$ and $j2^a < i \le (j+1)2^a$.
3. $\psi_t(e_i) = \gamma_t(j)$ if $2^{a+b} < i \le 2^{a+b+c}$ and $j2^{a+b} < i \le (j+1)2^{a+b}$.
4. $\psi_t(e_i) = \delta_t(j)$ if $2^{a+b+c} < i \le 2^t$ and $j2^{a+b+c} < i \le (j+1)2^{a+b+c}$.

Now we will prove that the above $\psi_t$ is totally correct.

**Theorem 7.** *The masking assignment $\psi_t$ based on four functions $\alpha_t$, $\beta_t$, $\gamma_t$ and $\delta_t$ as above is totally correct.*

**Proof.** We will define the mask defining algorithm $\mathsf{Mdef}_{4dim}$.
**Input:** $k, x, i, r_0, r_1, r_2, r_3, \psi_t$
**output:** $\mu = \mu_1 || \ldots || \mu_t$ such that $s^j(i,x,k,\mu,t) = r_j$ for $0 \le j \le 3$.

We can define $\mathsf{Mdef}_{4dim}$ for each case $j \in \{1,2,3,4\}$ where

1. $1 \le i < 2^a$.
2. $2^a \le i \le 2^a(2^b - 1)$.
3. $2^a(2^b-1) < i \le 2^{a+b}(2^c-1)$.
4. $2^{a+b}(2^c-1) < i \le 2^{a+b+c}(2^d-1)$.

But we will present the specific procedure of $\mathsf{Mdef}_{4dim}$ for only case 1 since the other cases are very similar and much simpler than case 1. Let $1 \le i < 2^a$.

1. (a) Let $D = 2^d - 1$. Let $\psi' : [1, D] \to [1, t]$ be a masking assignment such that $\psi'(j) = \psi_t(e_{i+(D+1-j)2^{a+b+c}})$ where $j \in [1, D]$.
   (b) Let $y^3 = y^3_0||y^3_1||\ldots||y^3_D$ where, $y^3_v = x_{i+(D-v)2^{a+b+c}}$ for $0 \le v \le D-1$ and $y^3_D = r_0||r_1||r_2||x_i$. Note that $|y^3_0| = n$ and $|y^3_j| = n - m$ for $1 \le j \le D$.
   (c) Run $\mathsf{Mdef}_{seq}(D, y^3, k, r_3, \psi')$ to get an output $\mu[a + b + c + 1, t]$.
   (d) Set $\mu = \mu[t] = \mu'[a + b + c]||\mu[a + b + c + 1, t]$, where $\mu'[a + b + c]$ is the $m(a + b + c)$-bit zero string.
2. (a) Let $C = 2^c - 1$. Let $\psi'' : [1, C] \to [1, t]$ be a masking assignment such that $\psi''(j) = \psi_t(e_{i+(C+1-j)2^{a+b}})$ where $j \in [1, C]$.
   (b) Let $y^2 = y^2_0||y^2_1||\ldots||y^2_C$, where $y^2_v = s^3(i + (C - v)2^{a+b}, x, k, \mu, t)||$ $x_{i+(C-v)2^{a+b}}$, for $0 \le v \le C - 1$ and $y^2_C = r_0||r_1||r_3||x_i$.
   (c) Run $\mathsf{Mdef}_{seq}(C, y^2, k, r_2, \psi'')$ to get an output $\mu[a + b + 1, a + b + c]$.
   (d) Set $\mu = \mu[t] = \mu'[a + b]||\mu[a + b + 1, a + b + c]||\mu[a + b + c + 1, t]$, where $\mu'[a + b]$ is the $m(a + b)$-bit zero string.
3. (a) Let $B = 2^b - 1$. Let $\psi''' : [1, B] \to [1, t]$ be a masking assignment such that $\psi'''(j) = \psi_t(e_{i+(B+1-j)2^a})$ where $j \in [1, B]$.
   (b) Let $y^1 = y^1_0||y^1_1||\ldots||y^1_B$, where $y^1_v = s^2(i + (B - v)2^a, x, k, \mu, t)||$ $s^3(i + (B-v)2^a, x, k, \mu, t)||x_{i+(B-v)2^a}$, for $0 \le v \le B-1$ and $y^1_B = r_0||r_2||r_3||x_i$.
   (c) Run $\mathsf{Mdef}_{seq}(B, y^1, k, r_1, \psi''')$ to get an output $\mu[a + 1, a + b]$.
   (d) Set $\mu = \mu[t] = \mu'[a]||\mu[a+1, a+b]||\mu[a+b+1, a+b+c]||\mu[a+b+c+1, t]$, where $\mu'[a]$ is the $ma$-bit zero string.
4. (a) Let $u = 2^a - i$ and $A = 2^a - 1$. Let $\psi'''' : [1, A] \to [1, t]$ be a masking assignment such that $\psi''''(j) = \psi_t(e_{A+2-j})$ where $j \in [1, A]$.
   (b) Let $y^0 = y^0_0||y^0_1||\ldots||y^0_A$ where, $y^0_v = s^1(A + 1 - v, x, k, \mu, t)||s^2(A + 1 - v, x, k, \mu, t)||s^3(A + 1 - v, x, k, \mu, t)||x_{A+1-v}$ for $0 \le v \le A - 1$ and $y^0_A = 1^{3m}||x_1$.
   (c) Run $\mathsf{Mdef}_{seq}(u, y^0, k, r_0, \psi'''')$ to get an output $\mu[a]$.
5. Output $\mu[t] = \mu[a]||\mu[a + 1, a + b]||\mu[a + b + 1, a + b + c]||\mu[a + b + c + 1, t]$.

It is easy to check that $s^j(i, x, k, \mu, t) = r_j$ for $0 \le j \le 3$. Therefore $\mathsf{Mdef}_{4dim}$ is correct for $1 \le i \le 2^a - 1$. If $r_0, r_1, r_2$, and $r_3$ are random strings then so is the output $\mu$ and hence $\psi_t$ is totally correct for $1 \le i \le 2^a - 1$. The other cases are very similar. So we omit the proof for these cases. ■

The following theorem shows that if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF, then so is $\{H_p\}_{p\in\mathcal{P}}$. Using the fact that $\psi_t$ is totally correct, we can prove this theorem in a much similar way in the proof of Theorem 4. So we omit this proof.

**Theorem 8.** ***(Validness of domain extension)*** *In case of 4-dimensional domain extension a totally correct masking assignment is always valid. More precisely, if there is an $(\varepsilon, \eta)$-extended strategy for $\{H_p\}_{p\in\mathcal{P}}$ then there is an $(\frac{\varepsilon}{2^t}, \eta+2^{t+1})$-strategy for $\{h_k\}_{k\in\mathcal{K}}$ whenever $\{H_p\}_{p\in\mathcal{P}}$ is based on a totally correct masking assignment.*

We now show the speed-up of 4-dimensional construction over the sequential construction. For the sake of simplicity we do not describe the case of $t \not\equiv 0 \bmod 4$.

**Theorem 9.** *The speed-up of* 4*-dimensional construction over the sequential construction in Section 4 is by a factor of* $\frac{2^t}{2^{2+t/4}-3}$ *if* $t \equiv 0$ *mod* 4 .

**Proof.** 4-dimensional construction hashes a message of length $n2^t - m(2^t - 1)$ into a digest of length $m$ using $2^a + 2^b + 2^c + 2^d - 3$ parallel rounds. Therefore, if $t \equiv 0$ mod 4 then $4 \times 2^{t/4} - 3$ parallel rounds are need to hash a message of length $n2^t - m(2^t - 1)$. The time taken by a single parallel round is proportional to the time required by a single invocation of the hash function $h_k$. The sequential construction requre $2^t$ invocations of the hash function $h_k$ on a message of length $n2^t - m(2^t - 1)$. Hence, the speed-up of the 4-dimensional construction over the sequential construction is by a factor of $\frac{2^t}{2^{2+t/4}-3}$ if $t \equiv 0$ mod 4. ■

By the definition of the masking assignment of 4-dimensional construction, the following theorem is clear.

**Theorem 10.** *The number of masks for* 4*-dimensional construction is* $t$.

## 6.2 Optimality of the 4-Dimensional Domain Extender

in [4] Mironov proved that among all the sequential algorithms Shoup's algorithm reuses the masks as much as possible. This means that among all the sequential algorithms there is no algorithm which has a more smaller key expansion than Shoup's algorithm.

As Mironov did in [4], we can also ask whether the masks can be re-used even more in the 4-dimensional domain extender. But, luckily, we can easily answer the question using the recent result of Sarkar [8]. Furthermore, using the result, we can prove that the key length expansion of the 4-dimensional domain extender is the minimum possible for any algorithms in a large class of "natural" domain extending algorithms including all the 4-dimensional type algorithms and all the previously proposed algorithms.

In [8] Sarkar provided a generic lower bound on the key length expansion required for securely extending the domain of a UOWHF. He first defined the large class $\mathcal{A}$ of "natural" domain extending algorithms. Then he proved that for any $A \in \mathcal{A}$ such that $A$ is correct for $s$ invocations of $h_k$ the number of masks required by $A$ is at least $\lceil \log_2 s \rceil$. (Details can be found in section 4 of [8].) Note that Shoup's algorithm is an element of the class $\mathcal{A}$. Therefore, it follows that Shoup's algorithm is optimal for the class $\mathcal{A}$.

On the other hand the 4-dimensional domain extender is also an element of the class $\mathcal{A}$. And note that for $2^t$ invocations of $h_k$ the 4-dimensional domain extender uses $t(= \lceil \log_2 2^t \rceil)$ masks to securely extend the domain of a UOWHF. Hence this shows that the 4-dimensional domain extender is also optimal for the class $\mathcal{A}$.

## 6.3 *l*-Dimensional Domain Extender

In the above we provided the 4-dimensional domain extender and considered the security and optimality of key length expansion. In fact the construction idea

**Table 2.** Specific comparison of domain extenders for UOWHF.

| Parameter | Shoup [10] | **$l$-DIM**$(l \geq 2)$ | **IBTC** | Sarkar [8] |
|---|---|---|---|---|
| seq/par | sequential | parallel | parallel | parallel |
| message length | $2^t n$ $-(2^t-1)m$ | $2^t n$ $-(2^t-1)m$ | $(2^t-1)n$ $-(2^t-2)m$ | $(2^t-1)n$ $-(2^t-2)m$ |
| # invocations of $h_k$ | $2^t$ | $2^t$ | $2^t-1$ | $2^t-1$ |
| # masks | $t$ | $t$ | t+ $O(\log_2^* t)$ | $t+\lceil \log_2 t \rceil - 1$ |
| # rounds | $2^t$ | $l2^{t/l}-l+1(t \equiv 0 \bmod l)$ | $t$ | $t$ |
| speed-up | 1 | $\frac{2^t}{l2^{t/l}-l+1}(t \equiv 0 \bmod l)$ | $\frac{2^t-1}{t}$ | $\frac{2^t-1}{t}$ |

can be generalized to any $l$-dimensional domain extender ($l \geq 2$). If $n \geq lm$, we can define the $l$-dimensional domain extender. We can start to define the $l$-dimensional domain extender with setting the function $g(t) = (a_1, \cdots, a_l)$ exactly in the similar way as we did for 4-dimensional. And the whole specification of $l$-dimensional domain extender can be similarly defined by using the description method of the 4-dimensional domain extender. We can also consider the security and optimality of the $l$-dimensional domain extender as in the case of 4-dimensional domain extender.

## 7 Comparison to Known Algorithms

In Table 2 we compare the specific performance of the different known algorithms with $l$-dimensional domain extender and Improved binary tree based construction. Note that the message length which can be handled varies with each of the known algorithms. For example, Shoup's and $l$-DIM can handle a $2^t n-(2^t-1)m$ bits message, however, Sakar's and IBTC can not handle the same length message. Therefore, we can not fix a message length in order to compare the different known algorithms with $l$-DIM and IBTC. Instead, we separately describe the message length for each of the algorithms as shown in Table 2.

The algorithms use one key for the base hash function and some number of $m$-bit mask keys. The number of masks described in Table 2 refers to the latter. The number of invocations of $h_k$ is the total cost. The number of rounds reflects the parallelizability arising via tree-based constructions, and indicates the total time to completion. In Shoup's sequential construction it is equal to the number of invocations of $h_k$. Speed-up (over the sequential algorithm or Shoup) is the ratio of the number of invocations of $h_k$ to that of rounds. For the sake of simplicity we do not describe the case of $t \not\equiv 0 \bmod l$ in the positions of the number of rounds and speed for our $l$-DIM.

Table 2 shows the key length expansion of $l$-DIM is the same as that of Shoup's and it doesn't have the same parallalizable performance with IBTC and Sarkar's construction. But if $l$ is getting larger, then the speed of the $l$-DIM is also getting near to the speed of IBTC and Sarkar. On the contrary, the parallalizable performance of IBTC is the same as that of Sarkar's and it doesn't have the same key length expansion with $l$-DIM and Shoup's construction.

## 8 Conclusion

In this paper we have provided two parallel domain extenders, IBTC and $l$-DIM, for UOWHF. Each of them has an important theoretical meaning in the study of efficient domain extanding method for UOWHF.

IBTC has the most efficient key length expansion among all the previously known *complete* $l$-ary ($l \geq 2$) tree based parallel constructions. But IBTC need slightly more key length expansion than Shoup's sequential construction. On the other hand, $l$-DIM has the same key length expansion as Shoup's. Furthermore, $l$-DIM and Shoup's construction are the minimum possible for any algorithms in a large class of "natural" domain extenders including all the previously proposed constructions. But $l$-DIM does not have the same parallelizability performance as complete $l$-ary ($l \geq 2$) tree based constructions.

This paper has concerned the efficient parallel construction. Of course, it would be very nice to have a parallel construction which has the optimal key length expansion and the same or more efficient parallelizability than complete tree based constructions simultaneously. But at this point, we do not have any such algorithm. Hence, in our opinion, we should separately consider both the key length expansion and the parallelizability with the same importance. And we would like to stress that the present work is important in regarding the former and the latter point of views, respectively.

We have also given a sufficient condition for valid domain extension for sequential extension and it is likely that the condition is necessary. So, that will characterize the valid domain extension for sequential construction.

It is likely that $l$-DIM has maximum parallelizability with optimal key length expansion. So one can try to prove whether this parallelizability is maximum among all the constructions with optimal key length expansion or not.

## Acknowledgments

We are grateful to Palash Sarkar for valuable discussions and unconditional encouragement. We also thank a number of anonymous referees for their helpful comments.

## References

1. M. Bellare and P. Rogaway. *Collision-resistant hashing: towards making UOWHFs practical*, Advances in Cryptology - Crypto'97, Lecture Notes in Computer Science, Vol. 1294, Springer-Verlag, pp. 470-484, 1997.
2. I. B. Damgard. *A design principle for hash functions*, Advances in Cryptology - Crypto'89, Lecture Notes in Computer Sciences, Vol. 435, Springer-Verlag, pp. 416-427, 1989.
3. R. Merkle. *One way hash functions and DES*, Advances in Cryptology - Crypto'89, Lecture Notes in Computer Sciences, Vol. 435, Springer-Verlag, pp. 428-446, 1989.

4. I. Mironov. *Hash functions: from Merkle-Damgard to Shoup*, Advances in Cryptology - Eurocrypt'01, Lecture Notes in Computer Science, Vol. 2045, Springer-Verlag, pp 166-181, 2001
5. M. Nandi. *A New Tree based Domain Extension of UOWHF*, Cryptology ePrint Archive, `http://eprint.iacr.org/2003/142`.
6. M. Nandi. *Study of Domain Extension of UOWHF and its Optimality*, Cryptology ePrint Archive, `http://eprint.iacr.org/2003/158`.
7. M. Naor and M. Yung. *Universal one-way hash functions and their cryptographic applications*, Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing, ACM Press, pp 33-43, 1989.
8. P. Sarkar. *Construction of UOWHF: Tree Hashing Revisited*, Cryptology ePrint Archive, `http://eprint.iacr.org/2002/058`.
9. P. Sarkar. *Domain Extenders for UOWHF: A Generic Lower Bound on Key Expansion and a Finite Binary Tree Algorithm*, Cryptology ePrint Archive, `http://eprint.iacr.org/2003/009`.
10. V. Shoup. *A composition theorem for universal one-way hash functions.* Advances in Cryptology - Eurocrypt'00, Lecture Notes in Computer Science, Vol. 1807, Springer-Verlag, pp 445-452, 2000.
11. D. Simon. *Finding collisions on a one-way street: can secure hash functions be based on general assumptions?*, Advances in Cryptology - Eurocrypt'98, Lecture Notes in Computer Science, Vol. 1403, Springer-Verlag, pp 334-345, 1998.