# Construction of UOWHF: Tree Hashing Revisited

Palash Sarkar
Cryptology Research Centre
Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road
Kolkata 700035, India
e-mail: palash@isical.ac.in

**Abstract**

We present a binary tree based parallel algorithm for extending the domain of a UOWHF. The key length expansion is $2m$ bits for $t = 2$; $m(t+1)$ bits for $3 \leq t \leq 6$ and $m \times (t + \lfloor \log_2(t-1) \rfloor)$ bits for $t \geq 7$, where $m$ is the length of the message digest and $t \geq 2$ is the height of the binary tree. The previously best known binary tree algorithm required a key length expansion of $m \times 2(t-1)$ bits. We also obtain the lower bound that any binary tree based algorithm must make a key length expansion of $2m$ bits if $t = 2$ and a key length expansion of $m \times (t+1)$ bits for $t \geq 3$. Hence for $2 \leq t \leq 6$ our algorithm makes optimal key length expansion and for practical sized processor trees the key length expansion is close to the lower bound.

**Keywords :** UOWHF, hash function, binary tree.

## 1 Introduction

Digital signature schemes are important constituents of modern cryptography. Customarily digital signatures are built out of trapdoor one-way functions. However, Naor and Yung [5] have shown that it is possible to build secure digital signature schemes from 1-1 one-way functions. This construction is important since a one-way function is a weaker primitive than a trapdoor one-way function. A key component of the Naor-Yung construction is universal one-way hash function (UOWHF), which was also introduced in [5].

A UOWHF is a keyed family of functions and is a weaker primitive than the usual collision resistant function. For a usual collision resistant hash function (CRHF), the adversary has to find a collision for the fixed hash function. On the other hand, in case of UOWHF the adversary has to commit to an input and then the function for which the adversary has to find a collision is specified. The adversary wins if he can successfully find a collision for the specified function. Since the adversary has to commit to the input before the function is specified, the adversary's task is more difficult than in the case of CRHF and hence a UOWHF is a weaker primitive. In fact, Simon [8] has shown that there is an oracle relative to which UOWHFs exist but not CRHFs.

There is another important practical reason for preferring UOWHFs to CRHFs. As mentioned in [1], the birthday attack does not apply to UOWHFs. Hence the size of the message digest can be significantly shorter.

From the above discussion it follows that it is important to look for efficient constructions of UOWHFs. However, like most basic cryptographic primitives (say symmetric ciphers) it is virtually impossible to construct a keyed family of hash functions and prove it to be a UOWHF. In view of this, the approach suggested by Bellare and Rogaway [1] is to key one of the standard hash functions like SHA-1 or RIPEMD-160 and assume it to be a UOWHF. It seems more reasonable to make this assumption when the input is a short fixed length string rather than in the case where the input can be arbitrarily long strings.

This brings us to the problem of extending the domain of UOWHF in a secure manner. For CRHF a technique for doing this has been described by Merkle [3] and Damgård [2]. However, in [1] it has been shown that this construction fails for UOWHFs. A consequence of this result is that any extension of the domain of a UOWHF entails an increase in the size of the key to the hash function. It has been shown in [1] that by signing $(k, h_k(x))$ ($k$ is the key, $x$ is the message and $h_k$ is the hash function) it is possible to build a secure signature scheme. *Thus minimizing the size of $k$ is of great practical significance. In other words, it is important to look for constructions which extend the domain of a UOWHF and for which the resulting increase in key length is the minimum possible.*

A sequential construction for extending UOWHF based on the Merkle-Damgård construcion was obtained by Shoup [7]. The scheme requires a key length expansion of $t \times m$, where $m$ is the size of the message digest and $2^t - 1$ is the number of times the hash function $h_k$ is invoked. (The Shoup construction works even if the number of invocations of $h_k$ is not of the form $2^t - 1$.) In a later work, Mironov [4] proved the key length expansion to be optimal for the Shoup construction.

For practical purposes, it is of interest to consider parallel hashing schemes. Binary tree based hash algorithm will provide speed-up by a factor of $\frac{2^t}{t}$ over the sequential algorithm to hash a message of length $2^t(n-m) - (n-2m)$ using a binary tree of height $t$ (see Proposition 2). This speed-up can provide substantial savings in time for hashing long messages especially in situations where such computations have to be performed repeatedly.

A tree based construction for securely extending the domain of UOWHF was described in [1]. For binary trees the construction required a key length expansion of $m \times 2(t-1)$, for a binary tree of $2^t - 1$ processors (and hence $2^t - 1$ invocations of $h_k$).

In this work, we consider binary tree based algorithm for extending the domain of a UOWHF. We show that the construction in [1] is not optimal and present a binary tree based parallel scheme for extending the domain of a UOWHF. The key length expansion is $2m$ bits for $t = 2$; $m(t+1)$ bits for $3 \le t \le 6$ and $m \times (t + \lfloor \log_2(t-1) \rfloor)$ bits for $t \ge 7$. This is a significant improvement over the scheme in [1]. The improvement is achieved by using the Shoup construction along certain paths in the binary tree. We use the proof technique used in [4] to show the correctness of our construction.

We obtain a lower bound on the amount of key expansion required by any binary tree based algorithm for extending the domain of a UOWHF. We show that the key length must increase by at least $2m$ bits if $t = 2$ and by at least $m \times (t+1)$ bits if $t \ge 3$. Hence for $2 \le t \le 6$ our algorithm makes optimal key length expansion. Further, for $t = 7, 8$, the key length expansion made by our algorithm is $m$ bits more than the lower bound and hence is nearly optimal. Note that practical processor trees will usually have $t \le 8$.

A consequence of our lower bound result is that the key length expansion made by any binary tree based algorithm must be $m$ bits more than the key length expansion made by the Shoup construction, which is a sequential algorithm. This suggests that there will be a trade-off of at least $m$ bits in key length expansion for achieving speed-up through parallelism.

For binary tree based algorithms with $t \ge 7$, there is a gap between the lower bound $(m \times (t+1))$ and what has been achieved $(m \times (t + \lfloor \log_2(t-1) \rfloor))$. It is an open problem to try and close this gap.

## 2 Preliminaries

Let $\{h_k\}_{k\in\mathcal{K}}$ be a keyed family of hash functions, where each $h_k : \{0,1\}^n \rightarrow \{0,1\}^m$. *In this paper we require $n \geq 2m$.* 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 $x \neq x'$ and $h_k(x) = h_k(x')$.

We say that $\{h_k\}_{k\in\mathcal{K}}$ is a universal one way hash family (UOWHF) if the adversary has a negligible probability of success with respect to any probabilistic polynomial time strategy. 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 $s$. In the second stage $\mathcal{A}^{\text{find}}(x, k, s)$, the adversary either finds a $x'$ which provides a collision for $h_k$ or it reports failure. Both $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x, k, s)$ are probabilistic algorithms. The success probability of the strategy is measured over the random choices made by $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x, k, s)$ and the random choice of $k$ in step 2 of the game. We say that $\mathcal{A}$ is an $(\epsilon, a)$-strategy if the success probability of $\mathcal{A}$ is at least $\epsilon$ and it invokes the hash function $h_k$ at most $a$ times. In this case we say that the adversary has an $(\epsilon, a)$-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.

In this paper we are interested in extending the domain of a UOWHF. Thus given a UOWHF $\{h_k\}_{k\in\mathcal{K}}$, with $h_k : \{0,1\}^n \rightarrow \{0,1\}^m$ and a positive integer $L \geq n$, we would like to construct another UOWHF $\{H_p\}_{p\in\mathcal{P}}$, with $H_p : \{0,1\}^L \rightarrow \{0,1\}^m$. We say that the adversary has an $(\epsilon, a)$-strategy for $\{H_p\}_{p\in\mathcal{P}}$ if there is a strategy $\mathcal{B}$ for the adversary with probability of success at least $\epsilon$ and which invokes the hash function $h_k$ at most $a$ times. 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 $(\epsilon, a)$-strategy for $\{H_p\}_{p\in\mathcal{P}}$, then there is an $(\epsilon_1, a_1)$-strategy for $\{h_k\}_{k\in\mathcal{K}}$, where $a_1$ is not much larger than $a$ and $\epsilon_1$ is not significantly lesser than $\epsilon$. This will show that if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF, then so is $\{H_p\}_{p\in\mathcal{P}}$.

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 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 $L$ 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 a major motivation is to minimise this increase in the key length.

## 3 Known Constructions

We briefly discuss the constructions which have already been proposed.

## 3.1 Sequential Construction

The Merkle-Damgård construction is a well known construction for collision resistant hash functions. However, Bellare and Rogaway [1] showed that the construction does not directly work in the case of UOWHF. In [7], Shoup presented a modification of the MD construction. We briefly describe the Shoup construction as presented in [4].

Let $\{h_k\}_{k\in\mathcal{K}}$ be the base family, where $\mathcal{K} = \{0,1\}^K$. Let $x$ be the input to $H_p$ with $|x| = r(n-m)$. We define $p = k||m_0||m_1||\ldots||m_{l-1}$ where $l = 1 + \lfloor \log r \rfloor$ and $m_i$ are $m$-bit binary strings called masks. The increase in key length is $lm$ bits. The output of $H_p$ is computed by the following algorithm.

1. Let $x = x_1||x_2||\ldots||x_r$, where $|x_i| = n - m$.
2. Let IV be an $n$-bit initialisation vector.
3. Define $z_0 = IV$, $s_0 = z_0 \oplus m_0$.
4. For $1 \le i \le r$, define $z_i = h_k(s_{i-1}||x_i)$ and $s_i = z_i \oplus m_{\nu(i)}$ where $\nu(i) = j$ if $2^j|i$ and $2^{j+1} \not| i$.
5. Define $z_r$ to be the output of $H_p(x)$.

The function $h_k$ is invoked $r$ times and is called the $r$-round Shoup construction. The construction was proved to be correct by Shoup in [7]. In a later work Mironov [4] provided an alternative correctness proof. More importantly, in [4] it was shown that the amount of key length expansion is the minimum possible for the construction to be correct.

## 3.2 Tree Based Construction

In [1] Bellare and Rogaway described a tree based construction for extending UOWHF. We briefly describe the construction for binary trees.

Consider a full binary tree with $t$ levels numbered $1,\ldots,t$. There are $2^{i-1}$ nodes at level $i$. Hence the total number of nodes in the tree is $2^t - 1$. The nodes are numbered 1 to $2^t - 1$ in the usual fashion (top to bottom and left to right). At each node $i$ there is a processor $P_i$, which is capable of computing the base hash function $h_k$. For the tree based construction we require that $n \ge 2m$. Let $x$ be the input to the hash function $\{H_p\}_{p\in\mathcal{P}}$. Here $p = k||\alpha_1||\beta_1||\ldots||\alpha_{t-1}||\beta_{t-1}$, where $\alpha_i$ and $\beta_j$ are $m$-bit strings called masks. The computation of the function $H_p(x)$ is the following.

1. Write $x = x_1||x_2||\ldots||x_{2^t-1}$, where $|x_1| = \ldots = |x_{2^{t-1}-1}| = n - 2m$ and $|x_{2^{t-1}}| = \ldots = |x_{2^t-1}| = n$.
2. For $2^{t-1} \le i \le 2^t - 1$, do
   (a) compute $z_i = h_k(x_i)$.
   (b) If $i$ is even, compute $s_i = z_i \oplus \alpha_{t-1}$ and if $i$ is odd, compute $s_i = z_i \oplus \beta_{t-1}$.
3. For $i = 2^{t-1} - 1$ downto 2, do
   (a) Let $j = level(i)$.
   (b) Compute $z_i = h_k(s_{2i}||s_{2i+1}||x_i)$.

    (c) If $i$ is even, compute $s_i = z_i \oplus \alpha_{j-1}$ and if $i$ is odd, compute $s_i = z_i \oplus \beta_{j-1}$.

4. Define $h_k(s_2||s_3||x_1)$ to be the output of $H_p(x)$.

Here $level(i)$ is the level of the tree to which $i$ belongs. Thus $level(i) = j$ if $2^{j-1} \le i \le 2^j - 1$. It is clear that all the nodes at the same level can work in parallel. We note that in the original algorithm in [1], the strings $x_1, \ldots, x_{2^{t-1}-1}$ were defined to be empty strings.

The amount of key expansion is $2(t-1)m$ bits for a tree with $t$ levels. Thus $2(t-1)$ masks each of length $m$ bits are required by the construction. We will call the above construction the BR construction.

## 4 Improved Tree Based Construction

There are $2^t - 1$ processors $P_1, \ldots, P_{2^t-1}$ connected in a full binary tree of $t$ levels numbered $1, \ldots, t$ with processors $P_{2^{i-1}}, \ldots, P_{2^i-1}$ at level $i$. The arcs in the binary tree point towards the parent, i.e. the arcs are of the form $(2i, i)$ and $(2i+1, i)$. Each processor is capable of computing the function $h_k$ for any $k \in \mathcal{K}$, i.e., $P_i(k, x) = h_k(x)$, for an $n$-bit string $x$. *In the rest of the paper we will always assume that $t \ge 2$.*

The input to the function $H_p$ is $x$ of length $2^{t-1}n + (2^{t-1}-1)(n-2m)$. The key $p$ for the function $H_p$ is formed out of the key $k$ for the function $h_k$ plus some additional $m$-bit strings. For convenience in describing the algorithm we divide these additional $m$-bit strings into two *disjoint* sets $\Gamma = \{\alpha_1, \ldots, \alpha_{t-1}\}$ and $\Delta = \{\beta_0, \ldots, \beta_{l-1}\}$, where $l = 1 + \lfloor \log_2(t-1) \rfloor$. The $m$-bit strings $\alpha_i$ and $\beta_j$ will be called masks. Recall that for integer $i$, the function $\nu(i) = j$ if $2^j | i$ and $2^{j+1} \not| i$.

**Improved Tree Construction (ITC)**

```
1. Let $x = x_1||\ldots||x_{2^t-1}$, where $|x_1| = \ldots = |x_{2^{t-1}-1}| = n - 2m$ and $|x_{2^{t-1}}| = \ldots = |x_{2^t-1}| = n$.
   Note $|x| = 2^t(n-m) - (n-2m)$.

2. For $2^{t-1} \le i \le 2^t - 1$, do in parallel
   (a) $z_i = P_i(k, x_i) = h_k(x_i)$.
   (b) Set $s_i = z_i \oplus \beta_0$ if $i$ is even and set $s_i = z_i \oplus \alpha_1$ if $i$ is odd.

3. For $j = t-1$ downto 2 do
   • For $i = 2^{j-1}$ to $2^j - 1$ do in parallel
     (a) $z_i = P_i(k, s_{2i}||s_{2i+1}||x_i) = h_k(s_{2i}||s_{2i+1}||x_i)$.
     (b) Set $s_i = z_i \oplus \beta_{\nu(t-j+1)}$ if $i$ is even and set $s_i = z_i \oplus \alpha_{t-j+1}$ if $i$ is odd.

4. Define the output of $H_p(x)$ to be $h_k(s_2||s_3||x_1)$.
```

**Remark :** Note that in algorithm ITC the processors at one level operate in parallel. Further, when the processors at one level are working, the processors at all other levels are idle. Thus processors can be reused and only $2^{t-1}$ processors are actually required to implement the algorithm. However, for the sake of clarity in further analysis, we will assume that $2^t - 1$ "virtual" processors are available.

We provide an explanation of the construction. Let $P = P_{i_r}P_{i_{r-1}}\ldots P_{i_1}$ be a path of processors of length $r$ from the leaf node $P_{i_r}$ to some internal node $P_{i_1}$ which is obtained by following only left links, i.e., $level(i_r) = t$ and $i_{j+1} = 2i_j$ for $j = 1, \ldots, r-1$. The arcs $(i_{j+1}, i_j)$ in the path are assigned masks according to the Shoup construction. Let $S$ be the set of arcs $\{(2i_1+1, i_1), (2i_2+1, i_2), \ldots, (2i_{r-1}+1, i_{r-1})\}$. The construction also ensures that no two arcs in $S$ get the same mask.

**Proposition 1** *The following are true for algorithm ITC.*

1. *$t$ parallel rounds are required to compute the output.*
2. *The function $h_k$ is invoked $2^t - 1$ times.*
3. *The amount of key length expansion ($|p| - |k|$) is $m(t + \lfloor \log_2(t-1) \rfloor$ bits.*

**Proof.** **(1)** Step 2 of ITC is one parallel round. Step 3 requires $(t-2)$ parallel rounds and Step 4 requires one round. Hence a total of $t$ rounds are required.
**(2)** There are $2^t - 1$ processors and each processor invokes the function $h_k$ exactly once. Hence $h_k$ is invoked exactly $2^t - 1$ times.
**(3)** The amount of key length expansion is $m \times |\Gamma \cup \Delta|$. By definition $|\Gamma| = t-1$ and $|\Delta| = 1 + \lfloor \log_2(t-1) \rfloor$. Also $\Gamma \cap \Delta = \emptyset$. ■
**Remark :** The amount of expansion in the BR construction is $2(t-1)m$ bits. Thus with respect to key length expansion ITC is a significant improvement over the BR construction.

**Proposition 2** *The speed-up of Algorithm ITC over the sequential algorithm in Section 3.1 is by a factor of $\frac{2^t}{t}$.*

**Proof.** Algorithm ITC hashes a message of length $2^t(n-m) - (n-2m)$ 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 $2^t$ invocations of the hash function $h_k$ on a message of length $2^t(n-m) - (n-2m)$. Hence the speed-up of the binary tree algorithm over the sequential algorithm is by a factor of $\frac{2^t}{t}$. ■
**Remark :** The speed-up achieved by Algorithm ITC is substantial even for moderate values of $t$. Such speed-up will prove to be advantageous for hashing long messages.

**Theorem 3 (Security Reduction for $H_p$)** *If there is an $(\epsilon, a)$ winning strategy $\mathcal{A}$ for $\{H_p\}_{p\in\mathcal{P}}$, then there is an $(\frac{\epsilon}{2^t-1}, a + 2(2^t-1))$ winning strategy $\mathcal{B}$ for $\{h_k\}_{k\in\mathcal{K}}$. Consequently, $\{H_p\}_{p\in\mathcal{P}}$ is a UOWHF if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF.*

**Proof.** We describe the two stages of the strategy $\mathcal{B}$ as follows.
$\mathcal{B}^{\text{guess}}$ **:** (output $(y, s)$, with $|y| = n$.)

1. Run $\mathcal{A}^{\text{guess}}$ to obtain $x \in \{0,1\}^L$ and state information $s'$.
2. Choose an $i$ uniformly at random from the set $\{1, \ldots, 2^t - 1\}$.
3. Write $x = x_1 || \ldots || x_{2^t-1}$, where $|x_1| = \ldots = |x_{2^{t-1}-1}| = n - 2m$ and $|x_{2^{t-1}}| = \ldots = |x_{2^t-1}| = n$.
4. If $2^{t-1} \le i \le 2^t - 1$, set $y = x_i$; $u_1, u_2$ to be the empty string and $s = (s', i, u_1, u_2)$. Output $(y, s)$ and stop.

5. If $1 \le i \le 2^{t-1} - 1$, then choose two strings $u_1$ and $u_2$ uniformly at random from the set $\{0,1\}^m$. Set $y = u_1||u_2||x_i$ and $s = (s', i, u_1, u_2)$. 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.

$\mathcal{B}^{\textbf{find}}(y, k, s)$ : (Note $s = (s', i, u_1, u_2)$.)

1. Define the masks $\alpha_1, \ldots, \alpha_{t-1}, \beta_0, \ldots, \beta_{l-1}$ by executing algorithm $\mathsf{MDef}(i, u_1, u_2)$ (called the mask defining algorithm). This defines the key $p$ for the function $H_p$. Here $p = k||\alpha_1|| \ldots ||\alpha_{t-1}||\beta_0|| \ldots ||\beta_{l-1}$, where $l = \lfloor \log_2(t-1) \rfloor + 1$.
2. Run $\mathcal{A}^{\text{find}}(x, p, s')$ to obtain $x'$.
3. Let $v$ and $v'$ be the inputs to processor $P_i$ corresponding to the strings $x$ and $x'$ respectively. Denote the corresponding outputs by $z_i$ and $z_i'$. If $z_i = z_i'$ and $v \neq v'$, then output $v$ and $v'$, else output "failure".

Note that Step 3 either detects a collision or reports failure. We now lower bound the probability of success. But first we have to specify the mask defining algorithm.

The task of the mask defining algorithm $\mathsf{MDef}$ is to define the masks $\alpha_1, \ldots, \alpha_{t-1}, \beta_0, \ldots, \beta_{l-1}$ (and hence $p$) so that the input to processor $P_i$ is $y$. Note that the masks are not defined until the key $k$ is given to the adversary. Once the key $k$ is specified we extend it to $p$ such that the extension is "consistent" with the input $y$ to $P_i$ to which the adversary has already committed. Another point that one has to be careful about is to ensure that the key $p$ is chosen uniformly at random from the set $\mathcal{P}$, i.e., the masks $\alpha_i$ and $\beta_j$ are chosen independently and uniformly to be $m$-bit strings.

The mask defining algorithm $\mathsf{MDef}$ is given below. The algorithm uses an array $A[]$ of length at most $(t-1)$ whose entries are pairs of the form $(j, v)$ where $j$ is an integer in the range $1 \le j \le 2^t - 1$ and $v$ is an $m$-bit string.

**Algorithm** $\mathsf{MDef}(i, u_1, u_2)$
(Note : $i$ was chosen by $\mathcal{B}^{\text{guess}}$ in Step 2. $u_1$ and $u_2$ were chosen by $\mathcal{B}^{\text{guess}}$ either in Step 4 or in Step 5.)

```
1. If $2^{t-1} \le i \le 2^t - 1$, then randomly define the masks $\alpha_1, \ldots, \alpha_{t-1}, \beta_0, \ldots, \beta_{l-1}$ and exit.
2. Append $(2i+1, u_2)$ to the array $A$.
3. Let $j = t - \mathit{level}(i)$, $j_1 = j - 2^{\nu(j)}$ and $i_1 = 2^{j-j_1}i$.
4. Randomly define all undefined masks in the set $\beta_{\nu(j_1+1)}, \ldots, \beta_{\nu(j-1)}$.
5. If $j_1 = 0$, then $z_{i_1} = h_k(x_{i_1})$,
6. else
   (a) randomly choose $u, v$ in $\{0,1\}^m$.
   (b) Append $(2i_1+1, v)$ to the array $A$.
   (c) $z_{i_1} = h_k(u||v||x_{i_1})$.
7. For $j_2 = j_1 + 1, \ldots, j - 1$ do
```

```
        (a) $i_2 = 2^{j-j_2} i$.
        (b) $s_{2i_2} = z_{2i_2} \oplus \beta_{\nu(j_2)}$.
        (c) Randomly choose $w$ in $\{0,1\}^m$.
        (d) Append $(2i_2 + 1, w)$ to the array $A$.
        (e) $z_{i_2} = h_k(s_{2i_2} || w || x_{i_2})$.
    8. $\beta_{\nu(j)} = z_{2i} \oplus u_1$.
    9. If $j_1 > 0$, then $u_1 = u$, $u_2 = v$, $j = j_1$ and go to Step 2.
    10. Randomly define all as yet undefined masks $\beta_i$, $0 \le i \le l-1$.
    11. Sort the array $A$ in descending order based on the first component of each entry $(j, v)$.
    12. For $i_1 = 1$ to $t - level(i)$ do
        (a) Let $(l, u) = A[i_1]$.
        (b) Compute $z_l$ to be the output of processor $P_l$. (This can be done, since at this point all
            masks used in the subtree rooted at $l$ have already been defined.)
        (c) Let $j = t - level(l) + 1$.
        (d) Define $\alpha_j = z_l \oplus u$.
    13. Randomly define all as yet undefined masks $\alpha_j$, $1 \le j \le t-1$.
```

Intuitively, algorithm MDef applies the mask reconstruction algorithm for the Shoup construction along the path $P_{i_r}, P_{i_{r-1}}, \ldots, P_{i_1}$, where $i_1 = i$, $i_j = 2^{j-1} i$ and $level(i_r) = t$. This defines the masks $\beta_{\nu(t-level(i_j))}$ for $1 \le j < r$. To do this the algorithm guesses the inputs that the processors $P_{i_1}, \ldots, P_{i_{r-1}}$ obtain from their right descendants. These inputs along with the proper processor numbers are added to the array $A$. Once the definition of the $\beta$ masks are complete, the algorithm begins the task of defining the $\alpha$ masks. The first element of the array $A$ is $(2i_{r-1} + 1, u)$ for some $m$-bit string $u$ and we are required to define $\alpha_1$. The processor $P_{2i_{r-1}+1}$ is at the leaf level and applies $h_k$ to $x_{2i_{r-1}+1}$ to produce $z_{2i_{r-1}+1}$. Now $\alpha_1$ is defined to be the XOR of $u$ and $z_{2i_{r-1}+1}$. Suppose for some $2 \le j \le r$, the masks $\alpha_1, \ldots, \alpha_{j-1}$ has already been defined. The current element of the array $A$ is $(2i_{r-j}+1, u)$ for some $m$-bit string $u$. At this point all masks present in the subtree rooted at processor $P_{2i_{r-j}+1}$ have already been defined. Thus the input to processor $P_{2i_{r-j}+1}$ is known. Hence processor $P_{2i_{r-j}+1}$ applies the hash function $h_k$ to its input to obtain the string $z_{2i_{r-j}+1}$. The mask $\alpha_j$ is now defined to be the XOR of $u$ and $z_{2i_{r-j}+1}$.

Notice that this procedure ensures that the input to processor $P_i$ is the string $y$ to which $\mathcal{B}^{\text{guess}}$ has committed. We now argue that the masks are chosen randomly from the set $\{0,1\}^m$. For this we note that in MDef each mask is either chosen to be a random string or is obtained by XOR with a random string. Hence all the masks are random strings from the set $\{0,1\}^m$. Also $k$ is a random string and hence $p$ is a randomly chosen key from the set $\mathcal{P}$.

Suppose $x$ and $x'$ collides for the function $H_p$. Then there must be a $j$ in the range $1 \le j \le 2^t - 1$ such that processor $P_j$ provides 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 $\frac{1}{2^t - 1}$. 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. ■

## 4.1 Improvement on Algorithm ITC for $t = 5, 6$

Algorithm ITC uses two disjoint sets of masks $\Gamma$ and $\Delta$. For $t = 5, 6$, we have $\Gamma = \{\alpha_1, \ldots, \alpha_{t-1}\}$ and $\Delta = \{\beta_0, \beta_1, \beta_2\}$. This results in a total of $t + 2$ distinct masks. The next result shows that $t + 1$ masks are sufficient.

**Theorem 4** *For $t = 5, 6$, it is possible to properly extend a UOWHF $\{h_k\}_{k \in \mathcal{K}}$ to a UOWHF $\{H_p\}_{p \in \mathcal{P}}$ using a processor tree of $2^t - 1$ processors and requiring exactly $t + 1$ masks.*

**Proof.** The algorithm is same as Algorithm ITC with the following small modification. In Algorithm ITC the sets $\Gamma = \{\alpha_1, \ldots, \alpha_{t-1}\}$ and $\Delta = \{\beta_0, \beta_1, \beta_2\}$ are disjoint. We remove this disjointness by setting $\alpha_1 = \beta_2$. This results in a total of $t + 1$ masks.

We have to show that setting $\alpha_1 = \beta_2$ does not affect the correctness of the construction. More precisely, we have to provide a security reduction similar to that of Theorem 3. A close examination of the proof of Theorem 3 shows that the only part of the proof which will be affected by setting $\alpha_1 = \beta_2$ is the mask defining algorithm. Thus it is sufficient to describe a proper mask defining algorithm. We describe the mask defining algorithm for $t = 6$ which will also cover the case $t = 5$.

Let the processors in $T_6 = (V_6, A_6)$ be $P_1, \ldots, P_{63}$. Suppose the output of $\mathcal{B}^{\text{guess}}$ is $(y, s = (s', i, u_1, v_1))$. If $i \geq 4$, then the mask defining algorithm of Theorem 3 is sufficient to define all the masks. This is because of the fact that in Algorithm ITC the mask $\beta_2$ does not occur in the subtree rooted at $i$ and hence we are required to define only $\alpha_1$. The problem arises when we have to define both $\alpha_1$ and $\beta_2$ using Algorithm MDef. Since in this case $\alpha_1 = \beta_2$, defining one will define the other. Thus we have to ensure that this particular mask is not redefined.

There are three values of $i$ that we have to consider, namely $i = 1, 2$ and 3. The cases 2 and 3 are essentially the same and correspond to the case for $t = 5$. Thus there are only two cases to consider. We describe the case $i = 1$, the other case ($i = 2, 3$) being similar. The following sequence of steps properly defines all the masks when $i = 1$.

1. Randomly choose two $m$-bit strings $u_2$ and $v_2$. Define $\beta_0 = u_1 \oplus h_k(u_2 || v_2 || x_2)$.
2. Randomly choose two $m$-bit strings $u_3$ and $v_3$.
   (a) Set $w_1 = h_k(u_3 || v_3 || x_8)$.
   (b) Set $w_2 = w_1 \oplus \beta_0$.
   (c) Randomly choose an $m$-bit string $v_4$.
   (d) Set $w_3 = h_k(w_2 || v_4 || x_4)$.
   (e) Define $\beta_2 = u_2 \oplus w_3$.
3. (a) Set $w_4 = \beta_0 \oplus h_k(x_{32})$.
   (b) Set $w_5 = \alpha_1 \oplus h_k(x_{33})$. (Note that $\alpha_1 = \beta_2$ has been defined in Step 2(e).)
   (c) Define $\beta_1 = u_3 \oplus h_k(w_4 || w_5 || x_{16})$.
4. Compute the output of processor $P_{17}$ and call it $w_6$. Define $\alpha_2 = w_6 \oplus v_3$.
5. Compute the output of processor $P_9$ and call it $w_7$. Define $\alpha_3 = w_7 \oplus v_4$.
6. Compute the output of processor $P_5$ and call it $w_8$. Define $\alpha_4 = w_8 \oplus v_2$.
7. Compute the output of processor $P_3$ and call it $w_9$. Define $\alpha_5 = w_9 \oplus v_2$.

It is not difficult to verify that the above algorithm properly defines all the masks. Further, each mask is obtained by XOR with a random $m$-bit string and hence the concatenation of all the $(t+1)$ masks is a random bit string of length $m(t+1)$. This completes the proof of the theorem. ∎

**Remark :** It seems difficult to extend the above technique for $t \geq 7$.

## 5 Lower Bound

In this section we obtain a lower bound on the number of masks that must be used for the tree based UOWHF construction to be correct. The lower bound is obtained from a necessary condition which we prove first.

Denote by $T_t = (V_t, A_t)$ the full binary tree with $t$ levels numbered $1, \ldots, t$. Here $V_t = \{1, \ldots, 2^t - 1\}$ is the vertex set of $T_t$ and $A_t$ is the arc set of $T_t$. Recall that the arcs point towards the parents, i.e. the arcs are of the form $(2i, i)$ or $(2i+1, i)$. We enumerate the arc set as $A_t = \{a_2, a_3, \ldots, a_{2^t-1}\}$, where $a_j = (j, \lfloor (j/2) \rfloor)$. Let $M$ be a set of masks and let $\psi_t : A_t \to M$ be an assignment of the masks to the arcs. When $t$ is clear from the context we will simply write $\psi$ instead of $\psi_t$. The general binary tree based algorithm for computing the message digest is given below.

**Algorithm 1 : (computation of $H_p(x)$)**

```
1. Write message $x = x_1||x_2||\ldots||x_{2^t-1}$, where $|x_1| = \ldots = |x_{2^{t-1}-1}| = n - 2m$ and $|x_{2^{t-1}}| = \ldots = |x_{2^t-1}| = n$.
2. For $i = 2^{t-1}, \ldots, 2^t - 1$, do in parallel
    (a) $z_i = P_i(k, x_i) = h_k(x_i)$.
    (b) $s_i = z_i \oplus \psi(a_i)$.
3. For $j = t-1$ downto 2 do
    • For $i = 2^{j-1}$ to $2^j - 1$ do in parallel
        (a) $z_i = P_i(k, s_{2i}||s_{2i+1}||x_i) = h_k(s_{2i}||s_{2i+1}||x_i)$.
        (b) $s_i = z_i \oplus \psi(a_i)$.
4. Output $h_k(s_2||s_3||x_1)$ as the output of $H_p(x)$.
```

**Remark :** It is easy to see that Algorithm ITC is a special case of Algorithm 1 above.

**Definition 5** *We say that an assignment $\psi_t : A_t \to M$ is proper if Algorithm 1 ensures that $\{H_p\}_{p \in \mathcal{P}}$ is a UOWHF whenever $\{h_k\}_{k \in \mathcal{K}}$ is a UOWHF.*

The optimality question that we consider is the following. What is the minimum value of $|M|$ such that there is a proper assignment $\psi_t : A_t \to M$? Thus we are interested in obtaining a lower bound on the number of masks that must be used for the construction to be correct. Note that our construction in Section 4 shows that $t + \lfloor \log_2(t-1) \rfloor$ masks are sufficient for $\{H_p\}_{p \in \mathcal{P}}$ to be a UOWHF. Hence $t + \lfloor \log_2(t-1) \rfloor$ is an upper bound on the minimum value of $|M|$ for which there is a proper assignment $\psi_t$.

Let $S = (V, A)$ be a subtree of $T_t$ with $|A| \geq 1$. The subtree $S$ is not necessarily full and in the degenerate case can also be a path. For any binary tree $T$ (not necessarily full), we denote by

$\mathcal{L}(T)$ (resp. $\mathcal{I}(T)$) the set of leaf (resp. internal) nodes of $T$. For any subtree $S = (V, A)$ of $T_t$ and any assignment $\psi_t : A_t \to M$, define $\sigma_{\psi_t}(S) = \bigoplus_{a \in A} \psi_t(a)$, i.e., $\sigma_{\psi_t}(S)$ is the XOR of all the masks that occur in the subtree $S$ under the assignment $\psi_t$. If the assignment $\psi_t$ is clear from the context then we will use only $\sigma(S)$ instead of $\sigma_{\psi_t}(S)$. We will use the notation $0^m$ to denote the all-zero string of length $m$.

**Lemma 6** *Let $T_t = (V_t, A_t)$ be the full binary tree of $t$ levels. Let $\psi_t : A_t \to M$ be a proper assignment of masks. Then for any subtree $S = (V, A)$ of $T_t$ with $|A| \geq 1$, we must have $\sigma_{\psi}(S) \neq 0^m$.*

**Proof.** We show that if for any nonempty subtree $S$ we have $\sigma(S) = 0^m$, then it is possible to find collisions for $\{H_p\}_{p\in\mathcal{P}}$ even if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF. For the proof we must assume that a UOWHF exists, otherwise the result is vacuously true. Let $\mathcal{K} = \{0,1\}^K$, $m = m' + K$ and $n = 2m + 3$. Let $\{h'_k\}_{k\in\mathcal{K}}$ be a UOWHF, where $h' : \{0,1\}^n \to \{0,1\}^{m'}$. We define $\{h_k\}_{k\in\mathcal{K}}$ from $\{h'_k\}_{k\in\mathcal{K}}$ in the following manner. The function $h_k$ is a map from $\{0,1\}^n$ to $\{0,1\}^m$. Let $y$ be the input to $h_k$. We write $y = y_1||y_2||y_3||y_4||y_5$, where $|y_1| = |y_3| = m'$, $|y_2| = |y_4| = K$ and $y_5 = 3$. Let $flag(y, k)$ be a Boolean valued function which is true ($\mathsf{T}$) if and only if $(y_2 \oplus y_4 = k)$ or $(y_2 = k)$ or $(y_4 = k)$.

$$
\begin{array}{rcll}
h_k(y_1, y_2, y_3, y_4, y_5) & = & h'_k(y_1, y_2, y_3, y_4, y_5)||k & \text{if } y_5 = 000 \text{ and } flag(y,k) = \mathsf{F} \\
& = & h'_k(y_1, y_2, y_3, y_4, y_5)||0^K & \text{if } y_5 = 001 \text{ and } flag(y,k) = \mathsf{F} \\
& = & h'_k(y_1, y_2, y_3, y_4, y_5)||y_2 & \text{if } y_5 = 010 \text{ and } flag(y,k) = \mathsf{F} \\
& = & h'_k(y_1, y_2, y_3, y_4, y_5)||y_4 & \text{if } y_5 = 011 \text{ and } flag(y,k) = \mathsf{F} \\
& = & h'_k(y_1, y_2, y_3, y_4, y_5)||y_2 \oplus y_4 & \text{if } y_5 = 100 \text{ and } flag(y,k) = \mathsf{F} \\
& = & h'_k(y_1, y_2, y_3, y_4, y_5)||1^K & \text{if } y_5 = 101, 110, 111 \text{ and } flag(y,k) = \mathsf{F} \\
& = & 1^m & \text{if } flag(y,k) = \mathsf{T}.
\end{array}
$$

We first argue that $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF if $\{h'_k\}_{k\in\mathcal{K}}$ is a UOWHF. From the definition of $h_k$, it is clear that for the first six cases finding a collision for $h_k$ implies finding a collision for $h'_k$. The probability that the adversary is able to determine $y_2$ and/or $y_4$ such that the last case occurs before he knows $k$ is negligible. Hence $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF.

Suppose $S = (V, A)$ is a subtree of $T_t$ with $|A| \geq 1$ and $\sigma(S) = 0^m$. We show that it is possible to define two strings $x \neq x'$ such that $H_p(x) = H_p(x')$. The strings $x$ and $x'$ will be written as $x = x_1||\ldots||x_{2^t-1}$ and $x' = x'_1||\ldots||x'_{2^t-1}$, where

$$
\begin{array}{c}
|x_1| = \ldots = |x_{2^{t-1}-1}| = |x'_1| = \ldots = |x'_{2^{t-1}-1}| = n - 2m = 3 \\
|x_{2^{t-1}}| = \ldots = |x_{2^t-1}| = |x'_{2^{t-1}}| = \ldots = |x'_{2^t-1}| = n.
\end{array}
$$

Let $s$ be the root node of $S$ with $j = level(s)$. Let $S_1 = (V', A')$ be the full binary tree of $t - j + 1$ levels rooted at $s$. Then $S$ is a subtree of $S_1$ and $S_1$ is a subtree of $T_t$. In fact, if $s = 1$, then $S_1 = T_t$. The constructions of the strings $x$ and $x'$ are described next.

1. Set $x_i = x'_i$ for all $i \in (\{1, \ldots, 2^t - 1\} \setminus V') \cup (\mathcal{I}(S_1) \setminus \mathcal{I}(S))$.
2. Set $x_i = 0^{2m+3}$ and $x'_i = 1^{2m+3}$ for all $i \in \mathcal{L}(S_1) \setminus \mathcal{L}(S)$.
3. If $(\mathcal{L}(S) \cap \mathcal{L}(S_1) = \emptyset)$ then
   (a) Let $\mathcal{L}(S) = \{i_1, \ldots, i_r\}$.
   (b) Set $x_{i_1} = 000 = x'_{i_1}$.
   (c) Set $x_i = 001 = x'_i$ for each $i \in \mathcal{L}(S) \setminus \{i_1\}$.

4. If $(\mathcal{L}(S) \cap \mathcal{L}(S_1) = \{i_1, \ldots, i_r\} \neq \emptyset)$ then
   (a) Set $x_{i_1} = 0^{2m+3}$, $x'_{i_1} = 1^{2m}000$.
   (b) Set $x_i = 0^{2m+2}1 = x'_i$, for each $i \in \{i_2, \ldots, i_r\}$.
5. For each $i \in \mathcal{I}(S)$
   (a) If $(2i, i), (2i+1, i) \in A$, then $x_i = 100 = x'_i$.
   (b) If $(2i, i) \in A$ and $(2i+1, i) \notin A$, then $x_i = 010 = x'_i$.
   (c) If $(2i, i) \notin A$ and $(2i+1, i) \in A$, then $x_i = 011 = x'_i$.

We first note that by construction $x \neq x'$. To see this note that if $\mathcal{L}(S_1) \setminus \mathcal{L}(S) \neq \emptyset$, then Step 2 ensures $x \neq x'$ else Step 4(a) ensures $x \neq x'$.

We claim that $H_p(x) = H_p(x')$. Our claim consists of two parts.

1. The output of processor $P_s$ is $1^m$ for both $x$ and $x'$.
2. For any processor $P_i$ with $i \in \{1, \ldots, 2^t - 1\} \setminus V'$, the output of $P_i$ on both $x$ and $x'$ are equal.

From Step 1 in the construction of $x$ and $x'$ we know that $x_i = x'_i$ for each $i \in \{1, \ldots, 2^t - 1\} \setminus V'$. Thus if the first point is true then the second point is also true. We now turn to the proof of the first point.

Let $z_i = z_{i,1} \ldots z_{i,m}$ be the output of processor $P_i$ where each $z_{i,j} \in \{0, 1\}$. We will call the bits $z_{i,m'+1}, \ldots, z_{i,m'+K}$ the *critical bits* of $z_i$ and the positions $m'+1, \ldots, m'+K$ the *critical positions.* For an $m$-bit string $z$ we will denote by $z^{(c)}$ the substring of $z$ present in the critical positions of $z$. The idea of the proof is for processor $P_s$ to "see" the key $k$ in its input and output the string $1^m$. Exactly one processor in $\mathcal{L}(S)$ writes $k$ in the critical positions of its output. We will call this processor the special processor. All other processors in $\mathcal{L}(S)$ write $0^K$ in the critical positions of their outputs. Thus the key is masked exactly once and for all the other leaf processors $P_i$, the mask itself is provided as part of the input to the parent $P_{2i}$.

Each processor $P_i$ in $\mathcal{I}(S)$ behaves in the following fashion.

1. If $(2i, i) \in A$ and $(2i+1, i) \notin A$, then $y_2$ is copied to the critical positions of the output.
2. If $(2i, i) \notin A$ and $(2i+1, i) \in A$, then $y_4$ is copied to the critical positions of the output.
3. If both $(2i, i), (2i+1, i) \in A$, then $y_2 \oplus y_4$ is copied to the critical positions of the output.

Let $R_i$ be the subtree of $S$ rooted at processor $P_i$. The above procedure ensures that for any processor $j \in \mathcal{I}(S)$, if $R_j$ contains the special processor, then $R_j$ 'sees' $\sigma(R_j) \oplus k$ in its input else it 'sees' only $\sigma(R_j)$ in its input.

Let $(y_1, y_2, y_3, y_4, y_5)$ be the input to processor $P_s$ which is the root of the subtree $S$. Now there are three cases.

**Case 1 :** $((2s, s) \in A$ and $(2s+1, s) \in A)$ Let $\mu_1$ and $\mu_2$ be the masks for the arcs $(2s, s)$ and $(2s+1, s)$ respectively. In this case $\sigma(S) = \mu_1 \oplus \mu_2 \oplus \sigma(R_{2s}) \oplus \sigma(R_{2s+1})$. Exactly one of the trees $R_{2s}$ and $R_{2s+1}$ contain the special processor. Suppose $R_{2s}$ contains the special processor (the other case is similar). Then the string $y_2$ equals $\mu_1^{(c)} \oplus \sigma^{(c)}(R_{2s}) \oplus k$ and $y_4$ equals $\mu_2^{(c)} \oplus \sigma^{(c)}(R_{2s+1})$. Hence $y_2 \oplus y_4 = \mu_1^{(c)} \oplus \mu_2^{(c)} \oplus \sigma^{(c)}(R_{2s}) \oplus \sigma^{(c)}(R_{2s+1}) \oplus k = \sigma^{(c)}(S) \oplus k = k$, since by assumption $\sigma(S) = 0^m$ (and hence $\sigma^{(c)}(S) = 0^K$). Thus $P_s$ outputs $1^m$.

**Case 2 :** ($(2s, s) \in A$ and $(2s+1, s) \notin A$) Let $\mu$ be the mask for the arc $(2s, s)$. In this case $\sigma(S) = \mu \oplus \sigma(R_{2s})$. Clearly in this case $R_{2s}$ must contain the special processor. Hence $y_2 = \mu^{(c)} \oplus \sigma^{(c)}(R_{2s}) \oplus k = \sigma^{(c)}(S) \oplus k = k$, since by assumption $\sigma(S) = 0^m$.

**Case 3 :** ($(2s, s) \notin A$ and $(2s, s) \in A$) This case is similar to Case 2 and hence the details are omitted.

Thus we have proved $H_p(x) = H_p(x')$ and hence we are able to obtain collisions for $H_p$. Therefore $\{H_p\}_{p \in \mathcal{P}}$ is not a UOWHF even though $\{h_k\}_{k \in \mathcal{K}}$ is a UOWHF. Hence $\psi$ is not a proper assignment. This contradicts the hypothesis. ■

We use Lemma 6 to obtain a lower bound on $|M|$, the number of masks that must be required for a proper assignment $\psi$ to exist.

Let $M = \{\mu_1, \ldots, \mu_r\}$ be a set of masks and $\psi_t : A_t \to M$ be an assignment (not necessarily proper). Let $S$ be a subtree of $T_t$. For $\mu \in M$, define $num_\psi(S, \mu)$ to be the number of times the mask $\mu$ occurs in the tree $S$ under the assignment $\psi$. Define $vec_\psi(S) = (num_\psi(S, \mu_1) \bmod 2, \ldots, num_\psi(S, \mu_r) \bmod 2)$. We will use the notation $vec(S)$ when the assignment $\psi$ is clear from the context.

**Proposition 7** *Let $\psi_t : A_t \to M$ be an assignment and $S$ a subtree of $T_t$. Then $\sigma(S) = 0^m$ if and only if $vec_\psi(S) = (0, \ldots, 0)$. Consequently, if $\psi$ is proper, then for any subtree $S$ with at least one arc, we have $vec_\psi(S) \neq (0, \ldots, 0)$.*

For any two subtrees $S = (V, A)$ and $S' = (V', A')$ of $T_t$ define $S \Delta S'$ to be the forest induced by the set $V \Delta V'$. Also we call a subtree of $T_t$ to be *nontrivial* if it contains at least one arc.

**Definition 8** *Let $\mathcal{F}$ be a family of subtrees of $T_t$ such that for any two distinct $S_1, S_2 \in \mathcal{F}$, we have $S_1 \Delta S_2$ to be a nontrivial subtree of $T_t$. We will call $\mathcal{F}$ to be a connected family.*

**Lemma 9** *Suppose Algorithm 1 uses $M$ to be the set of masks on the tree $T_t = (V_t, A_t)$ with a proper assignment $\psi_t : A_t \to M$. Let $\mathcal{F}$ be a connected family of subtrees of $T_t$. Then the following hold.*

1. *For any $S \in \mathcal{F}$, we have $vec_\psi(S) \neq (0, \ldots, 0)$.*
2. *For any two distinct $S_1, S_2 \in \mathcal{F}$, we have $vec_\psi(S_1) \neq vec_\psi(S_2)$.*

**Proof.** The first point is immediate from Proposition 7. We prove the second point. Suppose there are two distinct $S_1, S_2$ such that $vec(S_1) = vec(S_2)$. It is easy to see that $vec(S_1 \Delta S_2) = vec(S_1) \oplus vec(S_2) = (0, \ldots, 0)$. Hence from Proposition 7, we have $\sigma(S_1 \Delta S_2) = 0^m$. Since the assignment $\psi$ is proper and $S_1 \Delta S_2$ is a subtree of $T_t$ containing at least one arc, we obtain a contradiction to Lemma 6. ■

A direct consequence of Lemma 9 is the following result.

**Lemma 10** *Let $\delta$ be the maximum cardinality of a connected family $\mathcal{F}$ of subtrees of $T_t = (V_t, A_t)$. Then for any proper assignment $\psi_t : T_t \to M$, we must have $\delta \leq 2^{|M|} - 1$ (or equivalently, $|M| \geq \lceil \log_2(a+1) \rceil$).*

Lemma 10 reduces the problem of finding lower bound on $|M|$ to a combinatorial question about the full binary tree $T_t$.

**Theorem 11** *Suppose Algorithm 1 uses the full binary tree $T_t = (V_t, A_t)$, a set of masks $M$ and a proper assignment $\psi_t : A_t \rightarrow M$. Then $|M| \geq 2$ for $t = 2$ and $|M| \geq t + 1$ for $t \geq 3$.*

**Proof.** We provide a recursive construction of a connected family $\mathcal{F}_t$.

$\mathcal{F}_2 = \{S_1, S_2, S_3\}$, where $S_1$ consists of the single arc $(2, 1)$, $S_2$ consists of the single arc $(3, 1)$ and $S_3 = T_2$. Clearly, $\mathcal{F}_2$ is a connected family.

For $t > 2$, the construction of $\mathcal{F}_t$ is the following. Let $S_1$ and $S_2$ be the full binary trees rooted at nodes 2 and 3. Then $S_1$ and $S_2$ are isomorphic copies of $T_{t-1}$. Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be the isomorphic copies of $\mathcal{F}_{t-1}$ corresponding to the trees $S_1$ and $S_2$ respectively. Let $\mathcal{G}_1'$ be the family obtained from $\mathcal{G}_1$ by adding the arc $(2, 1)$ to each subtree in $\mathcal{G}_1$. Similarly let $\mathcal{G}_2'$ be the family obtained from $\mathcal{G}_2$ by adding the arc $(3, 1)$ to each tree in $\mathcal{G}_2$. Define $\mathcal{F}_t = \mathcal{G}_1' \cup \mathcal{G}_2' \cup \{S_1, S_2\}$. Then it is not difficult to verify that $\mathcal{F}_t$ is a connected family.

Let $N_t = |\mathcal{F}_t|$. By the construction above we have $N_2 = 3$ and for $t \geq 3$, $N_2 = 2N_{t-1} + 2$. Hence $N_t = 5.2^{t-2} - 2 \geq 2^t$ for $t \geq 3$. Hence by Lemma 10 we have $|M| \geq 2$ for $t = 2$ and $|M| \geq t + 1$ for $t \geq 3$. ■

We have already shown that $t + \lfloor \log(t - 1) \rfloor$ masks are sufficient for a proper assignment $\psi_t$ to exist. Combined with Theorem 11 we get the following result.

**Corollary 12** *The key length expansion made by Algorithm ITC is the minimum possible for $2 \leq t \leq 4$ and is at most $m$ bits more than the optimal for $5 \leq t \leq 8$.*

We note that the Shoup construction requires $t$ masks and it has been proved by Mironov [4] that one cannot use less number of masks. For the binary tree algorithm, Theorem 11 shows that at least $(t+1)$ masks are required. Thus in moving from sequential to parallel algorithm, the trade-off is going to be an increase by one in the number of masks required.

## 6 Conclusion

In this paper we have considered the problem of extending the domain of a UOWHF using a binary tree algorithm. As shown in [1] this requires an expansion in the length of the key to the hash function. Our algorithm makes a key length expansion of $2m$ bits for $t = 2$; $m(t + 1)$ bits for $3 \leq t \leq 6$ and $m(t + \lfloor \log_2(t - 1) \rfloor)$ for $t \geq 7$ using a binary tree of $t$ levels and a base hash function $h_k : \{0, 1\}^n \rightarrow \{0, 1\}^m$. The previous algorithm in [1] required a key length expansion of $2m(t - 1)$ with the same parameters. Hence the key length expansion in our algorithm is significantly lesser.

We prove that any proper extension of a UOWHF $\{h_k\}_{k \in \mathcal{K}}$, with $h_k : \{0, 1\}^n \rightarrow \{0, 1\}^m$ using a binary tree of $t$ levels (and $2^t - 1$ processors) must make a key length expansion of $2m$ bits for $t = 2$ and at least $m(t+1)$ bits for $t \geq 3$. This shows that with respect to key expansion our binary tree based algorithm is optimal for $2 \leq t \leq 6$ and is nearly optimal for $t = 7, 8$. For $t \geq 7$, it is an open problem to try and close the gap between the lower bound $(t + 1)$ and the upper bound $(t + \lfloor \log_2(t - 1) \rfloor)$ on the minimum number of masks required for a proper extension of a UOWHF.

## References

[1] M. Bellare and P. Rogaway. Collision-resistant hashing: towards making UOWHFs practical. *Proceedings of CRYPTO 1997*, pp 470-484.

[2] I. B. Damgård. A design principle for hash functions. *Lecture Notes in Computer Science*, 435 (1990), 416-427 (Advances in Cryptology - CRYPTO'89).

[3] R. C. Merkle. One way hash functions and DES. *Lecture Notes in Computer Science*, 435 (1990), 428-226 (Advances in Cryptology - CRYPTO'89).

[4] I. Mironov. Hash functions: from Merkle-Damgård to Shoup. *Lecture Notes in Computer Science*, 2045 (2001), 166-181 (Advances in Cryptology - EUROCRYPT'01).

[5] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic aplications. *Proceedings of the 21st Annual Symposium on Theory of Computing*, ACM, 1989, pp. 33-43.

[6] B. Preneel. The state of cryptographic hash functions. *Lecture Notes in Computer Science*, 1561 (1999), 158-182 (Lectures on Data Security: Modern Cryptology in Theory and Practice).

[7] V. Shoup. A composition theorem for universal one-way hash functions. *Proceedings of Eurocrypt 2000*, pp 445-452, 2000.

[8] D. Simon. Finding collisions on a one-way street: Can secure hash function be based on general assumptions?, *Lecture Notes in Computer Science - EUROCRYPT'98*, pp 334-345, 1998.

[9] D. R. Stinson. Some observations on the theory of cryptographic hash functions. IACR preprint server, http://eprint.iacr.org/2001/020/.