Construction of UOWHF: Tree Hashing Revisited

Palash Sarkar
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.

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.

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.

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 an 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.

2 Preliminaries

Let $\{h_k\}_{k \in \mathcal{K}}$ be a keyed family of hash functions, where each $h_k : \{0,1\}^n \to \{0,1\}^m$. *In this paper we require* $n \ge 2m$. 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 (ϵ, a)-strategy if the success probability of $\mathcal{A}$ is at least ϵ and it invokes the hash function h_k at most a times. In this case we say that the adversary has an (ϵ, 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 $h_k : \{0,1\}^n \to \{0,1\}^m$ and a positive integer $L \ge n$, we would like to construct another UOWHF $\{H_p\}_{p \in \mathcal{P}}$, with $H_p : \{0,1\}^L \to \{0,1\}^m$. We say that the adversary has an (ϵ, 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 ϵ

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 (ϵ, a)-strategy for $\{H_p\}_{p\in\mathcal{P}}$, then there is an (ϵ_1, a_1)-strategy for $\{h_k\}_{k\in\mathcal{K}}$, where a_1 is not much larger than a and ϵ_1 is not significantly lesser than ϵ. 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 sequential construction by Shoup [7] and the tree construction by Bellare-Rogaway [1].

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. Define $\nu(i) = j$ if $2^j | i$ and $2^{j+1} \not| i$.

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 $z_0 = \mathrm{IV}$, $s_0 = z_0 \oplus m_0$.
4. For $1 \le i \le r$, define $z_i = h_k(s_{i-1}||x_i)$ and $s_i = z_i \oplus m_{\nu(i)}$.
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 $n \geq 2m$. 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 α_i and β_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 $2^{t-1} \leq i \leq 2^t - 1$, 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 $2^{j-1} \leq i \leq 2^j - 1$. 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* $t \geq 2$.

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 α_i and β_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 $2^{t-1} \leq i \leq 2^t - 1$, 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 $P = P_{i_r}P_{i_{r-1}} \dots P_{i_1}$ 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 $j = 1, \dots, r-1$. 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 $\{(2i_1+1, i_1), (2i_2+1, i_2), \dots, (2i_{r-1}+1, i_{r-1})\}$. 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 an improvement over the BR construction.

Theorem 2 (Security Reduction for H_p) *If there is an (ϵ, 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.

$\mathcal{B}^{\mathbf{guess}}$ **:** (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 $\{1, \dots, 2^t - 1\}$.
3. Write $x = x_1 || \dots || x_{2^t-1}$, where $|x_1| = \dots = |x_{2^{t-1}-1}| = n - 2m$ and $|x_{2^{t-1}}| = \dots = |x_{2^t-1}| = n$.
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.

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

1. Define the masks $\alpha_1, \dots, \alpha_{t-1}, \beta_0, \dots, \beta_{l-1}$ 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 $p = k || \alpha_1 || \dots || \alpha_{t-1} || \beta_0 || \dots || \beta_{l-1}$, 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 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 α_i and β_j are chosen independently and uniformly to be m-bit strings.

The mask defining algorithm 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 $1 \leq j \leq 2^t - 1$ 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 $2^{t-1} \leq i \leq 2^t - 1$, 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 $j = t - level(i)$, $j_1 = j - 2^{\nu(j)}$ 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 β_i, $0 \leq i \leq l - 1$.
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 α_j, $1 \leq j \leq t - 1$.

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 β masks are complete, the algorithm begins the task of defining the α 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 α_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 α_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 α_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 ϵ, 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 Γ and Δ. 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 3 *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 2. A close examination of the proof of Theorem 2 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 $i \ge 4$, then the mask defining algorithm of Theorem 2 is sufficient to define all the masks. This is because of the fact that in Algorithm ITC the mask β_2 does not occur in the subtree rooted at i and hence we are required to define only α_1. The problem arises when we have to define both α_1 and β_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. ■

5 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 algorithm of Bellare and Rogaway [1] required a key length expansion of $2m(t-1)$ with the same parameters. Hence the key length expansion in our algorithm is lesser.

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/.