Domain Extenders for UOWHF: A Generic Lower Bound on Key Expansion and a Finite Binary Tree Algorithm

Palash Sarkar
Cryptology Research Group
Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road
Kolkata 700108, India
e-mail: palash@isical.ac.in

Abstract

We obtain a generic lower bound on the key expansion required for securely extending the domain of a UOWHF. Our lower bound holds over a large class of "natural" domain extending algorithms. A consequence of our result is the fact that the key length expansion in Shoup's algorithm is optimal for this class. Our second contribution is to obtain a *finite* binary tree algorithm to extend the domain of a UOWHF. The associated key length expansion is only a constant number of bits more than the minimum possible. Our finite binary tree algorithm is the first practical parallel algorithm to securely extend the domain of a UOWHF. Also the speed-up obtained by our algorithm is approximately proportional to the number of processors.

Keywords : UOWHF, hash function, lower bound, binary tree.

1 Introduction

Universal one-way hash function (UOWHF) was introduced by Naor and Yung [5] to prove that secure digital signatures can be based on one-way, 1-1 functions. 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, is then given a random $k \in \mathcal{K}$ and subsequently has to find a y such that $x \neq y$ but $h_k(x) = h_k(y)$. Intutively, a UOWHF is a weaker primitive than a collision-resistant function, 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. Simon [10] has shown that there is a oracle relative to which UOWHFs exist but collision resistant hash functions do not exist. See [6] for a survey on hash functions and [11] for some properties and reductions between different kinds of hash functions.

The study of UOWHF was later undertaken by several authors. Bellare and Rogaway [1] showed that it is possible to build practical and provably secure "hash-then-sign" schemes, where the hashing is done using a UOWHF. The paper also addresses the problem of constructing UOWHFs. Like most basic cryptographic primitives it is virtually impossible to define a family $\{h_k\}_{k\in\mathcal{K}}$ and prove it to be a

UOWHF. The idea suggested in [1] is to use one of the standard hash functions like SHA or RIPEMD in a keyed mode and assume it to be a UOWHF. It seems more reasonable to make this assumption when the domain is a short string rather than an arbitrarily long string. This leads to the question of extending the domain of a UOWHF while preserving the UOWHF property.

The Merkle-Damgård algorithm [2, 3] is a well known method of extending the domain of a collision-resistant hash function. However, as shown in [1] this method does not work in the case of a UOWHF. Several constructions for extending the domain of a UOWHF is presented in [1]. These constructions assume the existence of a UOWHF $\{h_k\}_{k\in\mathcal{K}}$, with $h_k : \{0,1\}^n \to \{0,1\}^m$ and show how to construct a UOWHF $\{H_p\}_{p\in\mathcal{P}}$ where the input to H_p can be a very long message. The constructions have an associated key length expansion of $|p| - |k|$. One of the major goals in extending the domain of a UOWHF is to *minimise* the expansion of the key length.

Shoup [9] provides a modification of the Merkle-Damgård algorithm to extend the domain of a UOWHF. The Merkle-Damgård construction and hence the Shoup construction is a sequential algorithm. A tree based scheme for extending the domain of a UOWHF was presented in [1]. For binary trees the scheme can be used to hash a message of length $L = 2^T(n-m) - (n-2m)$ using a full binary of height T and $2^T - 1$ processors, where the base UOWHF takes as input a message of length n and produces a digest of length m. The key length expansion made by the algorithm is $2m(T-1)$. In a recent work [7], an improved binary tree based construction has been presented which makes a key length expansion of $m(T + \lfloor \log(T-1) \rfloor)$ bits for $T \geq 2$. Further, it has been shown in [7] that a key length expansion of $m(T+1)$ bits is necessary for any full binary tree based algorithm to be correct. The main disadvantage of both the above binary tree algorithms is that the number of processors grows with the length of the message. See Table 1 in Section 7 for a comparison of different algorithms.

An important factor in the study of domain extender for UOWHFs is the minimum amount of key length expansion that is required for the domain extender to be correct. Mironov [4] has shown that the key length expansion in Shoup's algorithm is the minimum possible for any sequential Merkle-Damgård type construction. Sarkar [7] has obtained a lower bound on the amount of key length expansion required for any full binary tree algorithm. This brings up the question of obtaining a lower bound on the key expansion required by *any* domain extending algorithm for a UOWHF. We provide a partial answer to this question by obtaining a lower bound on the key expansion that must be made by a large class of algorithms $\boldsymbol{\mathcal{A}}$ (see Section 4). The sequential algorithm of Shoup [9] and the binary tree algorithms of [1] and [7] belong to this class $\boldsymbol{\mathcal{A}}$.

Let $\{h_k\}_{k\in\mathcal{K}}$ be a UOWHF, where $h_k : \{0,1\}^n \to \{0,1\}^m$. We show that any domain extender in the class $\boldsymbol{\mathcal{A}}$ which invokes $h_k()$ exactly s times must make a key length expansion of $m \times \lceil \log_2 s \rceil$ bits. A consequence of this result is that Shoup's algorithm is optimal for the class $\boldsymbol{\mathcal{A}}$. Also both the algorithms [1] and [7] are suboptimal, although [7] is better than that of [1]. Further, the lower bound obtained in [7] shows that any full binary tree algorithm is necessarily suboptimal with respect to key length expansion.

Our next task is to obtain a binary tree algorithm for which the key length expansion is a constant number of bits more than the minimum possible. This is made possible by using a *finite* binary tree of processors in contrast to [1, 7] where the height of the binary tree increases with the length of the message. Thus our algorithm yields the first *practical* parallel algorithm for securely extending the domain of a UOWHF. Another important consequence of using a finite binary tree is the fact that for moderately long messages the speed-up over sequential (Shoup's) algorithm is equal to the number of processors. Thus our algorithm makes efficient use of resources. We note that the speed-up obtained

in [1, 7] is logarithmic in the number of processors.

The finite binary tree algorithm is built using ideas from several existing work. The basic algorithm has been used to extend the domain of a collision resistant function in [8]. To this algorithm we add the masking techniques of [9] and [7]. The masking technique of [7] is itself built around the masking techniques of [9] and that of [1]. We show that all these techniques fit together nicely to provide a correct domain extender for UOWHFs.

2 Preliminaries

Let $\{h_k\}_{k\in\mathcal{K}}$ be a keyed family of hash functions, where each $h_k : \{0,1\}^n \to \{0,1\}^m$. 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 y such that $y \neq x$ and $h_k(x) = h_k(y)$.

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 σ. In the second stage $\mathcal{A}^{\text{find}}(x,k,\sigma)$, the adversary either finds a $y \neq x$ such that $h_k(x) = h_k(y)$ or it reports failure. Both $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x,k,\sigma)$ are probabilistic algorithms. The success probability of the strategy is measured over the random choices made by $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x,k,\sigma)$ 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 $\{h_k\}_{k\in\mathcal{K}}$. 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.

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, we would like to construct another UOWHF $\{H_p\}_{p\in\mathcal{P}}$, with $H_p : \{0,1\}^L \to \{0,1\}^m$. We also consider the following situation. Given a UOWHF $\{h_k\}_{k\in\mathcal{K}}$ where $h_k : \{0,1\}^n \to \{0,1\}^m$, we construct a UOWHF $\{H_p^*\}_{p\in\mathcal{P}}$ where $H_p^* : \cup_{i=1}^{2^{n-m}}\{0,1\}^i \to \{0,1\}^m$. Since we require $n \geq 2m$, we have $n - m \geq m$. For practical applications m is at least 64 bits, hence messages upto length 2^{64} can be hashed by H_p^*. This is sufficient for any conceivable purpose. Due to lack of space we present the construction and proof of security of $\{H_p^*\}_{p\in\mathcal{P}}$ in the Appendix.

We say that the adversary has an (ϵ, a)-extended 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)-extended 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 Algorithms

We briefly discuss the domain extending algorithms for UOWHFs which have already been proposed.

3.1 Sequential Algorithm

The Merkle-Damgård construction [3, 2] is a well known construction for extending the domain of a collision resistant hash function. However, Bellare and Rogaway [1] showed that the construction does not directly work in the case of UOWHF. In [9], Shoup presented a modification of the MD construction. We briefly describe the Shoup construction.

Let $\{h_k\}_{k \in \mathcal{K}}$, $h_k : \{0,1\}^n \to \{0,1\}^m$, $\mathcal{K} = \{0,1\}^K$ be the UOWHF whose domain is to be extended. Let x be the input to H_p with $|x| = n + r(n-m)$. We define $p = k||\mu_0||\mu_1||\ldots||\mu_{l-1}$ where $l = 1 + \lfloor \log r \rfloor$ and μ_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. For integer i, define $\nu(i) = j$ if $2^j | i$ and $2^{j+1} \not| i$.

Algorithm SeqUOWHF

1. Let $x = x_0||x_1||x_2||\ldots||x_r$, where $|x_0| = n$ and $|x_i| = n - m$ for $1 \le i \le r$.
2. Define $z_0 = h_k(x_0)$.
3. For $1 \le i \le r$, define $s_i = z_{i-1} \oplus \mu_j$ and $z_i = h_k(s_i||x_i)$ where $j = \nu(i)$.
4. Define z_r to be the output of $H_p(x)$.

For the sake of simplicity we do not include an initialisation vector. The function h_k is invoked $(r+1)$ times and the algorithm requires $\lceil \log_2(r+1) \rceil = 1 + \lfloor \log_2 r \rfloor$ masks. This algorithm was initially described in [9] and in [4] it was shown that the number of masks required is the minimum possible for any such sequential construction to be correct.

3.2 Tree Based Algorithm

Extending the domain of a UOWHF using a full binary tree of processors have been considered in the literature [1, 7]. A full binary tree of $2^T - 1$ processors numbered $P_1, \ldots, P_{2^T-1}$ is used. The length of the message x to be hashed is $|x| = L = 2^{T-1}n + (2^{T-1} - 1)(n - 2m)$. Let $\mathcal{T}_T = (V_T, A_T)$ be the full binary tree of $2^T - 1$ processors, where $V_T = \{1, \ldots, 2^T - 1\}$ and $A_T = \{(i, \lfloor (i/2) \rfloor) : 1 < i < 2^T\}$. We set $a_i = (i, \lfloor (i/2) \rfloor)$ and so $A_T = \{a_2, \ldots, a_{2^T-1}\}$. There is a set M of m-bit masks and a function $\psi : A_T \to M$, which assigns an m-bit string to each arc of $\mathcal{T}_T$. The algorithms of [1] and [7] have the same general form and differ only in the description of M and ψ. We first describe the general form of the algorithm.

Algorithm TreeUOWHF

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 $i = 2^{T-1}, \ldots, 2^T - 1$, do in parallel
   $z_i = h_k(x_i)$.
   $s_i = z_i \oplus \psi(a_i)$.

3. For $j = T - 1$ downto 2 do
   For $i = 2^{j-1}$ to $2^j - 1$ do in parallel
      $z_i = h_k(s_{2i}||s_{2i+1}||x_i)$.
      $s_i = z_i \oplus \psi(a_i)$.
4. Output $h_k(s_2||s_3||x_1)$ as the output of $H_p(x)$.

To complete the description of Algorithm 1 we have to define M and ψ. We do this separately for [1] and [7].

Bellare and Rogaway [1] : In this case $M = \{\alpha_1, \ldots, \alpha_{T-1}, \beta_1, \ldots, \beta_{T-1}\}$ and $\psi(a_i)$ is defined as follows: $\psi(a_i) = \alpha_{T+1-l}$ if $i \equiv 0 \bmod 2$; and $\psi(a_i) = \beta_{T+1-l}$ if $i \equiv 1 \bmod 2$. Here $l = level(i)$, i.e., $2^{l-1} \leq i \leq 2^l - 1$.

Sarkar [7] : In this case $M = \{\alpha_1, \ldots, \alpha_{T-1}, \beta_0, \ldots, \beta_{r-1}\}$ where $r = 1 + \lfloor \log_2(T-1) \rfloor$ and $\psi(a_i)$ is defined as follows: $\psi(a_i) = \beta_{\nu(T+1-l)}$ if $i \equiv 0 \bmod 2$; and $\psi(a_i) = \alpha_{T+1-l}$ if $i \equiv 1 \bmod 2$. Here again $l = level(i)$.

Note that the Bellare-Rogaway algorithm requires $2(T-1)$ masks whereas Sarkar's algorithm requires $T + \lfloor \log_2(T-1) \rfloor$ masks.

4 Generic Lower Bound on Key Length Expansion

We are interested in the class of algorithms $\boldsymbol{\mathcal{A}}$ which can securely extend the domain of a UOWHF $\{h_k\}_{k \in \mathcal{K}}$, where $h_k : \{0,1\}^n \to \{0,1\}^m$. We do not want the algorithm to be dependent on the structure of the UOWHF; in fact it should work for all UOWHF's which can "fit" into the structure of the algorithm. Any algorithm $A \in \boldsymbol{\mathcal{A}}$ behaves in the following manner.

1. It invokes $h_k()$ a finite number times.
2. The outputs of all but one invocation of $h_k()$ is masked by XORing with an m-bit string selected from the set $\{\mu_0, \ldots, \mu_{\rho-1}\}$.
3. The invocations of $h_k()$ whose outputs are masked are called *intermediate* invocations and the invocation whose output is not masked is called the *final* invocation.
4. The entire output of any intermediate invocation is fed into the input of exactly one separate invocation of $h_k()$.
5. Each bit of the message x is fed into exactly one invocation of $h_k()$.
6. The output of the final invocation is the output of A.

Proposition 1 *Let $A \in \boldsymbol{\mathcal{A}}$ be such that A invokes $h_k()$ a total of s times. Then the length of the message which is hashed is equal to $n + (s-1)(n-m)$.*

Proof. Each invocation of $h_k()$ requires n bits and hence the s invocations of $h_k()$ require a total of $s \times n$ bits. The outputs of exactly $(s-1)$ invocations of $h_k()$ are used as inputs to other invocations of $h_k()$. This accounts for $(s-1) \times m$ of the $n \times s$ bits. The other $ns - (s-1)m = n + (n-m)(s-1)$ bits are obtained from the message. Since each bit of the message is used exactly once, the message must be of length $n + (n-m)(s-1)$. ■

Thus the number of invocations of $h_k()$ and the parameters n and m determine the length of the message to be hashed irrespective of the actual structure of the algorithm. Hence any algorithm

$A \in \boldsymbol{\mathcal{A}}$ which invokes $h_k()$ a total of s times defines a family $\{H_p^{(A,s)}\}_{p\in\mathcal{P}}$, where $\mathcal{P} = \{0,1\}^{|k|+m\rho}$ and $H_p^{(A,s)} : \{0,1\}^{n+(n-m)(s-1)} \to \{0,1\}^m$. The structure of any algorithm $A \in \boldsymbol{\mathcal{A}}$ which makes s invocations of $h_k()$ is described by a labelled directed graph $D_s = (V_s, E_s, \psi_s)$, where

1. $V_s = \{v_1, \ldots, v_s\}$, i.e., there is a node for each invocation of $h_k()$.
2. $(v_i, v_j) \in E_s$ if and only if the output of the ith invocation is fed into the input of the jth invocation.
3. ψ_s is a map $\psi : E_s \to \{\mu_0, \ldots, \mu_{\rho-1}\}$, where $\psi(v_{i_1}, v_{i_2}) = \mu_j$ if the output of the i_1-th invocation of $h_k()$ is masked using μ_j.

The nodes corresponding to the intermediate invocations are called intermediate nodes and the node corresponding to the final node is called the final node. Without loss of generality we assume the final node to be v_s. Nodes with indegree zero are called leaf nodes and the others are called internal nodes. Define $\delta(D_s) = \max\{\mathsf{indeg}(v) : v \in V_s\}$. We call $\delta(D_s)$ to be the fan-in of algorithm A for s invocations.

Proposition 2 *The outdegree of any intermediate node in D_s is 1 and the outdegree of the final node is 0. Hence there are exactly $(s-1)$ arcs in D_s. Consequently, D_s is a rooted directed tree where the final node is the root of D_s.*

Proposition 3 *If $\delta = \delta(D_s)$, then $n \geq \delta m$.*

Proof. Let v be a node with in-degree δ. Then v gets inputs from δ other nodes. Each of these nodes provide m bits. Since the input to v is exactly n bits we must have $n \geq m\delta$. ■

Thus an algorithm A with fan-in δ cannot be used with all UOWHFs. The value of fan-in places a restriction on the values of n and m. However, given this restriction the actual structure of D_s does not depend on the particular UOWHF chosen.

Definition 4 *We say that an algorithm $A \in \boldsymbol{\mathcal{A}}$ is correct for s invocations if there is a set of masks $\{\mu_0, \ldots, \mu_{\rho-1}\}$ and an assignment $\psi_s : E_s \to \{\mu_0, \ldots, \mu_{\rho-1}\}$ such that $\{H_p^{(A,s)}\}_{p\in\mathcal{P}}$ is a UOWHF whenever $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF.*

The question that we are interested in is the following: Given s, what is the minimum value of ρ such that there is an algorithm $A \in \boldsymbol{\mathcal{A}}$, a set of masks $\{\mu_0, \ldots, \mu_{\rho-1}\}$ and an assignment $\psi_s : E_s \to \{\mu_0, \ldots, \mu_{\rho-1}\}$ such that A is correct for s invocations?

The next result is crucial to obtaining an answer to this question. A subtree T of D_s is called non-trivial if it contains at least one arc. Let T be a non-trivial subtree of D_s. Denote by $\mathsf{vec}_\psi(T)$ the ρ-tuple $(\mathsf{num}_{\mu_0}(T) \bmod 2, \ldots, \mathsf{num}_{\mu_{\rho-1}}(T) \bmod 2)$, where $\mathsf{num}_{\mu_i}(T)$ is the number of times the mask μ_i occurs in the tree T.

Lemma 5 *Let $s > 0$ and $A \in \boldsymbol{\mathcal{A}}$ be correct for s invocations. Let T be a non-trivial subtree of D_s. Then $\mathsf{vec}_\psi(T) \neq (0, \ldots, 0)$.*

Proof. Let δ be the fan-in of A for s invocations. Then any UOWHF $\{h_k\}_{k\in\mathcal{K}}$, where $h_k : \{0,1\}^n \to \{0,1\}^m$ that can be used with A must satisfy $n \geq \delta m$. Also since A is correct for s invocations, we must have that $\{H_p^{(A,s)}\}_{p\in\mathcal{P}}$ is a UOWHF whenever $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF. Following [1, 4], for the proof we assume the existence of a UOWHF with appropriate parameters as otherwise the question itself is

moot. Let $\{h'_k\}_{k\in\mathcal{K}}$ be a UOWHF such that $h'_k : \{0,1\}^n \to \{0,1\}^{m'}$, $\mathcal{K} = \{0,1\}^K$, $m = m' + K$ and $n = \delta m + \delta + 1$. Let $z \in \{0,1\}^n$ and write

$$z = z_{1,1}||z_{1,2}||z_{2,1}||z_{2,2}||\ldots||z_{i,1}||z_{i,2}||\ldots||z_{\delta,1}||z_{\delta,2}||y||b$$

where $|z_{i,1}| = m$, $|z_{i,2}| = K$ for $1 \le i \le \delta$, $|y| = \delta$ and $b \in \{0,1\}$. We write $y = y(z)$ and $b = b(z)$ to show the dependence of y and b on z.

Given $z \in \{0,1\}^n$, define $\mathsf{KLst} = \{z_{1,2}, z_{2,2}, z_{3,2}, \ldots, z_{\delta,2}\}$. Given $z \in \{0,1\}^n$ and $k \in \mathcal{K}$, define a Boolean function $\phi(z,k)$ to be true (T) if and only if $k = \bigoplus_{w\in S} w$ for some $\emptyset \neq S \subseteq \mathsf{KLst}(z)$. We define a family of functions $\{h_k\}_{k\in\mathcal{K}}$, $h_k : \{0,1\}^n \to \{0,1\}^m$ in the following manner.

$$\left.\begin{array}{rcll} h_k(z) & = & h'_k(z)||k & \text{if } b = 1 \text{ and } \phi(z,k) = \mathsf{F}; \\ & = & h'_k(z)||0^K & \text{if } b = 0, y = 0^\delta \text{ and } \phi(z,k) = \mathsf{F}; \\ & = & h'_k(z)||S_y & \text{if } b = 0, y \neq 0^\delta \text{ and } \phi(z,k) = \mathsf{F}; \\ & = & 1^m & \phi(z,k) = \mathsf{T}. \end{array}\right\} \quad (1)$$

Here $y = y(z)$ and $S_y = \oplus_{y_i=1} z_{i,2}$, i.e., the XOR's of the $z_{i,2}$'s for which the ith bit of y is 1.

We first argue that $\{h_k\}_{k\in\mathcal{K}}$ defined above is a UOWHF. To see this first note that if $\phi(z,k) = \mathsf{F}$, then finding a collision for $h_k()$ amounts to finding a collision for $h'_k()$. The probability that the adversary is able to guess z such that $\phi(z,k) = \mathsf{T}$ before knowing k is very small. We use algorithm A to extend the domain of $\{h_k\}_{k\in\mathcal{K}}$ to get a family $\{H_p^{(A,s)}\}_{p\in\mathcal{P}}$.

Now suppose that there is a subtree $T = (V,E)$ of D_s such that $\mathsf{vec}_\psi(T) = (0,\ldots,0)$. Then $\psi(T) = \bigoplus_{e\in E} \psi(e) = 0^m$. We show that under this condition it is possible to find a collision for $\{H_p^{(A,s)}\}_{p\in\mathcal{P}}$.

Let $\mathcal{L}(T)$ (resp. $\mathcal{I}(T)$) be the set of all leaf (resp. internal) nodes of the tree T. We identify one particular node in $\mathcal{L}(T)$ and denote it by v_θ. Let $\mathcal{L}'(T) = \mathcal{L}(T) \setminus \{v_\theta\}$. For each node we denote by $\mathsf{indeg}_T(v)$ (resp. $\mathsf{indeg}_{D_s}(v)$) the indegree of v in T (resp. D_s).

Suppose $\mathsf{indeg}_{D_s}(v) = t \le \delta$ and assume an ordering $(v_{j_1}, \ldots, v_{j_t})$ of the input neighbours of v. We extend this t-tuple to the δ-tuple $\mathsf{Nbr}_{D_s}(v) = (v_{j_1}, \ldots, v_{j_t}, 0, \ldots, 0)$. If $v \in V$, we define $\mathsf{Nbr}_T(v)$ as follows. For $1 \le i \le \delta$, $\mathsf{Nbr}_T(v)[i] = \mathsf{Nbr}_{D_s}(v)[i]$ if $\mathsf{Nbr}_{D_s}(v)[i] \in V$; and $\mathsf{Nbr}_T(v)[i] = 0$ otherwise. Also we define a δ-bit string $\mathsf{str}_T(v)$ as follows: $\mathsf{str}_T(v) = 1$ if $\mathsf{Nbr}_T(v)[i] > 0$; else $\mathsf{str}_T(v) = 0$.

Suppose $\mathsf{indeg}_{D_s}(v) = t \le \delta$. Then v receives inputs from t other nodes. This accounts for $m \times t$ of the n bits required for the input of v. The other $(n - mt)$ bits are obtained from the message x. Thus to each node v of D_s we associate a substring of the message x. This substring is denoted by $\chi(v,x)$. Further, specifying $\chi(v,s)$ for each v in D_s completely specifies the message x. We now define two messages x and x' of lengths $n + (s-1)(n-m)$ each. We do this by specifying $\chi(v,x)$ and $\chi(v,x')$ for each v in D_s.

Given a message x and a node v of D_s, we identify four components of the n-bit input to the node v.

1. $f(v,x)$ is the portion of the input n bits that is obtained from other nodes.
2. $y(v,x)$ is the y-portion of the input n bits to the node v (obtained from x).
3. $b(v,x)$ is the b-portion of the input n bits to the node v (obtained from x).
4. $g(v,x)$ is the rest of the input n bits (obtained from x).

Note that $|y(v,x)| = \delta$, $b(v,x) \in \{0,1\}$, $|f(v,x)| = mt$ and $|g(v,x)| = n - mt - \delta - 1 = (\delta - t)m$, where $t = \mathsf{indeg}_{D_s}(v)$. To specify $\chi(v,x)$ it is sufficient to specify $y(v,x)$, $b(v,x)$ and $g(v,x)$. We now specify the strings x and x'. Let w be an arbitrary $m(\delta - t)$-bit string.

1. If $v = v_\theta$: $g(v_\theta, x) = g(v_\theta, x') = w$; $b(v_\theta, x) = b(v_\theta, x') = 1$, $y(v_\theta, x) = 0^\delta$ and $y(v_\theta, x') = 1^\delta$.
2. If $v \in \mathcal{L}'(T)$: $g(v,x) = g(v,x') = w$; $b(v,x) = b(v,x') = 0$, $y(v,x) = 0^\delta = y(v,x')$.
3. If $v \in \mathcal{I}(T)$: $g(v,x) = g(v,x') = w$; $b(v,x) = b(v,x') = 0$, $y(v,x) = \mathsf{str}_T(v) = y(v,x')$.
4. If $v \in D_s \setminus T$: $\chi(v,x) = \chi(v,x') =$ any $(n - mt)$-bit string.

From the first point, it is clear that $x \neq x'$. We show that $H_p^{(A,s)}(x) = H_p^{(A,s)}(x')$. To do this, it is sufficient to show that the output of the root v_α of T is 1^m. The reason for this is the following. If v_α is the root of D_s, then this is clearly true. If v_α is an internal node of D_s, then the output of v_α is the same for both x and x'. The intermediate nodes in T do not influence the outputs of any node not in T. Further, by definition, $\chi(v,x) = \chi(v,x')$ for any node v not in T. Hence we must have $H_p^{(A,s)}(x) = H_p^{(A,s)}(x')$.

Now we proceed to show that the output of v_α is 1^m for both x and x'. Let z be a string of length $m = m' + K$. We call the substring $z_{m'+1}, \ldots, z_{m'+K}$ to be the *critical* substring of z and the positions $m'+1, \ldots, m'+K$ to be the *critical* positions of z. The notation $z^{(c)}$ denotes the substring $z_{m'+1} \cdots z_{m'+K}$.

The idea of the proof is the following. We have defined both x and x' in such a way that v_θ writes k into the critical positions of its output for both x and x'. Further all nodes in $\mathcal{L}'(T)$ write 0^K in the critical positions of its output. Now the idea is for v_α to "see" k in its input and hence output 1^m for both x and x'. This is done as follows. The output of each node is masked. Thus the output of v_θ and hence k is masked. The masking of the outputs of all nodes in $\mathcal{L}'(T)$ lead to the mask itself being available as input to the next node.

Let $v \in \mathcal{I}(T)$ and $v \neq v_\alpha$. Let T_v be the subtree rooted at v. There are two possibilities. Either $v_\theta \in T_v$ or $v_\theta \notin T_v$. If $v_\theta \in T_v$, then the critical substring of the output of v is $k \oplus \psi(T_v)$. On the other hand, if $v \notin T_v$, then the critical substring of the output of v is $\psi(T_v)$.

Let the n-bit inputs to v_α be z and z' corresponding to the messages x and x' respectively. Note that by the definition of $h_k()$ and x and x' we must have $z = z'$. Let the inputs to v_α in T be the nodes $v_1, \ldots, v_q$. Out of these v_θ belongs to exactly one T_{v_i}. Without loss of generality assume that $v_\theta \in T_{v_1}$. Then the critical substring of the output of T_{v_1} is $k \oplus \psi(T_{v_1})$ and the critical substring of the outputs of v_2 to v_q are $\psi(T_{v_2})$ to $\psi(T_{v_q})$ respectively. The string $y(v_\alpha, z) = \mathsf{str}_T(v_\alpha)$ encodes the fact that inputs of v_α in T are obtained from v_1 to v_q. Hence $\psi^{(c)}(T_{v_1}) \oplus \ldots \oplus \psi^{(c)}(T_{v_q}) = k$ which implies $\phi(z,k) = \mathsf{T} = \phi(z',k)$. Thus v_α outputs 1^m for both x and x'. Hence $\{H_p^{(A,s)}\}_{p \in \mathcal{P}}$ is not a UOWHF even if $\{h_k\}_{k \in \mathcal{K}}$ is a UOWHF. This gives us the required result. ■

We now translate Lemma 5 into a lower bound on the number of masks.

Definition 6 *Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be two subtrees of D_s. We denote by $T_1 \Delta T_2$ the subtree of D_s induced by the set of arcs $E_1 \Delta E_2$, where $E_1 \Delta E_2$ is the symmetric difference between E_1 and E_2.*

Definition 7 *Let $\mathcal{F}$ be a family of non-trivial subsets of D_s such that for any $T_1, T_2 \in \mathcal{F}$, the tree $T_1 \Delta T_2$ is also a non-trivial subtree of D_s. We call $\mathcal{F}$ a connected family of D_s.*

Lemma 8 *Let $s > 0$ and $A \in \boldsymbol{\mathcal{A}}$ be correct for s invocations. Let $\mathcal{F}$ be a connected family of D_s. Then (1) for any $T \in \mathcal{F}$, $\mathsf{vec}_\psi(T) \neq (0, \ldots, 0)$ and (2) for any $T_1, T_2 \in \mathcal{F}$, $\mathsf{vec}_\psi(T_1) \neq \mathsf{vec}_\psi(T_2)$. Consequently, $2^\rho - 1 \geq |\mathcal{F}|$ or equivalently $\rho \geq \lceil \log_2(|\mathcal{F}| + 1) \rceil$.*

Proof. The first point follows directly from Lemma 5. The second point is proved as follows. We have $\mathsf{vec}_\psi(T_1 \Delta T_2) = \mathsf{vec}_\psi(T_1) \oplus \mathsf{vec}_\psi(T_2)$. Since $\mathcal{F}$ is a connected family, $T_1 \Delta T_2$ is a non-trivial subtree of D_s. Thus if A is correct for s invocations, we must have $\mathsf{vec}_\psi(T_1 \Delta T_2) \neq (0, \ldots, 0)$, i.e., $\mathsf{vec}_\psi(T_1) \neq \mathsf{vec}_\psi(T_2)$. ■

Lemma 8 provides a lower bound on the number of masks in terms of sizes of connected families. Thus the task is to find a connected family of maximum size in D_s. We show the existence of a connected family of size $(s-1)$ in D_s. For each intermediate node $v \in D_s$, let P_v be the path from v to the final node of D_s. Define $\mathcal{F} = \{P_v : v \text{ is an intermediate node in } D_s\}$. It is easy to check that $\mathcal{F}$ is a connected family of size $(s-1)$. Hence we have the following result.

Theorem 9 *Let $s > 0$ and $A \in \boldsymbol{\mathcal{A}}$ be correct for s invocations. Then the number of masks required by A is at least $\lceil \log_2 s \rceil$.*

The bound in Theorem 9 is tight since Shoup's algorithm [9] meets this bound with equality. This also shows that Shoup's algorithm is optimal for the class $\boldsymbol{\mathcal{A}}$. Also we would like to point out that the lower bound of Theorem 9 can be improved for particular algorithms. For example, in [7] it has been shown that for a full binary tree algorithm using $2^T - 1$ processors which invokes $h_k()$ a total of $2^T - 1$ times, the number of masks required is at least $T + 1$. This is achieved by showing a connected family of cardinality $5.2^{T-2} - 2$ in the full binary tree of height T.

5 Finite Binary Tree Algorithm

Let $\{h_k\}_{k \in \mathcal{K}}$, $h_k : \{0,1\}^n \to \{0,1\}^m$ be a UOWHF whose domain is to be extended. *For our finite binary tree algorithm we require $n \geq 2m$.* A set of 2^t processors $P_0, \ldots, P_{2^t-1}$ will be used in the algorithm. Define $\delta(t) = 2^t n + (2^t - 1)(n - 2m) = 2^t(2n - 2m) - (n - 2m)$ and $\lambda(t) = 2^{t-1}n + 2^{t-1}(n - 2m) = 2^{t-1}(2n - 2m)$. The message x is of length $|x| = L$. We assume that $L \geq \delta(t)$. Otherwise we choose a $t' \in \{1, \ldots, t-1\}$, such that $L \geq \delta(t')$ and use only $2^{t'}$ of the 2^t processors. We define $\mathcal{I} = \{0, \ldots, 2^{t-1} - 1\}$, $\mathcal{L} = \{2^{t-1}, \ldots, 2^t - 1\}$.

We first define three parameters $q_t(L), r_t(L)$ and $b_t(L)$. If $L > \delta(t)$, then $q_t(L)$ and $r_t(L)$ are defined by the following equation: $L - \delta(t) = q_t(L)\lambda(t) + r_t(L)$, where $r_t(L)$ is the unique integer from the set $\{1, \ldots, \lambda(t)\}$. If $L = \delta(t)$, then $q_t(L) = r_t(L) = 0$. Define $b_t(L) = \lceil (r_t(L)/(2n - 2m)) \rceil$. *We will usually write q, r and t instead of $q_t(L), r_t(L)$ and $b_t(L)$ respectively.*

The message x is padded with $b(2n - 2m) - r$ many 0's to ensure that the length becomes $\delta(t) + q\lambda(t) + b(2n - 2m)$. The maximum number of 0's that are padded is $(2n - 2m) - 1$ and is independent of the message length. The algorithm goes through $R = q + t + 2$ rounds. In round 1, each processor gets an n-bit substring of the message x as input and produces an m-bit output. Also in rounds 2 to R and for $2^{t-1} \leq i \leq 2^t - 1$, each processor P_i gets as input either an n-bit substring of the message x or the empty string $\langle\rangle$ and correspondingly produces as output either an m-bit string or the empty string $\langle\rangle$.

For $1 < j \leq R$ and $0 \leq i \leq 2^{t-1} - 1$, in round j processor P_i reads the outputs of processors P_{2i} and P_{2i+1} in round $j-1$. Thus the same set of processors is used in each round. However, for convenience of description of our algorithm, we will consider R copies of the set of processors, denoted by $P_{i,j}$, $0 \leq i \leq 2^t - 1$ and $1 \leq j \leq R$. Thus we consider a directed graph $\mathcal{D} = (V, A)$, where $V = \{P_{i,j} : 0 \leq i \leq 2^t - 1, 1 \leq j \leq R\}$ and

$$A = \bigcup_{0 \leq i \leq \tau} \bigcup_{1 < j \leq R} \{(P_{2i,j-1}, P_{i,j}), (P_{2i+1,j-1}, P_{i,j})\}$$

where $\tau = 2^{t-1} - 1$. Since the outdegree of each vertex of $\mathcal{D}$ is at most one, we label the arcs as $A_{i,j}$ ($0 \leq i \leq 2^t - 1$, $1 \leq j \leq R-1$), where $A_{i,j} = (P_{i,j}, P_{\lfloor (i/2) \rfloor, j+1})$. For $0 \leq i \leq 2^{t-1} - 1$ and $1 \leq j \leq R-1$, we denote by $z_{i,j}$ the m-bit string which is the output of processor $P_{i,j}$. It is assumed that $z_{i,j}$ is associated with the arc $A_{i,j}$. Let M be a set of m-bit masks and $\psi : A \to M$ be an assignment of these masks to the arcs of the processor graph $\mathcal{D} = (V, A)$.

Each of the processors $P_{i,j}$ is given a string $u_{i,j}$ as input. For $1 \leq j \leq R-1$, define $\mathsf{UList}_j = \langle u_{0,j}, \ldots, u_{2^t-1,j} \rangle$ and U_j to be the concatenation of all the strings in UList_j. The strings U_j's are obtained from the message and $x = U_1 || \ldots || U_{R-1} || U_R$, where the lengths of the strings $u_{i,j}$'s and U_R is defined as follows. (See [8] for correctness of this formatting algorithm.)

$$\left.\begin{array}{rcl} |u_R| & = & \left\{\begin{array}{ll} n - 2m & \text{if } b > 0; \\ 0 & \text{otherwise.} \end{array}\right. \\ |u_{i,j}| & = & \left\{\begin{array}{ll} n & \text{if } (j = 1) \text{ or } (2 \leq j \leq q+1 \text{ and } i \in \mathcal{L}); \\ n & \text{if } j = q+2 \text{ and } 2^{t-1} \leq i \leq 2^{t-1} + b - 1; \\ 0 & \text{if } j = q+2 \text{ and } 2^{t-1} + b \leq i \leq 2^t; \\ 0 & \text{if } q+2 < j < R \text{ and } i \in \mathcal{L}; \\ n - 2m & \text{if } 2 \leq j \leq q+2 \text{ and } i \in \mathcal{I}; \\ n - 2m & \text{if } q+2 < j < R \text{ and } 0 \leq i \leq K_j - 1; \\ 0 & \text{if } q+2 < j < R \text{ and } K_j \leq i \leq 2^{t-1} - 1. \end{array}\right. \end{array}\right\} \quad (2)$$

Here $K_j = 2^{s-1} + k_s$, where $s = R - j$ and $k_s = \left\lfloor \frac{2^{t-s-1} + b - 1}{2^{t-s}} \right\rfloor$. For $0 \leq i \leq 2^t - 1$ and $1 \leq j \leq R-1$, the strings $z_{i,j}$'s are either m-bit strings or the empty string. If $z_{i,j}$ is an m-bit string, then define

$$s_{i,j} = z_{i,j} \oplus \psi(A_{i,j}), \quad (3)$$

We are now in a position to define the parallel UOWHF algorithm (PUA).

Parallel UOWHF Algorithm $\mathsf{PUA}(t, x)$

(a) **Case 1 ($j = 1$ and $0 \leq i \leq 2^t - 1$) :** $z_{i,1} = h_k(u_{i,1})$.

(b) **Case 2 ($j > 1$ and $2^{t-1} \leq i \leq 2^t - 1$) :** $\left\{\begin{array}{rcll} z_{i,j} & = & h_k(u_{i,j}) & \text{if } |u_{i,j}| = n; \\ & = & \langle\rangle & \text{otherwise.} \end{array}\right.$

(c) **Case 3 ($j > 1$ and $0 \leq i \leq 2^{t-1} - 1$) :** $\left\{\begin{array}{rcll} z_{i,j} & = & h_k(s_{2i,j-1} || s_{2i+1,j-1} || u_{i,j}) & \text{if } |u_{i,j}| = n - 2m; \\ & = & z_{2i,j-1} & \text{otherwise.} \end{array}\right.$

(d) **return** $z_{0,R}$.

end PUA

Note that for each j ($1 \leq j \leq R$), all the processors $P_{i,j}$ can operate in parallel to produce the strings $z_{i,j}$.

5.1 Definition of M and ψ

Algorithm PUA depends on the set of masks M and the map $\psi : A \to M$. Here we present our set of masks and our definition of ψ. Recall that $\nu(i) = j$ if $2^j | i$ and $2^{j+1} \not| i$. Also $l = level(i)$ is such that $2^{l-1} \leq i \leq 2^l - 1$. The set of masks M is the union of two disjoint sets of masks M_0 and M_1 defined as follows: $M_0 = \{m_0, \ldots, m_{c-1}\}$, where $c = 1 + \lfloor \log_2(R-1) \rfloor$; $M_1 = \{\alpha_1, \ldots, \alpha_t, \beta_0, \ldots, \beta_{d-1}\}$, where $d = 1 + \lfloor \log_2(t-1) \rfloor$. Thus $|M| = 2 + t + \lfloor \log_2(R-1) \rfloor + \lfloor \log_2(t-1) \rfloor$. The assignment of masks to the arcs is defined in the following manner.

$$\begin{aligned} \psi(A_{i,j}) &= m_{\nu(j)} && \text{if } i = 0; \\ &= \alpha_{t+1-l} && \text{if } i \equiv 1 \bmod 2,\ l = level(i); \\ &= \beta_{\nu(t+1-l)} && \text{if } i > 0 \text{ and } i \equiv 0 \bmod 2,\ l = level(i). \end{aligned}$$

We provide an explanation of the above assignment procedure. The assignment actually consists of two parts. In the first part, arcs of the type $A_{0,1}, A_{0,2}, \ldots, A_{0,R-1}$ are assigned masks according to the Shoup algorithm (see Section 3.1). The rest of the arcs are divided into rounds. Arcs A_{i,j_1} and A_{i,j_2} get the same mask for $i > 0$ and $1 \leq j_1, j_2 \leq R-1$. The arcs for a fixed round are assigned masks using the algorithm of Sarkar from Section 3.2. Thus the mask assigment algorithm combines the mask assignment algorithms of [9] and [7].

5.2 Definition of $\{H_p\}_{p \in \mathcal{P}}$

The UOWHF family $\{H_p\}_{p \in \mathcal{P}}$ is defined from the hash family $\{h_k\}_{k \in \mathcal{K}}$ using algorithm $\mathsf{PUA}(t, x)$ in the following manner.

$$H_p(x) = \mathsf{PUA}(t, x). \tag{4}$$

Here $p = k || \tilde{M}$, where $\tilde{M}$ is the concatenation of all the masks used by $\mathsf{PUA}(t, x)$.

5.3 Coping with Different Size Trees

The output of $H_p(x)$ depends on the parameter t which is the depth of the processor tree. Thus the output can be correctly recomputed only if the receiver has access to a binary tree of 2^t processors. This is clearly an undesirable situation. The way out of this situation is to have an algorithm where it is possible to simulate a binary tree of 2^t processors with a binary tree of $2^{t'}$ processors for any t' in $\{0, \ldots, t-1\}$. In fact, it is possible to perform such a simulation. For collision resistant hash functions the simulation algorithm has been presented in [8]. The same algorithm will also work in the case of UOWHF. Hence we do not present the simulation algorithm in this paper.

5.4 Speed-Up Over Sequential Algorithm

The number of invocations of h_k depends on the length of the message length L. Let $\eta(L)$ denote the number of invocations of h_k for a message of length L. Then $\eta(L) = \eta(L + b_t(L)(2n - 2m) - r_t(L))$. The value of $\eta(L)$ has been computed in [8] and is given by $\eta(L) = (q_t(L) + 2)2^t + 2b_t(L) - 1$. The same value also applies to the present case.

As shown in [8], $\eta(L)$ is also the number of invocations required in the sequential algorithm. The number of parallel rounds is $q_t(L) + t + 2$. Hence compared to the sequential construction, the tree construction is faster by a factor of $\mathsf{SF} = \frac{\eta(L)}{q+t+2} = \frac{(q+2)2^t+2b-1}{q+t+2}$. If $b > 0$, then $\mathsf{SF} \geq \frac{2^t(q+2)}{q+2+t} = \frac{2^t}{1+\frac{t}{q+2}}$. The parameter q grows linearly with the length of the message whereas t is fixed. Hence for moderately large messages, the speed-up is almost linear in the number of processors.

6 Security Reduction for $\{H_p\}_{p\in\mathcal{P}}$

Theorem 10 *If there is an (ϵ, N)-extended strategy for $\{H_p\}_{p\in\mathcal{P}}$, then there is an $(\frac{\epsilon}{\eta(L)}, N + 2\eta(L))$ strategy for $\{h_k\}_{k\in\mathcal{K}}$, where $h_k : \{0,1\}^n \to \{0,1\}^m$ and $H_p : \{0,1\}^L \to \{0,1\}^m$. Consequently, if the family $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF, then the family $\{H_p\}_{p\in\mathcal{P}}$ is also a UOWHF.*

Proof. Our proof is a reduction. We assume that there is an (ϵ, a)-extended strategy $\mathcal{B}$ for the family $\{H_p\}_{p\in\mathcal{P}}$ and construct an $(\frac{\epsilon}{\eta(L)}, N + 2\eta(L))$ strategy $\mathcal{A}$ for the family $\{h_k\}_{k\in\mathcal{K}}$. Thus if $\{H_p\}_{p\in\mathcal{P}}$ is not a UOWHF then $\{h_k\}_{k\in\mathcal{K}}$ is also not a UOWHF. The contrapositive of this statement gives us the desired result. We now turn to the actual reduction. The strategy $\mathcal{A}$ has two parts, $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}$. The algorithm $\mathcal{A}^{\text{find}}$ is as follows.

1. Run $\mathcal{B}^{\text{guess}}$ to obtain a string x of length L and state information σ_1.
2. Randomly choose an I, J $(0 \leq I \leq 2^t - 1,\ 1 \leq J \leq R)$ such that the string $u_{I,J} \neq \langle\rangle$.
3. If $|u_{I,J}| = n$, then set $w = u_{I,J}$.
4. If $|u_{I,J}| = n - 2m$, then randomly choose two m-bit strings w_0, w_1 and set $w = w_0||w_1||u_{I,J}$.
5. Output string w and state information $\sigma = (\sigma_1, I, J)$.

After $\mathcal{A}^{\text{guess}}$ produces an n-bit output w, the adversary is given a random $k \in \mathcal{K}$. Now the adversary runs the algorithm $\mathcal{A}^{\text{find}}$ given below.

1. If I and J are such that $|u_{I,J}| = n$, then define all the masks in M randomly.
2. If $I = 0$, then run mask defining algorithm MDEF 1 to define all the masks in M.
3. If $I > 0$, then run mask defining algorithm MDEF 2 to define all the masks in M.
4. Let $\tilde{M}$ be the concatenation of all the masks in M and set $p = k||\tilde{M}$.
5. Run $\mathcal{B}^{\text{find}}(x, p, \sigma_1)$ to obtain string x'.
6. Run algorithm PUA on x' and store all the intermediate values $u'_{i,j}$, $z'_{i,j}$ and $s'_{i,j}$.
7. If $|u_{I,J}| = n$, $w = u_{I,J} \neq u'_{I,J}$ and $z_{I,J} = z'_{I,J}$, then return $u_{I,J}$ and $u'_{I,J}$.
8. If $|u_{I,J}| = n - 2m$, $w = w_0||w_1||u_{I,J} \neq s'_{2I,J-1}||s'_{2I+1,J-1}||u'_{I,J}$ and $z_{I,J} = z'_{I,J}$, then return w and $s'_{2I,J-1}||s'_{2I+1,J-1}||u'_{I,J}$.
9. Else return failure.

The task of the mask defining algorithms is to define the masks so that the input to processor $P_{I,J}$ is the string w. If $|u_{I,J}| = n$, then $w = u_{I,J}$ and so the masks are defined randomly. On the other hand, if $|u_{I,J}| = n - 2m$, then $w = w_0||w_1||u_{I,J}$, where w_0 and w_1 are m-bit random strings. In this situation, we would like to have $w_0 = s_{2I,J-1}$ and $w_1 = s_{2I+1,J-1}$. However, the algorithm $\mathcal{B}^{\text{find}}$ produces the string x before knowing the key p. The key p can only be determined after the key k becomes known. Thus after k is revealed, the masks in M are defined so that $w_0 = s_{2I,J-1}$ and $w_1 = s_{2I+1,J-1}$. The key

for the algorithm $\mathcal{B}^{\text{find}}$ is then determined to be $k||\tilde{M}$. While defining the masks we must ensure that each mask is chosen according to the uniform distribution on the set of m-bit strings. We now describe the two mask defining algorithms.

The algorithm MDEF 1 does the following. First it randomly defines the masks $\alpha_1, \ldots, \alpha_{t-1}$. Then it does a partial run of the algorithm PUA in the following manner. It operates each processor $P_{i,j}$ for $i > 0$ and $1 \leq j \leq R$. (Note that since the masks $m_0, \ldots, m_{c-1}$ are as yet undefined, it is not possible to operate $P_{0,j}$ for any j.) This partial execution of the algorithm defines all the strings $z_{1,J}$ for $1 \leq J \leq R$. MDEF 1 now defines $\alpha_t = w_1 \oplus z_{1,J-1}$. Once α_t is defined, all the strings $s_{1,j}$ can be defined, since $s_{1,j} = z_{1,j} \oplus \psi(A_{1,j}) = z_{i,j} \oplus \alpha_t$. Now we consider the operation of $P_{0,1}, P_{0,2}, \ldots, P_{0,R}$ to be a sequential operation. For $j > 1$, the inputs to $P_{0,j}$ are $s_{0,j-1}, s_{1,j-1}$ and $u_{0,j}$. Of these the inputs $s_{1,j-1}$ and $u_{0,j}$ are already known. So we have to define the inputs $s_{0,j-1}$ for $j \geq 1$. Moreover, we have to ensure that $s_{0,J-1} = w_1$. This is exactly the problem for the sequential construction of UOWHF given in Section 3.1. Now the mask defining algorithm presented in [4] is used to correctly define the masks $m_0, \ldots, m_{c-1}$.

The first step of Algorithm MDEF 2 is to randomly define the masks $m_0, \ldots, m_{c-1}$ and the mask α_t. Let $L = level(I)$. There are two cases to consider: (a) $L \leq J$ and (b) $L > J$. We first describe Case (a).

In Case (a) we have $L \leq J$ and no processor $P_{i,j}$ with $j < J - L$ will be used in defining the masks in $M_1 \setminus \{\alpha_t\}$. In this situation, for $i \geq 1$, we merge all processors $P_{i,j}$ for $J - L \leq j \leq J$ into a single processor P_i. Then the algorithm becomes the binary tree based UOWHF algorithm of Sarkar described in Section 3.2. The mask defining algorithm of [7] is used to properly define the masks in this case.

In Case (b), it will not be possible to descend L steps in the tree starting from $P_{I,J}$. After J steps we will reach round 1. The mask defining algorithm of [7] uses the algorithm of [4] along certain paths in the full binary tree. In this case such paths will not be complete. However, it is not difficult to verify that this makes the task of mask definition easier. The details are quite straightforward and hence are omitted.

Thus in both Cases (a) and (b) it is possible to properly define the masks in the set M. Further, any mask is either chosen to be random or is obtained by XOR with a random string. Hence each mask is chosen independently and uniformly at random from the set of all m-bit strings, which shows that $\tilde{M}$ is a random string.

To complete the proof we need to lower bound the probability of success. Supppose that x and x' produce a collision for H_p. Then using a backward induction it is possible to prove that for some I_1, J_1, processor P_{I_1,J_1} must produce a collision for h_k. (Details of this backward induction for collision resistant function can be found in [8].) The probability that $(I, J) = (I_1, J_1)$ is $\frac{1}{\eta(L)}$. Since the probability that x and x' provide a collision for H_p is at least ϵ, it follows that the success probability for finding a collision for h_k is at least $\frac{\epsilon}{\eta(L)}$. Strategy $\mathcal{B}$ invokes h_k at most N times and strategy $\mathcal{A}$ invokes h_k at most $2\eta(L)$ additional times. ■

7 Comparison to Previous Algorithms

In Table 1 we compare the performance of the different known algorithms with PUA. The comparison is for messages of length $L = 2^T(n-m) - (n-2m)$ with $T > 2$. The parameters n and m are constant and hence L grows as T grows. Note that from Theorem 9, we know that at least T masks will be

Parameter	Sequential [9]	[1]	[7]	PUA
# processors	1	$2^T - 1$	$2^T - 1$	2^t
# masks	T	$2(T-1)$	$T + \lfloor \log(T-1) \rfloor$	$\simeq T + 2 + \lfloor \log(t-1) \rfloor$
speed-up	1	$\frac{2^T}{T}$	$\frac{2^T}{T}$	$\frac{2^t}{1+2^{t-T}(t+2)}$

Table 1: Comparison of domain extenders for UOWHF.

required by any algorithm in $\mathcal{A}$. For the purpose of comparison we assume that Algorithm PUA uses 2^t processors where t is a constant less than $T-1$. Table 1 clearly shows the superiority of PUA over previous binary tree algorithms in terms of key expansion and efficiency of speed-up.

A more detailed comparison is presented in the Appendix.

8 Concluding Remarks

In this paper we have obtained a generic lower bound on the length of key expansion required by a large class of algorithms to securely extend the domain of a UOWHF. Our bound shows that Shoup's algorithm is optimal for this class of algorithms.

We have also presented a *finite* binary tree algorithm for securely extending the domain of a UOWHF. In previously known binary tree algorithms the number of processors increased with the length of the message, which made such algorithms unsuitable for practical use. To the best of our knowledge our algorithm is the only practical parallel algorithm to extend the domain of a UOWHF. The key length expansion of our algorithm is a constant number of bits more than the lower bound and the efficiency of speed-up is roughly equal to the number of processors for moderately long messages.

Several open problems remain. The first problem is to investigate whether the lower bound that we have obtained holds for *any* domain extender for a UOWHF. If not, then is it possible to obtain a domain extender with key length expansion less than that of Shoup's algorithm? Clearly such an algorithm cannot belong to the "natural" class of algorithms for which we have obtained the lower bound.

For our finite binary tree algorithm, the number of masks used is more than the lower bound. One problem is to try and close this gap. This can be done on the one hand by obtaining a better mask assignment strategy and on the other hand by improving the lower bound for the finite binary tree algorithm. One way of tackling the second problem is to exhibit a larger connected family in the digraph defined by the algorithm.

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] P. Sarkar. Construction of UOWHF: Tree Hashing Revisited. *IACR e-print server*, 2002/058, http://eprint.iacr.org.

[8] P. Sarkar and P. J. Schellenberg. A Parallelizable Design Principle for Cryptographic Hash Functions. *IACR e-print server*, 2002/031, http://eprint.iacr.org.

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

[10] 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.

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

A Comparison of Key Length Expansion

The key length expansion is m times the number of masks used. Hence it is sufficient to compare the number of masks used by the different algorithms. First we compare the number of masks used by Algorithm PUA to the number of masks used by Algorithm SeqUOWHF.

Theorem 11 *Let x be a message of length L and N_1 be the number of masks required by Algorithm* PUA *to hash x. Further, let N_2 be the lower bound on the number of masks required by any algorithm in $\mathcal{A}$ to hash x. Then* $\log\left(1+\frac{t-2}{q+3}\right) < N_1 - N_2 - \lfloor \log(t-1) \rfloor < 2+\varepsilon+\log\left(1+\frac{t-1}{q+2}\right)$ *where $\varepsilon \ll 1$, and t is the height of the binary processor tree used by* PUA.

Proof. From Section 5.1, we have $N_1 = 2 + t + \lfloor \log(R-1) \rfloor + \lfloor \log(t-1) \rfloor$.

For Algorithm SeqUOWHF, the number of masks used is $N_2 = 1 + \lfloor \log(r-1) \rfloor$, where r is the number of times the hash function h_k is invoked. The number of times the hash function h_k is invoked by PUA is $\eta_t(L) = (q+2)2^t + 2b - 1$ (see Proposition 5.4). Also it has been proved in [8] that this is the number of times the hash function h_k will be invoked by SeqUOWHF. Hence $N_2 = 1 + \lfloor \log(\eta_t(L) - 1) \rfloor$. Thus

$$\begin{aligned} N_1 - N_2 &= (2 + t + \lfloor \log(R-1) \rfloor \lfloor \log(t-1) \rfloor) - (1 + \lfloor \log(\eta_t(L) - 1) \rfloor) \\ &= (2 + t + \lfloor \log(q+t+1) \rfloor + \lfloor \log(t-1) \rfloor) - (1 + \lfloor \log(2^t(q+2) + 2b - 2) \rfloor) \end{aligned}$$

The parameter b is either 0 or equal to $\lceil \frac{r}{2n-2m} \rceil$, where r ranges from 1 to $\lambda(t) = 2^{t-1}(2n-2m)$. Hence $0 \leq b \leq 2^{t-1}$. Thus we have $1 + \lfloor \log(2^t(q+2)-2) \rfloor \leq N_2 \leq 1 + \lfloor \log(2^t(q+3)-2) \rfloor$. From this we get $\log(2^t(q+2)-2) < N_2 \leq 1 + \log(2^t(q+3)-2)$. We can write this as $t + \log(q+2) - \varepsilon < N_2 \leq t + 1 + \log(q+3) - \sigma$ where $\varepsilon = -\log(1 - \frac{1}{2^{t-1}(q+2)})$ and $\sigma = -\log(1 - \frac{1}{2^{t-1}(q+3)})$. Clearly, $\varepsilon, \sigma \ll 1$. Thus we have

$$1 + \lfloor \log(q+t+1) \rfloor - \log(q+3) + \sigma \leq N_1 - N_2 - \lfloor \log(t-1) \rfloor < 2 + \varepsilon + \lfloor \log(q+t+1) \rfloor - \log(q+2).$$

This gives $\log\left(1 + \frac{t-2}{q+3}\right) < N_1 - N_2 - \lfloor \log(t-1) \rfloor < 2 + \varepsilon + \log\left(1 + \frac{t-1}{q+2}\right)$ which is the required result. ■

The height t of the binary tree is independent of the message length and is a constant for a particular implementation. For moderately long messages, Algorithm PUA requires at most $\lfloor \log(t-1) \rfloor + 2$ more masks than the minimum possible number of masks. Consequenly PUA makes only a *constant* amount of key length expansion compared to the best algorithm in the class $\mathcal{A}$.

We now compare the number of masks used by PUA to the number of masks used by TreeUOWHF. For TreeUOWHF we use the mask assignment procedure used by Sarkar [7] (see Section 3.2) and not the assignment procedure used by Bellare and Rogaway [1]. This is because the number of masks required in the first case is less than the number of masks required in the second case.

Theorem 12 *Suppose x is a message of length $L = 2^T(n-m) - (n-2m)$ and* TreeUOWHF *uses a processor tree of height $T > 2$ to hash x. Suppose* PUA *is used to hash x using a processor tree of size $t < T - 1$. Let A and B be the number of masks used by* TreeUOWHF *and* PUA *respectively. Then* $A - B > \lfloor \log(T-1) \rfloor - \lfloor \log(t-1) \rfloor - 2 - \left\lfloor \log(1 + \frac{t-2}{2^{T-t}}) \right\rfloor$.

Proof. The number of masks used by TreeUOWHF is $A = T + \lfloor \log(T-1) \rfloor$ for $T > 2$ (see [7]). The parameters q and r of PUA are determined as follows: $L - \delta(t) = (2^T - 2^{t+1})(n-m) = q2^t(n-m) + r$, where $q = 2^{T-t} - 3$ and $r = \lambda(t)$. The number of masks required by PUA is $B = 2 + t + \lfloor \log(t-1) \rfloor + \lfloor \log(q+t+1) \rfloor = 2 + t + \lfloor \log(t-1) \rfloor + \lfloor \log(2^{T-t} + t - 2) \rfloor$.

Note that $\log(2^{T-t} + t - 2) = T - t + \log(1 + \frac{t-2}{2^{T-t}})$. Using this in B, we get the required result. ■

Note that the parameter T grows with the length of the message while t is a constant. Hence the difference $A - B$ grows with the length of the message, which means that compared to TreeUOWHF, algorithm PUA becomes more and more efficient as the length of the message grows.

B Variable Length Inputs

Given a UOWHF $\{h_k\}_{k \in \mathcal{K}}$, where $h_k : \{0,1\}^n \to \{0,1\}^m$, we construct a UOWHF $\{H_p^*\}_{p \in \mathcal{P}}$, where $H^* : \cup_{i=1}^{L} \{0,1\}^i \to \{0,1\}^m$. The UOWHF $\{H_p^*\}_{p \in \mathcal{P}}$ can handle variable length messages with maximum length 2^{n-m}. Since we require $n \geq 2m$, we have $n - m \geq m$. Practical digests must be at least 64 bits long and hence $\{H_p^*\}_{p \in \mathcal{P}}$ can handle messages of maximum length 2^{64} which is sufficient for any conceivable purpose. We note that [1] provides a method for tackling variable length inputs based on the use of two keys.

The definition of $\{H_p^*\}_{p \in \mathcal{P}}$ is based on Algorithm PUA. Suppose Algorithm PUA uses a binary processor tree of height t and a set of masks M to extend the domain of a UOWHF $\{h_k\}_{k \in \mathcal{K}}$. Let $\tilde{M}$

be the concatenation of all the masks in M. Let $p = k||\tilde{M}$ and let x be a message of maximum length 2^{n-m}. Then $\{H_p^*\}_{p\in\mathcal{P}}$ is defined in the following manner:

$$H_p^*(x) = h_k(\mathsf{PUA}(t,x)||\mathsf{bin}_{n-m}(|x|)), \tag{5}$$

where $\mathsf{bin}_k(i)$ denotes the k-bit binary representation of i, $0 \le i < 2^k$. The idea is to compute the output z of $\mathsf{PUA}(t,x)$, concatenate $|x|$ as an $(n-m)$-bit binary number to z and apply h_k to the resulting n-bit string u.

Theorem 13 *Let $\mathcal{A}$ be an (ϵ, N)-extended strategy for $\{H_p^*\}_{p\in\mathcal{P}}$, with $H_p^* : \cup_{i=1}^{2^{n-m}}\{0,1\}^i \to \{0,1\}^m$. Then there is an $(\epsilon', N+N'+2)$-strategy $\mathcal{B}$ for $\{h_k\}_{k\in\mathcal{K}}$, where $\epsilon' \ge \frac{\epsilon}{\eta(2^{n-m})}$ and $N' \le \eta(2^{n-m})$. Consequently, if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF then so is $\{H_p^*\}_{p\in\mathcal{P}}$.*

Proof. We have to describe the two stages of the adversarial strategy $\mathcal{B}$ for $\{h_k\}_{k\in\mathcal{K}}$. The Algorithm $\mathcal{B}^{\text{guess}}$ is same as $\mathcal{B}^{\text{guess}}$ in the proof of Theorem 10. $\mathcal{B}^{\text{guess}}$ first invokes $\mathcal{A}^{\text{guess}}$ to obtain a string x of length $L \le 2^{n-m}$ and then outputs an n-bit string u and some state information s. The adversary is then given a random $k \in \mathcal{K}$. Now the algorithm $\mathcal{B}^{\text{find}}$ has to be described. The first task of $\mathcal{B}^{\text{find}}$ is to define the masks in M as in the proof of Theorem 10. Then $p = k||\tilde{M}$ is the key for the hash function H_p^*. $\mathcal{B}^{\text{find}}$ now invokes $\mathcal{A}^{\text{find}}$ with x and p to obtain a string x'. Suppose x and x' provide a collision for H_p^* with probability at least ϵ. If $|x| \ne |x'|$, then $\mathsf{bin}_{n-m}(|x|) \ne \mathsf{bin}_{n-m}(|x'|)$ and we immediately have a collision for h_k. On the other hand, if $|x| = |x'|$, then as in the proof of Theorem 10, we obtain a collision for h_k with probability at least $\frac{\epsilon}{\eta(L)}$. Also the number of times h_k is invoked is at most N plus twice the number of times h_k is invoked to compute H_p^*. The number of times h_k is invoked to compute H_p^* is equal to one plus the number of times h_k is invoked by Algorithm $\mathsf{PUA}(t,x)$ which is equal to $1+\eta(L)$. This gives us the required result. ■