Masking Based Domain Extenders for UOWHFs: Bounds and Constructions

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

Abstract

We study the class of masking based domain extenders for UOWHFs. Our first contribution is to show that any correct masking based domain extender for UOWHF which invokes the compression UOWHF s times must use at least $\lceil \log s \rceil$ masks. As a consequence we obtain the optimality of Shoup's algorithm among the class of masking based domain extenders. Our second contribution is to present a new parallel domain extender for UOWHF. The new algorithm obtains an asymptotically optimal speed-up over the sequential algorithm and the key expansion is almost everywhere optimal, i.e., it is optimal for almost all possible number of invocations of the compression UOWHF.

Keywords : UOWHF, domain extender, parallel algorithm.

1 Introduction

A universal one-way hash function (UOWHF) is a function family $\{h_k\}_{k\in\mathcal{K}}$ with $h_k : \{0,1\}^n \rightarrow \{0,1\}^m$, for which the following task of the adversary is computationally infeasible: the adversary chooses an n-bit string x, is then given a k chosen uniformly at random from $\mathcal{K}$ and has to find a x' such that $x \neq x'$ and $h_k(x) = h_k(x')$. The notion of UOWHF was introduced in [8].

Intuitively, a UOWHF is a weaker primitive than a collision resistant hash function (CRHF), since the adversary has to commit to the string x before knowing the actual hash function h_k for which a collision has to be found. In fact, Simon [13] has shown that there is an oracle relative to which UOWHFs exist but CRHFs do not exist. Further, as pointed out in [1], the birthday paradox does not apply to the UOWHF and hence the message digest can be smaller. Thus a construction for UOWHF may be faster than a construction for CRHF.

There is a second and perhaps a more important reason to prefer UOWHF over CRHF. A protocol built using a UOWHF maybe "more" secure than a protocol built using CRHF. The intuitive reason being that even if it is possible to find a collision for a hash function, it might still be difficult to find a collision for it when considered as a UOWHF. This situation is nicely summed up in [1]: "Ask less of a hash function and it is less likely to disappoint!"

The important paper by Bellare and Rogaway [1] provides the foundation for the current study on UOWHFs. They introduce the notion of domain extender for UOWHF; show that the classical Merkle-Damgård algorithm does not work for UOWHFs; provide several new constructions for

UOWHF domain extenders and finally provide a secure digital signature scheme based on UOWHF in the hash-then-sign paradigm.

The study in [1] shows that extending the domain usually requires as associated increase in key length. One of the major new ideas behind their domain extending algorithm is "masking" the outputs of intermediate invocations by random strings. This idea of masking based algorithms have been later pursued by several authors [12, 3, 11, 10, 7].

Our Contributions: We study the class $\boldsymbol{\mathcal{A}}$ of *all* masking based domain extenders for UOWHFs. The efficient domain extending algorithms of [1, 12, 3, 11, 10, 7] belong to $\boldsymbol{\mathcal{A}}$. We show that any correct algorithm in $\boldsymbol{\mathcal{A}}$ which makes s invocations of the compression UOWHF must use at least $\lceil \log s \rceil$ masks. This provides a lower bound on the key length expansion made by any masking based domain extender. This lower bound immediately shows that Shoup's algorithm [12] achieves the minimum key expansion among all masking based domain extenders.

Our second contribution is to describe an efficient parallel masking based domain extender. The main features of our algorithm are the following.

Parallelism : We obtain a practical parallel domain extender. From a practical point of view, the construction is efficient and easy to implement. From a theoretical point of view, the attained speed-up is asymptotically optimal. See Section C.2 for more details.

Key Expansion : The key expansion made by our algorithm is *almost everywhere* optimal, i.e., it is optimal for almost all possible number of invocations of the compression UOWHF. See Section 5.3 for more details.

Thus our algorithm provides a satisfactory parallel domain extender for UOWHF and to a certain extent completes the line of research on obtaining efficient domain extenders for UOWHFs which was started in [1].

2 Preliminaries

All logarithms in this paper are to the base 2. The notation $x \in_r A$ denotes the (uniformly at) random choice of the element x from the set A. Also $\boldsymbol{\lambda}$ denotes the empty string. By an (n,m) function f we will mean a function $f : \{0,1\}^n \to \{0,1\}^m$. A formal definition for UOWHF is given in [8]. In this paper we will be interested in "securely" extending the domain of a given UOWHF. Our proof technique will essentially be a reduction. We formalize this as a reduction between two suitably defined problems. This approach has previously been adopted by Stinson [14] in the study of collision resistant hash functions.

Let $\mathbf{F} = \{h_k\}_{k \in \mathcal{K}}$ be a keyed family of hash functions, where each h_k is an (n,m) function, with $n > m$. Consider the following adversarial game $\mathcal{G}(\mathbf{F})$ for the family $\mathbf{F}$.

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 $h_k(x) = h_k(x')$.

Based on $\mathcal{G}(\mathbf{F})$ we define the following problem for the family $\mathbf{F}$.

Problem	:	$\mathcal{G}(\mathbf{F})$-UOWHF
Instance	:	Family $\mathbf{F}$.
Task	:	To win the game $\mathcal{G}(\mathbf{F})$.

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 state. In the second stage $\mathcal{A}^{\text{find}}(x, k, \mathsf{state})$, the adversary either finds a x' which provides a collision for h_k or it reports failure. Both $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x, k, \mathsf{state})$ are probabilistic algorithms. The success probability of the strategy is measured over the random choices made by $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x, k, \mathsf{state})$ and the random choice of k in Step 2 of the game. We say that $\mathcal{A}$ is an (ϵ, q)-strategy for $\mathcal{G}(\mathbf{F})$-UOWHF if the success probability of $\mathcal{A}$ is at least ϵ and it invokes some hash function from the family $\mathbf{F}$ at most q times. Informally, we say that $\mathbf{F}$ is a UOWHF if there is no "good" winning strategy for the game $\mathcal{G}(\mathbf{F})$.

In this paper, we are interested in extending the domain of a UOWHF. Let $\mathbf{F} = \{h_k\}_{k \in \mathcal{K}}$, where each h_k is an (n, m) function. For $i \geq 1$, let $n_i = n + (i-1)(n-m)$. Define $\mathbf{F}_0 = \mathbf{F}$ and for $i > 0$, define $\mathbf{F}_i = \{H_{p_i}\}_{p_i \in \mathcal{P}_i}$, where each H_{p_i} is an (n_i, m) function. The family $\mathbf{F}_i$ is built from the family $\mathbf{F}$. In fact, as shown in Proposition 1, a function in $\mathbf{F}_i$ is built using exactly i invocations of some function in $\mathbf{F}$.

We consider the problem $\mathcal{G}(\mathbf{F}_i)$-$\mathsf{UOWHF}$. We say that the adversary has an (ϵ, q)-strategy for $\mathcal{G}(\mathbf{F}_i)$-$\mathsf{UOWHF}$ if there is a strategy $\mathcal{B}$ for the adversary with probability of success at least ϵ and which invokes some hash function from the family $\mathbf{F}$ at most q times. Note that $\mathbf{F}_i$ is built using $\mathbf{F}$ and hence while studying strategies for $\mathcal{G}(\mathbf{F}_i)$ we are interested in the number of invocations of hash functions from the family $\mathbf{F}$.

The correctness of our construction will essentially be a Turing reduction. We will show that if there is an (ϵ, q)-strategy for $\mathcal{G}(\mathbf{F}_i)$, then there is an (ϵ_1, q_1)-strategy for $\mathbf{F}_i$, where ϵ_1 is not "significantly" less than ϵ and q_1 is not "significantly" more than q. In fact, we will have $\epsilon_1 = \epsilon/i$ and $q_1 = q + 2i$. Since $\mathbf{F}_i$ invokes a hash function from $\mathbf{F}$ a total of i times, we "tolerate" a reduction in success probability by a factor of $1/i$. The intuitive interpretation of the reduction is that if $\mathbf{F}$ is a UOWHF then so is $\mathbf{F}_i$ for each $i \geq 1$.

The key length for the base hash family $\mathbf{F}$ is $\lceil \log |\mathcal{K}| \rceil$. On the other hand, the key length for the family $\mathbf{F}_i$ is $\lceil \log |\mathcal{P}_i| \rceil$. Thus increasing the size of the input from n bits to n_i bits results in an increase of the key size by an amount $\lceil \log |\mathcal{P}_i| \rceil - \lceil \log |\mathcal{K}| \rceil$.

3 Previous Constructions

Naor and Yung [8] introduced UOWHF and provided direct construction of UOWHF based on one way permutations and universal hash families. Other direct constructions based on general and algebraic assumptions are given in [4, 9]. However, as observed in [1], these constructions are not very efficient. This motivated [1] to consider a "UOW compression family" and then securely extend the domain to obtain a UOWHF. For practical applications, the compression family is to be built from the compression functions of SHA or RIPEMD by considering a part of the input to be the key.

The paper [1] considers two methods for securely extending the domain of a UOWHF: use of separate keys for different invocations of the compression function; use of one key for the compression function and some auxiliary keys which are used to "mask" intermediate outputs by XORing with them. As observed in [1], the second technique leads to a lesser key expansion and has been further developed in a sequence of papers. We provide a brief description of some of the masking type constructions which have already been proposed.

3.1 Sequential Algorithm

The Merkle-Damgård construction [5, 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 [12], 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 [12] and in [6] 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 Algorithms

Multi-way trees have been used for extending the domain of a UOWHF [1]. The binary tree version of this construction has also been considered in [11, 7]. We describe the binary tree algorithm. 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, 11] 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 TreeUOWHFwe have to define M and ψ. We do this separately for [1] and [11].

Bellare and Rogaway [1] : In this case $M = \{\alpha_1, \dots, \alpha_{T-1}, \beta_1, \dots, \beta_{T-1}\}$ 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 [11] : In this case $M = \{\alpha_1, \dots, \alpha_{T-1}, \beta_0, \dots, \beta_{r-1}\}$ 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.

Nandi [7] : A more complicated mask assignment algorithm is used which results in the use of only $T + \log^*(T)$ masks.

Other tree based algorithms are described in Lee at al [3] and Sarkar [10]. The algorithm of Lee et al [3] uses an "l-dimensional" construction and uses T masks for 2^T invocations of the compression UOWHF. On the other hand, the algorithm of [10] uses a finite binary tree of processors. All the other tree based algorithms have the (undesirable) property that the size of the tree grows with the size of the message.

4 Lower Bound on Key Expansion

In this section, we consider the problem of minimising the key expansion while securely extending the domain of a UOWHF. More precisely, we are interested in obtaining a lower bound on key length expansion. Obtaining a complete answer to this problem is in general difficult. Thus we adopt a simpler approach to the problem. We fix a class of possible domain extending algorithms and obtain a lower bound on key expansion for any algorithm in this class. (Note that in computer science, this is the usual approach for proving lower bounds on algorithmic problems. For example, the lower bound of $O(n \log n)$ for sorting n elements is obtained for the class of all *comparison based algorithms.*)

The usefulness of our lower bound depends on the class of algorithms that we consider. The class that we consider consists of all masking based domain extending algorithms. (We make this more precise later.) All the constructions described in Section 3 belong to our class. We consider this to be sufficient evidence for the usefulness of our lower bound. We would like to point out that our lower bound does not hold for *any* domain extending algorithm. Thus it might be possible to achieve lower key expansions. However, any such algorithm must adopt a different approach to the construction than the masking based algorithms described in [1, 12, 11, 10, 7, 3]. One possible approach could be to develop the technique of using separate keys for the compression functions (see [1]).

Let $\mathbf{F} = \{h_k\}_{k \in \mathcal{K}}$, where each h_k is an (n, m) function. We are interested in the class $\boldsymbol{\mathcal{A}}$ of masking based domain extension algorithms. 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 some function $h_k \in \mathbf{F}$ a finite number of 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 $\{\mu_0, \dots, \mu_{\rho-1}\}$.
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.

The sequential and the binary tree based algorithms described in Section 3 belong to $\boldsymbol{\mathcal{A}}$. Also the "l-dimensional algorithm" of Lee et al [3] and the finite binary tree of algorithm of [10] belong to the class $\boldsymbol{\mathcal{A}}$. In the following we make a general study of any algorithm in $\boldsymbol{\mathcal{A}}$, with particular emphasis on obtaining a lower bound on key expansion made by any algorithm in $\boldsymbol{\mathcal{A}}$.

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 $n + (s-1)(n-m)$.*

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 $\mathbf{F}^{(A,s)} = \{H_p^{(A,s)}\}_{p \in \mathcal{P}}$, where $\mathcal{P} = \{0,1\}^{|k|+m\rho}$ and each $H_p^{(A,s)}$ is an $(n + (n-m)(s-1), m)$ function. The structure of any algorithm $A \in \boldsymbol{\mathcal{A}}$ which makes s invocations of $h_k()$ is described by a labelled directed graph $D_s^A = (V_s, E_s, \psi_s)$, 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 $\delta(D_s^A) = \max\{\mathsf{indeg}(v) : v \in V_s\}$. We call $\delta(D_s^A)$ to be the fan-in of algorithm A for s invocations.

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

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

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^A does not depend on the particular family $\mathbf{F}$.

Let T be a non-trivial subtree of D_s^A. 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. We say that D_s^A is *null-free* if $\mathsf{vec}\psi(T) \neq (0, \ldots, 0)$ for each non-trivial subtree of D_s^A.

We now turn to the task of obtaining a lower bound on key expansion made by any algorithm A in $\boldsymbol{\mathcal{A}}$. This consists of two tasks. Firstly, we show that for any "correct UOWHF preserving domain extender" A which invokes some function from the compression UOWHF exactly s times, the DAG D_s^A must be null-free. This translates the problem into a combinatorial one. Our second task is to use use this combinatorial property to obtain the required lower bound.

The intuitive idea behind the first part is as follows. Given D_s^A and a family $\mathbf{F}'$ with suitable parameters, we construct a family $\mathbf{F}$ such that if $\mathbf{F}'$ is a UOWHF, then so is $\mathbf{F}$. Then we extend the domain of $\mathbf{F}$ using D_s^A to obtain the family $\mathbf{F}^{(A,s)}$ and show that if D_s^A is not null-free then it is possible to exhibit a collision for every function in $\mathbf{F}^{(A,s)}$. Now we argue as follows. If $\mathbf{F}'$ is a UOWHF and A is correct for s invocations, then $\mathbf{F}^{(A,s)}$ must also be a UOWHF and hence D_s^A must be null-free. This intuitive argument is now formalized in terms of reductions.

Let $A \in \boldsymbol{\mathcal{A}}$ and D_s^A be the DAG corresponding to s invocations of the compression family by A. We set $\delta = \delta(D_s^A)$. Let $\mathbf{F}' = \{h'_k\}_{k\in\mathcal{K}}$ where each h' is an (n, m') function with $\mathcal{K} = \{0,1\}^K$, $m = m' + K$ and $n = \delta m + \delta + 1$. For $z \in \{0,1\}^n$ 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 $\emptyset \ne S \subseteq \mathsf{KLst}(z)$. We define the family of functions $\mathbf{F} = \{h_k\}_{k\in\mathcal{K}}$, where each h_k is an (n,m) function in the following manner.

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

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.

Proposition 4 *Suppose there is an (ϵ, q)-strategy for $\mathcal{G}(\mathbf{F})$. Then there is an $(\epsilon - \frac{1}{2^K}, q)$-strategy for $\mathcal{G}(\mathbf{F}')$.*

Intuitively, this means that if $\mathbf{F}'$ is a UOWHF, then so is $\mathbf{F}$. In the next result we show that if D_s^A is not null-free, then it is possible to exhibit a collision for each function in $\mathbf{F}^{(A,s)}$.

Lemma 5 *Let $A \in \boldsymbol{\mathcal{A}}$ and $\mathbf{F}$ be defined as in (1). For $s > 0$, let $\mathbf{F}^{(A,s)}$ be the family obtained by extending the domain of $\mathbf{F}$ using D_s^A. If D_s^A is not null-free, then it is possible to define two strings x, x' such that $x \ne x'$ and $H_p^{(A,s)}(x) = H_p^{(A,s)}(x')$ for any $H_p^{(A,s)} \in \mathbf{F}^{(A,s)}$,*

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^A. We denote by $T_1 \Delta T_2$ the subtree of D_s^A 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^A 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^A. We call $\mathcal{F}$ a connected family of D_s^A.*

Lemma 8 *Let D_s^A be null-free and let $\mathcal{F}$ be a connected family of D_s^A. Then*

1. *For any $T \in \mathcal{F}$, $\mathsf{vec}_\psi(T) \ne (0, \ldots, 0)$.*
2. *For any $T_1, T_2 \in \mathcal{F}$, $\mathsf{vec}_\psi(T_1) \ne \mathsf{vec}_\psi(T_2)$.*

Consequently, $2^\rho - 1 \geq |\mathcal{F}|$ or equivalently $\rho \geq \lceil \log_2(|\mathcal{F}| + 1) \rceil$, where ρ is the number of masks used by A for s invocations.

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^A. We show the existence of a connected family of size $(s-1)$ in D_s^A. For each intermediate node $v \in D_s^A$, let P_v be the path from v to the final node of D_s^A. Define $\mathcal{F} = \{P_v : v \text{ is an intermediate node in } D_s^A\}$. 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 [12] 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.

Lemma 10 *Suppose D_s^A is the full binary tree on $s = 2^t - 1$ nodes. If $t = 2$, there is a connected family of size 3 in D_s^A and for $t \geq 3$, there is a connected family of size $5 \times 2^{t-2} - 2$ in D_s^A. Consequently, $\rho \geq 2$ for $t = 2$ and $\rho \geq t + 1$ for $t \geq 3$.*

5 New Construction

For $t > 0$, let $\mathcal{T}_t$ be the binary tree defined as $\mathcal{T}_t = (V_t = \{P_0, \ldots, P_{2^t-2}\}, A_t)$, where $A_t = \{(P_{2j+1}, P_j), (P_{2j+2}, P_j) : 0 \leq j \leq 2^{t-1} - 2\}$. The underlying digraph for our algorithm is a binary tree with sequential paths terminating at the leaf nodes of the tree. We define a digraph $\mathcal{G}_{t,i}$ which consists of the full binary tree $\mathcal{T}_t$ alongwith a total of i nodes on the sequential paths. The precise definition of $\mathcal{G}_{t,i} = (V_{t,i}, A_{t,i})$ is

$$\left.\begin{array}{rclcl} V_{t,i} & = & V_t & \cup & \{Q_0, \ldots, Q_{i-1}\} \\ A_{t,i} & = & A_t & \cup & \{(Q_j, P_{2^{t-1}+j-1}) : 0 \leq j \leq 2^{t-1} - 1\} \\ & & & \cup & \{(Q_j, Q_{j-2^{t-1}}) : 2^{t-1} \leq j \leq i - 1\}. \end{array}\right\} \qquad (2)$$

The total number of nodes in $\mathcal{G}_{t,i}$ is equal to $2^t - 1 + i$, where $2^t - 1$ nodes are in the binary tree part and i nodes are in the sequential part. We define parameters $r_{t,i}$ and $s_{t,i}$ (or simply r and s) in the following manner: If $i = 0$, then $r = s = 0$; if $i > 0$, then r and s are defined by the equation:

$$i = r2^{t-1} + s \qquad (3)$$

where s is a unique integer from the set $\{1, \ldots, 2^{t-1}\}$. For $i > 0$, we can write $i = (r+1) \times s + (2^{t-1} - s)r$. Thus in $\mathcal{G}_{t,i}$ there are s sequential paths of length $(r+1)$ each and these terminate on the left most s leaf nodes of $\mathcal{T}_t$. There are also $(2^{t-1} - s)$ sequential paths of length r each and these terminate on the other $(2^{t-1} - s)$ leaf nodes of $\mathcal{T}_t$. Figure 1 shows $\mathcal{G}_{4,19}$.

We define $\rho_{t,i}$ or (simply ρ) to be the maximum length (counting only Q nodes) of a path from a Q-node to a P-node. Hence $\rho = 0$ if $i = 0$ and $\rho = r + 1$ if $i > 0$.

When $i = 0$, $\mathcal{G}_{t,i}$ is simply the full binary tree $\mathcal{T}_t$ and when $t = 1$, $\mathcal{G}_{t,i}$ is a dipath of length $r + 1$. These are the two extreme cases – one leading to a full binary tree and the other leading to a single dipath. In practical applications, t will be fixed and there will be "long" dipaths terminating on the leaf nodes of $\mathcal{T}_t$. For implementation purpose, the number of processors required is 2^{t-1}. Hence for practical applications, the value of $t \leq 5$.

Remark: The idea of breaking a message into parts, hashing them independently and finally combining the outputs is present in Damgård [2] in the context of collision resistant hash functions. The current construction can be seen as a development of the "UOWHF version" of this idea.

5.1 Notation

We define a few notation for future reference.

1. t is the number of levels in the binary tree $\mathcal{T}_t$.
2. i is the total number of nodes in the sequential part of the algorithm.
3. r and s are as defined in (3).
4. $\rho = 0$ if $i = 0$ and $\rho = r + 1$ if $i > 0$.
5. $N = 2^t - 1 + i$ is the total number of nodes in $\mathcal{G}_{t,i}$.
6. For $U \in V_{t,i}$, define $\mathsf{nodenum}(U) = j$ if $U = P_j$ and $\mathsf{nodenum}(U) = j + 2^t - 1$ if $U = Q_j$.
7. For $U \in V_{t,i}$, we say that U is a P-node (resp. Q-node) if $U = P_j$ (resp. $U = Q_j$) for some j.

For $U \in V_{t,i}$, we define $\mathsf{indeg}(U)$ (resp. $\mathsf{outdeg}(U)$) to be the indegree (resp. outdegree) of U. Note that other than P_0 each node U has $\mathsf{outdeg}(U) = 1$. Thus for each node $U \neq P_0$ there is a unique out neighbour.

The concept of level is defined in the following manner. There are $L = \rho + t$ levels in $\mathcal{G}_{t,i}$ and the level number of each node is defined as follows.

$$\left.\begin{array}{lcll} \mathsf{level}(P_j) & = & L - 1 - j_1 & \text{if } 2^{j_1} - 1 \leq j \leq 2^{j_1+1} - 2 \text{ and } 0 \leq j_1 \leq t - 1; \\ \mathsf{level}(Q_j) & = & \rho - j_1 - 1 & \text{if } j_1 2^{t-1} \leq j \leq (j_1 + 1)2^{t-1} - 1 \text{ and } 0 \leq j_1 \leq r - 1; \\ \mathsf{level}(Q_j) & = & 0 & \text{if } r2^{t-1} \leq j \leq r2^{t-1} + s. \end{array}\right\} \quad (4)$$

Note that if $\rho = 0$, there are no Q-nodes and hence the level numbers of Q-nodes are not defined. The root node of $\mathcal{T}_t$ has the highest level. Nodes with indegree zero can be at levels zero and one. Let $U \in V_{t,i}$ and $j = \mathsf{nodenum}(U)$: If $0 \leq j \leq 2^{t-1} - 2$ then we define $\mathsf{lchild}(U) = P_{2j+1}$ and $\mathsf{rchild}(U) = P_{2j+2}$; if $2^{t-1} - 1 \leq j \leq N - 1$, then we define predecessor of U in the following manner:

$$\left.\begin{array}{lcll} \mathsf{pred}(U) & = & Q_{j+2^{t-1}} & \text{if } 2^{t-1} - 1 \leq j \leq 2^{t-1} + i - 2; \\ & = & \mathsf{NULL} & \text{if } 2^{t-1} + i - 1 \leq j \leq N - 1; \end{array}\right\} \quad (5)$$

For a node U, $\mathsf{pred}(U) = \mathsf{NULL}$ implies that the indegree of U is zero.

5.2 Mask Assignment Algorithm

There are two disjoint sets of masks $\{\alpha_0, \ldots, \alpha_{l-1}\}$ and $\{\beta_0, \ldots, \beta_{t-2}\}$ where $l = \lceil \log(\rho + t) \rceil$. The mask assignment

$$\psi : A_{t,i} \to \{\alpha_0, \ldots, \alpha_{l-1}\} \cup \{\beta_0, \ldots, \beta_{t-2}\}.$$

is a function from the set of arcs of $\mathcal{G}_{t,i}$ to the set of masks. The definition of ψ is as follows: Let $(U, V) \in A_{t,i}$ with $\mathsf{level}(U) = j - 1$ and $\mathsf{level}(V) = j$ for some $j \in \{1, \ldots, L - 1\}$.

- If ((U is a Q-node) or (U is a P-node and $U = \mathsf{lchild}(V)$)), then $\psi(U, V) = \alpha_{\nu(j)}$.
- If (U is a P-node and $U = \mathsf{rchild}(V)$) then $\psi(U, V) = \beta_{j-(\rho+1)}$.

Here $\nu(j)$ is defined to be the non negative integer j_1 such that $2^{j_1}|j$ and $2^{j_1+1} \not| j$. Also for the convenience of notation we write $\psi(U,V)$ instead of $\psi((U,V))$. The mask assignment for $\mathcal{G}_{4,19}$ is shown in Figure 1.

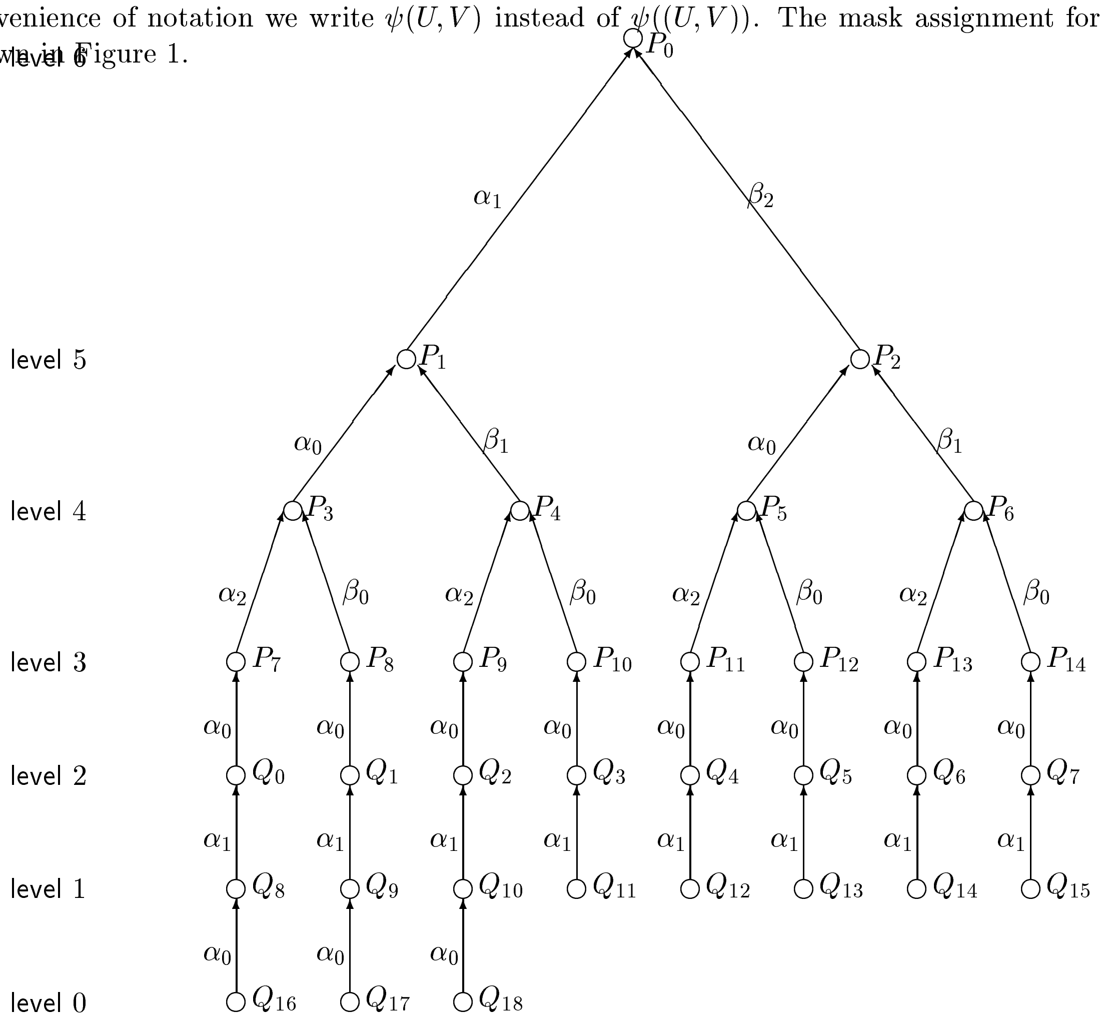

Figure 1: Example of mask assignment for $t = 4$ and $i = 19$.

5.3 Optimality of Mask Assignment

The total number of masks used is equal to $t - 1 + \lceil \log(\rho + t) \rceil$. The total number of nodes in $\mathcal{G}_{t,i}$ is equal to $N = 2^t - 1 + i$. Using the result of [10], at least $\mathcal{L}_{t,i} = \lceil \log(2^t - 1 + i) \rceil$ masks are required by any algorithm in class $\mathcal{C}$ (see Section 3 Our algorithm requires $\mathcal{R}_{t,i} = t - 1 + \lceil \log(\rho + t) \rceil$ masks. Define $\mathcal{D}_{t,i} = \mathcal{R}_{t,i} - \mathcal{L}_{t,i}$. We study $\mathcal{D}_{t,i}$.

Proposition 11

$$\left.\begin{array}{rcll} \mathcal{D}_{t,i} & = & 0 & \textit{if } i = 0 \textit{ and } t = 1; \\ & = & \lceil \log t \rceil - 1 & \textit{if } i = 0 \textit{ and } t > 1; \\ & = & \lceil \log(r + 1 + t) \rceil - \left\lceil \log \left(r + 2 + \frac{s-1}{2^{t-1}} \right) \right\rceil & \textit{if } i > 0. \end{array}\right\} \tag{6}$$

Furthermore, $\mathcal{D}_{t,i} = 0$ if and only if either $t = 1$; or ($t = 2$ and $i = 0$); or $2^j - 1 - \lceil (s-1)/2^{t-1} \rceil \leq r \leq 2^{j+1} - t - 1$ for some $j > 0$.

For $t = 1$, the mask assignment algorithm reduces to the mask assignment algorithm of Shoup [12] and for $i = 0$, the mask assignment algorithm reduces to the mask assignment algorithm of Sarkar [11]. Hence we concentrate on the case $t > 1$ and $i > 0$. For practical parallel implementation, the value of t will determine the number of processors and will be fixed whereas the value of i can grow in an unbounded manner.

Suppose $2^{\tau-1} < t-1 \leq 2^{\tau}$. For $j \geq 0$, define two intervals of integers in the following manner:

$$\left.\begin{array}{rcl} I_j & = & \{2^{\tau+j} - 1 - \left\lceil \frac{s-1}{2^{t-1}} \right\rceil, 2^{\tau+j} - \left\lceil \frac{s-1}{2^{t-1}} \right\rceil, \ldots, 2^{\tau+j+1} - t - 1\}; \\ J_j & = & \{2^{\tau+j+1} - t, 2^{\tau+j+1} - t + 1, \ldots, 2^{\tau+j+1} - 2 - \left\lceil \frac{s-1}{2^{t-1}} \right\rceil\}. \end{array}\right\} \qquad (7)$$

Clearly, $|I_j| = 2^{\tau+j} - t + 1 + \lceil (s-1)/2^{t-1} \rceil$, $|J_j| = t - 1 - \lceil (s-1)/2^{t-1} \rceil$ and $|I_j| + |J_j| = 2^{\tau+j}$.

Theorem 12 *Suppose $i > 0$, $2^{\tau-1} < t-1 \leq 2^{\tau}$ and for $j \geq 0$, I_j and J_j are as defined in 7. Then $\mathcal{D}_{t,i} = 0$ if $r \in I_j$; and $\mathcal{D}_{t,i} = 1$ if $r \in J_j$.*

From Theorem 12, it follows that for $j \geq \tau$, in any interval $2^j + 1 \leq r \leq 2^{j+1}$, there are exactly $t - 1 - \lceil (s-1)/2^{t-1} \rceil$ points where the algorithm is suboptimal with respect to the lower bound. Moreover, at these points it requires exactly one extra mask over the lower bound. In any practical parallel implementation, the value of t will be fixed, whereas the value of r will grow. In such a situation, the ratio $\frac{1}{2^j}(t-1-\lceil (s-1)/2^{t-1} \rceil)$ approaches zero very fast and hence we can say that for $t \geq 2$, the algorithm achieves optimal key length expansion *almost everywhere*. (Note that for $t = 1$, the algorithm reduces to Shoup's algorithm and hence achieves optimal key length expansion.) An example of the behaviour of $\mathcal{D}_{t,i}$ is given in the Appendix (Section C.1).

5.4 Computation of Message Digest

Let $\{h_k\}_{k\in\mathcal{K}}$, where each h_k is an (n,m) function, be the compression UOWHF whose domain is to be extended. *For $t > 1$, we require $n \geq 2m$.* The nodes of $\mathcal{G}_{t,i}$ represent the invocations of h_k. Thus h_k is invoked a total of N times. The output of P_0 is provided as output digest, whereas the outputs of all the other nodes are used as inputs to other invocations as defined by the arcs of $\mathcal{G}_{t,i}$. Using Proposition 1 of [10] we obtain the following: Suppose a message x is hashed using $\mathcal{G}_{t,i}$ and the compression UOWHF $\{h_k\}_{k\in\mathcal{K}}$, where each h_k is an (n,m) function. Then $|x| = N(n-m)+m$.

Thus the compression UOWHF $\{h_k\}_{k\in\mathcal{K}}$ is extended to a UOWHF $\{H_p^{(t,i)}\}_{p\in\mathcal{P}}$ where each $H_p^{(t,i)}$ is an $(N(n-m)+m, m)$ function and $p = k||\alpha_0||\ldots||\alpha_{l-1}||\beta_0||\ldots||\beta_{t-2}$. The message x of length $N(n-m)+m$ has to be formatted into small substrings and provided as input to the different invocations of h_k. Write $x = x_0||\ldots||x_{N-1}$, where the lengths of the x_j's are as follows.

$$\left.\begin{array}{rcll} |x_j| & = & n - 2m & \text{if } 0 \leq j \leq 2^{t-1} - 2; \\ & = & n - m & \text{if } 2^{t-1} - 1 \leq j \leq 2^{t-1} + i - 2; \\ & = & n & \text{if } 2^{t-1} + i - 1 \leq j \leq 2^{t} + i - 2. \end{array}\right\} \qquad (8)$$

The substring x_j is provided as input to node U with $\mathsf{nodenum}(U) = j$ and the m-bit output of U is denoted by z_j. The outputs $z_1, \ldots, z_{N-1}$ are masked using the α and β masks to obtain m-bit strings $y_1, \ldots, y_{N-1}$ in the following manner.

$$y_j = z_j \oplus \psi(U,V) \quad \text{if } \mathsf{nodenum}(U) = j. \qquad (9)$$

The inputs to the invocations of h_k are formed from the x's and the y's in the following manner. There are N invocations whose inputs are denoted by $w_0, \ldots, w_{N-1}$ and are defined as follows.

$$\left.\begin{array}{rcll} w_j & = & x_j||y_{2j+1}||y_{2j+2} & \text{if } 1 \leq j \leq 2^{t-1}-2; \\ & = & x_j||y_{j+2^{t-1}} & \text{if } r > 0 \text{ and } 2^{t-1}-1 \leq j \leq 2^{t-1}+i-2; \\ & = & x_j & \text{if } 2^{t-1}+i-1 \leq j \leq 2^t+i-2. \end{array}\right\} \quad (10)$$

Note that the length of each w_j is n and hence we can invoke h_k on w_j for all $j \in \{0, \ldots, N-1\}$. For any node $U \in V_{t,i}$ we define $x(U), y(U)$ and $w(U)$ to be the x, y and w strings associated to the node U as defined respectively in (8), (9) and (10). Similarly the output of node U will be denoted by $z(U)$.

Now we are ready to describe the digest computation algorithm. Most of the work has already been done, so that the description of the algorithm becomes simple. Suppose the compression UOWHF is $\{h_k\}_{k\in\mathcal{K}}$. We describe the digest computation of $H_p^{(t,i)}(x)$.

Algorithm to compute $H^{(t,i)}(x)$

```
1. for $j = 0$ to $L - 1$ do
2.       for all $U$ with $\mathsf{level}(U) = j$ do in parallel
3.           compute $z(U) = h_k(w(U))$;
4.       end do;
5. end do;
6. return $z_0$.
```

The security reduction for the construction is given in the Appendix B. Also the Appendix contains further discussion on the construction.

6 Comparison to Known Constructions

In Table 1, we present the parameters of earlier constructions along with the parameters of our construction. The efficacy of a construction can be judged from two aspects – key expansion and parallelism. We discuss these features below.

Key expansion : Constructions in [12] and [3] provide optimal key expansion. Among binary tree algorithms, the construction of [11] improves upon the construction of [1] for $T \geq 3$, while the construction of [7] improves upon the construction of [11] for $T \geq 15$. The construction of [10] uses a constant number of extra masks over the optimal and hence is better than [1], [11] and [7] even at the points where they are defined (for sufficiently large T). The construction in the current paper achieves optimal key expansion almost everywhere and hence is better than [10] and consequently better than [1], [11] and [7] at the points where they are defined (for sufficiently large T). On the other hand, when the number of invocations is of the form 2^T, the current construction requires one extra mask over the known lower bound, while the construction in [3] achieves the lower bound.

Parallelism : The construction in [12] is sequential whereas all the other constructions offer some degree of parallelism. One problem with the constructions in [1, 11, 3, 7] is that they use an unbounded tree structure. The constructions in [1, 11, 7] use a binary tree structure and it is possible to simulate this tree using a finite number of processors. However, the construction of [3] uses a very irregular tree structure and the degree of parallelism obtained is quite poor. On the other hand, the construction in [10] and the current paper uses a finite tree which leads to an asymptotically optimal speed-up.

Table 1: Comparison among different constructions. Note that the parameter t in Sarkar [10] and this paper is a constant and independent of the number of invocations.

parameter	BR-XTH [1]	Shoup [12]	Sarkar [11]	Lee et al [3]	Nandi [7]
# invocations	$2^T - 1$	j	$2^T - 1$	2^T	$2^T - 1$
# processors	$2^T - 1$	1	$2^T - 1$		$2^T - 1$
# masks	$2(T-1)$	$\lceil \log j \rceil$	$T + \lfloor \log(T-1) \rfloor$	T	$T + O(\log^* T)$
# rounds	T	j	T	$l2^{T/l} - l + 1$ $T \equiv 0 \bmod l$	T
speed-up	$\frac{2^T-1}{T}$	1	$\frac{2^T-1}{T}$	$\approx \frac{2^{T-\frac{T}{l}}}{l}$	$\frac{2^T-1}{T}$
optimality of key expansion	$(T-2)$ over opt.	optimal	$\lfloor \log(T-1) \rfloor$ over opt.	opt.	$O(\log^* T)$ over opt.

parameter	Sarkar [10]	This paper
# invocations	$(q+2)2^t + 2b - 1$, $q \geq 0$, $1 \leq b \leq 2^{t-1}$	$2^t + i - 1$, $i \geq 0$
# processors	2^t	2^{t-1}
# masks	$t + \lceil \log(q+t+2) \rceil + \lceil \log t \rceil$	$t - 1 + \lceil \log(\rho + t) \rceil$
# rounds	$q + t + 2$	$\rho + t$
speed-up	$\approx 2^t$ for large q	$\approx 2^{t-1}$ for large i
optimality of key expansion	constant number over optimal	almost everywhere optimal

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] W. Lee, D. Chang, S. Lee, S. Sung and M. Nandi. New Parallel Domain Extenders for UOWHF. *Proceedings of Asiacrypt 2003*, to appear.

[4] R. Impagliazzo and M. Naor. Efficient Cryptographic Schemes provably as secure as subset sum. *Journal of Cryptology*, Vol. 9, No. 4, 1996.

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

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

[7] M. Nandi. A New Tree based Domain Extension of UOWHF, Cryptology e-print archive, Report No. http://eprint.iacr.org, 2003/142.

[8] 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.

[9] J. Rompel. One-way functions are necessary and sufficient for digital signatures. *Proceedings of the 22nd Annual Symposium on Theory of Computing*, ACM, 1990.

[10] P. Sarkar. Domain Extenders for UOWHFs: A Generic Lower Bound on Key Expansion and a Finite Binary Tree Algorithm, Cryptology e-print archive, http://eprint.iacr.org, Report No. 2003/009.

[11] P. Sarkar. Construction of UOWHF: Tree Hashing Revisited, Cryptology e-print archive, http://eprint.iacr.org, Report No. 2002/058.

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

[13] 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.

[14] D. R. Stinson. Some observations on the theory of cryptographic hash functions.

`http://www.cacr.math.uwaterloo.ca/~dstinson/papers/newhash.ps.`

A Proofs

Proof of Proposition 1: 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 $n + (n-m)(s-1)$. ∎

Proof of Proposition 3: 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 $n \geq m\delta$. ∎

Proof of Proposition 4: Let $\mathcal{A}$ be an (ϵ, q)-strategy for $\mathcal{G}(\mathbf{F})$. We construct a strategy $\mathcal{B}$ for $\mathcal{G}(\mathbf{F}')$ as follows. $\mathcal{B}^{\text{guess}}$ invokes $\mathcal{A}^{\text{guess}}$ and returns the (x, state) pair returned by $\mathcal{A}^{\text{guess}}$. Now $\mathcal{B}$ is given $k \in_r \mathcal{K}$. $\mathcal{B}^{\text{find}}(s, k, \mathsf{state})$ invokes $\mathcal{A}^{\text{find}}(s, k, \mathsf{state})$. If $\mathcal{A}$ fails, then $\mathcal{B}$ fails. Suppose $\mathcal{A}$ succeeds and let $x' \neq x$ be the string returned by $\mathcal{A}^{\text{find}}(x, k, \mathsf{state})$. If $\phi(x', k) = \mathsf{T}$, then $\mathcal{B}$ returns failure, else $\mathcal{B}$ returns x'. The probability that $\phi(x', k) = \mathsf{T}$ is $\frac{1}{2^K}$. If $\phi(x', k) = \mathsf{F}$, then x' provides a collision for $\mathbf{F}'$. Hence $\mathcal{B}$ is an $(\epsilon - \frac{1}{2^K}, q)$-strategy for $\mathcal{G}(\mathbf{F}')$. ∎

Proof of Lemma 5: Since D_s^A is not null-free, there is a non-trivial subtree T of D_s^A such that $\mathsf{vec}_\psi(T) = (0, \ldots, 0)$. As a consequence, we have $\psi(T) = \bigoplus_{e \in E} \psi(e) = 0^m$. 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. $\mathsf{indeg}_{D_s^A}(v)$) the indegree of v in T (resp. D_s^A).

Suppose $\mathsf{indeg}_{D_s^A}(v) = t \leq \delta$ 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 $\mathsf{Nbr}_{D_s^A}(v) = (v_{j_1}, \ldots, v_{j_t}, 0, \ldots, 0)$. If $v \in V$, we define $\mathsf{Nbr}_T(v)$ as follows. For $1 \leq i \leq \delta$, $\mathsf{Nbr}_T(v)[i] = \mathsf{Nbr}_{D_s^A}(v)[i]$ if $\mathsf{Nbr}_{D_s^A}(v)[i] \in V$; 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 $\mathsf{indeg}_{D_s^A}(v) = t \leq \delta$. 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^A 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^A 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^A.

Given a message x and a node v of D_s^A, 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 $t = \mathsf{indeg}_{D_s^A}(v)$. 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 $v \in D_s^A \setminus T$: $\chi(v, x) = \chi(v, x') =$ any $(n - mt)$-bit string.

From the first point, it is clear that $x \neq x'$. We show that $H_p^{(A,s)}(x) = H_p^{(A,s)}(x')$ for any $H_p^{(A,s)} \in \mathbf{F}^{(A,s)}$. 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^A, then this is clearly true. If v_α is an internal node of D_s^A, 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 $z_{m'+1} \ldots z_{m'+K}$.

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 $\phi(z, k) = \top = \phi(z', k)$. Thus v_α outputs 1^m for both x and x'. This gives us the required result. ∎

Proof of Lemma 8: 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^A. 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., $\mathsf{vec}_\psi(T_1) \neq \mathsf{vec}_\psi(T_2)$. ∎

Proof of Lemma 10: For the purpose of the proof, we consider the nodes of the binary tree to be numbered by $1, 2, \ldots, 2^t - 1$ and the arcs are of the form $(i, \lfloor i/2 \rfloor)$ for $2 \leq i \leq 2^t - 1$. 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 G_1 and G_2 be the full binary trees rooted at nodes 2 and 3. Then G_1 and G_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 G_1 and G_2 respectively. Let $\mathcal{G}_1'$ be the family obtained from $\mathcal{G}_1$ by adding the arc $S_1 = (2, 1)$ to each subtree in $\mathcal{G}_1$. Similarly let $\mathcal{G}_2'$ be the family obtained from $\mathcal{G}_2$ by adding the arc $S_2 = (3, 1)$ 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 $N_t = 5 \times 2^{t-2} - 2 \geq 2^t$ for $t \geq 3$. Hence by Lemma 8 we have $\rho \geq 2$ for $t = 2$ and $\rho \geq t + 1$ for $t \geq 3$. ∎

Proof of Proposition 6: The first part of the result follows from the definition. If $t = 1$ and $i = 0$, then clearly $\mathcal{D}_{t,i} = 0$. On the other hand, if $t = 1$ and $i > 0$, then $s = 1$ and so $\mathcal{D}_{t,i} = \lceil \log(r + 2) \rceil - \lceil \log(r + 2 + (s - 1)) \rceil = 0$. For $t = 2$ and $i = 0$, $\mathcal{D}_{2,0} = \log 2 - 1 = 0$. If $i > 0$, then $\mathcal{D}_{t,i} = 0$ if and only if $2^j < r + 2 + (s - 1)/2^{t-1} \leq r + t + 1 \leq 2^{j+1}$ for some j. This condition is equivalent to the condition $2^j - 1 - \lceil (s - 1)/2^{t-1} \rceil \leq r \leq 2^{j+1} - t - 1$. In all other cases $\mathcal{D}_{t,i} > 0$. ∎

Proof of Theorem 12: If $r \in I_j$, then from Proposition 11, we have $\mathcal{D}_{t,i} = 0$. If $r \in J_j$, then

$$2^{\tau+j+1} - t \leq r \leq 2^{\tau+j+1} - 2 - \left\lceil \frac{s-1}{2^{t-1}} \right\rceil$$

which shows that

$$\begin{aligned} 2^{\tau+j+1} < r + t + 1 \leq 2^{\tau+j+1} + t - 1 - \left\lceil \frac{s-1}{2^{t-1}} \right\rceil \text{ and} \\ 2^{\tau+j+1} - t + 2 + \frac{s-1}{2^{t-1}} \leq r + 2 + \frac{s-1}{2^{t-1}} \leq 2^{\tau+j+1}. \end{aligned}$$

If $t = 2$ and $s > 1$, then $J_j = \emptyset$ for all $j \geq 0$. In the other cases, using $t \leq 2^\tau$, we have for all $j \geq 0$,

$$2^{\tau+j+1} + t - 1 - \left\lceil \frac{s-1}{2^{t-1}} \right\rceil \leq 2^{\tau+j+2} \text{ and } 2^{\tau+j} < 2^{\tau+j+1} - t + 2 + \frac{s-1}{2^{t-1}}.$$

Hence $\lceil \log(r + t + 1) \rceil = \tau + j + 2$ and $\lceil \log(r + 2 + (s - 1)/2^{t-1}) = \tau + j + 1$ which shows $\mathcal{D}_{t,i} = 1$. ∎

B Security Reduction

In this section, we prove that our algorithm preserves the UOWHF property.

Theorem 13 *If there is an (ϵ, η) winning strategy $\mathcal{A}$ for $\{H_p^{(t,i)}\}_{p \in \mathcal{P}}$, then there is an $(\frac{\epsilon}{N}, \eta + 2N)$ winning strategy $\mathcal{B}$ for $\{h_k\}_{k \in \mathcal{K}}$. Consequently, if $\{h_k\}_{k \in \mathcal{K}}$ is a UOWHF, then $\{H_p^{(t,i)}\}_{p \in \mathcal{P}}$ is also a UOWHF.*

Proof. We construct the strategy $\mathcal{B}$ from the strategy $\mathcal{A}$. We will assume that the values of t and i are globally available.

Algorithm $\mathcal{B}^{\mathbf{guess}}$

```
1. Invoke $\mathcal{A}^{\text{guess}}$ to obtain $(x, \mathsf{state})$ where $|x| = N(n-m)+m$.
2. Write $x = x_0||x_1||\ldots||x_{N-1}$ where $x_j$s are as defined in (8).
3. Choose $j \in_r \{0,\ldots,N-1\}$.
4. If $|x_j| = n$, then set $y = \boldsymbol{\lambda}$ else choose $y \in_r \{0,1\}^{n-|x_j|}$.
5. Output $(u = x_j||y, (\mathsf{state}, j, y))$.
```

At this point $\mathcal{B}$ is given a $k \in_r \mathcal{K}$ and we describe the find stage of $\mathcal{B}$.

Algorithm $\mathcal{B}^{\mathbf{find}}(u, k, (\mathsf{state}, j, y))$

```
1. Invoke $\mathsf{MDef}(y, k, j)$ to obtain the key $p$ for the family $\{H_p^{(t,i)}\}_{p\in\mathcal{P}}$.
2. Invoke $\mathcal{A}^{\text{find}}(x, p, \mathsf{state})$ to obtain $x'$.
3. If $u = w_j \neq w'_j$ and $z_j = z'_j$, then return $(w_j, w'_j)$.
4. Else return "failure".
```

In $\mathcal{B}^{\text{find}}(u, k, (\mathsf{state}, j, y))$ above and in the rest of the proof the primed variables denote the quantities corresponding to x'. First assume that the following two conditions hold.

C: the string u to which $\mathcal{B}$ commits in the guess stage is equal to w_j.
D: the key p returned by $\mathsf{MDef}(u, k, j)$ is chosen uniformly at random from $\mathcal{P}$.

We have the following claim.

Claim: If $x \neq x'$ and $H_p^{(t,i)}(x) = H_p^{(t,i)}(x')$, then there will be a $j_1 \in \{0,\ldots,N-1\}$ such that $w_{j_1} \neq w'_{j_1}$ and $h_k(w_{j_1}) = z_{j_1} = z'_{j_1} = h_k(w'_{j_1})$.

(Proof of claim): The proof is by backward induction on the levels of the nodes. Since $H_p^{(t,i)}(x) = H_p^{(t,i)}(x')$, the outputs of the node P_0 (at level $L-1$) for both x and x' are equal. If the corresponding inputs are not equal then we have a collision for h_k; if the inputs are equal, then this implies that the outputs of the nodes at level $L-2$ are the same for x and x'. Again applying the above argument, we see that either there is a collision for h_k or the outputs of the nodes in level $L-3$ are the same for both x and x'. Proceeding in this way, we ultimately obtain that either there is a collision for h_k or the inputs to all the nodes in $\mathcal{G}_{t,i}$ are same for both x and x'. As a consequence we have $x = x'$. Thus if we have $x \neq x'$, then there must be a collision for h_k at some node of $\mathcal{G}_{t,i}$. ■ (claim)

The probability that this j_1 is equal to the choice of j in $\mathcal{B}^{\text{guess}}$ is $1/N$. Hence if $\mathcal{A}$ has success probability at least ϵ, then $\mathcal{B}$ has success probability at least ϵ/N. Strategy $\mathcal{A}$ invokes some function from the family $\{h_k\}_{k\in\mathcal{K}}$ at most η times. Additionally strategy $\mathcal{B}$ needs to invoke h_k several times to compute z_j and z'_j and also in the mask definition algorithm $\mathsf{MDef}()$ (see below). The total number of such invocations is at most $2N$. Hence the total number of invocations is at most $\eta + 2N$.

To complete the proof, it is now enough to show that **C** and **D** hold. To do this, we have to describe algorithm $\mathsf{MDef}(y, k, j)$, which we do below. Let U be such that $\mathsf{nodenum}(U) = j$. If $\mathsf{indeg}(U) = 0$, then we define all the masks randomly. Otherwise, we follow the algorithm described below. Let $\mathsf{level}(U) = d$ and define an array $\mathsf{node}[\,]$ and the variable last in the following manner.

```
1. $\mathsf{tmp} = d$; $V = U$;
```

```
2. while ($V \neq \mathsf{NULL}$) do
3.     $\mathsf{node}[\mathsf{tmp}] = V$;
4.     if (($V$ is a $P$-node) and ($\mathsf{level}(V) > \rho$)) then $V = \mathsf{lchild}(V)$;
5.     else $V = \mathsf{pred}(V)$;
6.     $\mathsf{tmp} = \mathsf{tmp} - 1$;
7. end do;
8. $\mathsf{last} = \mathsf{tmp} + 1$.
```

The variable last is the level number of the last node in the reverse path starting at U and whose nodes are stored in the array $\mathsf{node}[\,]$. The value of last is either 0 or 1. We now present the rest of the mask defining algorithm. We require an array $\mathsf{arr}[\,]$ which stores elements of the form (V, v), where V is a node of $\mathcal{G}_{t,i}$ and v is an m-bit string. The array $\mathsf{arr}[\,]$ is initialized to the empty array.

In the first part of the algorithm we define the α masks and then we define the β masks. Note that the algorithm starts with $d = \mathsf{level}(U)$, where U is such that $\mathsf{nodenum}(U) = j$. Also recall from Section 5.4 that for any node $U \in V_{t,i}$, by $w(U), x(U), y(U)$ and $z(U)$ we denote the w, x, y and z strings associated with U.

```
1.  if ($|y| = 2m$)
2.      write $y = y_1||y_2$, where $|y_1| = |y_2| = m$;
3.      append $(\mathsf{rchild}(U), y_2)$ to $\mathsf{arr}[\,]$;
4.      set $y = y_1$;
5.  end if;
6.  while ($d > \mathsf{last}$) do
7.      $\mathsf{tmp} = \max(d - 2^{\nu(d)}, \mathsf{last})$;
8.      $V = \mathsf{node}[\mathsf{tmp}]$;
9.      if (($V$ is a $P$-node) and ($\mathsf{tmp} > \rho$)) then
10.         choose $v_{\mathsf{old}}, v_1 \in_r \{0,1\}^m$ and set $v = v_{\mathsf{old}}||v_1$;
11.         append $(\mathsf{rchild}(V), v_1)$ to $\mathsf{arr}[\,]$;
12.     else if ($\mathsf{tmp} > \mathsf{last}$) then choose $v = v_{\mathsf{old}} \in_r \{0,1\}^m$;
13.     else set $v = \boldsymbol{\lambda}$;
14.     if $\alpha_{\nu(\mathsf{tmp}+1)}$ is undefined, then define it randomly;
15.     set $y(V) = h_k(x(V)||v) \oplus \alpha_{\nu(\mathsf{tmp}+1)}$.
16.     for $d_1 = \mathsf{tmp} + 1$ to $d - 2$ do
17.         let $V = \mathsf{node}(d_1)$;
18.         if (($V$ is a $P$-node) and ($d_1 > \rho$))
19.             choose $v_1 \in_r \{0,1\}^m$ and set $v = y(\mathsf{node}[d_1 - 1]||v_1$;
20.             append $(\mathsf{rchild}(V), v_1)$ to $\mathsf{arr}[\,]$;
21.         else set $v = y(\mathsf{node}[d_1 - 1])$;
22.         if $\alpha_{\nu(d_1+1)}$ is undefined, then define it randomly;
23.         set $y(V) = h_k(x(V)||v) \oplus \alpha_{\nu(d_1+1)}$.
24.     end do;
25.     set $V = \mathsf{node}(d - 1)$;
26.     if (($V$ is a $P$-node) and ($d - 1 > \rho$))
27.         choose $v_1 \in_r \{0,1\}^m$ and set $v = y(\mathsf{node}[d - 2])||v_1$;
28.         append $(\mathsf{rchild}(V), v_1)$ to $\mathsf{arr}[\,]$;
29.     else set $v = y(\mathsf{node}[d - 2])$;
30.     set $\alpha_{\nu(d)} = y \oplus h_k(x(V)||v)$;
31.     $y = v_{\mathsf{old}}$; $d = \mathsf{tmp}$;
```

32. end do.

At this stage, some or all of the α masks have been defined. Randomly define all as yet undefined α masks. We now describe the definition of the β masks. Let $\mathsf{arr}[\,]$ be $[(V_1, v_1), \ldots, (V_\kappa, v_\kappa)]$ which is sorted in ascending order on the level number of the first component of each entry. The construction of $\mathsf{arr}[\,]$ ensures that $\mathsf{level}(V_{j_1}) \neq \mathsf{level}(V_{j_2})$ for $j_1 \neq j_2$.

```
1. for $j_1 = 1$ to $\kappa$ do
2.     compute $z = z(V_{j_1})$;
3.     set $\beta_{\mathsf{level}(V_{j_1})-\rho} = z \oplus v_{j_1}$;
4. end do.
```

Randomly define all as yet β masks. Note that in Step 2, it is possible to compute $z(V_{j_1})$ since at this stage all α masks have been defined and all required β masks have also been defined.

The mask definition algorithm (described above) satisfies **C** and **D**. **D** holds since each m-bit mask is either chosen randomly or as XOR between an m-bit string and an m-bit random string. **C** holds since the mask(s) used in the input to U are defined as XOR of the random m-bit string(s) and the output(s) of the previous invocation(s) of h_k. This compeletes the proof. ■

C Further Properties of the New Construction

We briefly discuss several other aspects of the new construction.

C.1 Example to Illustrate the Behaviour of $\mathcal{D}_{t,i}$

We present some values of $\mathcal{D}_{t,i}$ to give an idea of its behaviour. We consider $0 \leq r \leq 31$. For each value of r in this range there are two cases to consider: (a) $s = 1$ and (b) $s > 1$. For a fixed t and either case (a) or case (b) the value of $\mathcal{D}_{t,i}$ for the 32 different values of r can be represented by a 32-tuple. We present some examples below.

1. $t = 2$ and $s = 1$: $\mathcal{D}_{t,i} = (1,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0)$ for $16 \leq r \leq 31$.
2. $t = 2$ and $s > 1$: $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$ for $16 \leq r \leq 31$.
3. $t = 3$ and $s = 1$: $\mathcal{D}_{t,i} = (1,1,1,0,1,1,0,0,0,0,0,0,0,1,1,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0)$ for $16 \leq r \leq 31$.
4. $t = 3$ and $s > 1$: $\mathcal{D}_{t,i} = (0,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0)$ for $16 \leq r \leq 31$.
5. $t = 4$ and $s = 1$: $\mathcal{D}_{t,i} = (2,1,1,0,1,1,1,0,0,0,0,0,1,1,1,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0)$ for $16 \leq r \leq 31$.
6. $t = 4$ and $s > 1$: $\mathcal{D}_{t,i} = (1,1,0,0,1,1,0,0,0,0,0,0,1,1,0,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0)$ for $16 \leq r \leq 31$.
7. $t = 5$ and $s = 1$: $\mathcal{D}_{t,i} = (2,1,1,1,1,1,1,0,0,0,0,1,1,1,1,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0)$ for $16 \leq r \leq 31$.
8. $t = 5$ and $s > 1$: $\mathcal{D}_{t,i} = (1,1,0,1,1,1,0,0,0,0,0,1,1,1,0,0)$ for $0 \leq r \leq 15$.
   $\mathcal{D}_{t,i} = (0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0)$ for $16 \leq r \leq 31$.

The values clearly show the pattern of behaviour predicted by Theorem 12. As the value of r grows larger, the predominance of zeros in the distribution of $\mathcal{D}_{t,i}$ becomes much more marked, which indicates the almost-everywhere-optimal feature of the mask assignment algorithm.

C.2 Efficiency of Parallelism

The digest computation algorithm clearly requires $L = \rho + t$ parallel rounds when 2^{t-1} processors are available. Recall that $\rho = 0$ if $i = 0$ and $\rho = r + 1$, when $0 < i = r2^{t-1} + s$ for some unique $s \in \{1, \ldots, 2^{t-1}\}$. If $2^{t-1}|i$, then $r = (i/2^{t-1}) - 1$ and if $2^{t-1} \not| i$, then $r = \lfloor i/2^{t-1} \rfloor$. Since $\rho = r + 1$, we have $\rho = \lceil i/2^{t-1} \rceil$. The total number of invocations is $2^t - 1 + i$ and hence the time taken by any sequential algorithm will be proportional to $2^t - 1 + i$. Thus the speed-up obtained by our algorithm is equal to $(2^t - 1 + i)/(t + \lceil i/2^{t-1} \rceil)$. For fixed t, and sufficiently large i, this ratio approaches 2^{t-1}. Hence the speed-up obtained by our algorithm is asymptotically optimal.

C.3 Less Number of Invocations

Suppose that we have 2^{t-1} processors at our disposal. Then we can use the structure $\mathcal{G}_{t,i}$ to hash the message. However, using $\mathcal{G}_{t,i}$ implies that we make exactly $2^t + i - 1$ invocations of the compression UOWHF. We now describe how to tackle the situation where the required number of invocations Γ is less than $2^t - 1$. Let $t' \leq t$ be the largest integer such that $2^{t'} - 1 \leq \Gamma$ and let $i' = \Gamma - 2^{t'} + 1$. We use the digraph $\mathcal{G}_{t',i'}$ to compute the digest. Using this procedure allows us to extend the domain for any $\Gamma > 1$.

C.4 Variable Length Messages

The domain extender defined so far extends the domain from fixed length short strings to fixed length long strings. For practical applications, it will be required to handle variable length strings. Thus we extend the domain to be the set of all strings of lengths at most equal to $2^{n-m} - 1$. There are two steps to doing this.

The first step is to "close the gaps". If the number of invocations is Γ, then the length of the message is $\Gamma(n - m) + m$ and if the number of invocation is $\Gamma + 1$, then the length of the message is $(\Gamma + 1)(n - m) + m$. Thus there is gap of $(n - m)$ bits between these two values. For any message whose length falls in this gap, we pad it by zeros to $(\Gamma + 1)(n - m) + m$. Unfortunately, this padding means that it is now easy to find two different length messages whose hash value is same. To tackle this, we need the second step.

The second step is to mask the final digest using a new mask, concatenate the $(n - m)$-bit binary representation of the length of the original message to it and apply h_k to the resulting n-bit string. This approach also provides a UOWHF. We will provide details of this approach in the full version of the paper.