Extending UOWHF Using a Finite Binary Tree

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

Abstract

We present an algorithm to securely extend the domain of a UOWHF. The algorithm uses a *finite* binary tree of processors and achieves the following: (a) For (moderately) long messages the speed-up attained is roughly equal to the number of processors used; (b) the key length expansion is a constant number of bits more than the best known algorithm, which is the sequential Shoup algorithm. In previous work on binary tree based extension of the domain of a UOWHF, the number of processors increased with the length of the message; the speed-up attained was a logarithmic fraction of the total number of processors and the amount of key length expansion over the sequential algorithm increased with the length of the message.

Keywords : UOWHF, hash function, finite 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. Later Mironov [4] has shown that the key length expansion in the Shoup algorithm is the minimum possible for any Merkle-Damgård type algorithm to be correct.

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 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 $2m(T-1)$ for $T > 2$. In a recent work [7], a binary tree based construction has been presented which makes a key length expansion of $m(T + \lfloor\log(T-1)\rfloor)$ bits. Further, it has been shown in [7] that a key length expansion of $m(T+1)$ bits is necessary for a such a binary tree based algorithm to be correct. We note that for a message of length L, the key length expansion required by the Shoup construction is mT. *Thus the sequential Shoup construction makes the least amount of key length expansion among all the known algorithms (sequential or parallel).*

The speed-up of the algorithms in [1, 7] over the sequential algorithm is by a factor of $\frac{2^T}{T}$. Thus even though the key length expansion in the parallel algorithm [7] is more than the key length expansion in the sequential algorithm [9], the computation of the message digest is significantly faster. However, there are several drawbacks of the currently known binary tree algorithms for extending the domain of a UOWHF.

1. The number of processors used grows with the length of the message. This is clearly an undesirable situation for possible practical applications.

2. The speed-up obtained by binary tree algorithm is approximately $\frac{1}{T}$ times the number of processors $2^T - 1$. We would actually like to attain a speed-up which is equal to the number of processors used.

3. The key length expansion for parallel algorithms is significantly more than that for the sequential construction. Let $E_{bt}(L)$ and $E_{seq}(L)$ be the key length expansions made by the algorithms in [7] and [9] respectively. Then $E_{bt}(L) - E_{seq}(L) = m \times \lfloor\log(T-1)\rfloor$. The parameter T and hence the difference $E_{bt}(L) - E_{seq}(L)$ increases with the length of the message.

The purpose of the present paper is to solve the above mentioned problems. More specifically we obtain the following results.

1. We present a binary tree based algorithm to properly extend the domain of a UOWHF using a *finite* binary tree of 2^t processors, where t is independent of the message length.

2. For (moderately) long messages,

   - the speed-up obtained by our algorithm is roughly equal to the number of processors used;
   - the key length expansion made by our algorithm is at most $m(2 + \lfloor\log(t-1)\rfloor)$ bits more than the key length expansion made by the Shoup algorithm. Since t is independent of the message length, the algorithm requires only a constant number of extra bits compared to the sequential algorithm.

We combine several existing techniques to obtain our results. Our finite binary tree algorithm is based on the finite binary tree algorithm used in [8] to extend the domain of a collision resistant hash function. The assignment of "masks" is a combination of the assignment of masks for the binary tree algorithm of [7] and the sequential algorithm of [9].

2 Preliminaries

Let $\{h_k\}_{k\in\mathcal{K}}$ be a keyed family of hash functions, where each $h_k : \{0,1\}^n \rightarrow \{0,1\}^m$. *In this paper we require* $n \geq 2m$. Consider the following adversarial game.

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

We say that $\{h_k\}_{k\in\mathcal{K}}$ is a universal one way hash family (UOWHF) if the adversary has a negligible probability of success with respect to any probabilistic polynomial time strategy. A strategy $\mathcal{A}$ for the adversary runs in two stages. In the first stage $\mathcal{A}^{\text{guess}}$, the adversary finds the x to which he has to commit in Step 1. It also produces some auxiliary state information s. In the second stage $\mathcal{A}^{\text{find}}(x,k,s)$, the adversary either finds a $y \neq x$ such that $h_k(x) = h_k(y)$ or it reports failure. Both $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x,k,s)$ are probabilistic algorithms. The success probability of the strategy is measured over the random choices made by $\mathcal{A}^{\text{guess}}$ and $\mathcal{A}^{\text{find}}(x,k,s)$ and the random choice of k in Step 2 of the game. We say that $\mathcal{A}$ is an (ϵ, a)-strategy if the success probability of $\mathcal{A}$ is at least ϵ and it invokes the hash function h_k at most a times. In this case we say that the adversary has an (ϵ, a)-strategy for $\{h_k\}_{k\in\mathcal{K}}$. Note that we do not include time as an explicit parameter though it would be easy to do so.

In this paper we are interested in extending the domain of a UOWHF. Thus given a UOWHF $\{h_k\}_{k\in\mathcal{K}}$, with $h_k : \{0,1\}^n \rightarrow \{0,1\}^m$ and a positive integer $L \geq 1$, we would like to construct another UOWHF $\{H_p\}_{p\in\mathcal{P}}$, with $H_p : \{0,1\}^L \rightarrow \{0,1\}^m$. We also consider the following situation. Given a positive integer $L < 2^{n-m}$ and a UOWHF $\{h_k\}_{k\in\mathcal{K}}$ where $h_k : \{0,1\}^n \rightarrow \{0,1\}^m$, we construct a UOWHF $\{H_p^*\}_{p\in\mathcal{P}}$ where $H_p^* : \cup_{i=1}^{L}\{0,1\}^i \rightarrow \{0,1\}^m$. 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.

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

The correctness of our construction will essentially be a Turing reduction. We will show that if there is an (ϵ, a)-strategy for $\{H_p\}_{p\in\mathcal{P}}$, then there is an (ϵ_1, a_1)-strategy for $\{h_k\}_{k\in\mathcal{K}}$, where a_1 is not much larger than a and ϵ_1 is not significantly lesser than ϵ. This will show that if $\{h_k\}_{k\in\mathcal{K}}$ is a UOWHF, then so is $\{H_p\}_{p\in\mathcal{P}}$.

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

3 Known Constructions

We briefly discuss the constructions which have already been proposed.

3.1 Sequential Construction

The Merkle-Damgård construction [3, 2] is a well known construction for collision resistant hash functions. However, Bellare and Rogaway [1] showed that the construction does not directly work in the case of UOWHF. In [9], Shoup presented a modification of the MD construction. We briefly describe the Shoup construction as presented in [4].

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

Algorithm SeqUOWHF

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

The function h_k is invoked r times and is called the r-round Shoup construction. The construction was proved to be correct by Shoup in [9]. In a later work Mironov [4] provided an alternative correctness proof. More importantly, in [4] it was shown that the amount of key length expansion is the minimum possible for the construction to be correct.

3.2 Tree Based Construction

Extension of UOWHF using a binary tree of processors have been considered in the literature [1, 7]. We first describe these constructions and then discuss the main drawbacks of these construction.

The construction extends a UOWHF $\{h_k\}_{k\in\mathcal{K}}$ to a UOWHF $\{H_p\}_{p\in\mathcal{P}}$, where $h_k : \{0,1\}^n \to \{0,1\}^m$ and $H_p : \{0,1\}^L \to \{0,1\}^m$. The tree based construction uses a binary tree of $2^T - 1$ processors numbered $P_1, \ldots, P_{2^T-1}$. The length of the message x to be hashed is $|x| = L = 2^{T-1}n + (2^{T-1}-1)(n-2m)$. The processors are connected in a binary tree fashion. We consider $\mathcal{T}_T = (V_T, A_T)$ to be the 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 strings and a function $\psi : A_T \to M$, which assigns an m-bit string to each arc of $\mathcal{T}_T$. The elements of M are called masks. 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].

3.2.1 Construction 1 [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.

$$\begin{aligned} \psi(a_i) &= \alpha_{T+1-l} \quad \text{if } i \equiv 0 \bmod 2; \\ &= \beta_{T+1-l} \quad \text{if } i \equiv 1 \bmod 2. \end{aligned}$$

Here $l = level(i)$, i.e., $2^{l-1} \leq i \leq 2^l - 1$.

3.2.2 Construction 2 [7]

Here $M = \{\alpha_1, \ldots, \alpha_{T-1}, \beta_0, \ldots, \beta_{r-1}\}$ where $r = 1 + \lfloor \log_2(T-1) \rfloor$ and $\psi(a_i)$ is defined as follows.

$$\begin{aligned} \psi(a_i) &= \beta_{\nu(T+1-l)} \quad \text{if } i \equiv 0 \bmod 2; \\ &= \alpha_{T+1-l} \quad \text{if } i \equiv 1 \bmod 2. \end{aligned}$$

Again $l = level(i)$.

Construction 2 requires $T + \lfloor \log_2(T-1) \rfloor$ masks compared to Construction 1 which requires $2(T-1)$ masks. Also it has been proved in [7], that for $T \geq 3$, at least $(T+1)$ masks are required for TreeUOWHF to be correct.

3.2.3 Drawbacks of Known Tree Based Constructions

As mentioned in the Introduction, there are three major drawbacks of currently known tree based algorithms [1, 7].

1. A processor tree of height T and $2^T - 1$ processors is used to hash a message of length $L = 2^T(n - m) - (n - 2m)$. Thus the number of processors grows with the length of the message. This is clearly undesirable for possible practical applications.

2. The speed-up achieved over the sequential algorithm is $\frac{2^T}{T}$, which is a logarithmic fraction of the number of processors used. We would like the speed-up to be equal to the number of processors used.

3. The key length expansion for a message of length L is $E_{bt}(L) = m(T + \lfloor \log(T-1) \rfloor)$ in [7] and is $E_{seq}(L) = mT$ in [9]. Thus the quantity $E_{bt}(L) - E_{seq}(L)$ depends on T and hence grows with the length of the message.

4 Construction Using Finite Binary Tree

Our algorithm is based on the binary tree algorithm of [8] used to extend the domain of a collision resistant hash function. 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. The non utilization of some of the processors results in at most one additional parallel rounds when $t' = t - 1$ (see [8]).

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 $D = (V, A)$, 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 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 strings and $\psi : A \to M$ be an assignment of these strings to the arcs of the processor graph $D = (V, A)$. Thus for each arc $A_{i,j}$ we have a mask $\psi(A_{i,j})$ associated with the arc. This m-bit string is used to "mask" the m-bit string $z_{i,j}$. Hence we will call M to be a set of masks. For collision resistant hash functions it is sufficient to take M to be a singleton set consisting of the all-zero string only. However, for constructing UOWHF, it is necessary to take $|M| > 1$ (see [1]).

Each of the processors $P_{i,j}$ is given a string $u_{i,j}$ as input where the message x is the concatenation of the strings $u_{i,j}$ for $0 \leq i \leq 2^t - 1$ and $1 \leq j \leq R$. The parsing (or formatting) of the message x into the strings $u_{i,j}$ have to be done carefully to ensure that each bit of the message is hashed and the algorithm correctly terminates and produces an m-bit message digest. For collision resistant hash functions the details of the parsing algorithm is provided in [8]. The same parsing algorithm will also work for UOWHF. Hence we direct the reader to [8] for details of the parsing algorithm and instead here we only mention some properties of $u_{i,j}$ which are necessary for understanding of the UOWHF algorithm. The lengths of the strings $u_{i,j}$ satisfy the following conditions.

$$\begin{array}{lcll}
|u_{i,1}| & = & n & \text{for } 0 \leq i \leq 2^t - 1; \\
|u_{i,j}| & = & n \text{ or } 0 & \text{if } j > 1 \text{ and } 2^{t-1} \leq i \leq 2^t - 1; \\
|u_{i,j}| & = & n - 2m \text{ or } 0 & \text{if } j > 1 \text{ and } 0 \leq i \leq 2^{t-1} - 1. \\
|u_{i,j}| & = & n - 2m & \text{if and only if } |z_{2i,j-1}| = m.
\end{array}$$

If $z_{i,j}$ is an m-bit string, then define $s_{i,j} = z_{i,j} \oplus \psi(A_{i,j})$, for $0 \leq i \leq 2^t - 1$ and $1 \leq j \leq R - 1$. We are now in a position to define the parallel UOWHF algorithm (PUA).

Parallel UOWHF Algorithm PUA(t, x)

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

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

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

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

end PUA

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

Remark : Let 0^m denote the all zero string of length m. If $\psi(A_{i,j}) = 0^m$ for all $0 \le i \le 2^t - 1$ and $1 \le j \le R$, then the above algorithm is identical to Algorithm PHA(t, x) of [8] which extends the domain of a collision resistant function.

4.1 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 $2^{l-1} \le i \le 2^l - 1$. 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 $1 \le j_1, j_2 \le R - 1$. The arcs for a fixed round are assigned masks using the algorithm of Section 3.2.2. Thus the mask assigment algorithm combines the mask assignment algorithms of [9] and [7].

4.2 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 PUA(t, x). Each function h_k is a map from $\{0,1\}^n$ to $\{0,1\}^m$ and each function H_p is a map from $\{0,1\}^L$ to $\{0,1\}^m$. Let M be the set of masks defined in Section 4.1 and let $\tilde{M}$ be the concatenation of all the masks in M. Then the key p for the function H_p is defined as $p = k||\tilde{M}$. The function H_p is defined as follows.

$$H_p(x) = PUA(t, x). \tag{1}$$

4.3 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.

4.4 Speed-Up Over Sequential Algorithm

The number of invocations of h_k depends on the length of the message length L. Let $\psi(L)$ denote the number of invocations of h_k for a message of length L. Then $\psi(L) = \psi(L + b_t(L)(2n - 2m) - r_t(L))$. The value of $\psi(L)$ has been computed in [8]. The same value also applies to the present case. Hence we state the result without proof.

Proposition 1 $\psi(L) = (q_t(L) + 2)2^t + 2b_t(L) - 1$.

Remark : As shown in [8], $\psi(L)$ 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

$$SF = \frac{\psi(L)}{q + t + 2} = \frac{(q+2)2^t + 2b - 1}{q + t + 2}.$$

If $b > 0$, then

$$SF \geq \frac{2^t(q+2)}{q+2+t} = \frac{2^t}{1 + \frac{t}{q+2}}.$$

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.

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

Theorem 2 *If there is an (ϵ, N) strategy for $\{H_p\}_{p \in \mathcal{P}}$, then there is an $(\frac{\epsilon}{\psi(L)}, N + 2\psi(L))$ 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)-strategy $\mathcal{B}$ for the family $\{H_p\}_{p \in \mathcal{P}}$ and construct an $(\frac{\epsilon}{\psi(L)}, N + 2\psi(L))$ 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 s_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 $|u_{I,J}| = n$, 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 $s = (s_1, I, J)$.

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 $\mathcal{B}^{\text{find}}(x, p, s_1)$ to obtain string x'.
6. Run the tree hash algorithm 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 described in Section 3.2.2. 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 $\frac{1}{\psi(L)}$. 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 $\frac{\epsilon}{\psi(L)}$. Strategy $\mathcal{B}$ invokes h_k at most N times and strategy $\mathcal{A}$ invokes h_k at most $2\psi(L)$ additional times. ■

6 Comparison to Previous Algorithms

We provide two kinds of comparisons of Algorithm PUA to previous algorithms. The first comparison is based on key length expansion and the second is on the achieved speed-up.

6.1 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 3 *Let x be a message of length L and N_1 and N_2 be the number of masks required by Algorithms PUA and SeqUOWHF respectively 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 4.1, 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 $\psi_t(L) = (q+2)2^t+2b-1$ (see Proposition 1). 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 $N_2 = 1+\lfloor\log(\psi_t(L)-1)\rfloor$. Thus

$$N_1 - N_2 \quad = \quad (2+t+\lfloor\log(q+t+1)\rfloor+\lfloor\log(t-1)\rfloor) - (1+\lfloor\log(2^t(q+2)+2b-2)\rfloor)$$

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 $0 \le b \le 2^{t-1}$. Thus we have $1+\lfloor\log(2^t(q+2)-2)\rfloor \le N_2 \le 1+\lfloor\log(2^t(q+3)-2)\rfloor$. From this we get $\log(2^t(q+2)-2) < N_2 \le 1+\log(2^t(q+3)-2)$. We can write this as $t+\log(q+2)-\varepsilon < N_2 \le t+1+\log(q+3)-\sigma$ 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

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

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. ■

Remark : 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 $\lfloor\log(t-1)\rfloor+2$ extra masks compared to SeqUOWHF. Thus PUA makes a *constant* amount of additional key length expansion compared to SeqUOWHF. For previous binary tree algorithms the additional amount of key length expansion over the sequential algorithm increased with the length of the message.

We now compare the number of masks used by PUA to the number of masks used by TreeUOWHF.

Theorem 4 *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. ■

The parameter T grows with the length of the message and for $T > t + \log(t-2)$ we have $A - B$ to be at least $\lfloor \log(T-1) \rfloor - \lfloor \log(t-1) \rfloor - 2$. Hence the key length expansion made by Algorithm PUA is significantly smaller than that made by TreeUOWHF.

Remark : Based on Theorems 3 and 4 we can conclude that the key length expansion made by PUA is smaller than that made by TreeUOWHF and is a constant number of bits more than that made by SeqUOWHF.

6.2 Speed-Up

Both PUA and TreeUOWHF achieve speed-up over SeqUOWHF. Here we compare the speed-up factor of these algorithms. The speed-up factor of TreeUOWHF is $SF_1 = \frac{2^T}{T}$ when a message of length $L = 2^T(n-m) - (n-2m)$ is hashed using a processor tree of height $T > 2$. Suppose PUA is used to hash a message of length L using a processor tree of height $t < T - 1$. The number of parallel rounds required by PUA is $q + t + 2$, where $q = 2^{T-t} - 3$ as computed in the proof of Theorem 4. The speed-up factor in this case is $SF_2 = \frac{2^T}{2^{T-t}+t+2} = \frac{2^t}{1+2^{t-T}(t+2)}$.

Clearly, $SF_1 > SF_2$ and hence TreeUOWHF is faster than PUA. The reason for this is the fact that TreeUOWHF uses a larger processor tree compared to PUA. However, the efficiency of speed-up of PUA is better than that of TreeUOWHF. The parameter T grows with the message whereas the parameter t is constant. Hence the speed-up attained by TreeUOWHF is $(1/T)$ times the number of processors used (2^T), whereas for (moderately) long messages, the speed-up attained by PUA is almost equal to the number of processors used (2^t). Hence compared to TreeUOWHF, PUA makes better utilization of the processors.

7 Variable Length Inputs

Let L be a positive integer less than 2^{n-m}. Given a UOWHF $\{h_k\}_{k \in \mathcal{K}}$, where $h_k : \{0,1\}^n \to \{0,1\}^m$ and L, 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 $L < 2^{n-m}$. 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. Let $\tilde{M}$ be the concatenation of all the masks in M. Let $\{h_k\}_{k \in \mathcal{K}}$ be a UOWHF, and $p = k || \tilde{M}$. Suppose x is a message of maximum length $L < 2^{n-m}$. Then $\{H^*_p\}_{p \in \mathcal{P}}$ is defined in the following manner. For $0 \leq i < 2^k$, by $bin_k(i)$ we denote the k-bit binary representation of i.

Computation of $H_p^*(x)$:

1. Set $z = \mathrm{PUA}(t, x)$.
2. Set $u = z||bin_{n-m}(|x|)$.
3. Return $h_k(u)$.

The idea is to compute the output z of $\mathrm{PUA}(t, x)$, concatenate $|x|$ as an $(n-m)$-bit binary number to z and apply h_k to the resulting n-bit string u. We next show that the definition preserves the UOWHF property.

Theorem 5 *Let $L < 2^{n-m}$ be a positive integer. If there is an (ϵ, N)-strategy $\mathcal{A}$ for $\{H_p^*\}_{p\in\mathcal{P}}$, with $H_p^* : \cup_{i=1}^{L}\{0,1\}^i \to \{0,1\}^m$, then there is an $(\frac{\epsilon}{\psi(L)}, 2 + 2\psi(L) + N)$-strategy $\mathcal{B}$ for $\{h_k\}_{k\in\mathcal{K}}$. 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 2. $\mathcal{B}^{\text{guess}}$ first invokes $\mathcal{A}^{\text{guess}}$ to obtain a string x of length at most L 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 2. 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 $|x| \neq |x'$, then Step 3 in the computation of H_p^* provides a collision for h_k. On the other hand, if $|x| = |x'|$, then as in the proof of Theorem 2, we obtain a collision for h_k with probability at least $\frac{\epsilon}{\psi(L)}$. 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 $\mathrm{PUA}(t, x)$. This gives us the required result. ■

8 Concluding Remarks

In this paper we have developed a finite binary tree based algorithm to properly extend the domain of a UOWHF. Our algorithm is based on the finite binary tree algorithm described in [8] for extending the domain of a collision resistant hash function. The mask assignment procedure is a combination of the mask assignment procedure of the best known sequential algorithm [9] and the mask assignment procedure for the binary tree algorithm of [7].

Our algorithm achieves the following. For moderately long messages, the speed-up obtained is almost equal to the number of processors used. Further, the key length expansion of our algorithm is a constant number of bits more than the key length expansion made by the sequential Shoup algorithm [9]. In previously known binary tree algorithms, the number of processors increased with the length of the message. The speed-up attained was a logarithmic fraction of the total number of processors and the key length expansion was significantly more than the sequential Shoup algorithm.

Thus our binary tree based algorithm provides a good parallelizable design principle for building practical UOWHFs. Future work will focus on actual design of a practical UOWHF based on the principles described in this paper.

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/.