Optimal Domain Extension of UOWHF and a Sufficient Condition

Mridul Nandi
Applied Statistics Unit
Indian Statistical Institute
mridul_r@isical.ac.in

Abstract. In this paper we will provide a non-trivial sufficient condition for UOWHF-preserving (or valid) domain extension which will be very easy to verify. Using this result we will prove very easily that all known domain extension algorithms are valid. This will be a nice technique to prove and to construct a valid domain extension. We also propose an optimal (with respect to both time complexity and key size) domain extension algorithm based on an incomplete binary tree.

Keywords : Hash function, UOWHF, Domain Extension Algorithm, masking assignment.

1 Introduction

A UOWHF or Universal One-Way Hash Function is a family of hash functions \( \{h_k\}_{k \in K} \) with \( h_k : \{0,1\}^n \rightarrow \{0,1\}^m \), where the following task is hard: adversary has to commit an \( n \)-bit string \( x \) and then given a random key \( k \) he has to find another \( n \)-bit string \( x' \neq x \) such that \( h_k(x) = h_k(x') \). The pair \( (x, x') \) with \( x \neq x' \) and \( h_k(x) = h_k(x') \) is known as collision pair. More precisely, \( \{h_k\} \) is \((\epsilon, t)\)-UOWHF if every adversary with runtime at most \( t \) has success probability (i.e. probability of finding the collision pair in the above task) at most \( \epsilon \). We say the above hash family \( \{h_k\}_{k \in K} \) is \((n, m, K)\) hash family if \( K = \{0,1\}^K \) and for each \( k \), \( h_k \) is an \((n, m)\) hash function. \( K \) and \( K \) are known as the key space and key size respectively. Here, we are mainly interested in valid or UOWHF-preserving domain extension which means that given a \((n, m, K)\) hash family \( \{h_k\}_{k \in K} \) (called base hash family) which is \((\epsilon, t)\)-UOWHF we want to construct another \((N, m, P)\) \((\epsilon', t')\)-UOWHF \( \{H_p\}_{p \in P} \) (called extended hash family) based on \( \{h_k\} \) where \( N > n \) and \( \epsilon', t' \) are constant multiples of \( \epsilon, t \) respectively. We will be interested in valid domain extensions where key expansion i.e. \((P - K)\) is as small as possible. Also we will try to reduce the time complexity by considering parallel domain extension algorithms.

Brief History. To sign a big message it is always better to compress the message first and then run a short domain signing algorithm on the compressed message. To have the security of signature scheme we need a Collision Resistant
Hash Function or CRHF in which given a random key $k$ it is hard to find a collision pair. But, Bellare and Rogaway (BR) [1] constructed a generic signature scheme where a UOWHF is sufficient to prove the security of the signature scheme. In their algorithm, $\text{sig}_{sk}(h_k(\text{M})||k||k)$ is a signature of the message $M$. If the key size is large then one can use $\text{sig}_{sk}(h_{k_1}(h_{k_2}(\text{M})||k_1)||k_2)||k_1||k_2$ as a signature of the message $M$. Usually the key size is $O(\log(|M|))$ so input size of $\text{sig}_{sk}(\cdot)$ is $m + O(\log(\log|M|))$ which is very small. UOWHF is first introduced by Naor and Yung [7] and they constructed a UOWHF based on a one-way function. But the construction is much theoretical and slow. To construct a UOWHF of arbitrary domain we start with a construction of UOWHF with smaller domain from scratch and then extend it to an arbitrary domain. Natural domain extension method is MD construction which works for CRHF. But unfortunately, Bellare and Rogaway in the same paper [1] showed that MD construction will not work in case of UOWHF. They proposed a binary tree based construction with the notion of XOR-ing masks (parts of the key). Then Shoup [10] constructed a sequential domain extension and Mironov [4] proved that it is optimum in key size among all sequential construction (say $\mathcal{S}$ denotes set of all sequential construction). Sarkar [9] gave another binary tree based construction where the number of masks or key size is less than that of BR but it is more than that of Shoup. But this algorithm (also the BR algorithm) can be implemented in parallel. So these will be much faster than Shoup’s algorithm. Sarkar considered a general class of domain extension algorithm (say $\mathcal{C}$, $\mathcal{S} \subset \mathcal{C}$) which includes all known UOWHF-preserving domain extension algorithm and provided a lower bound for the number of masks (or key size) to have a valid domain extension from $\mathcal{C}$. In $\mathcal{S}$ the both bounds given by Sarkar and Mironov agree. Nandi [5] modified the Sarkar’s algorithm with less number of masks. Lee et al [6] constructed an optimum algorithm in the general class $\mathcal{C}$ but parallelism is much smaller than binary tree based algorithm because they used incomplete $l$-ary tree. Finally in this paper we have an algorithm which has maximum possible parallelism and minimum key size.

**Motivation.** It is clear that UOWHF is a weaker notion than CRHF in the sense that a hash family is UOWHF whenever it is CRHF but the converse need not be true. In fact, Simon [11] proved that there exists an oracle relative to which UOWHF exists but CRHF does not exist. Unlike CRHF the birth-day attack will not work in the case of UOWHF. So roughly, to have a collision in UOWHF one needs $O(2^m)$ many computations whereas in CRHF one can find a collision pair in $O(2^{m/2})$ computations. So one can use the signature scheme proposed by BR using any standard hash functions e.g. SHA-256 or RIPEMD-160. Till now, we believe that those hash functions are CRHF and we can study the security of the signature scheme under the assumption of UOWHF. We can treat SHA-256, RIPEMD-160 as a hash family keyed by the initial values. Even if somebody finds a collision for the above hash functions it will not give any immediate threat to the signature scheme. One disadvantage for UOWHF is that if the signer himself is dishonest then the signature scheme proposed by Bellare
and Rogaway will not be secure. Suppose \( \{h_k\}_k \) is UOWHF but not CRHF. So, there exists a collision pair \( M_1 \) and \( M_2 \) for \( h_k \). The signer can sign the message \( M_1 \) with the key \( k \) and then one can forge the signature of the message \( M_2 \). This problem could be solved if the signer does not have any control to choose the key \( k \).

**General Domain Extension Algorithm.** The general class of domain extension algorithm \( \mathcal{C} \) is described in detail in [9]. Here we give a brief discussion on this. It is very natural that to extend the domain of a function we have to apply the function repeatedly. The question is how we will combine this iteration. If the output of one invocation of \( h_k \) is completely fed into the input of another invocation then the method of combination can be completely captured by a rooted directed tree. \( T = (V, E, q) \) is called a rooted directed tree where \( V = [1, r] := \{1, \ldots, r\} \) is the set of vertices, \( E \) is the set of arcs and \( q \in V \) is a special vertex called the root of the tree with the property that, \( \text{outdeg}(q) = 0 \) whereas \( \text{outdeg}(i) = 1 \) for other vertices \( i \). Here, \( \text{outdeg}(i) = |\{j : (i, j) \in E\}| \).

A hash function \( h_k(\cdot) \) is placed on each node of \( T \). The output of \( h_k(\cdot) \) is passed through the arc i.e. if \( (i, j) \in E \) then the output of \( h_k(\cdot) \) at node \( i \) is fed into the input of \( h_k(\cdot) \) at node \( j \). For example, a sequential domain extension i.e. MD construction can be viewed by a sequential tree. As this is not UOWHF-preserving domain extension the notion of XOR-ing mask (a part of the key) is introduced. So, after each invocation the output is XOR-ed with some mask before feeding into next invocation. To determine the algorithm we have to specify which mask will be XOR-ed for every invocation. So we have a function \( \psi : E \to [1, l] := \{1, \ldots, l\} \) known as masking assignment. We also say that \( \psi \) is a \( l \)-masking assignment. We also say that \( \psi \) is strongly even-free (See Definition 1). We show that all known valid domain extensions belong to the class \( \mathcal{C} \) and satisfy the sufficient condition viz. the masking assignments are strongly even-free. Hence one can try to prove that the condition is also a necessary condition which will completely characterize the UOWHF-preserving domain extension algorithm. This sufficient condition will be very easy tool to prove that a domain extension is valid. It also helps to construct an optimal (with respect to both time complexity and key size) valid domain extension for the general class. In fact, we construct an optimum domain extension algorithm for the general class.

**Our Contributions and Future Work.** In this paper we provide a non trivial sufficient condition for valid domain extensions in the general class \( \mathcal{C} \) defined by Sarkar [9]. More precisely, a domain extension algorithm based on \( (T, \psi) \) is UOWHF-preserving or valid if \( \psi \) is strongly even-free (See Definition 1). We show that all known valid domain extensions belong to the class \( \mathcal{C} \) and satisfy the sufficient condition viz. the masking assignments are strongly even-free. Hence one can try to prove that the condition is also a necessary condition which will completely characterize the UOWHF-preserving domain extension algorithm. This sufficient condition will be very easy tool to prove that a domain extension is valid. It also helps to construct an optimal (with respect to both time complexity and key size) valid domain extension for the general class.
2 The General Domain Extension Algorithm

Some Notes on Rooted Directed Tree. In any rooted directed tree \( T = (V, E, q) \), from any vertex \( i \) there is one and only one path from that vertex \( i \) to the root \( q \). So, we can define \( l(i) \) (called level of the vertex \( i \)) by the number of vertices in the unique path from \( i \) to \( q \). Note, \( l(q) = 1 \). Write, \( V[k] = \{ i \in V : l(i) = k \} \) for each \( k \geq 1 \). Define \( h(T) \) (called height of the tree \( T \)) by \( \max_{i \in V} l(i) \). We also use the notation \( h(i) \) (called height of \( i \)) for \( t - l(i) + 1 \) when \( T \) is a complete binary tree of height \( t \). A sub-tree \( T_1 \) of \( T \) is the tree induced by the subset of the vertex set \( V \). Root of a sub-tree \( T_1 = (V_1, E_1) \) is the vertex with minimum level. More precisely, \( i \) is called a root of the sub-tree \( T_1 \) if \( i \in V_1 \) and \( l(i) = \min_{j \in V_1} l(j) \). If \( i \in V \), define \( V(i) \) by the set of all vertices from which there is a path to the vertex \( i \). Note, \( V(i) = \{ j \in V : \) there is a path from \( j \) to \( i \}\). We will say the induced full sub-tree rooted at \( i \) by the sub-tree induced by \( V(i) \) (in notation \( T(i) \)). Note that \( i \) becomes the root of the sub-tree \( T(i) \). Define \( \text{son}(i) = \{ j : (j, i) \in E \} \). In the next paragraph we state the general algorithm in the class \( C \) defined by Sarkar [9].

Domain Extension Algorithm (to compute \( H_p(X) \))

Let \( \psi \) be a \( l \)-masking assignment on \( T = (V, E, q) \) where \( V = [1, r] \) for some positive integers \( l \) and \( r \). We want to define \( (N, m, P) \) hash family \( \{ H_p \}_{p \in P} \) given a \((n, m, K)\) hash family \( \{ h_k \}_{k \in K} \) where, \( N = (n - m)r + m \) and \( P = K + m.l \).

Write, \( p = k_1||k_2|\ldots||k_l \) and \( X = x_1||\ldots||x_r \) where, \( |p| = P, |k| = K, |\mu_i| = m, |X| = N \) and \( |x_i| = n - \text{indeg}(i) \times m \). We need to assume that, \( n \geq \delta(T) \times m \) where, \( \delta(T) = \max_{i \in V} \text{indeg}(i) \). Note, \( |X| = \sum_{i=1}^{r} |x_i| = (n - m)r + m \). Here, \( p \) is a key of extended hash family and \( X \) is any input of that hash family. We will treat \( k \) as a key of base hash family. We use the term mask for \( \mu_i \)'s. Now we are ready to define \( H_p(X) \) using the hash function \( h_k \).

1. For \( i \in V[t] \) (\( t = h(T) \))
   Compute \( z_i = h_k(x_i) \).
2. For \( j = t - 1 \) down to 1
   For \( i \in V[j] \) do in parallel
   \( z_i = h_k((z_{i_1} \oplus \mu_{\psi(i_1)}) \ldots ((z_{i_d} \oplus \mu_{\psi(i_d)}) ||x_i) \) where \( \text{son}(i) = \{ i_1, \ldots, i_d \} \) and \( i_1 < \ldots < i_d \).
3. \( z_q \) is the output of \( H_p(X) \).

Main parameters of the above domain extensions:

1. Key expansion is most important parameter in practical point of view. For above type of domain extension algorithm (key expansion) = (number of masks) \( \times \) (size of range). So we need to have valid domain extension with smallest possible number of masks. In [8] author showed that at least \( \lfloor \log_2 r \rfloor \) many masks are necessary to have a valid domain extension where \( r \) is the total number of invocation i.e. number of vertices in the tree. Later we will
construct a parallel algorithm called opt which needs \( t \) many masks for \( 2^t \) many invocation which is minimum possible.

2. The algorithm from above class can be implemented in parallel (unless the tree is a sequential tree i.e. a path). If we run the algorithm in parallel, number of rounds is same as the height of the tree. If we have \( n \geq 2m \) then we can consider binary trees. So number of rounds should be at least \( \lfloor \log_2 (r + 1) \rfloor \) where \( r \) is the number of vertices of the tree binary tree. A complete binary tree of height \( t \) has \( 2^t - 1 \) many vertices and hence a domain extension algorithm based on a complete binary tree needs \( t \) rounds only. Note that, this will have maximum possible parallelism if we only assume that \( n \geq 2m \). Later we will show that our construction opt needs \( t + 1 \) rounds for \( 2^t \) invocation which is minimum possible for binary tree based domain extensions.

3 Sufficient Condition of UOWHF-preserving Domain Extension

In [8] it was proved that every valid domain extension should be based on even-free masking assignment (See Definition 1). In this section we will prove that any domain extension based on a strongly even-free masking assignment (See Definition 1) is valid. We also show that all known valid domain extensions satisfy this sufficient condition.

**Definition 1.** A \( l \)-masking assignment \( \psi \) on \( T = (V, E) \) is called even-free masking assignment if for any non-trivial sub-tree \( T_1 = (V_1, E_1) \) of \( T \) there exists \( i \in [1, l] \) such that \( i \) appears odd many times in the multi-set \( \psi(E_1) = \{ \psi(e) : e \in E_1 \} \). Similarly, \( \psi \) on \( T = (V, E) \) is called strongly even-free masking assignment if for any non-trivial sub-tree \( T_1 = (V_1, E_1) \) of \( T \) there exists \( i \in [1, l] \) such that \( i \) appears exactly once in the multi-set \( \psi(E_1) = \{ \psi(e) : e \in E_1 \} \). This \( i \) is called a single man for that sub-tree \( T_1 \).

**Theorem 1.** If a domain extension algorithm is based on a strongly even-free masking assignment \( \psi \) on \( T \) then it is a valid domain extension. More precisely, if for any \((\epsilon, t)\)-strategy for \( \{H_p \} \) there is an \((\epsilon', t')\)-strategy for \( \{h_k \} \) where \( \epsilon' = \epsilon/r \) and \( t' = t + O(r) \) where \( r \) is the size of the tree i.e. \( r = |V| \).

**Proof.** Let \( \mathcal{A} \) be an adversary with runtime at most \( t \) and success probability at least \( \epsilon \) for \( \{H_p \} \). Now we will define an adversary \( \mathcal{B} \) for \( \{h_k \} \) with runtime atmost \( t' \) and success probability at least \( \epsilon' \).

**guess:**

1. \((X, s') \leftarrow \mathcal{A}^{\text{guess}} \cdot (|X| = N)\)
2. Choose \( i \in_R V = [1, r] \) (\( i \) is chosen randomly from \([1, r] \)).
   - If \( i \in V[h] \), set \( y = x_i \), \( s = (s', i, y) \). Output \((y, s)\) and stop.
   - Else \( r_{i_1}, \ldots, r_{i_d} \in_R \{0, 1\}^m \) (randomly) where \( \text{son}(i) = \{i_1, \ldots, i_d\} \), \( y = r_{i_1}||\ldots||r_{i_d}||x_i \) and \( s = (s', i, y) \) where, \( i_1 < \ldots < i_d \). Output \((y, s)\) and stop.
At this point the adversary is given a $k$ which is chosen uniformly at random from the set $K = \{ 0, 1 \}^K$. The adversary then runs $B^\text{find}$ which is described below.

$B^\text{find}(y, k, s) :$ (Note $s = (s', i, y)$.)

1. $\mu_1||\ldots||\mu_l \leftarrow M^\text{def}(X, k, i, r_{i_1}||\ldots||r_{i_d}, T, \psi)$ (see the algorithm $M^\text{def}$ below).
2. $X' \leftarrow A^\text{find}(X, p, s')$ where $p = k||\mu_1||\ldots||\mu_l$. Let $y'$ be the input to processor at node $i$ while computing $H_p(X')$. Output $y'$.

Now we state a lemma which says that there exists an algorithm $M^\text{def}(\cdot)$ which outputs random string where input of a specified node is predefined random string. More precisely,

**Lemma 1.** There exists an algorithm $M^\text{def}(X, k, i, r_{i_1}||\ldots||r_{i_d}, T, \psi)$ which always returns a random string $\mu_1||\ldots||\mu_l$ whenever $r_{i_1}||\ldots||r_{i_d}$ is a random string. Also input of node $i$ while computing $H_p(x)$ is $r_{i_1}||\ldots||r_{i_d}|x_i$ if $i \notin V[h]$. 

**Proof of the lemma:**

First we describe the algorithm below Algorithm $M^\text{def}(X, k, i, r_{i_1}||\ldots||r_{i_d}, T, \psi)$ (Note $|r_i| = m$ and $d = \text{indeg}(i)$)

1. We can assume that $i = q$ the root of the tree (otherwise we can do the same thing for the induced full sub-tree rooted at $i, T(i)$). Suppose, $\psi(e_u) = l$ (say) is a single man for $T$. Let $T' = T - (T(u) \cup \{ e_u \}) = (V', E')$. If $r' = |E'|$ then, $r' < r$. Assume, $u \in \text{son}(j)$ and $\text{son}(j) = \{ j_1, \ldots, j_c = u \}$ where $j_1 < \ldots < j_c$. If $j = q$ then $R = r_u = r_{i_d}$ otherwise it is a random string. Define, $x_j' = R|x_j, x_k' = \varepsilon$ (empty string) if $k \in V' - \{ j \}$, otherwise $x_k' = x_k$. Now, we define $X' = x_1'||\ldots||x_{j_c}'$. Run recursively $M^\text{def}(X', k, q, r_{i_1}||\ldots||r_{i_d}, T', \psi')$ if $j = q$ or $M^\text{def}(X', k, q, r_{i_1}||\ldots||r_{i_d}, T', \psi')$ if $j \neq q$ to define the masks where $\psi'$ is $\psi$ restricted on $T'$. Note $M^\text{def}$ will always define those masks which are in the range of masking assignment. When we call $M^\text{def}(\cdot, \cdot, \psi')$ it will not define $\mu_l$ as $l$ is not in the range of $\psi'$.

2. If $|E| = 1$ then $\mu = \mu_1$ where $\mu_1 = h_k(x) \oplus r_u$.
3. Define all other yet undefined masks except $l$ by random strings. Compute the output at vertex $u$, call it by $z$. Define $\mu_l = R \oplus z$. This will completely define all masks.

Note that we assume that $j_c = u$ which need not be true. To avoid this problem we can redefine the general tree based domain extension by considering any order of the input at all nodes. Previously the input at node $i$ is $s_i||\ldots||s_{i_c}|x_i$ but we can modify it to $\sigma_i(s_i||\ldots||s_{i_c}|x_i)$ where $\sigma_i$ is some permutation of $n$-bit strings. In that case we have to redefine the permutation accordingly when we recursively call $M^\text{def}$. For simplicity of the proof we can ignore this. We can check easily that all the masks are random strings as either they are random strings or they are XOR of two strings one of which is a random string. So, we can prove the first part of the lemma by using induction on size of the tree. We
also prove the second part of the lemma by induction. So, if input at node \( j \) contributed by node \( u \) is \( R \) then input of node \( i \) will be \( r_i || \ldots || r_j || x_i \) by the induction hypothesis. The mask \( \mu_i \) is defined by \( \mu_i = z \oplus R \). As output of node \( u \) be \( z \) (see step-3 in the algorithm \( Mdef \) ) the input at node \( j \) contributed by node \( u \) is \( R \). Note that \( \mu_i \) is a single-man so it is not used any where else. This proves the lemma.

So, by above lemma input of node \( i \) while computing \( H_p(X) \) will be \( y \) which is already committed in \( Br\)uess. Also note that \( p \) is a randomly chosen key from the set \( P \) as both \( k \) and \( \mu_i \)'s are random strings. Now We now lower bound the winning probability. Suppose \( X \) and \( X' \) collides for the function \( H_p \). Then there must be a \( v \in V \) such that at vertex \( v \) there is a collision for the function \( h_k \).

(Otherwise it is possible to prove by a backward induction that \( X = X' \).) The probability that \( i = v \) is \( \frac{1}{|V|} \). Hence, if the winning probability of \( A \) is at least \( \epsilon \), then the winning probability of \( B \) is at least \( \frac{\epsilon}{|V|} \) as two events \( i = v \) and \( A \) wins are independent (the value of \( i \) is not known to \( A \)). Also the number of invocation of \( h_k \) by \( B \) is equal to the number of invocation of \( h_k \) by \( A \) plus at most \( 2|V| \).

(The number \( 2|V| \) is coming from the fact that in \( Mdef \) algorithm we need at most \( |V| \) many invocations and we may need at most \( |V| \) many invocation of \( h_k \) again to compute \( y' \).) We skip the checking of time parameter as it is easy to verify. This completes the proof of the theorem. □

### 3.1 Sufficient Condition and some of known previous constructions

One can check easily that all previously known domain extension algorithms belong to the class \( C \). We will prove some of previous constructions are valid using the above sufficient condition. The same technique will also work for other known secure domain extensions. So, we reduce a problem of computational reduction to verifying strongly even-free property of a function (i.e. whether a masking assignment is strongly even-free or not) which will be more easy task.

We list some known algorithms in terms of their structures.

1. **Shoup** [10] : \( V = [1, r] \), \( E = \{(i, i + 1) : 1 \leq i \leq r - 1\} \), \( q = r \) is the root and \( \psi(i) = \nu_2(i) + 1 \), \( \nu_2(i) = j \) means that \( 2^i | i \) but \( 2^{i+1} \not| i \).

2. **Bellare-Rogaway** [1] , **Sarkar** [9], **Nandi** [5] : The tree is full binary tree of height \( t \) and the masking assignments are given below. The complete binary tree of height \( t \) has a set of vertices \([1, 2^t - 1]\) and a set of edges \( E = \{e_i = (i, [i/2]) : 2 \leq i < 2^t\} \).

3. **Lee et al** [6] : In their paper a 4-dimensional construction is given which can be generalized to \( l \)-dim construction. Here we will describe 2-dimensional construction for simplicity. For integer \( t \), \( g(t) = (a, b) \), where \( a = \lfloor t/2 \rfloor \), \( b = \lfloor (t + 1)/2 \rfloor \). \( T_i = (V_i, E_i, 1) \) be a rooted binary tree, where \( V_i = [1, 2^i] \) and \( E_i = \{e_i : 2 \leq i \leq 2^t\} \) where \( e_i = (i, i - 1) \) for \( 2 \leq i \leq 2^a \), \( e_i = (i, i - 2^a) \) for \( 2^a < i \leq 2^{a+b} = 2^t \) (note, \( a + b = t \)).

The authors defined two functions \( \alpha_t \), \( \beta_t \) as follows.

(a) \( \alpha_t : [1, 2^a - 1] \rightarrow [1, a] \) is defined by \( \alpha_t(i) = 1 + \nu_2(2^a - i) \).

(b) \( \beta_t : [1, 2^b - 1] \rightarrow [a + 1, a + b] \) is defined by \( \beta_t(i) = a + 1 + \nu_2(2^b - i) \).
The masking assignment \(\psi_t(e_i)\) is defined as follow (See figure ??):

(a) \(\psi_t(e_i) = \alpha_t(j)\) if \(2 \leq i \leq 2^a\) and \(j = i - 1\).
(b) \(\psi_t(e_i) = \beta_t(j)\) if \(2^a < i \leq 2^{a+b}\) and \(j2^a < i \leq (j + 1)2^a\).

To define the masking assignment used in \([1,9,5]\) we have to define level-uniform masking assignments.

**Definition 2. (Level-uniform masking assignment)**

A masking assignment \(\psi\) is said to be a level-uniform masking assignment on a complete binary tree \(T_t = (V_t, E_t)\) of height \(t\) if there are two functions \(\alpha_t\) and \(\beta_t : [2, t] \to [1, t]\) such that \(\psi(2i + 1) = \alpha_t(j)\) and \(\psi(2i) = \beta_t(j)\) where \(2^t-1 \leq i < 2^t-1\). The edge \((2i+1, i)\) (or \((2i, i)\)) will be known as \(\alpha\)-edge (or \(\beta\)-edge).

In every complete binary tree all nodes excepts leave (the vertices \(i\) where \(2^t-1 \leq i < 2^t\)) have two sons called left or right son. So, a level-uniform masking assignments depends on the level of the vertex and type of son i.e. whether it is a left or right son of its father. Also the values of masking assignment of \(\alpha\)-edges (or \(\beta\)-edges) are determined the functions \(\alpha_t\) (or \(\beta_t\)). The masking assignments for Sarkar [9], BR [1] and Nandi [5] are level-uniform so we only describe the functions \(\alpha_t\) and \(\beta_t\).

1. **Bellare-Rogaway** [1] : \(\beta_t(i) = i - 1\) and \(\alpha_t(i) = t + i - 2\). no. of masks = \(2(t - 1)\).
2. **Sarkar** [9] : \(\alpha_t(i) = i - 1\) and \(\beta_t(i) = t + \nu(i - 1)\). no. of masks = \(t + \lfloor\log_2(t - 1)\rfloor\).
3. **Nandi** [5, 6] : Define two sequences \(\{l_k\}_{k \geq 0}\) and \(\{m_t\}_{t \geq 2}\) as follow : \(l_{k+1} = 2^l_k + l_k\) where, \(l_0 = 2\) and if \(k \geq 1, m_t = t + k\) for all \(t \in [l_{k-1} + 1, l_k]\).

Define \(m_2 = 2\). Note that, both \(l_k\) and \(m_t\) are strictly increasing sequences and if \(t = l_k\) for some \(k\) then \(m_{t+1} = m_t + 2\) and if for some \(k, l_k < t < l_{k+1}\) then \(m_{t+1} = m_t + 1\). It is proved in [5] that no of masks = \(m_t = t + O(\log_2 t)\).

Here, \(\log_2 t = j\) means that after applying log function \(j\) many times for \(t\) it becomes less than 1 for first time. The recursive definitions of \(\alpha_t\) and \(\beta_t\) are given below:

(a) \(\alpha_2(2) = 2\) and \(\beta_2(2) = 1\).
(b) For \(t \geq 3, \alpha_t(i) = \alpha_{t-1}(i)\) and \(\beta_t(i) = \beta_{t-1}(i)\) whenever \(2 \leq i \leq t - 1\).
(c) \(\alpha_t(t) = \alpha_{t-1}(t - 1) + 2\), \(\beta_t(t) = \alpha_{t-1}(t - 1) + 1\) if \(t = l_k + 1\) for some \(k\) and \(\alpha_t(t) = \alpha_{t-1}(t - 1) + 1\), \(\beta_t(t) = \alpha_{t-1}(t - 1) + 1\) if \(l_k < t - 1 < l_{k+1}\).

Now we can use the above theorem to prove the UOWHF-preserving property for the domain extension algorithms presented in this Section. In fact, one can check that all other valid domain extensions are based on strongly even-free masking assignments.

**Theorem 2.** The domain extension algorithms \([1,10,9,5,6]\) presented above are valid domain extensions. In fact, all these domain extension algorithms are based on strongly even-free masking assignments.
Proof. We only prove that all these domain extensions are based on strongly even-free masking assignments. So the theorem follows from Theorem 1. We will prove only in two cases. other cases are very easy to prove so we skip the proofs.

(1) (Nandi [5]) : Take a sub-tree say $S$ rooted at height $t'$ and $l_k + 1 \geq t' \geq l_k$ then from height $l_k + 1$ to $l_k + 1$ no $\alpha$-edge can be in $S$ (otherwise first such one i.e. the $\alpha$ edge having maximum height will be a single-man so we are done). So, $T'$ can contain at most one $\beta$ edge at height $l_k + 1$. If it contain that then $\beta_i(l_k + 1)$ is a single-man for that sub-tree. So, if $S$ does not contain that $\beta$ edge then again it is a sequential sub-tree consists of only $\beta$-edges from height $t'$ to at least height $l_k + 1$. But on that tree masking assignment is define by $\nu_2$ function which is itself strongly even-free masking assignment. So, the above masking assignment is strongly even-free.

(2) (Lee et al $(l-dim)$) [6] : Let $T' = (V', E')$ be any sub-tree. Note that, $[1, 2^a] \cap V'$ is an interval say, $[c, d]$. If $d > c$ then from the definition of the masking assignment it is clear that on $[1, 2^a]$ is same as Shoup's assignment which is strongly even-free. Also note that the masks used on $[1, 2^a]$ are totally different with the masks used in other parts. So the single man on $[c, d]$ is also single man of $T'$. Now if $c = d$ or $[1, 2^a] \cap V' = \phi$ then $T'$ is a sequential sub-tree of the tree induced by the vertices $\{i, i + 2^a, \ldots, 2^a(2^b - 1) + i\}$ for some $2 \leq i \leq 2^a$. Again along this tree the masking assignment is determined by $\beta_i$ which is strongly even-free (which is same as Shoup's assignment). So the masking assignment is strongly even-free. □

Remark : The fact that all known valid domain extensions satisfy the sufficient condition may lead to try to prove that strongly even-free is a necessary condition of valid domain extensions. If somehow we can prove that the minimum number of masks for existence of even-free is same as that for existence of strongly even-free then we can completely find out the best algorithm based on a given tree. Because, given a rooted directed tree one can find recursively the strongly even-free masking assignment with minimum number of masks. This idea will helps us to construct an optimum domain extension presented in Section 4.

4 Optimal Parallel Domain Extension

In this section we will construct valid and optimum with respect to both parallelism and key expansion (See Table 1) domain extension algorithm. Here, we will consider a rooted binary tree (not complete) instead of directed rooted binary tree. It is easy to correspond a directed binary tree to a binary tree and vice-versa. Let $T = (V, E, v_0)$ be a rooted binary tree i.e. $v_0 \in V$ and $deg(v) \leq 3$ for all $v \in V$ and $deg(v_0) \leq 2$. $v_0$ is called the root of the binary tree. To construct a valid domain extender it is enough to construct a strongly even-free masking assignment on a tree by our sufficient condition in the section 3.

Definition 3. ($i$-binary tree) $T = (V, E, v_1)$ is called $i$-binary tree if there exists a binary sub-tree $T_1 = (V_1, E_1, v_i)$ of $T$ such that $E = E_1 \cup \{v_1v_2, \ldots, v_{i-1}v_i\}$
and \( v_k \) are not in \( V_1 \) for \( 1 \leq k \leq i - 1 \). The path \( v_1v_2\ldots v_i \) is called the \( i \)-path of the \( i \)-binary tree \( T \).

See examples of \( i \)-binary tree in figure 2. A \( i \)-binary tree of size \( i \) is a sequential tree i.e. a path of length \( i - 1 \). Given two disjoint binary tree (vertex sets are disjoint) \( T_1 = (V_1, E_1) \) and \( T_2 = (V_2, E_2) \) we can concatenate as follow:

\[
T = T_1 + uv T_2 \quad \text{notation}
\]

where, \( T = (V, E) \), \( V = V_1 \cup V_2 \), \( u \in V_1 \), \( v \in V_2 \) and \( E = E_1 \cup E_2 \cup \{uv\} \). Like concatenation of two trees we can define concatenation of two masking assignments as follow (see figure 1).

Suppose, \( \psi_i \) is a \( k \)-masking assignment on \( T_i \) for \( i = 1, 2 \) then, we can define \( \psi \) a \((k + 1)\)-masking assignment on \( T_1 + uv T_2 \) where, \( \psi \) on \( T_i \) is same as \( \psi_i \) on \( T_i \) and \( \psi(uv) = k + 1 \). We will denote \( \psi \) as \( \psi_1 + uv \psi_2 \). If both \( \psi_1 \) and \( \psi_2 \) are strongly even-free then so is \( \psi \).

**Some Useful Observations:** We know that if \( \psi \) is even-free (also for strongly even-free) \( m \)-masking assignment then, \( 2^m \geq |V| \) (proved in [8]). A \( m \)-masking assignment \( \psi \) on \( T = (V, E) \) is called optimal masking assignment if it is strongly even-free and \( 2^m \geq |V| > 2^{m-1} \). So, an optimal masking assignment is a strongly even-free masking assignment whose size of image is minimum possible. One such example is given by Shoup’s [10] sequential construction. If both \( \psi_1 \) and \( \psi_2 \) are optimal then so is \( \psi \).

**Fig. 1.** Concatenation of two masking assignments

**Definition 4.** A \( m \)-masking assignment is called \((m, l, i)\)-optimal masking assignment if it is optimal masking assignment on a \( i \)-binary tree \( T \) such that \( \ell(T) = l \) and \( |V| = 2^m \).

**Fig. 2.** Some optimal masking assignments (the numbers besides edges denote the values of masking assignment)

**Theorem 3.** There exists an \((n, n+i, i)\)-optimal masking assignment if \( i \leq 2^n \).

**Proof.** Let \( f(k) = 2^k + k + 1 \) for \( k \geq 0 \). It is strictly increasing function. So, given positive integers \( n \) and \( i \) there exists a unique \( k \geq 1 \) such that \( f(k) > (n + i) \geq f(k - 1) \). We will prove the theorem by induction on \( n + i \). For small values of \( n + i \) we have shown some examples in figure 2. Now given \( n \) and
Table 1. Specific comparison of domain extenders for UOWHF 1:seq/par, 2:message length, 3:# invocation of $h_k$, 4:# masks, 5:# rounds, 6:speed-up, 7:rank in parallelism, 8:rank in key expansion

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>sequential</td>
<td>parallel</td>
<td>parallel</td>
<td>parallel</td>
</tr>
<tr>
<td>2</td>
<td>$2^t n$</td>
<td>$2^t n$</td>
<td>$2^t n$</td>
<td>$2^t n$</td>
</tr>
<tr>
<td></td>
<td>$-(2^t - 1)m$</td>
<td>$-(2^t - 1)m$</td>
<td>$(2^t - 1)n$</td>
<td>$2^t n$</td>
</tr>
<tr>
<td>3</td>
<td>$2^t$</td>
<td>$2^t - 1$</td>
<td>$2^t - 1$</td>
<td>$2^t$</td>
</tr>
<tr>
<td>4</td>
<td>$t$</td>
<td>$t$</td>
<td>$t + O(\log_2 t)$</td>
<td>$t$</td>
</tr>
<tr>
<td>5</td>
<td>$\frac{2^t}{2^t/t - l + 1} (t \equiv 0 \mod l)$</td>
<td>$\frac{2^t}{t+1}$</td>
<td>$\frac{2^t}{t+1}$</td>
<td>$\frac{2^t}{t+1}$</td>
</tr>
<tr>
<td>6</td>
<td>$1$</td>
<td>$2$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>7</td>
<td>$3$</td>
<td>$1$</td>
<td>$2$</td>
<td>$1$</td>
</tr>
<tr>
<td>8</td>
<td>$1$</td>
<td>$1$</td>
<td>$3$</td>
<td>$2$</td>
</tr>
</tbody>
</table>

$i$ we assume that the theorem is true for any $i_1$ and $n_1$ such that $i_1 \leq 2^{n_1}$ and $i_1 + n_1 < i + n$. Choose $k$ as above for these $n$ and $i$. First assume that, 

$f(k) > (n + i) > f(k - 1)$. Let $j = (n + i) - f(k - 1) \geq 1$. By induction hypothesis there is a $(k - 1, k - 1 + j, j)$-optimal masking assignment $(2^{k-1} \geq j$ as $f(k) > n + i)$. Call this by $\psi_{k-1}$. Now, $\psi_{k-1}$ is a masking assignment on a $j$-binary tree $T_{k-1} = (V_{k-1}, E_{k-1}, v_1)$ where, $\{v_1, \ldots, v_j\}$ is the $i$-path. Now take the sequential $(k - 1)$-optimal masking assignment $\psi$ on $T = (V, E)$ and define $\psi_k = \psi_{k-1} + v_j \psi_{j+1}$ where, $V = \{v_1, \ldots, v_{j+2^{k-1}}\}$. Now we can add optimal masking assignment one by one with $\psi_k$ at $v_i$'s. More precisely, let $\psi_i^l$ be a $(l - 1, l, i)$-optimal masking assignment on $T_i = (V_i, E_i, u_i)$ for $k + 1 \leq l \leq n$. Define $\psi_i = \psi_{i-1} + v_{i+k+1} u_i \psi_i^l$ recursively for $k + 1 \leq l \leq n$. Now it can be checked easily that $\psi_n$ is $(n, n + i, i)$-optimal masking assignment.

We leave with other possible case where $n + i = f(k)$ for some $k$. In this case construct a $k$-sequential optimal masking assignment $\psi$ on a sequential tree $T_k = (V_k, E_k)$ where, $V_k = \{v_1, \ldots, v_{2^k}\}$. For each $l$, $k \leq l \leq n - 1$ we have $(l, l + 1, i)$-optimal masking assignment $\psi_l$. We can concatenate $\psi_l$ with $\psi$ one by one. Finally we will have $(n, n + i, i)$-optimal masking assignment. □

One immediate corollary is given below which tells that we have a domain extension algorithm which needs $t$ many masks and $t + 1$ many rounds for $2^t$ many invocation of base hash function. Note that both number of rounds and number of keys are minimum possible.

**Fig. 3.** Construction of $(n, n + i, i)$-optimal masking assignment
Corollary 1. There exists an \((m, m + 1, 1)\)-optimal masking assignment i.e. there exists a binary tree \(T\) of size \(2^m\) with \(l(T) = m + 1\) which is minimum possible (a complete binary tree of level \(m\) has size \(2^m - 1\)) so that an optimal masking assignment \(\psi\) on \(T\) exists.

5 Conclusion

This paper has both theoretical and practical interest. Here, we will construct a UOWHF-preserving domain extension algorithm which is optimum in both key size and number of rounds. We also show that the construction given in [5] is optimum in a wide sub class of complete binary tree based algorithm. In this paper we also study how to check UOWHF-preserving property of a domain extension algorithm by just verifying a simple property called strongly even-free. It is very interesting to note that all known UOWHF-preserving domain extension algorithms satisfy the sufficient condition. So one can try to prove that the condition is a necessary condition. The sufficient condition make easy to construct a UOWHF-preserving domain extension algorithm.

References