Hash Based Digital Signature Schemes

C. Dods, N.P. Smart, and M. Stam

Dept. Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom
chris@rydertech.co.uk, {nigel, stam}@cs.bris.ac.uk

Abstract. We discuss various issues associated with signature schemes based solely upon hash functions. Such schemes are currently attractive in some limited applications, but their importance may increase if ever a practical quantum computer was built. We discuss issues related to both their implementation and their security. As far as we are aware this is the first complete treatment of practical implementations of hash based signature schemes in the literature.

1 Introduction

Digital signature schemes based on non-number theoretic assumptions are interesting for a number of reasons. Firstly they extend the number of underlying problems on which we base our security, secondly all number theoretic assumptions currently used in cryptography will be insecure if a quantum computer is ever built. Of particular interest are those schemes based solely on the security of cryptographic hash functions; such schemes are not only of historical interest but are also relatively easy to implement. See [6] for a survey of other possible signature schemes in a post-quantum computing world.

However, systems based on hash functions have a number of drawbacks. The main problem is that they usually arise in the context of one-time signatures. One-time signatures are public key signature schemes which have the property that they can only be used to sign one single message. On the other hand such schemes are usually highly efficient and easy to implement in very constrained devices, since they only require the implementation of hash functions and not any advanced arithmetic operations.

Usually, however, one requires multi-time signatures. A one-time signature can be turned into a multi-time signature in one of two ways. The first way allows one to sign as many messages as one wishes over time, however this comes at the cost of our signatures growing in size with every signature produced. This not only gives efficiency problems, but also reveals extra information since a signature size reveals information about the prior number of signatures produced.

Another way of turning a one-time signature into a multi-time signature is to fix beforehand the total number of signatures which will ever be produced and then to use a Merkle tree to authenticate the different one-time signatures.

N.P. Smart (Ed.): Cryptography and Coding 2005, LNCS 3796, pp. 96–115, 2005.

Despite the drawback of needing to fix the number of signatures beforehand it is this latter approach which is used when hash based one-time signatures are implemented.

There are a number of hash based one-time signature schemes available. Of particular interest is the Winternitz scheme which is easy to understand, relatively efficient and has been used in practical situations. Also of interest is the construction of Bleichenbacher and Maurer which is more complicated but which is theoretically the most efficient construction known.

If one wished to protect digital signatures for a long time, in particular against the possible advent of quantum computers, one is currently restricted to the use of hash based one-time signature schemes with Merkle trees. In this paper we investigate what is the most efficient construction for such schemes, by comparing the Winternitz based approach with the Bleichenbacher–Maurer approach. These are combined with a method for constructing authentication paths in Merkle trees due to Szydlo.

We present our experiments with a combination of Szydlo's methods and the Winternitz and Bleichenbacher–Maurer constructions. In addition we present various security results which generalise on the results in the literature. In particular our security proofs are built upon the assumption that any practical scheme will be implemented using only one type of hash function, hence we focus on what properties this function should have rather than requiring different properties and hence possibly different hash functions for different parts of the construction.

2 Efficient One-Time Signature Schemes

In this section we describe two schemes. The first, due to Winternitz, is a generalisation of the scheme of Merkle [16], which itself is based on the scheme of Lamport [11] (although this latter scheme is highly inefficient). In the second scheme we present the scheme of Bleichenbacher and Maurer which attempts to achieve the best possible efficiency in terms of signature size and number of hash function evaluations per bit of the message. We present a full explanation of the Bleichenbacher and Maurer construction since the original papers [4,5] concentrated on the combinatorial constructions and did not cover the issue of how this is actually used in practice to construct a one-time signature scheme.

We let f denote a hash function with domain $\{0,1\}^*$ and codomain $\{0,1\}^n$. Repeated application of f is denoted by superscripts, i.e., $f^2(x) = f(f(x))$. We assume the actual message to be signed, say M, has first been hashed via f to obtain an n-bit hash value m which will be signed by the one-time signature scheme.

In all schemes we assume that from a single n-bit key x one can derive a set of pseudo-random bit strings x_i all of length n. This can be done, for example, by using f in a key derivation function such as ANSI-X9.63-KDF [1].

Following [4] we define the efficiency of a graph based one-time signature scheme, which signs n bits, to be equal to $\Gamma = \frac{n}{v+1}$ where v is number of vertices

in the associated graph. This is essentially equal to the number of message bits which can be signed per hash function evaluation during the key generation phase. It also assumes a binary tree is used to hash the public key into a single vertex. Whilst this is a crude measure, in practice it does give some guide as to the practical efficiency of such schemes, but we shall see that it does not tell the whole truth in a practical situation.

2.1 Winternitz Scheme

The Winternitz scheme is parametrised by a value w, typically chosen to be a small power of two. For convenience, the explanation below is tailored for the case where $w = 2^t$. The generalization to arbitrary w is straightforward.

One defines $N = \lceil n/t \rceil + \lceil (\lceil \log_2 n - \log_2 t \rceil + t)/t \rceil$.

- **Key Generation:** The private key is given by x_i for $i = 1, \ldots, N$. The public key is then computed by first computing $y_i = f^{2^t-1}(x_i)$ and then computing $Y = f(y_1 \| y_2 \| \cdots \| y_N)$.
- **Signing:** The hash of the message m is split into $\lceil n/t \rceil$ segments $b_1, \ldots, b_{\lceil n/t \rceil}$ of t-bits in length (using padding with zero if necessary). Treating the m values of b_i as integers, we form the check symbol

$$C = \sum_{i=1}^{\lceil n/t \rceil} 2^t - b_i.$$

  Note, that $C \leq \lceil n/t \rceil 2^t$ and so we can write in base 2^t as $b_{\lceil n/t \rceil + 1}, \ldots, b_N$. The signature is given by $s_i = f^{b_i}(x_i)$, for $i = 1, \ldots, N$.
- **Verification:** From m generate the b_i as in the signing process and compute $v_i = f^{2^t - b_i - 1}(s_i)$. The signature is said to verify if and only if $Y = f(v_1 \| v_2 \| \cdots \| v_N)$
- **Efficiency:** The number of vertices in the associated graph is easily seen to be equal to

$$v = 2^t \cdot N = 2^t \left(\lceil n/t \rceil + \lceil (\lceil \log_2 n - \log_2 t \rceil + t)/t \rceil \right) \approx 2^t \left(n + \log_2 n \right) / t.$$

  Hence, $\Gamma \approx t/2^t$ as $n \longrightarrow \infty$. So we expect the scheme to be most efficient when $t = 2$.

2.2 Bleichenbacher–Maurer Scheme

In this section we describe the best known graph construction of a one-time signature scheme [5]. We present the construction in full generality and explain how one obtains from the combinatorial construction a one-time signature scheme, by using an analogue of the check symbol from above. This contrasts with the original description [5], where a non-constructive argument is given based on the pigeon hole principle.

The scheme is parametrised by an integer w (originally chosen to be three) and an integer B, the value of B is dependent on n and w and will be given

below. We define the scheme via a set of B blocks, each block is an array of width w and height $w+1$. There is also an additional 0th block (not included in the B above) which consists of a single row of w entries. We use the subscripts $z_{b,r,c}$ to refer to the entry in the bth block and in the rth row and cth column, where rows and columns are numbered from zero. The entries are assumed to hold values, and they are inferred from the following computational rule:

$$z_{b,r,c} = \begin{cases} f(z_{b,r-1,c} \| z_{b-1,w,(c+r) \pmod w}) & \text{If } r > 0 \text{ and } b > 1, \\ f(z_{b,r-1,c} \| z_{b-1,0,(c+r) \pmod w}) & \text{If } r > 0 \text{ and } b = 1, \\ x_{bw+c} & \text{If } r = 0. \end{cases} \quad (1)$$

To define a signature we first need to define a signature pattern. This is an ordered list of w numbers $\mathfrak{p} = (r_0, r_1, \ldots, r_{w-1})$, i.e. one height or row per column. We select the set of patterns $\mathfrak{p}$ such that

$$\bigcup_{i \in \{0,\ldots,w-1\}} \{i + j \pmod w : r_i \leq j < w\} = \{0, \ldots, w-1\}.$$

We let p denote the number of such signature patterns, which depends on the choice of w. For example when $w = 2$ one has $p = 6$, when $w = 3$ one has $p = 51$ and when $w = 4$ one has $p = 554$. To each pattern we assign a weight given by

$$wt(\mathfrak{p}) = \sum_{i=0}^{w-1} (w + 1 - r_i).$$

Note that $w^w < p < (w+1)^w$ and $wt(\mathfrak{p}) \leq w(w+1) \leq p$. For asymptotic purposes we will use $\log_2 p = \Theta(w \log_2 w)$.

- **Key Generation:** The secret key is assumed to consist of $N = (B+1)w$ values $x_0, \ldots, x_{N-1}$ which are placed in the bottom row of each block as above. The public key is produced by

$$Y = f(z_{B,w,0} \| z_{B,w,1} \| \cdots \| z_{B,w,w-1}),$$

  i.e., by hashing together the values in the top row of the final block.
- **Signing:** The (hash of the) message m is first written in base p, as $m = \sum_{i=0}^{l-1} b_i p^i$ where $l = \lceil n / \log_2 p \rceil$. We then compute the check symbol via

$$C = \sum_{i=0}^{l-1} wt(\mathfrak{p}_{b_i}) = b_l + b_{l+1} p \ldots + b_{l+l'-1} p^{l'-1}.$$

  If we set $l' = \lceil 1 + \log_p l \rceil$ then the above base p expansion of C is valid. To sign the n-bit message we will require B blocks where $B = l + l' = \lceil n / \log_2 p \rceil + \lceil \log_p \lceil n / \log_2 p \rceil \rceil$. The signature is computed by, for each $0 \leq i \leq B-1$, taking the value b_i and taking the signature pattern $\mathfrak{p}_{b_i} = (r_{i,0}, \ldots, r_{i,w-1})$. Then releasing the values of the entries $\{z_{i+1,r_{i,j},j}\}$, for $0 \leq i \leq B-1$ and $0 \leq j \leq w-1$, plus the values of $z_{0,0,j}$, for $j = 0, \ldots, w-1$.

– **Verification:** Verification proceeds by again writing m in base p. The check symbol is computed and also written in base p. The released entries are then taken, and assumed to correspond, to the patterns given by the message and the check symbol. Using the computation rule (1) the values of $z_{B,w,0}, z_{B,w,1}, \ldots, z_{B,w,w-1}$ are computed and the public key Y is checked to be equal to

$$Y' = f(z_{B,w,0} \| z_{B,w,1} \| \cdots \| z_{B,w,w-1}).$$

– **Efficiency:** We have

$$\Gamma = \frac{B \log_2 p}{1 + w(w+1)B} \longrightarrow \frac{\log_2 p}{w(w+1)} \approx \frac{\log_2 w}{w} \text{ as } n \longrightarrow \infty.$$

We expect therefore the scheme to be most efficient when $w = 3$.

3 One-Time to Many-Time

As mentioned in the introduction there are two mechanisms to convert a one-time signature into a many-time signature. The first technique, see [15–Section 11.6.3], allows an arbitrary number of signatures to be signed, however as the number of signatures grows so do the signature size and the verification time. A second approach is to use Merkle authentication trees. Here one bounds the total number of signatures which can be issued at key generation time. The benefit is that signature size and verification time are then constant.

A Merkle tree in the context of a one-time signature scheme is defined as follows. It is a complete binary tree equipped with a function h, with codomain the set of bit strings of length k, and an assignment ϕ which maps the set of nodes to the set of k-length bit strings. For two interior child nodes, n_l and n_r, of any interior node, n_p, the assignment satisfies

$$\phi(n_p) = h(\phi(n_l) \| \phi(n_r)).$$

For each leaf l_i the value of $\phi(l_i)$ is defined to be the public-key of a one-time signature scheme. From these values the values of the interior nodes can be computed. The value of $\phi(n)$ for the root node n is defined to be the public key of the many-time signature scheme. The leaf nodes are numbered from left to right and if the tree has height H, then we can sign at most 2^H messages.

For a tree of height H and the ith node l_i, we define $A_{h,i}$ to be the value of the sibling of the height h node on the path from the leaf l_i to the root. The values $A_{h,i}$ are called the authentication path for the leaf l_i.

To sign a message one signs a message with the next unused one-time signature, and then one authenticates the public key of this one-time signature using the Merkle tree. This is done by revealing the authentication path from the used leaf up-to the public key at the root. To verify the signature one verifies both the one-time signature and the authentication path. The current best algorithm for creating authentication paths in Merkle trees is described by Szydlo [23]. This algorithm creates the paths in a sequential manner, i.e., the path for leaf l_{i+1} is

created after the path for leaf l_i. Since to verify the authentication path one also needs to know the value of i, a one-time signature implemented using Szydlo's algorithm will reveal the time ordering of different issued signatures. In some applications this may be an issue.

4 Provable Security of One-Time Signatures

Prior practical constructions of one-time signatures do not come with security proofs. Hevia and Micciancio [10] prove the security of a different class of graph based signatures, to those considered above. Unfortunately, most known schemes as they are implemented in real life, do not seem to be covered by the proof directly. In particular, Hevia and Micciancio do not include the use of length-preserving one-way functions, even though these are most commonly used in proposals for practical one-time signature schemes. In theory, one could simulate a length-preserving function by composing hash function and pseudorandom generators, but this would incur a considerable performance loss. A second problem is that Hevia and Micciancio use a slightly different definition of graph based signatures. In particular, Bleichenbacher and Maurer allow the output of a hash function to be the input to several other hashes, whereas Hevia and Micciancio enforce the use of a pseudorandom generator to perform fanout. Moreover, Hevia and Micciancio pose a strong restriction on the type of signatures that is allowed. Consequently, there is a gap between what Hevia and Micciancio prove secure and what Bleichenbacher and Maurer describe as a graph based one-time signature scheme.

In this section we generalise Hevia and Micciancio's result to a broader class of graphs based on weaker, more natural assumptions, all in the standard model. However, even this broad class does not include the construction of Bleichenbacher and Maurer. To prove a security result for this latter construction we need to apply to the Random Oracle Model.

4.1 Definitions

Building signature schemes requires composing the functions in non-trivial ways and the properties of the original function need not hold for the composition. For a length-preserving one-way function, the most obvious composition is repeated application, for instance $f^2(x) = f(f(x))$. Composition for a collection of functions is done pointwise.

The properties of the functions we require in our proofs are one-wayness, collision resistance and undetectability. We intuitively describe these properties here, leaving the technical definitions to Appendix A.

- **One wayness:** A function is one-way (or preimage resistant) if it is easy to compute but hard to invert.
- **Collision resistance:** A function is collision resistant if it is infeasible to find any pair (x, x') in the domain that map to the same value.

- **Second pre-image resistance:** A function is second pre-image resistant if, given some x in the domain, it is hard to find some x' unequal to x that maps to the same value.
- **Undetectable:** A function is called undetectable if an adversary cannot distinguish the output from f with a uniform distribution over its range. In practice, this means that, given some n-bit element, an adversary is not able to tell whether f was applied to it or not.

4.2 Generalities on Graph Based One-Time Signatures

Graph based signatures, such as that of Winternitz or the Bleichenbacher–Maurer construction, are based on directed acyclic graphs. The use of these graphs for one-time signature schemes was first described by Bleichenbacher and Maurer [3]. Later a different model was given by Hevia and Micciancio [10]. There are some subtle but important differences between these two models. We give a unifying framework, mainly based on Bleichenbacher and Maurer's terminology, although we will follow Hevia and Micciancio's suggestion to swap the meaning of nodes and edges.

Let (V, E) be a directed acyclic graph. The sources and sinks of the graph correspond to the secret keys and public keys respectively. The graph denotes how the public key is computed from the secret key in the following way. The internal nodes correspond to functions, the edges to input and output values associated to those functions. There are two basic types of internal nodes: those with indegree one and those with indegree two. A node with indegree two represents a call to a hash function h. Some convention has to be used to distinguish between the first and second input for h. A node with indegree one represents a call to a length-preserving one-way function f_i. Here $i \in \{1, \ldots, w\}$ needs to be specified for the node at hand. If a node has several outgoing arcs, these all represent copies of the same output value. We assume that the graph is optimised in the sense that there are no two nodes representing the same function that have the same input (one could make an exception for hash functions, where the two possible input orderings presumably give different outcomes, but we do not).

As in any directed acyclic graph, the graph defines a partial ordering on the nodes and allows us to talk of predecessors, successors, parents, children, sinks and sources. We also fix some total ordering that is consistent with the induced partial ordering. For instance, the private key is smaller than the public key.

A signature corresponds to the release of the values associated with a certain set of edges. We will also call this set of edges a signature or sometimes a signature-set or signature-pattern. Given a set of edges S, the set of directly computable nodes is defined as the set of nodes all of whose incoming arcs are in S. The set of computable nodes is defined recursively as the set of nodes computable from the union of S with the outgoing edges of the directly computable nodes.

We will impose three conditions on a set of edges representing a signature:

1. It is verifiable: the entire public key is in the set of computable nodes.
2. It is consistent: either all arcs originating from a node are in a signature or none is. (This requirement is needed because all arcs originating from a single node represent the same output value.)
3. It is minimal: leaving out any edge would destroy verifiability or consistency.

Intuitively, the set of computable nodes from a signature denote the functions that the signer, or a forger, has to invert given the public key. Two signatures are called compatible if neither of the sets of computable nodes is a subset of the other. As a result, given one signature forging a second, compatible one still requires finding some non-trivial preimage. A signature scheme is a collection $\mathcal{S}$ of mutually compatible sets $S \subseteq V$ together with an efficiently computable collision-resistant mapping from the message space into $\mathcal{S}$.

The total number of nodes $|V|$ in the graph corresponds to the total number of operations during the key-generation, taking into account that for the leaves a call to the pseudorandom function is needed (and handing out a penalty for too many public keys).

Since all nodes have only one distinct output Bleichenbacher and Maurer identify (the value of) the output with the function-node itself. Hence, they describe a signature in terms of nodes and not edges (as a result, consistency is immediate). Moreover, Bleichenbacher and Maurer's main focus is on using only one type of one-way function. We will refer to a scheme with multiple one-way functions as a *generalised BM scheme*.

Hevia and Micciancio consider a restricted model by allowing only certain types of graphs and certain types of signature sets given the graph. Their graphs consist of two types of internal nodes: nodes with indegree two and outdegree one (representing a hash function), and nodes with indegree one and outdegree two (representing a pseudorandom generator). An important difference with our model is that fanout is only allowed by using pseudorandom generators, with the result that the arcs leading out of this node represent *different* values instead of copies of the same output value (as is the case in our model). An easy consequence of this model is that consistency is immediate.

One can transform a Hevia-Miccancio graph into a generalised BM-graph. To do this, simply replace every pseudorandom node in the Hevia-Micciancio graph with an f_1- and an f_2-node in the transformed graph, where both nodes have the same input and f_1 takes care of the first output of the generator and f_2 of the second. This transformation increases the number of nodes, which influences the efficiency measure. We believe it is reasonable to assume that a pseudorandom generator will take twice the time of a length preserving function or a hash function (on compatible input/output lengths).

Another restriction of Hevia and Micciancio's approach is that they only consider cuts for their signature sets. Given a graph, a cut consists of the arcs between the two parts of a non-trivial partition $(S, \tilde{S})$ of the set of nodes. It is a requirement that all arcs point from S to $\tilde{S}$ (or equivalently, that S is predecessor closed or that $\tilde{S}$ is successor closed). It is easy to see that there is a one-to-one

correspondence between the cuts in a Hevia-Micciancio graph and the consistent cuts in the corresponding generalised BM-graph.

As mentioned before, most proposals for one-time signature schemes do not actually fit into the Hevia-Micciancio framework. A small number of one-time signature schemes do not fit into the generalised Bleichenbacher–Maurer framework [17,19,20] either. Although the relation between the public key and the secret key can still be expressed in terms of a graph and a signature still corresponds to a set of edges, it is no longer a requirement that all possible signatures are incompatible. Instead of using a mapping from the message space to the signature space $\mathcal{S}$ that is collision resistant, a stronger mapping is used, such that it is hard to find "incompatible collisions".

4.3 Security Proofs

We generalise Hevia and Micciancio's work in three ways. We allow more diverse graphs, we allow more diverse signature patterns and our requirements on the functions with which the scheme will be instantiated are more lenient.

Hevia and Micciancio gave a proof of security for the signature schemes that fit into their model (all fanout is provided by pseudorandom generators and signatures are based on cuts) under the assumption that the hash function h is regular and collision-resistant and that an injective pseudorandom generator is used. Regularity of h is a structural requirement, as is injectivity of g. We show that these structural requirements can be replaced by more lenient and to-the-point security properties.

We also generalise Hevia and Micciancio's work by allowing more one-way functions f_i and several private and public key nodes. The latter has a slight advantage even if all public key nodes are subsequently hashed together (since then the final hash only needs to be collision-resistant). Consequently, our results directly apply to schemes using primarily a single length-preserving function (such as Winternitz' scheme), whereas in Hevia and Micciancio's work one seemingly needed to replace this with hashing the output of a pseudorandom generator (to fit in what was proven). We do however still assume that if a node has several arcs leading out, they all lead to different f_i nodes (and not to h-nodes). This opens the door for compressing the graph into something more akin to a Hevia-Micciancio graph by replacing all the f_i-nodes serviced from the same node by a single new node of indegree one, but a larger outdegree. Let $W \subseteq \{1, \ldots, w\}$ be the set of indices for which f_i was replaced. We will model the new node as representing the function $f_W : \{0,1\}^n \to \{0,1\}^{|W|n}$. Each outgoing arc corresponds to one of the f_i that has been replaced. We will refer to the graph that arises from this reinterpretation (for all nodes) as the compressed graph $(\tilde{V}, \tilde{E})$.

Finally, we relax the requirement that signatures are based on cuts. Suppose two different signatures s and $\tilde{s}$ are given (i.e., two sets of arcs). Consider the set of nodes computable from s but not from $\tilde{s}$ (this set is non-empty by definition of compatibility). Call a node in this set allowable if the intersection of the predecessor edges with $\tilde{s}$ is empty. Call s and $\tilde{s}$ strongly compatible if the set of allowable nodes is non-empty (note that, unlike compatibility, strong compati-

bility is not a symmetric relation). We require from our signature scheme that all pairs of signatures are strongly compatible. It is clear that if all signatures are cuts, the set of allowable nodes is exactly the same as the set of nodes computable from s but not from $\tilde{s}$, thus the schemes by Hevia and Micciancio form a (proper) subset of our schemes.

We show that any compressed graph based one-time signature scheme with strongly compatible signature sets is secure if h and all the relevant f_W are collision-resistant, one-way and undetectable.

Theorem 1. *Let* Pr[FORGE] *be the success probability of any forger of the signature scheme. Let* Pr[DETECT] *be the success probability for detecting using algorithms running in polynomial time with oracle access to the signature forger and let* Pr[INVERT *or* COLLISION] *be the maximum of the probabilities of finding a collision or preimage for some f_W or h, using algorithms as above. Let $|\tilde{V}|$ be the number of nodes in the compressed graph and let $\alpha = |\tilde{V}|^2$. Then the following holds*

$$\Pr[\text{FORGE}] \leq |\tilde{V}|(\Pr[\text{INVERT } \textit{or } \text{COLLISION}] + \alpha \Pr[\text{DETECT}]) \ .$$

Proof. The proof is given in Appendix B.

Our result implies Hevia and Micciancio's result [10–Theorem 1], apart from the tightness regarding detection. The proofs are similar, but we make the hybrids more explicit. This gives a shorter proof that is hopefully closer to intuition and easier to understand. The price we have to pay is that our reduction is less tight.

On the other hand the above theorem does not apply to the types of graphs used in the Bleichenbacher–Maurer construction, whilst it does apply to the Winternitz construction. The problem with the Bleichenbacher–Maurer schemes is that there might be two edges in a single signature that are comparable under the partial ordering induced by the graph. Hence to prove a result for such graphs we are required to resort to the random oracle model.

Theorem 2. *Let* Pr[FORGE] *be the success probability of any forger of the signature scheme. Let* Pr[INVERT] *be the maximum success probabilities for inversion of some f_i using algorithms running in polynomial time with oracle access to the signature forger, let* Pr[COLLISION] *be the maximum probability of finding a collision for f_i and let* Pr[DETECT] *be the maximum probability for detecting a relevant f_W, using algorithms as above. Furthermore, let $|V|$ be the number of nodes in the graph, let $\alpha = |V|^2$, and let δ be a neglible quantity that upper bounds the probability of inverting a random oracle or finding a collision for it in polynomial time. Then the following holds*

$$\Pr[\text{FORGE}] \leq |V|(\Pr[\text{COLLISION}] + \Pr[\text{INVERT}] + \alpha \Pr[\text{DETECT}] + \delta) \ .$$

Proof. The proof is given in Appendix C.

5 Practical Considerations

A theoretical analysis shows that the Bleichenbacher–Maurer and the Winternitz scheme have similar asymptotics. That is, for both schemes key generation, signature generation and signature verification take time $O(nw/\log w)$ with signature size $O(n/\log w)$. Hence any theoretical difference is hidden in the constants. Table 1 gives a theoretical comparison of the two one-time signature schemes. Here we adhere to Bleichenbacher and Maurer's assumption that both hashing two n-bit values into a single n-bit value and hashing a single n-bit value cost unit time, whereas hashing a dn-bit value for $d > 1$ takes $d - 1$ units.

Table 1. Comparison of Winternitz and Bleichenbacher–Maurer, in the number of signature bits, respectively number of hash-operations per message bit

	Winternitz				Bleichenbacher–Maurer		
	$w = 2$	$w = 3$	$w = 4$	$w = 5$	$w = 2$	$w = 3$	$w = 4$
Signature Size	1.00	0.69	0.50	0.43	0.77	0.53	0.44
Key Generation	3.00	2.52	2.50	2.58	2.32	2.11	2.19
Signature Generation	1.50	1.26	1.25	1.29	2.32	2.11	2.19
Signature Verification	1.50	1.26	1.25	1.29	1.16	1.05	1.09

Note that the numbers for key generation are the reciprocals of the efficiency measure as introduced by Bleichenbacher and Maurer. Based on these figures they concluded that their scheme with $w = 3$ fixed, is the most efficient. However, the Bleichenbacher–Maurer scheme requires more complicated code and bookkeeping, seems to rely on the assumption about the cost of hash functions, and its benefits are mainly based on key generation and to a lesser extent signature verification.

To test the relative merits of the different schemes we implemented both, and the associated Merkle trees. The various timings, in milli-seconds, can be found in Tables 2 and 3. Surprisingly we found that the Winternitz scheme was more efficient, regardless of the output size of the hash function used.

Our experiments show that the Winternitz scheme is significantly faster than the Bleichenbacher–Maurer scheme. Moreover, Winternitz with $w = 3$ has slightly cheaper key generation than $w = 4$; Winternitz signatures and verification do no take the same time, contrary to the theoretical expectation; for the Bleichenbacher–Maurer scheme, signing is slightly more expensive than keygeneration.

There are a couple of causes for the differences between the theory put forward by Bleichenbacher and Maurer and practice as we implemented it. In the theory only calls to the hash functions are counted, however in reality mapping the message to the signature set is not for free. Thus, in the Bleichenbacher-Maurer scheme, where key generation and signing take almost the same number of hash calls, signature creation also requires the additional creation of a radix

p representation of the message and mapping from this to signature sets, so one would expect signing to be slower than key generation as shown in the table. Another example is the Winternitz scheme for $w = 4$, where we implemented both a radix 4 representation using modular reductions (timings for these are within parentheses) and a direct bitwise method. The latter gave slightly faster times as expected.

Our experiments also show that the assumption on the cost of hashing different size messages is inaccurate. For instance, in the Winternitz scheme the signing and verification equally divide the number of hash calls made within the rakes. However, when signing the private keys need to be generated whereas in the verification the public keys need to be hashed together. In the theoretical model we assume that the costs for these two operations are equal, but in practice hashing many short values is clearly more expensive than hashing one long value. In the extreme and unrealistic case—where hashing one long value takes unit time, so hashing the public key is essentially ignored—the theoretical advantage of Bleichenbacher–Maurer dissipates and Winternitz is most efficient, in particular with $w = 3$.

Finally, one sees that for the Bleichenbacher-Maurer scheme the time for key generation increases from $w = 2$ to $w = 4$ (skipping $w = 3$), whereas that of signature verification decreases. This actually does follow from a theoretically more thorough analysis, where one takes into account the forbidden signature patterns. The patterns that would have been cheapest to verify are banned, hence increasing the cost of verification to just over half the cost of key generation. However, for small w this unbalancing effect weighs in stronger than for larger w.

To conclude, the theoretical advantage promised by Bleichenbacher and Maurer may not be a real advantage for real-life hash functions and practical values of n. For larger values of n, it is possible that Bleichenbacher and Maurer's scheme starts performing better, but the likely cause of the discrepancy between theory and practice carries through in the asymptotic case. It is recommended to use the Winternitz' scheme with $w = 4$, since it is very fast, easy to implement and provides short signatures.

6 Key Sizes

In this section we examine what the security reductions of our proofs and generic attacks on one-way functions imply for the recommended security parameters. In Table 4 we summarize the best known black-box attacks against different functions given different goals. Here $f : \{0,1\}^n \to \{0,1\}^n$ is length-preserving, $g : \{0,1\}^n \to \{0,1\}^{dn}$ can be a function with any expansion factor d and $h : \{0,1\}^{2n} \to \{0,1\}^n$ is a hash function.

In the security reduction, an inverter, collision finder or sample-distinguisher is described that basically runs in the same time as a potential signature forger, but with decreased success probability. By running the simulator multiple times, the probability of (say) inverting can be boosted. Assuming indepedence, this boost

Table 2. Timings (in ms) of Operations for One-Time Signature Schemes

	f	Params	Key Gen	Sign	Verify
Winternitz	SHA-1	$w = 2$	1.2	0.9	0.5
Winternitz	SHA-1	$w = 3$	1.1	0.7	0.5
Winternitz	SHA-1	$w = 4$	1.1 (1.1)	0.7 (0.7)	0.5 (0.5)
Winternitz	SHA-1	$w = 5$	1.2	0.7	0.6
BM	SHA-1	$w = 2$	1.4	1.5	0.7
BM	SHA-1	$w = 3$	1.3	1.4	0.6
BM	SHA-1	$w = 4$	1.4	1.4	0.7
Winternitz	SHA-256	$w = 2$	2.1	1.4	1.0
Winternitz	SHA-256	$w = 3$	1.8	1.2	0.9
Winternitz	SHA-256	$w = 4$	1.9 (1.9)	1.1 (1.1)	0.9 (1.0)
Winternitz	SHA-256	$w = 5$	2.0	1.2	1.0
BM	SHA-256	$w = 2$	3.3	3.5	1.9
BM	SHA-256	$w = 3$	3.2	3.3	1.7
BM	SHA-256	$w = 4$	3.5	3.6	1.8
Winternitz	SHA-512	$w = 2$	14.7	9.3	6.1
Winternitz	SHA-512	$w = 3$	13.1	7.9	5.9
Winternitz	SHA-512	$w = 4$	13.1 (13.4)	7.6 (7.7)	5.9 (6.3)
Winternitz	SHA-512	$w = 5$	14.1	8.1	7.0
BM	SHA-512	$w = 2$	22.8	23.1	12.4
BM	SHA-512	$w = 3$	21.8	21.9	11.0
BM	SHA-512	$w = 4$	23.9	24.0	11.7

Table 3. Timings (in ms) of Operations for Many-Time Signature Schemes. Note, maximum number of signatures is 2^R.

	f	Params	R	Key Gen	Sign	Verify
Winternitz	SHA-1	$w = 4$	8	300	8	1
Winternitz	SHA-1	$w = 4$	14	18700	8	1
BM	SHA-1	$w = 3$	8	350	9	1
BM	SHA-1	$w = 3$	14	22100	12	1
Winternitz	SHA-256	$w = 4$	8	500	14	2
Winternitz	SHA-256	$w = 4$	14	30800	12	2
BM	SHA-256	$w = 3$	8	800	24	2
BM	SHA-256	$w = 3$	14	56400	24	2

is linear, so to achieve an overwhelming probability of inverting we would have to invest at most $|V|/\Pr[\textsc{Forge}]$ time. Our assumption is that inverting using the signature forger takes at least as much time as a generic attack, so the time taken by a generic attack should in any case be less than $|V|/\Pr[\textsc{Forge}]$, which implies an upper bound in the probability of a signature forger expressed in $|V|$ and the runtime of a generic inverter. Since this runtime is a function of the security parameter, i.e., the length of the bitstrings that f works on, it is now possible to determine a value for the security parameter that is guaranteed to provide security (based on

Table 4. Complexity of generic attacks on one-way functions

	Classical			Quantum		
	f	g	h	f	g	h
one-wayness	2^n	2^n	2^n	$2^{n/2}$	$2^{n/2}$	$2^{n/2}$
collision-resistance	2^n	2^n	$2^{n/2}$	2^n	2^n	$2^{n/3}$
undetectability	2^n	2^n	2^{2n}	$2^{n/2}$	$2^{n/2}$	2^n

the assumption that generic attacks are the best available). These are necessarily conservative estimates: the reduction might not be tight. In concreto, finding collisions for the hash function is going to be the dominating factor in determining the minimum recommended security parameter.

In the classical world, we have that $\Pr[\textsc{Forge}] \leq |V|/2^{n/2}$, where the number of nodes $|V|$ is still to be determined. It is fair to assume that the messages will be hashed, so we need to sign $2k$-bit hash to achieve a security of at least 2^{-k}. Using the Winternitz scheme processing two bits at a time, this results in approximately $|V| = 3k$ (effectively), so we recommend a keysize $n \geq 2k + 2\lg k + 2\lg 3$.

In the quantum world it is easier to find collisions [2]. Indeed, we need $\Pr[\textsc{Forge}] \leq |V|/2^{n/3}$, where the number of nodes $|V|$ is approximately $4.5k$ using a Winternitz scheme processing two bits at a time (we need to sign a $3k$-bit hash to achieve 2^{-k}-security). The recommended keysize then becomes $n \geq 3k + 3\lg k + 3\lg 4.5$.

To conclude, the security parameter should be 1.5 times as long in the quantum setting as in the classical setting. Note that the tightness in the reduction (for instance of simulator failure) or the fact that we need to sign longer hashes in the quantum setting hardly influences the recommendations for the security parameter.

References

1. ANSI X9.63. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.* October 1999, Working Draft.
2. G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In C. L. Lucchesi and A. V. Moura, editors, *LATIN'98*, volume 1380 of *Lecture Notes in Computer Science*, pages 163–169. Springer-Verlag, 1998.
3. D. Bleichenbacher and U. M. Maurer. Directed acyclic graphs, one-way functions and digital signatures. In Y. Desmedt, editor, *Advances in Cryptography—Crypto'94*, volume 839 of *Lecture Notes in Computer Science*, pages 75–82. Springer-Verlag, 1994.
4. D. Bleichenbacher and U. Maurer. Optimal tree-based one-time digital signature schemes. *Proc. Symp. Theoretical Aspects of Comp. Sci. – STACS '96*, Springer-Verlag LNCS 1046, 363–374, 1996.
5. D. Bleichenbacher and U. Maurer. On the efficiency of one-time digital signature schemes. *Advances in Cryptology – ASIACRYPT '96*, Springer-Verlag LNCS 1163, 145–158, 1996.

6. J. Buchmann, C. Coronado, M. Döring, D. Engelbert, C. Ludwig, R. Overbeck, A. Schmidt, U. Vollmer and R.-P. Weinmann. Post-quantum signatures. Preprint, 2004.
7. S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.
8. J. Håstad and M. Näslund. Practical construction and analysis of pseudo-randomness primitives. In C. Boyd, editor, *Advances in Cryptography—Asiacrypt'01*, volume 2248 of *Lecture Notes in Computer Science*, pages 442–459. Springer-Verlag, 2001.
9. R. Hauser, M. Steiner, and M. Waidner. Micro-payments based on iKP. Technical report, IBM Research, 1996.
10. A. Hevia and D. Micciancio. The provable security of graph-based one-time signatures and extensions to algebraic signature schemes. In Y. Zheng, editor, *Advances in Cryptography—Asiacrypt'02*, volume 2501 of *Lecture Notes in Computer Science*, pages 379–396. Springer-Verlag, 2002.
11. L. Lamport. Constructing digital signatures from a one-way function. *SRI International, CSL-98*, 1979.
12. L. A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.
13. R. J. Lipton and R. Ostrovsky. Micro-payments via efficient coin-flipping. In R. Hirschfeld, editor, *FC'98*, volume 1465 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 1998.
14. M. Lomas, editor. *Security Protocols*, volume 1189 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
15. A.J. Menezes and P.C. van Oorschot and S.A. Vanstone. *CRC-Handbook of Applied Cryptography*, CRC Press, 1996.
16. R. Merkle. A certified digital signature. *Advances in Cryptology – CRYPTO '89*, Springer-Verlag LNCS 435, 218–238, 1990.
17. M. Mitzenmacher and A. Perrig. Bounds and improvements for BiBa singature schemes. Technical report, Harvard University, Cambridge, Massachusetts, 2002.
18. T. P. Pedersen. Electronic payments of small amount. In Lomas [14], pages 59–68.
19. A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In M. Reiter and P. Samarati, editors, *CCS'01*, pages 28–37. ACM Press, 2001.
20. L. Reyzin and N. Reyzin. Better than BiBa: Short one-time signatures with fast signing and verifying. In L. M. Batten and J. Seberry, editors, *ACISP'02*, volume 2384 of *Lecture Notes in Computer Science*, pages 144–153. Springer-Verlag, 2002.
21. R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In Lomas [14], pages 69–78.
22. D. R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In K. Nyberg, editor, *Advances in Cryptography—Eurocrypt'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 334–345. Springer-Verlag, 1998.
23. M. Szydlo. Merkle tree traversal in log space and time. *Advances in Cryptology – EUROCRYPT 2004*, Springer-Verlag LNCS 3027, 541–554, 2004.

A Properties of One-Way Functions

In this section we will review some well-known properties of one-way functions. Our starting point will be a collection $\mathcal{F}$ of length-regular functions acting on bitstrings, together with a probabilistic polynomial time (ppt) algorithm I that

samples elements from $\mathcal{F}$. On input a string of ones I returns a description of some $f \in \mathcal{F}$ of length polynomially related to the length of I's input. Let $m(f)$ denote the input length of the function indexed by f and let $n(f)$ denote the corresponding output length, so we can write $f : \{0,1\}^m \to \{0,1\}^n$ (omitting the dependence of n and m on f). We will assume that an efficient algorithm is given that, given f and $x \in \{0,1\}^m$ computes $f(x)$. We denote the uniform distribution over all strings of length n by U_n.

Definition 1 (One wayness). *Let $\mathcal{F}$ and I be given as above. Then $(\mathcal{F}, I)$ is one-way if and only if (iff) for all ppt adversaries A and all polynomials p eventually (there is an integer N such that for all $k > N$)*

$$\Pr[f(x') = f(x) \wedge |x'| = m : x' \leftarrow A(f, y), y \leftarrow f(x), x \leftarrow U_m, f \leftarrow I(1^k)] < \frac{1}{p(k)}.$$

Unfortunately, a function f can be one-way but f^2 not. Vice versa, a function f^2 can be one-way without f being one-way. However, if f is one-way and undetectable, then so is f^2 and similarly, if f is one-way and collision resistant, then so is f^2.

Definition 2 (Collision Resistance). *Let $\mathcal{F}$ and I be given as above. Then $(\mathcal{F}, I)$ is collision resistant iff for all ppt adversaries A and all polynomials p eventually (in k)*

$$\Pr[f(x') = f(x) \wedge x \neq x' \wedge |x| = |x'| = m : (x, x') \leftarrow A(f), f \leftarrow I(1^k)] < \frac{1}{p(k)} \ .$$

It is clear that collision-resistance implies second pre-image resistance, but the converse is unlikely to hold. Any injective function is collision-resistant and second pre-image resistant. It is also well known that collision-resistance implies one-wayness for compressing functions that are regular or for which the compression ratio can be asymptotically bounded away from 1. It can be easily shown that a function f is collision resistant if and only if f^2 is collision resistant.

Definition 3 (Undetectability). *Let $\mathcal{F}$ be a collection of functions with sampling algorithm I. Then we say $(\mathcal{F}, I)$ is undetectable iff for all ppt adversaries A and all polynomials p eventually (in k)*

$$\begin{aligned} |\Pr[b = b' | b' \leftarrow A(f, y), y \leftarrow y_b, b \leftarrow \{0,1\}, y_0 \leftarrow U_n, y_1 \leftarrow f(x), \\ x \leftarrow U_m, f \leftarrow I(1^k)] - \frac{1}{2}| < \frac{1}{p(k)} \ . \end{aligned}$$

In the literature the security requirement undetectability is usually associated with the morphological property of f being an expanding function. In this case f is known as a pseudorandom generator. We deviate from the literature by considering the two separately, especially so since undetectability is a useful property even for non-expanding functions in certain proofs. Note that naming this property pseudorandomness would have conflicted with the existing notion of pseudorandom functions.

If f is undetectable, then so is f^2 by a straightforward hybrid argument. Suppose we would have a distinguisher A_2 for f^2. Since A_2 distinguishes $f^2(U_n)$ and U_n by assumption, it must also distinguish between at least one of the pairs $f^2(U_n), f(U_n)$ and $f(U_n), U_n$. In the latter case A_2 distinguishes f, in the former case $A_2 \circ f$ does the trick. It is possible however to construct a function that is detectable applied only once, but undetectable when applied twice.

For the security of hash-chains in authentication schemes it usually suffices if the function is one-way on its iterates [8,9,18,12,21,13], but for signature schemes something potentially stronger is needed. The relevant problem has been defined by Even, Goldreich and Micali [7] (under the name quasi-inversion). We give a slightly different definition with different terminology.

Definition 4 (One-deeper preimage resistance). *Let $l_1 \geq q$ and $l_0 \geq 0$. A collection of length-preserving functions $(\mathcal{F}, I)$ is called (l_0, l_1)-one-deeper preimage resistant if for all probabilistic polynomial time adversaries A, all polynomials p and all sufficiently large k the following holds:*

$$\Pr[f^{l_0}(y) = f^{l_0+1}(x') : x' \leftarrow A(f, y), y \leftarrow f^{l_1}(w), w \leftarrow U_n, f \leftarrow I(1^k)] < \frac{1}{p(k)}.$$

If $l_0 = 0$ with ranging l_1, the definition reduces to that of one-wayness on its iterates. If the property holds for all (l_0, l_1) polynomial in k we call the function chainable. It is well known and easy to verify that one-way permutations are chainable. We give sufficient conditions for a length-preserving one-way function to be chainable.

Lemma 1. *If a family of length-preserving functions $(\mathcal{F}, I)$ is one-way, undetectable and collision resistant, then it is chainable.*

If a function f is secure for some l_0 and l_1, this does imply f to be one-way, but it does not necessitate f to be either undetectable or second-preimage resistant. These properties are needed in the proof to amplify the onewayness of f. Our result should not be confused with that of Even et al. [7] who remark that the existence of one-way functions suffices for the existence of functions that are one-deeper preimage resistant (for polynomially bounded l_0 and l_1). This is interesting, since the existence of one-way functions is not known to be sufficient for the existence of one-way permutations or collision-resistant functions in general [22].

If our goal is to prove that a function is one-way on its iterates, it suffices that f is one-way and undetectable (which is hardly surprising). If a function is one-way, undetectable and second preimage-resistant, then it is $(1, l_1)$-one-deeper preimage resistant for all (polynomially bounded) l_1.

B Proof of Theorem 1

The idea of the proof is simple and will take place in the compressed graph $(\tilde{V}, \tilde{E})$. The forger asks for a signature and then has to produce another one.

The signature scheme is set up in such a way that there are nodes computable from the forged signature that were not computable from the queried one. We will say that the forger inverts on these nodes. If we can guess beforehand which node the forger is going to invert, we can put the value we wish to invert in that node. The problem is that this action changes the distribution of the public key (and the queried signature), so we need to show that an adversary capable of spotting the difference between the two distribution can be used to detect the application of some combination of functions. Using hybrids, this is fairly easy to show. We are helped in this task by the fact that if an adversary inverts a node, the predecessor edges are completely unknown (this is where we need that the signatures are partitions), so the simulator does not need to know or have consistent values for these edges, and by the fact that all fanout is the work of pseudorandom generators.

Let y_h be generated according to $h(U_n)$ and y_W according to $f_W(U_n)$, for $W \subseteq \{1, \ldots, w\}$. We will describe three games, show that a success in Game 3 leads to either a collision (for h or some f_W) or a preimage (for some y_W or y_h) and that if the success probability of Game 3 is negligible but an efficient signature forger exists, some function is detectable.

Game 1. The original game. The simulator creates the public key as in the original key generation and feeds it to the forger. Because the simulator knows the entire secret key, he can easily provide any one-time signature the forger requests. The success probability of this game is by definition $\Pr[\textsc{Forge}]$, so

$$\Pr[\textsc{Success}_1] = \Pr[\textsc{Forge}] \ . \tag{2}$$

Game 2. As Game 1, but with the modification that the simulator picks an internal node (the target) at random and aborts if the forger does not invert on this node or if the forger's signature query includes an arc predecessing the target node. Since the actions of the simulator are independent of the chosen target, the forger behaves as in Game 1. The success probability is at least $\Pr[\textsc{Success}_1]/|\tilde{V}|$, which, combined with (2), gives

$$\Pr[\textsc{Success}_2] \geq \Pr[\textsc{Forge}]/|\tilde{V}| \ . \tag{3}$$

Game 3. As Game 2, but with the following modification to the key generation. The simulator replaces the output value of the targeted node with whichever he wishes to invert (obviously, this value should be from a suitable distribution for the node at hand). The simulator also replaces all arcs leading out of the predecessor graph of the target node with random values. The simulator recomputes the public key.

The simulator can produce for all arcs a value, except for those within the predecessor graph of the target. If the target node is allowable, the adversary's query will not contain any arcs within the predecessor graph (by definition). Thus for allowable targets, the simulator will be able to answer the signature query correctly.

We claim the following, from which the theorem follows:

$$\varepsilon = \Pr[\textsc{Success}_2] - \Pr[\textsc{Success}_3] \leq |\tilde{V}| \Pr[\textsc{Detect}]$$
$$\Pr[\textsc{Success}_3] \leq \Pr[\textsc{Invert or Collision}]$$

First consider the case in which Game 3 is successful. For the targeted node the forger has thus supplied a value y' and its preimage. If $y' = y$, we have found a preimage to the target and are done. On the other hand, if $y' \neq y$, it is possible to find a collision somewhere in the path to the root.

Now consider the difference ε in success probability between Game 2 and Game 3. Remember that a full ordering on the nodes of the graph is fixed consistent with the partial ordering induced by the graph.

Game 3'. Consider Game 3', parametrised by an internal node t of the graph (we also allow a node at infinity). The simulator picks a random internal node: if it is less than t it plays Game 2, otherwise it plays Game 3. If t is the minimal element, Game 3' is identical to Game 3, whereas the other extreme, t is the node at infinity, corresponds to Game 2. A standard hybrid argument implies the existence of a node t^* such that the difference in success probability between Game 3' with $t = t^*$ and with t the next node (in the full ordering) is at least $\varepsilon/|\tilde{V}|$.

Game 3". Now define Game 3", parametrised by an internal node r that is a predecessor of t^*. The simulator picks an internal node in the full graph at random. If it is less than t^* the simulator plays Game 2, if it is larger than t^* it plays Game 3. If it equals t^*, the simulator assigns random values to all edges emanating from a node smaller than r and recomputes the public key.

If r is minimal, then Game 3" equals Game 3' with $t = t^*$, but if $r = t^*$, then Game 3" equals Game 3' with t the next node. As a result, there must be some node r^* such that the difference in success probability between Game 3" with $r = r^*$ and r being the next node is at least $\varepsilon/|\tilde{V}|^2$. The only difference between these two instantiations of Game 3" can be traced back to node r^*. If $r = r^*$, it is computed based on an input computed uniformly at random. If r is the next node, then the output of r^* will be made uniformly random by the simulator. This finally leads to a distinguisher for the function that node r^* represents.

C Proof of Theorem 2

The idea of the proof is similar to the previous proof. If we can guess which node the forger is going to invert, we can put the value we wish to invert in that node, provided it is an f_i-node. An adversary that does not invert any f-node, is forced to invert the random oracle, which can only be done with negligible probability given only a polynomial number of oracle queries.

Let y_i be generated according to $f_i(U_m)$ for $i = 1, \ldots, w$. We will describe three games, show that a success in Game 3 leads to either a collision (for some f_i or the random oracle), or an inversion of f_i and that the success probability of Game 3 cannot be negligible if an efficient signature forger exists and f_W are undetectable.

Game 1. The original game, where calls to the hash function are replaced by oracle queries. The simulator creates the public key as in the original key generation and feeds it to the forger. Because the simulator knows the entire secret key, he can easily provide any one-time signature the forger requests. The success probability of this game is by definition $\Pr[\textsc{Forge}]$.

Game 2. As Game 1, but with the modification that the simulator picks an internal node at random and aborts if the forger does not invert on this node. Although the simulator makes up his mind for several oracle queries, he does not make the calls anymore (the forger can of course query based on the public key and signature he receives). Since the actions of the simulator are independent of the node he picked, the forger behaves as in Game 1. The success probability is at least $\Pr[\textsc{Success}_1]/|V|$.

Game 3. As Game 2, but with the modification that if the simulator has picked an f_i-node it replaces its output value with y_i. Moreover, it looks at f_i's predecessor nodes until it hits an h-node. The simulator replaces the output of the f_i-children of the f_i-predecessors in a direct line with random values and recomputes the public key. The only arcs for which the simulator does not know a value are the arcs directly leading out of the f_i-predecessors in a direct line of the target node. From any of these arcs the target is computable and hence, if the forger inverts on the target, by minimality of signature patterns these arcs cannot have been queried.

It is easy to see that an adversary that distinguishes between Game 2 and Game 3 also distinguishes between a correct evaluation of the tree consisting of the f_i-predecessors in a direct line of the target together with their f_i children (which will include the target itself) and an equal number of independently chosen uniform random values. ¿From the result in the previous section it already follows that if all the f_W involved are undetectable, this probability is negligible. In fact, $|\Pr[\textsc{Success}_3] - \Pr[\textsc{Success}_2]| \leq \alpha \Pr[\textsc{Detect}]$.

If the forger inverts some f-node, the above leads either to a one-deeper inversion, a collision on f or a collision on the random oracle. The probability of finding a collision on the random oracle in polynomial time is negligible.

If the forger does not invert any f-node, he will have to invert an h-node, corresponding to inverting the random oracle. Since it is an inversion, not both edges can be known from the signature-query allowed to the forger. So one edge is still unknown. Presumably, the simulator already knows the value of this edge and copies of it might be used elsewhere in the signature scheme. However, no information can be known to the adversary if it is solely used as input to other h-nodes (by virtue of the random oracle modelling these nodes). Moreover, it cannot be the input to an f-node, since in that case the adversary would also invert an f-node, which it does not (by assumption). Hence, the adversary only has a negligible guessing probability in succeeding.