Hybrid classical-quantum digital signature schemes based on k-walk/k-path problems on weighted graphs

Nimish Mishra,[1,*] Bikash K. Behera,[2,†] and Prasanta K. Panigrahi[2,‡]

[1] *Department of Computer Science and Engineering, Indian Institute of Information Technology, West Bengal, India*

[2] *Department of Physical Sciences, Indian Institute of Science Education and Research Kolkata, Mohanpur 741246, West Bengal, India*

We propose a new approach in creating digital signature schemes, wherein quantum properties are leveraged to do away with publishing the public/verification key in the signing procedure. This removes a potential forger's access to the verification oracle. Our scheme is based on the established hardness of k-walk/k-path problems on weighted directed graphs, with explicit algorithms for concrete instantiation. The scheme is provable to be intractable for classical computers within exponential operations, with a conjectured improbability of a quantum attack. We provide algorithms for instantiating arbitrary boolean mapping between qubits, as well as for the implementation of such mappings.

Keywords: Digital Signatures, k-walk/k-path problems

I. INTRODUCTION

A. Digital signatures

Digital signatures are digital counterparts of actual signatures that aid the recipient to verify the received document is actually from the authorized sender. The only underlying requirement is that there is absolutely no way for an eavesdropper to forge sender's signatures that can evade the recipient's verification. Concretely, for fixed finite sets of valid messages M and of valid signatures S, a general digital signature scheme consists of three algorithms: $KeyGen$, $Sign$ and $Verify$ wherein,

- $KeyGen$ is a probabilistic, polynomial-time algorithm that on input the security parameter outputs a pair of keys: public key K_{public} and a private key $K_{private}$ which are unique for every specified user of the signature protocol.

- $Sign(d, K_{private})$: a probabilistic polynomial time algorithm which, given an arbitrary document d, creates the signature D^{sig} for the document.

- $Verify(d, D^{sig}, K_{public})$: a probabilistic polynomial time algorithm that delivers a binary decision whether D^{sig} is the signature of d with $K_{private}$. $K_{private}$ isn't accessible to the recipient, but this decision can still be made by using the supplementary key to $K_{private}$, i.e. K_{public}.

Given that any eavesdropper can access K_{public}, d, D^{sig}, the digital signature scheme needs to satisfy an important properties:

- Unforgeability: Except with negligible probability, there is no way, even when given access to n samples of document-signature pairs, to generate *any* key (may be exactly $K_{private}$ or other key like $K_{private}$) that creates a valid signature D^{sig}_{forged} using K_{public} for arbitrary document d. The signature scheme fails if the forged signature is accepted by $Verify(d, D^{sig}_{forged}, K_{public})$.

- Non-repudiation: Except with negligible probability, the signer should not be able to refuse signing a legitimate signature.

Recent advancements in quantum algorithms have provided a shift in the interests of the research community from number theoretic cryptographic primitives to post-quantum primitives like lattice-based cryptography. The latter is usually notorious for not well adapted to practical requirements (large signature sizes, need to sample from Gaussians and likewise, intense matrix operations and so forth).

In this paper, we propose a different direction from the normal and leverage quantum-inspired algorithms to probe schemes that might be secure and practical with large scale quantum computers. The notable change in our scheme is doing away with the public/verification key thus stripping a potential forger of any access to the verification black box.

B. Graph problems

In this sub-section, we consider formulations of two graph problems, and their associated complexities [1]. We use results from this section to establish the security of our scheme.

Definition 1 ***(k-walk problem)****: Given a digraph $G = (V, E)$, vertex set V, and edge set E, with a length function $w: E \to N$, a pair of nodes $s, t \in V$, and an integer $k \geq 0$, does there exist in G a walk from s to t of length k?*

Lemma 1 *k-walk problem is NP-complete when G is a directed graph, an undirected graph, or a directed acyclic graph.*

* nimish_bt18@iiitkalyani.ac.in
† bikash@bikashsquantum.com
‡ pprasanta@iiserkol.ac.in

Lemma 2 *k-walk problem is polynomial time solvable on unweighted graphs and for $k = |V|^{\mathcal{O}(1)}$. Equivalently, k-walk problem is polynomial time tractable on a weighted graph with unit edge weights.*

From lemma 2, it is straightforward that for k being polynomially bounded in the size of G, the problem becomes tractable on undirected graphs.

Definition 2 ***(k-path problem)**: Given similar information as in 1, does there exist a path of length k?*

Lemma 3 *k-path problem is tractable in time of order $\mathcal{O}(2^k)$ ignoring small multiplicative factors. [2, 3].*

In all the following sections, $\mathcal{D}(u, v)$ for any two vertices of the graph representation considered later represents the hamming distance between u and v, or concretely, the number of 1s in $(u)_2 \oplus (v)_2$ for the usual binary representation of u and v and the usual XOR operation on them.

C. Related works

A key difference between our scheme and the post-quantum lattice based schemes described in [4] (as well as their early number theoretic counterparts) is that we do away with the concept of publishing a verification/public key. Verification proceeds by a shared secret between communicating parties. This evidently reduces the implementation scope of our scheme from a general purpose digital signature scheme to a scheme for communication between authenticated parties, while it strips from potential forger the access to the verification black box. As with the schemes in [4], a forger in our setting has no means to verify a forged signature (as well as responses from the signer oracle to the forger's queries) because there is no verification/public key the forger has access to. The only attack the forger is left with is to reconstruct the signing secret shared between the authenticated communication parties from the responses received from the black-box signer oracle. The question of introducing a public key and investigating the trade-off between the forger's advantage and a general purpose signature scheme for communication between (preferably unknown to each other) parties is something we leave to future work.

On the other hand, fully quantum digital signatures [5] have still a long way to go to be realizable in modern networks. Our scheme is *quantum* in the sense that it employs quantum circuits to ensure true randomness. Communication and associated tasks are classical. A complete classical counterpart would assume true randomness to be provided by some black box; we, however, look for ways of constructing that black box by utilizing quantum phenomena.

TABLE I: Arbitrary n-bit map. Subscripts 2 and 10 represent the binary and the decimal representation respectively.

Input	**Output**
0_{10} $(000....00_2)$	m n-bit output for $m \leq n$
1_{10} $(000....01_2)$	m n-bit output for $m \leq n$
2_{10} $(000....10_2)$	m n-bit output for $m \leq n$
.	.
.	.
.	.
$(2^n - 1)_{10}$ $(111....11_2)$	m n-bit output for $m \leq n$

II. SCHEMES

We begin by considering an arbitrary n-bit map as in Table I. Each n-bit input is represented in binary form, and mapped to m different output bit-strings. It is straightforward to implement this abstract idea using quantum superposition. Consider $|i\rangle$ to be one of the 2^n input states, represented by n qubits according to the following preparation: jth qubit is prepared in $|0\rangle$ state if jth bit in binary representation of i is 0, else the former is prepared in state $|1\rangle$. Let $\{i_1, i_2, i_3, ..., i_m\}$ be the set of the m output states i maps to. The mapping can then be represented as a function f:

$$f(|i\rangle) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |i_j\rangle$$

where the probability of obtaining a state $|i_l\rangle : 1 \leq l \leq m$ upon measuring $f(|i\rangle)$ is $\frac{1}{m}$.

We now convert the representation to a weighted, directed graph. Consider a digraph $G = (V, E)$ with V being the set of vertices and E being the set of edges. Let the distinct 2^n states (given in the input column of Table I) constitute V. Intuitively, $E = \{(u, v) : u, v \in V; v = f(|u\rangle)\}$ or there exists a directed edge from u to v iff $|v\rangle$ is a probable output upon measuring $f(|u\rangle)$. Let the weight function be given as $w : E \to \mathbb{N}$ such that $w(u, v) = \mathcal{D}(u, v)\ \forall (u, v) \in E$. We now define a walk on this graph representation of Table I.

Definition 3 ***(k-walk of graph representation of I)**: k-walk on the considered graph representation of Table I is given as $f^l(|i\rangle)$, or l applications of the mapping function f.*

$$f^l(|i\rangle) = f(f(f(...f(|i\rangle)))) \quad l \text{ times}$$

such that the sum of weights of the l edges traversed equals k.

Regarding implementation, it is straightforward to see that we prepare the input qubit state $|i\rangle$, apply the mapping function f, measure, and prepare the next input

qubit state according to the output bit-string. The process is repeated until the sum of edge weights in the walk equals k.

For sake of clarity, we defer discussion of instantiating Table I and determining the mapping function f to subsections of the Appendix. We note that instances of Table I are represented as strings of n-qubit $AND(.)$ and n-qubit $H(.)$ functions (where the latter represents the controlled Hadamard function) whose individual results are bitwise-OR ed to obtain the final result. Our main assumption is that the communicating parties are able to *at most once* share this string representation of Table I and some additional information (a suitable data structure like a dictionary) related to it, without being leaked to a potential adversary.

A. k-walk based

By nature of the algorithm 5, we note that every n-qubit input is mapped to m distinct n-qubit states if $c = \frac{m}{2}$. We focus on the case where $m < n$ (equality is straightforward). It is to be noted from the algorithm 5 that $c = \frac{m}{2}$ arbitrary bit positions are selected uniformly at random and Hadamard is applied to them. The remaining $n - \frac{m}{2}$ bit positions are assigned from {0, 1} arbitrarily. Application of Hadamard on $\frac{m}{2}$ bit positions results in m distinct possible output states. Concretely, consider the n-qubit input state $|i_1 i_2 i_3 ... i_n\rangle$ and let j_1, j_2, j_3, ..., $j_{\frac{m}{2}}$ denote the bit positions where Hadamard has to be applied. Let p_1, p_2, p_3, ..., $p_{n-\frac{m}{2}}$ be the remaining bit positions that are assigned arbitrarily from {0, 1}. Then we get m equally probable output states with each of j_i attaining 0 or 1 with equal probability. We also note the instantiating algorithm outputs a suitable data structure (we consider dictionary here for its constant time look-ups) that stores, against each input state, a list storing p_1, p_2, p_3, ..., $p_{n-\frac{m}{2}}$. We leave considerations about storage size for later analysis.

A high level overview of the k-walk scheme is presented in Table II.

TABLE II: k-path based scheme overview. *cum_hamming* represents the cumulative hamming distance. Refer algorithm 2

Sender	**Recipient**
Input: Message *mess*, dictionary d and value of k. Initialise a quantum circuit with n length quantum and classical registers.	
Signing: initialiseStartingState(*mess*); algo. 1 signMessage(*mess*, k, d) algo. 2, receive signature s and ending state e. **Send**: (*mess*, e, s, *cum_hamming*)	
	Verification: verify(*mess*, e, s, *cum_hamming*, d) Refer algo. 3

Algorithm 1: initialiseStartingState(message)

Result: Prepares the input quantum state based on n-bit input message. Requires $\Theta(n)$ comparisons.

```
if message[0] == 1 then
|apply X gate on 0th qubit
end
if message[1] == 1 then
|apply X gate on 1st qubit
end
. . .
if message[n] == 1 then
|apply X gate on nth qubit
end
```

The signing algorithm in 2 signs a message by repeating the function mapping f and appending the output state to the signature to be sent. We modify the definition 3 slightly to make the signing procedure easier for the signer.

Algorithm 2: signMessage(message, k, d)

Result: Signs the message. Returns signature, ending state, and cumulative hamming distance.

```
signature = ""
initialiseStartingState(message)
input_state = message
step = 0
while step ≤ k do
|fixed_index_list = d[input_state]
|// run circuit and obtain n bit output in output_state
|step += D(input_state, output_state)
|index = 0
|while index ≤ length(output_state) do
|if index ∈ fixed_index_list then
|output_state = output_state.replace(index, "x")
|end
|end
|signature = signature + output_state
|// clear circuit and ready for next iteration
|initialiseStartingState(output_state)
|input_state = output_state
end
return (signature, output_state, step)
```

Definition 4 ***(Alternative k-walk of graph representation of I)****: k-walk on the considered graph representation of Table I is given as $f^l(|i\rangle)$, or l applications of the mapping function f.*

$$f^l(|i\rangle) = f(f(f(...f(|i\rangle)))) \quad l \text{ times}$$

such that the sum of weights of the l edges traversed equals ***at least*** *k.*

We note from table II that k is a security parameter chosen to make the scheme unforgeable by an adversary. We claim that if the scheme is unforgeable for k, it should be unforgeable for values not much greater than k (precisely for values of the order $\mathcal{O}(k + max(\mathcal{D})$ with $max(\mathcal{D})$ denoting the maximum Hamming distance in the function mapping f). The reason why we do not fix the length of walk as k since then it will be equally difficult for the signer to find a suitable walk. On the other hand, here the signer is given the freedom to continue searching until they encounter a walk of cumulative weight at least k (say for $k^{'} > k$), and broadcast their result. The adversary will then have to forge a walk of $k^{'}$.

A short discussion on how, given the randomness of f, can a verifier recreate the entire walk (and hence the signature) from the signer is now due. It is to be mentioned from our earlier discussion at the start of subsection II A that the dictionary d in table II contains, against each input combination, a list of indices which were **not** Hadamarded in designing the function mapping f. This implies the recipient when runs the circuit upon the input state $|i_1 i_2 i_3 ... i_n\rangle$, they will always get the same bit values for indices p_1, p_2, p_3, ..., $p_{n-\frac{m}{2}}$. We claim this information inherent to the nature of f and thus unnecessary for the sender to actually send. This is the motivation behind *replacing "x"* in the algorithm 2 whenever such an index comes. The important information the verifier needs to know in order to recreate the walk are the bit positions j_1, j_2, j_3, ..., $j_{\frac{m}{2}}$ which truly control which of the m equally-probable states the verifier achieves. The sender thus sends this information in order for the verifier to reconstruct the initial walk and verify the signature accordingly.

A sample signature shall then derive symbols from $\{0, 1, x\}$ (where each symbol is represented by 2 bits, say 00 for 0, 11 for 1, and 01 for x). The verifier (as well as an adversary) reads the received bit pattern in pairs and reconstructs the symbolic representation of the signature (in terms of 0, 1, and x). The length of the symbolic representation of the signature is of size $\Theta(g(k + max(\mathcal{D})).n^{'})$ (recall k is still a reasonable constant and $n^{'}$ is the length of the message), while communication complexity will be $\Theta(g(k + max(\mathcal{D})).n^{'}.2)$ ignoring additional transmissions for the ending state ($\Theta(n^{'})$) or for the integer *steps* ($\Theta(1)$). The signing procedure in algorithm 2 runs in time $\mathcal{O}(g(k).n^{'})$ (if both are within reasonable limits, we expect the procedure to run efficiently). In this discussion and all discussions hereafter, we refer $g(k)$ to output l (refer definition 4).

The verifier verifies the signature as follows in algorithm 3.

The algorithm 3 is similar to that of the sender in sense that it tries to recreate the walk by the sender, and then compare the two walks/signatures. The main portion aiding recreation of the walk is enclosed within #. Let us recall the signature consists of symbols from {0, 1, x}, where x denotes the indices which are fixed by

Algorithm 3: verify(message, ending_state, signature, steps, d)

Result: Verify the received signature. ACCEPT or REJECT.

```
recreated_signature = ""
initialiseStartingState(message)
hamming_distance = 0
input_state = message
step = 0
signature_index = 0
while not(step == steps) do
|fixed_index_list = d[input_state]
|// run circuit and obtain n bit output in output_state
|step += 1
|// ##################
|i = 0
|while i ≤ length(output_state) do
|if signature[signature_index] == 1 then
|output_state[i] = output_state[i] OR 1
|end
|if signature[signature_index] == 0 then
|output_state[i] = output_state[i] AND 0
|end
|signature_index ++
|i++
|end
|// ##################
|hamming_distance += D(input_state, output_state)
|index = 0
|while index ≤ length(output_state) do
|if index ∈ fixed_index_list then
|output_state = output_state.replace(index, "x")
|end
|end
|recreated_signature = recreated_signature +
|output_state
|// clear circuit and ready for next iteration
|initialiseStartingState(output_state)
|input_state = output_state
|end
|if recreated_signature == signature and
|output_state == ending_state and
|hamming_distance == steps then
|ACCEPT
|end
|else
|REJECT
|end
```

design of f and the others denote the specific results of the indices which were Hadamarded. The verifier, since using the same mapping f as the signer, does not need to care about x. The only difference, in usual quantum measurement, arises in the values of the indices that are Hadamarded. The verifier thus bit-masks, hence *forces*, even the Hadamarded indices to capture exactly the same value as the signer did. The verifier is thus able to recreate the entire walk as the signer. As an extra layer of surety, the verifier checks if the recreated walk has the same cumulative Hamming distance as that of the signer. The verification algorithm runs in time similar to that of the sender (differing by small constant multiplicative factors due to introduction of the bit-mask loop).

B. k-path based

k-path is generally a harder problem than k-walk due to the additional requirement to ensure distinct vertices in the walk. For the k-path based algorithm, we make changes in the signer algorithm 4 wherein we add a list tracking states already traversed and mechanism to rollback if needed.

Algorithm 4: signMessage(message, k, d)

Result: Signs the message. Returns signature, ending state, and cumulative hamming distance.

```
signature = ""
initialiseStartingState(message)
input_state = message
used_states = []
step = 0
while step ≤ k do
fixed_index_list = d[input_state]
used_states.append(input_state)
// run circuit and obtain n bit output in output_state
######## new code##########
if output_state in used_states then
rollback
end
######## new code##########
step += D(input_state, output_state)
index = 0
while index ≤ length(output_state) do
if index ∈ fixed_index_list then
output_state = output_state.replace(index, "x")
end
end
signature = signature + output_state
// clear circuit and ready for next iteration
initialiseStartingState(output_state)
input_state = output_state
end
return (signature, output_state, step)
```

We set an upper bound $\mathcal{O}(1)$ on the number of rollbacks, and abort/restart if rollbacks exceed this count. The verification algorithm remains the same since it simply recreates the path from the signer. Additional layer of security can be placed if the verification algorithm also checks whether the recreated walk is a path. Since modifying algorithm 3 to this end is straightforward, we skip the details.

III. SECURITY

A general sense of security of digital signature schemes is the probability with which a polynomial time forger, using the signer as an oracle, forges signatures of previously unseen messages [6]. Concretely, for n queries to the signing oracle with messages m_1, m_2, m_3, ..., m_n, the forger is given valid signatures s_1, s_2, s_3, ..., s_n and asked to create signature for a new message m_{n+1}. Additionally, *strong unforgeability* implies the forger should not be able to come up with a different signature for any m_i for $i \in \{1, 2, 3, ..., n\}$.

Security reductions take the following general form [7]: an adversary $\mathcal{A}$ who can break the scheme can be used to build an algorithm $\mathcal{B}$ that solves an underlying hard problem. Tightness of the reduction is determined by the extra work $\mathcal{B}$ does in order to convert $\mathcal{A}$ to an algorithm solving the problem.

Theorem 1 ***(Reduction)****: An adversary $\mathcal{A}$ who breaks the k-walk (k-path) based signature scheme as described in the previous section has to efficiently solve k-walk (k−path) problem on the weighted directed graph representation of I. Concretely, $\mathcal{A}$ can be converted to an algorithm $\mathcal{B}$ that solves k-walk (k-path) problem on weighted directed graphs.*

Proof: By construction of schemes described in subsections II A and II B, an adversary/forger looking to imitate a signer needs to create a walk (or path) as described in definition 3. Concretely, for a query m to the signing oracle, the forger gets in response a tuple *(message/starting state, signature, ending state, cumulative Hamming distance)*. In the sense of *strong unforgeability*, the forger is faced with the task to create an alternative signature (precisely an alternative walk/path) from starting state to ending state of the specified cumulative Hamming distance. Definitions 1 and 2 and the lemmas therein establish the hardness of finding such walks/paths.

We note that the adversary $\mathcal{A}$ rather does *more "work"* than $\mathcal{B}$, and since $\mathcal{B}$ is intractable in polynomial time, $\mathcal{A}$ is intractable in time polynomial in the size of the graph (for $k = \mid V \mid^{\mathcal{O}(1)}$). The notion of *more "work"* is established in the rest of the section.

Claim 1 *The forger $\mathcal{A}$ has no access to the mapping function f and the dictionary d (as used in subsections II A and II B).*

We had stated before starting subsection II A that our main assumption is a *one-time* secure communication of information required to construct f and d. Without

knowing either of these, the adversary $\mathcal{A}$ is left to determine f (and thus the weighted graph representation of I) from the signer oracle's response to the forger's queries before algorithm $\mathcal{A}$ can be used to construct algorithm $\mathcal{B}$. We now analyse the *more "work"* for $\mathcal{A}$ to reconstruct f. For the subsequent portion of this section, consider a message/starting state μ of length n, a walk of l steps (precisely l edges) that has cumulative Hamming distance at least k (refer definition 4).

Definition 5 *Given a message μ of length n, consider distributions $\mathcal{F}$ and $\mathcal{U}$. Distribution $\mathcal{F}$ outputs a signature of length ln based on the original mapping function f starting from initial quantum state μ; distribution $\mathcal{U}$ outputs a signature uniformly at random from $\{0, 1, x\}^{ln}$*

Lemma 4 *To a forger, $\mathcal{F}$ and $\mathcal{U}$ are indistinguishable.*

Proof: As stated before, consider the n-qubit input state $|i_1 i_2 i_3 ... i_n\rangle$ to be the quantum representation of μ and let j_1, j_2, j_3, ..., $j_{\frac{m}{2}}$ denote the bit positions where Hadamard has to be applied according to f. Let p_1, p_2, p_3, ..., $p_{n-\frac{m}{2}}$ be the remaining bit positions that are assigned arbitrarily from $\{0, 1\}$. Then we get m equally probable output states with each of j_i attaining 0 or 1 with equal probability. Application of $f(|\ \mu\rangle)$ leads to one of these m states as the output. In the output, according to algorithm 2, the *fixed* bits p_1, p_2, p_3, ..., $p_{n-\frac{m}{2}}$ are replaced by 'x' and the bits j_1, j_2, j_3, ..., $j_{\frac{m}{2}}$ are left untouched. The forger $\mathcal{A}$ recognises j_1, j_2, j_3, ..., $j_{\frac{m}{2}}$ to be the bit positions that are Hadamarded, and thus correctly guesses the m equally-probable states from μ. Now the rest of the bit positions p_1, p_2, p_3, ..., $p_{n-\frac{m}{2}}$ are fixed for the signer, who is a black box to the forger and thus the forger has no idea about their true value. The forger is thus left to analyse all possible values these 'x' can take, thus leading to $2^{n-\frac{m}{2}}$ states, which in addition to the states obtained by all combinations of j_1, j_2, j_3, ..., $j_{\frac{m}{2}}$ lead to 2^n states from $|\ \mu\rangle$, which is statistically equivalent to $\mathcal{U}$.

Consider the simple case of $n = 4$. Let $|\ \mu\rangle = |\ 0000\rangle$ map to states $|\ \mathbf{10}0\mathbf{0}\rangle$, $|\ \mathbf{10}0\mathbf{1}\rangle$, $|\ \mathbf{11}0\mathbf{0}\rangle$, $|\ \mathbf{11}0\mathbf{1}\rangle$. We note bit positions 1 and 3 are fixed, while 2 and 4 are Hadamarded. Say the forger $\mathcal{A}$ receives $x0x1$. $\mathcal{A}$ determines the fixed and the Hadamarded bit positions and quickly deduces signer's f is such that $|\ 0000\rangle$ maps to $x0x0$, $x0x1$. $x1x0$, and $x1x1$. Now for the remaining bit positions, there is no way for $\mathcal{A}$ to deduce anything except to evaluate all possible combinations of the two 'x', leading to $\{0, 1\}^4$. It is straightforward to see that had the forger been given a response uniformly at random from $\{0, 1, x\}^4$, the same distribution $\{0, 1\}^4$ would be obtained. $\mathcal{A}$ thus can not deduce whether responses from signer oracle came from $\mathcal{F}$ or from $\mathcal{U}$.

Lemma 5 *With probability depending upon the value of k (and thus l) and m, $\mathcal{U}$ outputs a string contained in $\mathcal{F}$. For reasonable $m < 2^n$, this probability $\ll 1$.*

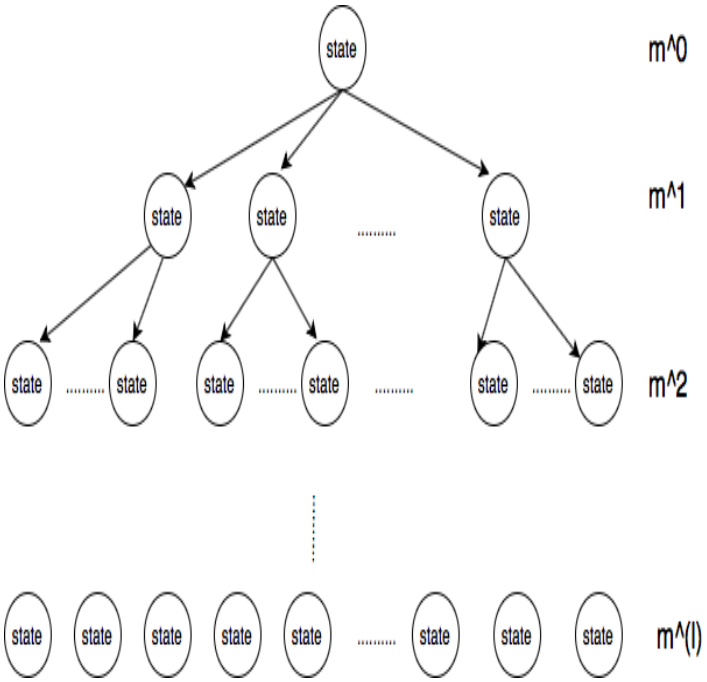

FIG. 1: l step walk using f with branching factor m.

Proof: $\mathcal{U}$ has, by definition, 3^{ln} combinations from which one is outputted uniformly at random. To analyse $\mathcal{F}$, consider the branching factor m, concretely, using f, each input state maps to m output states as in table I. A l step walk using f starting from μ can be represented as in figure 1.

Evidently, f running for l steps results in m^l distinct walks, one of which is outputted by $\mathcal{F}$. By standard probability argument, the probability of $\mathcal{U}$ outputting a walk contained in $\mathcal{F}$ is equivalent to the probability of $\mathcal{U}$ outputting a walk in figure 1. The probability of this happening, for a l step walk, is $m^l/3^{ln}$. For n length input in f, the maximum value of m is 2^n. The probability is thus upper bounded by $(2/3)^{nl}$.

$$Pr[u \leftarrow \mathcal{U} : u \in \mathcal{F}] \leq (2/3)^{ln}$$

.

Claim 2 *The forger has to try and decode the 'x' received in the signature.*

The claim is well-founded in the sense that when we try to analyse the options available to the forger where decoding 'x' isn't required, we find such options limited. By definition, a forger $\mathcal{A}$ is given black box access to the signing algorithm, from which it receives a string in $\{0, 1, x\}^{ln}$. The forger has the option of replacing arbitrary number of 'x' with binary digits or flipping some Hadamarded bits or changing some Hadamarded bits to 'x'. All these operations are random from the viewpoint of $\mathcal{A}$ and the forger has no way to determine (since $\mathcal{A}$ has

no access to the verification oracle since there is no public verification key) whether the modified string is a valid alternative signature without actually reconstructing f, which is turn requires decoding 'x' which contradicts the assumption that 'x' are not decoded.

The notion of more 'work' of $\mathcal{A}$ to convert to $\mathcal{B}$ is clear from the lemmas and claims above. $\mathcal{A}$ needs to recreate f from queries to the signing oracle, and from the discussion just concluded, it is clear that, for convenient parameter values, with probability close to 1, $\mathcal{A}$ can not distinguish the signing algorithm (using $\mathcal{U}$ or $\mathcal{F}$) used by the signing oracle. Even if $\mathcal{A}$ receives from $\mathcal{U}$ a signature belonging to $\mathcal{F}$, the probability of constructing the actual signature requires decoding 'x' present in the signature. In a walk of length ln, if the number of 'x' are $\approx \frac{ln}{c}$ for some constant $c > 0$, the probability of correctly decoding is $1/(2)^{ln/c}$. The constant c is determined by the algorithm [INSERT ALGORITHM HERE] that instantiates f.

Theorem 2 *$\mathcal{A}$ is strictly harder than $\mathcal{B}$.*

Proof: $\mathcal{A}$ is left to determine f from signer's responses to its queries. With a tight assumption that the signing algorithm returns a different response every time the same query is made, by standard probability definition and the result in lemma 5, we conclude by pigeonhole principle that $\mathcal{A}$ is forced to make the same query $\Omega(3^{ln} - 2^{ln} + 1) = \Omega(3^{\mathcal{O}(ln)})$ times before it can hope to receive at least one response that is sure to be in $\mathcal{F}$ (or concretely, $\mathcal{F}$'s response signature to the query made). From the $\Omega(3^{\mathcal{O}(ln)})$ responses received and using lemma 4, $\mathcal{A}$ fails to distinguish the exact response from $\mathcal{F}$.

Even if we further tighten the assumptions and say the forger is somehow able to deduce the signatures from $\mathcal{F}$ amongst $\mathcal{O}(3^{\Omega(ln)})$ signatures, $\mathcal{A}$ is faced with the task to decode $\approx \frac{ln}{c}$ number of 'x' in those signatures (claim 2). With probability $1/2^{ln/c}$, $\mathcal{A}$ is able to reconstruct a valid walk and thus it is able to reconstruct a part of f. This scenario holds for one such query (or one such message/starting state). Entire reconstruction of f involves repeating the entire procedure several times for distinct queries/messages/starting states. Given the randomness inherent in quantum computation, we may crudely say the assumptions taken here are unlikely to occur. $\mathcal{A}$ can thus be treated to be strictly harder than $\mathcal{B}$ because of the overhead involved to reconstruct f (and from f the weighted directed graph) before beginning to solve the k-path/k-walk problem on that graph.

IV. SAMPLE IMPLEMENTATION

A crude quantifier of security is the *bit security*, which states a scheme is λ-bit secure if an adversary has to perform $\Omega(2^{\lambda})$ operations to break it. From discussion in theorem 2 and the fact (refer discussion I A) that a forger has no access to the verification oracle since there is no public verification key, we concluded that in order to mount an attack, a forger shall have to make $\Omega(3^{\mathcal{O}(ln)})$ queries to the signer oracle to ensure at least one valid signature and will have to perform further $2^{ln/c}$ operations in order to decode 'x' in the received responses (decoding 'x' is the only way forward for a forger, see claim 2). Overall operations performed by the forger are crudely at least $(3.2^{1/c})^{ln}$.

For the sake of clarity, conciseness, and complexity of boolean reductions, we instantiate a toy example with $n = 4$ and $c = 2$. Consider the following mapping function f in table III.

TABLE III: Sample instantiation of the mapping function f for $n = 4$ and $c = 2$, as well as the corresponding dictionary.

A	B	C	D	P Q R S	d
0	0	0	0	1000 or 1001 or 1100 or 1101	[0, 2]
0	0	0	1	0000 or 0100 or 1000 or 1100	[2, 3]
0	0	1	0	1100 or 1101 or 1110 or 1111	[0, 1]
0	0	1	1	0000 or 0010 or 1000 or 1010	[1, 3]
0	1	0	0	0010 or 0110 or 1010 or 1110	[2, 3]
0	1	0	1	0110 or 0111 or 1110 or 1111	[1, 2]
0	1	1	0	0010 or 0011 or 0110 or 0111	[0, 2]
0	1	1	1	1010 or 1011 or 1110 or 1111	[0, 2]
1	0	0	0	1000 or 1010 or 1100 or 1110	[0, 3]
1	0	0	1	0100 or 0101 or 0110 or 0111	[0, 1]
1	0	1	0	0010 or 0011 or 1010 or 1011	[1, 2]
1	0	1	1	0001 or 0011 or 0101 or 0111	[0, 3]
1	1	0	0	0101 or 0111 or 1101 or 1111	[1, 3]
1	1	0	1	0001 or 0101 or 1001 or 1101	[2, 3]
1	1	1	0	0100 or 0110 or 1100 or 1110	[1, 3]
1	1	1	1	1000 or 1001 or 1010 or 1011	[0, 1]

In table IV, we instantiate the k-walk implementation for $n = 4$ and $c = 2$. The implementation is in Python and uses Qiskit. No parallelization or threading was used; parallelizing the burden of 100 runs will reduce the running time significantly. Signature lengths are represented as the symbolic representation lengths. Bit lengths of the signatures would be twice the length reported.

TABLE IV: Concrete values for different values of k ($n = 4$ and $c = 2$). **l**, **x**(number of 'x' in the signature for $c = 2$), and **Time** values are averaged over 100 iterations. *ops* is short for operations, i.e. the number of operations a potential forger has to perform as discussed before. Security(**Sec.**) is reported as $ln + \epsilon$ bits for arbitrary ϵ since $\mathcal{O}(3.\sqrt{2})^{ln} = \omega(2^2)^{ln} = \omega(2)^{ln}$. **Sig. len.** or the signature length is also averaged over 100 iterations.

k	l	x	$(3.\sqrt{2})^{ln}$ ops.	Sec./Sig.len.	Time
20	11.65	23.2	$1.325e+29$	$46.6+\epsilon$	4.48 s
30	38.24	76.48	$1.000e+96$	$152.96+\epsilon$	6.83 s
40	61.07	122.14	$2.08e+153$	$244.28+\epsilon$	9.03 s
50	89.32	178.64	$1.74e+224$	$357.28+\epsilon$	11.12 s
60	123.17	246.34	$(3.\sqrt{2})^{492.68}$	$492.68+\epsilon$	13.3 s
70	164.08	328.16	$(3.\sqrt{2})^{656.32}$	$656.32+\epsilon$	16.12 s
80	209.85	419.9	$(3.\sqrt{2})^{839.8}$	$839.8+\epsilon$	18.12 s
90	261.34	522.68	$(3.\sqrt{2})^{1045.36}$	$1045.36+\epsilon$	20.18 s
100	318.3	636.6	$(3.\sqrt{2})^{1273.2}$	$1273.2+\epsilon$	21.48 s

Instantiating a random z bit message, where $z \in \mathbb{Z}^+ : z \equiv 0 \, mod \, 4$ is next described. Groups of 4 bits are formed and each group is signed independently using table III. Recall from discussion in subsection II A that the signatures here are symbolic representations from {0, 1, x} where each symbol requires two bits for bit representation. We selected arbitrarily 00 for 0, 11 for 1, and 01 for x. Since the signing algorithm in algo. 2 runs as long as the cumulative hamming distance does not exceed k, the length of signatures from each run is not constant. We introduce a new symbol o depicted by the bits 10 that acts as a separator. Concretely, for a 100 bit message, 25 groups of four are formed. Let $l_i.n$ denote the length of the signature of the i-th group. The final communication is of the form: $...l_1.n....o...l_2.n....o...l_3.n....o.....o...l_{25}.n....o$. The recipient separates the signatures of separate groups using the separator and verifies normally. Since the bits {00, 01, 10, 11} is prefix code, there is no ambiguity in forming the symbolic representation of the signature from the bit representation: the verifier reads two bits at a time and converts to appropriate symbol. This procedure speeds up considerably depending on the value of n (larger n ensures smaller number of groups) and whether the implementation is parallelized.

V. CONCLUSION

In this paper, we described a digital signature scheme that works without the need to publish a verification key and thus stripping a potential forger of access to a verification oracle.

Future work on the same lines would include primarily investigating quantum attacks on such constructions. Crudely, a quantum adversary would need to process the multiple walks as in figure 1 in parallel to attain speedups. We remark such a processing is insufficient using n qubits because every level of figure 1 maps to the next level via Hadamards on certain positions. However, in a quantum setting, Hardamarding from level i to level $i+1$ works fine, but any further application to move from level $i+1$ to level $i+2$ through Hadamarding cancels the effect of Hadamards used in $i \to i+1$. Thus, continued superposition is not possible while exploring the different possible walks. A forger may try to associate new qubits for each level. This does away with the previous problem but introduces a new worry of managing such a large number of qubits and the decoherence they cause on each other. Moreover, the forger would need to repeat such a setup again and again because quantum measurement would output only a single walk which might or might not satisfy the forgery criteria the forger is working with. And finally, the forger would need to find a way to encode and decode 'x' as used in our construction. Overall, at this point, we conjecture the improbability of a quantum attack.

Future work would involve further investigations, primarily the complexity of the mapping function f that would reduce the running time of the signing procedure.

VI. ACKNOWLEDGEMENTS

N.M. acknowledges the hospitality of Indian Institute of Science Education and Research Kolkata during the project work. B.K.B. acknowledges the financial support of Institute Fellowship provided by IISER Kolkata. We thank Subhayan Roy Moullick for interesting discussions on possible black-boxed classical counterparts of the scheme proposed here. We also acknowledge the support of IBM Quantum Experience for using the quantum processors. The views expressed are those of the authors and do not reflect the official position of IBM or the IBM quantum experience team.

[1] S. Basagni, D. Bruschi, and F. Ravasio. On the difficulty of finding walks of length k. *Informatique théorique et applications*. p. 429-435, 1997.

[2] N. Alon, R. Yuster and U. Zwick, Color-coding, Journal of the ACM, 1995, 42, 4, pp. 844-856.

[3] Dekel Tsur, Faster deterministic parameterized algorithm for k-Path, Theoretical Computer Science, Volume 790, 2019, Pages 96-104.

[4] H. Nejatollahi, N. Dutt, S. Ray, F. Regazzoni, I. Banerjee, and R. Cammarota. 2019. Post-Quantum Lattice-Based Cryptography Implementations: A Survey. ACM Comput. Surv. 51, 6, Article 129 (February 2019), 41 pages. DOI:https://doi.org/10.1145/3292548

[5] Pirandola, S., Andersen, U. L., Banchi, L., Berta, M., Bunandar, D., Colbeck, R., Englund, D., Gehring, T., Lupo, C., Ottaviani, C., Pereira, J., Razavi, M., Shaari, J. S., Tomamichel, M., Usenko, V. C., Vallone, G., Villoresi, P., and Wallden, P. (2020). Advances in Quantum Cryptography. Manuscript submitted for publication.
[6] Goldwasser S. and Bellare M. Lecture notes on cryptography. Chapter 10. MIT course 6.87. 2008.
[7] Alkim E. et al. (2017) Revisiting TESLA in the Quantum Random Oracle Model. In: Lange T., Takagi T. (eds) Post-Quantum Cryptography. PQCrypto 2017. Lecture Notes in Computer Science, vol 10346. Springer, Cham
[8] N. Mishra, S. C. Rayala, B. K. Behera, and P. K. Panigrahi, Adding degrees of freedom to automated quantum Braitenberg vehicles, DOI: 10.13140/RG.2.2.34787.50723

VII. APPENDIX

A.

In this subsection, we provide an algorithm that can output a concrete instantiation of table I.

Algorithm 5: instantiate_mapping(state, d)

Result: Given the input state *state* and the dictionary *d*, the algorithm randomizes the mapping.

```
c = random(1, n)
hadamarded_positions = []
i = 1
while i < c do
choose a random position p from 1 to n
hadamarded_positions.append(p)
end
i = 1
fixed_positions = []
while i < n do
if(i ∉ hadamarded_positions)
choose value of position i arbitrarily from {0, 1}
fixed_positions.append(i)
end
d[state] = fixed_positions
```

At the end, the algorithm would return a structure like table III which could be further simplified by classical boolean reductions (involving Hadamard reductions) as depicted in [8] before being shared. Sharing happens by communicating strings of the form $P = AND(.) \,|\, AND(.) \,|\, H(.) \ldots$ and not the entire table structure as it is. Here, $AND(.)$ represents calls to the algorithm 8 while $H(.)$ represents calls to the algorithm 7. $|$ is simple bitwise-OR.

B. Implementing the main circuit

In this subsection, we provide a total of 3 algorithms from [8] that help in instantiating the circuit, concretely the mapping function f. For a detailed discussion, we refer the interested reader to [8].

Algorithm 6: twoQubitAND(qubit1, qubit2, qubit1NOT, qubit2NOT, cPos)

Result: Performs AND operation on qubits qubit1 and qubit2 and stores the measurement in position cPos in the classical register. Binary variables qubit1NOT and qubit2NOT denote whether qubit1 and qubit2 respectively must be flipped before the AND operation.

```
if qubit1NOT then
circuit.x(quantumRegister[qubit1])
end
if qubit2NOT then
circuit.x(quantumRegister[qubit2])
end
circuit.ccx(quantumRegister[qubit1],
quantumRegister[qubit2],
quantumRegister[auxiliaryQubit])
circuit.measure(quantumRegister[auxiliaryQubit],
classicalRegister[cPos])
if qubit1NOT then
circuit.x(quantumRegister[qubit1])
end
if qubit2NOT then
circuit.x(quantumRegister[qubit2])
end
circuit.reset(quantumRegister[auxiliaryQubit])
```

Algorithm 7: nQubitHadamard(qubitList, qubitNOTList)

Result: perform $H(qubitList)$ or Hadamard operation on the auxiliary qubit if $AND(qubitList)$ returns 1

```
output = nQubitAND(qubitList, qubitNOTList)
if output == 1 then
circuit.h(quantumRegister[auxiliaryQubit])
circuit.measure(quantumRegister[auxiliaryQubit],
classicalRegister[0])
circuit.reset(quantumRegister[auxiliaryQubit])
// execute the circuit
return classicalRegister[0]
end
return 0
```

The algorithms work by taking two input lists, one containing the qubit index to operate on and the other containing binary values whether to flip those qubits before the operation. In terms of qubit requirement, we need only one extra qubit (in addition to the mandatory n qubits to represent n bit inputs). All qubits are $|\,0\rangle$ or $|\,1\rangle$ since the AND operation is deterministic, and the auxiliary qubit is measured as soon as one AND operation is complete.

Reduction of the number of additional qubits required, deterministic qubit values, and short time interval between output and measurement means our scheme is not

Algorithm 8: nQubitAND(qubitList, qubitNOTList)

Result: Performs AND operation n qubits given by the list qubitList. qubitNOTList is a list of n binary values depicting whether each of the qubits in qubitList need to be flipped before *AND* is performed on them.

```
classicalRegisterIndex = 0
length = length(qubitList)
// handling odd number of qubits
if length % 2 != 0 then
if qubitNOTList[length - 1] == 1 then
circuit.x(quantumRegister[qubitList[length - 1]])
end
// cx operation on the last qubit as discussed before
circuit.cx(quantumRegister[qubitList[length - 1]],
quantumRegister[auxiliaryQubit])
circuit.measure(quantumRegister[auxiliaryQubit],
classicalRegister[classicalRegisterIndex])
circuit.reset(quantumRegister[auxiliaryQubit])
classicalRegisterIndex = classicalRegisterIndex + 1
if qubitNOTList[length - 1] == 1 then
circuit.x(quantumRegister[qubitList[length - 1]])
end
length = length - 1
end
i = length - 1
while i ≥ 0 do
twoQubitAND(qubitList[i], qubitList[i - 1],
qubitNOTList[i], qubitNOTList[i - 1],
classicalRegisterIndex)
classicalRegisterIndex = classicalRegisterIndex + 1
i = i - 2
end
```

much prone to decoherence and we may rely on the results produced.