Homomorphic Signature Schemes

Collin Berman, Dasith Gunawardhana, Sumit Narain

April 13, 2015

1 Introduction

The idea of a digital signature is a natural notion that closely parallels written signatures. In particular we wish to compute a signature of some data so that the signature can be publicly verified. In addition, we would like signatures to be unforgeable so that a signer cannot deny the validity of their signature [6].

For example, suppose someone wishes to send an electronic message and add their signature to show that the message is really from them. Then they can use their private key to generate a signature and send the signature along with their message. The recipient can then use the sender's public key to verify the authenticity of the signature.

1.1 Signatures from RSA

Recall the public key cryptosystem RSA [6]. Given a public key (e, n), we encrypt message M by

$$C \equiv \text{Enc}(M) \equiv M^e \pmod{n}$$

and decrypt using private key (d, n) by

$$\text{Dec}(C) \equiv C^d \pmod{n}$$

where $e \cdot d \equiv 1 \pmod{(p-1)\cdot(q-1)}$ for $n = p \cdot q$.

We can use this same cryptosystem to form a signature scheme. Correctness of RSA tells us that $\text{Dec}(\text{Enc}(M)) = M$ for a message M. However, we can also compute in the opposite order, where we still find that for a message M, $\text{Enc}(\text{Dec}(M)) = M$.

This gives an immediate way to compute a signature. A signer can use their private key on a message to compute a signature $\sigma = \text{Dec}(M)$. Then

anyone can publicly verify whether $\text{Enc}(\sigma) \stackrel{?}{=} M$. It is computationally infeasible for an attacker to come up with a signature for a given message, because deriving a signature is the same as decrypting a ciphertext, which is hard without the private key.

1.2 Homomorphic Signatures

With recent advances in cloud computing, it has become common to outsource computation to more powerful hosts. In this situation we can consider two different forms of security we may want. If the data we wish to compute over is confidential, we would want to provide encrypted data to the computing body. If the server is then able to compute over the encrypted data, we are using a *homomorphic* encryption scheme.

We may also consider the case where we do not care whether the server can read our data, but we wish to verify whether the computations the server has performed are correct. A natural way to do this is to extend signature schemes to also be homomorphic. Suppose we have produced signatures for some data $x_1, \ldots, x_n$, and we want the server to compute $y = g(x_1, \ldots, x_n)$ for some function g. Then we can verify the result of the computation if the server can also provide us with a signature $\sigma_{g,y}$ which certifies that y is indeed the result of computing $g(x_1, \ldots, x_n)$. If we are able to outsource the computation of *any* function g, we say the signature scheme is *fully* homomorphic. Finding an algorithm to compute such signatures is the subject of this paper.

Note that RSA is weakly homomorphic with respect to multiplication: let σ_1, σ_2 be signatures for x_1, x_2, that is,x'' $\sigma_i \equiv x_i^e \mod n$. Then $\sigma_1 \cdot \sigma_2 \equiv x_1^e \cdot x_2^e \equiv (x_1 \cdot x_2)^e$ is a valid signature for $x_1 \cdot x_2$.

1.3 Security of Signatures Schemes

As with written signatures, we want digital signatures to be unforgeable. We define an *existential forgery* to be the creation of a valid signature σ for a message M, where the legitimate signer has not produced such a signature. If no efficient adversary can produce such a signature, the signature scheme is *secure against existential forgery.*

This notion of forgery will obviously not work for homomorphic signature schemes; signatures need to be publicly computable for functions of signed data. Instead, we require that no efficient adversary can compute a valid signature for a message y that cannot be computed from any x that has been signed. That is, if $x_1, \ldots, x_n$ have been legitimately signed under a

scheme homomorphic with respect to a binary operation $\odot$, an adversary should not be able to compute a signature for some $y \notin \text{span}_{\odot}(x_1, \ldots, x_n)$.

Even this notion of security will not work for fully homomorphic signatures, as any y can be computed from inputs $x_1, \ldots, x_n$. Instead we say the scheme is secure if no efficient adversary can compute a valid signature $\sigma_{g,y}$ where $y \neq g(x_1, \ldots, x_n)$. This is the essential property of security we need for verifying the result of outsourced computation.

1.4 Paper Overview

In the rest of this paper we present three homomorphic signature schemes. In Section 2, we give a construction of a weakly homomorphic scheme over a redaction operation. Given a signature of some message, signature holders will be able to compute a valid signature of a redaction of the message. In Sections 3 and 4, we give two constructions of fully homomorphic signature schemes. These constructions are highly similar, but are extended in different directions. The construction in Section 3 gives a second construction using the random oracle model resulting in smaller public parameters, while the construction in Section 4 allows separation of data into multiple data sets by tagging.

2 Homomorphic Signature Scheme

The paper of Homomorphic Signature Schemes by Robert Johnson, David Molnar, Dawn Song, and David Wagner [5] focuses on the idea that signature holders are used to create a signature on a sub-message of the original message using unions and subsets. The author goes into the idea of redactable signatures and possible solutions to this problem, and uses the hash and sign paradigm to verify the legitimacy of the redaction.

In essence, we are interested in the signature scheme for a message that is authenticated. Then we also want to see if a subset or redaction of that message can also be authenticated. For example, let's assume there is a message M with several strings $x = x_1x_2...x_n$, where each x_i represents a letter or word in the message. This string x was authenticated or signed by a person, say Alice. Alice sends the string and it's signature to another party. Alice wants to show the message to another party, say Bob, but Bob is not authorized to view certain parts of the string, say x_1x_2. Then Alice shows Bob the updated string, $x_{updated} = **x_3x_4...x_n$. The stars represent the location of the removed word or letter from the message. What if Bob wants to verify the authenticity of the message $x_{updated}$? Then Alice can

produce a valid signature for the $x_{updated}$ and pass that to Bob. From the new signature from $x_{updated}$, Bob should be able to verify that the message is a subset of the string signed by the authority who is Alice.

2.1 Redactable Signatures Basics and Obvious Scheme

The issue that redactable signatures tries to solve is that a censor can delete certain substrings of a signed document without destroying the ability of the recipient to examine the redacted (the text with censors on the parts that are removed) document. The censors tend to be a symbol which represents the where the hidden text is placed. In this article, the author used the pound symbol to signify that characters have been hidden. The author mentioned several applications such as proxy cryptography or incremental cryptography.

The importance of showing the positions where the element has been deleted is that an authenticated message could have its meaning changed when some elements are deleted. Imagine a message Alice sends to Bob: Our co-worker, Alex, spends 60 hours a week trying to find ways to add value to our bottom line, and never have I known him to shirk his duties. Alex is a true asset to our company, and I cannot think of one person better suited to your requirements [5]. If Bob got that message and removed some of the wording without the removed words leaving any traces, the updated message could become: Alex spends 60 hours a week trying to find ways to shirk his duties, and I can think of one person better suited to your requirements. This message would also be authenticated, but the entire meaning of the message has been changed. The way to prevent this would be to put censors where the deleted portions have been: Alex spends 60 hours a week trying to find ways to ********** shirk his duties *******, and I can*** think of one person better suited to your requirements (the * are meant to block out each letter in the original message sent by Alice).

The simplified or obvious way to solve this problem is to have a fixed signature scheme Sig_0. There is no special homomorphic properties from Sig_0. We would sign a character of message x at index i and generate a key value pair (s, v) for the index and message. The structure for this scheme would be

$$\text{Sig}(x) = \langle \text{Sig}_0(n, v), s(1, x_1), ..., s(n, x_n) \rangle$$

How this works is that every time that a person wants to censor an element. The index of the character is chosen and then the whole key-value pair is removed from the set. This is easily verified since one can check all the indexes which represents the values in the message. By removing the key

value pair, the contents of that index cannot be revealed but it is easy to tell which portions have been redacted.

Although this scheme is simple and quite easy to understand, the issue is that the Sig(x) is very long. If each s in the Sig(x) will produce m-bit signatures, this would result in a construction of signatures which would be $m * n + O(1)$ for the length. This would make it difficult to compare to the length of the message which would be n; so the next step is to find a scheme which has much shorter signatures.

2.2 Author's Scheme Construction

The author creates a tree structure for the signature scheme. In this portion, the author uses the hash and sign paradigm along the tree to allow the person to correctly verify the Sig(x). A k value will be generated at random and then create a tree structure which is used to associate each k value as a key to each node. Each node is defined as the recursive relation which is completed when the all the values in the message are associated with a key value. Once all the words or characters in the tree are corresponding to k values, then a hash value for each leaf will be calculated by $v = H(0, k_l, x_l)$. This will be the individual hash value for each leaf. Then to calculate the total hash value, the graph will begin at the leaves and recurse to the node by a summation hash function $v_{node} = H(1, v_{nodeLeft}, v_{nodeRight})$. If there is a missing child then we just remove that portion from the summation hash function: $v_{node} = H(1, v_{nodeRight})$ or $v_{node} = H(1, v_{left})$. Finally after the summation hash function is complete, then one would calculate the Sig(x) since we are given the k value as well as the summation hash value: $\text{Sig}(x) = \langle k, \text{Sig}_0(v_{node})\rangle$. The verification for the Sig(x) is easy since we just compare the signature of the original message with the $Sig_0(v_{node})$. This portion did not involve redaction, but the redaction portion is not difficult to integrate in this tree structure.

The next part would describe how to remove a symbol and still maintain secrecy of the message. If we keep the same tree structure and simply remove the character or word, that would make revealing $v_l = H(0, k_l, x_l)$ a security risk since we have given k_{root} which can determine all the values of k, and then the person can iterate through all the possible values of x_l and learn the value of the erased symbol. To keep the data hidden, when we redact a character or word, we also redact the k_{root} as well as the ancestor's keys for the specific word or character. To access the sibling or the removed parent's child, we will reveal the sibling. This information will be enough to calculate the signature as well as maintaining secrecy. The algorithm of

removing a leaf includes removing the root node key, the the ancestors' keys, and revealing the sibling. If we remove the sibling, then we could replace the $v_{sibling}$ with the v_{parent}. This would give the same result since the v_{parent} will no longer have any children and it would make the compact the signature.

2.3 Conclusion and Security

The security measures that this scheme has is secure against random forgeries and existential forgeries. By random forgeries, it means that the adversary cannot create valid signatures for random messages. The existential forgery is that the adversary cannot form a signature for an unredacted message if all they have are signatures for redactions of the message.

Homomorphic signatures present interesting capabilities for hiding information from people. The authors have presented a novel way to create a signature on redacted sub-message of the original message.

3 Leveled Fully Homomorphic Signatures

We now present a fully homomorphic signature scheme due to Daniel Wichs [7]. This scheme is achieved using a construction called *homomorphic trapdoor functions* as a building block. The security of these functions and the signature scheme as a whole follows from the hardness of standard lattice problems.

Just as evaluating functions over encrypted data increases noise values in fully homomorphic encryption, so does evaluating circuits over signatures. Because however the signature scheme presented here does not have the bootstrapping renormalization functionality required to reduce noise, we cannot perform unbounded computation over signatures. Instead the maximum depth of a circuit is bounded, resulting in a *leveled* fully homomorphic signature scheme.

After constructing homomorphic trapdoor functions, the resulting signature scheme is shown to have different properties depending on the model of computation. In the standard model, the scheme requires public parameters larger than the total size of all signed data in order to be secure. Although this would make the scheme infeasible to use in practice, the scheme achieved in the random oracle model has short public parameters.

3.1 Background on Lattice Problems

The *small integer solution* (SIS) problem asks for a vector orthogonal (mod some integer) to a randomly chosen matrix. Formally, given integer parameters n, m, q, β, we look for a vector $\mathbf{u} \in \mathbb{Z}_q^m$ such that $\mathbf{A} \cdot \mathbf{u} = 0$ given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$. We further require $\mathbf{u}$ to be a nontrivial solution, that is, $\mathbf{u} \neq 0$, and to be small, that is, $||\mathbf{u}|| \leq \beta$. The norm used in this paper is

$$||\mathbf{u}||_\infty = \max_{1 \leq i \leq m} u_i$$

but any norm $|| \cdot ||$ will work.

Note that the matrix $\mathbf{A}$ is chosen uniformly randomly. While some lattice problems are hard in the worst case, SIS is hard in the average case [1]. The hardness of SIS is also implied by the hardness of Learning with Errors (LWE) in lattices.

A property of SIS that makes it attractive for trapdoor functions is that although solving SIS for a random matrix is hard, there is an algorithm to sample a random matrix $\mathbf{A}$ along with a trapdoor for $\mathbf{A}$ that makes SIS easy. Additionally, the trapdoor functions constructed here rely on the fact that there are also some matrices $\mathbf{G}$ for which SIS is easy to solve, even without a trapdoor.

Finally we remark that solving SIS for matrix $\mathbf{A}$ is equivalent to finding a short matrix $\mathbf{U}$ such that $\mathbf{A} \cdot \mathbf{U} = \mathbf{V}$ for a given matrix $\mathbf{V}$ [3]. This problem of "inverting" $\mathbf{A}$ is the problem that will actually be used in constructing the signature scheme.

3.2 Homomorphic Trapdoor Functions (HTDF)

To construct the homomorphic signature scheme, HTDFs are introduced as a building block. An HTDF is a function $f_{pk,x}(\cdot)$ parameterized by a public key pk and a data input x. The function can be inverted using a trapdoor secret key sk (but not without the trapdoor) to find some input u such that $f_{pk,x}(u) = v$ for some given v (note that the function need not be one-to-one, so there may be multiple such inputs u that work). The function is also homomorphic in that given values

$$u_i, x_i, v_i = f_{pk,x_i}(u_i), 1 \leq i \leq N$$

and a function $g : \{0,1\}^N \to \{0,1\}$, an input u^* and output v^* can be publicly computed so that

$$f_{pk,g(x_1,\ldots,x_N)}(u^*) = v^*.$$

To construct an HTDF, we first select a public/private key pair by sampling a random matrix $\mathbf{A}$ along with its SIS trapdoor. $\mathbf{A}$ is then used to compute the HTDF over an input matrix $\mathbf{U}$ by

$$f_{\mathbf{A},x}(\mathbf{U}) = \mathbf{A} \cdot \mathbf{U} - x \cdot \mathbf{G} = \mathbf{V}$$

where $\mathbf{G}$ is a matrix for which SIS is easy without a trapdoor. Then this function can be inverted by using the trapdoor to find a short matrix $\mathbf{U}$ such that $\mathbf{A} \cdot \mathbf{U} = \mathbf{V} + x \cdot \mathbf{G}$.

Homomorphic addition is easy to perform with $\mathbf{U}^* = \mathbf{U}_1 + \mathbf{U}_2$ and $\mathbf{V}^* = \mathbf{V}_1 + \mathbf{V}_2$. Then

$$\begin{aligned} f_{\mathbf{A},x_1+x_2}(\mathbf{U}_1 + \mathbf{U}_2) &= \mathbf{A} \cdot (\mathbf{U}_1 + \mathbf{U}_2) - (x_1 + x_2) \cdot \mathbf{G} \\ &= (\mathbf{A} \cdot \mathbf{U}_1 - x_1 \cdot \mathbf{G}) + (\mathbf{A} \cdot \mathbf{U}_2 - x_2 \cdot \mathbf{G}) \\ &= f_{\mathbf{A},x_1}(\mathbf{U}_1) + f_{\mathbf{A},x_2}(\mathbf{U}_2) = \mathbf{V}_1 + \mathbf{V}_2. \end{aligned}$$

However, multiplication is harder. We begin by finding a short matrix $\mathbf{R}$ such that $\mathbf{G} \cdot \mathbf{R} = -\mathbf{V}_1$ by exploiting the special structure of $\mathbf{G}$. Then set $\mathbf{V}^* = \mathbf{V}_1 + \mathbf{V}_2$ and $\mathbf{U}^* = x_2 \cdot \mathbf{U}_1 + \mathbf{U_2} \cdot \mathbf{R}$. It can be verified that

$$f_{\mathbf{A},x_1 \cdot x_2}(\mathbf{U}^*) = \mathbf{V}^*$$

so the HTDF is fully homomorphic.

3.3 Signatures from HTDFs

A homomorphic signature scheme requires public parameters, a way to sign data and verify signatures, and a way to compute a circuit over a data/signature pair. These components are built out of an HTDF with public/secret key pair (pk, sk). The public parameters used in the signature scheme consist of N random outputs of the HTDF, $(v_1, \ldots, v_N)$ where N is a bound on the number of data points that has been set beforehand.

To sign data $(x_1, \ldots, x_N)$ the secret key is used to invert the HTDF to find inputs $(u_1, \ldots, u_N)$ such that $f_{pk,x_i}(u_i) = v_i$ for each $1 \leq i \leq N$. Then anyone can verify the signature by checking each $f_{pk,x_i}(u_i) \stackrel{?}{=} v_i$ with the public parameters.

Furthermore anyone can publicly homomorphically compute a signature $u^*_{g,y}$ for $y = g(x_1, \ldots, x_N)$ if they have signed data $(x_1, \ldots, x_N)$. This signature will satisfy $f_{pk,y}(u^*_{g,y}) = v^*$ where v^* can also be homomorphically computed by a verifier from g and the public parameters.

3.4 Signatures in the Random Oracle Model

A problem with the above construction is that we need public parameters as long as the message being signed. One way to eliminate this requirement is through the random oracle model. In the random oracle model, hash functions behave like completely random functions. This is what hash functions are designed to do, but a real hash function cannot be truly random. However, assuming hash functions do behave randomly often leads to simpler proofs or constructions.

In the scheme presented here, we no longer need N outputs of the HTDF to serve as public parameters. Instead, generate $(v_1, \ldots, v_N)$ when signing by using a random hash function H to set $v_i = H(r, i)$ where r is randomness generated by the signing procedure (and must be included as a part of the signature). Whereas in the standard model we had a public key of $(pk, v_1, \ldots, v_n)$ where pk is the public key for the HTDF, our public key is now just pk.

Verifying a signature is essentially the same as before, but now the verifier computes $(v_1, \ldots, v_N)$ themselves using H and r. Homomorphic computation over signatures is fundamentally the same. Note that in both models of computation, a verifier does not need the original data $(x_1, \ldots, x_N)$ to verify a signature $u^*_{g,y}$ for $y = g(x_1, \ldots, x_N)$.

4 Another Leveled Fully Homomorphic Scheme

4.1 Introduction

The motivation for fully homomorphic signature schemes arises from the shift from local computing to remote storage and computation. Homomorphic encryption schemes already allow for computation on encrypted data. For outsourced data it is desirable to have a verification method to ensure that the computations performed were correct and by the desired party. Homomorphic signature schemes allow for a way to verify remote computation. The idea is that the computing party will provide a small signature σ. The signature must be easily verifiable, but difficult for anyone other than the computing party to forge.

We present a fully homomorphic signature scheme due to Gorbunov and Vaikuntanathan [4]. The construction is based on the signature scheme from trapdoor permutations by Bellare and Rogaway [2]. The security of the scheme is based on the assumed hardness of finding short integer solutions in hard lattices. The novel construction is based on homomorphic hash

functions and fully key-homomorphic trapdoor functions.

4.2 Key-Homomorphic Trapdoor Functions

The base signature scheme is based on a trapdoor permutation. The signature algorithm generates a trapdoor permutation pair of functions f_{pk} and f_{pk}^{-1} and a hash function $H(\cdot)$. The signature is obtained by hashing the message m, and then using the inverse function on the result: $\sigma = f_{pk}^{-1}(H(m))$. To verify check that $f_{pk}(\sigma) = H(m)$. The trapdoor function used here does not have to be a trapdoor permutation; it is sufficient to have a pre-image sampleable family of surjective functions.

In order to make the signature scheme homomorphic, the hash function and trapdoor function must also be fully homomorphic. Given two messages m_1, m_2 signed by functions $f_{pk_1}^{-1}, f_{pk_2}^{-1}$ respectively, and hash function $H(\cdot)$, the signature can be obtained as follows:

$$\begin{aligned}\sigma' &= f_{pk_1}^{-1}(H(m_1)) + f_{pk_2}^{-1}(H(m_2)) \\ &= f_{pk_1+pk_2}^{-1}(H(m_1) + H(m_2)) = f_{pk_1+pk_2}^{-1}(H(m_1 + m_2))\end{aligned}$$

The verification algorithm will check that $f_{pk_1+pk_2}(H(m_1 + m_2)) = \sigma'$.

4.3 Construction of the Trapdoor Function

The difficulty in breaking this signature scheme is based on the hardness short integer solutions (SIS) problem, which requires a short solution to $\mathbf{AR} = 0$, given a uniformly random $\mathbf{A}$. Although there might exist many solutions, they are difficult to find which follows from the difficulty of finding solutions to standard arbitrary lattice problems. However, given a trapdoor for $\mathbf{A}$, finding the solution to SIS becomes easy.

The hash function H takes l-bit input $\vec{\mu} = (\mu_1, \ldots, \mu_l)$.

$$H(\vec{\mu}) = (\mathbf{D}_1 + \mu_1\mathbf{B}, \mathbf{D}_2 + \mu_2\mathbf{B}, \ldots, \mathbf{D}_l + \mu_l\mathbf{B})$$

$\mathbf{B}, \mathbf{D}_1, \ldots, \mathbf{D}_l$ are uniformly random matrices in $\mathbb{Z}_q^{m\times n}$. n is the lattice dimension and $m = \Omega(n \log q)$. The hash function contains a trapdoor for the matrix $\mathbf{B}$.

The trapdoor function for signing takes the form

$$f_{\mathbf{A}}(\mathbf{R}_1, \ldots, \mathbf{R}_l) = (\mathbf{A}_1\mathbf{R}_1, \ldots, \mathbf{A}_l\mathbf{R}_l)$$

A is a uniformly random matrix in $\mathbb{Z}_q^{m\times n}$. $\mathbf{R}_i$ are $m \times m$ matrices with "small" entries. This function is pre-image sampleable, as shown by Gentry,

Peikert, and Vaikuntanathan [3]. Using a short basis of the lattice it is possible to sample a random inverse of any element in the range of the function. This is a necessary component of the base signature scheme.

4.4 Homomorphic Property for Circuits

Given a circuit C (reduced to NAND gates), one should be able to compute $C(H(\vec{\mu}))$ and get the encrypted output $H(C(\vec{\mu}))$. In addition, given $f_{pk}^{-1}(H(\vec{\mu}))$ one should be able to compute $f_{C(pk)}^{-1}(H(C(\vec{\mu})))$. For each NAND gate, the signature can be computed publicly by anyone that knows the signature for the two input bits. For bits μ_1, μ_2 let $\mathbf{D}_1 + \mu_1\mathbf{B}, \mathbf{D}_1 + \mu_1\mathbf{B}$ be the hash values, respectively. Compute a matrix $\tilde{\mathbf{D}_\mathbf{2}}$ such that $\mathbf{B}\tilde{\mathbf{D}}_2 = \mathbf{D}_2$. The hash for the the NAND of two bits is as follows:

$$(\mathbf{D}_1 + \mu_1\mathbf{B}) \cdot \tilde{\mathbf{D}}_2 - \mu_1 \cdot (\mathbf{D_2} + \mu_2\mathbf{B}) = \mathbf{D}_1\tilde{\mathbf{D}}_2 - \mu_1\mu_2 \cdot \mathbf{B} =$$
$$(\mathbf{D}_1\mathbf{D}_2 - \mathbf{B}) + (\mu_1 \text{NAND} \mu_2) \cdot \mathbf{B} = \mathbf{D}_{\text{out}} + (\mu_1 \text{NAND} \mu_2) \cdot \mathbf{B}$$

The signature for the entire circuit is computable gate by gate. All that remains is to compute the signature for the NAND gate. For the two input bits μ_1 and μ_2, suppose the signatures are "short" matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ such that:

$$\mathbf{A}\mathbf{R}_1 = \mathbf{D}_1 + \mu_1\mathbf{B} \text{ and } \mathbf{A}\mathbf{R}_2 = \mathbf{D}_2 + \mu_2\mathbf{B}$$

Without a trapdoor for the signing function, these "short" matrices are hard to find. The homomorphic NAND of the signature of the two bits is shown here:

$$\mathbf{A}(\mathbf{R}_1\tilde{\mathbf{D}}_2 - \mu_1\mathbf{R}_2) = (\mathbf{D}_1 + \mu_1\mathbf{B}) \cdot \tilde{\mathbf{D}}_2 - \mu_1 \cdot (\mathbf{D_2} + \mu_2\mathbf{B}) =$$
$$\mathbf{D}_1\tilde{\mathbf{D}}_2 - \mu_1\mu_2 \cdot \mathbf{B} = \mathbf{D}_{\text{out}} + (\mu_1 \text{NAND} \mu_2) \cdot \mathbf{B}$$

The signature for the NAND is $\mathbf{R}_1\tilde{\mathbf{D}}_2 - \mu_1\mathbf{R}_2$. In the same manner as computing the hash for the entire circuit, the signature for the circuit is computed gate by gate.

4.5 Multiple Datasets

For multiple datasets $\{\vec{\mu_j}\}$, in order to ensure that an adversary cannot reuse signatures, each dataset must be tagged with a tag t that is unique to the dataset, i.e the tag of the computed circuit output μ' must match the tag of the input dataset $\vec{\mu}$. The security is defined as follows: A PPT adversary should not be able to forge a signature for a previously unseen tag, nor should the adversary be able to forge a signature for a previously seen tag but with incorrect computation.

4.6 Limitations

The size of the signature will be $m^{O(d)}$, which grows by a polynomial factor m based on the depth of the circuit, d. For correctness and security q must be larger than $m^{O(d)}$. This implies that the for a fixed q the depth of the circuits acceptable by this signature scheme are capped. Hence the signature scheme is "leveled."

References

[1] M. Ajtai. Generating hard instances of lattice problems (extended abstract). STOC '96, pages 99–108, New York, NY, USA, 1996. ACM.

[2] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[3] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. *STOC*, pages 197–206, 2008.

[4] S. Gorbunov and V. Vaikuntanathan. (leveled) fully homomorphic signatures from lattices. Cryptology ePrint Archive, Report 2014/463, 2014.

[5] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In B. Preneel, editor, *Topics in Cryptology CT-RSA 2002*, volume 2271 of *Lecture Notes in Computer Science*, pages 244–262. Springer Berlin Heidelberg, 2002.

[6] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.

[7] D. Wichs. Leveled fully homomorphic signatures from standard lattices. Cryptology ePrint Archive, Report 2014/451, 2014.