SANDIA REPORT
SAND2019-0111
Printed November 2018

Sandia National Laboratories

Post-Quantum Cryptographic System Assessment

Cordaro, Jennifer
Ehn, Carollan
Marshall, Nathan
Torgerson, Mark

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

Recent research and development in exploiting quantum phenomenon have solidified the creation of large-scale quantum computers as a reality. These machines will have the ability to solve intractable problems defined on conventional computers. This has a significant impact on current cryptographic systems. A viable quantum computer will require an increase in symmetric key sizes and the replacement of asymmetric cryptographic schemes. Specifically, new constructs for public key cryptosystems must be established in order to continue to ensure the security that digital signatures and key exchange protocols provide. Understanding the post-quantum landscape is critical to applying Sandia-developed capabilities to post-quantum cyber areas. Developing a cohort of experts in this challenging and volatile space has enabled our ability to adapt to the new challenges in various customer mission areas.

CONTENTS

1. Hash-based Cryptography 8
   - 1.1. Preliminaries and Notation 8
   - 1.2. Lamport 8
   - 1.3. Winternitz 10
   - 1.4. Merkle Signature Scheme 11
   - 1.5. MSS key Generation via PRNG and Tree Chaining 12
   - 1.6. BiBa 15
   - 1.7. HORS 15
   - 1.8. HORS++ 16
   - 1.9. HORSE 16
   - 1.10. HORST 17
   - 1.11. Modern Methods up for Standardization 18
2. Lattice Cryptography 21
   - 2.1. The Cryptograhic Family 22
   - 2.2. Common and Promising Algorithms 23

This page left blank

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
PQC	Post-Quantum Cryptography

1. HASH-BASED SIGNATURE ALGORITHMS

This note outlines several hash-based public key signature algorithms. The intent here is to provide the simplest explanations possible. To facilitate that goal, we sluff off various size considerations to simplify all notation. Only in cases where it might facilitate understanding do we add that level of detail.

In the literature, pains are made to distinguish between one-way functions and cryptographic hash functions. When trying to judge and compare efficiency and security this distinction might be important. For most of this discussion, the distinction is immaterial. Here we assume we have a cryptographic hash function H. The hash function has input s and output $p = H(s)$, called the preimage and image respectively. Generally, s would be a secret/private key and p would be a public key.

The security of the schemes described here relies on the fact that determining the preimage from a given image is difficult and cannot be reasonably be done unless one starts with the preimage and computes the image. Further, if the image is published and "committed" to, then, at some future time, revelation of the preimage says something about the foreknowledge of the commitment. All presented schemes are an attempt to turn the commitment and revelation of the preimage into an efficient public key signature scheme.

The chronologically first schemes are simple and straightforward. In the intervening years, complexity has been added to these simple schemes to address one perceived problem or other. For instance, the first schemes were one-time-signature (OTS) schemes. Allowing one use of the signature. Later, few-time-signatures (FTS) were introduced to allow the keys to be used more than once. Sometimes, the transition from OTS to FTS happens in the same paper. Almost always, the security of FTS methods is a decreasing function of the number of times the same keys are used. The literature goes to great lengths to quantify how the security degrades with use. We omit these concerns to focus on the methods.

1.1. Preliminaries and Notation

We will attempt to be consistent in the notation throughout this note. Let H be a cryptographic hash function that takes an arbitrary input and outputs an n-bit digest. The signature schemes will sign a message M. To sign the message, the hash of M will be computed. Let $H(M) = h = (h_{n-1} \dots h_1 h_0)$, where h_i is the i-th bit in the binary representation of h. For a few of the methods, the hash output h will be broken into equal sized pieces larger than a single bit in length. If $n = a * b$ then h will be written as $h = h_{a-1}|| \dots ||h_1||h_0$, where each of these h_i are b bits in length.

For each scheme, there is a difference in the exact nature of the public and private/secret keys, but when possible p and s will be used for public and secret/private values, respectively.

Throughout, $\lg(\cdot)$ is the base two logarithm. As mentioned in the introduction, our intent is to get to the heart of the method. Often, we choose parameters to simplify the discussion rather than keep more general possibilities. For instance, in the literature, a method may say, "Let $v = \lfloor \lg(x) \rfloor + 1$". We assume for simplicity that $x = 2^y$ for some integer y and set $v = y + 1$. Often the most general parameters do not fit together, and padding is required. We choose the parameters so that everything fits without the need to discuss padding. Even with this level of simplification, the

parameters may not match up exactly, in that case the reader is encouraged to go to the literature and get the exact details.

1.2. Lamport [LAM 79]

The first hash-based public key signature method in the literature is attributed to Lamport [Lam 79]. The signer has a list of pairs of secret numbers $SK = \{(s_{n-1,0}, s_{n-1,1}), \dots, (s_{1,0}, s_{1,1}), (s_{0,0}, s_{0,1})\}$ that are the secret key. For each i and j set $p_{i,j} = H(s_{i,j})$. $PK = \{(p_{n-1,0}, p_{n-1,1}), \dots, (p_{1,0}, p_{1,1}), (p_{0,0}, p_{0,1})\}$ is the public key.

1.2.1. Sign

To sign a message M, the signer first computes $h = H(M) = (h_{n-1} \dots h_1 h_0)$. The signature of M is $Sig(M) = (s_{n-1,h_{n-1}}, \dots, s_{1,h_1}, s_{0,h_0})$.

1.2.2. Verify

To verify the received string of values $(a_{n-1}, \dots, a_1, a_0)$ is the signature of M, the verifier computes $H(M) = (h_{n-1} \dots h_1 h_0)$, then for each i checks to see if $H(a_i) = p_{i,h_i}$

1.2.3. Notes

This is the first of the signatures of this type. It relies heavily on the fact that knowledge preimage/private values of the image/public key values accounts for something.

Once one message has been signed, half of the private values have been revealed. One may sign another message, but already revealed preimages might be reused. Anytime a preimage is reused, that bit in the hash is not actually verified. After a couple of signatures, most of the private values have been revealed and no security is possible.

The up-front work to create the public key dominates the work for the signer. When it is time to sign a message, a single hash is computed, and the stored private values are retrieved and published.

The verifier must perform $n + 1$ hashes to verify the signature.

This scheme is an OTS method. The signature strength decays rapidly with multiple signatures on random messages. However, there are chosen message attacks that allow forgeries after a couple uses with the same key [Ber 09].

The OTS nature of the signature plus the up-front work combined with the amount of work to verify has spawned a plethora of methods attempting to improve some part of Lamport's scheme.

1.3. Winternitz [Mer 79, 89]

Let k, t, u, v and w be integers such that $u = 2^{w*k}$, $n = w * u$, $v = 1 + k$ and $t = u + v$.

Let $SK = \{s_{t-1}, \dots, s_1, s_0\}$. Instead of letting the public key be the single hash of the private key, for each i set $p_i = H^{2^w-1}(s_i)$. The public key $PK = \{p_{t-1}, \dots, p_1, p_0\}$ is published.

Including s_i, there is a 2^w long chain of successively hashed values reaching from s_i to p_i. The signature involves the computation of a series of hash chains determined by the hash of the message and a checksum.

1.3.1. Sign

Since $n = w * u$, split $h = H(M)$ into u equal sized pieces of width w, giving $h = b_{u-1}|| \dots ||b_1||b_0$.

Set $c = \sum_{i=0}^{u-1}(2^w - b_i)$. Since $c \leq u * 2^w$, the checksum, c, can be represented with $w + \lg(u)$ bits. Note that $w + \lg(u) = w + w * k = w * (k + 1) = w * v$. Thus c may be split into v equal sized pieces of width w. Recall $t = v + u$, so set the checksum to $c = b_{t-1}|| \dots ||b_{u+1}||b_u$.

In all cases, b_i is w bits in length and viewed as an integer $0 \leq b_i < 2^w$.

The signature of M is $Sig(M) = (H^{b_{t-1}}(s_{t-1}), \dots, H^{b_1}(s_1), H^{b_0}(s_0))$

1.3.2. Verify

To verify that $(a_{t-1}, \dots, a_1, a_0)$ is the signature of M, the verifier computes $c = b_{t-1}|| \dots ||b_{u+1}||b_u$ and $h = b_{u-1}|| \dots ||b_1||b_0$ as described and then checks for each i that $H^{2^w-1-b_i}(a_i) = p_i$.

1.3.3. Notes

A chain of hashed values $(s_i = H^0(s_i), H^1(s_i) = H(H^0(s_i), \dots, p_i = H^{2^w-1}(s_i) = H(H^{2^w-2}(s_i)))$ is computed. The b_is tell the signer how far down in the hash chain to reveal. The verifier computes the chain the rest of the way for verification.

This scheme is an OTS method. The signature strength decays rapidly after multiple signatures. There are chosen message attacks that allow forgeries after a couple uses with the same key [Ber 09].

1.4. Merkle Signature Scheme (MSS), [MER 89]

The MSS has two parts:

- A hash tree
- A one-time signature scheme

The OTS mentioned here is not actually specified as part of the MSS. Nearly any OTS may be used. The primary contribution of the MSS is the notion of the Merkle's hash tree. The majority of this section will cover that concept, which is at the heart of most modern hash-based signatures.

The hash tree may be thought of as a certificate for the public keys used in the OTS, except that it is not created or managed by a third party. (Maybe it could be in some future time, but it is not part of any current notion.) Any signature verification must verify the signature of the given message using the given public verification keys and then use the hash tree to verify the public keys used.

The hash tree is a complete binary tree of height $J \geq 2$. The root node of the tree is the public verification value for the tree or the public key of the tree. The 2^J leaves of the tree correspond to separate instances of the OTS, each of which allow a separate message to be signed.

1.4.1. Key Set Up

For each $0 \leq i < 2^J$, let (PK_i, SK_i) be a separate pair of public and private keys of a OTS scheme and let $N_{0,i} = H(PK_i)$. In this scheme, the i-th public verification key is not revealed until the i-th message is signed. One OTS key pair is associated with each leaf of the tree.

The 2^J hashed public keys, $N_{0,i}$, are the leaf nodes of the Merkle hash tree. To construct the rest of the tree, each node in one layer is the hash of two nodes from a lower layer. For $1 \leq j \leq J$ and $0 \leq i < 2^{J-j}$ set $N_{j,i} = H(N_{j+1,2i}, N_{j+1,2i+1})$. Finally, $N_{J,0}$ is the root of the tree and is published as the public key for the tree. All other nodes of the tree must be kept secret until they are revealed as part of the signature.

1.4.2. Sign

To sign the k-th message, M_k, the public and private key pairs, (PK_k, SK_k), are used as usual to create the OTS signature, $Sig(M_k)$. In addition, a portion of the Merkle tree must be revealed. PK_k is revealed to verify the signature and to compute the leaf node, $N_{0,k}$. The J nodes closet to the path from $N_{0,k}$ to $N_{J,0}$ must also be revealed so that each successive node in the tree may be computed and finally verified.

Write $k = k_{J-1} \dots k_1 k_0$ as the binary representation of k. Then for each $0 \leq j \leq J-1$ set $a_j = k_{J-1} \dots k_{j+1} k_j$, which are the $J-j$ most significant bits of k. For the same range, set $b_j = a_j \; xor \; 1$.

The full MSS signature of M_k is the long list $(k, Sig(k, M_k), PK_k, N_{J-1,b_{J-1}}, \dots, N_{1,b_1}, N_{0,b_0})$.

1.4.3. Verify

To verify the signature, the verifier must verify $Sig(k, M_k)$ according to the OTS scheme. The verifier must compute $H(PK_k) = N_{0,k} = N_{0,a_0}$. With N_{0,a_0} and N_{0,b_0} the verifier can compute N_{1,a_1}. If computed in order from $1 \leq j \leq J$ the verifier sets $N_{j,a_j} = H(N_{j-1,a_{j-1}}, N_{j-1,b_{j-1}})$ if a_{j-1} is even and b_{j-1} is odd or else the verifier sets $N_{j,a_j} = H(N_{j-1,b_{j-1}}, N_{j-1,a_{j-1}})$ if a_{j-1} is odd and b_{j-1} is even. Finally, the verifier checks that the computed N_{J,a_J} is equal to the stored public root of the tree $N_{J,0}$.

1.4.4. Notes

The chain of nodes, $N_{J-1,b_{J-1}}, \dots, N_{1,b_1}, N_{0,b_0}$ is not exactly the path taken from the signature public key to the root public key. However, they are the nodes closest to that path and are often called the authentication path. We will call the $B = N_{J-1,b_{J-1}}, \dots, N_{1,b_1}, N_{0,b_0}$ the authentication path.

As mentioned, the Merkle hash tree is a fundamental piece of many modern hash-based signature methods. If J is large, the effort to compute the public root key is considerable, if not infeasible. The entire tree is required for the signing process, so either it is stored for later use or is reconstituted. There are many time-memory tradeoffs that appear in the literature. Most of the efficiency gains focus on transitioning from one authentication path to the next, storing only the portions of the path that will help in the next signature.

1.5. MSS Key Generation via PRNG and Tree Chaining

For a given Merkle tree, there are 2^J pairs of the form (PK_i, SK_i). If the secret keys are k bits long, then there are $k * 2^J$ bits associated with the secret keys at the leaf nodes. With 2^J leaf nodes there must be $2^J - 1$ interior nodes in the binary tree. Each of these nodes must be computed to get to the root public key of the tree. These internal nodes must be reconstituted or saved for use when a signature is created. If $J = 10$, this is not unbearably large. But it means that only 1024 messages can be signed with the given key. On the other hand, if $J = 50$, one would be able to sign any practical number of messages, but the size of the tree is unmanageable, having nearly 2^{51} nodes to keep track of to either store or reconstitute every time.

There are two primary directions for efficiency enhancements of the Merkle Signature Scheme that uses the tree approach: Key Generation via PRNGs and Tree Chaining.

1.5.1. Key Generation via PRNG

As above, assume that there are 2^J secret values each of which is k bits in length. Also let $PRNG(.)$ be a function that accepts a seed S and outputs a string of $k * 2^J$ pseudorandom bits. That string of bits may be broken into 2^J different k bit values and those values then may be used as the private keys of the tree. That is, set $PRNG(S) = SK_{2^J-1} \dots SK_1 SK_0$. Once the SK_i have been established, the PK_i may be computed as needed.

With the use of the pseudorandom function, we may replace storage of the 2^J secret values with storage of the seed S. The entire tree must be created at some point to compute the public key, but

with the seed value and the ability to save state in the $PRNG$ the entire tree never needs to be stored. Only pertinent pieces based on the number of messages signed need to be kept in memory.

A modification of this idea is to change the $PRNG$ to accept two parameters: A seed and an index. The idea is that if $PRNG(S, i)$ reconstitutes only the secret keys at index i, then one would not need to recreate the entire set of keys, rather only those needed for a given signature.

Tree Chaining

In a traditional public key system, when Alice sends a message and signature to Bob, he must have her public key to verify the signature. One common method for Alice to pass her public key to Bob is via a public key certificate. A certificate binds Alice's identity to her public key. The certificate is signed by a certification authority (CA) that Alice and Bob both trust.

In complicated networks Alice might have a chain of certificates. One certificate binds her identity and public key and is signed by a certificate authority. The next certificate binds the previous certificate authority's identity and public key and is signed by another authority. In total, the certificate chain provides a list of public keys that can be verified by the next certificate. If Bob knows the public key of the last certificate in the chain, he may trust the binding of all identities and public keys found in the chain, even if he has never heard of the certificate authorities within the chain.

Merkle hash tree chaining is a lot like a certificate chain. Each hash tree can be thought of as a certificate whose root is a public key. When a message is signed, it is signed with an OTS and the tree associated with the signature is verified. However, the root of that tree may not be known to the verifier. The next Merkle tree in the chain provides a signature of the previous root. That continues until the signatures and chains end at the public key of the tree chain. The verifier must know that public key.

As an example, suppose that Alice wishes to send a signature to Bob and that he knows $N_{0,0}$ as the public key of the base tree $T_{0,0}$. Suppose that the tree$T_{0,0}$ has height 2, so that it has four leaves. Associated with leaf $0 \leq i \leq 3$ of the base tree are two things: A OTS key pair $(PK_{0,i}, SK_{0,i})$ that Alice knows and another tree $T_{1,i}$ with its root node $N_{1,i}$. The trees in the second layer trees must all have the same height, but they need not necessarily be the same height as the base, but suppose they are all of height 2, anyhow.

Each secondary tree has four leaf nodes. For $0 \leq i \leq 3$ and tree $T_{1,i}$ and $0 \leq j \leq 3$ let $(PK_{1,i,j}, SK_{1,i,j})$ be the OTS keys for the leaves. In this configuration, a total of 16 messages can be signed.

1.5.2. Sign

To sign the 10th message, M_{10}, for instance, the signer creates $Sig(10, M_{10})$ using OTS keys $(PK_{1,2,1}, SK_{1,2,1})$ from tree T_2, providing authentication path $B_{1,2,1}$ to root node $N_{1,2}$. The root node of tree $T_{1,2}$, is $N_{1,2}$, and must be signed using OTS keys $(PK_{0,2}, SK_{0,2})$ on the base tree to create $Sig(N_{1,2})$. From there, an additional authentication path, $B_{0,2}$, to the root node $N_{0,0}$ must

also be provided. In all, the signature of M_{10} includes:
$(10, Sig(10, M_{10}), PK_{1,2,1}, B_{1,2,1}, Sig(N_{1,2}), PK_{0,2}, B_{0,2})$. The final computation uses the last nodes in the base tree to compute the root $N_{0,0}$ and a comparison with the stored value.

1.5.3. Verify

To verify the received message, the verifier checks that $Sig(10, M_{10})$ is valid via the information provided in the OTS. If so, he uses the signature values and the authentication path $B_{1,2,1}$ to compute $N_{1,2}$. After verifying the OTS $Sig(N_{1,2})$ the verifier uses that and the authentication path $B_{0,2}$ to compute the root node $N_{0,0}$ and compare with the stored value. If that final comparison is valid, then the entire chain, including the first signature is deemed valid and the message M_{10} is accepted.

1.5.4. Notes

This chaining can extend indefinitely. One may create a tree with many layers to support enough signatures for all practical purposes. However, the shape of the chained tree structure should not be changed on the fly, both signer and verifier should know the tree's shape at the time the public root key is communicated. Further, one must dutifully keep track of where in the tree the current set of signatures are. Reusing previous parts of the tree chain is disastrous for security.

The real benefit of doing the chaining is that the entire tree does not need to be constructed before the root public key can be published, which can be an absurd amount of work for a large single tree. By breaking the large tree into independent pieces, the roots of the smaller trees can be computed independently, making the entire effort manageable. For instance, a tree with four layers, each of height 20, supports 2^{4*20} signatures, but only requires work on the order of 2^{20} to create the trees and keys necessary for any given signature. The entire tree need not be constructed all at once, only the subtrees involved in the authentication path are required for any given signature and those can be created as needed. Of course, if all 2^{80} signatures are used, then all the message signing leaves have been used and all 2^{80} authentication paths must have been created at one time or another.

1.5.5. PRNG and Tree Chaining Combined

The real benefit comes from combining PRNG key generation with tree chaining. The signature keys in each subtree must come from somewhere. They could/should be randomly generated, but then they would have to be stored, which is unfeasible for a large tree. The alternative is for the signer to store the seeds for each tree and then generate the necessary keys at the time of a signature.

Or a separate master seed could be used for each layer. The key generation seed for a particular tree would include the layer master seed and the index of the tree. Or possibly, the key generation seed could include the layer master seed, the index of the tree and the index of the leaf node. As long as the key generation for one node can be accomplished independently of another node, the method will be fairly efficient from a storage and computation perspective.

Once an authentication path has been established for one signature, the next signature will, more than likely, have a very similar authentication path. If the previous path has been saved, then the new path can be a modification of the old, saving on computation during signing.

Similarly, with verification. If one path has been verified, then a set of root keys for subtrees have been computed and verified. If those are saved, they may be able to shorten the verification process of subsequent signatures. Once a root, that has been verified previously, no further work is necessary. To stop prematurely, one must make sure that the authentication path is in the correct position in the tree. The message index is included in the message in the signature, so that the verifier may stay in sync with the signer. If an adversary can cut and paste previous paths and move one tree to a new spot and have the verifier verify only part of the full authentication path, there are security problems.

1.6. BiBa [Per 01]

The signer has a list of secret numbers $SK = \{s_k, \dots, s_1, s_0\}$ called seals. In addition to the hash function H, there is a public function G. The signer publishes the hashed secret $PK = \{p_k, \dots, p_1, p_0\}$ as its public key.

1.6.1. Sign

To sign a message M, the signer first computes $h = H(M)$. Then the signer searches for $i \neq j$ such that $G(h, s_i) = G(h, s_j)$. The signer publishes (s_i, s_j) as the signature of M.

1.6.2. Verify

To verify a signature (M, s_i, s_j), the verifier computes $h = H(M)$ and checks that $G(h, s_i) = G(h, s_j)$, $p_i = H(s_i)$ and $p_j = H(s_j)$.

1.6.3. Notes

This can only work if the output of G is such that collisions can occur. It may be as simple as a truncated hash.

This is advertised as an OTS but may be used with degraded security if used as a FTS. Each time the signature is used with the same public key, there is an increasing chance that a forgery is possible. One way to combat this is for the signer and verifier to have a common state of what has been sent and never reuse anything that has already been revealed. Synchronicity becomes the base of the security protocol.

Either one discards revealed Seals or reuses them. Reuse comes with a security degradation. Discarding Seals upon use reduces the chance that the signer will be able to find a collision. There becomes a growing chance that the Signer will not be able to sign a message.

1.7. HORS [REY 02]

The authors of HORS (Hash to Obtain Random Subset) claim this was inspired by BiBa. The signer has a list of secret numbers $SK = \{s_{k-1}, \dots, s_1, s_0\}$ and publishes the hashed secrets $PK = \{p_{k-1}, \dots, p_1, p_0\}$ as its public key. Suppose that $k = 2^b$ and the size of an output of H is $n = a * b$ bits in length. A hash function output can be broken into a pieces of b bits in length each.

1.7.1. Sign

To sign a message M, the signer computes $h = H(M)$ and splits h into a equal sized pieces. That is, $h = h_{a-1}||\ ...\ ||h_0$. The h_i are b bits in length and are the correct size to index into SK. The signature of M is $(s_{h_{a-1}}, \ldots, s_{h_1}, s_{h_0})$.

1.7.2. Verify

To verify $(s_{h_1}, s_{h_2}, \ldots, s_{h_a})$ is a signature of M, the verifier computes $h = H(M)$ and splits $h = h_{a-1}||\ ...\ ||h_0$. Then, for each element in the signature, verifies that $p_{h_i} = H(s_{h_i})$.

1.7.3. Notes

The parameters do not have to be chosen exactly as above. However, things fit nicely together, size wise, if they are.

Just like BiBa, this is really an FTS. Each time a message is signed, the security degrades. Degradation comes from the fact that the secret keys are not tossed out once they are revealed. The intent is to reuse the revealed secrets a couple times. As long as, something new is revealed, the signature may be valid.

1.8. HORS++ [PIE 03]

HORS++ is an attempt to modify the OTS nature of HORS. In the [Rey 02], there are three constructions outlining how to index into the private key. In HORS $h = H(M)$ is broken into a pieces. Each of those a pieces are used to index into the SK to select the particular private keys to reveal. After a few signatures it is possible to find a message that does not reveal anything new. That message can then be signed by an unauthorized entity.

The primary method of selection for HORS uses pieces of h as integers to accomplish the index. There are other functions of the pieces that can be used.

HORS++ selection functions index into a "cover-free" family. A cover free family is a subset of the larger set. After r calls to the family, the larger set cannot be covered by the union of the responses from the r calls. So, before r messages have been signed with the same private keys, there is no way to find a special message that will use only previously revealed information. Each of the r signatures are guaranteed to reveal some new information. [Pie 03] gives a concrete construction using polynomials (and certain error correction codes) to construct its cover-free sets.

The HORS++ methods require larger keys than HORS, but security does not diminish until after r signatures.

1.9. HORSE [NEU 04]

Hash chains are a nice way to extend the FTS into a few times more signature. The idea is to have some base secret s_i and then hash it many times. With $SK = \{s_n, \ldots, s_2, s_1\}$ as the secret key decide on an appropriate m. Then for each i, set $p_i = H^m(s_i)$ and set $PK = \{p_n, \ldots, p_2, p_1\}$. Each time a signature is made, one reveals the first preimage remaining in the hash chain.

1.9.1. Sign

To sign a message, M, the signer proceeds just as in HORS with one bookkeeping step at the end. That is, he computes $h = H(M)$ and splits h into a pieces. That is, $h = h_a || \dots || h_1$. The h_i should be the correct size to index into SK. The first signature of M is $(H^{m-1}(s_{h_a}), \dots, H^{m-1}(s_{h_2}), H^{m-1}(s_{h_1}))$.

The revealed values in the signature now become the public values for the respective index. Once the first signature has been created, the secret portions of the n hash chains are of different lengths. The second signature will index into chains of length m and chains of length $m-1$. The signer reveals the preimages of the $m-1$ and $m-2$ length chains as indicated by the hashed message. Again, the revealed portions become the newest public key. The hash chains are now likely $m, m-1$ or $m-2$ long. This continues with the signer keeping track of what has been revealed in each chain. Each signature will reveal up to a values in the collection.

1.9.2. Verify

To verify the first message $(M, (H^{m-1}(s_{h_a}), \dots, H^{m-1}(s_{h_2}), H^{m-1}(s_{h_1})))$, the verifier computes $h = H(M)$ and splits $h{=}h_a || \dots || h_1$. For each element in the signature, the verifier verifies that $p_{h_i} = H(H^{m-1}(s_{h_i}))$. If the signature verifies, the public key is updated to include the revealed members of the chains: p_{h_i} is replaced by $H^{m-1}(s_{h_i})$.

1.9.3. Notes

There are book keeping requirements to ensure the hash chains remain in sync. If the Signer and Verifier get out of sync and an Adversary becomes an active man-in-the-middle delaying messages, the security of HORSE defaults to that of HORS being used multiple times. If the adversary remains a passive observer and the communicating parties stay in sync, the adversary's chance to forge a HORSE signature does not increase with the number of signatures created.

If the hash chains are exceptionally long, one may be able to sign a significant number of messages with the same key. However, the signer must compute the hash chain to find the preimage of the current public value. For long chains this becomes a computational burden. Alternatively, one could store some or all of the chain, trading computational work for storage requirements. The work for the verifier is the same either way, as long at the current public values are updated with the verified preimages each time a signature is made.

1.10. HORST [BER 15]

Bernstein et. al. [Ber 15] suggested another modification to the basic HORS scheme. They suggested that a tree be attached to the public keys. So, for each of the $k = 2^b$ leaves of a height b tree be associated with one of the k public keys of the HORS scheme.

When a message is signed there are $a = n/b$ values that index into the tree. For each of those values an authentication path to the root of the HORST tree is provided as part of the signature. The verifier verifies the signature by verifying that all paths combine into the HORST root, properly.

One can show that for a given set of parameters the HORS and the HORST scheme will sign the same number messages with the same security level.

The main benefit of HORST is that the number of keys that has to be stored and communicated during a signature is far less with HORST than with HORS.

1.11. Modern Methods up for Standardization

This section introduces three hash-based signature methods that have begun to take on a life of their own. In most cases they do not bring fundamentally new ideas into the table, but rather take foundational methods and add small modifications that enhance security, key size, and/or computational efficiency.

The methods presented in this section have two versions: The original academic version and a later attempt at standardization. In all cases, the original versions have general parameter sets and indicate generic function types. Whereas, the later specification attempts have much more specific parameter sets and may have identified specific functions to be used.

The methods discussed in this section are: LMS, XMSS, and SPHINCS. LMS and XMSS have a tree chain approaches called HSS and XMSS^{MT}, respectively. There are several documents comparing LMS and XMSS, [Kam 17] is one. There are a few documents that compare all three, [McG 16] is one.

1.11.1. LMS

LMS is the oldest of these methods [Lei 95]. The base signature scheme is the Winternitz OTS supported by a Merkle tree. The Merkle tree is modified with the inclusion of the signer's ID string, and a small-digit code in the node computation. The WOTS includes a signature of the message counter. [LMS 18] is the internet draft and has a complete specification including byte orders, parameter sizes and similar items.

It is recognized that large Merkle trees have a computational drawback. [LMS 18] includes a discussion of tree chaining in the context of LMS called HSS.

1.11.2. XMSS

The extended Merkle Signature Scheme (XMSS) scheme is a modified single tree Merkle scheme that uses a the Winternitz OTS signatures. There is a slight difference in the way XMSS constructs it Merkle trees nodes when compared to the original MSS method. For each level of the binary tree there is an additional parameter that masks inputs of the node calculations. For each layer i there is pair of bit strings $(b_{L,i}, b_{R,i})$ that are used in the computation of a nodes, so

$$N_{j,a_j} = H(N_{j-1,a_{j-1}} xor\ b_{L,j-1}\ , N_{j-1,b_{j-1}}\ xor\ b_{R,j-1})$$

Each of the $(b_{L,i}, b_{R,i})$ pairs are part of the tree's public key along with the root node. The primary purpose of the $(b_{L,i}, b_{R,i})$ is to satisfy a few very technical requirements within a proof of security.

The IETF internet draft [XMSS 18] includes a few changes over the original paper. The Winternitz OTS is changed slightly and called WOTS+. Also, the draft includes XMSS^MT that modifies the original to include the tree chaining described above.

The WOTS+ does not seem to be an actual modification of the algorithm but does include parameter values that keep track of data strings that are not kept track of in the original. The IETF document gives many target data sizes and parameter choices.

1.11.3. SPHINCS

SPHINCS was originally introduced in [Ber 15] and is also the basis of two submissions into the PQCrypto "not-competition". The main claim for SPHINCS is that it is stateless. Every FTS method described thus far in this document must carefully manage state. Signer must ensure that strict One-Time usage be enforced for OTS. Signer and Verifier must carefully synchronize the number of signatures have been completed in a FTS.

One method to obtain this stateless quality is to have a tree with, say, 2^{256} leaves. Then each time a message is hashed to 256 bits, the output is treated as an integer and becomes the index for the leaf in the tree and its signature with a OTS. Sequential use of the tree leaves is abandoned. If the hash function is collision resistant, state will not need to be maintained. It also means that the tree will be very, very large. A vast majority of the tree will never be used. Most other stateless technologies suffer the same problem, to gain the benefit of statelessness the methods have a certain amount of inefficiencies: long signatures, long keys, long signing times, etc.

SPHINCS claims to be able to remove many of the inefficiencies associated with previous stateless methods. Its keys and signatures are larger than LMS or XMSS, but do not require careful management of state. It then becomes a true public key signature scheme.

SPHINCS uses nearly every technique described up to this point, namely:

- Merkle Tree Chaining
- Winternitz One Time Signatures
- HORST Few Time Signatures
- Key Generation via PRNG

The Winternitz OTS is modified and called WOTS+. It is not clear how this version of WOTS+ relates to the WOTS+ in XMSS, but neither modification is substantial. It appears both versions of WOTS+ have extra bookkeeping measures included. In any event, the WOTS+ scheme is used to sign the subtree root keys.

HORST is used to sign the input messages. A separate HORST FTS tree must be created for each leaf node of the Merkle tree. The FTS nature of HORST means that when a message to be signed is hashed and the leaf index is chosen as a function of the hash value, it does not matter if there is a repeat in the leaf selection the HORST signature will compensate. Of course, if there are no collisions in the leaf selection, there is no repeat of any specific HORST key reuse. Each time a Merkle tree leaf is reused there is some chance that the HORST keys have been overused. Because of the FTS nature of HORST, the entire tree may be smaller than it otherwise would be.

The downside to using HORST for the leaves is that another layer of key generation must take place and signature computation times are increased to account for the FTS.

There are several revisions, refinements, and enhancements of SPHINCS in the literature from various authors. Some have been debunked, but overall the construction of SPINCS seems to be the current base method of choice to improve upon.

1.11.3.1. SPHINCS-256

SPHINCS-256 is a specific instantiation of the SPHINCS method and was introduced in the original [Ber 15] paper. It outlines all the variable sizes and indicates which hash functions are to be used and specifies all the various pieces.

The core hash function used in SPHINCS-256 is SHA-256. The need for a fast PRNG is substantial, so CHACHA is used rather than using iterated SHA's for all the PRNG key generation.

The PQCrypto SPHINCS related submissions are based on SPHINCS-256.

2. LATTICE CRYPTOGRAPHY

Lattices are mathematical sets of points embedded in real space (usually very high dimensional real space) at regular intervals in a crystalline structure, evoking the image of high-dimensional latticework extended ad infinitum. Lattices pose some very difficult mathematical problems which can be exploited to serve as the foundation for a family of cryptographic algorithms, i.e. the difficulty to subvert the algorithm is related to the difficulty of hard lattice problems. While lattices have also been used as hashing primitives and in signing schemes, this report focuses on asymmetrical cryptosystems. For what follows, the common convention of representing vectors as lowercase, bold letters and matrices as uppercase letters is observed.

Lattices are fascinatingly simple structures. An n-dimensional lattice, embedded in d-dimensional real space, can be defined in terms of linearly independent basis vectors $\{\mathbf{b}_i \mid \mathbf{b}_i \in \mathbb{R}^d, 1 \leqslant i \leqslant n\}$. These basis vectors can be written in matrix form as column vectors in $B \in \mathbb{R}^{dxn}$. The lattice may then be succinctly defined as $\mathcal{L}(B) = \{B\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^n\}$. In words, a basis is a linear operator that maps integer-valued vectors into a lattice point.

The basis for a given lattice is not unique. In fact, for any unimodular matrix $U \in \mathbb{Z}^{nxn}$, $\mathcal{L}(B) = \mathcal{L}(BU)$. An important metric to establish the orthogonality of the basis vectors is the Hadamard Ratio, defined as

$\mathcal{H}(B) = \left(\frac{|det(B)|}{\prod_{i=1}^{n} \|b_i\|}\right)^{1/n}$ where $\mathbf{b}_i$ is a basis vector. Notice that $\mathcal{H}(B) \in \mathbb{R}$ such that $0 < \mathcal{H}(B) \leq 1$. The more orthogonal the basis vectors, the closer to one is the Hadamard Ratio, the less orthogonal, the closer to zero. A very important result in connection with the Hadamard Ratio is the LLL algorithm. LLL (Lenstra-Lenstra-Lovász) is a relatively efficient, basis reduction algorithm that can transform a basis with low Hadamard Ratio into one with a much higher ratio. Furthermore, the basis vectors tend to be shorter as well. Nevertheless, for sufficiently high dimensional lattices, the LLL can be impractical.

Despite the simplicity of their definition, lattices are the root of several very hard problems:

- The Shortest Vector Problem (SVP) is the challenge to find the shortest, non-zero vector in the lattice.
- The Closest Vector Problem (CVP) is the problem of finding the lattice vector that is closest to another vector (not necessarily in the lattice).
- The Shortest Independent Vectors Problem (SIVP) is the task of finding a set of linearly independent lattice vectors that span the lattice such that the length of the longest of these vectors is minimized.

For the lattice-based cryptosystems, some of the most important lattices are the "q-ary lattice" and the "orthogonal q-ary lattice". For $A \in \mathbb{Z}_q^{nxd}$ they are defined as:

- $\Lambda_q(A) = \{\mathbf{y} \in \mathbb{Z}^d \mid (\mathbf{y} = A^T\mathbf{s}) \bmod q, \mathbf{s} \in \mathbb{Z}^n\}$
- $\Lambda_q^{\perp}(A) = \{\mathbf{y} \in \mathbb{Z}^d \mid (A\mathbf{y} = \mathbf{0}) \bmod q\}$

Additional terminology and symbolic representations in this report include:

- The probability density/mass functions, $\mathcal{F}$, of a random variable X are written as $X \sim \mathcal{F}(\cdot)$. Distributions referred to in this report include the uniform distribution, $\mathcal{U}$, and the rounded normal distribution, Ψ_σ.
- The standard basis vectors are often denoted as $\mathbf{e}_i$ where $\mathbf{e}_i$ has a "1" in the ith position and "0" everywhere else. However, in this report $\mathbf{e}$ is reserved for the error vector. Hence, to avoid confusion the standard basis vectors will be denoted as $\tilde{\mathbf{e}}_i$.
- A circulant matrix $Y = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n]$ is a square, n by n matrix generated by its first column vector $\mathbf{y}_1$ such that $\mathbf{y}_{i+1} = [\tilde{\mathbf{e}}_2, \tilde{\mathbf{e}}_3 \ldots \tilde{\mathbf{e}}_n, \tilde{\mathbf{e}}_1]\mathbf{y}_i$. In other words, $\mathbf{y}_{i+1}$ is obtained by shifting every entry in $\mathbf{y}_i$ down one position and moving the bottom entry to the top of the matrix. Hence every column vector after the first in the circulant matrix Y is a circularly cycled version of the preceding column vector.

2.1. The Cryptographic Family

Lattices as a family of cryptographic schemes are so called because an attack on such a cryptosystem has been proven to be equivalent to an efficient solution to one of the aforementioned lattice problems (SVP, CVP, SIVP). Lattices have garnered significant interest and research in recent years for various reasons but primarily for two reasons: Fully Homomorphic Encryption (FHE) and quantum-resistance.

FHE has been sought after since the emergence of asymmetrical cryptography in the 1970's but the first FHE scheme, which was based on lattice cryptography, wasn't developed until 2009. Since then there has been increased effort in FHE schemes, most of which are built upon RLWE and NTRU (lattice-based cryptosystems).

Lattice cryptography is also believed to be secure against brute-force attacks by quantum computers. In addition, lattice performance is sufficiently good for practical application. For these reasons lattice-based cryptography is considered to be viable post-quantum cryptographic schemes. In response to the progress being made in quantum computing, and in anticipation of a slow and laborious transition between standards, the U.S. National Institute of Standards and Technology (NIST) put out a call for proposals in 2016 for a new, quantum-resistant, cryptographic standard.

The submissions will be studied for half a decade and will go through multiple rounds of evaluation before a new standard is selected. It is telling, however, that among the algorithms being considered in the first round, 38% are lattice-based: the most highly represented cryptographic family (code-based is second with 30% of submissions). Arguably, this may serve as an a priori indicator of the viability of lattices as post-quantum contenders.

2.2. Common and Promising Algorithms

2.2.1. GGH

This cryptosystem, named after its creators Goldreich, Goldwasser and Halevi, relies on the ability to generate arbitrarily many bases for a given lattice. The idea is to start with a "good" basis of nearly orthogonal vectors – this will be the private key – and derive an alternative, "bad", basis of highly skewed vectors – this will be the public key. A message vector is encoded as a point in the lattice using the "bad" basis and encrypted by adding a noise vector.

2.2.1.1. The Cryptosystem

Note that for GGH cryptosystems it is traditional to transpose the basis matrix and consider horizontal vectors rather than vertical. For consistency with other cryptosystems in this report, that convention will not be followed.

For d, n, $\sigma \in \mathbb{Z}_{>0}$ define:

- Plaintext Space: $\mathbb{Z}^n$
- Cyphertext Space: $\mathbb{R}^d$
- Key Generation:
  - The *private* key is given by basis $B \in \mathbb{R}^{dxn}$ such that $\mathcal{H}(B) \approx 1$
  - The *public* key is given by basis $B' \in \mathbb{R}^{dxn}$ such that $\mathcal{H}(B') \approx 0$ and $B' = BU$ for some unimodular matrix U, i.e. $\mathcal{L}(B) = \mathcal{L}(B')$.
- Encryption: Given $\mathbf{m} \in \mathbb{Z}^n$, the cyphertext is generated by $\mathbf{c} = B'\mathbf{m} + \mathbf{e}$, where $\mathbf{e} \in \{-\sigma, \sigma\}^d$ such that $e_i \sim \mathcal{U}(\{-\sigma, \sigma\})$.
- Decryption: Given the cyphertext $\mathbf{c}$, compute $B^{-1}\mathbf{c} = B^{-1}(B'\mathbf{m} + \mathbf{e}) = B^{-1}(BU\mathbf{m} + \mathbf{e}) = B^{-1}BU\mathbf{m} + B^{-1}\mathbf{e} = U\mathbf{m} + B^{-1}\mathbf{e}$. Now pass the result of this computation into Babai's algorithm which can efficiently remove the error leaving just $U\mathbf{m}$. The plaintext can then be extracted by computing $U^{-1}U\mathbf{m} = \mathbf{m}$.

A variant of the GGH scheme swaps the roles of the message and error vectors, where the error vector is the lattice point and the message serves as noise.

2.2.1.2. Security

The idea behind GGH is that one can efficiently solve the CVP with a basis that is nearly orthogonal – i.e. with a Hadamard ratio close to one – but the problem becomes intractable with a highly skewed basis – i.e. with a Hadamard ratio close to zero. However, this system is subject to some practical attacks.

First, and in general, to make the CVP impractical to solve, one should not only choose a basis with

low Hadamard ratio and a high dimensional lattice, but also ensure that the target vector is not too close to a lattice point. From the perspective of the GGH scheme, this means choosing an error vector that is not too small.

Another attack is to transform the public key, the "bad" basis, to another basis with a Hadamard ratio closer to one. Even if the new basis isn't quite as orthogonal as the private key, using the LLL algorithm one can obtain a basis that is close enough to make the CVP tractable. Hence, to remain secure the lattice dimension should be large enough to prevent the LLL algorithm from finding a basis with sufficiently orthogonal basis vectors.

Yet another attack, delineated by Nguyen, exploits a grave weakness in the error vector as originally described by the GGH authors. The attack allows one to compute a new error vector from the original, yielding a simpler closest vector instance which can be solved far more easily. A countermeasure would be to increase the range of values the error vector can take on.

Lastly, under the GGH scheme an adversary can distinguish between two distinct messages by virtue of the different cyphertexts. In other words, GGH is not semantically secure. In this sense it is not too different from RSA and the current recommendation is the same for both: message padding.

A recommendation by Micciancio to increase security is to use the Hermite Normal Form (HNF) of the good basis to serve as the bad basis as any successful attack on a basis in HNF can easily be adapted for any other basis (i.e. an HNF basis is in the worst form).

2.2.2. NTRU

The NTRU cryptosystem (Number Theorists R Us), proposed by Hoffstein, Pipher and Silverman in 1998, was initially defined over polynomial rings. The system can be redefined in terms of lattices where it is a special case of GGH. In particular, the private key basis is a circulant matrix and the public key basis is a special case of an HNF matrix which also includes a circulant matrix. This allows both keys to be represented far more compactly. For the NTRU cryptosystem below, if there is a matrix using the same letter as a vector, then that matrix is the circulant matrix generated by that vector (i.e. A would be the circulant matrix generated by **a**).

2.2.2.1. The Cryptosystem

For n, p, q, u $\in \mathbb{Z}_{>0}$ such that n is prime, p is small and u $\leq$ (n-1)/2, define the following:

- Plaintext Space: $\{-1, 0, 1\}^n$
- Cyphertext Space: $\mathbb{Z}^n$
- Key Generation:
  - The 2n-dimensional *private* key (**f**, **g**) is a concatenation of $\mathbf{f} \in \tilde{\mathbf{e}}_1 + \{-p, 0, p\}^n$ and $\mathbf{g} \in \{-p, 0, p\}^n$ such that **f** - $\tilde{\mathbf{e}}_1$ and **g** both have u+1 positive entries and u negative entries. A further constraint is the F is invertible mod q.

- The n-dimensional *public* key is the vector $\mathbf{h} = F^{-1}\mathbf{g} \bmod q$.

- Encryption: Given a plaintext message $\mathbf{m} \in \{-1, 0, 1\}^n$ with u + 1 positive entries and u negative entries, and a random vector $\mathbf{r} \in \{-1, 0, 1\}^n$ which also contains u + 1 positive and u negative entries, the cyphertext is generated by $\mathbf{c} = (\mathbf{m} + H\mathbf{r}) \bmod q$.

- Decryption: Given the cyphertext **c**, the plaintext is retrieved by $\mathbf{m} = (F\mathbf{c} \bmod q) \bmod p$.

To see what is happening here, and to understand how it is a special case of GGH, the cryptosystem will be rephrased in the GGH framework. Consider the q-ary lattice $\Lambda_q(A^T)$ where $A = (F, G) \in \mathbb{Z}^{2nxn}$, i.e. the matrix obtained by stacking F onto G. Notice that this lattice is completely determined by the private keys **f** and **g**, which generate the equivalent of the "good" basis. Define the "bad" basis as $B' = \begin{bmatrix} I & O \\ H & qI \end{bmatrix}$ where O represents the zero matrix. Notice that this basis is completely determined by q and the vector **h**. Now, given the plaintext message **m** and the random vector **r**, concatenate the two to create a 2n-dimensional "noise" vector, **e**.

NTRU encryption is defined as $\mathbf{c} = \mathbf{e} \bmod B'$. Notice that this is equivalent to $\mathbf{c} = \mathbf{e} - B'\mathbf{x}$, for some vector $\mathbf{x} \in \mathbb{Z}^{2n}$. But $-B'\mathbf{x}$ is a point in the lattice. Hence **c** is simply a lattice point with noise added. Notice that this follows the variant of GGH where the message is encoded as the noise vector rather than the lattice point. Since both the lattice point and the noise vector have special structure, this is a special case of the GGH cryptosystem. It turns out that the structure of the cyphertext is $\mathbf{c} = \begin{bmatrix} 0 \\ (Hr + m) mod q \end{bmatrix}$. Since the coordinates in the first half of the vector are zero, they may be omitted, and we can redefine the cyphertext as the n-dimensional vector $\mathbf{c} = (H\mathbf{r} + \mathbf{m}) \bmod q$.

Before stepping through the steps of decryption, observe that:

- $FH = G \bmod q$. This can be derived from the specific way **h** was constructed, as shown earlier.

- $F \bmod p = I$

- $G \bmod p = O$

The formula for decryption is derived by observing that $F\mathbf{c} \bmod q = F((H\mathbf{r} + \mathbf{m}) \bmod q) \bmod q = F(H\mathbf{r} + \mathbf{m}) \bmod q = (FH\mathbf{r} + F\mathbf{m}) \bmod q = (G\mathbf{r} + F\mathbf{m}) \bmod q$. From this, $G\mathbf{r} + F\mathbf{m}$ can be immediately computed since all entries in $G\mathbf{r} + F\mathbf{m}$ are bounded in absolute value by q/2, as long as $u < \frac{q/2 - 2p - 1}{4p}$. Next compute $(G\mathbf{r} + F\mathbf{m}) \bmod p = O\mathbf{r} + I\mathbf{m} = \mathbf{m}$. Notice that the special structure of the cyphertext and the keys allowed for their compact representation and that the encryption/decryption processes can also be carried out more efficiently.

2.2.2.2. Security

While the NTRU system is more efficient than the general GGH system, there is no proof of

security for NTRU as it is unknown if the special structures create weaknesses not present in the general GGH case. The security then derives from the most effective attacks on the system, similar to many common encryption algorithms today, such as RSA. Furthermore, as with GGH systems in general, NTRU is not semantically secure, thus requiring message padding.

2.2.3. LWE

This subset of lattice-based cryptography is built upon the presumed hardness of the Learning With Errors (LWE) problem. This problem can be formulated as follows.

For d, n, q, r, $\in \mathbb{Z}_{>0}$ and $\sigma \in \mathbb{R}_{>0}$

- $A \in \mathbb{Z}_q^{dxn}$ such that $A_{ii} \sim \mathcal{U}(0, q-1)$
- $\mathbf{s} \in \mathbb{Z}_q^{n}$ such that $s_i \sim \mathcal{U}(0, q-1)$
- $\mathbf{e} \in \mathbb{Z}_q^{d}$ such that $e_i \sim \Psi_\sigma$
- $\mathbf{v} \in \mathbb{Z}_q^{d}$ such that either
  - $v_i \sim \mathcal{U}(0, q-1)$, or
  - $\mathbf{v} = A\mathbf{s} + \mathbf{e}$

The challenge is to discriminate between the two possible constructions of $\mathbf{v}$, given the pair $(A, \mathbf{v})$. Thus far, lattices have not seemed to play a role in the LWE problem, but notice that it can be reformulated as trying to detect the difference between a uniformly random vector $\mathbf{v}$ and a vector in the q-ary lattice $\Lambda_q(A^T)$ which has been shifted according to the probability distribution Ψ_σ. This then becomes a case of solving the CVP.

2.2.3.1. The Cryptosystem

For d, n, q, r, t, w $\in \mathbb{Z}$ and $\sigma \in \mathbb{R}_{>0}$, define the following:

- Plaintext Space: $\mathbb{Z}_t^{w}$
- Cyphertext Space: $\mathbb{Z}_q^{n} \times \mathbb{Z}_q^{w}$
- Key Generation:
  - The *private* key is given by $S \in \mathbb{Z}_q^{nxw}$ such that $S_{ii} \sim \mathcal{U}(0, q-1)$
  - The *public* key is the pair (A, P) where $A \in \mathbb{Z}_q^{dxn}$ such that $A_{ii} \sim \mathcal{U}(0, q-1)$, $P \in \mathbb{Z}_q^{dxw}$ such that $P = AS + E$ and $E \in \mathbb{Z}_q^{dxw}$ such that $E_{ii} \sim \Psi_\sigma$.
- Encryption: Define $f: \mathbb{Z}_t^{w} \rightarrow \mathbb{Z}_q^{w}$ such that for $\mathbf{m} \in \mathbb{Z}_t^{w}$, $f(m_i) = \lfloor (q/t)m_i \rceil$. Choose a vector $\mathbf{a} \in \{-r, -r+1, \ldots, 0, \ldots, r-1, r\}^d$ such that $a_i \sim \mathcal{U}(-r, r)$. The cyphertext is generated by $(\mathbf{u} = A^T\mathbf{a}, \mathbf{c} = P^T\mathbf{a} + f(\mathbf{m}))$.

- Decryption: Define $f^{-1}: \mathbb{Z}_q^w \rightarrow \mathbb{Z}_t^w$ such that $f^{-1}(c_i) = \lfloor (t/q)c_i \rceil$. Then the cleartext is retrieved by $f^{-1}(\mathbf{c} - S^T\mathbf{u}) = f^{-1}(E^T\mathbf{a} + f(\mathbf{m}))$ which equals $\mathbf{m}$ when the noise, E, is not too great.

A very common variant to LWE is R-LWE, or Ring-LWE in which the lattice consists not of vectors over the reals, but over polynomial rings.

2.2.3.2. Security

The proof of security relies on several facts:

1. If one could distinguish between the pair of public keys (A, P) and a pair of matrices (A′, P′) which are generated uniformly at random, then one could also solve the LWE problem. Since we are assuming this is not possible, we can assume that the public keys are, for all practical purposes, uniformly random.
2. If one encrypts a message using uniformly random keys (A, P) then the resulting cipher contains no statistically relevant information about the clear text message.

Combining these two facts leads to the conclusion that deriving relevant information from an LWE encrypted message would result in a solution to the LWE problem and, equivalently, to the CVP.

REFERENCES

[1] [Ber 09] D. Bernstein, J. Buchmann, E. Dahmen: *Hash-based Digital Signature Schemes*. In Post-Quantum Cryptography, pages 35-91. Springer, 2009.

[2] [Lam 79] L. Lamport: *Constructing digital signatures from a one way function*. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.

[3] [Mer 82] R. Merkle: *Secrecy, Authentication, and Public Key Systems*. UMI Research Press, 1982. (Also appears as a Stanford Ph.D. thesis in 1979.)

[4] [Mer 89] R. Merkle: *A Certified Digital Signature*. In proceedings, Advances in Cryptology - CRYPTO 1989.

[5] [Lei 95] T. Leighton, S. Micali: *Large provably fast and secure digital signature schemes from secure hash functions*, U.S. Patent 5,432,852, July 1995.

[6] [Per 01] A. Perrig: *The BiBa one-time signature and broadcast authentication protocol*. In, Eighth ACM Conference on Computer and Communication Security, pages 28–37. ACM, 2001.

[7] [Rey 02] L. Reyzin, N. Reyzin: *Better than BiBa: Short One-time Signatures with Fast Signing and Verifying*. In information Security and Privacy 2002, volume 2384 of LNCS, pages 1–47. Springer, 2002.

[8] [Pie 03] J. Pieprzyk, H. Wang, C. Xing: Multiple-time signature schemes against adaptive chosen message attacks. In Proceedings, SAC 2003. International Conference on Selected Areas in Cryptography, 2003.

[9] [Neu 04] W. Newumann: *HORSE: An Extension of an r-Time Signature Scheme With Fast Signing and Verification*. In Proceedings, ITCC 2004. International Conference on Information Technology: Coding and Computing, 2004.

[10] [Buc 11] J. Buchmann, E. Dahmen, and A. Hulsing: *XMSS – A Practical Forward Secure Signature Scheme based on Minimal Security Assumptions Second Version*. Proceedings of PQCrypto 2011

[11] [Ber 15] D. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, AND Z. Wilcox-O'Hearn: *SPHINCS: practical stateless hash-based signatures*. In Advances in Cryptology – EUROCRYPT, 2015.

[12][McG 16] https://csrc.nist.gov/csrc/media/events/ssr-2016-security-standardisation-research/documents/presentation-tue-gazdag.pdf

[13][Kam 17] https://eprint.iacr.org/2017/349.pdf

[14][LMS 18] https://datatracker.ietf.org/doc/draft-mcgrew-hash-sigs/?include_text=1

[15][XMSS 18] https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures/

This page left blank

This page left blank

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.