<u>Homomorphic Encryption Standard</u>

Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, Vinod Vaikuntanathan

January 7, 2019

We met as a group during the Homomorphic Encryption Standardization Workshop on July 13-14, 2017, hosted at Microsoft Research in Redmond, and again during the second workshop on March 15-16, 2018 in MIT. Researchers from around the world represented government, industry, and academia. There are several research groups around the world who have made libraries for general-purpose homomorphic encryption available for applications and general-purpose use. Some examples include [SEAL], [HElib], [PALISADE], [cuHE], [cuFHE], [NFLlib], [Lattigo], [HEAAN], and [TFHE]. Most general-purpose libraries for homomorphic encryption implement schemes that are based on the ring learning-with-error (RLWE) problem, and many of them display common choices for the underlying rings, error distributions, and other parameters.

Homomorphic Encryption is a breakthrough new technology which can enable private cloud storage and computation solutions, and many applications were described in the literature in the last few years. But before Homomorphic Encryption can be adopted in medical, health, and financial sectors to protect data and patient and consumer privacy, it will have to be standardized, most likely by multiple standardization bodies and government agencies. An important part of standardization is broad agreement on security levels for varying parameter sets. Although extensive research and benchmarking has been done in the research community to establish the foundations for this effort, it is hard to find all the information in one place, along with concrete parameter recommendations for applications and deployment.

This document is an attempt to capture (at least part of) the collective knowledge regarding the currently known state of security of these schemes, to specify the schemes, and to recommend a wide selection of parameters to be used for homomorphic encryption at various security levels. We describe known attacks and their estimated running times in order to make these parameter recommendations. We also describe additional features of these encryption schemes which make them useful in different applications and scenarios.

Outline of the document:

HES Section 1.1 standardizes the encryption schemes to be used. Section 1.1 consists of:

Section 1.1.1: introduces notation and definitions.
Section 1.1.2: defines the security properties for homomorphic encryption.
Section 1.1.3: describes the BGV and BFV schemes.
Section 1.1.4: described the GSW scheme.
Section 1.1.5: mentions some alternative schemes: [YASHE13], [HPS98]/[LTV12], and [CKKS17].
Section 1.1.6: describes additional features of the schemes.

HES Section 2.1 recommends parameter choices to achieve security. Section 2.1 consists of:

Section 2.1.1: describes the hard problems: the LWE and RLWE assumptions.
Section 2.1.2: describes known lattice attacks and their estimated running times.
Section 2.1.3: mentioned the Arora-Ge attack on LWE.
Section 2.1.4: discusses algebraic attacks on RLWE.
Section 2.1.5: recommends concrete parameters to achieve various security levels.

It is expected that future work to update and expand this Homomorphic Encryption Standard will use the following numbering convention:

- updates to the encryption schemes or additional schemes may be added as Sections 1.2, 1.3, …
- updates to security levels or recommended parameters may be added as Sections 2.2, 2.3, …
- a new section to cover API design is planned to be added as Section 3.0, and updated as 3.1, …
- a new section to cover applications may be added as Section 4.0, and updated as 4.1, …

In the appendix we list some aspects that are not specified in this document and are expected to be covered by future documents.

Homomorphic Encryption Standard Section 1.1
Recommended Encryption Schemes

Section 1.1.1 Notation and Definitions

- ParamGen(λ, PT, K, B) → Params

The parameter generation algorithm is used to instantiate various parameters used in the HE algorithms outlined below.

- λ denotes the desired security level of the scheme. For instance, 128-bit security ($\lambda = 128$) or 256-bit security.
- PT denotes the underlying plaintext space. Currently this standard specifies two types of parametrized plaintext spaces: modular integers (MI), and extension fields/rings (EX). We expect future versions of this document to introduce a third type of approximate numbers (AN).
  - (MI) Modular integers are parametrized by the modulus *p* of the plaintext numbers to be encrypted, namely the plaintext space is Z_p. For instance, the parameter $p = 1024$ means that the plaintext space is Z_{1024}, *i.e.,* each individual element of the message space is an integer from the range $[0, 1023]$ and all operations on individual elements are performed modulo p.
  - (EX) Extension rings/fields are parameterized by a modulus $p$as above, and in addition by a polynomial $f(x)$ over Z_p, specifying the plaintext space as $Z[x]/(p, f(x))$. Namely, each element of the message space is an integer polynomial of degree smaller than $f(x)$ with coefficients from the range $[0, p - 1]$, and all operations over individual elements are performed modulo $f(x)$, and modulo p.
- K denotes the dimension of the vectors to be encrypted. For instance, $K = 100$, $PT = (MI, 1024)$ means the messages to be encrypted are vectors $(V_1, ..., V_K)$ where each V_i is chosen from the range $[0, 1023]$ and operations are performed component-wise. That is, by definition, $(V_1, ..., V_K) + (V_1', ..., V_K') = (V_1 + V_1', ..., V_K + V_K')$. The multiplication operation over two vectors is defined similarly. The space of all possible vectors $\left(V_1, ..., V_K\right)$ is referred to as the message space (MS).
- B denotes an auxiliary parameter that is used to control the complexity of the programs/circuits that one can expect to run over the encrypted messages. Lower parameters denote “smaller”, or less expressive, or less complex programs/circuits. Lower parameters generally mean smaller parameters of the entire scheme. This, as a result, translates into smaller ciphertexts and more efficient evaluation procedures. Higher parameters generally increase key sizes, ciphertext sizes, and complexity of the evaluation procedures. Higher parameters are, of course, necessary to evaluate more complex programs.

- PubKeyGen(Params) → SK, PK, EK

The public key-generation algorithm is used to generate a pair of secret and public keys. The public key can be shared and used by anyone to encrypt messages. The secret key should be kept private by a user and can be used to decrypt messages. The algorithm also generates an evaluation key that is needed to perform homomorphic operations over the ciphertexts. It should be given to any entity that will perform homomorphic operations over the ciphertexts. Any entity that has only the public and the evaluation keys cannot learn anything about the messages from the ciphertexts only.

- SecKeyGen(Params) ⟶ SK, EK

The secret key-generation algorithm is used to generate a secret key. This secret key is needed to both encrypt and decrypt messages by the scheme. It should be kept private by the user. The algorithm also generates an evaluation key that is needed to perform homomorphic operations over the ciphertexts. The evaluation key should be given to any entity that will perform homomorphic operations over the ciphertexts. Any entity that has only the evaluation key cannot learn anything about the messages from the ciphertexts only.

- PubEncrypt(PK, M) ⟶ C

The public encryption algorithm takes as input the public key of the scheme and any message M from the message space. The algorithm outputs a ciphertext C. This algorithm generally needs to be randomized (that is, use random or pseudo-random coins) to satisfy the security properties.

- SecEncrypt(SK, M) ⟶ C

The secret encryption algorithm takes as input the secret key of the scheme and any message M from the message space. The algorithm outputs a ciphertext C. This algorithm generally needs to be randomized (that is, use random or pseudo-random coins) to satisfy the security properties.

- Decrypt(SK, C) ⟶ M

The decryption algorithm takes as input the secret key of the scheme, SK, and a ciphertext C. It outputs a message M from the message space. The algorithm may also output a special symbol FAIL, if the decryption cannot successfully recover the encrypted message M.

- EvalAdd(Params, EK, C1, C2) ⟶ C3.

EvalAdd is a randomized algorithm that takes as input the system parameters Params, the evaluation key EK, two ciphertexts C1 and C2 and outputs a ciphertext C3.

The correctness property of EvalAdd is that if C1 is an encryption of plaintext element M1 and C2 is an encryption of plaintext element M2, then C3 should be an encryption of M1+M2.

- EvalAddConst(Params, EK, C1, M2) → C3.

EvalAddConst is a randomized algorithm that takes as input the system parameters Params, the evaluation key EK, a ciphertext C1 and a plaintext M2, and outputs a ciphertext C3.

The correctness property of EvalAddConst means that if C1 is an encryption of plaintext element M1, then C3 should be an encryption of M1+M2.

- EvalMult(Params, EK, C1, C2) → C3.

EvalMult is a randomized algorithm that takes as input the system parameters Params, the evaluation key EK, two ciphertexts C1 and C2, and outputs a ciphertext C3.

The correctness property of EvalMult is that if C1 is an encryption of plaintext element M1 and C2 is an encryption of plaintext element M2, then C3 should be an encryption of M1*M2.

- EvalMultConst(Params, EK, C1, M2) → C3.

EvalMultConst is a randomized algorithm that takes as input the system parameters Params, the evaluation key EK, a ciphertext C1, and a plaintext M2, and outputs a ciphertext C3.

The correctness property of EvalMultConst is that if C1 is an encryption of plaintext element M1, then C3 should be an encryption of M1*M2.

- Refresh(Params, flag, EK, C1) → C2.

Refresh is a randomized algorithm that takes as input the system parameters Params, a multi-valued flag (which can be either one of “Relinearize”, “ModSwitch” or “Bootstrap”), the evaluation key EK, and a ciphertext C1, and outputs a ciphertext C2.

The correctness property of Refresh is that if C1 is an encryption of plaintext element M1, then C2 should be an encryption of M1 as well.

The desired property of the Refresh algorithm is that it turns a “complex” ciphertext of a message into a “simple” one encrypting the same message. Two embodiments of the Refresh algorithm are (a) the bootstrapping procedure, which takes a ciphertext with large noise and outputs a ciphertext of the same message with a fixed amount of noise; and (b) the key-switching procedure, which takes a ciphertext under one key and outputs a ciphertext of the same message under a different key.

- ValidityCheck(Params, EK, [C], COMP) → flag.

ValidityCheck is an algorithm that takes as input the system parameters Params, the evaluation key EK, an array of ciphertexts [C], and a specification of the homomorphic computation encoded as a straight-line program COMP, and outputs a Boolean flag.

The correctness property of ValidityCheck is that if ValidityCheck outputs flag = 1, then doing the homomorphic computation COMP on the vector of ciphertexts [C] produces a ciphertext that decrypts to the correct answer.

Section 1.1.2 Properties

Semantic Security or IND-CPA Security: At a high level, a homomorphic encryption scheme is said to be secure if no adversary has an advantage in guessing (with better than 50% chance) whether a given ciphertext is an encryption of either one of two equally likely distinct messages. This requires encryption to be randomized so that two different encryptions of the same message do not look the same.

Suppose a user runs the parameter and the key-generation algorithms to provide the key tuple. An adversary is assumed to have the parameters, the evaluation key EK, a public key PK (only in the public-key scheme) and can obtain encryptions of messages of its choice. The adversary is then given an encryption of one of two messages of its choice, computed by the above encryption algorithm, without knowing which message the encryption corresponds to. The security of HE then guarantees that the adversary cannot guess which message the encryption corresponds to with a significant advantage better than a 50% chance. This captures the fact that no information about the messages is revealed in the ciphertext.

Compactness: The compactness property of a homomorphic encryption scheme guarantees that homomorphic operations on the ciphertexts do not expand the length of the ciphertexts. That is, any evaluator can perform an arbitrary supported list of evaluation function calls and obtain a ciphertext in the ciphertext space (that does not depend on the complexity of the evaluated functions).

Efficient decryption: Efficient decryption property says that the homomorphic encryption scheme always guarantees that the decryption runtime does not depend on the functions which were evaluated on the ciphertexts.

Section 1.1.3. The BGV and BFV Homomorphic Encryption Schemes

In this section, we describe the two primary schemes for implementation of homomorphic encryption, [BGV12] and [B12]/[FV12], these two schemes are very similar. In Section 1.1.4. below we describe the GSW scheme, which is somewhat different. In Section 1.1.5, we also mention some alternative schemes [YASHE13], [HPS98]/[LTV12], and [CKKS17], but they are not described in this standard.

a. Brakerski-Gentry-Vaikuntanathan (BGV)

We focus here on describing the basic version of the BGV encryption scheme. Optimizations to the basic scheme will be discussed at the end of this section.

- BGV.ParamGen(λ, PT, K, B) ⟶ Params.

Recall that λ is the security level parameter, for BGV the plaintext space PT is either of type MI or EX with integer modulus $p > 1$, and $K \geq 1$ is an integer vector length.

In the basic BGV scheme, the auxiliary input B is an integer that determines the maximum multiplicative depth of the homomorphic computation. This is simply the maximum number of sequential multiplications required to perform the computation. For example, the function $g(x_1, x_2, x_3, x_4) = x_1x_2 + x_3x_4$ has multiplicative depth 1.

In the basic BGV scheme, the parameters param include the ciphertext modulus parameter q and a ring $R = Z[x]/f(x)$ and corresponding plaintext ring R/pR and ciphertext ring R/qR. The parameters param also specify a "key distribution" D_1 and an "error distribution" D_2 over R, the latter is based on a Gaussian distribution with standard deviation set according to the security guidelines specified in Section 2.1.5.

- BGV.SecKeygen(params) ⟶ SK, EK

In the basic BGV scheme, the secret key SK is an element s in the ring R chosen from distribution D_1.

In the basic BGV scheme, there is no evaluation key EK.

- BGV.PubKeygen(params) ⟶ SK, PK, EK.

In the basic BGV scheme, PubKeygen first runs SecKeygen and obtains (SK, EK) where SK is an element s that belongs to the ring R.

PubKeygen chooses a uniformly random element a from the ring R/qR and outputs the public key PK which is a pair of ring elements $(pk_0, pk_1) = (-a, as + pe)$ where e is chosen from the error distribution D_2.

- BGV.SecEncrypt(SK, M) ⟶ C

In the basic BGV scheme, SecEncrypt first maps the message M which comes from the plaintext space (either $Z_p^{\ r}$ or $(Z_p[x]/f(x))^r$) into an element $\hat{M}$ of the ring R/pR.

SecEncrypt then samples a uniformly random element a from the ring R/qR and outputs the pair of ring elements $(c_0, c_1) = (-a, as + pe + \hat{M})$ where e is chosen from the error distribution D_2. See Comments 1, 2 below for more general methods of encoding the message during encryption. The same comments apply also to public-key encryption with BGV.

- BGV.PubEncrypt(PK, M) → C

In the basic BGV scheme, Pub.Encrypt first maps the message M which comes from the plaintext space Z_p^k into an element $\hat{M}$ of the ring R/pR. Recall that the public key PK is a pair of elements (pk_0, pk_1).

PubEncrypt then samples three elements u from distribution D_1 and e_1, e_2 from the error distribution D_2 and outputs the pair of ring elements $(c_0, c_1) = (pk_0u + pe_1, pk_1u + pe_2 + \hat{M})$.

- BGV.Decrypt(SK, C) → M

In the basic BGV scheme, Decrypt takes as input the secret key which is an element s of the ring R, and a ciphertext $C = (c_0, c_1)$ which is a pair of elements from the ring R/qR.

We remark that a ciphertext C produced as the output of the encryption algorithm has two elements in R/qR, but upon homomorphic evaluation, ciphertexts can grow to have more ring elements. The decryption algorithm can be modified appropriately to handle such ciphertexts.

Decrypt first computes the ring element $c_0s + c_1$ over R/qR and interprets it as an element c' in the ring R. It then computes $c' \ (mod\ p)$, an element of R/pR, which it outputs.

- BGV.EvalAdd(Params, EK, C1, C2) → C3.

In the basic BGV scheme, EvalAdd takes as input ciphertexts $C1 = (c_{1,0}, c_{1,1})$ and $C2 = (c_{2,0}, c_{2,1})$ and outputs $C3 = (c_{1,0} + c_{2,0}, c_{1,1} + c_{2,1})$, where the operations are done in R/qR.

- BGV.EvalMult(Params, EK, C1, C2) → C3.

In the basic BGV scheme, EvalMult takes as input ciphertexts $C1 = (c_{1,0}, c_{1,1})$ and $C2 = (c_{2,0}, c_{2,1})$ and outputs $C3 = \left(c_{1,0}c_{2,0}, c_{1,0}c_{2,1} + c_{1,1}c_{2,0}, c_{1,1}c_{2,1}\right)$, where the operations are done in R/qR.

Comment 1. The noise term $pe + \hat{M}$ in the encryption procedure can be generalized to an error term drawn from the coset $\hat{M} + pR$, according to an error-sampling procedure. All the considerations discussed below for the error distribution D_2, apply equally to the error-sampling procedure in this more general implementation.

Comment 2. There is also an equivalent "MSB encoding" of the message for BGV encryption, where the message is encoded as $W\hat{M} + e$ (with $W = \lfloor q/p \rfloor$, similarly to the BFV scheme below). There are lossless conversions between these two encoding methods, as long as the plaintext modulus p is co-prime with the ciphertext modulus q.

The Full BGV Scheme

In the basic BGV scheme, ciphertexts grow as a result of EvalMult. For example, given two ciphertexts each composed of two ring elements, EvalMult as described above results in three ring elements. This can be further repeated, but has the disadvantage that upon evaluating a degree-d polynomial on the plaintexts, the resulting ciphertext has $d + 1$ ring elements.

This deficiency is mitigated in the full BGV scheme, with two additional procedures. The first is called "Key Switching" or "Relinearization" which is implemented by calling the Refresh subroutine with flag = "KeySwitch", and the second is "Modulus Switching" or "Modulus Reduction" which is implemented by calling the Refresh subroutine with flag = "ModSwitch". Support for key switching and modulus switching also necessitates augmenting the key generation algorithm.

For details on the implementation of the full BGV scheme, we refer the reader to [BGV12].

Properties Supported. The BGV scheme supports many features described in Section 6, including packed evaluations of circuits and can be extended into a threshold homomorphic encryption scheme. In terms of security, the BGV homomorphic evaluation algorithms can be augmented to provide evaluation privacy (with respect to semi-honest adversaries).

b. Brakerski/Fan-Vercauteren (BFV)

We follow the same notations as the previous section.

- BFV.ParamGen(λ, PT, K, B) ⟶ Params.

We assume the parameters are instantiated following the recommendations outlined in Section 5. Similarly to BGV, the parameters include:

- Key- and error-distributions D_1, D_2;
- a ring R and its corresponding integer modulus q;
- Integer modulus p for the plaintext.

In addition, the BFV parameters also include:

- Integers T and $L = log_T q$. T is the bit-decomposition modulus;
- Integer $W = \lfloor q/p \rfloor$.

- BFV.SecKeygen(Params) -> SK, EK

The secret key SK of the encryption scheme is a random element$s$ from the distribution D_1 defined as per Section 5. The evaluation key consists of L LWE samples encoding the secret s in a specific fashion. In particular, for $i = 1, ..., L$, sample a random a_i from R/qR and error e_i from D_2, compute

$$EK_i = \left(-\left(a_i s + e_i\right) + T^i s^2, a_i\right),$$

and set $EK = (EK_1, ..., EK_L)$.

- BFV.PubKeygen(params) -> SK, PK, EK.

The secret key SK of the encryption scheme is a random element s from the distribution D_1. The public key is a random LWE sample with the secret s. In particular, it is computed by sampling a random element a from R/qR and an error e from the distribution D_2 and setting:
$PK = (-(as + e), a)$, where all operations are performed over the ring R/qR.

The evaluation key is computed as in BFV.SecKeygen.

- BFV.PubEncrypt(PK, M) -> C

BFV.Pub.Encrypt first maps the message M which comes from the message space into an element in the ring R/pR.

To encrypt a message M from R/pR, parse the public key as a pair (pk_0, pk_1). Encryption consists of two LWE samples using a secret u where (pk_0, pk_1) is treated as public randomness. The first LWE sample encodes the message M, whereas the second sample is auxiliary.

In particular, $C = (pk_0 u + e_1 + WM, pk_1 u + e_2)$ where u is a sampled from D_1 and e_1, e_2 are sampled from D_2.

- BFV.SecEncrypt(PK, M) -> C

- BFV.Decrypt(SK, C) -> M

The main invariant of the BFV scheme is that when we interpret the elements of a ciphertext C as the coefficients of a polynomial then, $C(s) = W M + e$ for some "small" error e. The message M can be recovered by dividing the polynomial $C(s)$ by W, rounding each coefficient to the nearest integer, and reducing each coefficient modulo p.

- BFV.EvalAdd(EK, C1, C2) -> C3

Parse the ciphertexts as $Ci = (c_{i,0}, c_{i,1})$. Then, addition corresponds to component-wise addition of two ciphertext components. That is, $C3 = (c_{1,0} + c_{2,0}, c_{1,1} + c_{2,1})$.
It is easy to verify that $C3(s) = W(M_1 + M_2) + e$, where M_1, M_2 are messages encrypted in $C1, C2$ and e is the new error component.

- BFV.EvalMult(EK, C1, C2) -> C3

EvalMult takes as input ciphertexts $C1 = (c_{1,0}, c_{1,1})$ and $C2 = (c_{2,0}, c_{2,1})$. First, it computes $C3' = \left(c_{1,0}c_{2,0}, c_{1,0}c_{2,1} + c_{1,1}c_{2,0}, c_{1,1}c_{2,1}\right)$ over the integers (instead of mod q as in BGV scheme above). Then set $C3 = round(\left(\frac{p}{q}\right)C3') \ (mod\ q)$.
One can verify that $C3(s) = W(M_1 \cdot M_2) + e$, for some error term e.

Note that the ciphertext size increases in this operation. One may apply a Relinearization algorithm as in the BGV scheme to obtain a new ciphertext of the original size encrypting the same message $M_1 \cdot M_2$.

Properties Supported. The complete BFV scheme supports many features described in Section 6, including packed evaluations of circuits and can be extended into a threshold homomorphic encryption scheme. In terms of security, the BFV homomorphic evaluation algorithms can be augmented to provide evaluation privacy.

For details on the implementation of the full BFV scheme, we refer the reader to [B12], [FV12].

c. Comparison between BGV and BFV

When implementing HE schemes, there are many choices which can be made to optimize performance for different architectures and different application scenarios. This makes a direct comparison of these schemes quite challenging. A paper by Costache and Smart [CS16] gives some initial comparisons between BGV, BFV and two of the schemes described below: YASHE and LTV/NTRU. A paper by Kim and Lauter [KL15] compares the performance of the BGV and YASHE schemes in the context of applications. Since there is further ongoing work in this area, we leave this comparison as an open research question.

Section 1.1.4. The GSW Scheme and bootstrapping

Currently, the most practical homomorphic encryption schemes are limited to performing bounded depth computations. These schemes can be transformed into fully homomorphic ones (capable of arbitrary computations) using a “bootstrapping” technique introduced by Gentry [G09], which essentially entails a homomorphic evaluation of the decryption algorithm given the encryption of the secret key. Bootstrapping is a very time-consuming operation and improving on its efficiency is still a very active research area. So, it may still not be ready for standardization, but it is the next natural step to be considered.

Bootstrapping using the BGV or BFV schemes requires assuming that lattice problems are computationally hard to approximate within factors that grow *superpolynomially* in the lattice dimension n. This is a stronger assumption than the inapproximability within *polynomial* factors required by standard (non-homomorphic) lattice-based public key encryption.

In [GSW13], Gentry, Sahai and Waters proposed a new homomorphic encryption scheme (still based on lattices) that offers a different set of trade-offs than BGV and BFV. An important feature of this scheme is that it can be used to bootstrap homomorphic encryption based on the assumption that lattice problems are hard to approximate within polynomial factors. Here we briefly describe the GSW encryption and show how both its security and applicability to bootstrapping are closely related to LWE encryption, as used by the BGV and BFV schemes. So, future standardization of bootstrapping (possibly based on the GSW scheme) could build on the current standardization effort.
For simplicity, we focus on secret key encryption, as this is typically enough for applications of bootstrapping. The GSW secret key encryption scheme (or, more specifically, its secret key, ring-based variant presented in [AP14, DM15]) can be described as follows:

- GSW.Keygen(params):

  This is essentially the same as the key generation procedure of the BGV or BFV schemes, taking a similar set of security parameters, and producing a random ring element *S* which serves as a secret key.

- GSW.SecEncrypt(S,M):

  Choose an uniformly random vector A in $R^{2\log\log(q)}$, a small random vector E (with entries chosen independently at random from the error distribution), and output the ciphertext $C = (A, A \cdot S + E) + M \cdot G$ where $G = [I, 2\,I, ..., 2^{k-1}I]$ is a gadget matrix consisting of $k = \log(q)$ copies of the 2x2 identity matrix I (over the ring), scaled by powers of 2.

We note that there are other possibilities for choosing the gadget matrix *G* above (for example the constants 2, 4, ..., 2^{k-1} can be replaced by others). Other choices may be described in future documents.

We omit the description of the decryption procedure, as it is not needed for bootstrapping. Notice that:

- The secret key generation process is the same as most other LWE-based encryption schemes, including BGV and BFV.
- The encryption procedure essentially consists of $2\log(q)$ independent application of the basic LWE/BGV/BFV encryption: choose random key elements a and e, and outputs $(a, as + e + m)$, but applied to scaled copies of the message $m = 2^i M$. (The even rows of the GSW ciphertext

encrypt the message as $(a + m,\ as + e)$, but this is just a minor variant on LWE encryption, and equivalent to it from a security standpoint.)

- Security rests on the standard LWE assumption, as used also by BGV and BFV, which says that the distribution $(A,\ A \cdot S + E)$ is pseudorandom.

So, GSW can be based on LWE security estimates similar to those used to instantiate the BGV or BFV cryptosystems.

In [GSW13] it is shown how (a public key version of) this cryptosystem supports both addition and multiplication, without the need for an evaluation key, which has applications to identity-based and attribute-based homomorphic encryption. Later, in [BV14] it was observed how the GSW multiplication operation exhibits an asymmetric noise growth that can be exploited to implement bootstrapping based on the hardness of approximating lattice problems within polynomial factors. Many subsequent papers (e.g., [AP14, DM15, GINX16, CGGI16]) improve on the efficiency of [BV14], but they all share the following features with [BV14]:

- They all use variants of the GSW encryption to implement bootstrapping.
- Security only relies on the hardness of approximating lattice problems within polynomial factors.
- They are capable of bootstrapping any LWE-based encryption scheme, i.e., any scheme which includes an LWE encryption of the message as part of the ciphertext. LWE-based schemes include BGV, BFV and GSW.

In particular, GSW can be used to implement the bootstrapping procedure for BGV and BFV and turn them into fully homomorphic encryption (FHE) schemes.

Section 1.1.5. Other Schemes

Yet Another Somewhat Homomorphic Encryption ([YASHE13]) is similar to the BGV and BFV schemes and offers the same set of features.

The scheme NTRU/Lopez-Alt-Tromer-Vaikuntanathan ([HPS98]/[LTV12]) relies on the NTRU assumption (also called the “small polynomial ratios assumption”). It offers all the features of BGV and BFV, and in addition, also offers an extension that supports multi-key homomorphism. However, it must be used with a much wider error distribution than the other schemes that are described in this document (or else it becomes insecure), and therefore it should only be used with a great deal of care. This standard does not cover security for these schemes.

Another scheme, called CKKS, with plaintext type approximate numbers, was recently proposed by Cheon, Kim, Kim and Song [CKKS17]. This scheme is not described here, but we expect future version of this standard to include it.

Section 1.1.6. Additional Features & Discussion

a. Distributed HE

Homomorphic Encryption is especially suitable to use for multiple users who may want to run computations on an aggregate of their sensitive data. For the setting of multiple users, an additional property which we call threshold-HE is desirable. In threshold-HE the key-generation algorithms, encryption and decryption algorithms are replaced by a distributed-key-generation (DKG) algorithm, distributed-encryption (DE) and distributed-decryption (DD) algorithms. Both the distributed-key-generation algorithm and the distributed-decryption algorithm are executed via an interactive process among the participating users. The evaluation algorithms EvalAdd, EvalMult, EvalMultConst, EvalAddConst and Refresh remain unchanged.

We will now describe the functionality of the new algorithms.

We begin with the distributed-key-generation (DKG) algorithm to be implemented by an interactive protocol among t parties $p_1, \ldots, p_t$. The DKG algorithm is a randomized algorithm. The inputs to DKG are: security parameter, number of parties t, and threshold parameter d. The output of DKG is a vector of secret keys $s = (s_1, \ldots, s_t)$ of dimension t and a public evaluation key EK where party p_i receives (EK,s_i). We remark that party p_i doesn't receive s_j for $i \neq j$ and party i should maintain the secrecy of its secret key s_i.

Next, the distributed-encryption (DE) algorithm is described. The DE algorithm is a randomized algorithm which can be run by any party p_i. The inputs to DE run by party p_i are: the secret key s_i and the plaintext M. The output of DE is a ciphertext C

Finally, we describe the distributed-decryption (DD) algorithm to be implemented by an interactive protocol among a subset of the t parties $p_1, \ldots, p_t$. The DD algorithm is a randomized algorithm. The inputs to DD are: a subset of secret keys $s = (s_1, \ldots, s_t)$, the threshold parameter d, and a ciphertext C. In particular, every participating party p_i provides the inputs_i. The ciphertext C can be provided by any party. The output of DD is: plaintext M.

The correctness requirement that the above algorithms should satisfy is as follows.
If at least d of the parties correctly follow the prescribed interactive protocol that implements the DD decryption algorithm, then the output of the decryption algorithm will be correct.
The security requirement is for semantic security to hold as long as fewer than d parties collude adversarially.

An example usage application for (DKG, DE, DD) is for two hospitals, $t = 2$ and $d = 2$ with sensitive data sets M_1 and M_2(respectively) who want to compute some analytics F on the joint data set without revealing anything about M_1 and M_2 except for what is revealed by $F\left(M_1, M_2\right)$.

In such a case the two hospitals execute the interactive protocol for DKG and obtain their respective secret keys s_1 and s_2 and the evaluation key EK. They each use DE on secret key s_i and data M_i to produce ciphertext Ci. The evaluation algorithms on C1, C2 and the evaluation key EK allow the computation of a ciphertext C which is an encryption of $F(M_1, M_2)$. Now, the hospitals execute the interactive protocol DD using their secret keys and ciphertext C to obtain $F(M_1, M_2)$.

b. Active Attacks

One can consider stronger security requirements beyond semantic security. For example, consider an attack on a client that holds data M and wishes to compute $F(M)$ for a specified algorithm F, and wants to outsource the computation of $F(M)$ to a cloud, while maintaining the privacy of M. The client encrypts M into ciphertext C and hands C to the cloud server. The server is supposed to use the evaluation algorithms to compute a ciphertext C' which is an encryption of $F(M)$ and return this to the client for decryption.
Suppose that instead the cloud computes some other C'' which is the encryption of $G(M)$ for some other function G. This may be problematic to the client as it would introduce errors of potentially significant consequences. This is an example of an active attack which is not ruled out by semantic security.

Another, possibly even more severe attack, is the situation where the adversary somehow gains the ability to decrypt certain ciphertexts, or glean some information about their content (perhaps by watching the external behavior of the client after decrypting them). This may make it possible to the attacker to mount (perhaps limited) *chosen-ciphertext attacks*, which may make it possible to compromise the security of encrypted data. Such attacks are not addressed by the semantic security guarantee, countering them requires additional measures beyond the use of homomorphic encryption.

c. Evaluation Privacy

A desirable additional security property beyond semantic security would be that the ciphertext C hides which computations were performed homomorphically to obtain C. We call this security requirement *Evaluation Privacy.* For example, suppose a cloud service offers a service in the form of computing a proprietary machine learning algorithm F on the client's sensitive data. As before, the client encrypts its data M to obtain C and sends the cloud C and the evaluation key EK. The cloud now computes C' which is an encryption of $F(M)$ to hand back to the client. Evaluation privacy will guarantee that C' does not reveal anything about the algorithm F which is not derivable from the pair $(M, F(M))$. Here we can also distinguish between semi-honest and malicious evaluation privacy depending on whether the ciphertext C is generated correctly according to the Encrypt algorithm.

A weaker requirement would be to require evaluation privacy only with respect to an adversary who does not know the secret decryption key. This may be relevant for an adversary who intercepts encrypted network traffic.

d. Key Evolution

Say that a corpus of ciphertexts encrypted under a secret key SK is held by a server, and the client who owns SK realizes that SK may have been compromised. It is desirable for an encryption scheme to have the following *key evolution* property. Allow the client to generate a new secret key SK’ which replaces SK, a new evaluation key EK’, and a transformation key TK such that: the server, given only TK and EK’, may convert all ciphertexts in the corpus to new ciphertexts which (1) can be decrypted using SK’ and (2) satisfy semantic security even for an adversary who holds SK.

Any sufficiently homomorphic encryption scheme satisfies the key evolution property as follows. Let TK be the encryption of SK under SK’. Namely, TK is a ciphertext which when decrypted using secret key SK’ yields SK. A server given TK and EK’, can convert a ciphertext C in the corpus into C’ by homomorphically evaluating the decryption process. Security follows from semantic security of the original homomorphic encryption scheme.

e. Side Channel Attacks

Side channel attacks consider adversaries who can obtain partial information about the secret key of an encryption scheme, for example by running timing attacks during the execution of the decryption algorithm. A desirable security requirement from an encryption scheme is resiliency against such attacks, often referred to as *leakage resiliency*. That is, it should be impossible to violate semantic security even in presence of side channel attacks. Naturally, leakage resilience can hold only against limited information leakage about the secret key.

An attractive feature of encryption schemes based on intractability of integer lattice problems, and in particular known HE schemes based on intractability of integer lattice problems, is that they satisfy leakage resilience to a great extent. This is in contrast to public-key cryptosystems such as RSA.

f. Identity Based Encryption

In an identity based encryption scheme it is possible to send encrypted messages to users without knowing either a public key or a secret key, but only the identity of the recipient where the identity can be a legal name or an email address.

This is possible as long as there exists a trusted party (TP) that publishes some public parameters PP and holds a master secret key MSK. A user with identity X upon authenticating herself to the TP (e.g. by showing a government issued ID), will receive a secret key SK_X that the user can use to decrypt any ciphertext that was sent to the identity X. To encrypt message M to identity X, one needs only to know the public parameters PP and X.

Identity based homomorphic encryption is a variant of public key homomorphic encryption which may be desirable.

Homomorphic Encryption Standard Section 2.1
Recommended Security Parameters

Section 2.1.1. Hard Problems

This section describes the computational problems whose hardness form the basis for the security of the homomorphic encryption schemes in this document. Known security reductions to other problems are not included here. Section 2.1.2 below describes the best currently known attacks on these problems and their concrete running times. Section 2.1.5 below recommends concrete parameter choices to achieve various security levels against currently known attacks.

a. The Learning with Errors (LWE) Problem

The LWE problem is parametrized by four parameters $(n, m,\ q,\ \chi)$, where n is a positive integer referred to as the "dimension parameter", m is "the number of samples", q is a positive integer referred to as the "modulus parameter" and χ is a probability distribution over rational integers referred to as the "error distribution".

The LWE assumption requires that the following two probability distributions are computationally indistinguishable:

Distribution 1. Choose a uniformly random matrix $m \times n$ matrix A, a uniformly random vector s from the vector space Z_q^n, and a vector e from Z^m where each coordinate is chosen from the error distribution χ. Compute $c := \ As\ +\ e$, where all computations are carried out $mod\ q$. Output $(A,\ c)$.

Distribution 2. Choose a uniformly random $m \times n$ matrix A, and a uniformly random vector c from Z_q^m. Output $(A,\ c)$.

The error distribution χ can be either a discrete Gaussian distribution over the integers, a continuous Gaussian distribution rounded to the nearest integer, or other distributions supported on small integers. We refer the reader to Section 2.1.5 for more details on particular error distributions, algorithms for sampling from these distributions, and the associated security implications. We also mention that the secret vector s can be chosen from the error distribution.

b. The Ring Learning with Errors (RLWE) Problem

The RLWE problem can be viewed as a specific case of LWE where the matrix A is chosen to have a special algebraic structure. RLWE is parametrized by parameters $(m,\ q,\ \chi)$ where m is the number of samples, as in the LWE problem above, q is a positive integer (the "modulus parameter") and χ is a probability distribution over the ring $R\ =\ Z[X]/f(X)$ (the "error distribution").

The RLWE assumption requires that the following two probability distributions are computationally indistinguishable:

Distribution 1. Choose $m + 1$ uniformly random elements $s, a_1, \ldots, a_m$ from the ring R/qR, and m more elements $e_1, \ldots, e_m$ from the ring R chosen from the error distribution χ. Compute $b_i := sa_i + e_i$ where all computations carried out over the ring R/qR. Output $\{(a_i, b_i) : i = 1, \ldots, m\}$.

Distribution 2. Choose $2m$ uniformly random elements $a_1, \ldots, a_m, b_1, \ldots, b_m$ from the ring R/qR. Output $\{(a_i, b_i) : i = 1, \ldots, m\}$.

The error distribution χ must be supported on "small" elements in the ring R (with geometry induced by the canonical embedding). For RLWE, it is important to use an error distribution that matches the specific ring R. See Section 2.1.5 for more details on the error distributions, algorithms for sampling from these distributions, and the associated security implications. Here too, the secret element *s* can be chosen from the error distribution.

c. The Module Learning with Errors (MLWE) Problem

We mention here that there is a general formulation of the learning with errors problem that captures both LWE and RLWE, as well as many other settings. In this formulation, rather than n-vectors over Z (as in LWE) or 1-vectors over $R = Z[x]/f(X)$ (as in RLWE), we work with vectors of dimension n_1 over a ring of dimension n_2, where the security parameter is related to $n_1 \cdot n_2$. This document only deals with LWE and RLWE, but we expect future versions to be extended to deal with more settings.

Section 2.1.2 Attacks on LWE and their Complexity

We review algorithms for solving the LWE problem and use them to suggest concrete parameter choices. The schemes described above all have versions based on the LWE and the RLWE assumptions. When the schemes based on RLWE are instantiated with error distributions that match the cyclotomic rings (as described later in this document), we do not currently have attacks on RLWE that are meaningfully better than the attacks on LWE. The following estimates and attacks refer to attacks on the LWE problem with the specified parameters.

Much of this section is based on the paper by Albrecht, Player, and Scott [APS15], the online *Estimator* tool which accompanies that paper, and [Alb17, AGVW17]. Indeed, we reuse text from those works here. Estimated security levels in all the tables in this section were obtained by running the *Estimator* based on its state in March 2018. The tables in this section give the best attacks (in terms of running time expressed in $\log_2$) among all known attacks as implemented by the *Estimator* tool. As attacks or implementations of attacks change, or as new attacks are found, these tables will need to be updated.

First, we describe all the attacks which give the best running times when working on parameter sizes in the range which are interesting for Homomorphic Encryption.

The LWE problem asks to recover a secret vector $s \in Z_q^n$, given a matrix $A \in Z_q^{m \times n}$ and a vector $c \in Z_q^m$ such that $As + e = c \ (mod\ q)$ for a short error vector $e \in Z_q^m$ sampled coordinate-wise from an error distribution χ. The decision variant of LWE asks to distinguish between an LWE instance (A, c) and uniformly random $(A, c) \in Z_q^{m \times n} \times Z_q^m$. To assess the security provided by a given set of parameters m, χ, q, two strategies are typically considered.

The *primal* strategy finds the closest vector to c in the integral span of columns of $A \ (mod\ q)$, i.e. it solves the corresponding *Bounded Distance Decoding* problem (BDD) directly as is explained in [LP11] and [LL15].

a. Primal (uSVP variant)

Assume that $m > n$, i.e. the number of samples available is greater than the dimension of the lattice. Writing $\left[I_n | A'\right]$ for the reduced row echelon form of $A^T \in Z_q^{n \times m}$ (with high probability and after appropriate permutation of columns), this task can be reformulated as solving the *unique Shortest Vector Problem* (uSVP) in the $m + 1$ dimensional q-ary lattice

$$\Lambda = Z^{m+1} \cdot \begin{pmatrix} I_n & A' & 0 \\ 0 & qI_{m-n} & 0 \\ c & * & t \end{pmatrix} .$$

by Kannan's embedding, with embedding factor t.

The lattice Λ has volume $t \cdot q^{m-n}$ and contains a vector of norm $\sqrt{\|e\|^2 + t^2}$ which is unusually short, i.e. the gap between the first and second Minkowski minimum $\lambda_2(\Lambda) / \lambda_1(\Lambda)$ is large. If the secret vector s is also short, there is a second established embedding reducing LWE to uSVP. By inspection, it can be seen that the vector $(\nu s \mid e \mid 1)$, for some $\nu \neq 0$, is contained in the lattice Λ of dimension $d = m + n + 1$

$$\Lambda = \left\{ x \in (\nu Z)^n \times Z^{m+1} \mid x \cdot \left(\frac{1}{\nu} A \middle| I_m \middle| - c \right)^T \equiv 0 \ (mod\ q) \right\},$$

where ν allows to balance the size of the secret and the noise. An $(n + m + 1) \times (n + m + 1)$ basis M for Λ can be constructed as

$$M = \begin{pmatrix} \nu I_n & -A & 0 \\ 0 & qI_{m-n} & 0 \\ 0 & c & 1 \end{pmatrix} .$$

To find short vectors, lattice reduction can be applied. Thus, to establish the cost of solving an LWE instance, we may consider the cost of lattice reduction for solving uSVP. In [ADPS16] it is predicted that e can be found if:

$$\sqrt{\beta/d}\ \|\ (e|1)\| \approx \sqrt{\beta}\,\sigma \leq \delta_0^{2\beta-d} Vol(\Lambda)^{1/d},$$

where δ_0 denotes the root Hermite factor achievable by BKZ, which depends on β which is the block size of the underlying blockwise lattice reduction algorithm. This prediction was experimentally verified in [Alb17].

b. Primal by BDD Enumeration (decoding).

This attack is due to Lindner and Peikert [LP11]. It starts with a sufficiently reduced basis, e.g., using BKZ in block size β, and then applies a modified version of the recursive *Nearest Plane* algorithm due to Babai [Bab86]. Given a basis B and a target vector t, the Nearest Plane algorithm finds a vector such that the error vector lies in the fundamental parallelepiped of the Gram-Schmidt orthogonalization (GSO) of B.

Lindner and Peikert note that for a BKZ-reduced basis B, the fundamental parallelepiped is long and thin, by the Geometric Series Assumption (GSA) due to Schnorr that the GSO of a BKZ-reduced basis decay geometrically and this makes the probability that the Gaussian error vector e falls in the corresponding fundamental parallelepiped very low. To improve this success probability, they “fatten” the parallelepiped by essentially scaling its principal axes. They do this by running the Nearest Plane algorithm on several distinct planes at each level of recursion. For a Gaussian error vector, the probability that it falls in this fattened parallelepiped is expressed in terms of the scaling factors and the lengths of the GSO of B. This can be seen as a form of pruned CVP enumeration (Liu & Nguyen, 2013).

The run time of the Nearest Planes algorithm mainly depends on the number of points enumerated, which is the product of the scaling factors. The run time of the basis reduction step depends on the quality of the reduced basis, expressed, for instance, by the root Hermite factor δ_0. The scaling factors and the quality of the basis together determine the success probability of the attack. Hence to maximize the success probability, the scaling factors are determined based on the (predicted) quality of the BKZ-reduced basis. There is no closed formula for the scaling factors. The *Estimator* uses a simple greedy algorithm to find these parameters due to ([LP11]), but this is known to not be optimal. The scaling factors and the quality of the basis are chosen to achieve a target success probability and to minimize the running time (by balancing the running time of BKZ reduction and the final enumeration step).

c. Dual.

The dual strategy finds short vectors in the lattice

$$q\Lambda^* = \left\{x \in Z_q^m \mid x \cdot A \equiv 0\ (mod\ q)\right\},$$

i.e. it solves the *Short Integer Solutions* problem (SIS). Given such a short vector v, we can decide if an instance is LWE by computing $\langle v, c\rangle = \langle v, e\rangle$ which is short whenever v and e are sufficiently short [MR09].

We must however ensure that $\langle v, e\rangle$ indeed is short enough, since if is too large, the (Gaussian) distribution of will be too flat to distinguish from random. Following [LP11], for an LWE instance with parameters n, α, q and a vector v of length $\|v\|$ such that $v \cdot A \equiv 0 \ (mod\ q)$, the advantage of distinguishing $\langle v, e\rangle$ from random is close to

$$exp\left(-\ \pi(\|v\| \cdot \alpha)^2\right).$$

To produce a short enough v, we may again call a lattice-reduction algorithm. In particular, we may call the BKZ algorithm with block size β. After performing BKZ-β reduction the first vector in the transformed lattice basis will have norm $\delta_0^m \cdot Vol\left(q\Lambda^*\right)^{1/m}$. In our case, the expression above simplifies to $\|v\| \approx \delta_0^m \cdot q^{n/m}$ whp. The minimum of this expression is attained at $m = \sqrt{\frac{n \log q}{\log \delta_0}}$ (see [MR09]). The attack can be modified to take small or sparse secrets into account [Alb17].

Lattice Reduction algorithm: BKZ

BKZ is an iterative, block-wise algorithm for basis reduction. It requires solving the SVP problem (using sieving or enumeration, say) in a smaller dimension β, the block size. First, the input lattice Λ is LLL reduced, giving a basis $b_0, \ldots, b_{n-1}$. For $0 \le i < n$, the vectors $b_i, \ \ldots, \ b_{min(i+\beta-1, n-1)}$ are projected onto the orthogonal complement of the span of $b_0, \ldots b_{i-1}$; this projection is called a local block. In the local block, we find a shortest vector, view it as a vector $b \in \Lambda$ of and perform LLL on the list of vectors $b_i, \ldots, b_{min(i+\beta-1, n-1)}, b$ to remove linear dependencies. We use the resulting vectors to update $b_i, \ldots, b_{min(i+\beta-1, n-1)}$. This process is repeated until a basis is not updated after a full pass.

There have been improvements to BKZ, which are collectively referred to BKZ 2.0 (see [CN11] for example). There are currently several different assumptions in the literature about the cost of running BKZ, distinguished by how conservative they are, the “sieve” and “ADPS16” cost models, as explained below. In our use of the *Estimator* we rely on the cost model in the “sieve” implementation, as it seems the most relevant to the parameter sizes which we use for Homomorphic Encryption.

a. Block Size.

To establish the required block size β, we solve

$$\log \delta_0 = \log\left(\frac{\beta}{2\pi e}(\pi\beta)^{1/\beta}\right) \cdot \frac{1}{2(\beta-1)}$$

for β, see the PhD Thesis of Yuanmi Chen [Che13] for a justification of this.

b. Cost of SVP.

Several algorithms can be used to realize the SVP oracle inside BKZ. Asymptotically, the fastest known algorithms are sieving algorithms. The fastest, known classical algorithm runs in time $2^{0.292\beta+o(\beta)}$ (see [BDGL16]). The fastest, known quantum algorithm runs in time $2^{0.265\beta+o(\beta)}$ [Laa15].

The "sieve" estimate approximates $o(\beta)$ by 16.4 based on experimental evidence in [BDGL16]. The "ADPS16" from [ADPS16] suppresses the $o(\beta)$ term completely. All times are expressed in elementary bit operations.

c. Calls to SVP.

The BKZ algorithm proceeds by repeatedly calling an oracle for computing a shortest vector on a smaller lattice of dimension β. In each "tour" on a d-dimensional lattice, d such calls are made and the algorithm is typically terminated once it stops making sufficient progress in reducing the basis. Experimentally, it has been established that only the first few tours make significant progress [Che13], so the "sieve" cost model assumes that one BKZ call costs as much as $8d$ calls to the SVP oracle. However, it seems plausible that the cost of these calls can be amortized across different calls, which is why the "ADPS16" cost model from [ADPS16] assumes the cost of BKZ to be the same as *one* SVP oracle call, which is a strict *underestimate* of the attack cost.

d. BKZ Cost.

In summary:

sieve
a call to BKZ-β costs $8d \cdot 2^{0.292\beta+16.4}$ operations classically and $8d \cdot 2^{0.265\beta+16.4}$ operations quantumly.

ADPS16
a call to BKZ-β costs $2^{0.292\beta}$ operations classically and $2^{0.265\beta}$ operations quantumly.

We stress that both of these cost models are very conservative, and that no known implementation of lattice reduction achieves these running times. Furthermore, these estimates completely ignore memory consumption, which, too, is $2^{\Theta(\beta)}$.

e. Calls to BKZ.

To pick parameters, we normalize running times to a fixed success probability. That is, all our expected costs are for an adversary winning with probability 51%. However, as mentioned above, it is often more efficient to run some algorithm many times with parameters that have a low probability of success instead of running the same algorithm under parameter choices which ensure a high probability of success.

2.1.3 The Arora-Ge Attack.

The effectiveness of the lattice attacks above depend on the size of the error and the modulus q, in contrast Arora and Ge described in [AG11] an attack whose complexity depends only on the size of the error and poly-logarithmically on the modulus q. Very roughly, for dimension n and noise of magnitude bounded by some positive integer d in each coordinate, the attack uses $n^{O(d)}$ samples and takes $n^{O(d)}$ operations in the ring of integers modulo q. For the relevant range of parameters for homomorphic encryption, this attack performs worse than the above lattice attacks even when the error standard deviation is a small constant (e.g., $\sigma = 2$).

2.1.4 Algebraic Attacks on instances of Ring-LWE

In practice the ring R is taken to be the ring of integers in a cyclotomic field, $R = Z[x]/\Phi_k(x)$, where Φ_k is the cyclotomic polynomial for the cyclotomic index k, and the degree of Φ_k is equal to the dimension of the lattice, $n = \phi(k)$ where ϕ is the Euler totient function.

As mentioned above, for ring-LWE the choice of the error distribution matters, and there are known examples of natural high-entropy error distributions that are insecure to use in certain rings. Such examples were first given in [ELOS15] and [CLS15], and were subsequently improved in [CIV16a], [CIV16b], and [CLS16]. For example, in [CLS15] it was shown that for a prime cyclotomic index m, choosing the coefficients of the error polynomial $e \in Z[x]/\Phi_k(x)$ independently at random from a distribution of standard deviation sufficiently smaller than $\sqrt{k}$, can sometimes make this instance of RLWE easy to solve. It is therefore crucial to select an error distribution that "matches" the ring at hand.

The form of the error distribution for general cyclotomic rings was investigated, e.g., in [LPR13, DD12, LPR13b, P16]. We summarize these results in Section 2.1.5 below, but the current document only specifies concrete parameters for power-of-two cyclotomic fields, i.e. $k = 2^l$. We expect future versions of this document to extend the treatment also for generic cyclotomic rings. We stress that when the error is chosen from a sufficiently wide and "well spread" distributions that match the ring at hand, we do not have meaningful attacks on RLWE that are better than LWE attacks, regardless of the ring. For power-of-two cyclotomics, it is sufficient to sample the noise in the polynomial basis, namely choosing the coefficients of the error polynomial $e \in Z[x]/\Phi_k(x)$ independently at random from a very "narrow" distribution.

2.1.5 Secure Parameter Selection for Ring LWE

Specifying a Ring-LWE scheme for encryption requires specifying a ring, R, of a given dimension, n, along with a ciphertext modulus q, and a choice for the error distribution and a choice for a secret distribution.

Ring. In practice, we take the ring R to be a cyclotomic ring $R = Z[x]/\Phi_k(x)$, where m is the cyclotomic index and $n = \phi(k)$ is the ring dimension. For example, a power of 2 cyclotomic with index $k = 2^l$ is $R = Z[x]/(x^{k/2} + 1)$, of degree $n = k/2 = 2^{l-1}$.

Error distribution, power-of-two cyclotomics. For the special case of power-of-two cyclotomics, it is safe to sample the error in the polynomial basis, namely choosing the coefficients of the error polynomial $e(x) \in Z[x]/(x^{k/2} + 1)$ independently at random from a very "narrow" distribution. Specifically, it is sufficient to choose each coefficient from a Discrete Gaussian distribution (or even rounded continuous Gaussian distribution) with a small constant standard deviation σ. Selecting the error according to a Discrete Gaussian distribution is described more often in the literature, but choosing from a rounded continuous Gaussian is easier to implement (in particular when timing attacks need to be countered).

The LWE attacks mentioned above, however, do not take advantage of the shape of the error distribution, only the standard deviation. Moreover, the security reductions do not apply to the case where the error standard deviation is a small constant and would instead require that the error standard deviation grows at least as n^{ϵ} for some constant $\epsilon > 1/2$ (or even $\epsilon > 3/4$). The analysis of the security levels given below relies on running time estimates which assume that the shape of the error distribution is Gaussian.

The standard deviation that we use below is chosen as $\sigma = 8/\sqrt{2\pi} \approx 3.2$, which is a value that is used in many libraries in practice and for which no other attacks are known (some proposals in the literature suggest even smaller values of σ). Over time, if our understanding of the error standard deviation improves, or new attacks are found, the standard deviation of the error may have to change.

Error distribution, general cyclotomics. For non-power-of-two cyclotomics, choosing a spherical error in the polynomial basis (i.e., choosing the coefficients independently) may be insecure. Instead, there are two main methods of choosing a safe error polynomial for the general case:

The method described in [DD12] begins by choosing an "extended" error polynomial $e' \in Q[X]/(\Theta_k(x))$, where $\Theta_k(x) = x^k - 1$ if k is odd, and $x^{k/2} + 1$ if k is even. The rational coefficients of e' are chosen independently at random from the continuous Gaussian of standard deviation $\sigma\sqrt{k}$ (for the same σ as above), and with sufficient precision, e.g., using double float numbers. Then, the error is computed as

$$e = Round(e' \bmod \Phi_k(x))$$

- The method described in [CP16] chooses an error of the form $e = Round(e' \cdot t_k)$, where $t_k \in R$ is a fixed ring element (see below), and e' is chosen from a spherical continuous Gaussian distribution in the canonical embedding, of standard deviation σ (for the same σ as above). One way of sampling such error polynomial is to choose a spherical e' in the canonical embedding, then multiply by t_k and round, but there are much more efficient methods of sampling the error (cf. [CP16]).

  Note that the error so generated may not be very small, since t_k is not tiny. It is possible to show that e is somewhat small, but moreover it is shown in [CP16] that homomorphic computations can be carried out to maintain the invariant that e/t_k is small (rather than the invariant that e itself is small).

The element t_k is a generator of the "different ideal", and it is only defined up to multiplication by a unit, so implementations have some choice for which specific element to use. One option is $t_k(x) = \Phi_k'(x)$ (i.e., the formal derivative of $\Phi_k(x)$), but other options may lead to more efficient implementations.

We stress that this document does not make recommendations on the specific parameters to use for non-power-of-two cyclotomic rings, in particular Tables 1-4 below only apply to power-of-two cyclotomic rings.

Secret key. For most homomorphic encryption schemes, not only the error but also the secret key must be small. The security reductions ensure that choosing the key from the same distribution as the error does not weaken the scheme. However, for many homomorphic encryption schemes (including BGV and BFV), choosing an even smaller secret key has a significant performance advantage. For example, one may choose the secret key from the *ternary distribution* (i.e., each coefficient is chosen uniformly from $\{-1, 0, 1\}$). In the recommended parameters given below, we present tables for three choices of secret-distribution: uniform, the error distribution, and ternary.

In some extreme cases, there is a reason to choose an even smaller secret key, e.g., one with sparse coefficient vector. However, we will not present tables for sparse secrets because the security implications of using such sparse secrets is not well understood yet. We expect to specify concrete parameters for sparse secret keys in future versions of this standard.

Number of samples. For most of the attacks listed in the tables below, the adversary needs a large number of LWE samples to apply the attack with maximum efficiency. Collecting many samples may be feasible in realistic systems, since from one ring-LWE sample one can extract many "LWE-like" samples. The evaluation keys may also contain some samples.

Sampling Methods. All the error distributions mentioned above require choosing the coefficients of some initial vector independently at random from either the discrete or the continuous Gaussian with some standard deviation $\sigma > 0$. Sampling from a continuous Gaussian with small parameter is quite straightforward, but sampling from a discrete Gaussian distribution is harder. There are several known methods to sample from a discrete Gaussian, including rejection sampling, inversion sampling, Discrete Zuggurat, Bernoulli-type, Knuth-Yao, and Von Neumann-type. For efficiency, we recommend the Von Neumann-type sampling method introduced by Karney in [Kar16].

Constant-time sampling. In some of the aforementioned sampling methods, the time it takes to generate one sample could leak information about the actual sample. In many applications, it is therefore important that the entire error-sampling process is constant-time. This is easier to do when sampling from the continuous Gaussian distribution, but harder for the discrete Gaussian. One possible method is to fix some upper bound $T > 0$ such that sampling all the n coordinates e_i sequentially without interruption takes time less than T time with overwhelming probability. Then after these samples are generated, using time t, we wait for $(T - t)$ time units, so that the entire error-generating time always takes time T. In this way, the total time does not reveal information about the generated error polynomial.

TABLES of RECOMMENDED PARAMETERS

In practice, in order to implement homomorphic encryption for a particular application or task, the application will have to select a dimension n, and a ciphertext modulus q, (along with a plaintext modulus and a choice of encoding which are not discussed here). For that reason, we give pairs of (n, q) which achieve different security levels for each n. In other words, given n, the table below recommends a value of q which will achieve a given level of security (e.g. 128 bits) for the given error standard deviation $\sigma \approx 3.2$.

We have the following tables for 3 different security levels, 128-bit, 192-bit, and 256-bit security, where the secret follows the uniform, error, and ternary distributions. For applications, we give values of n from $n = 2^k$ where $k = 10, \ldots, 15$. We note that we used commit (560525) of the LWE-estimator of [APS15], which the authors continue to develop and improve. The tables give estimated running times (in bits) for the three attacks described in Section 5.1: uSVP, dec (decoding attack), and dual.

Table 1: Cost model = BKZ.sieve

distribution	n	security level	logq	uSVP	dec	dual
uniform	1024	128	29	131.2	145.9	161.0
		192	21	192.5	225.3	247.2
		256	16	265.8	332.6	356.7
	2048	128	56	129.8	137.9	148.2
		192	39	197.6	217.5	233.7
		256	31	258.6	294.3	314.5
	4096	128	111	128.2	132.0	139.5
		192	77	194.7	205.5	216.4
		256	60	260.4	280.4	295.1
	8192	128	220	128.5	130.1	136.3
		192	154	192.2	197.5	205.3
		256	120	256.5	267.3	277.5
	16384	128	440	128.1	129.0	133.9
		192	307	192.1	194.7	201.0
		256	239	256.6	261.6	269.3
	32768	128	880	128.8	129.1	133.6
		192	612	193.0	193.9	198.2
		256	478	256.4	258.8	265.1

distribution	n	security level	logq	uSVP	dec	dual
error	1024	128	29	131.2	145.9	141.8
		192	21	192.5	225.3	210.2
		256	16	265.8	332.6	300.5
	2048	128	56	129.8	137.9	135.7
		192	39	197.6	217.5	209.6
		256	31	258.6	294.3	280.3
	4096	128	111	128.2	132.0	131.4
		192	77	194.7	205.5	201.5
		256	60	260.4	280.4	270.1
	8192	128	220	128.5	130.1	130.1
		192	154	192.2	197.5	196.9
		256	120	256.5	267.3	263.8
	16384	128	440	128.1	129.3	130.2
		192	307	192.1	194.7	196.2
		256	239	256.6	261.6	264.5
	32768	128	883	128.5	128.8	130.0
		192	613	192.7	193.6	193.4
		256	478	256.4	258.8	257.9

distribution	n	security level	logq	uSVP	dec	dual
(-1, 1)	1024	128	27	131.6	160.2	138.7
		192	19	193.0	259.5	207.7
		256	14	265.6	406.4	293.8
	2048	128	54	129.7	144.4	134.2
		192	37	197.5	233.0	207.8
		256	29	259.1	321.7	273.5
	4096	128	109	128.1	134.9	129.9
		192	75	194.7	212.2	198.5
		256	58	260.4	292.6	270.1
	8192	128	218	128.5	131.5	129.2
		192	152	192.2	200.4	194.6
		256	118	256.7	273.0	260.6
	16384	128	438	128.1	129.9	129.0
		192	305	192.1	196.2	193.2
		256	237	256.9	264.2	259.8
	32768	128	881	128.5	129.1	128.5
		192	611	192.7	194.2	193.7
		256	476	256.4	260.2	258.2

Post-quantum security. The BKZ.qsieve model assumes access to a quantum computer and gives lower estimates than BKZ.sieve. In what follows, we give tables of recommended ("Post-quantum") parameters which achieve the desired levels of security against a quantum computer. We also present tables computed using the ``quantum" mode of the BKZ.ADPS16 model, which contain more conservative parameters.

Table 2: Cost model = BKZ.qsieve

distribution	**n**	**security level**	**logq**	**uSVP**	**dec**	**dual**
uniform	1024	128	27	132.2	149.3	164.5
		192	19	199.3	241.6	261.6
		256	15	262.9	341.1	360.8
	2048	128	53	128.1	137.6	147.6
		192	37	193.6	215.8	231.4
		256	29	257.2	297.9	316.6
	4096	128	103	129.1	134.2	141.7
		192	72	193.8	206.2	217.2
		256	56	259.2	281.9	296.5
	8192	128	206	128.2	130.7	136.6
		192	143	192.9	199.3	207.3
		256	111	258.4	270.8	280.7
	16384	128	413	128.2	129.0	132.7
		192	286	192.1	195.3	201.4
		256	222	257.2	263.1	270.6
	32768	128	829	128.1	128.4	130.8
		192	573	192.0	193.3	197.5
		256	445	256.1	259.0	265.2

distribution	n	security level	logq	uSVP	dec	dual
error	1024	128	27	132.2	149.3	144.5
		192	19	199.3	241.6	224.0
		256	15	262.9	341.1	302.3
	2048	128	53	128.1	137.6	134.8
		192	37	193.6	215.8	206.7
		256	29	257.2	297.9	281.4
	4096	128	103	129.1	134.2	133.1
		192	72	193.8	206.2	201.8
		256	56	259.2	281.9	270.4
	8192	128	206	128.2	130.7	130.1
		192	143	192.9	199.3	198.5
		256	111	258.4	270.8	266.6
	16384	128	413	128.2	129.0	130.1
		192	286	192.1	195.3	196.6
		256	222	257.2	263.1	265.8
	32768	128	829	128.1	128.4	129.8
		192	573	192.0	193.3	192.8
		256	445	256.1	259.0	260.4

distribution	n	security level	logq	uSVP	dec	dual
(-1, 1)	1024	128	25	132.6	165.5	142.3
		192	17	199.9	284.1	222.2
		256	13	262.6	423.1	296.6
	2048	128	51	128.6	144.3	133.4
		192	35	193.5	231.9	205.2
		256	27	257.1	327.8	274.4
	4096	128	101	129.6	137.4	131.5
		192	70	193.7	213.6	198.8
		256	54	259.7	295.2	270.6
	8192	128	202	129.8	130.7	128.0
		192	141	192.9	202.5	196.1
		256	109	258.3	276.6	263.1
	16384	128	411	128.2	129.5	129.0
		192	284	192.0	196.8	193.7
		256	220	257.2	265.8	260.7
	32768	128	827	128.1	128.7	128.4
		192	571	192.0	194.1	193.1
		256	443	256.1	260.4	260.4

Appendix A

Organizers

Kristin Lauter	klauter@microsoft.com
Vinod Vaikuntanathan	vinod.nathan@gmail.com

Contributors

Martin Albrecht	martinralbrecht@googlemail.com
Melissa Chase	melissac@microsoft.com
Hao Chen	haoche@microsoft.com
Jintai Ding	jintai.ding@gmail.com
Shafi Goldwasser	shafi@theory.csail.mit.edu
Sergey Gorbunov	sgorbunov100@gmail.com
Shai Halevi	shaih@alum.mit.edu
Jeffrey Hoffstein	hoffsteinjeffrey@gmail.com
Satya Lokam	Satya.Lokam@microsoft.com
Daniele Micciancio	daniele@cs.ucsd.edu
Dustin Moody	dustin.moody@nist.gov
Travis Morrison	txm950@psu.edu
Amit Sahai	amitsahai@gmail.com

References

[Alb17] Albrecht, M. R. (2017). On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In J. Coron & J. B. Nielsen (Eds.), *EUROCRYPT 2017, part ii* (Vol. 10211, pp. 103–129). Springer, Heidelberg.

[AFG14] Martin R. Albrecht, Robert Fitzpatrick, and Florian Gopfert: *On the Efficacy of Solving LWE by Reduction to Unique-SVP*. In Hyang-Sook Lee and Dong-Guk Han, editors, ICISC 13, volume 8565 of LNCS, pages 293-310. Springer, November 2014.

[AGVW17] Albrecht, M. R., Göpfert, F., Virdia, F., & Wunderer, T. (2017). Revisiting the expected cost of solving uSVP and applications to LWE. In T. Takagi & T. Peyrin (Eds.), *ASIACRYPT 2017, part i* (Vol. 10624, pp. 297–322). Springer, Heidelberg.

[APS15] Martin R. Albrecht, Rachel Player and Sam Scott. *On the concrete hardness of Learning with Errors*. Journal of Mathematical Cryptology. Volume 9, Issue 3, Pages 169–203, ISSN (Online) 1862-2984, October 2015.

[ADPS16] Alkim, E., Ducas, L., Pöppelmann, T., & Schwabe, P. (2016). Post-quantum key exchange - A new hope. In T. Holz & S. Savage (Eds.), *25th USENIX security symposium, USENIX security 16* (pp. 327–343). USENIX Association. Retrieved from https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim

[AP14] Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 297–314.

[AG11] Sanjeev Arora and Rong Ge. *New algorithms for learning in the presence of errors.* In ICALP, volume 6755 of Lecture Notes in Computer Science, pages 403–415. Springer, 2011.

[Bab86] László Babai: *On Lovász' lattice reduction and the nearest lattice point problem,* Combinatorica, 6(1):1-3, 1986.

[BDGL16] Becker, A., Ducas, L., Gama, N., & Laarhoven, T. (2016). New directions in nearest neighbor searching with applications to lattice sieving. In R. Krauthgamer (Ed.), *27th soda* (pp. 10–24). ACM-SIAM. https://doi.org/10.1137/1.9781611974331.ch2

[BGV12]: Zvika Brakerski, Craig Gentry, Vinod Vaikuntanathan. *(Leveled) fully homomorphic encryption without bootstrapping.* In ITCS '12 Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. Pages 309-325.

[B12] Zvika Brakerski. *Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP,* In CRYPTO 2012. Pages 868 – 886.

[CIV16a] W. Castryck, I. Iliashenko, F. Vercauteren, *Provably weak instances of ring-lwe revisited.* In: Eurocrypt 2016. vol. 9665, pp. 147–167. Springer (2016)

[CIV16b] W. Castryck, I. Iliashenko, F. Vercauteren, *On error distributions in ring-based LWE.* LMS Journal of Computation and Mathematics 19(A), 130–145 (2016) 7.

[Che13] Chen, Y. (2013). *Réduction de réseau et sécurité concrète du chiffrement complètement homomorphe* (PhD thesis). Paris 7.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, Yongsoo Song. *Homomorphic encryption for arithmetic of approximate numbers*, In *International Conference on the Theory and Applications of Cryptology and Information Security*, pp. 409–437. Springer, Cham. 2017.

[CLS15] Hao Chen, Kristin Lauter, Katherine E. Stange, *Attacks on the Search RLWE Problem with Small Errors*, SIAM J. Appl. Algebra Geometry, Society for Industrial and Applied Mathematics, Vol. 1, pp. 665–682. (2017) https://eprint.iacr.org/2015/971

[CLS16] Hao Chen, Kristin Lauter, Katherine E. Stange. *Security Considerations for Galois Non-dual RLWE Families, SAC 2016: Selected Areas in Cryptography – SAC 2016* Lecture Notes in Computer Science, Vol. 10532. Springer pp 443-462.

[CN11] Y. Chen, P.Q. Nguyen. *BKZ 2.0: Better Lattice Security Estimates*. In: Lee D.H., Wang X. (eds) Advances in Cryptology – ASIACRYPT 2011. ASIACRYPT 2011. Lecture Notes in Computer Science, vol. 7073. Springer, Berlin, Heidelberg.

[CGGI16] Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 3–33.

[CS16] Ana Costache, Nigel P. Smart, *Which Ring Based Somewhat Homomorphic Encryption Scheme is Best?* Topics in Cryptology - CT-RSA 2016, LNCS, volume 9610, Pages 325-340.

[CP16] Eric Crockett and Chris Peikert. *Λ∘λ: Functional Lattice Cryptography*. In ACM-CCS 2016.

[DM15] Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 617–640.

[ELOS15] Yara Elias, Kristin Lauter, Ekin Ozman, Katherine E. Stange*, Provably weak instances of Ring-LWE,* CRYPTO 2015

[FV12] J. Fan and F. Vercauteren. *Somewhat practical fully homomorphic encryption*. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144.pdf

[GINX16] Gama, N., Izabachène, M., Nguyen, P.Q., Xie, X.: Structural lattice reduction: generalized worst-case to average-case reductions. In: EUROCRYPT 2016, https://eprint.iacr.org/2014/283.pdf

[GSW] C. Gentry, A. Sahai, and B. Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based*. In *CRYPTO 2013* (Springer).

[Kar16] C.F.F. Karney, *Sampling Exactly from the Normal Distribution*. ACM Transactions on Mathematical Software, 42, Article No. 3.

[KL15] Miran Kim and Kristin Lauter, *Private Genome Analysis through Homomorphic Encryption,* BioMedCentral Journal of Medical Informatics and Decision Making 2015 15 (Suppl 5)**:** S3.

[LL15] Kim Laine and Kristin Lauter, Key Recovery for LWE in Polynomial Time. https://eprint.iacr.org/2015/176

[Laa15] Laarhoven, T. (2015). *Search problems in cryptography: From fingerprinting to lattice sieving* (PhD thesis). Eindhoven University of Technology.

[LMvP13] Laarhoven T., Mosca M., van de Pol J. (2013) *Solving the Shortest Vector Problem in Lattices Faster Using Quantum Search.* In: Gaborit P. (eds) Post-Quantum Cryptography. PQCrypto 2013. Lecture Notes in Computer Science, vol 7932. Springer, Berlin, Heidelberg.

[LP11] Richard Lindner and Chris Peikert: *Better key sizes (and attacks) for LWE-based encryption.* In Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, Aggelos Kiayias, Editor, volume 6558 of LNCS, pages 319—339.

[LN13] Liu, M., & Nguyen, P. Q. (2013). Solving BDD by enumeration: An update. In E. Dawson (Ed.), *CT-rsa 2013* (Vol. 7779, pp. 293–309). Springer, Heidelberg. https://doi.org/10.1007/978-3-642-36095-4_19

[LTV12] A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan. *On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption.* In STOC, pages 1219–1234, 2012.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev : *On Ideal Lattices and Learning with Errors over Rings.* Journal of the ACM (JACM), Volume 60, Issue 6, November 2013, Article No. 43.

[LPR13b] Vadim Lyubashevsky, Chris Peikert, and Oded Regev : *A toolkit for ring-LWE cryptography.* Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2013.

[MR09] Micciancio, D., & Regev, O. (2009). Lattice-based cryptography. In D. J. Bernstein, J. Buchmann, & E. Dahmen (Eds.), *Post-quantum cryptography* (pp. 147–191). Berlin, Heidelberg, New York: Springer, Heidelberg.

[HPS98] J. Hoffstein, J. Pipher, and J. H. Silverman. *NTRU: A ring-based public key cryptosystem.* In J. Buhler, editor, ANTS, volume 1423 of Lecture Notes in Computer Science, pages 267–288. Springer, 1998.

[P16] C. Peikert, *How Not to Instantiate Ring-LWE*, in SCN'16, volume 9841 of LNCS, Springer, 2016.

[YASHE13] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. *Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme*, in IMA CC 2013. http://eprint.iacr.org/2013/075.pdf

Software references for publicly available Homomorphic Encryption libraries:

[cuFHE] **https://github.com/vernamlab/cuFHE**

[cuHE] **https://github.com/vernamlab/cuHE**

[HEAAN] **https://github.com/snucrypto/HEAAN**

[HElib] **https://github.com/shaih/HElib**

[Lattigo] **https://github.com/ldsec/lattigo**

[NFLlib] **https://github.com/CryptoExperts/FV-NFLlib**

[PALISADE] **https://git.njit.edu/groups/PALISADE**

[SEAL] **http://sealcrypto.org**

[TFHE] **https://tfhe.github.io/tfhe/**

Homomorphic Encryption Standard Appendix

Anticipated Extensions to this Document

This document is only a first step in standardizing various aspects of homomorphic encryption, and we expect many other aspects to be standardized in future documents. Some aspects that were not specified here and we expect to be specified in future versions include the following:

- The homomorphic encryption scheme for approximate numbers by Cheon, Kim, Kim and Song [CKKS17], which is mentioned in Section 1.1.5.
- Homomorphic encryption based on Module LWE, mentioned in Section 2.1.1.
- Concrete parameters and sampling methods for non-power-of-two cyclotomic rings, as discussed in Section 2.1.3.
- Parameter choices when using sparse secret key, as mentioned in Section 2.1.3.