HuFu: <u>H</u>ash-and-Sign Signat<u>u</u>res <u>F</u>rom Powerf<u>u</u>l Gadgets

Algorithm Specifications and Supporting Documentation

Yang Yu[1,2], Huiwen Jia[3], Leibo Li[4], Delong Ran[1], Zhiyuan Qiu[4],
Shiduo Zhang[1], Xiuhan Lin[5], and Xiaoyun Wang[1,2,5]

[1] Tsinghua University, China
[2] Zhongguancun Laboratory, China
[3] Guangzhou University, China
[4] Shandong Institute of Blockchain, China
[5] Shandong University, China

Table of Contents

1 Introduction

HuFu is a digital signature scheme whose security is based on the hardness of standard worst-case problems on generic lattices. Besides not using structured lattices, HuFu has a fairly different design compared to CRYSTALS-Dilithium [LDK+20] and Falcon [PFH+20]. At a high level, HuFu is a hash-and-sign signature scheme constructed using the lattice trapdoor framework proposed by Gentry, Peikert and Vaikuntanathan [GPV08]. It is instantiated on hard random lattices following the gadget trapdoor construction [MP12] and using the compact gadget technique [YJW23] to achieve overall good performance. In a nutshell, the ingredients of HuFu can be described as follows:

$$\text{HuFu} \ = \ \text{GPV framework} \ + \ \text{Hard random lattices} \ + \ \text{Compact gadget}.$$

1.1 Lattice-based Hash-and-Sign Signatures and the GPV Framework

Hash-and-sign is one of two main paradigms (the other is Fiat-Shamir) for lattice-based signatures. A high-level description of this paradigm is as follows:

- The secret key, also called the trapdoor, is a "good" representation of a lattice Λ, which allows to efficiently find lattice points close to a random target. The public key is a "bad" representation of Λ.
- To sign a message m, the signer first hashes m to a random point $\mathbf{c} = \mathsf{H}(m)$ in the ambient space, then uses the trapdoor to compute $\mathbf{v} \in \Lambda$ close to $\mathbf{c}$. The output signature is $\mathbf{s} = \mathbf{v} - \mathbf{c}$.
- Given a message m along with its signature $\mathbf{s}$, the verifier first computes $\mathbf{c} = \mathsf{H}(m)$ and then checks if $\mathbf{s}$ is short and if $\mathbf{s} + \mathbf{c} \in \Lambda$ with the public key.

Early hash-and-sign signatures used Babai's algorithm [Bab86] to find a lattice vector close to the hashed message, then each signature would leak partial information about the trapdoor. Such leakages were exploited to mount devastating attacks [NR06, DN12, YD18].

To prevent such attacks, it requires the signatures to follow some distribution independent of the secret key. The standard method is due to Gentry, Peikert and Vaikuntanathan [GPV08]. In the GPV framework, the lattice vector $\mathbf{v}$ is sampled from a distribution negligibly close to the lattice discrete Gaussian $D_{\Lambda,r,\mathbf{c}}$ for a small width r. This procedure, called trapdoor sampling, can be implemented by various Gaussian samplers [GPV08, Pei10, Pre15, DP16] as per different achieved width.

1.2 Practical GPV Hash-and-Sign Signatures

The instantiation of the GPV framework is determined by two ingredients: the trapdoor construction and the trapdoor sampler. Practical GPV hash-and-sign signatures can be basically classified into two families as per their used trapdoor constructions: NTRU trapdoor based and gadget based.

The NTRU trapdoor based GPV instantiations, dating back to [SS11], use a short NTRU basis as the trapdoor. With a series of follow-up optimizations on the trapdoor generation [DLP14, PP19, EFG+22] and the Gaussian sampling [DP16, Pre15], the representative schemes, e.g. Falcon and its variant Mitaka [EFG+22], have been practically efficient. In particular, Falcon and Mitaka offer a smaller bandwidth compared to other lattice-based signatures, which makes them attractive in some constrained protocol scenarios. However, the good performance of NTRU trapdoor based signatures heavily relies on the underlying polynomial ring structures and the unstructured adaptions would be rather complicated and inefficient. Given the primary interest of NIST in *"additional general-purpose signature schemes that are not based on structured lattices"*, the NTRU trapdoor based GPV signatures does not seem to be a promising candidate.

The gadget based GPV instantiations were originally proposed by Micciancio and Peikert [MP12]. The trapdoor in this family is a linear relation between the public matrix and the gadget matrix, which is not a full basis. Exploiting this linear relation, the trapdoor sampling on hard random lattices can be converted into the sampling on the gadget lattices, and the latter is convenient and fast. The gadget based signatures offer significant advantages from an implementation standpoint and the unstructured adaptions are straightforward. In addition, the gadget-based GPV framework turns out to be versatile for the constructions of

advanced primitives. This provides the potential of extending basic signatures to powerful cryptosystems. The downside of gadget based schemes is their larger size. Fortunately, by using the approximate trapdoor [CGM19] and the compact gadget [YJW23] techniques, the gadget-based signatures have sizes on par with their Fiat-Shamir counterparts.

1.3 Design Rationale

Given the desired long-term lifetime of deployed cryptosystems and the unpredictable progress of cryptanalysis and quantum computing, we believe it would be worthy to reduce attack surface at some efficiency cost, at least for the usecases where the assuring security is much more important than performance. In recent years, many advanced cryptographic primitives have been brought to real-world applications enriching the notion of security, thus the post-quantum standard supporting convenient extension to versatile applications would be preferable in the future. Hence strong security assurances and versatile future uses are two major considerations for the design of HuFu.

1.3.1 Standard Worst-case Problems on Generic Lattices. HuFu is based on the SIS and LWE problems that are at least as hard as standard worst-case lattice problems on generic lattices [Ajt96, Reg05]. Such conservative security assumptions avoid the risk of the algebraic attacks against ideal lattice problems [CDW17, PHS19, DPW19, BR20, BLNR22] and the sublattice attacks against NTRU [ABD16, KF17, LW20, DvW21].

1.3.2 Gadget Trapdoor Framework. HuFu is constructed within the Micciancio-Peikert gadget trapdoor framework [MP12]. As a result, HuFu has an online/offline structure and its online operations are simple, fast and fully over integers. This feature would be very beneficial to certain usecases. Furthermore, the gadget framework provides powerful versatility leading to a wide range of advanced cryptosystems such as attribute-based encryption [GVW13], group signatures [dLS18], blind signatures [dK22] and even obfuscations [HHSS17]. This makes HuFu easier to adapt to offer enhanced functionality.

1.3.3 Compact Lattice Gadget. HuFu makes use of the compact gadget technique introduced in [YJW23]. In the compact gadget framework, the gadget is a square matrix instead of the wide one used in previous constructions [MP12, CGM19], which significantly reduces the public key and signature sizes. The main algorithmic component is the semi-random sampler that is highly parallelizable and efficient. With the state-of-the-art technique, HuFu achieves good overall performance.

2 Preliminaries

2.1 Notations

Let $\mathbb{Z}_Q = \{-\lfloor Q/2 \rfloor, -\lfloor Q/2 \rfloor + 1, \cdots, Q - \lfloor Q/2 \rfloor - 1\}$ for a positive integer Q. For $a \in \mathbb{Z}$, let $(a \bmod Q)$ be the unique integer $a' \in \mathbb{Z}_Q$ such that $a = a' \bmod Q$. The notation is naturally generalized to integer vectors and integer matrices. Let $\ln$ (resp. $\log$) denote the natural (resp. base 2) logarithm.

We use bold lower-case letters to denote vectors, i.e. $\mathbf{v} = (v_1, \ldots, v_n)$. Vectors are in column form. Let $(\mathbf{v}_1, \mathbf{v}_2)$ be the concatenation of $\mathbf{v}_1$ and $\mathbf{v}_2$. Given $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, their inner product is $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{i=1}^n u_i v_i$ and the ℓ_2 norm of $\mathbf{u}$ is $\|\mathbf{u}\| = \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle}$. Matrices are denoted with bold upper-case letters, i.e. $\mathbf{A} = [\mathbf{a}_1 \mid \cdots \mid \mathbf{a}_n]$ where $\mathbf{a}_i$ is the i-th column. Let $\mathbf{I}_n$ denote the n-dimensional identity matrix. When the context is clear, we write $\mathbf{I}$ simply. Let $\mathbf{A}^t$ be the transpose of $\mathbf{A}$ and $s_1(\mathbf{A}) = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}$ be the largest singular value.

For a symmetric matrix $\boldsymbol{\Sigma}$, we write $\boldsymbol{\Sigma} \succ 0$ when $\boldsymbol{\Sigma}$ is positive definite, i.e. $\mathbf{z}^t \boldsymbol{\Sigma} \mathbf{z} > 0$ for all nonzero $\mathbf{z}$. We write $\boldsymbol{\Sigma}_1 \succ \boldsymbol{\Sigma}_2$ when $\boldsymbol{\Sigma}_1 - \boldsymbol{\Sigma}_2 \succ 0$. When $\boldsymbol{\Sigma} \succ s \cdot \mathbf{I}$, we write $\boldsymbol{\Sigma} \succ s$. We call $\mathbf{B}$ a Gram root of $\boldsymbol{\Sigma}$ if $\boldsymbol{\Sigma} = \mathbf{B}\mathbf{B}^t$. A general method to find a Gram root of $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ is the Cholesky decomposition.

Let $f(S) = \sum_{x \in S} f(x)$ when f is a real-valued function and S is a countable set, assuming this sum is absolutely convergent. For a finite set S, we use $U(S)$ to represent the uniform distribution over it. We write $a \leftarrow D$ when the sample a drawn from the distribution D. The centered binomial distribution B_η with parameter η is the distribution of $X = \sum_{i=1}^{\eta}(a_i - b_i)$ where $a_i, b_i \leftarrow U(\{0,1\})$. For $X \leftarrow B_\eta$, its expectation is 0 and its standard deviation is $\sqrt{\eta/2}$.

2.2 Cryptographic Definitions

We recall the definition of digital signature schemes and the security notion.

Definition 2.1 (Digital Signature). *A digital signature scheme $\mathcal{S}$ is a tuple of polynomial-time (possibly probabilistic) algorithms ($\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}$):*

- *$\mathsf{KeyGen}() \rightarrow (\mathsf{pk}, \mathsf{sk})$: the key generation algorithm outputs the public key pk and the secret key sk;*
- *$\mathsf{Sign}(\mathsf{m}, \mathsf{sk}) \rightarrow \mathsf{s}$: given a message m, the signing algorithm uses sk to generate a valid signature s;*
- *$\mathsf{Verify}(\mathsf{m}, \mathsf{s}, \mathsf{pk}) \rightarrow Accept/Reject$: given a message m along with a signature s, the verification algorithm uses pk to check if s is a valid signature for m. If yes, accept; otherwise reject.*

Strong Unforgeability under Chosen Message Attacks (SUF-CMA) is regarded as a standard security notion for digital signature schemes. In this security model, the adversary gets the public key and has access to a signing oracle to sign messages of his choice. The adversary wins if it produces a different signature of a message that he has already seen.

The standard security notion for digital signature schemes is UF-CMA (Unforgeability under Chosen Message Attacks). Our signature scheme achieves a slightly stronger security, SUF-CMA (Strong UF-CMA), defined as follows.

Definition 2.2 (SUF-CMA). *Let $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a digital signature scheme. A SUF-CMA adversary $\mathcal{A}$ has access to the public key and a signing oracle. It makes N queries and collects N message-signature pairs $(\mathsf{m}_i, \mathsf{s}_i)$'s. Then $\mathcal{A}$ asks to generate a new message-signature pair $(\mathsf{m}^*, \mathsf{s}^*)$. The advantage of $\mathcal{A}$ is $Adv_{\mathcal{S}}^{SUF\text{-}CMA}(\mathcal{A}) =$*

$$\Pr\left[(\mathsf{m}^*, \mathsf{s}^*) \notin \{(\mathsf{m}_i, \mathsf{s}_i)\}_i \text{ and } \mathsf{Verify}(\mathsf{m}^*, \mathsf{s}^*, \mathsf{pk}) = Accept \mid (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(), (\mathsf{m}^*, \mathsf{s}^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot, \mathsf{sk})}\right].$$

2.3 Lattices and Gaussians

A lattice Λ is the set $\{\mathbf{B}\mathbf{z} \mid \mathbf{z} \in \mathbb{Z}^n\}$ for some full-rank $\mathbf{B} \in \mathbb{R}^{n \times n}$. We call n the dimension and $\mathbf{B}$ a basis of Λ. Given $\boldsymbol{\Sigma} \succ 0$, let $\rho_{\boldsymbol{\Sigma}}(\mathbf{x}) = \exp\left(-\frac{1}{2}\mathbf{x}^t \boldsymbol{\Sigma}^{-1}\mathbf{x}\right)$ for any $\mathbf{x} \in \mathrm{span}(\boldsymbol{\Sigma})$. For an n-dimensional lattice Λ, a positive definite $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ and $\mathbf{c} \in \mathrm{span}(\Lambda)$, the discrete Gaussian distribution $D_{\Lambda+\mathbf{c},\boldsymbol{\Sigma}}$ is defined as: for any $\mathbf{x} \in \Lambda+\mathbf{c}$, $D_{\Lambda+\mathbf{c},\boldsymbol{\Sigma}}(\mathbf{x}) = \frac{\rho_{\boldsymbol{\Sigma}}(\mathbf{x})}{\rho_{\boldsymbol{\Sigma}}(\Lambda+\mathbf{c})}$. When $\boldsymbol{\Sigma} = s^2\mathbf{I}$, we simply write ρ_{s^2} and $D_{\Lambda+\mathbf{c},s^2}$. Let $\eta'_\epsilon(\mathbb{Z}^n) = \frac{1}{\pi}\sqrt{\frac{\ln(2n(1+1/\epsilon))}{2}}$ be an upper bound of the smoothing parameter (scaled by $\sqrt{2\pi}$) of $\mathbb{Z}^n$.

Let $D_{\mathbb{Z},r^2}^+$ be the half integer Gaussian defined by $\rho_{r^2}(x)/\rho_{r^2}(\mathbb{N})$ for any $x \in \mathbb{N}$. We denote $\mathcal{N}_k(\mathbf{c}, \boldsymbol{\Sigma})$ as the k-dimensional normal distribution with center $\mathbf{c}$ and covariance $\boldsymbol{\Sigma}$. If $\mathbf{c} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}$, we write $\mathcal{N}_k(\mathbf{c}, \boldsymbol{\Sigma})$ as $\mathcal{N}_k$.

2.4 SIS and LWE

The SIS (Short Integer Solution) problem and the LWE (Learning With Errors) problem are two widely-used hardness assumptions in lattice-based cryptography. As shown in [Ajt96, Reg05], the average-case SIS and LWE problems are at least as hard as the worst-case hard problems on lattices, which sets a firm theoretical grounding for lattice-based schemes.

Definition 2.3 (SIS and inhomogeneous SIS). *Let $n, m, Q > 0$ be integers and $B > 0$.*

- $SIS_{n,m,Q,B}$: *Given a uniformly random* $\mathbf{A} \in \mathbb{Z}_Q^{n \times m}$, *find non-zero* $\mathbf{x} \in \mathbb{Z}^m$ *such that* $\mathbf{Ax} = \mathbf{0} \bmod Q$ *and* $\|\mathbf{x}\| \leq B$.
- $ISIS_{n,m,Q,B}$: *Given a uniformly random* $\mathbf{A} \in \mathbb{Z}_Q^{n \times m}$ *and* $\mathbf{y} \in \mathbb{Z}_Q^n$, *find* $\mathbf{x} \in \mathbb{Z}^m$ *such that* $\mathbf{Ax} = \mathbf{y} \bmod Q$ *and* $\|\mathbf{x}\| \leq B$.

The matrix $\mathbf{A}$ in SIS and ISIS problems can be in the Hermite Normal Form (HNF), i.e. $\mathbf{A} = [\mathbf{I}_n \mid \mathbf{A}']$. This gives the HNF version of SIS problems, HNF.SIS and HNF.ISIS that are as hard as the standard version.

Definition 2.4 (LWE). *Let* $n, m, Q > 0$ *be integers and* χ *be a distribution over* $\mathbb{Z}$. *Given* $\mathbf{s} \in \mathbb{Z}_Q^n$, *let* $A_{\mathbf{s},\chi}$ *be the distribution of* $(\mathbf{a}, b)$ *where* $\mathbf{a} \leftarrow U(\mathbb{Z}_Q^n)$ *and* $b = \langle \mathbf{a}, \mathbf{s} \rangle + e \bmod Q$ *with* $e \leftarrow \chi$. *The* $LWE_{n,m,Q,\chi}$ *problem is defined as follows: given* m *independent samples from either* $A_{\mathbf{s},\chi}$ *with* $\mathbf{s} \leftarrow \chi$ *(fixed for all* m *samples) or* $U(\mathbb{Z}_Q^n \times \mathbb{Z}_Q)$, *distinguish which is the case.*

3 Specification

This section gives a complete specification of the HuFu signature scheme. We first summarize the parameters and notations associated with HuFu in Table 3.

	Description
(m, n)	matrix dimensions
(p, q)	gadget parameters
Q	global modulus $Q = pq$
χ	secret distribution $\chi = B_1$
$\bar{r}$	base Gaussian parameter $\bar{r} = \eta'_\epsilon(\mathbb{Z})$ with $\epsilon = 2^{-49}$
r	gadget Gaussian parameter $r = q\bar{r}$
σ	standard deviation of the preimage $\sigma = 1.05 \cdot r\sqrt{\frac{1+q^2}{2q^2}}(\sqrt{m} + \sqrt{n+m})$
B	acceptance bound $B = \left\lfloor 1.04\sqrt{(n+2m)\sigma^2 + \frac{m(p^2-1)}{12}} \right\rfloor$
L	maximal bitlength of encoded signatures
$(\mathbf{S}, \mathbf{E})$	secret matrices $\mathbf{S} \leftarrow \chi^{n \times m}, \mathbf{E} \leftarrow \chi^{m \times m}$
$(\hat{\mathbf{A}}, \mathbf{B})$	public matrices $\hat{\mathbf{A}} \leftarrow U(\mathbb{Z}_Q^{m \times n})$, $\mathbf{B} = p \cdot \mathbf{I} - (\hat{\mathbf{A}}\mathbf{S} + \mathbf{E}) \bmod Q$
$\mathbf{A}$	parity check matrix $\mathbf{A} = [\mathbf{I}\ \hat{\mathbf{A}}\ \mathbf{B}]$ such that $\mathbf{A}\begin{bmatrix}\mathbf{E}\\\mathbf{S}\\\mathbf{I}\end{bmatrix} = p \cdot \mathbf{I}$
$\mathbf{\Sigma_p}$	perturbation covariance $\mathbf{\Sigma_p} = \sigma^2 \mathbf{I}_{n+2m} - r^2 \cdot \begin{bmatrix}\mathbf{E}\\\mathbf{S}\\\mathbf{I}_m\end{bmatrix} \cdot [\mathbf{E}^t\ \mathbf{S}^t\ \mathbf{I}_m] \succ \bar{r}^2$
$\mathbf{C} = \begin{bmatrix}\mathbf{L}_{33}\ \mathbf{L}_{32}\ \mathbf{L}_{31}\\ \ \ \mathbf{L}_{22}\ \mathbf{L}_{21}\\ \ \ \ \ \mathbf{L}_{11}\end{bmatrix}$	Gram root of $\mathbf{\Sigma_p} - \bar{r}^2\mathbf{I}$, $\mathbf{L}_{22} \in \mathbb{R}^{n \times n}, \mathbf{L}_{32} \in \mathbb{R}^{m \times n}, \mathbf{L}_{33} \in \mathbb{R}^{m \times m}$
$\mathsf{seed}_{\hat{\mathbf{A}}}$	seed for generating $\hat{\mathbf{A}}$
XOF	ideal extendable-output function
H	hash function $\{0,1\}^* \to \mathbb{Z}_Q^m$
salt	salt $\mathsf{salt} \leftarrow U(\{0,1\}^{320})$

Table 1. Description of parameters and notations.

3.1 Key Generation

HuFu uses the LWE-style key pair. Its secret key is $(\mathbf{S}, \mathbf{E}) \leftarrow \chi^{n \times m} \times \chi^{m \times m}$ and the public key is essentially $(\hat{\mathbf{A}}, \mathbf{B} = p \cdot \mathbf{I} - (\hat{\mathbf{A}}\mathbf{S} + \mathbf{E}) \bmod Q)$ where $\hat{\mathbf{A}}$ is uniformly random over $\mathbb{Z}_Q^{m \times n}$ and $p \cdot \mathbf{I}$ is the gadget matrix. The

matrix $\hat{\mathbf{A}}$ is generated by an ideal extendable-output function XOF (Algorithms 2, 3) with a 32-byte seed $\mathsf{seed}_{\hat{\mathbf{A}}}$. For compactness, it is stored as $\mathsf{seed}_{\hat{\mathbf{A}}}$. The key generation also uses BlockCholesky (Algorithm 4), a block variant of Cholesky decomposition, to compute a Gram root $\mathbf{C}$ of $\mathbf{\Sigma_p} - \bar{r}^2\mathbf{I}$ and $\mathbf{C}$ is included as a part of the secret key. Exploiting the structure of $\mathbf{\Sigma_p} - \bar{r}^2\mathbf{I}$, the BlockCholesky algorithm converts the $(2m+n)$-dimensional decomposition into one m-dimensional and one n-dimensional Cholesky decomposition along with one n-dimensional matrix inversion. A formal description of the key generation is given in Algorithm 1.

Algorithm 1: KeyGen

Input: None
Output: public key pk, secret key sk
1: $\mathsf{seed}_{\hat{\mathbf{A}}} \leftarrow U(\{0,1\}^{256})$, $\hat{\mathbf{A}} \leftarrow \mathsf{XOF}(\mathsf{seed}_{\hat{\mathbf{A}}})$
2: **repeat**
3:    $(\mathbf{S}, \mathbf{E}) \leftarrow \chi^{n \times m} \times \chi^{m \times m}$
4:    $\mathbf{\Sigma_p} = \sigma^2\mathbf{I}_{n+2m} - r^2 \cdot \begin{bmatrix} \mathbf{E} \\ \mathbf{S} \\ \mathbf{I}_m \end{bmatrix} \cdot [\mathbf{E}^t\ \mathbf{S}^t\ \mathbf{I}_m]$
5: **until** $\mathbf{\Sigma_p} \succ \bar{r}^2$
6: $\mathbf{B} = p \cdot \mathbf{I} - (\hat{\mathbf{A}}\mathbf{S} + \mathbf{E}) \bmod Q$
7: $\mathbf{C} = \begin{bmatrix} \mathbf{L}_{33}\ \mathbf{L}_{32}\ \mathbf{L}_{31} \\ \quad\mathbf{L}_{22}\ \mathbf{L}_{21} \\ \qquad\quad\mathbf{L}_{11} \end{bmatrix} \leftarrow \mathsf{BlockCholesky}(\mathbf{\Sigma_p} - \bar{r}^2\mathbf{I})$ with $\mathbf{L}_{22} \in \mathbb{R}^{n \times n}, \mathbf{L}_{32} \in \mathbb{R}^{m \times n}, \mathbf{L}_{33} \in \mathbb{R}^{m \times m}$
8: **return** $\mathsf{pk} = (\mathsf{seed}_{\hat{\mathbf{A}}}, \mathbf{B}), \mathsf{sk} = (\mathbf{E}, \mathbf{S}, \mathbf{L}_{22}, \mathbf{L}_{32}, \mathbf{L}_{33})$

Algorithm 2: XOF based on AES256

Input: a 32-byte seed seed
Output: a matrix $\hat{\mathbf{A}} \in \mathbb{Z}_Q^{m \times n}$
1: $b \leftarrow (Q \gg 16)$
2: **for** $i = 0, 1, \ldots, (nmb \gg 3) - 1$ **do**
3:    $(\mathbf{c}_{i,0}, \mathbf{c}_{i,1}, \ldots, \mathbf{c}_{i,8/b-1}) \leftarrow \mathsf{AES256}_{\mathsf{seed}}(i)$ with $\mathbf{c}_{i,j} \in \{0,1\}^{16 \cdot b}$
4:    **for** $j = 0, 1, \ldots, 8/b - 1$ **do**
5:      $k \leftarrow (8/b) \cdot i + j$, $i' \leftarrow \lfloor k/n \rfloor$, $j' = k - i'n$
6:      $\hat{\mathbf{A}}_{i',j'} \leftarrow \langle \mathbf{c}_{i,j}, \mathbf{g} \rangle$ with $\mathbf{g} = (Q/2, Q/4, \ldots, 1, 0, \ldots, 0)$
7:    **end for**
8: **end for**
9: **return** $\hat{\mathbf{A}}$

Algorithm 3: XOF based on SHAKE256

Input: a 32-byte seed seed
Output: a matrix $\hat{\mathbf{A}} \in \mathbb{Z}_Q^{m \times n}$
1: $b \leftarrow (Q \gg 16)$, $d \leftarrow 2mnb$
2: **for** $i = 0, 1, \ldots, 3$ **do**
3:    $(\mathbf{c}_{i,0}, \mathbf{c}_{i,1}, \ldots, \mathbf{c}_{i,(mn \gg 2)-1}) \leftarrow \mathsf{SHAKE256}(\mathsf{seed}||i, (d \gg 2))$ with $\mathbf{c}_{i,j} \in \{0,1\}^{16 \cdot b}$
4:    **for** $j = 0, 1, \ldots, (mn \gg 2) - 1$ **do**
5:      $k \leftarrow (mn \gg 2) \cdot i + j$, $i' \leftarrow \lfloor k/n \rfloor$, $j' = k - i'n$
6:      $\hat{\mathbf{A}}_{i',j'} \leftarrow \langle \mathbf{c}_{i,j}, \mathbf{g} \rangle$ with $\mathbf{g} = (Q/2, Q/4, \ldots, 1, 0, \ldots, 0)$
7:    **end for**
8: **end for**
9: **return** $\hat{\mathbf{A}}$

Algorithm 4: BlockCholesky

Input: a real matrix $\boldsymbol{\Sigma}_{\mathbf{p}} = \sigma^2 \mathbf{I}_{n+2m} - r^2 \cdot \begin{bmatrix} \mathbf{E} \\ \mathbf{S} \\ \mathbf{I}_m \end{bmatrix} \cdot [\mathbf{E}^t \ \mathbf{S}^t \ \mathbf{I}_m] \succ \bar{r}^2 \mathbf{I}$

Output: some Gram root $\mathbf{C}$ of $\boldsymbol{\Sigma}_{\mathbf{p}} - \bar{r}^2 \mathbf{I}$

1: $x \leftarrow \sqrt{\sigma^2 - r^2 - \bar{r}^2}$, $y \leftarrow \sqrt{\sigma^2 - \bar{r}^2}$

2: $\mathbf{L}_{11} \leftarrow x \cdot \mathbf{I}_m$, $\mathbf{L}_{21} = -\frac{r^2}{x}\mathbf{S}$, $\mathbf{L}_{31} = -\frac{r^2}{x}\mathbf{E}$

3: $\mathbf{L}_{22} \leftarrow \mathsf{Cholesky}(y^2\mathbf{I} - \frac{r^2y^2}{x^2}\mathbf{SS}^t)$ {Cholesky denotes the standard Cholesky decomposition}

4: $\mathbf{L}_{32} \leftarrow -\frac{r^2y^2}{x^2} \cdot \mathbf{ES}^t(\mathbf{L}_{22}^t)^{-1}$

5: $\mathbf{L}_{33} \leftarrow \mathsf{Cholesky}(y^2\mathbf{I} - \frac{r^2y^2}{x^2}\mathbf{EE}^t - \mathbf{L}_{32}\mathbf{L}_{32}^t)$

6: **return** $\mathbf{C} = \begin{bmatrix} \mathbf{L}_{33} & \mathbf{L}_{32} & \mathbf{L}_{31} \\ & \mathbf{L}_{22} & \mathbf{L}_{21} \\ & & \mathbf{L}_{11} \end{bmatrix}$

3.2 Signature Generation

Given a message m, the signing procedure (shown in Algorithm 5) outputs a short preimage $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ such that $\mathbf{Ax} = \mathsf{H}(\mathsf{m}, \mathsf{salt}) - \mathbf{e} \bmod Q$ for random salt and small $\mathbf{e}$. The signing procedure consists of two phases: offline and online, following the idea of [Pei10, MP12]. In the offline (message independent) phase, it samples an integer perturbation vector $\mathbf{p}$ from $D_{\mathbb{Z}^{n+2m}, \boldsymbol{\Sigma}_{\mathbf{p}}}$. Then in the online (message dependent) phase, it produces an approximate preimage using the semi-random sampling technique [YJW23]. The output signature is essentially $(\mathsf{salt}, (\mathbf{x}_1, \mathbf{x}_2))$, as the short term $(\mathbf{x}_0 + \mathbf{e}) = \mathsf{H}(\mathsf{m}, \mathsf{salt}) - \hat{\mathbf{A}}\mathbf{x}_1 - \mathbf{B}\mathbf{x}_2 \bmod Q$ can be recovered during verification. For compactness, we use some encoding technique to compress $(\mathbf{x}_1, \mathbf{x}_2)$.

Algorithm 5: Sign

Input: a message m, the secret key sk and the acceptance bound B

Output: a signature $\mathsf{s} = (\mathsf{salt}, \mathsf{str})$

Offline phase:

1: $\hat{\mathbf{A}} \leftarrow \mathsf{XOF}(\mathsf{seed}_{\hat{\mathbf{A}}})$, $\mathbf{A} \leftarrow [\mathbf{I} \ \hat{\mathbf{A}} \ \mathbf{B}]$

2: $\mathbf{p} = (\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2) \leftarrow \mathsf{SampleP}(\mathsf{sk})$ with $\mathbf{p}_0 \in \mathbb{Z}^m, \mathbf{p}_1 \in \mathbb{Z}^n, \mathbf{p}_2 \in \mathbb{Z}^m$ $\{\mathbf{p} \leftarrow D_{\mathbb{Z}^{n+2m}, \boldsymbol{\Sigma}_{\mathbf{p}}}\}$

3: $\mathbf{c} \leftarrow \mathbf{Ap} \bmod Q$

Online phase:

4: $\mathsf{salt} \leftarrow U(\{0,1\}^{320})$, $\mathbf{u} \leftarrow \mathsf{H}(\mathsf{m}, \mathsf{salt})$

5: $\mathbf{v} \leftarrow \mathbf{u} - \mathbf{c} \bmod Q$

6: $\mathbf{e} \leftarrow (\mathbf{v} \bmod p)$, $\mathbf{v}' \leftarrow (\mathbf{v} - \mathbf{e})/p$

7: **for** $i = 1, \ldots, m$ **do**

8: $\quad z_i \leftarrow q \cdot \mathsf{SampleZ}_d(v_i'/q)$ $\{\mathbf{z} \leftarrow D_{q \cdot \mathbf{I}_m + \mathbf{v}', r^2}\}$

9: **end for**

10: $\mathbf{x}_0 \leftarrow \mathbf{Ez} + \mathbf{p}_0$, $\mathbf{x}_1 \leftarrow \mathbf{Sz} + \mathbf{p}_1$, $\mathbf{x}_2 \leftarrow \mathbf{z} + \mathbf{p}_2$

11: **if** $\|(\mathbf{x}_0 + \mathbf{e}, \mathbf{x}_1, \mathbf{x}_2)\| > B$ **then**

12: $\quad$ restart

13: **end if**

14: $\mathsf{str} \leftarrow \mathsf{Compress}((\mathbf{x}_1, \mathbf{x}_2))$

15: **if** $\mathsf{str} = \perp$ **then**

16: $\quad$ restart

17: **end if**

18: **return** $\mathsf{s} = (\mathsf{salt}, \mathsf{str})$

3.2.1 Perturbation Sampler This perturbation sampling is implemented with Peikert's Gaussian convolution technique [Pei10]. It proceeds in two steps as follows. First, it samples a continuous Gaussian vector of covariance $\Sigma_{\mathbf{p}} - \bar{r}^2 \mathbf{I}$, which can be done by applying the linear transformation defined by the Gram root of $(\Sigma_{\mathbf{p}} - \bar{r}^2 \mathbf{I})$ on a vector from the normal distribution. Then it rounds the real coefficients of the continuous Gaussian vector to some near integer by the (arbitrary-centered) integer Gaussian sampler $\mathsf{SampleZ}_c$ (Algorithm 7). The detailed algorithm is shown in Algorithm 6.

Algorithm 6: $\mathsf{SampleP}$

Input: the secret key sk
Output: a perturbation vector $\mathbf{p} \leftarrow D_{\mathbb{Z}^{n+2m}, \Sigma_{\mathbf{p}}}$
1: $x \leftarrow \sqrt{\sigma^2 - r^2 - \bar{r}^2}$
2: $\mathbf{L}_{11} \leftarrow x \cdot \mathbf{I}_m, \mathbf{L}_{21} = -\frac{r^2}{x}\mathbf{S}, \mathbf{L}_{31} = -\frac{r^2}{x}\mathbf{E}$
3: $\mathbf{C} \leftarrow \begin{bmatrix} \mathbf{L}_{33} & \mathbf{L}_{32} & \mathbf{L}_{31} \\ & \mathbf{L}_{22} & \mathbf{L}_{21} \\ & & \mathbf{L}_{11} \end{bmatrix}$
4: $\mathbf{y} \leftarrow \mathcal{N}_{n+2m}$
5: $\mathbf{c} = \mathbf{C} \cdot \mathbf{y}$ $\{\mathbf{c} \leftarrow \mathcal{N}_k(\mathbf{0}, \Sigma_{\mathbf{p}} - \bar{r}^2 \mathbf{I})\}$
6: **for** $i = 1, \ldots, n + 2m$ **do**
7:   $p_i \leftarrow \mathsf{SampleZ}_c(c_i)$
8: **end for**
9: **return** $\mathbf{p}$

3.2.2 Arbitrary-centered Integer Gaussian Sampler Algorithm 7 shows the sampler for $c + D_{\mathbb{Z}-c, \bar{r}^2}$ with arbitrary center c. It is adapted from the integer sampler of FALCON [PFH+20, HPRR20]. It first samples some fixed Gaussian using table-based approach (Algorithm 8) and then uses rejection sampling to make the output following the target distribution. The rejection probability is computed by the Exp algorithm (Algorithm 9) and its subroutine $\mathsf{ApproxExp}$ (Algorithm 10).

Algorithm 7: $\mathsf{SampleZ}_c$

Input: a center c
Output: $z \leftarrow c + D_{\mathbb{Z}-c, \bar{r}^2}$
1: $d \leftarrow c - \lfloor c \rfloor$
2: $z^+ \leftarrow \mathsf{BaseSample}()$
3: $b \leftarrow U(\{0, 1\})$
4: $z \leftarrow b + (2b - 1)z^+$
5: $x \leftarrow \frac{(z-d)^2 - (z^+)^2}{2\bar{r}^2}$
6: $r \leftarrow U(\{0, 1, \ldots, 2^{64} - 1\})$
7: **if** $r > \mathsf{Exp}(x)$ **then**
8:   restart
9: **end if**
10: **return** $z + \lfloor c \rfloor$

Algorithm 8: $\mathsf{BaseSample}$

Input: None
Output: $z^+ \leftarrow D_{\mathbb{Z}, \bar{r}^2}^+$
1: $u \leftarrow U(\{0, 1\}^{72})$
2: $z^+ \leftarrow 0$;
3: **for** $i = 0, \ldots, 12$ **do**
4:   $z^+ \leftarrow z^+ + \llbracket u < RCDT[i] \rrbracket$
5: **end for**
6: **return** z^+

Algorithm 9: Exp

Input: A floating-point value x
Output: An integral approximation of $2^{64} \cdot \exp(-x)$
1: $s \leftarrow \lfloor x/\ln 2 \rfloor$
2: $r \leftarrow x - s \cdot \ln 2$
3: $s \leftarrow \min(s, 63)$
4: $z \leftarrow (2 \cdot \mathsf{ApproxExp}(r) - 1) \gg s$
5: **return** z

Algorithm 10: ApproxExp

Input: Floating point values $x \in [0, \ln(2)]$
Output: An integral approximate of $2^{63} \cdot \exp(-x)$
1: $C = [$0x00000004741183A3, 0x00000036548CFC06, 0x0000024FDCBF140A, 0x0000171D939DE045,
   0x0000D00CF58F6F84, 0x000680681CF796E3, 0x002D82D8305B0FEA, 0x011111110E066FD0,
   0x0555555555070F00, 0x155555555581FF00, 0x400000000002B400, 0x7FFFFFFFFFFFF4800,
   0x8000000000000000$]$
2: $y \leftarrow C[0]$
3: $z \leftarrow \lfloor 2^{63} \cdot x \rfloor$
4: **for** $i = 1, \cdots, 12$ **do**
5: $\quad y \leftarrow C[i] - (z \cdot y) \gg 63$
6: **end for**
7: **return** y

3.2.3 Quantized Integer Gaussian Sampler The online phase needs to sample from $D_{\mathbb{Z}+v', \bar{r}^2}$. This task can be done by the $\mathsf{SampleZ}_c$ sampler (Algorithm 7). However, we notice that v' is in the discrete set $\frac{1}{q} \cdot \mathbb{Z}$, thus one can build q CDT tables for each coset and sample via looking up the according table. Compared to the arbitrary-centered sampler, the table-based one is faster, simpler and implemented fully over integers. In practice, we only need $(\frac{q}{2} + 1)$ CDT tables for the cosets $S_c = \frac{1}{q} \cdot \{0, 1, \cdots, \frac{q}{2}\}$, as $x \leftarrow D_{\mathbb{Z}-c, \bar{r}^2}$ is equivalent to $(-x) \leftarrow D_{\mathbb{Z}+c, \bar{r}^2}$. The detailed algorithm is given in Algorithm 11. The table used in Algorithm 11 consists of the $(\frac{q}{2} + 1)$ RCDT tables for $D_{\mathbb{Z}+c_i, \bar{r}^2} - c_i$ for $c_i \in S_c$.

Algorithm 11: SampleZ$_d$

Input: a coset $c \in \frac{1}{q} \cdot \mathbb{Z}$
Output: $z \leftarrow D_{\mathbb{Z}+c, \bar{r}^2}$
1: $c' \leftarrow c - \lfloor c \rfloor, b \leftarrow (c' \in S_c)$
2: $h \leftarrow (2b - 1)c' + (1 - b)$
3: $u \leftarrow U(\{0, 1\}^{72})$
4: $z_h \leftarrow 0$
5: **for** $i = -12, \ldots, 12$ **do**
6: $\quad z_h \leftarrow z_h + [\![u < RCDT[h][i]]\!]$
7: **end for**
8: $z_h \leftarrow z_h + h$ $\hspace{8cm} \{z_h \leftarrow D_{\mathbb{Z}+h, \bar{r}^2}\}$
9: $z \leftarrow (2b - 1)z_h$ $\hspace{8cm} \{z \leftarrow D_{\mathbb{Z}+c', \bar{r}^2}\}$
10: **return** z

3.3 Signature Verification

The verification is straightforward as in other GPV hash-and-sign signatures. A formal description is given in Algorithm 12.

Algorithm 12: Verify

Input: a message m, its signature $(\mathsf{salt}, \mathsf{str})$, the public key pk and the acceptance bound B
Output: Accept or Reject
1: $\hat{\mathbf{A}} \leftarrow \mathsf{XOF}(\mathsf{seed}_{\hat{\mathbf{A}}})$
2: **if** $|\mathsf{str}| \neq L$ **then**
3:     Reject
4: **end if**
5: $(\mathbf{x}_1, \mathbf{x}_2) \leftarrow \mathsf{Decompress}(\mathsf{str})$ with $\mathbf{x}_1 \in \mathbb{Z}^n, \mathbf{x}_2 \in \mathbb{Z}^m$
6: $\mathbf{u} \leftarrow \mathsf{H}(\mathsf{m}, \mathsf{salt})$, $\mathbf{x}'_0 \leftarrow (\mathbf{u} - \hat{\mathbf{A}}\mathbf{x}_1 - \mathbf{B}\mathbf{x}_2) \bmod Q$
7: Accept if $\|(\mathbf{x}'_0, \mathbf{x}_1, \mathbf{x}_2)\| \leq B$, otherwise Reject

3.4 Signature Encoding

The compression encoding is implemented with the range Asymmetric Numeral Systems (rANS) [Dud13]. As discovered in [ETWY22], the rANS encoding gives a nearly optimal storage for Gaussian vectors close to the entropic bound. Each element is split into the sign, the low bits, and the high bits. The sign and low bits are nearly uniform, so they are stored directly. The high bits are compressed by the rANS encoding. Details are shown in Algorithm 13 and Algorithm 14.

Algorithm 13: Compress

Input: a vector $\mathbf{x} \in \mathbb{Z}^{n+m}$
Output: a string str of length L, or $\perp$
1: **for** $i = 1, \ldots, n+m$ **do**
2:     $s_i \leftarrow (x_i < 0)$
3:     $h_i \leftarrow |x_i| \gg 7$
4:     $l_i \leftarrow |x_i| \& \mathsf{0x7f}$
5: **end for**
6: $\mathsf{str} \leftarrow \mathsf{rANSEncode}(h_1 \,\|\, \cdots \,\|\, h_{n+m}) \,\|\, s_1 \,\|\, l_1 \,\|\, \cdots \,\|\, s_{n+m} \,\|\, l_{n+m} \,\|\, 1$
7: **if** $|\mathsf{str}| > L$ **then**
8:     **return** $\perp$
9: **else**
10:     $\mathsf{str} \leftarrow \mathsf{str} \,\|\, 0^{L - |\mathsf{str}|}$
11: **end if**
12: **return** str

Algorithm 14: Decompress

Input: a string str of length L
Output: a vector $\mathbf{x} \in \mathbb{Z}^{n+m}$, or $\perp$
1: $r \,\|\, s_1 \,\|\, l_1 \,\|\, \cdots \,\|\, s_{n+m} \,\|\, l_{n+m} \,\|\, 1 \,\|\, 0^* \leftarrow \mathsf{str}$
2: $h_1 \,\|\, \cdots \,\|\, h_{n+m} \leftarrow \mathsf{rANSDecode}(r)$
3: **for** $i = 1, \ldots, n+m$ **do**
4:     **if** $s_i = 1$ **then**
5:         $x_i \leftarrow -(h_i \ll 7 \& l_i)$
6:     **else**
7:         $x_i \leftarrow h_i \ll 7 \& l_i$
8:     **end if**
9: **end for**
10: **return** $\mathbf{x} = (x_1, x_2, \ldots, x_{n+m})$

3.5 Recommended Parameters

We specify three sets of parameter for the NIST-1, NIST-3, NIST-5 security levels respectively. They are shown in Table 2. The numbers for concrete security are estimated as the core-SVP hardness of known attacks. Details are shown in Section 4.

Security level		NIST-1	NIST-3	NIST-5
Dimensions (m, n)		$(736, 848)$	$(1024, 1232)$	$(1312, 1552)$
Modulus Q		2^{16}	2^{17}	2^{17}
Gadget parameters (p, q)		$(2^{12}, 2^4)$	$(2^{13}, 2^4)$	$(2^{13}, 2^4)$
Acceptance bound B		62521	108493	130320
Signature size (in bytes)		2455	3540	4520
Public key size (in kilobytes)		1059	2177	3573
Key recovery:	BKZ blocksize	443	663	878
	Classical core-SVP security	129	194	256
	Quantum core-SVP security	117	176	233
Forgery:	BKZ blocksize	438	659	883
	Classical core-SVP security	128	192	258
	Quantum core-SVP security	116	175	234

Table 2. Recommended parameters.

4 Security

4.1 Security Reduction

4.1.1 Simulatable Signatures The HuFu signatures can be simulated without knowing the secret key in the random oracle model (ROM). The proof is sketched as follows. We follow the notations in Algorithm 5. Let $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2)$ and $\mathbf{T} = \begin{bmatrix} \mathbf{E} \\ \mathbf{S} \\ \mathbf{I} \end{bmatrix}$, then $\mathbf{x} = \mathbf{p} + \mathbf{Tz}$, $\mathbf{AT} = p \cdot \mathbf{I} \bmod Q$ and

$$\mathbf{Ax} = p\mathbf{z} + \mathbf{c} = \mathbf{v} - \mathbf{e} + \mathbf{v} = \mathbf{u} - \mathbf{e} \bmod Q.$$

In the ROM, the target $\mathbf{u}$ is uniformly random over $\mathbb{Z}_Q^m$ and then $\mathbf{e}$ is uniformly random over $\mathbb{Z}_p^m$. In HuFu, $r = q\bar{r} \geq \eta'_\epsilon(q\mathbb{Z}^m)$ and $\sigma^2 \geq (r^2 + \bar{r}^2)(s_1(\mathbf{T})^2 + 1)$. By the same arguments in [YJW23], it follows that the distribution of $(\mathbf{u}, \mathbf{x}, \mathbf{e})$ in the signing procedure is statistically close to the one where $\mathbf{x} \leftarrow D_{\mathbb{Z}^{n+2m}, \sigma^2}, \mathbf{e} \leftarrow U(\mathbb{Z}_p^m)$ and $\mathbf{u} = \mathbf{Ax} + \mathbf{e} \bmod Q$. The latter is publicly simulatable.

4.1.2 Strong Unforgeability We now prove the SUF-CMA security in the ROM. We basically follows the same arguments for the GPV signatures, and thus only give a proof sketch. Under the $\mathsf{LWE}_{n,m,Q,\chi}$ assumption, the public key $(\hat{\mathbf{A}}, \mathbf{B})$ is computationally indistinguishable from uniform. Since the signature distribution is simulatable without using the secret key, one can simulate the random oracle and the signing oracle , and interact with the forgery adversary. Once a successful forgery is made, a short solution to the $\mathsf{SIS}_{m,n,Q,2B}$ problem is constructed with high probability. The security proof in the quantum ROM can be also given by the arguments in [BDF+11, CD20].

   The above security reduction is mainly in an asymptotic sense. As is the case for FALCON, the SIS bound $2B$ is larger than the modulus Q for the recommended parameters of HuFu, which allows q-vectors as trivial solutions to $\mathsf{SIS}_{m,n,Q,2B}$. However, this is not known to affect the concrete security, as the forgery attack in effect solves $\mathsf{ISIS}_{m,n,Q,B}$ where $B < Q$. Moreover, this can be addressed by using larger parameters (leading to $\approx 10\%$ increase on the sizes), if so desired.

4.2 Concrete Security

We analyze the cost of known lattice attacks and translate the analysis into concrete bit-security following the core-SVP methodology.

4.2.1 Lattice Reduction and Core-SVP Hardness

The BKZ lattice reduction algorithm [SE94] and its optimized variants [CN11, MW16] are the best known algorithms for solving lattice problems. The BKZ algorithm can find short lattice vectors and this strength increases with the blocksize β of BKZ. For a d-dimensional lattice Λ, BKZ with blocksize β would find some short $\mathbf{v} \in \Lambda$ with

$$\|\mathbf{v}\| \leq \delta_\beta^d \operatorname{vol}(\Lambda)^{1/d} \quad \text{and} \quad \delta_\beta \approx \left(\frac{(\pi\beta)^{\frac{1}{\beta}} \beta}{2\pi e} \right)^{\frac{1}{2(\beta-1)}}$$

when $d > \beta > 50$.

The core-SVP methodology, proposed in [ADPS16], gives a common method to assess the cost of lattice attacks. Following this methodology, one first estimates the blocksize β required for successful attacks and then quantify the attack cost with the core-SVP hardness model that is conservative. Specifically, the cost of BKZ with blocksize β is estimated as $2^{0.292\beta}$[BDGL16] in the classical setting and $2^{0.265\beta}$[Laa16] in the quantum setting.

4.2.2 Key Recovery

The key recovery attack against HuFu boils down to solving the underlying LWE problem. It can be done by finding short $(\mathbf{s}, \mathbf{e}) \in \mathbb{Z}^n \times \mathbb{Z}^l$ such that $\mathbf{As} + \mathbf{e} = \mathbf{b} \bmod Q$ where $\mathbf{A} \in \mathbb{Z}_Q^{l \times n}$ and $\mathbf{b} \in \mathbb{Z}_Q^l$ are public. The primal attack is a primary method for this. As shown in [ADPS16], the primal attack succeeds when the blocksize β of the used BKZ algorithm satisfies

$$\|(\mathbf{s}, \mathbf{e}, 1)\| \sqrt{\frac{3\beta}{4(l+n+1)}} \leq \delta_\beta^{2\beta - (l+n+1) - 1} \cdot Q^{\frac{l}{l+n+1}}$$

where $\sqrt{3/4}$ is set for a conservative estimate as per [Duc18]. Note the the number of LWE samples l is between 0 and m in our case, and we choose the one minimizing the cost of the attack.

4.2.3 Signature Forgery

The forgery attack against HuFu is essentially to solve the approximate-CVP problem. A common used approach for this is the nearest-colattice algorithm [EK20]. In our case, given $(\mathbf{A}, \mathbf{u}) \in \mathbb{Z}_Q^{m \times (2m+n)} \times \mathbb{Z}_Q^m$ and $B > 0$, the nearest-colattice algorithm can find a short vector $\mathbf{x}$ such that $\mathbf{Ax} = \mathbf{u} \bmod Q$ and $\|\mathbf{x}\| \leq B$ by calling BKZ with blocksize β satisfying

$$B \geq \min_{k \leq m+n} \left(\delta_\beta^{2m+n-k} Q^{\frac{m}{2m+n-k}} \right).$$

5 Implementation and Performances

The submission includes three implementation versions written in standard C:

- Reference implementation. This is mainly used for proof-of-concept.
- Optimized implementation. This version uses some code optimizations excluding AVX2 instructions.
- AVX2 implementation. Based on the optimized implementation, the AVX2 implementation further utilizes AVX2 instructions for x86_64 platform.

In this section, we describe the technical implementation details and report the performance of our implementation.

5.1 Public Key Packing

The public key pk consists of a 32-byte string $\mathsf{seed}_{\hat{\mathbf{A}}}$ and a matrix $\mathbf{B} \in \mathbb{Z}_Q^{m \times m}$. In the NIST-1 parameter set, the elements of $\mathbf{B}$ are in 2 bytes and densely packed one after another. In other two parameter sets, the elements of $\mathbf{B}$ are in 17 bits. In this case, the low 16 bits of each element are densely packed into 2 bytes, and the rest of 1 most significant bit of all elements are collected and packed together.

5.2 Floating-Point Arithmetic

The key generation and the offline phase of the signing procedure involve the computations on real numbers. We implement these with double-precision floating-point numbers that is the IVerify standard for floating-point arithmetic (IVerify 754). This floating-point arithmetic offers 53 bits of precision. Note that the online operations in signing procedure and the verification are implemented fully over integers.

5.3 Performances

The implementation is complied by gcc 11.3.0 with command `-mavx2 -mbmi2 -mpopcnt -O3 -std=gnu11 -march=native -Wextra -DNIX -mfpmath=sse -msse2 -ffp-contract=off` and runs on Ubuntu 22.04.2. Table 3 reports the performance of our implementations on a single core of Intel Core i9-12900K @ 3.20 GHz with 32GB RAM.

	NIST-1	NIST-3	NIST-5
Optimized implementation (AES256NI)			
KeyGen	1,193,896	8,916,915	9,727,510
Sign (offline)	6,618	16,606	28,639
Sign (online)	704	1,807	3,257
Sign (total)	7,322	18,413	31,896
Verify	1,804	6,105	10,424
Optimized implementation (SHAKE256)			
KeyGen	1,158,074	9,686,255	8,848,380
Sign (offline)	10,550	24,631	42,303
Sign (online)	1,165	2,749	4,515
Sign (total)	11,715	27,380	46,818
Verify	6,133	15,658	26,152
AVX2 implementation (AES256NI)			
KeyGen	703,222	2,500,839	5,109,743
Sign (offline)	3,473	6,324	10,933
Sign (online)	399	677	1,158
Sign (total)	3,872	7,001	12,091
Verify	801	2,358	3,811
AVX2 implementation (SHAKE256)			
KeyGen	707,749	2,494,678	5,098,358
Sign (offline)	5,695	11,786	19,658
Sign (online)	647	1,236	2,191
Sign (total)	6,342	13,022	21,849
Verify	3,069	8,510	13,697

Table 3. Performance (in kilocycles) of HuFu on a single core of Intel Core i9-12900K @ 3.20 GHz. Numbers are the average measured over $1,000$ executions.

6 Advantages and Limitations

6.1 Advantages

Strong security assurance: HuFu is based on the plain SIS and LWE that have the least amount of structure among the security assumptions of lattice-based schemes. This mitigates the potential cryptanalytic risk from algebraic structures [CDW17, DPW19] and dense sublattices [KF17, DvW21].

Extended applications: Thanks to the gadget framework, HuFu has the potential to be adapted to advanced primitives, e.g. (hierarchical) identity-based encryption and aggregate signatures. Such powerful versatility is the unique advantage of HuFu, making HuFu support a wide range of applications.

Online/offline structure: Most complex and costly signing operations can be done in the offline phase. The online operations are particularly simple, fast and fully over integers. For this, Sign is of special interest in certain scenarios, e.g. an architecture in which the offline phase is performed on FPGA or CPU while the online phase on a regular device. Furthermore, such an online/offline structure may greatly ease the side-channel protection that remains a challenging problem for lattice-based hash-and-sign signatures.

Short signatures & **Fast speed:** The signature size of HuFu is on par with CRYSTALS-DILITHIUM, while HuFu is not based on structured lattices. The signing and verification of HuFu are efficient. HuFu is also highly parallelizable, which provides some room for optimizations. In addition, its online/offline structure allows to further reduce the online runtime and computation resource.

6.2 Limitations

Large public keys: The public key size of HuFu is around 1 to 3.5 megabytes for three different security levels, thus HuFu may not be so desirable for many applications. Nevertheless, HuFu is totally qualified for the usecases where keys are not transmitted frequently.

Floating-point arithmetic: While the online phase in the signing procedure is implemented fully over integers, the offline phase still heavily uses floating-point arithmetic. This may be a limitation when the online/offline mode is disabled, especially for the implementations on constraint devices. However, this can be addressed by using the integral Gram decomposition technique [DGPY20]. We leave the fully integer implementation as the follow-up works.

Bibliography

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 153–178. Springer, Heidelberg, August 2016.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996.

[Bab86] László Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016.

[BLNR22] Olivier Bernard, Andrea Lesavourey, Tuong-Huy Nguyen, and Adeline Roux-Langlois. Log-*S*-unit lattices using explicit stickelberger generators to solve approx ideal-SVP. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 677–708. Springer, Heidelberg, December 2022.

[BR20] Olivier Bernard and Adeline Roux-Langlois. Twisted-PHS: Using the product formula to solve approx-SVP in ideal lattices. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 349–380. Springer, Heidelberg, December 2020.

[CD20] André Chailloux and Thomas Debris-Alazard. Tight and optimal reductions for signatures based on average trapdoor preimage sampleable functions and applications to code-based signatures. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 453–479. Springer, Heidelberg, May 2020.

[CDW17] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-SVP. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EURO-CRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 324–348. Springer, Heidelberg, April / May 2017.

[CGM19] Yilei Chen, Nicholas Genise, and Pratyay Mukherjee. Approximate trapdoors for lattices and smaller hash-and-sign signatures. In Steven D. Galbraith and Shiho Moriai, editors, *ASI-ACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 3–32. Springer, Heidelberg, December 2019.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2011.

[DGPY20] Léo Ducas, Steven Galbraith, Thomas Prest, and Yang Yu. Integral matrix gram root and lattice gaussian sampling without floats. In Anne Canteaut and Yuval Ishai, editors, *EURO-CRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 608–637. Springer, Heidelberg, May 2020.

[dK22] Rafaël del Pino and Shuichi Katsumata. A new framework for more efficient round-optimal lattice-based (partially) blind signature via trapdoor sampling. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 306–336. Springer, Heidelberg, August 2022.

[DLP14] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 22–41. Springer, Heidelberg, December 2014.

[dLS18] Rafaël del Pino, Vadim Lyubashevsky, and Gregor Seiler. Lattice-based group signatures and zero-knowledge proofs of automorphism stability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 574–591. ACM Press, October 2018.

[DN12] Léo Ducas and Phong Q. Nguyen. Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 433–450. Springer, Heidelberg, December 2012.

[DP16] Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In *ISSAC 2016*, pages 191–198, 2016.

[DPW19] Léo Ducas, Maxime Plançon, and Benjamin Wesolowski. On the shortness of vectors to be found by the ideal-SVP quantum algorithm. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 322–351. Springer, Heidelberg, August 2019.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 125–145. Springer, Heidelberg, April / May 2018.

[Dud13]   Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

[DvW21]   Léo Ducas and Wessel P. J. van Woerden. NTRU fatigue: How stretched is overstretched? In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 3–32. Springer, Heidelberg, December 2021.

[EFG⁺22] Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 222–253. Springer, Heidelberg, May / June 2022.

[EK20]   Thomas Espitau and Paul Kirchner. The nearest-colattice algorithm: Time-approximation trade-off for approx-cvp. *ANTS XIV*, 4(1):251–266, 2020.

[ETWY22] Thomas Espitau, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Shorter hash-and-sign lattice-based signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 245–275. Springer, Heidelberg, August 2022.

[GPV08]   Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GVW13]   Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 545–554. ACM Press, June 2013.

[HHSS17] Shai Halevi, Tzipora Halevi, Victor Shoup, and Noah Stephens-Davidowitz. Implementing BP-obfuscation using graph-induced encoding. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 783–798. ACM Press, October / November 2017.

[HPRR20] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 53–71. Springer, Heidelberg, 2020.

[KF17]   Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 3–26. Springer, Heidelberg, April / May 2017.

[Laa16]   Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, PhD thesis, Eindhoven University of Technology, 2016., 2016.

[LDK⁺20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

[LW20]   Changmin Lee and Alexandre Wallet. Lattice analysis on mintru problem. Cryptology ePrint Archive, Paper 2020/230, 2020. `https://eprint.iacr.org/2020/230`.

[MP12]   Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[MW16]   Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 820–849. Springer, Heidelberg, May 2016.

[NR06]   Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 271–288. Springer, Heidelberg, May / June 2006.

[Pei10]   Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 80–97. Springer, Heidelberg, August 2010.

[PFH+20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyuba-shevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhen-fei Zhang. FALCON. Technical report, National Institute of Standards and Technol-ogy, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

[PHS19] Alice Pellet-Mary, Guillaume Hanrot, and Damien Stehlé. Approx-SVP in ideal lattices with pre-processing. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 685–716. Springer, Heidelberg, May 2019.

[PP19] Thomas Pornin and Thomas Prest. More efficient algorithms for the NTRU key generation using the field norm. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 504–533. Springer, Heidelberg, April 2019.

[Pre15] Thomas Prest. *Gaussian Sampling in Lattice-Based Cryptography*. PhD thesis, PhD thesis, École Normale Supérieure Paris, 2015., 2015.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[SE94] Claus-Peter Schnorr and Martin Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66:181–199, 1994.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 27–47. Springer, Heidelberg, May 2011.

[YD18] Yang Yu and Léo Ducas. Learning strikes again: The case of the DRS signature scheme. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 525–543. Springer, Heidelberg, December 2018.

[YJW23] Yang Yu, Huiwen Jia, and Xiaoyun Wang. Compact lattice gadget and its applications to hash-and-sign signatures. In *CRYPTO 2023*, page (to appear), 2023.