Practical Multilinear Maps over the Integers*

Jean-Sébastien Coron[1], Tancrède Lepoint[2,3], and Mehdi Tibouchi[4]

[1] University of Luxembourg
jean-sebastien.coron@uni.lu
[2] CryptoExperts, France
[3] École Normale Supérieure, France
tancrede.lepoint@cryptoexperts.com
[4] NTT Secure Platform Laboratories, Japan
tibouchi.mehdi@lab.ntt.co.jp

Abstract. Extending bilinear elliptic curve pairings to multilinear maps is a long-standing open problem. The first plausible construction of such multilinear maps has recently been described by Garg, Gentry and Halevi, based on ideal lattices. In this paper we describe a different construction that works over the integers instead of ideal lattices, similar to the DGHV fully homomorphic encryption scheme. We also describe a different technique for proving the full randomization of encodings: instead of Gaussian linear sums, we apply the classical leftover hash lemma over a quotient lattice. We show that our construction is relatively practical: for reasonable security parameters a one-round 7-party Diffie-Hellman key exchange requires less than 40 seconds per party. Moreover, in contrast with previous work, multilinear analogues of useful, base group assumptions like DLIN appear to hold in our setting.

1 Introduction

Multilinear maps. Extending bilinear elliptic curve pairings to multilinear maps is a long-standing open problem. In 2003 Boneh and Silverberg showed two interesting applications of multilinear maps [BS03], namely multipartite Diffie-Hellman and very efficient broadcast encryption; however they were pessimistic about the existence of such maps from the realm of algebraic geometry.

The first plausible construction of multilinear maps has recently been described by Garg, Gentry and Halevi, based on ideal lattices [GGH13]. The main difference with bilinear pairings is that the encoding $a \cdot g$ of an element a is randomized (with some noise) instead of deterministic; only the computed multilinear map $e(a_1 \cdot g, \ldots, a_\kappa \cdot g)$ is a deterministic function of the a_i's only. The construction has bounded degree with a maximum degree κ at most polynomial in the security parameter. Indeed, the encoding noise grows linearly with the degree, and when the noise reaches a certain threshold the encoding can become incorrect, as for ciphertexts in a somewhat homomorphic encryption scheme. The security of the construction relies on new hardness assumptions which are natural extensions of the Decisional Diffie-Hellman (DDH) assumption. To gain more confidence in their scheme the authors provide an extensive cryptanalytic survey. The authors focus on one application: the multipartite Diffie-Hellman key exchange.

The construction from [GGH13] works in the polynomial ring $R = \mathbb{Z}[X]/(X^n + 1)$, where n is large enough to ensure security. One generates a secret short ring element $\boldsymbol{g} \in R$, generating a principal ideal $\mathcal{I} = \langle \boldsymbol{g} \rangle \subset R$. One also generates an integer parameter q and another random secret $\boldsymbol{z} \in R/qR$. One encodes elements of the quotient ring $R/\mathcal{I}$, namely elements of the form $\boldsymbol{e} + \mathcal{I}$ for some $\boldsymbol{e}$, as follows: a level-i encoding of the coset $\boldsymbol{e} + \mathcal{I}$ is an element of the form $u_k = [\boldsymbol{c}/\boldsymbol{z}^i]_q$, where $\boldsymbol{c} \in \boldsymbol{e} + \mathcal{I}$ is short. Such encodings can be both added and multiplied, as long as the norm of the numerators remain shorter than q; in particular the product of κ encodings at level 1 gives an encoding at level κ. For such level-κ encodings one can then define a zero-testing parameter $\boldsymbol{p}_{zt} = [\boldsymbol{h}\boldsymbol{z}^\kappa/\boldsymbol{g}]_q$, for some small $\boldsymbol{h} \in R$. Then given a level-κ encoding $\boldsymbol{u} = [\boldsymbol{c}/\boldsymbol{z}^\kappa]$ one can compute $[\boldsymbol{p}_{zt} \cdot \boldsymbol{u}]_q = [\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}]_q$. When $\boldsymbol{c}$ is an encoding of zero we have $\boldsymbol{c}/\boldsymbol{g} \in R$, which implies that $\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}$ is small in R, and therefore $[\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}]_q$ is small; this provides a way to test whether a level-κ encoding $\boldsymbol{c}$ is an encoding of 0. For the same reason the high-order bits of $[\boldsymbol{p}_{zt} \cdot \boldsymbol{u}]_q = [\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}]_q$ only depend on the

* An extended abstract appeared at Crypto 2013; this is the full version.

coset $\boldsymbol{e}+\mathcal{I}$ and not on the particular $\boldsymbol{c} \in \boldsymbol{e}+\mathcal{I}$; this makes it possible to extract a representation of cosets encoded at level κ, and eventually defines a degree-κ multilinear map for level-1 encodings.

Our contributions. Our main contribution is to describe a different construction that works over the integers instead of ideal lattices, similar to the DGHV fully homomorphic encryption scheme [DGHV10] and its batch variant [CCK+13]. Our construction offers the same flexibility as the original from [GGH13]; in particular it can be modified to support the analogue of asymmetric maps and composite-order maps. Moreover, it does not seem vulnerable to the "zeroizing" attack that breaks base group hardness assumptions like the analogues of DLIN and subgroup membership for the multilinear maps of [GGH13]. Since those assumptions are believed necessary to adapt constructions of primitives like adaptively secure functional encryption and NIZK, our construction seems even more promising for applications than [GGH13]. In Appendix C, we provide a comparison with the original scheme, to better highlight the similarities.

As in [GGH13], the security of our construction relies on new assumptions; it cannot be derived from "classical" assumptions such as the Approximate-GCD assumption used in [DGHV10]. We describe various possible attacks against our scheme; this enables us to derive parameters for which our scheme remains secure against these attacks.

Our new construction works as follows: one first generates n secret primes p_i and publishes $x_0 = \prod_{i=1}^{n} p_i$ (where n is large enough to ensure correctness and security); one also generates n small secret primes g_i, and a random secret integer z modulo x_0. A level-k encoding of a vector $\boldsymbol{m} = (m_i) \in \mathbb{Z}^n$ is then an integer c such that for all $1 \leqslant i \leqslant n$:

$$c \equiv \frac{r_i \cdot g_i + m_i}{z^k} \pmod{p_i} \tag{1}$$

for some small random integers r_i; the integer c is therefore defined modulo x_0 by CRT. It is clear that such encodings can be both added and multiplied modulo x_0, as long as the numerators remain smaller than the p_i's. In particular the product of κ encodings c_j at level 1 gives an encoding at level κ where the corresponding vectors $\boldsymbol{m}_j$ are multiplied componentwise. For such level-κ encodings one defines a zero-testing parameter p_{zt} with:

$$p_{zt} = \sum_{i=1}^{n} h_i \cdot \left(z^\kappa \cdot g_i^{-1} \bmod p_i\right) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0$$

for some small integers h_i. Given a level-κ encoding c as in (1), one can compute $\omega = p_{zt} \cdot c \bmod x_0$, which gives:

$$\omega = \sum_{i=1}^{n} h_i \cdot \left(r_i + m_i \cdot (g_i^{-1} \bmod p_i)\right) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0 \ .$$

Then if $m_i = 0$ for all i, since the r_i's and h_i's are small, we obtain that ω is small compared to x_0; this enables to test whether c is an encoding of $\mathbf{0}$ or not. Moreover for non-zero encodings the leading bits of ω only depend on the m_i's and not on the noise r_i; for level-κ encodings this enables to extract a function of the m_i's only, which eventually defines as in [GGH13] a degree-κ multilinear map.[1]

Our second contribution is to describe a different technique for proving the full randomization of encodings. As in [GGH13] the randomization of encodings is obtained by adding a random subset-sum of encodings of $\mathbf{0}$ from the public parameters. However as in [GGH13] the Leftover Hash Lemma (LHL) cannot be directly applied since the encodings live in some infinite ring instead of a finite group. The solution in [GGH13] consists in using linear sums with Gaussian coefficients; it is shown in [AGHS12] that the resulting sum has a Gaussian distribution (over some lattice). As noted by the authors, this can be seen as a "leftover hash lemma over lattices". In this paper we

[1] Technically for p_{zt} we use a *vector* of integers instead of a single integer (see Sec. 3).

describe a different technique that does not use Gaussian coefficients; instead it consists in working modulo some lattice $L \subset \mathbb{Z}^n$ and applying the leftover hash lemma over the quotient $\mathbb{Z}^n/L$, which is still a finite group. Such technique was already used to prove the security of the batch variant of the DGHV fully homomorphic encryption scheme [CCK$^+$13,CLT13]. Here we provide a more formal description: we clearly state our "LHL over lattices" so that it can later be applied as a black-box (as the corresponding Theorem 3 in [AGHS12]). Our quotient lattice technique can independently be applied to the original encoding scheme from [GGH13], while the Gaussian sum technique from [AGHS12] is also applicable to ours.

Our third contribution is to describe the first implementation of multilinear maps. It appears that the basic versions of both [GGH13] and our scheme are rather unpractical, because of the huge public parameter size required to randomize the encodings. Therefore we use a simple optimization that consists in storing only a small subset of the public elements and combining them pairwise to generate the full public-key. Such optimization was originally described in [GH11] for reducing the size of the encryption of the secret-key bits in the implementation of Gentry's FHE scheme [Gen09]. It was also used in [CMNT11] to reduce the public-key size of the DGHV scheme; however, as opposed to the latter work our randomization of encodings is heuristic only, whereas in [CMNT11] the semantic security was still guaranteed. Thanks to this optimization our construction becomes relatively practical: for reasonable security parameters a multipartite Diffie-Hellman computation with 7 users requires less than 40 seconds, with a public parameter size of roughly 2.6 GBytes; a proof-of-concept implementation is available at [Lep].

2 Definition of Randomized Encodings and Multilinear Maps

In this section we recall the setting introduced in [GGH13] for the notion of randomized encodings and multilinear maps, which they call *graded encoding schemes*. There are essentially two main differences with classical bilinear pairings (and their generalization to cryptographic multilinear maps as considered in [BS03]):

1. In bilinear pairings (and more generally cryptographic multilinear maps) we have a map $e\colon G^\kappa \to G_T$ that is linear with respect to all its κ inputs:
$$e(a_1 \cdot g, \ldots, a_\kappa \cdot g) = \left(\prod_{i=1}^{\kappa} a_i\right) \cdot e(g, \ldots, g)\ . \tag{2}$$
One can view $a \cdot g$ as an "encoding" of the integer $a \in \mathbb{Z}_p$ over the group G of order p generated by g. The main difference in our setting is that encodings are now randomized. This means that an element $a \in R$ (where R is a ring that plays the role of the exponent space $\mathbb{Z}_p$) has many possible encodings; only the final multilinear map $e(a_1 \cdot g, \ldots, a_\kappa \cdot g)$ is a deterministic function of the a_i's only, and not on the randomness used to encode a_i into $a_i \cdot g$.
2. The second main difference is that to every encoding is now associated a *level*. At level 0 we have the "plaintext" ring elements $a \in R$, at level 1 we have the encoding $a \cdot g$, and by combining k such encodings $a_i \cdot g$ at level 1 one obtains a level-k encoding where the underlying elements a_i are homomorphically multiplied in R. The difference with "classical" cryptographic multilinear maps is that we can now multiply any (bounded) subset of encodings, instead of strictly κ at a time as with (2). For encodings at level κ we have a special zero-testing parameter $\boldsymbol{p}_{zt}$ that can extract a deterministic function of the underlying ring elements. This enables to define a degree-κ multilinear map for encodings at level 1.

2.1 Graded Encoding System

We recall the formal definition of a κ-Graded Encoding System from [GGH13]. For simplicity we only consider the symmetric case throughout the paper; we refer to [GGH13] for a more general framework that can handle the asymmetric case.

Definition 1 (κ-Graded Encoding System [GGH13]). *A κ-Graded Encoding System for a ring R is a system of sets $\mathcal{S} = \{S_v^{(\alpha)} \in \{0,1\}^* : v \in \mathbb{N}, \alpha \in R\}$, with the following properties:*

1. *For every $v \in \mathbb{N}$, the sets $\{S_v^{(\alpha)} : \alpha \in R\}$ are disjoint.*
2. *There are binary operations $+$ and $-$ (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every $v \in \mathbb{N}$, and every $u_1 \in S_v^{(a_1)}$ and $u_2 \in S_v^{(a_2)}$, it holds that $u_1 + u_2 \in S_v^{(\alpha_1+\alpha_2)}$ and $u_1 - u_2 \in S_v^{(\alpha_1-\alpha_2)}$ where $\alpha_1 + \alpha_2$ and $\alpha_1 - \alpha_2$ are addition and subtraction in R.*
3. *There is an associative binary operation $\times$ (on $\{0,1\}^*$) such that for every $\alpha_1, \alpha_2 \in R$, every v_1, v_2 with $0 \leqslant v_1 + v_2 \leqslant \kappa$, and every $u_1 \in S_{v_1}^{(\alpha_1)}$ and $u_2 \in S_{v_2}^{(\alpha_2)}$, it holds that $u_1 \times u_2 \in S_{v_1+v_2}^{(\alpha_1 \cdot \alpha_2)}$ where $\alpha_1 \cdot \alpha_2$ is multiplication in R.*

2.2 Multilinear map Procedures

We also recall the definition of the procedures for manipulating encodings. As previously we consider only the symmetric case; we refer to [GGH13] for the general case.

Instance Generation. The randomized $\mathsf{InstGen}(1^\lambda, 1^\kappa)$ takes as inputs the parameters λ and κ, and outputs $(\mathsf{params}, \boldsymbol{p}_{zt})$, where params is a description of a κ-Graded Encoding System as above, and $\boldsymbol{p}_{zt}$ is a zero-test parameter.

Ring Sampler. The randomized $\mathsf{samp}(\mathsf{params})$ outputs a "level-zero encoding" $a \in S_0^{(\alpha)}$ for a nearly uniform element $\alpha \in_R R$. Note that the encoding a does not need to be uniform in $S_0^{(\alpha)}$.

Encoding. The (possibly randomized) $\mathsf{enc}(\mathsf{params}, a)$ takes as input a level-zero encoding $a \in S_0^{(\alpha)}$ for some $\alpha \in R$, and outputs the level-one encoding $u \in S_1^{(\alpha)}$ for the same α.

Re-Randomization. The randomized $\mathsf{reRand}(\mathsf{params}, i, u)$ re-randomizes encodings relative to the same level i. Specifically, given an encoding $u \in S_v^{(\alpha)}$, it outputs another encoding $u' \in S_v^{(\alpha)}$. Moreover for any two $u_1,\ u_2 \in S_v^{(\alpha)}$, the output distributions of $\mathsf{reRand}(\mathsf{params}, i, u_1)$ and $\mathsf{reRand}(\mathsf{params}, i, u_2)$ are nearly the same.

Addition and negation. Given params and two encodings relative to the same level, $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_i^{(\alpha_2)}$, we have $\mathsf{add}(\mathsf{params}, u_1, u_2) \in S_i^{(\alpha_1+\alpha_2)}$ and $\mathsf{neg}(\mathsf{params}, u_1) \in S_i^{(-\alpha_1)}$. Below we write $u_1 + u_2$ and $-u_1$ as a shorthand for applying these procedures.

Multiplication. For $u_1 \in S_i^{(\alpha_1)}$ and $u_2 \in S_j^{(\alpha_2)}$, we have $\mathsf{mul}(\mathsf{params}, u_1, u_2) = u_1 \times u_2 \in S_{i+j}^{(\alpha_1 \cdot \alpha_2)}$.

Zero-test. The procedure $\mathsf{isZero}(\mathsf{params}, \boldsymbol{p}_{zt}, u)$ outputs 1 if $u \in S_\kappa^{(0)}$ and 0 otherwise.

Extraction. The procedure extracts a random function of ring elements from their level-κ encoding. Namely $\mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, u)$ outputs $s \in \{0,1\}^\lambda$, such that:

1. For any $\alpha \in R$ and $u_1,\ u_2 \in S_\kappa^{(\alpha)}$, $\mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, u_1) = \mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, u_2)$.
2. The distribution $\{\mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, u) : \alpha \in_R R, u \in S_\kappa^{(\alpha)}\}$ is nearly uniform over $\{0,1\}^\lambda$.

This concludes the definition of the procedures. In [GGH13] the authors consider a slightly relaxed definition of isZero and ext, where isZero can still output 1 even for some non-zero encoding u with negligible probability, and ext can extract different outputs when applied to encodings of the same elements, also with negligible probability; see [GGH13] for the corresponding definitions.

2.3 Hardness Assumptions

Finally we recall the new hardness assumptions for multilinear maps introduced in [GGH13]; as previously we consider only the symmetric case and refer to [GGH13] for the general case. In this symmetric case given a set of $\kappa + 1$ level-one encodings of random elements, it should be unfeasible to distinguish a level-κ encoding of their product from random.

Graded Decisional Diffie-Hellman (GDDH) Let $\mathcal{GE}$ be a graded encoding scheme consisting of all the routines above. For an adversary $\mathcal{A}$ and parameters λ, κ we consider the following process:

1. $(\mathsf{params}, \boldsymbol{p}_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^\kappa)$
2. Choose $a_j \leftarrow \mathsf{samp}(\mathsf{params})$ for all $1 \leqslant j \leqslant \kappa + 1$
3. Set $u_j \leftarrow \mathsf{reRand}(\mathsf{params}, 1, \mathsf{enc}(\mathsf{params}, 1, a_j))$ for all $1 \leqslant j \leqslant \kappa + 1$ // encodings at level 1
4. Choose $b \leftarrow \mathsf{samp}(\mathsf{params})$ // encoding at level 0
5. Set $\tilde{u} = \mathsf{reRand}(\mathsf{params}, \kappa, \mathsf{enc}(\mathsf{params}, \kappa, \prod_{i=1}^{\kappa+1} a_i))$ // encoding of the right product at level κ
6. Set $\hat{u} = \mathsf{reRand}(\mathsf{params}, \kappa, \mathsf{enc}(\mathsf{params}, \kappa, b))$ // encoding of a random value at level κ

The GDDH distinguisher is given as input the $\kappa + 1$ level-one encodings u_j and either $\tilde{u}$ (encoding the right product) or $\hat{u}$ (encoding a random value), and must decide which is the case. The GDDH assumption states that the advantage of any efficient adversary is negligible in the security parameter.

Remark 1. For the GDDH problem to be hard, it must hold that, for random and uniformly generated $a_i \in R$, $\prod_{i=1}^{\kappa+1} a_i \neq 0$ with overwhelming probability (otherwise one could distinguish by the procedure isZero).

3 Our new Encoding Scheme

System parameters. The main parameters are the security parameter λ and the required multilinearity level $\kappa \leqslant \mathsf{poly}(\lambda)$. Based on λ and κ, we choose the vector dimension n, the bit-size η of the primes p_i, the bit-size α of the primes g_i, the maximum bit-size ρ of the randomness used in encodings, and various other parameters that will be specified later; the constraints that these parameters must satisfy are described in the next section. For integers z, p we denote the reduction of z modulo p by $(z \bmod p)$ or $[z]_p$ with $-p/2 < [z]_p \leqslant p/2$.

In our scheme a level-k encoding of a vector $\boldsymbol{m} = (m_i) \in \mathbb{Z}^n$ is an integer c such that for all $1 \leqslant i \leqslant n$:

$$c \equiv \frac{r_i \cdot g_i + m_i}{z^k} \pmod{p_i} \tag{3}$$

where the r_i's are ρ-bit random integers (specific to the encoding c), with the following secret parameters: the p_i's are η-bit prime integers, the g_i's are α-bit primes, and the denominator z is a random (invertible) integer modulo $x_0 = \prod_{i=1}^n p_i$. The integer c is therefore defined by CRT modulo x_0, where x_0 is made public. Since the p_i's must remain secret, the user cannot encode the vectors $\boldsymbol{m} \in \mathbb{Z}^n$ by CRT directly from (3); instead one includes in the public parameters a set of ℓ level-0 encodings x'_j of random vectors $\boldsymbol{a}_j \in \mathbb{Z}^n$, and the user can generate a random level-0 encoding by computing a random subset-sum of those x'_j's.

Remark 2. From (3) each integer m_i is actually defined modulo g_i. Therefore our scheme encodes vectors $\boldsymbol{m}$ from the ring $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$.

Instance generation: $(\mathsf{params}, \boldsymbol{p}_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^\kappa)$. We generate n secret random η-bit primes p_i and publish $x_0 = \prod_{i=1}^n p_i$. We generate a random invertible integer z modulo x_0. We generate n random α-bit prime integers g_i and a secret matrix $\boldsymbol{A} = (a_{ij}) \in \mathbb{Z}^{n \times \ell}$, where each component a_{ij} is randomly generated in $[0, g_i) \cap \mathbb{Z}$. We generate an integer y, three sets of integers $\{x_j\}_{j=1}^{\tau}$, $\{x'_j\}_{j=1}^{\ell}$ and $\{\Pi_j\}_{j=1}^n$, a zero-testing vector $\boldsymbol{p}_{zt}$, and a seed s for a strong randomness extractor, as described later. We publish the parameters $\mathsf{params} = \left(n, \eta, \alpha, \rho, \beta, \tau, \ell, y, \{x_j\}_{j=1}^{\tau}, \{x'_j\}_{j=1}^{\ell}, \{\Pi_j\}_{j=1}^n, s\right)$ and $\boldsymbol{p}_{zt}$.

Sampling level-zero encodings: $c \leftarrow \mathsf{samp}(\mathsf{params})$. We publish as part as our instance generation a set of ℓ integers x'_j, where each x'_j encodes at level-0 the column vector $\boldsymbol{a}_j \in \mathbb{Z}^n$ of the secret matrix $\boldsymbol{A} = (a_{ij}) \in \mathbb{Z}^{n \times \ell}$. More precisely, using the CRT modulo x_0 we generate integers x'_j such that:

$$1 \leqslant j \leqslant \ell, \qquad x'_j \equiv r'_{ij} \cdot g_i + a_{ij} \pmod{p_i} \tag{4}$$

where the r'_{ij}'s are randomly generated in $(-2^\rho, 2^\rho) \cap \mathbb{Z}$.

Our randomized sampling algorithm $\mathsf{samp}(\mathsf{params})$ works as follows: we generate a random binary vector $\boldsymbol{b} = (b_j) \in \{0,1\}^\ell$ and output the level-0 encoding

$$c = \sum_{j=1}^{\ell} b_j \cdot x'_j \bmod x_0 \ .$$

From Equation (4), this gives $c \equiv \left(\sum_{j=1}^{\ell} r'_{ij} b_j\right) \cdot g_i + \sum_{j=1}^{\ell} a_{ij} b_j \pmod{p_i}$; as required the output c is a level-0 encoding:

$$c \equiv r_i \cdot g_i + m_i \pmod{p_i} \tag{5}$$

of some vector $\boldsymbol{m} = \boldsymbol{A} \cdot \boldsymbol{b} \in \mathbb{Z}^n$ which is a random subset-sum of the column vectors $\boldsymbol{a}_j$. We note that for such level-0 encodings we get $|r_i \cdot g_i + m_i| \leqslant \ell \cdot 2^{\rho+\alpha}$ for all i.

The following Lemma states that, as required, the distribution of $\boldsymbol{m}$ can be made statistically close to uniform over $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$; the proof is based on applying the LHL over the set R (see Appendix A).

Lemma 1. *Let* $c \leftarrow \mathsf{samp}(\mathsf{params})$ *and write* $c \equiv r_i \cdot g_i + m_i \pmod{p_i}$. *Assume* $\ell \geqslant n \cdot \alpha + 2\lambda$. *The distribution of* $(\mathsf{params}, \boldsymbol{m})$ *is statistically close to the distribution of* $(\mathsf{params}, \boldsymbol{m}')$ *where* $\boldsymbol{m}' \leftarrow R$.

Encodings at higher levels: $c_k \leftarrow \mathsf{enc}(\mathsf{params}, k, c)$. To allow encoding at higher levels, we publish as part of our instance-generation a level-one random encoding of $\mathbf{1}$, namely an integer y such that:

$$y \equiv \frac{r_i \cdot g_i + 1}{z} \pmod{p_i}$$

for random integers $r_i \in (-2^\rho, 2^\rho) \cap \mathbb{Z}$; as previously the integer y is computed by CRT modulo x_0.

Given a level-0 encoding c of $\boldsymbol{m} \in \mathbb{Z}^n$ as given by (5), we can then compute a level-1 encoding of the same $\boldsymbol{m}$ by computing $c_1 = c \cdot y \bmod x_0$. Namely we obtain as required:

$$c_1 \equiv \frac{r_i^{(1)} \cdot g_i + m_i}{z} \pmod{p_i} \tag{6}$$

for some integers $r_i^{(1)}$, and we get $|r_i^{(1)} \cdot g_i + m_i| \leqslant \ell \cdot 2^{2(\rho+\alpha)}$ for all i. More generally to generate a level-k encoding we compute $c_k = c_0 \cdot y^k \bmod x_0$.

In multipartite Diffie-Hellman key-exchange every user keeps a private level-0 encoding c and publishes a level-1 encoding of the same underlying (unknown) $\boldsymbol{m}$; however one cannot publish $c_1 = c \cdot y \bmod x_0$ directly since the private level-0 encoding c could be recovered immediately from $c = c_1 / y \bmod x_0$. Instead the level-1 encoding c_1 must first be re-randomized into a new level-1 encoding c'_1 whose distribution does not depend on the original c as long as it encodes the same $\boldsymbol{m}$.

Re-randomization: $c' \leftarrow \mathsf{reRand}(\mathsf{params}, k, c)$. To allow re-randomization of encodings at level $k = 1$,[2] we publish as part of our instance-generation a set of n integers Π_j which are all level-1 random encodings of zero:

$$1 \leqslant j \leqslant n, \qquad \Pi_j \equiv \frac{\varpi_{ij} \cdot g_i}{z} \pmod{p_i} \ .$$

[2] One can easily adapt this procedure to randomize at level $k \geqslant 1$ by publishing additional similarly-defined integers.

The matrix $\boldsymbol{\varPi} = (\varpi_{ij}) \in \mathbb{Z}^{n\times n}$ is a diagonally dominant matrix generated as follows: the non-diagonal entries are randomly and independently generated in $(-2^\rho, 2^\rho) \cap \mathbb{Z}$, while the diagonal entries are randomly generated in $(n2^\rho, n2^\rho + 2^\rho) \cap \mathbb{Z}$.

We also publish as part of our instance-generation a set of τ integers x_j, where each x_j is a level-1 random encoding of zero:

$$1 \leqslant j \leqslant \tau, \qquad x_j \equiv \frac{r_{ij} \cdot g_i}{z} \pmod{p_i}$$

and where the column vectors of the matrix $(r_{ij}) \in \mathbb{Z}^{n\times\tau}$ are randomly and independently generated in the half-open parallelepiped spanned by the columns of the previous matrix $\boldsymbol{\varPi}$; see Appendix E for a concrete algorithm.

Given as input a level-1 encoding c_1 as given by (6), we randomize c_1 with a random subset-sum of the x_j's and a linear combination of the $\varPi_j$'s:

$$c_1' = c_1 + \sum_{j=1}^{\tau} b_j \cdot x_j + \sum_{j=1}^{n} b_j' \cdot \varPi_j \bmod x_0 \tag{7}$$

where $b_j \leftarrow \{0,1\}$, and $b_j' \leftarrow [0, 2^\mu) \cap \mathbb{Z}$. The following Lemma shows that as required the distribution of c_1' is nearly independent of the input (as long as it encodes the same $\boldsymbol{m}$). This follows essentially from our "leftover hash lemma over lattices"; see Section 4.

Lemma 2. *Let $c \leftarrow \mathsf{samp}(\mathsf{params})$, $c_1 \leftarrow \mathsf{enc}(\mathsf{params}, 1, c)$, and $c_1' \leftarrow \mathsf{reRand}(\mathsf{params}, 1, c_1)$. Write $c_1' \equiv (r_i \cdot g_i + m_i)/z \pmod{p_i}$ for all $1 \leqslant i \leqslant n$, and $\boldsymbol{r} = (r_1, \ldots, r_n)^T$. Let $\tau \geqslant n\cdot(\rho + \log_2(2n)) + 2\lambda$ and $\mu \geqslant \rho + \alpha + \lambda$. The distribution of $(\mathsf{params}, \boldsymbol{r})$ is statistically close to that of $(\mathsf{params}, \boldsymbol{r'})$, where $\boldsymbol{r'} \in \mathbb{Z}^n$ is randomly generated in the half-open parallelepiped spanned by the column vectors of $2^\mu \boldsymbol{\varPi}$.*

Writing $c_1' \equiv (r_i' \cdot g_i + m_i)/z \pmod{p_i}$, and using $|r_{ij} \cdot g_i| \leqslant 2n2^{\rho+\alpha}$ for all i, j, we obtain $|r_i' \cdot g_i + m_i| \leqslant \ell 2^{2(\rho+\alpha)} + \tau \cdot 2n2^{\rho+\alpha} + n \cdot 2n2^{\mu+\rho+\alpha}$. Using $\mu \geqslant \rho + \alpha + \lambda$ this gives $|r_i' \cdot g_i + m_i| \leqslant 4n^2 2^{\mu+\rho+\alpha}$.

Adding and Multiplying Encodings. It is clear that one can homomorphically add encodings. Moreover the product of κ level-1 encodings u_i can be interpreted as an encoding of the product. Namely, given level-one encodings u_j of vectors $\boldsymbol{m}_j \in \mathbb{Z}^n$ for $1 \leqslant j \leqslant \kappa$, with $u_j \equiv (r_{ij} \cdot g_i + m_{ij})/z \pmod{p_i}$, we simply let:

$$u = \prod_{j=1}^{\kappa} u_j \bmod x_0 \ .$$

This gives:

$$u \equiv \frac{\prod\limits_{j=1}^{\kappa} (r_{ij} \cdot g_i + m_{ij})}{z^\kappa} \equiv \frac{r_i \cdot g_i + \left(\prod\limits_{j=1}^{\kappa} m_{ij}\right) \bmod g_i}{z^\kappa} \pmod{p_i}$$

for some $r_i \in \mathbb{Z}$. This is a level-κ encoding of the vector $\boldsymbol{m}$ obtained by componentwise product of the vectors $\boldsymbol{m}_j$, as long as $\prod_{j=1}^{\kappa}(r_{ij} \cdot g_i + m_{ij}) < p_i$ for all i. When computing the product of κ level-1 encodings from reRand and one level-0 encoding from samp (as in multipartite Diffie-Hellman key exchange), we obtain from previous bounds $|r_i| \leqslant (4n^2 2^{\mu+\rho+\alpha})^\kappa \cdot \ell \cdot 2^{\rho+1}$ for all i.

Zero Testing. $\mathsf{isZero}(\mathsf{params}, \boldsymbol{p}_{zt}, u_\kappa) \overset{?}{=} 0/1$. We can test equality between encodings by subtracting them and testing for zero. To enable zero testing we randomly generate an integer matrix $\boldsymbol{H} = (h_{ij}) \in \mathbb{Z}^{n\times n}$ such that $\boldsymbol{H}$ is invertible in $\mathbb{Z}$ and both $\|\boldsymbol{H}^T\|_\infty \leqslant 2^\beta$ and $\|(\boldsymbol{H}^{-1})^T\|_\infty \leqslant 2^\beta$, for some parameter β specified later; here $\|\cdot\|_\infty$ is the operator norm on $n \times n$ matrices with respect to the

ℓ^∞ norm on $\mathbb{R}^n$. A technique for generating such $\boldsymbol{H}$ is discussed in Appendix F. We then publish as part of our instance generation the following zero-testing vector $\boldsymbol{p}_{zt} \in \mathbb{Z}^n$:

$$(\boldsymbol{p}_{zt})_j = \sum_{i=1}^{n} h_{ij} \cdot \left(z^\kappa \cdot g_i^{-1} \bmod p_i\right) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0 \; . \tag{8}$$

To determine whether a level-κ encoding c is an encoding of zero or not, we compute the vector $\boldsymbol{\omega} = c \cdot \boldsymbol{p}_{zt} \bmod x_0$ and test whether $\|\boldsymbol{\omega}\|_\infty$ is small:

$$\mathsf{isZero}(\mathsf{params}, \boldsymbol{p}_{zt}, c) = \begin{cases} 1 & \text{if } \|c \cdot \boldsymbol{p}_{zt} \bmod x_0\|_\infty < x_0 \cdot 2^{-\nu} \\ 0 & \text{otherwise} \end{cases}$$

for some parameter ν specified later.

Namely for a level-κ ciphertext c we have $c \equiv (r_i \cdot g_i + m_i)/z^\kappa \pmod{p_i}$ for some $r_i \in \mathbb{Z}$; therefore for all $1 \leqslant i \leqslant n$ we can write:

$$c = q_i \cdot p_i + (r_i \cdot g_i + m_i) \cdot \left(z^{-\kappa} \bmod p_i\right) \tag{9}$$

for some $q_i \in \mathbb{Z}$. Therefore combining (8) and (9), we get:

$$(\boldsymbol{\omega})_j = (c \cdot \boldsymbol{p}_{zt} \bmod x_0)_j = \sum_{i=1}^{n} h_{ij} \cdot \left(r_i + m_i \cdot (g_i^{-1} \bmod p_i)\right) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0 \; . \tag{10}$$

Therefore if $m_i = 0$ for all $1 \leqslant i \leqslant n$, then $\|\boldsymbol{\omega}\|_\infty = \|c \cdot \boldsymbol{p}_{zt} \bmod x_0\|_\infty$ is small compared to x_0 when the r_i's are small enough, *i.e.* a limited number of additions/multiplications on encodings has been performed. Conversely if $m_i \neq 0$ for some i we show that $\|\boldsymbol{\omega}\|_\infty$ must be large. More precisely we prove the following lemma in Appendix B.

Lemma 3. *Let n, η, α and β be as in our parameter setting. Let ρ_f be such that $\rho_f + \lambda + \alpha + 2\beta \leqslant \eta - 8$, and let $\nu = \eta - \beta - \rho_f - \lambda - 3 \geqslant \alpha + \beta + 5$. Let c be such that $c \equiv (r_i \cdot g_i + m_i)/z^\kappa \pmod{p_i}$ for all $1 \leqslant i \leqslant n$, where $0 \leqslant m_i < g_i$ for all i. Let $\mathbf{r} = (r_i)_{1 \leqslant i \leqslant n}$ and assume that $\|\mathbf{r}\|_\infty < 2^{\rho_f}$. If $\boldsymbol{m} = 0$ then $\|\boldsymbol{\omega}\|_\infty < x_0 \cdot 2^{-\nu-\lambda-2}$. Conversely if $\boldsymbol{m} \neq 0$ then $\|\boldsymbol{\omega}\|_\infty \geqslant x_0 \cdot 2^{-\nu+2}$.*

Extraction. $sk \leftarrow \mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, u_\kappa)$. This part is essentially the same as in [GGH13]. To extract a random function of the vector $\boldsymbol{m}$ encoded in a level-κ encoding c, we multiply c by the zero-testing parameter $\boldsymbol{p}_{zt}$ modulo x_0, collect the ν most significant bits of each of the n components of the resulting vector, and apply a strong randomness extractor (using the seed s from params):

$$\mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, c) = \mathsf{Extract}_s\left(\mathsf{msbs}_\nu(c \cdot \boldsymbol{p}_{zt} \bmod x_0)\right)$$

where msbs_ν extracts the ν most significant bits of the result. Namely if two encodings c and c' encode the same $\boldsymbol{m} \in \mathbb{Z}^n$ then from Lemma 3 we have $\|(c - c') \cdot \boldsymbol{p}_{zt} \bmod x_0\|_\infty < x_0 \cdot 2^{-\nu-\lambda-2}$, and therefore we expect that $\boldsymbol{\omega} = c \cdot \boldsymbol{p}_{zt} \bmod x_0$ and $\boldsymbol{\omega'} = c' \cdot \boldsymbol{p}_{zt} \bmod x_0$ agree on their ν most significant bits, and therefore extract to the same value.[3]

Conversely if c and c' encode different vectors then by Lemma 3 we must have $\|(c - c') \cdot \boldsymbol{p}_{zt} \bmod x_0\|_\infty > x_0 \cdot 2^{-\nu+2}$, and therefore the ν most significant bits of the corresponding $\boldsymbol{\omega}$ and $\boldsymbol{\omega'}$ must be different. This implies that for random $\boldsymbol{m} \in R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ the min-entropy of $\mathsf{msbs}_\nu(c \cdot \boldsymbol{p}_{zt} \bmod x_0)$ when c encodes $\boldsymbol{m}$ is at least $\log_2 |R| \geqslant n(\alpha - 1)$. Therefore we can use a strong randomness extractor to extract a nearly uniform bit-string of length $\lfloor \log_2 |R| \rfloor - \lambda$.

This concludes the description of our multilinear encoding scheme. In Appendix C we provide a comparison with the original scheme from [GGH13].

[3] Two coefficients ω and ω' from $\boldsymbol{\omega}$ and $\boldsymbol{\omega'}$ could still be on the opposite sides of a boundary, with $\lfloor \omega/2^k \rfloor = v$ and $\lfloor \omega'/2^k \rfloor = v + 1$, so that ω and ω' would extract to different MSBs v and $v + 1$. Heuristically this happens with probability $\mathcal{O}(2^{-\lambda})$. The argument can be made rigorous by generating a public random integer W modulo x_0 in the parameters, and extracting the MSBs of $\omega + W \bmod x_0$ instead of $\omega \bmod x_0$ for all coefficients ω of the vector $\boldsymbol{\omega}$.

3.1 Setting the Parameters

The system parameters must satisfy the following constraints:

- The bit-size ρ of the randomness used for encodings must satisfy $\rho = \Omega(\lambda)$ to avoid brute force attack on the noise, including the improved attack from [CN12] with complexity $\tilde{\mathcal{O}}(2^{\rho/2})$.
- The bit-size α of the primes g_i must be large enough so that the order of the group $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ does not contain small prime factors; this is required to prove the security of the multipartite Diffie-Hellman Key Exchange (see Appendix D). One can take $\alpha = \lambda$.
- The parameter n must be large enough to thwart lattice-based attacks on the encodings, namely $n = \omega(\eta \log \lambda)$; see Section 5.1.
- The number ℓ of level-0 encodings x'_j for samp must satisfy $\ell \geqslant n \cdot \alpha + 2\lambda$ in order to apply the leftover hash lemma; see Lemma 1.
- The number τ of level-1 encodings x_j must satisfy $\tau \geqslant n \cdot (\rho + \log_2(2n)) + 2\lambda$ in order to apply our "leftover hash lemma over lattices". For the same reason the bit-size μ of the linear sum coefficients must satisfy $\mu \geqslant \alpha + \rho + \lambda$; see Lemma 2.
- The bitsize β of the matrix $\boldsymbol{H}$ entries must satisfy $\beta = \Omega(\lambda)$ in order to avoid the GCD attack from Section 5.2. One can take $\beta = \lambda$.
- The bit-size η of the primes p_i must satisfy $\eta \geqslant \rho_f + \alpha + 2\beta + \lambda + 8$, where ρ_f is the maximum bit size of the randoms r_i a level-κ encoding (see Lemma 3). When computing the product of κ level-1 encodings and an additional level-0 encoding (as in a multipartite Diffie-Hellman key exchange with $N = \kappa + 1$ users), one obtains $\rho_f = \kappa \cdot (\mu + \rho + \alpha + 2\log_2 n + 2) + \rho + \log_2 \ell + 1$ (see previous Section).
- The number ν of most significant bits to extract can then be set to $\nu = \eta - \beta - \rho_f - \lambda - 3$ (see Lemma 3).

3.2 Security of our Construction

As in [GGH13] the security of our construction relies on new assumptions that do not seem to be reducible to more classical assumptions. Namely, as in [GGH13] one can make the assumption that solving the Graded DDH problem (GDDH) recalled in Section 2.3 is hard in our scheme. This enables to prove the security of the one-round N-way Diffie-Hellman key exchange protocol [GGH13]. Ideally one would like to reduce such assumption to a more classical assumption, such as the Approximate-GCD assumption, but that does not seem feasible. Therefore to gain more confidence in our scheme we describe various attacks in Section 5.

4 Another Leftover Hash Lemma over Lattices

As mentioned in the introduction, to prove the full randomization of encodings (Lemma 2) one cannot apply the classical Leftover Hash Lemma (LHL) because the noise in the encodings lives in some infinite ring instead of a finite group. In [GGH13] the issue was solved by using linear sums with Gaussian coefficients. Namely the analysis in [AGHS12] shows that the resulting sum has a Gaussian distribution (over some lattice). As noted by the authors this technique can be seen as a "leftover hash lemma over lattices". Such a technique would be applicable to our scheme as well.

In this section we describe an alternative technique (without Gaussian coefficients) that can also be seen as a "leftover hash lemma over lattices". It consists in working modulo a lattice $L \subset \mathbb{Z}^n$ and applying the classical leftover hash lemma over the finite group $\mathbb{Z}^n/L$. This technique was already used in [CCK+13,CLT13] to prove the security of a batch extension of the DGHV scheme. In this paper we provide a more formal description; namely we clearly state our "LHL over lattices" so that it can later be applied as a black-box (as the corresponding Theorem 3 in [AGHS12]). Namely our quotient lattice technique can independently be applied to the original encoding scheme from [GGH13].

4.1 Classical Leftover Hash Lemma

We first recall the classical Leftover Hash Lemma. We say that the distributions D_1, D_2 over a finite domain X are ε-statistically close if the statistical distance $\Delta(D_1, D_2) = \frac{1}{2}\sum_{x\in X}|D_1(x) - D_2(x)|$ is smaller than ε. A distribution D is ε-uniform if its statistical distance from the uniform distribution is at most ε. A family $\mathcal{H}$ of hash functions from X to Y, both finite sets, is said to be pairwise-independent if for all distinct $x, x' \in X$, $\Pr_{h\leftarrow H}[h(x) = h(x')] = 1/|Y|$.

Lemma 4 (Leftover Hash Lemma [HILL99]). *Let $\mathcal{H}$ be a family of pairwise hash functions from X to Y. Suppose that $h \leftarrow \mathcal{H}$ and $x \leftarrow X$ are chosen uniformly and independently. Then, $(h, h(x))$ is $\frac{1}{2}\sqrt{|Y|/|X|}$-uniform over $\mathcal{H} \times Y$.*

One can then deduce the following Lemma for random subset sums over a finite abelian group.

Lemma 5. *Let G be a finite abelian group. Set $x_1, \ldots, x_m \leftarrow G$ uniformly and independently, set $s_1, \ldots, s_m \leftarrow \{0, 1\}$, and set $y = \sum_{i=1}^{m} s_i x_i \in G$. Then $(x_1, \ldots, x_m, y)$ is $1/2\sqrt{|G|/2^m}$-uniform over G^{m+1}.*

Proof. We consider the following hash function family $\mathcal{H}$ from $\{0, 1\}^m$ to G. Each member $h \in \mathcal{H}$ is parameterized by the elements $(x_1, \ldots, x_m) \in G^m$. Given $\boldsymbol{s} \in \{0, 1\}^m$, we define $h(\boldsymbol{s}) = \sum_{i=1}^{m} s_i \cdot x_i \in G$. The hash function family is clearly pairwise independent. Therefore by Lemma 4, $(h, h(x))$ is $\frac{1}{2}\sqrt{|G|/2^m}$-uniform over G^{m+1}. □

4.2 Leftover Hash Lemma over Lattices

Let $L \subset \mathbb{Z}^n$ be a lattice of rank n of basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$. Then every $\boldsymbol{x} \subset \mathbb{Z}^n$ can be uniquely written as:

$$\boldsymbol{x} = \xi_1 \boldsymbol{b}_1 + \ldots + \xi_n \boldsymbol{b}_n$$

for some real numbers ξ_i. Moreover, for every vector $\boldsymbol{x} \in \mathbb{Z}^n$ there is a unique $\boldsymbol{a} \in L$ such that:

$$\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{a} = \xi'_1 \boldsymbol{b}_1 + \ldots + \xi'_n \boldsymbol{b}_n$$

where $0 \leqslant \xi'_i < 1$; we write $\boldsymbol{y} = \boldsymbol{x} \bmod \boldsymbol{B}$. Therefore each vector of $\mathbb{Z}^n/L$ has a unique representative in the half-open parallelepiped defined by the previous equation.

We denote by $\mathcal{D}_{\boldsymbol{B}}$ the distribution obtained by generating a random element in the quotient $\mathbb{Z}^n/L$ and taking its unique representative in the half-open parallelepiped generated by the basis $\boldsymbol{B}$. Given a basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ and $\mu \in \mathbb{Z}^*$ we denote by $\mu\boldsymbol{B}$ the basis $(\mu\boldsymbol{b}_1, \ldots, \mu\boldsymbol{b}_n)$. We are now ready to state our "Leftover Hash Lemma over Lattices".

Lemma 6. *Let $L \subset \mathbb{Z}^n$ be a lattice of rank n of basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$. Let $\boldsymbol{x}_i$ for $1 \leqslant i \leqslant m$ be generated independently according to the distribution $\mathcal{D}_{\boldsymbol{B}}$. Set $s_1, \ldots, s_m \leftarrow \{0, 1\}$ and $t_1, \ldots, t_n \leftarrow \mathbb{Z} \cap [0, 2^\mu)$. Let $\boldsymbol{y} = \sum_{i=1}^{m} s_i \boldsymbol{x}_i + \sum_{i=1}^{n} t_i \boldsymbol{b}_i$ and $\boldsymbol{y'} \leftarrow \mathcal{D}_{2^\mu \boldsymbol{B}}$. Then the distributions $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y})$ and $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y'})$ are ε-statistically close, with $\varepsilon = mn \cdot 2^{-\mu} + 1/2\sqrt{|\det L|/2^m}$.*

Proof. We consider the intermediate variable:

$$\boldsymbol{y''} = \left(\sum_{i=1}^{m} s_i \boldsymbol{x}_i \bmod \boldsymbol{B}\right) + \sum_{i=1}^{n} t_i \boldsymbol{b}_i \; . \tag{11}$$

Firstly by applying the leftover hash lemma over the finite abelian group $G = \mathbb{Z}^n/L$, we obtain that the distributions $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \sum_{i=1}^{m} s_i \boldsymbol{x}_i \bmod \boldsymbol{B})$ and $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{\psi})$ are ε_1-statistically close, where $\boldsymbol{\psi} \leftarrow \mathcal{D}_{\boldsymbol{B}}$ and

$$\varepsilon_1 = 1/2\sqrt{|G|/2^m} = 1/2\sqrt{|\det(L)|/2^m} \; .$$

This implies that the distributions $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y''})$ and $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y'})$ are also ε_1-statistically close.

Secondly we write:

$$\sum_{i=1}^{m} s_i \boldsymbol{x}_i \bmod \boldsymbol{B} = \sum_{i=1}^{m} s_i \boldsymbol{x}_i - \sum_{j=1}^{n} \chi_j \boldsymbol{b}_j \tag{12}$$

where $\chi_j \in \mathbb{Z}$ for all j. We also write $\boldsymbol{x_i} = \sum_j \xi_{ij} \boldsymbol{b}_j$ where by definition $0 \leqslant \xi_{ij} < 1$ for all i, j. This gives:

$$\sum_{i=1}^{m} s_i \boldsymbol{x}_i \bmod \boldsymbol{B} = \sum_{i=1}^{m} s_i \sum_{j=1}^{n} \xi_{ij} \boldsymbol{b}_j - \sum_{j=1}^{n} \chi_j \boldsymbol{b}_j = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} s_i \xi_{ij} - \chi_j \right) \boldsymbol{b}_j \ ,$$

which implies $0 \leqslant \sum_{i=1}^{m} s_i \xi_{ij} - \chi_j < 1$ for all j, and therefore $0 \leqslant \chi_j \leqslant m$ for all j. Combining Equations (11) and (12) we have:

$$\boldsymbol{y''} = \sum_{i=1}^{m} s_i \boldsymbol{x}_i + \sum_{i=1}^{n} (t_i - \chi_i) \boldsymbol{b}_i \ ,$$

where as shown above $0 \leqslant \chi_i \leqslant m$ for all i. This implies that the distributions $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y})$ and $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y''})$ are ε_2-statistically close, with $\varepsilon_2 = mn2^{-\mu}$. Therefore the distributions $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y})$ and $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m, \boldsymbol{y'})$ are $(\varepsilon_1 + \varepsilon_2)$-statistically close, which proves the Lemma. □

We also show that the previous distribution $\mathcal{D}_{2^\mu \boldsymbol{B}}$ is not significantly modified when a small vector $\boldsymbol{z} \in \mathbb{Z}^n$ is added and the operator norm of the corresponding matrix B^{-1} is upper-bounded.

Lemma 7. *Let $L \subset \mathbb{Z}^n$ be a full-rank lattice of basis $\boldsymbol{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, and let $B \in \mathbb{Z}^{n \times n}$ be the matrix whose column vectors are the $\mathbf{b}_i$'s. For any $\boldsymbol{z} \in \mathbb{Z}^n$, the distribution of $\boldsymbol{z} + \mathcal{D}_{2^\mu \boldsymbol{B}}$ is ε-statistically close to that of $\mathcal{D}_{2^\mu \boldsymbol{B}}$, where $\varepsilon = 2^{-\mu} \cdot (\|\boldsymbol{z}\|_\infty \cdot \|B^{-1}\|_\infty + 1)$.*

Proof. Let $\boldsymbol{u} \leftarrow \mathcal{D}_{2^\mu \boldsymbol{B}}$ and $\boldsymbol{u''} \leftarrow \boldsymbol{z} + \mathcal{D}_{2^\mu \boldsymbol{B}}$. We can write:

$$\boldsymbol{u} = \boldsymbol{v} + \sum_{i=1}^{n} s_i \boldsymbol{b}_i$$

$$\boldsymbol{u''} = \boldsymbol{z} + \boldsymbol{v} + \sum_{i=1}^{n} s_i \boldsymbol{b}_i$$

where $\boldsymbol{v} \leftarrow \mathcal{D}_{\boldsymbol{B}}$ and $s_i \leftarrow [0, 2^\mu) \cap \mathbb{Z}$. We consider the intermediate variable:

$$\boldsymbol{u'} = ((\boldsymbol{z} + \boldsymbol{v}) \bmod \boldsymbol{B}) + \sum_{i=1}^{n} s_i \boldsymbol{b}_i \ .$$

The distribution of $\boldsymbol{u}$ and $\boldsymbol{u'}$ are clearly the same. Let $\boldsymbol{\psi} = \boldsymbol{z} + \boldsymbol{v}$. We have:

$$\boldsymbol{\psi} \bmod \boldsymbol{B} = \boldsymbol{\psi} - B \cdot \lfloor B^{-1} \cdot \boldsymbol{\psi} \rfloor = \boldsymbol{\psi} - \sum_{i=1}^{n} t_i \boldsymbol{b}_i$$

where $\boldsymbol{t} = \lfloor B^{-1} \cdot \boldsymbol{\psi} \rfloor$. This gives:

$$\boldsymbol{u'} = \boldsymbol{z} + \boldsymbol{v} + \sum_{i=1}^{n} (s_i - t_i) \boldsymbol{b}_i \ .$$

We have $\boldsymbol{t} = \lfloor B^{-1} \cdot \boldsymbol{z} + B^{-1} \cdot \boldsymbol{v} \rfloor$. Since $\boldsymbol{v}$ is in the half-open parallelepiped spanned by $\boldsymbol{B}$ we have that the components of $B^{-1} \cdot \boldsymbol{v}$ are in $[0, 1)$, which gives:

$$\|\boldsymbol{t}\|_\infty \leqslant \|B^{-1} \cdot \boldsymbol{z}\|_\infty + 1 \leqslant \|B^{-1}\|_\infty \cdot \|\boldsymbol{z}\|_\infty + 1 \ .$$

Therefore the variables $\boldsymbol{u'}$ and $\boldsymbol{u''}$ are ε-statistically close, with $\varepsilon = 2^{-\mu} \left(\|B^{-1}\|_\infty \cdot \|\boldsymbol{z}\|_\infty + 1 \right)$. This proves the Lemma. □

4.3 Re-Randomization of Encodings: Proof of Lemma 2

We are now ready to apply our "LHL over lattices" to prove the full randomization of encodings as stated in Lemma 2. Namely the re-randomization equation (7) can be rewritten in vector form as:

$$\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{X} \cdot \boldsymbol{b} + \boldsymbol{\Pi} \cdot \boldsymbol{b}'$$

where $\boldsymbol{b} \leftarrow \{0,1\}^\tau$ and $\boldsymbol{b'} \leftarrow ([0, 2^\mu) \cap \mathbb{Z})^n$, and the columns of the matrix $\boldsymbol{X} \in \mathbb{Z}^{n\times\tau}$ are uniformly and independently generated in the parallelepiped spanned by the columns of the matrix $\boldsymbol{\Pi} \in \mathbb{Z}^{n\times n}$. To conclude, it therefore suffices to apply Lemma 6 and Lemma 7, using additionally an upper bound on $\|\boldsymbol{\Pi}^{-1}\|_\infty$. For this we use the fact that $\boldsymbol{\Pi}$ has been generated as a diagonally dominant matrix.

Given a matrix $B = (b_{ij}) \in \mathbb{R}^{n\times n}$, we let $\Lambda_i(B) = \sum_{k\neq i} |b_{ik}|$. A matrix $B = (b_{ij}) \in \mathbb{R}^{n\times n}$ is said to be *diagonally dominant* if $|b_{ii}| > \Lambda_i(B)$ for all i. We recall the following fact for diagonally dominant matrices [Var75,Pri51].

Fact 1 *Let $B = (b_{ij}) \in \mathbb{R}^{n\times n}$ be a diagonally dominant matrix. Then the matrix B is invertible and $\|B^{-1}\|_\infty \leqslant \max_{i=1,\dots,n} (|b_{ii}| - \Lambda_i(B))^{-1}$ where $\|\cdot\|_\infty$ is the operator norm on $n \times n$ matrices with respect to the ℓ^∞ norm on $\mathbb{R}^n$.*

Fact 2 *Let $B = (b_{ij}) \in \mathbb{R}^{n\times n}$ be a diagonally dominant matrix. Then*

$$\prod_{i=1}^{n} (|b_{ii}| - \Lambda_i(B)) \leqslant |\det B| \leqslant \prod_{i=1}^{n} (|b_{ii}| + \Lambda_i(B))$$

Proof (of Lemma 2). We write $c_1 \equiv \left(r_i^{(1)} \cdot g_i + m_i\right)/z \bmod p_i$ for all $1 \leqslant i \leqslant n$ and define $\boldsymbol{r^{(1)}} = (r_i^{(1)}) \in \mathbb{Z}^n$. We also write $x_j \equiv r_{ij} \cdot g_i/z \pmod{p_i}$ and $\Pi_j \equiv \varpi_{ij} \cdot g_i/z \pmod{p_i}$ and define the matrix $\boldsymbol{X} = (r_{ij}) \in \mathbb{Z}^{n\times\tau}$ and $\boldsymbol{\Pi} = (\varpi_{ij}) \in \mathbb{Z}^{n\times n}$. From the re-randomization equation (7), we can write:

$$\boldsymbol{r} = \boldsymbol{r^{(1)}} + \boldsymbol{X} \cdot \boldsymbol{b} + \boldsymbol{\Pi} \cdot \boldsymbol{b}' \ ,$$

where $\boldsymbol{b} \leftarrow \{0,1\}^\tau$, and $\boldsymbol{b'} \leftarrow ([0, 2^\mu) \cap \mathbb{Z})^n$.

Since at instance generation the columns of $\boldsymbol{X}$ are generated uniformly and independently in the parallelepiped spanned by the columns of $\boldsymbol{\Pi}$, applying our "leftover hash lemma over lattices" (Lemma 6) we obtain that the distribution of $(\mathsf{params}, \boldsymbol{r})$ is ε_1-close to the distribution of $(\mathsf{params}, \boldsymbol{r^{(1)}} + \mathcal{D}_{2^\mu \boldsymbol{\Pi}})$, with

$$\varepsilon_1 = \tau n 2^{-\mu} + \frac{1}{2}\sqrt{|\det \boldsymbol{\Pi}|/2^\tau} \ .$$

Since $\boldsymbol{\Pi}$ is a diagonally dominant matrix, we obtain from Fact 2

$$|\det \boldsymbol{\Pi}| \leqslant \prod_{i=1}^{n} (|\varpi_{i,i}| + \Lambda_i(\boldsymbol{\Pi})) \leqslant (2n2^\rho)^n \leqslant 2^{n(\rho+\log_2(2n))} \ ,$$

which gives $\varepsilon_1 \leqslant n\tau 2^{-\mu} + 2^{(n(\rho+\log_2(2n))-\tau)/2}$. Therefore given the constraints $\tau \geqslant n\cdot(\rho+\log_2(2n))+2\lambda$ and $\mu \geqslant \rho + \alpha + \lambda$ we have that $\varepsilon_1 = \mathsf{negl}(\lambda)$.

Now, using Lemma 7 we obtain that the distribution of $(\mathsf{params}, \boldsymbol{r})$ if $(\varepsilon_1 + \varepsilon_2)$-close to that of $(\mathsf{params}, \mathcal{D}_{2^\mu \boldsymbol{\Pi}})$ for

$$\varepsilon_2 = 2^{-\mu}\Big(\|\boldsymbol{r^{(1)}}\|_\infty \cdot \|\boldsymbol{\Pi}^{-1}\|_\infty + 1\Big) \ .$$

We consider the initial level-0 encoding c and write $c \equiv r_j^{(c)} \cdot g_j + m_j \bmod p_j$ and $y \equiv r_j^{(y)} \cdot g_j + 1 \bmod p_j$, and define

$$\boldsymbol{r^{(c)}} = \left(r_1^{(c)}, \dots, r_n^{(c)}\right)^T \quad \text{and} \quad \boldsymbol{r^{(y)}} = \left(r_1^{(y)}, \dots, r_n^{(y)}\right)^T \ .$$

Since $c_1 = c \cdot y \bmod x_0$, we have

$$\|\boldsymbol{r}^{(\mathbf{1})}\|_\infty = \left\| \left(r_j^{(c)} + r_j^{(c)} r_j^{(y)} g_j + m_j r_j^{(y)} \right)_{j=1,\ldots,n} \right\|_\infty \leqslant \|\boldsymbol{r}^{(\boldsymbol{c})}\|_\infty \cdot \|\boldsymbol{r}^{(\boldsymbol{y})}\|_\infty \cdot 2^{\alpha+2} .$$

Therefore $\|\boldsymbol{r}^{(\mathbf{1})}\|_\infty \leqslant 2^{2\rho+\log_2(\ell)+\alpha+2}$. Now, by Fact 1, we have

$$\|\boldsymbol{\Pi}^{-1}\|_\infty \leqslant \frac{1}{\min_{i=1,\ldots,n}(|\varpi_{i,i}| - \Lambda_i(\boldsymbol{\Pi}))} \leqslant \frac{1}{n 2^\rho - (n-1)2^\rho} \leqslant 2^{-\rho} .$$

This gives $\varepsilon_2 \leqslant 2^{-\mu} + 2^{\rho+\log_2(\ell)+\alpha+2-\mu}$. With the constraint $\mu \geqslant \rho + \alpha + \lambda$, Lemma 2 is proved. □

5 Attacks against our Multilinear Scheme

5.1 Lattice Attack on the Encodings

We first describe a lattice attack against level-0 encodings. We consider an element $x_0 = \prod_{i=1}^n p_i$ and a set of τ integers $x_j \in \mathbb{Z}_{x_0}$ such that:

$$x_j \bmod p_i = r_{ij} g_i$$

where $r_{ij} \in (-2^\rho, 2^\rho) \cap \mathbb{Z}$. We want to estimate the complexity of the classical orthogonal lattice attack for recovering (some of) the noise values $r_{ij} g_i$.

This attack works by considering the integer vector formed by a subset of the x_j's, say $\boldsymbol{x} = (x_j)_{1 \leqslant j \leqslant t}$ for some $n < t \leqslant \tau$, and relating the lattice of vectors orthogonal to $\boldsymbol{x}$ mod x_0 to the lattice of vectors orthogonal to each of the corresponding noise value vectors $\boldsymbol{r}_i = (r_{ij} g_i)_{1 \leqslant j \leqslant t}$.

More precisely, let $L \subset \mathbb{Z}^t$ the lattice of vectors $\boldsymbol{u}$ such that:

$$\boldsymbol{u} \cdot \boldsymbol{x} \equiv 0 \pmod{x_0}.$$

Clearly, L contains $x_0 \mathbb{Z}^t$ so it is of full rank t. Moreover, we have

$$\mathrm{vol}(L) = [\mathbb{Z}^t : L] = x_0 / \gcd(x_0, x_1, \ldots, x_t) = x_0 .$$

As a result, we heuristically expect the successive minima of L to be around $\mathrm{vol}(L)^{1/t} \approx 2^{n \cdot \eta / t}$, and hence applying lattice reduction should yield a reduced basis $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t)$ with vectors of length $\|\boldsymbol{u}_k\| \approx 2^{n \cdot \eta/t + \iota t}$ for some constant $\iota > 0$ depending on the lattice reduction algorithm ($2^{\iota t}$ is the Hermite factor).

Now, a vector $\boldsymbol{u} \in L$ satisfies $\boldsymbol{u} \cdot \boldsymbol{x} \equiv 0 \pmod{x_0}$, so for each $i \in \{1, \ldots, n\}$, reducing modulo p_i gives:

$$\boldsymbol{u} \cdot \boldsymbol{r}_i \equiv 0 \pmod{p_i}.$$

In particular, if $\boldsymbol{u}$ is short enough to satisfy $\|\boldsymbol{u}\| \cdot \|\boldsymbol{r}_i\| < p_i$, this implies $\boldsymbol{u} \cdot \boldsymbol{r}_i = 0$ in $\mathbb{Z}$. As a result, if we have:

$$2^{n \cdot \eta/t + \iota t} \cdot 2^{\rho+\alpha} < 2^\eta, \tag{13}$$

we expect the vectors $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{t-n})$ from the previous lattice reduction step to be orthogonal to the $\boldsymbol{r}_i$'s, and hence computing the rank n orthogonal lattice to the lattice spanned by those vectors should reveal the $\boldsymbol{r}_i$'s.

Since t must be greater than n for the attack to apply, condition (13) implies in particular that:

$$\iota < \frac{\eta - \rho - \alpha}{n}.$$

Since a Hermite factor of $2^{\iota t}$ is achieved in time $2^{\Omega(1/\iota)}$ (usually by carrying out BKZ reduction with block size $\beta = \omega(1/\iota)$, in which each block is BKZ-reduced in time exponential in β, see e.g. [HPS11]), we obtain that this orthogonal lattice attack has a complexity exponential in n. In fact, with $\gamma = \eta \cdot n$, we get a time complexity of $2^{\Omega(\gamma/\eta^2)}$, similar to [DGHV10, §5.2] (see also [CH12]).

5.2 GCD Attack on the Zero-testing Parameter

We consider the ratio modulo x_0 of two coefficients from the zero-testing vector $\boldsymbol{p}_{zt}$, namely $u := (\boldsymbol{p}_{zt})_1/(\boldsymbol{p}_{zt})_2 \bmod x_0$. From (8) we obtain for all $1 \leqslant i \leqslant \ell$:

$$u \equiv h_{i1}/h_{i2} \pmod{p_i}$$

We can therefore recover p_i by computing $\gcd(h_{i2} \cdot u - h_{i1}, x_0)$ for all possible h_{i1}, h_{i2}. Since the h_{ij}'s are upper bounded by 2^β in absolute value, the attack has complexity $\mathcal{O}(2^{2\beta})$. By using a technique similar to [CN12], the attack complexity can be reduced to $\tilde{\mathcal{O}}(2^\beta)$.

5.3 Hidden Subset Sum Attack on Zero Testing

One can consider an attack similar to Section 5.1 on the zero-testing parameters. The zero-testing vector $\boldsymbol{\omega} = c \cdot \boldsymbol{p}_{zt} \bmod x_0$ corresponding to an encoding c of zero can be written as:

$$\boldsymbol{\omega} = \sum_{i=1}^{n} \Big(r_i \cdot \prod_{i' \neq i} p_{i'} \Big) \boldsymbol{h}_i = \sum_{i=1}^{n} R_i \boldsymbol{h}_i,$$

where the $\boldsymbol{h}_i$'s are the lines of the zero-testing matrix $\boldsymbol{H}$. This gives an expression of $\boldsymbol{\omega}$ as a linear combination of the unknown vectors $\boldsymbol{h}_i$, so we can think of an approach similar to the hidden subset sum attack of Nguyen and Stern [NS99] to recover the unknown $\boldsymbol{h}_i$'s or the R_i's.

Such an approach, like the one from the previous section, would first involve computing the lattice $L \subset \mathbb{Z}^n$ of vectors $\boldsymbol{u}$ orthogonal to $\boldsymbol{\omega}$ modulo x_0, and hoping that the vectors in a reduced basis of L are short enough that they must necessarily be orthogonal to each of the $\boldsymbol{h}_i$'s in $\mathbb{Z}^n$.

However, L is a lattice of rank n and volume $x_0 \approx 2^{n \cdot \eta}$, so we expect its shortest vectors to be of length roughly 2^η. The attack can then only work if such a short vector $\boldsymbol{u}$ is necessarily orthogonal to each of the $\boldsymbol{h}_i$'s in $\mathbb{Z}^n$. Equivalently, this will happen if the vector $\boldsymbol{v} = (\boldsymbol{u} \cdot \boldsymbol{h}_1, \dots, \boldsymbol{u} \cdot \boldsymbol{h}_n) \in \mathbb{Z}^n$, which we know is orthogonal to $(R_1, \dots, R_n)$ modulo x_0, is significantly shorter than the shortest vector in the lattice L' of vectors orthogonal to $(R_1, \dots, R_n)$ modulo x_0. But again L' is of rank n and volume x_0, so its shortest vectors is of length about 2^η, and hence $\boldsymbol{v}$ will typically not be shorter.

Therefore, the Nguyen–Stern hidden subset sum attack does not apply to our setting.

5.4 Attacks on the Inverse Zero Testing Matrix

A related observation is that if $\boldsymbol{\omega} = c \cdot \boldsymbol{p}_{zt} \bmod x_0$ denotes again the zero-testing vector corresponding to an encoding c of zero, and $\boldsymbol{T} = (\boldsymbol{H}^{-1})^T$ is the inverse zero-testing matrix, then, by Equation (10):

$$\boldsymbol{T}\boldsymbol{\omega} = (R_1, \dots, R_n) \in \mathbb{Z}^n,$$

where, as above, $R_i = r_i \cdot (x_0/p_i)$. In particular, if the lines of $\boldsymbol{T}$ are written as $\boldsymbol{t}_i$, we get that for each $i \in \{1, \dots, n\}$:

$$p_i \boldsymbol{t}_i \cdot \boldsymbol{\omega} \equiv 0 \pmod{x_0}.$$

Thus, since $p_i \boldsymbol{t}_i$ is relatively small, we might hope to recover it as a short vector in the lattice of vectors orthogonal to $\boldsymbol{\omega}$ modulo x_0. However, as we have seen previously, we expect a reduced basis of that lattice to have vectors of length roughly 2^η, whereas $p_i \boldsymbol{t}_i$ is of length about $2^{\eta+\beta}$. Therefore, provided that, say, β is super-logarithmic, there are exponentially many linear combinations of the reduced basis vectors under the target length, and hence we cannot hope to recover $p_i \boldsymbol{t}_i$ in that fashion.

A more sophisticated attack based on the same observation uses the result of Hermann and May [HM08] on solving linear equations modulo an unknown factor of a public modulus. In our case, the components t_{ij} of $\boldsymbol{t}_i$ form a small solution (smaller than 2^β) of the linear equation:

$$\sum_{j=1}^{n} \omega_j \cdot t_{ij} \equiv 0 \pmod{x_0/p_i}$$

modulo the unknown factor x_0/p_i of x_0. The technique of Hermann–May can thus recover $\boldsymbol{t}_i$ and factor x_0 provided that β is small enough relative to x_0. For sufficiently large n, by [HM08, Theorem 4], this should be possible as long as $\beta/\eta < 1$. However, as noted by the authors, the complexity of the attack is exponential in n (it involves reducing a lattice of dimension $\Omega(\exp n)$), and there does not seem to be a way to approximate the solution in polynomial time, so the attack does not apply to our setting even though we do choose $\beta < \eta$.

5.5 A Note on GGH's Zeroizing Attack

In [GGH13] the authors describe a "zeroizing" attack against their scheme that consists in multiplying a given level-i encoding c by a level-$(\kappa - i)$ encoding of 0 to get a level-κ encoding of 0, and then multiplying by the zero-testing parameter $\boldsymbol{p}_{zt}$; one obtains an encoding of a (deterministic) multiple of the coset of c but in the plaintext space. This attack does not enable to solve the GDDH problem because one does not get a small representative of that coset, but it enables to solve some decisional problems involving low-level (below κ) encodings, such as the decisional subgroup membership problem using composite-order maps, and the decisional linear problem.

Surprisingly this attack does not seem to apply against our scheme; namely we do not get a similar encoding in the plaintext space from the zero-testing parameter. Therefore the subgroup membership assumption and the decision linear assumption could still hold in our scheme.

6 Optimizations and Implementation

In this section we describe an implementation of our scheme in the one-round N-way Diffie-Hellman key exchange protocol; we recall the protocol in Appendix D, as described in [BS03,GGH13].

We note that without optimizations the size of the public parameters makes our scheme completely unpractical; this is also the case in [GGH13]. Namely, for sampling we need to store at least $n \cdot \alpha$ encodings (resp. $n \cdot \rho$ encodings for re-randomization), each of size $n \cdot \eta$ bits; the public-key size is then at least $n^2 \cdot \eta \cdot \alpha$ bits. With $n \simeq 10^4$, $\eta \simeq 10^3$ and $\alpha \simeq 80$, the public-key size would be at least 1 TB.[4] Therefore we use three heuristic optimizations to reduce the memory requirement.

1. Non-uniform sampling: for the sampling algorithm we use a small number of encodings ℓ only; this implies that the sampling cannot be proved uniform anymore.
2. Quadratic re-randomization: we only store a small subset of encodings which are later combined pairwise to generate the full set of encodings. This implies that the randomization of encodings becomes heuristic only.
3. Integer p_{zt}: we use a single integer p_{zt} instead of a vector $\boldsymbol{p}_{zt}$ with n components. An encoding c of zero still gives a small integer $\omega = p_{zt} \cdot c \bmod x_0$, but the converse does not necessarily hold anymore.

[4] In [GGH13] the following approximate setting is suggested: $n = \tilde{\mathcal{O}}(\kappa\lambda^2)$, $q = 2^{n/\lambda}$ and $m = \mathcal{O}(n^2)$. The public-key size contains at least m encodings of size $n \log_2 q$ bits each. Taking exactly $n = \kappa\lambda^2$ and $m = n^2$, the public-key size is then $m \cdot n \cdot (n/\lambda) = n^4/\lambda = \kappa^4\lambda^7$. With $\kappa = 6$ and $\lambda = 80$, we get a public-key size of 3400 TB.

6.1 Non-uniform Sampling

For sampling level-zero encodings we use a smaller value for ℓ, the number of encodings x_j in the public parameters. There is a simple meet-in-the-middle attack with complexity $\mathcal{O}(2^{\ell/2})$; therefore we take $\ell = 2\lambda$. In this case the sampling cannot be proved uniform in $R = \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_n}$ anymore. However this does not seem to make the GDDH problem easier.

Note also that for such small ℓ given a level-0 encoding c, one can efficiently recover the coefficients of the subset sum with LLL, since this is a subset-sum problem with density $\ell/(\eta \cdot n) \ll 1$; however this does not give an attack, as in GDDH such level-0 encoding c is not available.[5]

6.2 Quadratic Re-randomization

To reduce the parameters size we use a simple optimization that consists in storing only a small subset of the public elements and combining them pairwise to generate the full public-key. Such optimization was originally described in [GH11] for reducing the size of the encryption of the secret-key bits in the implementation of Gentry's FHE scheme [Gen09]. It was also used in [CMNT11] to reduce the public-key size of the DGHV scheme; however, as opposed to [CMNT11] our randomization of encodings becomes heuristic only, whereas in [CMNT11] the semantic security was still guaranteed.

For re-randomization we only store $\Delta = \lfloor \sqrt{n} \rfloor$ encoding $x_j^{(0)}$ at level 0 and also Δ encodings $x_j^{(1)}$ at level 1. The $x_j^{(0)}$ encode random $\boldsymbol{m}_j \in R$, while the $x_j^{(1)}$ are encodings of $\mathbf{0}$. Then by pairwise multiplication we can generate $\Delta^2 \simeq n$ randomization elements at level 1, which are all encodings of $\mathbf{0}$. More precisely, we have for $b = 0, 1$ and $1 \leqslant j \leqslant \Delta$:

$$x_j^{(b)} \equiv \frac{r_{ij}^{(b)} \cdot g_i + (1-b) \cdot f_{ij}}{z^b} \pmod{p_i} ,$$

where $r_{ij}^{(b)}$ are random ρ-bit integers, and f_{ij} are random integers modulo g_i.

Given a level-1 encoding c_1, we randomize it using a random subset-sum of pairwise products of the previous encodings:

$$c_1' = c_1 + \sum_{i,j=1}^{\Delta} \alpha_{ij} \cdot x_i^{(0)} \cdot x_j^{(1)} \bmod x_0 ,$$

where the α_{ij}'s are random bits; note that we don't use the encodings Π_j anymore. As a further optimization, we can use as in [CMNT11] a sparse vector α_{ij}, with small Hamming weight θ. There is a meet-in-the-middle attack of complexity $\mathcal{O}(n^{\theta/2})$. In our implementation we take $\theta = 16$; the reRand operation then becomes very efficient.

Writing as previously $c_1' \equiv (r_i' \cdot g_i + m_i)/z \pmod{p_i}$, we obtain under this optimization:

$$|r_i' \cdot g_i + m_i| \leqslant (\ell + \theta) \cdot 2^{2(\rho+\alpha)} .$$

When computing the product of κ such level-1 encodings and one level-0 encoding as in multipartite Diffie-Hellman key exchange, we obtain the following updated bound for the $\log_2$ infinite norm of the vector $\boldsymbol{r}$ from Lemma 3:

$$\rho_f \leqslant \kappa \cdot (2\rho + 2\alpha + \log_2(\ell + \theta)) + \rho + \log_2 \ell + 1 .$$

[5] Alternatively one could use the same quadratic technique as in [CMNT11]; in that case the sampling could still be proved uniform in R.

6.3 Zero-Testing Element

Instead of generating a zero-testing vector $\boldsymbol{p}_{zt}$ with n components, we publish a zero-testing element p_{zt} which is a single integer:

$$p_{zt} = \sum_{i=1}^{n} h_i \cdot (z^{\kappa} \cdot g_i^{-1} \bmod p_i) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0$$

where the h_i's are random β-bit integers. Therefore, we obtain a single integer $\omega = p_{zt} \cdot c \bmod x_0$, with

$$\omega = \sum_{i=1}^{n} h_i \cdot \big(r_i + m_i \cdot (g_i^{-1} \bmod p_i)\big) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0.$$

As before, if $\|\boldsymbol{r}\|_\infty < 2^{\rho_f}$ we still have $|\omega| < x_0 \cdot 2^{-\nu-\lambda-2}$. However the converse is no longer true: we can have $|\omega| < x_0 \cdot 2^{-\nu}$ for an encoding of a non-zero vector $\boldsymbol{m}$. This implies that two encodings of different vectors can now extract to the same value. While it is actually easy to generate such collisions using LLL, this does not seem to give an attack against the GDDH problem.

6.4 Parameters and Timings

We have implemented a one-round N-way Diffie-Hellman key exchange protocol with $N = 3, 5, 7$ users, in C++ using the Gnu MP library [Gt13] to perform operations on large integers. We refer to Appendix D for a description of the protocol. We provide our concrete parameters and the resulting timings in Table 1, for security parameters ranging from 52 to 80 bits.[6]

Table 1. Parameters and timings to instantiate a *one-round N-way Diffie-Hellman key exchange protocol* with $\ell = 160$, $\beta = 80$, $\alpha = 80$, $N = 3, 5, 7$ (*i.e.* $\kappa = 2, 4, 6$ respectively) and $\nu = 160$ on a 16-core computer (Intel(R) Xeon(R) CPU E7-8837 at 2.67GHz) using GMP 5.1.1. Note that the Setup was parallelized on the 16 cores to speed-up the process while the other steps ran on a single core. We only derived a common ν-bit session key without using a randomness extractor.

(a) 3-partite Diffie-Hellman

Instantiation	λ	κ	n	η	Δ	ρ	$\gamma = n \cdot \eta$	pk size	Setup (once)	Publish (per party)	KeyGen (per party)
Small	52	2	615	897	24	52	$0.5 \cdot 10^6$	14 MB	1.8 s	0.08 s	0.04 s
Medium	62	2	2445	957	49	62	$2.2 \cdot 10^6$	70 MB	20 s	0.45 s	0.22 s
Large	72	2	10080	1027	100	74	$7.3 \cdot 10^6$	330 MB	1008 s	2.4 s	1.2 s
Extra	80	2	14190	1110	119	89	$12.0 \cdot 10^6$	599 MB	3835 s	3.4 s	1.7s

(b) 5-partite Diffie-Hellman

Instantiation	λ	κ	n	η	Δ	ρ	$\gamma = n \cdot \eta$	pk size	Setup (once)	Publish (per party)	KeyGen (per party)
Small	52	4	555	1439	23	52	$0.8 \cdot 10^6$	20 MB	5 s	0.13 s	0.10 s
Medium	62	4	2205	1539	46	62	$3.4 \cdot 10^6$	102 MB	30 s	0.69 s	0.56 s
Large	72	4	8880	1648	94	73	$14.6 \cdot 10^6$	536 MB	1205 s	3.4 s	2.8 s
Extra	80	4	26550	1782	162	87	$47.3 \cdot 10^6$	1.6 GB	24554 s	14.8 s	11.8 s

(c) 7-partite Diffie-Hellman

Instantiation	λ	κ	n	η	Δ	ρ	$\gamma = n \cdot \eta$	pk size	Setup (once)	Publish (per party)	KeyGen (per party)
Small	52	6	525	1981	22	52	$1.0 \cdot 10^6$	26 MB	7 s	0.18 s	0.20 s
Medium	62	6	2055	2121	45	62	$4.4 \cdot 10^6$	133 MB	38 s	0.86 s	1.05 s
Large	72	6	8250	2261	90	72	$18.7 \cdot 10^6$	709 MB	2038 s	4.9 s	5.7 s
Extra	80	6	26115	2438	161	85	$63.7 \cdot 10^6$	2.6 GB	27295 s	17.8 s	20.2 s

[6] More precisely, for a security level λ we expect that the best attack requires at least 2^{λ} clock cycles. Note that we can take $\rho < 2\lambda$ because the GCD attack from [CN12] has a large polynomial factor in front of $2^{\rho/2}$.

The timings above show that our scheme is relatively practical, as the KeyGen phase of the multipartite Diffie-Hellman protocol requires only a few seconds per user; however the parameter size is still very large even with our optimizations.

Acknowledgments

We would like to thank the Crypto referees for their helpful comments.

References

[AGHS12] Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. Discrete Gaussian Leftover Hash Lemma over infinite domains. Cryptology ePrint Archive, Report 2012/714, 2012. `http://eprint.iacr.org/`.

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

[BS03] D. Boneh and A. Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.

[CCK$^+$13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 315–335. Springer, 2013.

[CH12] Henry Cohn and Nadia Heninger. Approximate common divisors via lattices. In *ANTS X*, 2012.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Batch fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/036, 2013. `http://eprint.iacr.org/`.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2011.

[CN12] Yuanmi Chen and Phong Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 502–519. Springer, 2012.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[GH11] Craig Gentry and Shai Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In Kenneth Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011.

[Gt13] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.1.1 edition, 2013. `http://gmplib.org/`.

[HILL99] Johan Håstad, Russel Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28:12–24, 1999.

[HM08] Mathias Herrmann and Alexander May. Solving linear equations modulo divisors: On factoring given any bits. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 406–424. Springer, 2008.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 447–464. Springer, 2011.

[Lep] Tancrède Lepoint. *An Implementation of Multilinear Maps over the Integers*. Available under the Creative Commons License BY-NC-SA at `https://github.com/tlepoint/multimap`.

[NS99] Phong Q. Nguyen and Jacques Stern. The hardness of the hidden subset sum problem and its cryptographic implications. In Michael J. Wiener, editor, *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 1999.

[Pri51] G. Baley Price. Bounds for determinants with dominant principal diagonal. *Proceedings of the American Mathematical Society*, 2(3):497–502, 1951.

[Sto96] Arne Storjohann. Near optimal algorithms for computing Smith Normal Forms of integer matrices. In Erwin Engeler, B. F. Caviness, and Yagati N. Lakshman, editors, *ISSAC '96*, pages 267–274. ACM, 1996.

[Var75] James M. Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and its Applications*, 11(1):3–5, 1975.

A Proof of Lemma 1

Write $x'_j \equiv r'_{ij} \cdot g_i + a_{ij} \pmod{p_i}$ for $1 \leqslant j \leqslant \ell$ and $1 \leqslant i \leqslant n$. Each component a_{ij} is randomly generated in $[0, g_i) \cap \mathbb{Z}$; therefore the column vectors $\boldsymbol{a}_j$ of the matrix (a_{ij}) are randomly and independently generated in R. By the leftover hash lemma over finite groups (Lemma 5), we have that $(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_\ell, \boldsymbol{m})$ with $\boldsymbol{m} = \boldsymbol{A} \cdot \boldsymbol{b}$ is ε-uniform over $R^{\ell+1}$, where

$$\varepsilon = \frac{1}{2}\sqrt{\frac{|R|}{2^\ell}} \leqslant 2^{(\alpha \cdot n - \ell)/2} .$$

Therefore by taking $\ell \geqslant n \cdot \alpha + 2\lambda$ we obtain that the distribution of $(\mathsf{params}, \boldsymbol{m})$ is statistically close to the distribution of $(\mathsf{params}, \boldsymbol{m'})$ for $\boldsymbol{m'} \leftarrow R$. $\square$

B Proof of Lemma 3

We have assumed

$$\rho_f + \lambda + \alpha + 2\beta \leqslant \eta - 8 ,$$

which gives:

$$\nu = \eta - \beta - \rho_f - \lambda - 3 \geqslant \alpha + \beta + 5 . \tag{14}$$

We consider the vector $\boldsymbol{R} = (R_i)_{1 \leqslant i \leqslant n}$ where:

$$R_i = ((r_i + m_i \cdot g_i^{-1}) \bmod p_i) \cdot (x_0/p_i) . \tag{15}$$

Equation (10) can then be written:

$$\boldsymbol{\omega} = \boldsymbol{H}^T \cdot \boldsymbol{R} \bmod x_0 . \tag{16}$$

If $\boldsymbol{m} = 0$ then we have $R_i = r_i \cdot x_0/p_i$ for all i, which gives using $p_i \geqslant 2^{\eta-1}$ for all i:

$$\|\boldsymbol{R}\|_\infty \leqslant \|\boldsymbol{r}\|_\infty \cdot \max_{1 \leqslant i \leqslant n} (x_0/p_i) \leqslant \|\boldsymbol{r}\|_\infty \cdot x_0 \cdot 2^{-\eta+1} .$$

Since by definition $-p/2 < (z \bmod p) \leqslant p/2$, we have $|z \bmod p| \leqslant |z|$ for any z, p; therefore we obtain from (16) using $\|\boldsymbol{r}\|_\infty < 2^{\rho_f}$

$$\|\boldsymbol{\omega}\|_\infty = \|\boldsymbol{H}^T \cdot \boldsymbol{R} \bmod x_0\|_\infty \leqslant \|\boldsymbol{H}^T \cdot \boldsymbol{R}\|_\infty \leqslant \|\boldsymbol{H}^T\|_\infty \cdot \|\boldsymbol{R}\|_\infty < x_0 \cdot 2^{\beta+\rho_f-\eta+1} = x_0 \cdot 2^{-\nu-\lambda-2} .$$

Conversely assume that $\|\boldsymbol{\omega}\|_\infty < x_0 \cdot 2^{-\nu+2}$. From (16) we have:

$$\boldsymbol{R} \equiv (\boldsymbol{H}^{-1})^T \cdot \boldsymbol{\omega} \pmod{x_0} . \tag{17}$$

From (15) we have $\|\boldsymbol{R}\|_\infty < x_0/2$. Moreover from (14) we have $\nu - \beta \geqslant \alpha + 5$, which gives

$$\|(\boldsymbol{H}^{-1})^T \cdot \boldsymbol{\omega}\|_\infty \leqslant \|(\boldsymbol{H}^{-1})^T\|_\infty \cdot \|\boldsymbol{\omega}\|_\infty \leqslant x_0 \cdot 2^{\beta-\nu+2} \leqslant x_0 \cdot 2^{-\alpha-3} < x_0/2 . \tag{18}$$

This shows that Equation (17) holds in $\mathbb{Z}$ and not only modulo x_0; therefore we must have

$$\|\boldsymbol{R}\|_\infty \leqslant x_0 \cdot 2^{\beta-\nu+2} .$$

Letting $v_i = (r_i + m_i \cdot g_i^{-1}) \bmod p_i$ for $1 \leqslant i \leqslant n$, this gives $|v_i| \cdot (x_0/p_i) \leqslant x_0 \cdot 2^{\beta-\nu+2}$, and therefore $|v_i| \leqslant p_i \cdot 2^{\beta-\nu+2}$ for all i. We have $g_i \cdot (v_i - r_i) \equiv m_i \pmod{p_i}$; we show that the equality actually holds over $\mathbb{Z}$. Namely for all i we have $|m_i| < g_i < p_i/2$ and with $g_i < 2^\alpha$ we get:

$$|g_i \cdot (v_i - r_i)| \leqslant g_i \cdot (|v_i| + |r_i|) \leqslant p_i \cdot 2^{\alpha+\beta-\nu+2} + 2^{\alpha+\rho_f} \leqslant p_i/8 + p_i/8 < p_i/2$$

which implies that the equality $g_i \cdot (v_i - r_i) = m_i$ holds over $\mathbb{Z}$. Therefore $m_i \equiv 0 \pmod{g_i}$ for all i, which implies $\boldsymbol{m} = \boldsymbol{0}$. This proves Lemma 3. $\square$

C Comparison with GGH Multilinear Maps

In this section we rewrite our scheme using exactly the same notations as in [GGH13] whenever possible, to better highlight the similarities.

The construction in [GGH13] works in the polynomial ring $R = \mathbb{Z}[X]/(X^n+1)$, where n is large enough to ensure security. One generates a secret short ring element $\boldsymbol{g} \in R$, generating a principal ideal $\mathcal{I} = \langle \boldsymbol{g} \rangle \subset R$. One also generates an integer parameter q and another random secret $\boldsymbol{z} \in R/qR$. One encodes elements of the quotient ring $R/\mathcal{I}$, namely elements of the form $\boldsymbol{e} + \mathcal{I}$ for some $\boldsymbol{e}$, as follows: a level-i encoding of the coset $\boldsymbol{e}+\mathcal{I}$ is an element of the form $u_k = [\boldsymbol{c}/\boldsymbol{z}^i]_q$, where $\boldsymbol{c} \in \boldsymbol{e}+\mathcal{I}$ is short. Such encodings can be both added and multiplied, as long as the norm of the numerators remain shorter than q; in particular the product of κ encodings at level 1 gives an encoding at level κ. For such level-κ encodings one can then define a zero-testing parameter $\boldsymbol{p}_{zt} = [\boldsymbol{h}\boldsymbol{z}^\kappa/\boldsymbol{g}]_q$, for some small $\boldsymbol{h} \in R$. Then given a level-κ encoding $\boldsymbol{u} = [\boldsymbol{c}/\boldsymbol{z}^\kappa]$ one can compute $[\boldsymbol{p}_{zt} \cdot \boldsymbol{u}]_q = [\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}]_q$; when $\boldsymbol{c}$ is an encoding of zero we have $\boldsymbol{c}/\boldsymbol{g} \in R$, which implies that $\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}$ is small in R, and therefore $[\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}]_q$ is small; this provides a way to test whether a level-κ encoding $\boldsymbol{c}$ is an encoding of 0.

In our construction one could write $R = \mathbb{Z}^n$, and define a secret short ring element $\boldsymbol{g} \in R$, generating a principal ideal $\mathcal{I} = \langle \boldsymbol{g} \rangle \subset R$, which gives $\mathcal{I} = (g_i\mathbb{Z})_{1 \leqslant i \leqslant n}$. We also generate a ring element $\boldsymbol{p} \in R$ and let the principal ideal $\mathcal{J} = \langle \boldsymbol{p} \rangle \subset R$, which gives $\mathcal{J} = (p_i\mathbb{Z})_{1 \leqslant i \leqslant n}$. We let $q := x_0 = \prod_{i=1}^n p_i$ and for convenience we denote by $[\boldsymbol{u}]_q$ the CRT isomorphism from $R/\mathcal{J}$ to $\mathbb{Z}_q$. As in [GGH13], in our scheme a level-i encoding of the coset $\boldsymbol{e}_\mathcal{I} = \boldsymbol{e} + \mathcal{I}$ is an element of the form $u = [\boldsymbol{c}/\boldsymbol{z}^i]_q$ where $\boldsymbol{c} \in \boldsymbol{e}_\mathcal{I}$ is short. Such encodings can be both added and multiplied, by working over the integers via the CRT isomorphism $[\cdot]_q$.

However, we cannot apply the zero-testing procedure from [GGH13] in a straightforward way. Namely one could define the zero-testing parameter $p_{zt} = [\boldsymbol{h}\boldsymbol{z}^\kappa/\boldsymbol{g}]_q$ as in [GGH13] where $\boldsymbol{h} \in \mathbb{Z}^n$ is a relatively small ring element. As in [GGH13] given a level-κ encoding $u = [\boldsymbol{c}/\boldsymbol{z}^\kappa]_q$ one would compute the element:

$$\omega = p_{zt} \cdot u = \left[\frac{\boldsymbol{h}\boldsymbol{z}^\kappa}{\boldsymbol{g}} \cdot \frac{\boldsymbol{c}}{\boldsymbol{z}^\kappa}\right]_q = \left[\frac{\boldsymbol{h}\boldsymbol{c}}{\boldsymbol{g}}\right]_q . \tag{19}$$

As in [GGH13] if u is an encoding of 0 then $\boldsymbol{c}$ is a multiple of $\boldsymbol{g}$ over $\mathbb{Z}^n$ hence $\boldsymbol{c}/\boldsymbol{g} \in \mathbb{Z}^n$ is short and therefore the vector $\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g} \in \mathbb{Z}^n$ is short. However this does not imply that the corresponding integer ω obtained by CRT in (19) is small, and we do not have a simple way of identifying integers whose reductions modulo the unknown p_i's are small.

Instead, we can define a slightly different notation: we consider the following additive homomorphism $R/\mathcal{J} \longrightarrow \mathbb{Z}_q$

$$\boldsymbol{u} \to \{\boldsymbol{u}\}_q = \sum_{i=1}^n u_i \cdot \prod_{i' \neq i} p_{i'} \bmod q$$

where as before $q = x_0 = \prod_{i=1}^n p_i$ and we define the zero-testing parameter:

$$p_{zt} := \{\boldsymbol{h}\boldsymbol{z}^\kappa/\boldsymbol{g}\}_q$$

As in [GGH13] given a level-κ encoding $u = [\boldsymbol{c}/\boldsymbol{z}^\kappa]_q$ one can compute the element:

$$\omega = p_{zt} \cdot u \bmod q = \left\{\frac{\boldsymbol{h}\boldsymbol{z}^\kappa}{\boldsymbol{g}}\right\}_q \cdot \left[\frac{\boldsymbol{c}}{\boldsymbol{z}^\kappa}\right]_q = \left\{\frac{\boldsymbol{h}\boldsymbol{z}^\kappa}{\boldsymbol{g}} \cdot \frac{\boldsymbol{c}}{\boldsymbol{z}^\kappa}\right\}_q = \left\{\frac{\boldsymbol{h}\boldsymbol{c}}{\boldsymbol{g}}\right\}_q$$

As in [GGH13] if u is an encoding of 0 then $\boldsymbol{c}$ is a multiple of $\boldsymbol{g}$ over $\mathbb{Z}^n$ hence $\boldsymbol{c}/\boldsymbol{g} \in \mathbb{Z}^n$ is short and therefore the vector $\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g} \in \mathbb{Z}^n$ is short; this time, this implies that $\omega = \{\boldsymbol{h}\boldsymbol{c}/\boldsymbol{g}\}_q$ is a short integer.

D One-Round N-Way Diffie-Hellman Key Exchange Protocol

In 2003, Boneh and Silverberg showed how to perform a multipartite Diffie-Hellman key exchange using multilinear maps [BS03]. Consider N parties wishing to set up a shared secret key s using a one-round protocol (*i.e.* in which each party broadcasts one value to all other parties and the N broadcasts occur simultaneously). Once the N broadcast values are known, each party should be able to locally compute a global shared secret s. Let us recall the definition of such a protocol using the notation from [BS03,GGH13]. A one-round N-way key exchange scheme consists of the following three randomized probabilistic polynomial-time algorithms:

$\mathsf{Setup}(\lambda, N)$. From a security parameter λ and the number of participants N, this algorithm runs in polynomial time in λ, N and outputs public parameters params.

$\mathsf{Publish}(\mathsf{params}, i)$. Given a value $i \in \{1, \ldots, N\}$, this algorithm outputs a key pair $(\mathsf{pub}_i, \mathsf{priv}_i)$. Party i broadcasts pub_i to all other parties and keep priv_i secret.

$\mathsf{KeyGen}(\mathsf{params}, i, \mathsf{priv}_i, \{\mathsf{pub}_j\}_{j \neq i})$. Party i computes KeyGen on all the collected public (broadcast) values $\{\mathsf{pub}_j\}_{j \neq i}$ and its secret value priv_i. This algorithm outputs a key s_i.

The protocol is said to be correct if the N parties generate the same shared key s with high probability, *i.e.* $s = s_1 = \cdots = s_N$. A correct protocol is said to be secure if, given all N public values pub_i, no polynomial time algorithm can distinguish the true shared secret s from a random string.

As in [GGH13], our scheme can be used to instantiate a one-round N-way Diffie-Hellman key exchange protocol in the common reference string model, under the GDDH assumption with $N = \kappa + 1$ users. We recall and adapt the construction from [GGH13]:

$\mathsf{Setup}(1^\lambda, 1^N)$. Output $(\mathsf{params}, \boldsymbol{p}_{zt}) \leftarrow \mathsf{InstGen}(1^\lambda, 1^\kappa)$ as the public parameter, with $\kappa = N - 1$.

$\mathsf{Publish}(\mathsf{params}, i)$. Each party i samples a random $c_i \leftarrow \mathsf{samp}(\mathsf{params})$ as a secret value, and publishes as the public value the corresponding level-1 encoding

$$c_i' \leftarrow \mathsf{reRand}(\mathsf{params}, 1, \mathsf{enc}(\mathsf{params}, 1, c_i)) .$$

$\mathsf{KeyGen}(\mathsf{params}, \boldsymbol{p}_{zt}, i, c_i, \{c_j'\}_{j \neq i})$. Each party i computes $\tilde{c}_i = c_i \cdot \prod_{j \neq i} c_j'$, and uses the extraction routine to locally compute the key $s \leftarrow \mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, \tilde{c}_i)$.

The correctness of the protocols follows from the fact that all parties get valid encodings of the same vector, hence with the parameters as given in Section 3.1, the extraction property implies that they should extract the same key with overwhelming probability.

The security of the protocol follows from the randomness property of the extraction property of the extraction procedure and the GDDH hardness assumption.

Theorem 3 ([GGH13]). *The protocol described above is a secure one-round N-way Diffie-Hellman key exchange protocol if the GDDH assumption holds for the underlying encoding scheme.*

Proof. We need to show that an attacker who sees all the public keys cannot distinguish the output of the first party (say) from a uniform random string. Now Party 1 extracts the same string as one would extract from $c = \mathsf{reRand}(\mathsf{params}, \kappa, \prod_{i=1}^{N} c_i)$.

By GDDH, the adversary cannot distinguish c from $c' = \mathsf{reRand}(\mathsf{params}, \kappa, b)$ for a random and independent $b \leftarrow \mathsf{samp}(\mathsf{params})$. Now by the randomness property of the sampling procedure (*i.e.* Lemma 1), b is nearly uniformly distributed in R. Therefore, by the randomness property of the extraction function, we conclude that $\mathsf{ext}(\mathsf{params}, \boldsymbol{p}_{zt}, c')$ is a nearly uniform string, completing the proof. □

E Uniform Sampling of a Parallelepiped

In order to sample a uniformly random element in the half-open parallelepiped defined by the column vectors $\boldsymbol{\varpi}_j$ of matrix $\boldsymbol{\Pi} \in \mathbb{Z}^{n\times n}$, one can proceed as follows.

First, compute the Smith Normal Form for $\boldsymbol{\Pi}$. This is easily done in polynomial time, and can be done with near optimal complexity using Storjohann's algorithm [Sto96]. This yields a basis $(\boldsymbol{b}_1, \dots, \boldsymbol{b}_n)$ of $\mathbb{Z}^n$ and positive integers $d_1, \dots, d_n$ such that $(d_1\boldsymbol{b}_1, \dots, d_n\boldsymbol{b}_n)$ is a basis of the lattice L generated by the columns of $\boldsymbol{\Pi}$.

Then, if we pick integers $x_1, \dots, x_n$ at random such that x_i is uniformly distributed in $\{0, \dots, d_i - 1\}$, then clearly, the vector $\boldsymbol{x} = x_1\boldsymbol{b}_1 + \cdots + x_n\boldsymbol{b}_n$ is uniformly distributed modulo L.

To get a uniformly distributed vector in the half-open parallelepiped defined by the $\boldsymbol{\varpi}_j$'s, it then suffices to apply Babai's round-off algorithm [Bab86], *i.e.* write $\boldsymbol{x}$ as a rational linear combination $\xi_1\boldsymbol{\varpi}_1 + \cdots + \xi_n\boldsymbol{\varpi}_n$ of the $\boldsymbol{\varpi}_j$'s and return the vector $\boldsymbol{y}$ given by:

$$\boldsymbol{y} = \sum_{j=1}^{n} \big(\xi_j - \lfloor \xi_j \rceil\big)\boldsymbol{\varpi}_j.$$

That vector is congruent to $\boldsymbol{x}$ modulo L, so it is also in $\mathbb{Z}^n$ and uniformly distributed modulo L, and it belongs to the parallelepiped by construction, so it is indeed a uniformly distributed element of the parallelepiped.

F Generation of the Matrix $\boldsymbol{H}$

For the construction of zero-testing parameters, we need to pick, with sufficient entropy, an invertible matrix $\boldsymbol{H} \in \mathbb{Z}^{n\times n}$ in such a way that both its operator norm and the norm of its inverse are not too large, namely $\|\boldsymbol{H}\|_\infty \leqslant 2^\beta$ and $\|\boldsymbol{H}^{-1}\|_\infty \leqslant 2^\beta$. In Section 3 the bounds must actually hold for $\boldsymbol{H}^T$, so we take the transpose of the resulting matrix.

For that purpose, we propose the following approach. For any matrix A of size $\lfloor n/2 \rfloor \times \lceil n/2 \rceil$ with coefficients in $\{-1, 0, 1\}$, define $\boldsymbol{H}_A \in \mathbb{Z}^{n\times n}$ as:

$$\boldsymbol{H}_A = \begin{pmatrix} I_{\lfloor n/2 \rfloor} & A \\ 0 & I_{\lceil n/2 \rceil} \end{pmatrix}.$$

Each line of $\boldsymbol{H}_A$ has at most $1 + \lceil n/2 \rceil$ non zero coefficients, each in $\{-1, 0, 1\}$, so we clearly have $\|\boldsymbol{H}_A\|_\infty \leqslant 1 + \lceil n/2 \rceil$. Moreover, $\boldsymbol{H}_A$ is invertible with $\boldsymbol{H}_A^{-1} = \boldsymbol{H}_{(-A)}$, so that the operator norm of its inverse admits a similar bound.

Similarly, the transpose $\boldsymbol{H}'_A$ of $\boldsymbol{H}_A$ also satisfies $\|\boldsymbol{H}'_A\|_\infty \leqslant 1 + \lceil n/2 \rceil$ (in fact, the slightly better bound by $1 + \lfloor n/2 \rfloor$ also holds) and so does its inverse.

Now let:

$$\beta' = \left\lfloor \frac{\beta}{\lceil \log_2(1 + \lceil n/2 \rceil) \rceil} \right\rfloor,$$

and generate β' uniformly random matrices A_i of size $\lfloor n/2 \rfloor \times \lceil n/2 \rceil$ with coefficients in $\{-1, 0, 1\}$; then pick $\boldsymbol{H}_i$ randomly as either $\boldsymbol{H}_{A_i}$ or its transpose for each $i \in \{1, \dots, \beta'\}$, and finally compute $\boldsymbol{H}$ as the product of the $\boldsymbol{H}_i$'s. Then, since operator norms are sub-multiplicative, we have:

$$\|\boldsymbol{H}\|_\infty \leqslant \prod_{i=1}^{\beta'} \|\boldsymbol{H}_i\|_\infty \leqslant (1 + \lceil n/2 \rceil)^{\beta'} \leqslant 2^\beta,$$

and $\boldsymbol{H}^{-1}$ satisfies the same bound. The set of matrices $\boldsymbol{H}$ obtained in this manner is not very simple to describe but it is exponentially large.

G Optimization on the Zero-Testing Elements

In order to test whether a level-κ encoding encodes $\mathbf{0}$, we publish as part of the instance generation a zero-testing vector $\boldsymbol{p}_{zt} \in \mathbb{Z}^n$. Unfortunately, this requires to store n integers of $n \cdot \eta$ bits, increasing the public-key size by $n^2 \cdot \eta$ bits. For example with $n = 26115$ and $\eta = 2438$ as in our "Extra" parameters set in Table 1, the zero-testing vector size would be larger than 200GB. Moreover, its construction relies on an intricate procedure to generate an invertible matrix $\boldsymbol{H} \in \mathbb{Z}^{n\times n}$ such that its operator norm, and the operator norm of its inverse are bounded by 2^β (see Appendix F).

In this section, we explain how to reduce the number of elements of $\boldsymbol{p}_{zt}$ from n to only two and simplify the requirement on $\boldsymbol{H}$. One drawback is that two encodings of different vectors can now extract to the same value. While it is actually easy to generate such collisions using LLL, this does not seem to give an attack against the GDDH problem; therefore one-round N-way Diffie-Hellman key exchange as described in Appendix D should remain secure under this optimization.

G.1 Zero-Testing Element

A first idea would be to publish a zero-testing element p_{zt} as a single integer, instead of a vector $\boldsymbol{p}_{zt}$ with n components:

$$p_{zt} = \sum_{i=1}^{n} h_i \cdot (z^\kappa \cdot g_i^{-1} \bmod p_i) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0$$

where the h_i's are random integers in $[1, 2^\beta)$. Therefore given as input an integer c such that $c \equiv (r_i \cdot g_i + m_i)/z^\kappa \pmod{p_i}$, we obtain a single integer $\omega = p_{zt} \cdot c \bmod x_0$, with

$$\omega = \sum_{i=1}^{n} h_i \cdot R_i \bmod x_0\,, \tag{20}$$

where $R_i = ((r_i + m_i \cdot g_i^{-1}) \bmod p_i) \cdot (x_0/p_i)$. As before, if $\|\boldsymbol{r}\|_\infty < 2^{\rho_f}$, we still have $|\omega| < x_0 \cdot 2^{-\nu-2}$. However the converse is no longer true: we can have $|\omega| < x_0 \cdot 2^{-\nu+2}$ for an encoding of a non-zero vector $\boldsymbol{m}$. However, if $m_i = 0$ for all $i > 1$, we have the following Lemma, whose proof is similar to the proof of Lemma 3:

Lemma 8. *Let n, η, α and β be as in our parameter setting. Let ρ_f be such that $\beta + \alpha + \rho_f + \log_2 n \leqslant \eta - 9$, and let $\nu = \eta - \beta - \rho_f - \log_2 n - 3 \geqslant \alpha + 6$. Let c be such that $c \equiv (r_i \cdot g_i + m_i)/z^\kappa \pmod{p_i}$ for all $1 \leqslant i \leqslant n$, where $0 \leqslant m_1 < g_1$ and $m_i = 0$ for all $i > 1$. Let $\boldsymbol{r} = (r_i)_{1\leqslant i\leqslant n}$ and assume that $\|\boldsymbol{r}\|_\infty < 2^{\rho_f}$. If $m_1 = 0$ then $|\omega| < x_0 \cdot 2^{-\nu-2}$. Conversely if $m_1 \neq 0$ then $|\omega| \geqslant x_0 \cdot 2^{-\nu+2}$.*

Proof. If $m_1 = 0$ then we have $R_i = r_i \cdot x_0/p_i$ for all i, which gives using $p_i \geqslant 2^{\eta-1}$ for all i:

$$\|\boldsymbol{R}\|_\infty \leqslant \|\boldsymbol{r}\|_\infty \cdot \max_{1\leqslant i\leqslant n} (x_0/p_i) \leqslant \|\boldsymbol{r}\|_\infty \cdot x_0 \cdot 2^{-\eta+1}\,.$$

Since by definition $-p/2 < (z \bmod p) \leqslant p/2$, we have $|z \bmod p| \leqslant |z|$ for any z, p; therefore we obtain from (20) using $\|\boldsymbol{r}\|_\infty < 2^{\rho_f}$

$$|\omega| = \|\boldsymbol{h}^t \cdot \boldsymbol{R} \bmod x_0\|_\infty \leqslant \|\boldsymbol{h}^t \cdot \boldsymbol{R}\|_\infty \leqslant n \cdot \|\boldsymbol{h}^t\|_\infty \cdot \|\boldsymbol{R}\|_\infty < x_0 \cdot 2^{\log_2 n + \beta + \rho_f - \eta + 1} = x_0 \cdot 2^{-\nu-2}\,.$$

Conversely assume that $|\omega| < x_0 \cdot 2^{-\nu+2}$. We have from (20):

$$h_1 \cdot R_1 \equiv \omega - \sum_{i=2}^{n} h_i \cdot R_i \pmod{x_0}\,.$$

Now, using the fact that $m_i = 0$ for $i > 1$, we have as previously

$$\left|\sum_{i=2}^{n} h_i \cdot R_i\right| \leqslant x_0 \cdot 2^{\log_2(n-1)+\beta+\rho_f-\eta+1} \leqslant x_0 \cdot 2^{-\nu-2}$$

Since $|h_1 \cdot R_1| < x_0/2$ and $|\omega - \sum_{i=2}^{n} h_i \cdot R_i| \leqslant |\omega| + |\sum_{i=2}^{n} h_i \cdot R_i| < x_0 \cdot 2^{-\nu+3}$, the previous equality must hold over $\mathbb{Z}$. This gives $|h_1 \cdot R_1| < x_0 \cdot 2^{-\nu+3}$, and since $h_1 \neq 0$, we get $|R_1| < x_0 \cdot 2^{-\nu+3}$. Letting $v_1 = (r_1 + m_1 \cdot g_1^{-1}) \bmod p_1$, we obtain $|v_1| \leqslant p_1 \cdot 2^{-\nu+3}$. We show that the equality $g_1 \cdot (v_1 - r_1) \equiv m_1 \pmod{p_1}$ must therefore hold over $\mathbb{Z}$. Indeed from $|m_1| < g_1 < p_1/2$ and $g_1 \leqslant 2^{\alpha}$:

$$|g_1 \cdot (v_1 - r_1)| \leqslant |g_1| \cdot (|v_1| + |r_1|) \leqslant p_1 \cdot 2^{\alpha-\nu+3} + 2^{\alpha+\rho_f} \leqslant p_1/8 + p_1/8 < p_1/2$$

which implies $m_1 \equiv 0 \pmod{p_1}$ and finally $m_1 = 0$. □

Thus if c and c' encode vectors differing only on their first element, by Lemma 8 we must have $|(c - c') \cdot p_{zt} \bmod x_0| > x_0 \cdot 2^{-\nu+2}$, and therefore the ν most significant bits of the corresponding ω and ω' must be different. This implies that the min-entropy of $\mathsf{msbs}_\nu(c \cdot p_{zt})$ when c encodes a message $(m_1, m_2, \ldots, m_n)$ for fixed m_i's, $i > 1$ and a random $m_1 \in \mathbb{Z}_{g_1}$ is at least $\log_2|\mathbb{Z}_{g_1}| \geqslant \alpha - 1$. Therefore, the min-entropy of $\mathsf{msbs}_\nu(c \cdot p_{zt})$ when c encodes a random message in R is at least $\alpha - 1$. Finally we can use a strong randomness extractor to extract a nearly-uniform bit-string of length $\alpha - 1 - \lambda$ bits. Thus to extract λ bits, we must have $\alpha \geqslant 2\lambda + 1$, instead of $\lambda = \alpha$ as recommended in Section 3.1. The previous bound is therefore not optimal, as a larger α increases the size η of the primes p_i and therefore the encoding size.

G.2 Extension to $t \leqslant n$ elements

We generalize the previous result to a zero-testing vector with t elements instead of one, namely $\boldsymbol{p}_{zt} \in \mathbb{Z}^t$ for $1 \leqslant t \leqslant n$:

$$(\boldsymbol{p}_{zt})_j = \sum_{i=1}^{n} h_{ij} \cdot (z^{\kappa} \cdot g_i^{-1} \bmod p_i) \cdot \prod_{i' \neq i} p_{i'} \bmod x_0 \,.$$

The matrix $\boldsymbol{H} = (h_{ij}) \in \mathbb{Z}^{n \times t}$ is randomly generated such that $\boldsymbol{H} = \begin{pmatrix} \boldsymbol{H_t} \\ \boldsymbol{H_{n-t}} \end{pmatrix}$ where the submatrix $\boldsymbol{H_t} \in \mathbb{Z}^{t \times t}$ is invertible in $\mathbb{Z}$ with both $\|\boldsymbol{H_t}^t\|_\infty \leqslant 2^{\beta}$ and $\|(\boldsymbol{H_t}^{-1})^t\|_\infty \leqslant 2^{\beta}$ (see Section 3 and Appendix F), and the coefficients of $\boldsymbol{H_{n-t}}$ are random β-bit integers. As previously, if $m_i = 0$ for all $i > t$, we have the following Lemma:

Lemma 9. *Let n, t, η, α and β be as in our parameter setting. Let ρ_f be such that $2\beta + \alpha + \rho_f + \log_2(n - t + 1) \leqslant \eta - 9$, and let $\nu = \eta - \beta - \rho_f - \log_2(n - t + 1) - 3 \geqslant \beta + \alpha + 6$. Let c be such that $c \equiv (r_i \cdot g_i + m_i)/z^{\kappa} \pmod{p_i}$ for all $1 \leqslant i \leqslant n$, where $0 \leqslant m_i < g_i$ for all $i \leqslant t$ and $m_i = 0$ for all $i > t$. Let $\mathbf{r} = (r_i)_{1 \leqslant i \leqslant n}$ and assume that $\|\boldsymbol{r}\|_\infty < 2^{\rho_f}$. If $m_i = 0$ for all i then $|\omega| < x_0 \cdot 2^{-\nu-2}$. Conversely if there exists $j \in [1, t]$ such that $m_j \neq 0$ then $|\omega| \geqslant x_0 \cdot 2^{-\nu+2}$.*

Proof. The proof of this Lemma is similar to the proofs of Lemmas 3 and 8. We sketch it for completeness. If $\boldsymbol{m} = 0$, we get as previously that $\|\boldsymbol{\omega}\|_\infty \leqslant x_0 \cdot 2^{-\nu-2}$. Assume now that $\|\boldsymbol{\omega}\|_\infty \leqslant x_0 \cdot 2^{-\nu+2}$, and denote $\boldsymbol{R}^t = (\boldsymbol{R_t}^t, \boldsymbol{R_{n-t}}^t)$. We have that

$$\boldsymbol{\omega} = \boldsymbol{H_t}^t \cdot \boldsymbol{R_t} + \boldsymbol{H_{n-t}}^t \cdot \boldsymbol{R_{n-t}} \,.$$

Now $\|\boldsymbol{H_{n-t}}^t \cdot \boldsymbol{R_{n-t}}\|_\infty \leqslant x_0 \cdot 2^{\log_2(n-t)+\beta-\eta+1+\rho_f} \leqslant x_0 \cdot 2^{-\nu-2}$ and

$$\boldsymbol{R_t} \equiv (\boldsymbol{H_t}^{-1})^t \cdot (\boldsymbol{\omega} - \boldsymbol{H_{n-t}}^t \cdot \boldsymbol{R_{n-t}}) \bmod x_0 \,,$$

and this latter equation holds over $\mathbb{Z}$. This yields $\|\boldsymbol{R_t}\|_\infty \leqslant x_0 \cdot 2^{\beta-\nu+3}$ and we conclude as in Lemma 3 that $\boldsymbol{m} = \boldsymbol{0}$. □

G.3 Two-element vector

Let us consider the case $t = 2$. In this case, we do not need to use the intricate generation procedure for $\boldsymbol{H_2}$ described in Appendix F. Indeed, $\boldsymbol{H_2} = (h_{ij})$ is invertible over $\mathbb{Z}$ if and only if $h_{11} \cdot h_{22} - h_{12} \cdot h_{21} = \epsilon \in \{\pm 1\}$ and its inverse is given by $\boldsymbol{H_2}^{-1} = \epsilon \begin{pmatrix} h_{22} & -h_{12} \\ -h_{21} & h_{22} \end{pmatrix}$. Therefore if $\|\boldsymbol{H_t}^t\|_\infty \leqslant 2^\beta$, then $\|(\boldsymbol{H_t}^{-1})^t\|_\infty \leqslant 2^\beta$.

Now, if c and c' encode vectors $\boldsymbol{m}$ and $\boldsymbol{m'}$ with $(m_1, m_2) \neq (m'_1, m'_2)$ and $m_i = m'_i$ for all $i > 2$, by Lemma 9 we must have $\|(c - c') \cdot \boldsymbol{p}_{zt} \bmod x_0\| > x_0 \cdot 2^{-\nu+2}$, and therefore the ν most significant bits of the corresponding $\boldsymbol{\omega}$ and $\boldsymbol{\omega'}$ must be different. This implies that the min-entropy of $\mathsf{msbs}_\nu(c \cdot \boldsymbol{p}_{zt})$ when c encodes a message $(m_1, m_2, \ldots, m_n)$ for fixed m_i's, $i > 2$ and a random tuple $(m_1, m_2) \in \mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2}$ is at least $\log_2 |\mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2}| \geqslant 2(\alpha - 1)$. Therefore, the min-entropy of $\mathsf{msbs}_\nu(c \cdot \boldsymbol{p}_{zt})$ when c encodes a random message in R is at least $2(\alpha - 1)$. Finally we can use a strong randomness extractor to extract a nearly-uniform bit-string of length $2(\alpha - 1) - \lambda$. Thus to extract λ bits, this implies to take $\alpha \geqslant \lambda + 1$ which is nearly optimal.