

On Kilian’s Randomization of Multilinear Map Encodings

Jean-Sébastien Coron and Hilder V. L. Pereira

University of Luxembourg

November 20, 2018

Abstract. Indistinguishability obfuscation constructions based on matrix branching programs generally proceed in two steps: first apply Kilian’s randomization of the matrix product computation, and then encode the matrices using a multilinear map scheme. In this paper we observe that by applying Kilian’s randomization *after* encoding, the complexity of the best attacks is significantly increased for CLT13. This implies that much smaller parameters can be used, which improves the efficiency of the constructions by several orders of magnitude.

As an application, we describe the first concrete implementation of non-interactive Diffie-Hellman key exchange secure against existing attacks. Key exchange was originally the most straightforward application of multilinear maps; however it was quickly broken for the three known families of multilinear maps (GGH13, CLT13 and GGH15). Here we describe the first implementation of key exchange based on CLT13 that is resistant against the Cheon et al. attack. For $N = 4$ users and a medium (62 bits) level of security, our implementation requires 8 GB of public parameters, and a few minutes for the derivation of a shared key. Without Kilian’s randomization of encodings our construction would be completely unpractical, as it would require more than 100 TB of public parameters.

1 Introduction

Multilinear maps and indistinguishability obfuscation. Since the breakthrough construction of Garg, Gentry and Halevi [GGH13a], cryptographic multilinear maps have shown amazingly powerful applications in cryptography, most notably the first plausible construction of program obfuscation [GGH⁺13b]. A multilinear map scheme encodes plaintext values $\{a_i\}$ into encodings $\{[a_i]\}$ such that the a_i ’s are hidden; only a restricted class of polynomials can then be evaluated over these encoded values; eventually one can determine whether the evaluation is zero or not, using the zero testing procedure of the multilinear map scheme.

The goal of program obfuscation is to hide secrets in arbitrary running programs. The first plausible construction of general program obfuscation was described by Garg, Gentry, Halevi, Raykova, Sahai and Waters (GGHRSW) in [GGH⁺13b], based on multilinear maps; the construction has opened many new research directions, because the notion of indistinguishability obfuscation (iO) has tremendous applications in cryptography [SW14]. Since the publication of the GGHRSW construction, many variants of GGHRSW have been described [MSW14, AGIS14, PST14, BGK⁺14, BMSZ16]. All constructions of program obfuscation rely on multilinear maps, for which there are essentially only three known candidate constructions:

- **GGH13.** The first candidate construction of multilinear maps is based on ideal lattices [GGH13a]. Its security relies on the difficulty of the NTRU problem and the principal ideal problem (PIP) in certain number fields.
- **CLT13.** An analogous construction but over the integers was described in [CLT13], based on the DGHV fully homomorphic encryption scheme [DGHV10].
- **GGH15.** Gentry, Gorbunov and Halevi described another multilinear maps scheme [GGH15], based on the Learning With Errors (LWE) problem with encoding over matrices, and defined with respect to a directed acyclic graph.

However the security of multilinear maps is still poorly understood. The most important attacks against multilinear maps are “zeroizing attacks”, which consist in using linear algebra to recover

the secrets of the scheme from the encodings of zero. At Eurocrypt 2015, Cheon *et al.* described a devastating zeroizing attack against CLT13; when CLT13 is used to implement non-interactive multipartite Diffie-Hellman key exchange, the attack completely breaks the protocol [CHL⁺15]. The attack was also extended to encodings variants, where encodings of zero are not directly available [CGH⁺15]. The key-exchange protocol based on GGH13 was also broken by a zeroizing attack in [HJ16]. Finally, the Diffie-Hellman key exchange protocol under GGH15 was broken in [CLLT16], using an extension of the Cheon *et al.* zeroizing attack.

However, not all attacks against the above multilinear map schemes can be applied to indistinguishability obfuscation. While multipartite key exchange based on any of the three families of multilinear map schemes is broken, iO is not necessarily broken by zeroizing attacks, because of the particular structure that iO constructions induce on the computation of multilinear map encoded values. Namely, in iO constructions, no low-level encodings of zeroes are available, and the obfuscation of a matrix branching program can only produce zeroes at the last level, moreover when evaluated in a very specific way. However some partial attacks against iO constructions have already been described. In [CGH⁺15] it was shown how to break the GGHRWS branching-program obfuscator when instantiated using CLT13, when the branching program to be obfuscated has a very simple structure (input partition). For GGH13, Miles, Sahai and Zhandry introduced “annihilation attacks” [MSZ16] that can break many obfuscation schemes based on GGH13; however, the attack does not apply to the GGHRWS construction, because in GGHRWS the matrix program is embedded in a larger matrix with random entries (diagonal padding). In [CGH17], the authors showed how to break iO constructions under GGH13, using a variant of the input partitioning attack; the attack applies against the GGHRWS construction with diagonal padding. A new tensoring technique was introduced in [CLLT17] to break iO constructions for branching programs without the input partition structure. Finally, an attack against iO over GGH15 was described in [CVW18] based on computing the rank of a certain matrix.

Obfuscating matrix branching programs. The GGHRWS construction and its variants consist of a “core component” for obfuscating matrix branching programs, and a bootstrapping procedure to obfuscate arbitrary programs based on the core component, using fully homomorphic encryption and proofs of correct computation. The core component relies on multilinear maps for evaluating a product of encoded matrices corresponding to a branching program, without revealing the underlying value of those matrices.

More precisely, the core component of the GGHRWS construction and its variants proceeds in two steps: first apply Kilian’s randomization of the matrix product computation, and then encode the matrices using a multilinear map scheme. In this paper, our main observation is that for CLT13 multilinear maps, the complexity of the best attacks is significantly increased when Kilian’s randomization is also applied *after* encoding. We note that applying Kilian’s randomization “on the encoding side” was already used in GGH15 multilinear maps as an additional safeguard [GGH15, §5.1]. For CLT13 this implies that one can use much smaller parameters (noise and encoding size), which improves the efficiency of the constructions by several orders of magnitude.

More precisely, a matrix branching program BP of length n is evaluated on input $x \in \{0, 1\}^\ell$ by computing:

$$C(x) = \mathbf{b}_0 \times \prod_{i=1}^n \mathbf{B}_{i, x_{\text{inp}(i)}} \times \mathbf{b}_{n+1} \quad (1)$$

where $\{\mathbf{B}_{i,b}\}_{1 \leq i \leq n, b \in \{0,1\}}$ are square matrices and \mathbf{b}_0 and \mathbf{b}_{n+1} are bookend vectors; then $\text{BP}(x) = 0$ if $C(x) = 0$, and $\text{BP}(x) = 1$ otherwise. The function $\text{inp}(i)$ indicates which bit of x is read at step i of the product matrix computation. To obfuscate a matrix branching program, the GGHRWS construction proceeds in two steps. First one randomizes the matrices $\mathbf{B}_{i,b}$ as in Kilian’s protocol [Kil88]: choose $n + 1$ random invertible matrices $\{\mathbf{R}_i\}_{i=0}^n$ and set $\tilde{\mathbf{B}}_{i,b} = \mathbf{R}_{i-1} \mathbf{B}_{i,b} \mathbf{R}_i^{-1}$, with also $\tilde{\mathbf{b}}_0 = \mathbf{b}_0 \mathbf{R}_0^{-1}$ and $\tilde{\mathbf{b}}_{n+1} = \mathbf{R}_n \mathbf{b}_{n+1}$. The randomized matrix branching program can then be evaluated

by computing

$$C(x) = \tilde{\mathbf{b}}_0 \times \prod_{i=1}^n \tilde{\mathbf{B}}_{i, x_{\text{inp}(i)}} \times \tilde{\mathbf{b}}_{n+1}.$$

Namely the successive randomization matrices \mathbf{R}_i cancel each other; therefore the matrix product computation evaluates to the same result as in (1).

The second step in the GGHRSW construction is to encode the entries of the matrices $\tilde{\mathbf{B}}_{i,b}$ using a multilinear map scheme. Every entry of a given matrix is encoded separately; the bookend vectors $\tilde{\mathbf{b}}_0$ and $\tilde{\mathbf{b}}_n$ are also encoded similarly. Therefore one defines the matrices and vectors

$$\hat{\mathbf{B}}_{i,b} = \text{Encode}_{\{i+1\}}(\tilde{\mathbf{B}}_{i,b}), \quad \hat{\mathbf{b}}_0 = \text{Encode}_{\{1\}}(\tilde{\mathbf{b}}_0), \quad \hat{\mathbf{b}}_n = \text{Encode}_{\{n+2\}}(\tilde{\mathbf{b}}_{n+2}).$$

The matrix branching program from (1) can then be evaluated over the encoded matrices:

$$\hat{C}(x) = \hat{\mathbf{b}}_0 \times \prod_{i=1}^n \hat{\mathbf{B}}_{i, x_{\text{inp}(i)}} \times \hat{\mathbf{b}}_{n+1} \quad (2)$$

Eventually one obtains an encoded $\hat{C}(x)$ over the universe set $S = \{1, \dots, n+2\}$, and one can use the zero-testing procedure of the multilinear map scheme to check if $C(x) = 0$, thereby learning the output of the branching program $\text{BP}(x)$, without revealing the values of the matrices $\mathbf{B}_{i,b}$. It was shown in [BGK⁺14] that if the multilinear map scheme is ideal, *i.e.* if the multilinear map only reveals whether or not the evaluation is zero and does not leak anything else, then the above obfuscation scheme is secure.

(In)efficiency of iO. However, even with some efficiency improvements (as in [AGIS14]), the main issue is that indistinguishability obfuscation is currently not feasible to implement in practice. The first obstacle is that when converting the input circuit to a matrix branching program using Barrington’s theorem [Bar86], one induces an enormous cost in performance, as the length of the branching program grows exponentially with the depth of the circuit being evaluated. The second obstacle is that the multilinear map noise and parameters grow with the degree of the polynomial being computed over encoded elements, which corresponds to the length of the matrix branching program.

In this paper, we consider both issues. For the second issue, we show that for CLT13 multilinear maps, when applying Kilian’s randomization “on the encoding side”, one can significantly reduce the noise and encoding size for the same level of security, which can lead to major improvements of performance. For the first issue, we consider a matrix branching program that only performs a multipartite DH key-exchange, rather than originating from a circuit through Barrington’s theorem, so that its degree becomes much more manageable. Thanks to Kilian’s randomization of the encodings, we can then describe the first concrete implementation of DH key-exchange based on multilinear maps that is resistant against existing attacks.

First contribution: security of Kilian’s randomization on the encoding side. As already observed in [GGH15], Kilian’s randomization can also be applied over the encoding space, as an additional safeguard. Namely starting from the encoded matrices $\hat{\mathbf{B}}_{i,b}$ used to compute $\hat{C}(x)$ as in Equation (2), one can again choose $n+1$ random invertible matrices $\{\hat{\mathbf{R}}_i\}_{i=0}^n$ and then randomize the matrices $\hat{\mathbf{B}}_{i,b}$ with:

$$\bar{\mathbf{B}}_{i,b} = \hat{\mathbf{R}}_{i-1} \hat{\mathbf{B}}_{i,b} \hat{\mathbf{R}}_i^{-1}$$

with also $\bar{\mathbf{b}}_0 = \hat{\mathbf{b}}_0 \hat{\mathbf{R}}_0^{-1}$ and $\bar{\mathbf{b}}_{n+1} = \hat{\mathbf{R}}_n \hat{\mathbf{b}}_{n+1}$. Since the matrices $\hat{\mathbf{R}}_i$ cancel each other in the matrix product computation, the evaluation proceeds exactly as in (2), with

$$\hat{C}(x) = \bar{\mathbf{b}}_0 \times \prod_{i=1}^n \bar{\mathbf{B}}_{i, x_{\text{inp}(i)}} \times \bar{\mathbf{b}}_{n+1},$$

and therefore the same zero-testing procedure can be applied to $\hat{C}(x)$. Note that the $\hat{\mathbf{R}}_i$ matrices are applied on the encoding side, that is on the encoded matrices $\hat{\mathbf{B}}_{i,b}$, instead of the plaintext matrices $\mathbf{B}_{i,b}$ as previously; obviously both randomizations (before and after encoding) can be applied independently.

In this paper we focus on Kilian’s randomization on the encoding side in the context of the CLT13 multilinear maps. In CLT13 the encoding space is the set of integers modulo x_0 , where $x_0 = \prod_{j=1}^n p_j$; therefore the matrices $\{\hat{\mathbf{R}}_i\}_{i=0}^n$ are random invertible matrices modulo x_0 . Our first contribution is to show that the complexity of the best attacks against CLT13 is significantly increased thanks to Kilian’s randomization of the encodings. One can therefore use much smaller parameters (noise size and encoding size), which can considerably improve the efficiency of a construction.

More precisely, the security of CLT13 is based on the hardness of the Approximate-GCD problem. Given $x_0 = q \cdot p$ for some prime p and polynomially-many encodings

$$c_i = q_i \cdot p + r_i \quad (3)$$

for small integers r_i ’s of ρ bits, the goal is to recover the secret prime p . For simplicity we first consider a single prime p only, and we later consider the generalization to multiple primes p_i ’s as in CLT13. The two main approaches for solving the Approximate-GCD problem are the GCD attacks and the orthogonal lattice attacks.

GCD attacks. The naive GCD attack consists in computing $\gcd(c_1 - r_1, x_0)$ for all possible r_1 and has complexity $\mathcal{O}(2^\rho)$. At Eurocrypt 2012, Chen and Nguyen [CN12] described an improved attack based on multipoint polynomial evaluation, with complexity $\tilde{\mathcal{O}}(2^{\rho/2})$. The Chen-Nguyen attack was later extended by Lee and Seo at Crypto 2014 [LS14], when the c_i ’s are multiplicatively masked by a random secret z modulo x_0 , as it is the case in the CLT13 scheme; their attack has the same complexity $\tilde{\mathcal{O}}(2^{\rho/2})$.

In the context of branching programs with matrices of dimension m , instead of considering a set of scalar values c_i as in (3), we must consider a set of row vectors \mathbf{v}_i of dimension m , where all m components $(\mathbf{v}_i)_j$ of \mathbf{v}_i are small modulo p :

$$(\mathbf{v}_i)_j = r_{ij} \pmod{p}$$

Without Kilian’s randomization, the vectors \mathbf{v}_i would be published directly (up to some random multiplicative mask z in CLT13), and the Lee-Seo attack could be applied directly to any of the components of \mathbf{v}_i . However, with Kilian’s randomization, one only obtains the randomized vectors:

$$\tilde{\mathbf{v}}_i = \mathbf{v}_i \cdot \mathbf{K} \pmod{x_0}$$

for some secret random invertible matrix \mathbf{K} modulo x_0 ; the matrix \mathbf{K} corresponds for example to the matrix $\hat{\mathbf{R}}_0^{-1}$ used to randomize the encoded bookend vector $\hat{\mathbf{b}}_0$. The goal is still to recover the secret prime p from the randomized vectors $\tilde{\mathbf{v}}_i$; we call this the Vector Approximate-GCD problem.

In this paper we extend the Lee-Seo attack to the Vector Approximate-GCD problem, and we obtain a complexity $\tilde{\mathcal{O}}(2^{m \cdot \rho/2})$ instead of $\tilde{\mathcal{O}}(2^{\rho/2})$, where m is the vector dimension. Assuming that this is the best possible attack, thanks to Kilian’s randomization one can therefore divide the noise size ρ by a factor m . Similarly, when Kilian’s randomization is applied to a $m \times m$ matrix, we show that the attack complexity becomes $\tilde{\mathcal{O}}(2^{m^2 \cdot \rho/2})$, and therefore the noise size ρ used to encode those matrices in CLT13 can be divided by m^2 . In both cases this is a significant improvement compared to the original CLT13.

Lattice attacks. The classical orthogonal lattice attack against the single-prime Approximate GCD-problem has complexity $2^{\Omega(\gamma/\eta^2)}$, where γ is the size of x_0 and η is the size of the prime p ;

see [DGHV10, §5.2]. However, extending the attack to the multiprime case is actually not straightforward; in particular, we argue that the approach described in [CLT13] is incomplete and does not recover the primes p_i 's, except for small values of n ; solving the multiprime variant was actually considered as an open problem in [GGM16].

In this paper, we solve this open problem with an attack that proceeds in two steps. The first step is the classical orthogonal lattice attack; it recovers a basis of the lattice generated by the vectors $\mathbf{r}_i = \mathbf{c} \bmod p_i$, where $\mathbf{c} = (c_1, \dots, c_t)$. However, the vectors \mathbf{r}_i cannot necessarily be recovered directly (except possibly in low dimension with LLL). Therefore, in the second step, using the lattice basis obtained from the first step, we apply a variant of the Cheon *et al.* attack [CHL⁺15]; we show that by computing the eigenvalues of a well chosen matrix, we can recover the primes p_i 's. The asymptotic complexity of the attack is the same as in the single-prime case; using $\gamma = \eta \cdot n$ for the size of x_0 as previously, where n is the number of primes p_i , the complexity is then $2^{\Omega(n/\eta)}$. Therefore, as in [CLT13], one must take $n = \omega(\eta \log \lambda)$ to prevent the lattice attack, where λ is the security parameter.

In this paper we then extend the orthogonal lattice attack to the Vector Approximate-GCD problem, and we show that the extended attack has complexity $2^{\Omega(m \cdot \gamma / \eta^2)}$. This implies that the new condition on the number n of primes p_i in CLT13 becomes:

$$n = \omega\left(\frac{\eta}{m} \log \lambda\right)$$

Assuming again that this is the best possible attack, thanks to Kilian's randomization on the encoding side, the number of primes n in CLT13 can therefore be divided by a factor m , for the same level of security, where m is the matrix dimension. This implies that the encoding size γ can also be divided by m . In Section 7, we show that by combining with the previous improvement on the noise ρ and with some additional optimizations, the encoding size γ can be divided by a factor m^3 . Even for moderate values of m , this improves the efficiency of the constructions by several orders of magnitude. Note however that one cannot decrease γ too much, as in CLT13 the size η of the primes p_i is lower-bounded from the ECM factoring attack (see Section 7).

Second contribution: non-interactive DH key exchange from multilinear maps. In principle the most straightforward application of multilinear maps is non-interactive multipartite Diffie-Hellman (DH) key exchange with N users, a natural generalization of the DH protocol for 3 users based on the bilinear pairing. Such application was originally described for GGH13, CLT13 and GGH15; however it was broken for the three families of multilinear maps; in particular, key exchange based on CLT13 was broken by the Cheon *et al.* attack [CHL⁺15]. The main question is therefore: can we construct a practical N -way non-interactive key-exchange protocol from multilinear maps?

In this paper we provide a first step in that direction. Namely our second contribution is to describe the first implementation of DH key exchange based on CLT13 that is resistant against the Cheon *et al.* attack and its variants. For $N = 4$ users and a medium (62 bits) level of security, our implementation requires 8 GB of public parameters, and a few minutes for the derivation of a shared key. We note that without Kilian's randomization of encodings our construction would be completely unpractical, as it would require more than 100 TB of public parameters.

Our construction contains many ingredients from the GGHRWS construction and its variants. Namely we express the session key as the result of a matrix product computation, and we embed the matrices into larger randomized matrices before encoding, together with some special "bookend" components at the start and end of the computation, as in [GGH⁺13b]. We use the "multiplicative bundling" technique from [GGH⁺13b] to prevent the adversary from combining the matrices in arbitrary ways. We also use the straddling set systems from [BGK⁺14] to further constrain the attacker. As opposed to [GGH⁺13b] we don't apply Kilian's randomization at the plaintext level, but we use Kilian's randomization on the encoding side. Finally, we use k repetitions in order to

prevent the Cheon *et al.* attack against CLT13, when considering input partitioning attacks as in [CGH⁺15], and its extension with the tensoring attack [CLLT17]. We argue that the extended Cheon *et al.* attack has complexity $\Omega(m^{2k-1})$ in our scheme, where m is the matrix dimension and k the number of repetitions.

We now describe our construction in more details. Let N be the number of users and \mathbb{Z}_g be the plaintext space; when using [CLT13], we have $g := \prod_{i=1}^n g_i$ for small primes g_i 's. For each user $1 \leq u \leq N$, we consider a set of $m \times m$ matrices $\mathbf{A}_{i,b}^{(u)}$ in \mathbb{Z}_g , for $1 \leq i \leq \ell$ and $b \in \{0, 1\}$. We also let $\mathbf{s}^{(u)}$ and $\mathbf{t}^{(u)}$ be bookend vectors of dimension m . Each user will compute a session-key that is a function of the following scalar value:

$$v_u = \mathbf{s}^{(u)} \cdot \left(\prod_{i=1}^{\ell} \mathbf{A}_{i,b_i}^{(u)} \right) \cdot \mathbf{t}^{(u)} \quad (4)$$

where the bits b_i are obtained by concatenating the secret-keys $\mathbf{sk}^{(u)}$ of the N users, moreover with k repetitions; more precisely,

$$(b_i)_{1 \leq i \leq \ell} = \underbrace{(\mathbf{sk}^{(1)}, \dots, \mathbf{sk}^{(N)})}_{\text{First repetition}}, \dots, \underbrace{\mathbf{sk}^{(1)}, \dots, \mathbf{sk}^{(N)}}_{k\text{-th repetition}} \quad (5)$$

To ensure that all users $1 \leq u \leq N$ compute the same session-key, we define $\mathbf{A}_{i,b}^{(u)}$ as a larger matrix embedding a matrix $\mathbf{B}_{i,b}$ that is the same for all users, with some random block padding in the diagonal and the multiplicative bundling scalars $\alpha_{i,b}^{(u)}$ to prevent the adversary from switching the corresponding bits b_i 's between the k repetitions of the secret keys. More precisely, we let:

$$\mathbf{A}_{i,b}^{(u)} \sim \begin{pmatrix} \$ & \dots & \$ & & & \\ \vdots & \ddots & \vdots & & & \\ \$ & \dots & \$ & & & \\ & & & \$ & \dots & \$ \\ & & & \vdots & \ddots & \vdots \\ & & & \$ & \dots & \$ \\ & & & & & \alpha_{i,b}^{(u)} \cdot \mathbf{B}_{i,b} \end{pmatrix} \quad (6)$$

As in [GGH⁺13b], the bookend vectors are defined in such a way that the contribution from the random diagonal blocks is canceled:

$$\mathbf{s}^{(u)} \sim (0, \dots, 0, \$, \dots, \$, \mathbf{s}^*), \quad \mathbf{t}^{(u)} \sim (\$, \dots, \$, 0, \dots, 0, \mathbf{t}^*)$$

We denote by $\tilde{\mathbf{A}}_{i,b}^{(u)}$ the matrix encoding of $\mathbf{A}_{i,b}^{(u)}$, where the matrix is encoded under CLT13 elementwise. We use Kilian's randomization at the encoding level, that is, we publish the matrices:

$$\mathbf{C}_{i,b}^{(u)} = \mathbf{K}_{i-1}^{(u)} \cdot \tilde{\mathbf{A}}_{i,b}^{(u)} \cdot \left(\mathbf{K}_i^{(u)} \right)^{-1} \pmod{x_0}$$

Finally, each user u computes and publishes an encoding of all partial matrix products of the $\mathbf{C}_{i,b}^{(u)}$ corresponding to his $\mathbf{sk}^{(u)}$ in the k repetitions from (5), for all rows $v \neq u$. This ensures that from the extraction procedure of the multilinear map scheme, each user u can derive the session key from his own $\mathbf{sk}^{(u)}$ by computing on his row u the partial products corresponding to his $\mathbf{sk}^{(u)}$, combined with the published partial matrix products from the other users.

Related work. In [MZ18], Ma and Zhandry describe a multilinear map scheme built on top of CLT13 that is provably resistant against zeroizing attack, and which can be used to directly construct a non-interactive DH key-exchange. More precisely, the authors develop a new weak multilinear map model for CLT13 to capture all known attack strategies against CLT13; their model is different from the weak multilinear model developed in [MSZ16] for GGH13. In particular this enables to demonstrate that the obfuscation constructions from [BMSZ16] and [BGK⁺14] are secure against CLT13 zeroizing attacks. The authors then construct a new multilinear map scheme on top of CLT13 that is secure in this model. The construction is based on multiplying matrices of CLT13 encodings as in iO schemes. To prevent zeroizing attacks, the same input is read multiple times, as in iO constructions. The input consistency is ensured by a clever use of “enforcing” matrices based on some permutation invariant property. Finally, the authors construct a non-interactive DH key-exchange scheme based on their new multilinear map scheme. However, the authors do not provide implementation results nor concrete parameters (except for multilinear map degree and number of public encodings), so it is difficult to assess the practicality of the construction. In any case, the Ma-Zhandry construction can certainly benefit from our analysis, since Kilian’s randomization on the encoding side can also be applied “for free” in their construction.

2 Preliminaries

We denote by $[a]_n$ or $a \bmod n$ the unique integer $x \in (-\frac{n}{2}, \frac{n}{2}]$ which is congruent to a modulo n . The set $\{1, 2, \dots, n\}$ is denoted by $[n]$.

2.1 The CLT13 multilinear map

We briefly recall the (asymmetric) CLT13 multilinear map scheme; we refer to [CLT13] for a full description. For large secret primes p_i ’s, let $x_0 = \prod_{k=1}^n p_i$, where n is the number of primes. We denote by η the bitsize of the p_i ’s, and by γ the bitsize of x_0 ; therefore $\gamma \simeq n \cdot \eta$. The plaintext space of CLT13 is $\mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2} \times \dots \times \mathbb{Z}_{g_n}$ for secret prime integers g_i ’s of α bits.

The CLT13 scheme is based on CRT representations. We denote by $\text{CRT}(a_1, \dots, a_n)$ or $\text{CRT}(a_i)_i$ the number $a \in \mathbb{Z}_{x_0}$ such that $a \equiv a_i \pmod{p_i}$ for all $i \in [n]$. An encoding of a vector $\mathbf{m} = (m_1, \dots, m_n)$ at level set $S = \{j\}$ is an integer $c \in \mathbb{Z}_{x_0}$ such that $c = [\text{CRT}(m_1 + g_1 r_1, \dots, m_n + g_n r_n) / z_j]_{x_0}$ for integers r_i of size ρ bits, where z_j is a secret mask in \mathbb{Z}_{x_0} uniformly chosen during the parameters generation procedure of the multilinear map. This gives:

$$c \equiv \frac{m_i + g_i r_i}{z_j} \pmod{p_i}$$

for all $1 \leq i \leq n$. To support a ℓ -level multilinearity, one uses ℓ distinct z_j ’s.

It is clear that encodings from the same level can be added via addition modulo x_0 . Similarly multiplication between encodings can be done by modular multiplication in \mathbb{Z}_{x_0} , but the encodings must be of disjoint level sets; the resulting encoding level set is then the union of the input level sets. At the top level set $S = \{1, \dots, \ell\}$, one can zero-test an encoding by multiplication with the zero-test parameter

$$p_{zt} = \left(\prod_{j=1}^{\ell} z_j \right) \cdot \text{CRT}(p_i^* h_i g_i^{-1})_i \bmod x_0,$$

where $p_i^* = x_0/p_i$ and the h_i ’s are random β -bit integers. Namely given a top-level encoding c with

$$c = \frac{\text{CRT}(m_i + g_i r_i)_i}{\prod_{j=1}^{\ell} z_j} \bmod x_0,$$

we obtain after multiplication by p_{zt} :

$$c \cdot p_{zt} = \text{CRT}(h_i p_i^* (m_i g_i^{-1} + r_i))_i = \sum_{i=1}^n h_i p_i^* (m_i g_i^{-1} + r_i) \pmod{x_0}. \quad (7)$$

and therefore if $m_i = 0$ for all $1 \leq i \leq n$ then the result will be small compared to x_0 . From the previous equation the high-order bits of $c \cdot p_{zt} \pmod{x_0}$ only depend on the m_i 's; therefore from the zero-testing procedure one can extract a value that only depends on the m_i 's.

2.2 The Approximate-GCD Problem and its Variant

The security of the CLT13 multilinear map scheme is based on the Approximate-GCD problem. For a specific η -bit prime integer p , we use the following distribution over γ -bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) = \left\{ \text{Choose } q \leftarrow \mathbb{Z} \cap [0, 2^\gamma/p), r \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho) : \text{Output } x = q \cdot p + r \right\}$$

We also consider a noise-free $x_0 = q_0 \cdot p$ where q_0 is a random $(\gamma - \eta)$ -bit prime integer (alternatively the product of $\gamma/\eta - 1$ primes of size η bits each).

Definition 1 (Approximate-GCD problem with noise-free x_0). For a random η -bit prime integer p , given $x_0 = q_0 \cdot p$ and polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$, output p .

We also consider the following variant, in which instead of being given elements from $\mathcal{D}_{\gamma,\rho}(p)$, we get vectors of elements multiplied by a random invertible matrix \mathbf{K} modulo x_0 .

Definition 2 (Vector Approximate-GCD problem with noise-free x_0). For a random η -bit prime integer p , generate $x_0 = q_0 \cdot p$ and a random invertible $m \times m$ matrix \mathbf{K} modulo x_0 . Given x_0 and polynomially many samples $\tilde{\mathbf{v}} = \mathbf{v} \cdot \mathbf{K} \pmod{x_0}$ where $\mathbf{v} \leftarrow (\mathcal{D}_{\gamma,\rho}(p))^m$, output p .

The vector variant of the Approximate-GCD problem cannot be easier than the original problem, since any algorithm solving the vector variant can be used to solve the Approximate-GCD problem, simply by generating vectors $\tilde{\mathbf{v}} = \mathbf{v} \cdot \mathbf{K} \pmod{x_0}$ for some random matrix \mathbf{K} . However, the vector variant could be harder to solve, so that smaller parameters could be used when dealing with the Vector Approximate-GCD problem. We show in the next section that this is indeed the case.

In the context of the CLT13 scheme, one actually works with multiple primes p_i 's. Therefore we consider the multiprime variant of the Approximate-GCD problem.

Definition 3 (Multi-prime Approximate-GCD problem). For n random η -bit prime integers p_i , let $x_0 = \prod_{i=1}^n p_i$. Given x_0 and polynomially many integers $c_j = \text{CRT}(r_{ij})_i$ where $r_{ij} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$, output the primes p_i .

Finally, we consider the vector variant of the multi-prime Approximate-GCD problem.

Definition 4 (Vector multi-prime Approximate-GCD problem). For n random η -bit prime integers p_i , let $x_0 = \prod_{i=1}^n p_i$. Let \mathbf{K} be a random invertible $m \times m$ matrix modulo x_0 . Given x_0 and polynomially many vectors $\tilde{\mathbf{v}} = \mathbf{v} \cdot \mathbf{K} \pmod{x_0}$, where $\mathbf{v} = (v_1, \dots, v_m)$ and $v_j = \text{CRT}(r_{ij})_i$ where $r_{ij} \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)$, output the primes p_i .

The two main approaches for solving the Approximate-GCD problem are the GCD attacks and the orthogonal lattice attacks. For both approaches we first recall the known algorithms, and we then describe our extension to the vector variant (Definition 2). We consider the GCD attacks in Section 3, and the orthogonal lattice attacks in Section 4.

3 GCD Attacks against the Approximate-GCD Problem and its Variants

3.1 The Naive GCD Attack

We consider $x_0 = q \cdot p$ and an encoding c with

$$c \equiv r \pmod{p},$$

where r is a small integer of size ρ bits. The naive GCD attack, which has complexity $\mathcal{O}(2^\rho)$, consists in performing an exhaustive search of r and computing $\gcd(c - r, x_0)$ to obtain the factor p .

3.2 The Chen-Nguyen Attack

At Eurocrypt 2012, Chen and Nguyen described an improved attack based on multipoint polynomial evaluation [CN12], with complexity $\tilde{\mathcal{O}}(2^{\rho/2})$. One starts from the equation:

$$p = \gcd \left(x_0, \prod_{i=0}^{2^{\rho/2}-1} (c - i) \pmod{x_0} \right) \quad (8)$$

The main observation is that the above product modulo x_0 can be written as the product of $2^{\rho/2}$ evaluations of a single polynomial of degree $2^{\rho/2}$. Using a tree structure, it is possible to evaluate a polynomial of degree $2^{\rho/2}$ at $2^{\rho/2}$ points in $\tilde{\mathcal{O}}(2^{\rho/2})$ time and memory, instead of $\mathcal{O}(2^\rho)$.

More precisely, one can define the following polynomial $f(x)$ of degree $2^{\rho/2}$, with coefficients modulo x_0 ; we assume for simplicity that ρ is even:

$$f(x) = \prod_{i=0}^{2^{\rho/2}-1} (c - (x + i)) \pmod{x_0}$$

One can then rewrite (8) as the product of $2^{\rho/2}$ evaluations of the polynomial $f(x)$:

$$p = \gcd \left(x_0, \prod_{k=0}^{2^{\rho/2}-1} f(2^{\rho/2}k) \pmod{x_0} \right)$$

There are classical algorithms which can evaluate a polynomial $f(x)$ of degree d at d points, using at most $\tilde{\mathcal{O}}(d)$ operations in the coefficient ring; see for example [Ber03]. The technique is as follows. First, one must compute the coefficients of the polynomial $f(x)$; using a product tree, the product of the $d = 2^{\rho/2}$ factors can be computed in time $\tilde{\mathcal{O}}(d)$. Secondly, one must compute the evaluation of $f(x)$ at d points x_1, \dots, x_d . This can also be performed in time $\tilde{\mathcal{O}}(d)$ using a remainder tree. The basic observation is that the evaluation of $f(x)$ at the first half $x_1, \dots, x_{d/2}$ is equal to the evaluation of the degree $d/2$ polynomial $f_l(x) = f(x) \pmod{(x - x_1) \cdots (x - x_{d/2})}$ on $x_1, \dots, x_{d/2}$. Therefore the evaluation of $f(x)$ can proceed with a recursive algorithm. First compute the left polynomial $f_l(x) = f(x) \pmod{(x - x_1) \cdots (x - x_{d/2})}$; the computation of $(x - x_1) \cdots (x - x_{d/2})$ can be done in time $\tilde{\mathcal{O}}(d)$ with a product tree, and the remainder can also be computed in time $\tilde{\mathcal{O}}(d)$. Proceed similarly with the right polynomial $f_r(x) = f(x) \pmod{(x - x_{d/2+1}) \cdots (x - x_d)}$. Then recursively evaluate $f_l(x)$ and $f_r(x)$ on the two halves with $d/2$ points each. It is easy to see that the full algorithm has time and memory complexity $\tilde{\mathcal{O}}(d)$. Therefore, the Chen-Nguyen Attack has time and memory complexity $\tilde{\mathcal{O}}(2^{\rho/2})$.

We provide in Appendix C.1 an implementation of the Chen-Nguyen attack in Sage; our running time is similar to [CN12, Table 1]. In practice, the running time in number of clock cycles of the Chen-Nguyen attack with a γ -bit x_0 is well approximated by:

$$T_{CN}(\rho, \gamma) = 0.3 \cdot \rho^2 \cdot 2^{\rho/2} \cdot \gamma \cdot \log^2 \gamma \quad (9)$$

3.3 The Lee-Seo Attack

The Chen-Nguyen attack was later extended by Lee and Seo at Crypto 2014 [LS14], when the encodings are multiplicatively masked by a random secret z modulo x_0 , as it is the case in the CLT13 scheme; their attack has the same complexity $\tilde{\mathcal{O}}(2^{\rho/2})$. Namely in the asymmetric CLT13 scheme recalled in Section 2.1, an encoding c at level set $\{i_0\}$ is such that:

$$c \equiv \frac{r_i \cdot g_i + m_i}{z_{i_0}} \pmod{p_i}$$

for some random secret z_{i_0} modulo x_0 . Therefore, we consider the following variant of the Approximate-GCD problem. Instead of being given encodings c_i with $c_i \equiv r_i \pmod{p}$ for small r_i 's, we are given encodings c_i with:

$$c_i \equiv r_i \cdot z \pmod{p}$$

for some random integer z modulo x_0 , where the r_i 's are still ρ -bit integers. Since $c_1/c_2 \equiv r_1/r_2 \pmod{p}$, the naive GCD attack consists in guessing r_1 and r_2 and computing $p = \gcd(c_1/c_2 - r_1/r_2 \pmod{x_0}, x_0)$, with complexity $\mathcal{O}(2^{2\rho})$.

The Lee-Seo attack with complexity $\tilde{\mathcal{O}}(2^{\rho/2})$ is as follows. First, one generates two lists L_1 and L_2 of such encodings, and we look for a collision modulo p between those two lists; such collision will appear with good probability when the size of the two lists is at least $2^{\rho/2}$. More precisely, let c_i be the elements of L_1 and d_j be the elements of L_2 , with $c_i \equiv r_i \cdot z \pmod{p}$ and $d_j = s_j \cdot z \pmod{p}$. If $r_i = s_j$ for some pair (i, j) , then $c_i \equiv d_j \pmod{p}$ and therefore:

$$p = \gcd \left(\prod_{i,j} (c_i - d_j) \pmod{x_0, x_0} \right)$$

where the product is over all $c_i \in L_1$ and $d_j \in L_2$. A naive computation of this product would take time $|L_1| \cdot |L_2| = 2^\rho$; however, as in the Chen-Nguyen attack, this product can be computed in time and memory $\tilde{\mathcal{O}}(2^{\rho/2})$. Namely one can define the polynomial

$$f(x) = \prod_i (c_i - x) \pmod{x_0}$$

of degree $|L_1| = 2^{\rho/2}$ and the previous equation can be rewritten:

$$p = \gcd \left(\prod_j f(d_j) \pmod{x_0, x_0} \right)$$

This corresponds to the multipoint evaluation of the degree $2^{\rho/2}$ polynomial $f(x)$ at the $2^{\rho/2}$ points of the list L_2 ; therefore, this can be computed in time and memory $\tilde{\mathcal{O}}(2^{\rho/2})$.

As observed in [LS14], if only a small set of elements c_i is available (much less than $2^{\rho/2}$), one can still generate exponentially more c_i 's by using small linear integer combinations of the original c_i 's, and the above attack still applies, with only a slight increase in the noise ρ . We provide in Appendix C.2 an implementation of the Lee-Seo attack in Sage. Its running time is the same as Chen-Nguyen, except that the attack is probabilistic only; its success probability can be increased by taking slightly larger lists L_1 and L_2 to improve the collision probability.

3.4 GCD Attack against the Vector Approximate GCD Problem

We now consider the Vector Approximate-GCD problem (Definition 2). We consider a set of row vectors \mathbf{v}_i of dimension m , such that for each vector \mathbf{v}_i , all components $(\mathbf{v}_i)_j$ of \mathbf{v}_i are small modulo p :

$$(\mathbf{v}_i)_j = r_{ij} \pmod{p}$$

However, we only obtain the randomized vectors:

$$\tilde{\mathbf{v}}_i = \mathbf{v}_i \cdot \mathbf{K} \pmod{x_0}$$

for some random invertible matrix \mathbf{K} modulo x_0 . The goal is still to recover the prime p .

Our attack is similar to the Lee-Seo attack recalled previously. We only consider the first component $c_i = (\tilde{\mathbf{v}}_i)_1$ of each vector $\tilde{\mathbf{v}}_i$. We have:

$$c_i = (\tilde{\mathbf{v}}_i)_1 = \sum_{j=1}^m (\mathbf{v}_i)_j \cdot \mathbf{K}_{j1} = \sum_{j=1}^m r_{ij} \cdot \mathbf{K}_{j1} \pmod{p}$$

We build the two lists L_1 and L_2 from the c_i 's as in the Lee-Seo attack. Since each c_i is a linear combination of m randoms r_{ij} , it has $m \cdot \rho$ bits of entropy modulo p , instead of ρ in the Lee-Seo attack. Therefore a collision between the two lists will occur with good probability when the lists have size at least $2^{m \cdot \rho/2}$. This implies that the attack has time and memory complexity $\tilde{O}(2^{m \cdot \rho/2})$. Note that the entropy of each c_i modulo p is actually upper-bounded by the bitsize η of p . If $m \cdot \rho > \eta$, the attack complexity becomes $\tilde{O}(2^{\eta/2})$, which corresponds to the complexity of the Pollard's rho factoring algorithm. We provide in Appendix C.3 an implementation of the attack in Sage.

With an attack complexity $\tilde{O}(2^{m \cdot \rho/2})$ instead of $\tilde{O}(2^{\rho/2})$, one can therefore divide the size of the noise ρ by a factor m compared to the original CLT13, which is a significant improvement. For example, it is recommended in [CLT13] to take $\rho = 89$ bits for $\lambda = 80$ bits of security; with a vector dimension $m = 10$, one can now take $\rho = 9$ for the same level of security.

With matrices. The previous GCD attack can be generalized to $m \times m$ matrices \mathbf{V}_i instead of m -dimensional vectors \mathbf{v}_i . More precisely, we consider a set of matrices \mathbf{V}_i of dimension $m \times m$ with small components modulo p , that is:

$$(\mathbf{V}_i)_{jk} = r_{ijk} \pmod{p} \tag{10}$$

for ρ -bit integers r_{ijk} . As previously, instead of publishing the matrices \mathbf{V}_i , we publish the randomized matrices

$$\tilde{\mathbf{V}}_i = \mathbf{K} \cdot \mathbf{V}_i \cdot \mathbf{K}' \pmod{x_0} \tag{11}$$

for two random invertible $m \times m$ matrices \mathbf{K} and \mathbf{K}' modulo x_0 . In that case, each component of $\tilde{\mathbf{V}}_i$ depends on the m^2 elements of the matrix \mathbf{V}_i . This implies that the entropy of each component of $\tilde{\mathbf{V}}_i$ is now $m^2 \cdot \rho$ and therefore the GCD attack has complexity $\tilde{O}(2^{m^2 \cdot \rho/2})$.

Formally, using the Kronecker product (see Appendix A), we can rewrite (11) as

$$\text{vec}(\tilde{\mathbf{V}}_i) = (\mathbf{K}'^T \otimes \mathbf{K}) \text{vec}(\mathbf{V}_i),$$

where $\text{vec}(\mathbf{V}_i)$ denotes the column vector of dimension m^2 formed by stacking the columns of \mathbf{V}_i on top of one another, and similarly for $\text{vec}(\tilde{\mathbf{V}}_i)$. We can therefore apply the previous attack with vectors of dimension m^2 instead of m ; the attack complexity is therefore $\tilde{O}(2^{m^2 \cdot \rho/2})$. This implies that we can divide the noise size ρ by a factor m^2 compared to [CLT13], where m is the matrix dimension.

With discrete Gaussian distribution. Since for matrices the noise size ρ can be divided by a large factor m^2 , it can be more convenient to use a different distribution for the noise r_{ijk} in (10). For our construction in Section 5, instead of the uniform distribution, we use the discrete Gaussian distribution on \mathbb{Z} with mean 0 and parameter σ , denoted D_σ . We let E be the random variable on \mathbb{Z} such that for $x \in \mathbb{Z}$,

$$\Pr[E = x] = \frac{1}{S} e^{-x^2/(2\sigma^2)},$$

where $S = \sum_{k=-\infty}^{\infty} e^{-k^2/(2\sigma^2)}$. Let h_σ be the entropy of the distribution D_σ . When using D_σ to generate the integers r_{ijk} instead of the uniform distribution in $[0, 2^\rho)$, the entropy of each component of $\tilde{\mathbf{V}}_i$ is now $m^2 \cdot h_\sigma$ instead of $m^2\rho$, and the attack complexity is therefore $\tilde{\mathcal{O}}(2^{m^2 \cdot h_\sigma/2})$. Since the attack is based on the same multipoint polynomial evaluation as in the Chen-Nguyen attack, its running time can be approximated by $T_{CN}(\rho, \gamma)$ from (9), with $\rho = m^2 h_\sigma$.

With multiple primes p_i 's. Instead of considering an encoding c that is small modulo a single prime p , we consider as in CLT13 a modulus $x_0 = \prod_{i=1}^n p_i$ and an integer $c \in \mathbb{Z}_{x_0}$ such that

$$c \bmod p_i = r_i$$

for ρ -bit integers r_i . With good probability, we have $|r_i| \leq 2^\rho/n$ for some i but not all i , and Equation (8) from the Chen-Nguyen attack can be rewritten:

$$p_i \mid \gcd \left(x_0, \prod_{j=0}^{\lfloor 2^\rho/n \rfloor} (c - j) \pmod{x_0} \right)$$

where the gcd is not equal to x_0 ; therefore a sub-product of the p_i 's is revealed. Since the number of terms in the product is divided by n , the complexity of the Chen-Nguyen attack for recovering a single p_i (or a sub-product of the p_i 's) is divided by \sqrt{n} . By repeating the same attack n times in different intervals of the r_i 's, one can recover all the p_i 's; the running time of the Chen-Nguyen attack is then increased by a factor \sqrt{n} .

Similarly, in the Lee-Seo attack with multiple primes p_i 's, the collision probability for recovering a single p_i is multiplied by n , and therefore the attack complexity is divided by \sqrt{n} for recovering a single p_i . The same applies to our variant attack against the Vector Approximate GCD problem and to the matrix variant. In the later case, the running time of the attack in number of clock cycles can therefore be approximated by

$$T_{GCD}(m, \gamma, h_\sigma, n) = T_{CN}(\rho, \gamma)/\sqrt{n} \tag{12}$$

with $\rho = m^2 h_\sigma$. We will use that approximation to provide concrete parameters for our scheme in Section 7.

4 Lattice attack against the Approximate-GCD Problem

We first recall the lattice attack against the single-prime Approximate-GCD problem [DGHV10, §B.1], based on the Nguyen-Stern orthogonal lattice attack [NS01]. As mentioned in introduction, extending the attack to the multiprime case is actually not straightforward; in particular, we argue that the approach described in [CLT13] is incomplete and does not recover the primes p_i 's, except for small values of n .

Our new attack against the multiprime case proceeds in two steps. The first step is the classical orthogonal lattice attack; it recovers a basis of the lattice generated by the vectors $\mathbf{r}_i = \mathbf{x} \bmod p_i$ for $1 \leq i \leq n$, where $\mathbf{x} = (x_1, \dots, x_t)$. However, the vectors \mathbf{r}_i cannot be recovered directly (except possibly in low dimension with LLL). Therefore, in the second step, thanks to the lattice basis obtained from the first step, we apply a variant of the Cheon *et al.* attack; we show that by computing the eigenvalues of a well chosen matrix, we can recover the primes p_i 's.

Finally we extend the attack to the vector variant of the Approximate-GCD problem, and we provide a more detailed analysis of the lattice reduction step of the attack, in order to derive concrete parameters.

4.1 The orthogonal lattice

We first recall the definition of the orthogonal lattice, following [NS97]. Let L be a lattice in \mathbb{Z}^m . The orthogonal lattice L^\perp is defined as the set of elements in \mathbb{Z}^m which are orthogonal to all the lattice points of L , for the usual dot product. We define the lattice $\bar{L} = (L^\perp)^\perp$; it is the intersection of \mathbb{Z}^m with the \mathbb{Q} -vector space generated by L ; we have that $L \subset \bar{L}$ and the determinant of \bar{L} divides the determinant of L . We have that $\dim(L) + \dim(L^\perp) = m$ and $\det(L^\perp) = \det(\bar{L})$.

From Minkowski's bound, we expect that a reduced basis of a "random" lattice L has vectors of norm $\simeq (\det L)^{1/\dim L}$. For a "random" lattice L , we also expect that $\det(L) \simeq \det(\bar{L}) = \det(L^\perp)$. Moreover, for a lattice L generated by a set of d "random" vectors $\mathbf{b}_i \in \mathbb{Z}^m$, from Hadamard inequality we expect that $\det L \simeq \prod_{i=1}^d \|\mathbf{b}_i\|$. In that case, we therefore expect the short vectors of L^\perp to have norm $\simeq (\det L^\perp)^{1/(m-d)} \simeq (\det L)^{1/(m-d)} \simeq (\prod_{i=1}^d \|\mathbf{b}_i\|)^{1/(m-d)}$.

4.2 The classical orthogonal lattice attack against the single-prime Approximate-GCD problem

In this section we recall the lattice attack against the Approximate-GCD problem, based on the Nguyen-Stern orthogonal lattice attack [NS01]; see also the analysis in [DGHV10, §B.1]. We consider a set of t integers $x_i = p \cdot q_i + r_i$ and $x_0 = p \cdot q_0$, for $r_i \in (-2^\rho, 2^\rho) \cap \mathbb{Z}$. We consider the lattice L of vectors \mathbf{u} that are orthogonal to \mathbf{x} modulo x_0 , where $\mathbf{x} = (x_1, \dots, x_t)$:

$$L = \{ \mathbf{u} \in \mathbb{Z}^t \mid \mathbf{u} \cdot \mathbf{x} \equiv 0 \pmod{x_0} \}$$

The lattice L is of full rank t since it contains $x_0 \mathbb{Z}^t$. Moreover, we have:

$$\det L = x_0 / \gcd(x_0, x_1, \dots, x_t) = x_0.$$

Therefore, applying lattice reduction should yield a reduced basis $(\mathbf{u}_1, \dots, \mathbf{u}_t)$ with vectors of length

$$\|\mathbf{u}_k\| \leq 2^{\iota t} \cdot (\det L)^{1/t} \approx 2^{\iota t + \gamma/t} \quad (13)$$

where γ is the size of x_0 , for some constant $\iota > 0$ depending on the lattice reduction algorithm, where $2^{\iota t}$ is the Hermite factor.

Now given a vector $\mathbf{u} \in L$, we have $\mathbf{u} \cdot \mathbf{x} \equiv 0 \pmod{x_0}$, which implies that $\mathbf{u} \cdot \mathbf{r} \equiv 0 \pmod{p}$ where $\mathbf{r} = (r_1, \dots, r_t)$. The main observation is that if \mathbf{u} is short enough, the equality will hold over \mathbb{Z} . More precisely, if $\|\mathbf{u}\| \cdot \|\mathbf{r}\| < p$, we get $\mathbf{u} \cdot \mathbf{r} = 0$ in \mathbb{Z} . From (13), this happens under the condition:

$$2^{\iota t + \gamma/t} \cdot 2^\rho < 2^\eta. \quad (14)$$

In that case, the vectors $(\mathbf{u}_1, \dots, \mathbf{u}_{t-1})$ from the previous lattice reduction step should be orthogonal to the vector \mathbf{r} . One can therefore recover $\pm \mathbf{r}$ by computing the rank 1 lattice orthogonal to those vectors. From \mathbf{r} one can recover p by computing $p = \gcd(x_0, x_1 - r_1)$.

Asymptotic complexity. From condition (14) the attack requires $t > \gamma/\eta$; therefore from the same condition we must have

$$\iota < \eta^2/\gamma.$$

Achieving an Hermite factor of $2^{\iota t}$ heuristically requires $2^{\Omega(1/\iota)}$ time, by using BKZ reduction with block-size $\beta = \omega(1/\iota)$ [HPS11]. Therefore, the orthogonal lattice attack has time complexity $2^{\Omega(\gamma/\eta^2)}$.

4.3 Lattice attack against multi-prime Approximate GCD

We consider the setting of CLT13, that is we are given a modulus $x_0 = \prod_{i=1}^n p_i$ and a set of integers $x_j \in \mathbb{Z}_{x_0}$ such that:

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

for $r_{ij} \in (-2^\rho, 2^\rho) \cap \mathbb{Z}$, and the goal is to recover the secret primes p_i .

First step: orthogonal lattice attack. As previously we consider the integer vector \mathbf{x} formed by the first t integers x_j , and we consider the lattice L of vectors \mathbf{u} that are orthogonal to \mathbf{x} modulo x_0 :

$$L = \{ \mathbf{u} \in \mathbb{Z}^t \mid \mathbf{u} \cdot \mathbf{x} \equiv 0 \pmod{x_0} \}$$

Note that the lattice L is of full rank t since it contains $x_0\mathbb{Z}^t$. For $1 \leq i \leq n$, let $\mathbf{r}_i = \mathbf{x} \bmod p_i$. For any $\mathbf{u} \in \mathbb{Z}^t$, if $\mathbf{u} \cdot \mathbf{r}_i = 0$ in \mathbb{Z} for all $1 \leq i \leq n$, then $\mathbf{u} \cdot \mathbf{x} \equiv 0 \pmod{x_0}$. Therefore, denoting by $L_{\mathbf{r}_i}$ the lattice generated by the vectors \mathbf{r}_i , the lattice L contains the sublattice $L_{\mathbf{r}_i}^\perp$ of the vectors orthogonal in \mathbb{Z} to the n vectors \mathbf{r}_i 's. Assuming that the n vectors \mathbf{r}_i 's are linearly independent, we have $\dim L_{\mathbf{r}_i}^\perp = t - n$, and we expect a reduced basis of $L_{\mathbf{r}_i}^\perp$ to have vectors of norm $(\prod_{i=1}^n \|\mathbf{r}_i\|)^{1/(t-n)} \simeq 2^{\rho n/(t-n)}$.

Given a vector $\mathbf{u} \in L$, we have $\mathbf{u} \cdot \mathbf{x} \equiv 0 \pmod{x_0}$, which implies that $\mathbf{u} \cdot \mathbf{r}_i \equiv 0 \pmod{p_i}$ for all $1 \leq i \leq n$. As previously, if \mathbf{u} is short enough, the equalities will hold over \mathbb{Z} . More precisely, if $\|\mathbf{u}\| \cdot \|\mathbf{r}_i\| < p_i$ for all $1 \leq i \leq n$, we get $\mathbf{u} \cdot \mathbf{r}_i = 0$ in \mathbb{Z} for all i ; therefore we must have $\mathbf{u} \in L_{\mathbf{r}_i}^\perp$ under the condition $\|\mathbf{u}\| < (\min p_i)/(\max \|\mathbf{r}_i\|) \simeq 2^{\eta-\rho}$. Hence, when applying lattice reduction to the lattice L , we expect to recover the vectors from the sublattice $L_{\mathbf{r}_i}^\perp$ if there is a gap at least 2^{t-t} between the short vectors in $L_{\mathbf{r}_i}^\perp$ and the other vectors in $L \setminus L_{\mathbf{r}_i}^\perp$, where 2^{t-t} is the Hermite factor. Since the vectors in $L \setminus L_{\mathbf{r}_i}^\perp$ must have norm at least approximately $2^{\eta-\rho}$, this gives the condition:

$$2^{\rho n/(t-n)} \cdot 2^{t-t} < 2^{\eta-\rho}, \quad (15)$$

In that case, applying lattice reduction to L should yield a reduced basis $(\mathbf{u}_1, \dots, \mathbf{u}_t)$ where the first $t - n$ vectors belong to the sublattice $L_{\mathbf{r}_i}^\perp$. By computing the rank n lattice orthogonal to those vectors, one recovers a basis $B = (\mathbf{b}_1, \dots, \mathbf{b}_n)$ of the lattice $\bar{L}_{\mathbf{r}_i} = (L_{\mathbf{r}_i}^\perp)^\perp$. However this does not necessarily reveal the original vectors \mathbf{r}_i . Namely even by applying LLL or BKZ on the basis B , we do not necessarily recover the short vectors \mathbf{r}_i 's, except possibly in low dimension.

However, the main observation is that since each vector \mathbf{b}_j of the basis B is a linear combination of the vectors \mathbf{r}_i , it can play the same role as a zero-tested value in the CLT13 scheme. More precisely, since the vectors $\mathbf{b}_1, \dots, \mathbf{b}_n$ form a basis of $\bar{L}_{\mathbf{r}_i}$, we can write for all $1 \leq j \leq n$:

$$\mathbf{b}_j = \sum_{i=1}^n \lambda_{ji} \mathbf{r}_i$$

for unknown coefficients $\lambda_{ji} \in \mathbb{Q}$. The above equation is analogous to Equation (7) on the zero-tested value $c \cdot p_{zt}$, which is a linear combination of the \mathbf{r}_i 's over \mathbb{Z} when all m_i 's are zero. Therefore, we can apply a variant of the Cheon *et al.* attack to recover the primes p_i 's, by computing the eigenvalues of a well chosen matrix. Since we have n vectors \mathbf{b}_j instead of a single p_{zt} value, we only need to work with equations of degree 2 in the x_j 's, instead of degree 3 as in [CHL⁺15].

Second step: algebraic attack. The second step of the attack is similar to the Cheon *al.* attack. Recall that we receive as input $x_0 = \prod_{i=1}^n p_i$ and a set of integers $x_j \in \mathbb{Z}_{x_0}$ such that $x_j \bmod p_i = r_{ij}$ for $r_{ij} \in (-2^\rho, 2^\rho) \cap \mathbb{Z}$. Since we must work with an equation of degree 2 in the inputs, we consider an additional integer $y \in \mathbb{Z}_{x_0}$ with $y \bmod p_i = s_i$ with $s_i \in (-2^\rho, 2^\rho) \cap \mathbb{Z}$ for all $1 \leq i \leq n$.

We define the column vector $\mathbf{x} = [x_1 \dots x_n]^T$. Instead of running the orthogonal lattice attack with \mathbf{x} , we run the orthogonal lattice attack from the previous step with the column vector \mathbf{z} of dimension $t = 2n$ defined as follows:

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ y \cdot \mathbf{x} \end{bmatrix}$$

Letting $\mathbf{r}_i = \mathbf{x} \bmod p_i$, this gives the column vectors for $1 \leq i \leq n$:

$$\mathbf{z} \bmod p_i = \begin{bmatrix} \mathbf{r}_i \\ s_i \cdot \mathbf{r}_i \end{bmatrix}$$

We denote by \mathbf{Z} the $2n \times n$ matrix of column vectors $\mathbf{z} \bmod p_i$:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{r}_1 & \cdots & \mathbf{r}_n \\ s_1 \cdot \mathbf{r}_1 & \cdots & s_n \cdot \mathbf{r}_n \end{bmatrix} = \begin{bmatrix} \mathbf{R} \\ \mathbf{R} \cdot \mathbf{U} \end{bmatrix}$$

where \mathbf{R} is the $n \times n$ matrix of column vectors \mathbf{r}_i , and $\mathbf{U} := \text{diag}(s_1, \dots, s_n)$.

By applying the orthogonal lattice attack of the first step on the vector \mathbf{z} , we obtain a basis of the lattice intersection of \mathbb{Z}^{2n} with the \mathbb{Q} -vector space generated by the n vectors $\mathbf{z} \bmod p_i$, which corresponds to the columns of the matrix \mathbf{Z} . Therefore we obtain two matrices \mathbf{W}_0 and \mathbf{W}_1 such that:

$$\begin{aligned} \mathbf{W}_0 &= \mathbf{R} \cdot \mathbf{A} \\ \mathbf{W}_1 &= \mathbf{R} \cdot \mathbf{U} \cdot \mathbf{A} \end{aligned}$$

for some unknown matrix $\mathbf{A} \in \mathbb{Q}^{n \times n}$. Therefore, as in the Cheon *et al.* attack, we compute the matrix:

$$\mathbf{W} = \mathbf{W}_1 \cdot \mathbf{W}_0^{-1} = \mathbf{R} \cdot \mathbf{U} \cdot \mathbf{R}^{-1}$$

and by computing the eigenvalues of \mathbf{W} , one recovers the components s_i of the diagonal matrix \mathbf{U} , from which we recover the p_i 's by taking gcd's. We provide the source code of the attack in Appendix D.1.

Asymptotic complexity. Since the attack requires $t = 2n$, condition (15) gives $3\rho/2 + 2\iota n < \eta$ which implies the condition

$$\iota < \frac{\eta}{2n}$$

Achieving an Hermite factor of $2^{t\iota}$ heuristically requires $2^{\Omega(1/\iota)}$ time, by using BKZ reduction with block-size $\beta = \omega(1/\iota)$ [HPS11]. Therefore, the orthogonal lattice attack has time complexity $2^{\Omega(n/\eta)}$. Note that with $\gamma = \eta \cdot n$, we get the same time complexity $2^{\Omega(\gamma/\eta^2)}$ as for the single-prime Approximate-GCD problem. In summary, as shown in [CLT13], to prevent the orthogonal lattice attack, one must take:

$$n = \omega(\eta \log \lambda) \tag{16}$$

4.4 Lattice attack against the Vector Approximate-GCD Problem

In this section we extend the previous orthogonal lattice attack to the vector variant of the Approximate-GCD problem with multiple primes p_i 's. We still consider a modulus $x_0 = \prod_{i=1}^n p_i$, but instead of scalar values x_j , we consider row vectors \mathbf{v}_j , each with m components $(\mathbf{v}_j)_k$, such that:

$$(\mathbf{v}_j)_k = r_{ijk} \pmod{p_i}$$

for all components $1 \leq k \leq m$ and all $1 \leq i \leq n$, where $r_{ijk} \in (-2^\rho, 2^\rho) \cap \mathbb{Z}$. We consider the $t \times m$ matrix \mathbf{V} of row vectors \mathbf{v}_j . We don't publish the matrix \mathbf{V} directly; instead we first generate a random secret $m \times m$ invertible matrix \mathbf{K} modulo x_0 and publish the $t \times m$ matrix:

$$\tilde{\mathbf{V}} = \mathbf{V} \cdot \mathbf{K} \pmod{x_0}$$

The goal is to recover the primes p_i 's as in the previous attack.

First step: orthogonal lattice attack. In our extended attack we consider the lattice L of vectors \mathbf{u} that are orthogonal to all columns of $\tilde{\mathbf{V}}$ modulo x_0 :

$$L = \{ \mathbf{u} \in \mathbb{Z}^t \mid \mathbf{u} \cdot \tilde{\mathbf{V}} \equiv 0 \pmod{x_0} \}$$

Since the matrix \mathbf{K} is invertible, we obtain:

$$L = \{ \mathbf{u} \in \mathbb{Z}^t \mid \mathbf{u} \cdot \mathbf{V} \equiv 0 \pmod{x_0} \} \quad (17)$$

The lattice L is of full rank t since it contains $x_0 \mathbb{Z}^t$. Let $\mathbf{R}_i = \mathbf{V} \bmod p_i$. As previously, the lattice L contains the sublattice L' of dimension $t - m \cdot n$ of the vectors orthogonal in \mathbb{Z} to the $m \cdot n$ column vectors in \mathbf{R}_i for $1 \leq i \leq n$. We expect a reduced basis of L' to have vectors of norm $\simeq 2^{\rho \cdot m \cdot n / (t - m \cdot n)}$. Therefore, applying lattice reduction to L should yield a reduced basis $(\mathbf{u}_1, \dots, \mathbf{u}_t)$ where the first $t - m \cdot n$ vectors belong to the sublattice L' , under the modified condition:

$$2^{t + \rho \cdot m \cdot n / (t - m \cdot n)} < 2^{\eta - \rho} \quad (18)$$

As previously, by computing the rank $n \cdot m$ lattice orthogonal to the vectors $(\mathbf{u}_1, \dots, \mathbf{u}_{t - m \cdot n})$, we obtain a basis of the lattice intersection of \mathbb{Z}^t with the \mathbb{Q} -vector space generated by the column vectors of the \mathbf{R}_i 's.

Second step: algebraic attack. Actually, we cannot solve the original multiprime vector Approximate-GCD problem, since the algebraic step of the attack requires degree 2 equations in the inputs. Instead, we assume that we can additionally obtain the two $m \times m$ matrices:

$$\begin{aligned} \tilde{\mathbf{C}}_0 &= \mathbf{K}^{-1} \cdot \mathbf{C}_0 \cdot \mathbf{K}' \pmod{x_0} \\ \tilde{\mathbf{C}}_1 &= \mathbf{K}^{-1} \cdot \mathbf{C}_1 \cdot \mathbf{K}' \pmod{x_0} \end{aligned}$$

for some random invertible matrix \mathbf{K}' modulo x_0 , where the components of the matrices $\mathbf{C}_0, \mathbf{C}_1 \in \mathbb{Z}_{x_0}^{m \times m}$ are small modulo each p_i . This assumption is verified in our construction of Section 5. Therefore, considering the original $t \times m$ matrix $\tilde{\mathbf{V}}$ and using $t = nm$ rows:

$$\tilde{\mathbf{V}} = \mathbf{V} \cdot \mathbf{K} \pmod{x_0}$$

we can obtain the two $nm \times m$ matrices:

$$\begin{aligned} \tilde{\mathbf{D}}_0 &= \tilde{\mathbf{V}} \cdot \tilde{\mathbf{C}}_0 = \mathbf{V} \cdot \mathbf{C}_0 \cdot \mathbf{K}' = \mathbf{D}_0 \cdot \mathbf{K}' \pmod{x_0} \\ \tilde{\mathbf{D}}_1 &= \tilde{\mathbf{V}} \cdot \tilde{\mathbf{C}}_1 = \mathbf{V} \cdot \mathbf{C}_1 \cdot \mathbf{K}' = \mathbf{D}_1 \cdot \mathbf{K}' \pmod{x_0} \end{aligned}$$

where $\mathbf{D}_0 = \mathbf{V} \cdot \mathbf{C}_0$ and $\mathbf{D}_1 = \mathbf{V} \cdot \mathbf{C}_1$.

Instead of applying the lattice attack with $\tilde{\mathbf{V}}$, we apply the lattice attack of the first step to the $2nm \times m$ matrix $\tilde{\mathbf{D}} = \begin{bmatrix} \tilde{\mathbf{D}}_0 \\ \tilde{\mathbf{D}}_1 \end{bmatrix}$, with $t = 2nm$ rows. Since $\tilde{\mathbf{D}} = \mathbf{D} \cdot \mathbf{K}'$ where $\mathbf{D} = \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{D}_1 \end{bmatrix}$, this is the same as applying the lattice attack against the matrix \mathbf{D} . As previously, for all $1 \leq i \leq n$ we let $\mathbf{R}_i = \mathbf{V} \bmod p_i$, and we let $\mathbf{S}_{0,i} = \mathbf{C}_0 \bmod p_i$ and $\mathbf{S}_{1,i} = \mathbf{C}_1 \bmod p_i$. This gives:

$$\mathbf{D} \bmod p_i = \begin{bmatrix} \mathbf{V} \cdot \mathbf{C}_0 \bmod p_i \\ \mathbf{V} \cdot \mathbf{C}_1 \bmod p_i \end{bmatrix} = \begin{bmatrix} \mathbf{R}_i \cdot \mathbf{S}_{0,i} \\ \mathbf{R}_i \cdot \mathbf{S}_{1,i} \end{bmatrix}$$

We denote by \mathbf{E} the $2nm \times nm$ matrix obtained by concatenating the columns of $\mathbf{D} \bmod p_i$ for $1 \leq i \leq n$. Similarly we denote by \mathbf{R} the $nm \times nm$ matrix obtained by concatenating the columns of the matrices \mathbf{R}_i . We denote by $\hat{\mathbf{S}}_0$ the $nm \times nm$ block-diagonal matrix $\hat{\mathbf{S}}_0 = \text{diag}(\mathbf{S}_{0,1}, \dots, \mathbf{S}_{0,n})$, and similarly $\hat{\mathbf{S}}_1 = \text{diag}(\mathbf{S}_{1,1}, \dots, \mathbf{S}_{1,n})$. We can write:

$$\mathbf{E} = \begin{bmatrix} \mathbf{R}_1 \cdot \mathbf{S}_{0,1} & \cdots & \mathbf{R}_n \cdot \mathbf{S}_{0,n} \\ \mathbf{R}_1 \cdot \mathbf{S}_{1,1} & \cdots & \mathbf{R}_n \cdot \mathbf{S}_{1,n} \end{bmatrix} = \begin{bmatrix} \mathbf{R} \cdot \hat{\mathbf{S}}_0 \\ \mathbf{R} \cdot \hat{\mathbf{S}}_1 \end{bmatrix}$$

By applying the orthogonal lattice attack of the first step on the matrix $\tilde{\mathbf{D}}$, we obtain as previously a basis of the lattice intersection of \mathbb{Z}^{2nm} with the \mathbb{Q} -vector space generated by the nm columns of the matrices $\mathbf{D} \bmod p_i$, which corresponds to the columns of the matrix \mathbf{E} . Therefore as previously we obtain two matrices \mathbf{W}_0 and \mathbf{W}_1 such that:

$$\begin{aligned}\mathbf{W}_0 &= \mathbf{R} \cdot \hat{\mathbf{S}}_0 \cdot \mathbf{A} \\ \mathbf{W}_1 &= \mathbf{R} \cdot \hat{\mathbf{S}}_1 \cdot \mathbf{A}\end{aligned}$$

for some unknown matrix $\mathbf{A} \in \mathbb{Q}^{nm \times nm}$. Therefore we can compute the matrix:

$$\mathbf{W} = \mathbf{W}_1 \cdot \mathbf{W}_0^{-1} = \mathbf{R} \cdot \hat{\mathbf{S}}_1 \cdot \hat{\mathbf{S}}_0^{-1} \mathbf{R}^{-1}$$

The characteristic polynomial $f(X)$ of \mathbf{W} is therefore the same as the characteristic polynomial of $\hat{\mathbf{S}}_1 \cdot \hat{\mathbf{S}}_0^{-1}$ which is the product of the n characteristic polynomials $f_i(X)$ of the matrices $\mathbf{S}_{1,i} \cdot \mathbf{S}_{0,i}^{-1}$. By the Cayley-Hamilton theorem, we must have $f_i(\mathbf{S}_{1,i} \cdot \mathbf{S}_{0,i}^{-1}) = 0$ for all i , which implies $f_i(\mathbf{C}_1 \cdot \mathbf{C}_0^{-1}) = 0 \pmod{p_i}$, which also implies $f_i(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_0^{-1}) = 0 \pmod{p_i}$ for all $1 \leq i \leq n$. Therefore, if the polynomials $f_i(X)$ are irreducible, they can be recovered by computing $f(X)$ and factoring $f(X)$ into irreducible polynomials; then each prime p_i can be recovered by computing the gcd of the entries of $\mathbf{M}_i = f_i(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_0^{-1}) \bmod x_0$ with x_0 . We provide the source code of the attack in Appendix D.2.

Alternatively, if the polynomials $f_i(X)$ are not irreducible, one can still factor $f(X)$ into monic irreducible factors $f'_1, \dots, f'_N \in \mathbb{Q}[X]$. Then for $k \in [N]$, the attacker defines $F_k := f/f'_k \in \mathbb{Q}[X]$ and $G_k = F_k \cdot d_k \in \mathbb{Z}[X]$, where d_k is the common denominator of F_k 's coefficients. Since in F_k we have removed one irreducible factor from f , by the Cayley-Hamilton theorem we have that $G_k(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_0^{-1}) = 0$ modulo all primes except one, and therefore the remaining prime p_i can be recovered by computing the gcd of the entries of $\mathbf{M}_k = G_k(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_0^{-1}) \bmod x_0$ with x_0 .

Asymptotic complexity. Since the attack requires $t = 2mn$, condition (18) gives $3\rho/2 + 2\iota mn < \eta$ which gives the new condition

$$\iota < \frac{\eta}{2mn}$$

Therefore, the orthogonal lattice attack has time complexity $2^{\Omega(n \cdot m/\eta)}$. This implies that to prevent the orthogonal lattice attack, we must have:

$$n = \omega\left(\frac{\eta}{m} \log \lambda\right)$$

Compared to the original condition of [CLT13] recalled by (16), the value of n can therefore be divided by m . This implies that the encoding size $\gamma = \eta \cdot n$ can also be divided by m . Again this can bring a significant improvement in practice (see Section 7).

4.5 Practical complexity of the lattice attack

To derive concrete parameters for our construction from Section 5, we have run some experiments with LLL and BKZ lattice reduction algorithms applied to a lattice similar to the lattice L of the previous section. Recall that we have:

$$L = \{\mathbf{u} \in \mathbb{Z}^t \mid \mathbf{u} \cdot \tilde{\mathbf{V}} \equiv 0 \pmod{x_0}\}$$

with $t = 2nm$. We write $\mathbf{u} = [\mathbf{u}_1, \mathbf{u}_2]$ where \mathbf{u}_2 is a vector of dimension m . Similarly we write $\tilde{\mathbf{V}} = \begin{bmatrix} \mathbf{A} \\ \mathbf{W} \end{bmatrix}$ where \mathbf{W} is a $m \times m$ matrix. With high probability \mathbf{W} is invertible modulo x_0 , otherwise we can partially factor x_0 . We obtain

$$\begin{aligned}\mathbf{u} \in L &\iff \mathbf{u}_1 \mathbf{A} + \mathbf{u}_2 \mathbf{W} \equiv 0 \pmod{x_0} \\ &\iff \mathbf{u}_1 \mathbf{A} \mathbf{W}^{-1} + \mathbf{u}_2 \equiv 0 \pmod{x_0}\end{aligned}$$

Therefore, a basis of L is given by the matrix:

$$L = \begin{bmatrix} \mathbf{I}_{t-m} & -\mathbf{A}\mathbf{W}^{-1} \\ & x_0\mathbf{I}_m \end{bmatrix}$$

For simplicity, we have performed our experiments on a simpler lattice:

$$L' = \begin{bmatrix} \mathbf{I}_{t-m} & \mathbf{A}' \\ & x_0\mathbf{I}_m \end{bmatrix}$$

where the components of \mathbf{A}' are randomly generated modulo x_0 . We expect to obtain a reduced basis $(\mathbf{u}_1, \dots, \mathbf{u}_t)$ with vectors of norm:

$$\|\mathbf{u}_k\| \simeq 2^{\iota \cdot t} (\det L)^{1/t} \simeq 2^{\iota \cdot t + m \cdot \gamma / t}$$

where $2^{\iota \cdot t}$ is the Hermite factor, and γ the size of x_0 . Experimentally, we observed the following running time (expressed in number of clock cycles) for the LLL lattice reduction algorithm in the Sage implementation:

$$T_{LLL}(t, \gamma, m) \simeq 2 \cdot t^{3.3} \cdot \gamma \cdot m \quad (19)$$

The Sage implementation also includes an implementation of BKZ 2.0 [CN11]. Experimentally we observed the following running-times (in number of clock cycles):

$$T_{BKZ}(t, \beta) \simeq b(\beta) \cdot t^{4.3} \quad (20)$$

where the observed constant $b(\beta)$ and the Hermite factor are given in Table 1. However we were not able to obtain experimental results for block-sizes $\beta > 60$, so for BKZ-80 and BKZ-100 we used extrapolated values, assuming that the cost of BKZ sieving with blocksize β is $\text{poly}(t) \cdot 2^{0.292\beta + o(\beta)}$ (see [BDGL16]). The Hermite factors for BKZ-80 and BKZ-100 are from [CN11].

	LLL	BKZ-60	BKZ-80	BKZ-100
(Hermite factor) ^{1/t} = 2 ^ι	1.021	1.011	1.01	1.009
Running time parameter $b(\beta)$	–	10 ³	6 · 10 ⁴	3 · 10 ⁶

Table 1. Experimental values of running time and Hermite factor for LLL and BKZ as a function of the blocksize β . The parameters for $\beta = 80, 100$ are extrapolated.

When applying LLL or BKZ with blocksize β on the original lattice L , we obtain an orthogonal vector \mathbf{u} under the condition (18), which gives with $t = 2nm$:

$$\iota \cdot 2nm + \frac{3\rho}{2} < \eta \quad (21)$$

Therefore we must run LLL or BKZ- β with a large enough blocksize β so that ι is small enough for condition (21) to hold. For security parameter λ , we require that

$$T_{lat}(t, \gamma) \geq 2^\lambda,$$

with $t = 2nm$, where the running time (in number of clock cycles) $T_{lat}(t, \gamma)$ is given by (19) or (20), for $\gamma = \eta \cdot n$. We will use that condition to provide concrete parameters for our scheme in Section 7.

5 Our Construction

In this section we describe our construction of a non-interactive multipartite Diffie-Hellman key exchange scheme based on the CLT13 multilinear maps. We first recall the definition of such a scheme.

5.1 Non-interactive Multipartite Diffie-Hellman Key Exchange

A multipartite key exchange protocol aims to derive a shared value between N parties. This is achieved via a procedure in which the parties broadcast some values and then use some secret information together with the values broadcasted by the other parties to set up the shared key. In a non-interactive protocol, the parties broadcast their public values only once and at the same time (or equivalently, the values broadcasted by each party do not depend on the values broadcasted by the others). Following the notation of [BS03], such protocol can be described with three randomized probabilistic polynomial-time algorithms as follows.

- **Setup**($1^\lambda, N$): This algorithm runs in polynomial time in the security parameter $\lambda \in \mathbb{N}$ and in the number of parties N , and outputs the public parameters **params**.
- **Publish**(**params**, u): Given a party $u \in [N]$, this algorithm generates a pair of keys $(\mathbf{sk}_u, \mathbf{pk}_u)$. Party u broadcasts \mathbf{pk}_u and keeps \mathbf{sk}_u secret.
- **KeyGen**(**params**, v , \mathbf{sk}_v , $\{\mathbf{pk}_u\}_{u \neq v}$): Party $v \in [N]$ uses its secret \mathbf{sk}_v and all the values \mathbf{pk}_u broadcasted by other parties to generate a session key s_v .

We say that the protocol is correct if $s = s_1 = s_2 = \dots = s_N$, i.e., if all the parties share the same value at the end. We say that the protocol is secure if no probabilistic polynomial-time adversary can distinguish the shared value s from a random string given the public parameters **params** and the broadcasted values $\mathbf{pk}_1, \dots, \mathbf{pk}_N$.

5.2 Our Construction

We describe our N -party one-round key exchange protocol. We start with the **Setup** procedure, which is run a single time to generate the public parameters. As illustrated in Table 2, **Setup** generates for each party v two sequences of matrices $(\mathbf{C}_{i,b}^{(v)})_{i=1,\dots,\ell}$ for $b \in \{0, 1\}$. In the **KeyGen** procedure, each party v will use the product of the matrices $\mathbf{C}_{i,b}^{(v)}$ on his row v to generate the session-key. The product is computed according to the secret-key \mathbf{sk}_v of Party v and the secret-keys \mathbf{sk}_u of the other parties. Therefore, in the **Publish** procedure, each party u will compute and publish the partial sub-products corresponding to his \mathbf{sk}_u on the other rows $v \neq u$, to be used by each party v on his row v .

Party 1	$\mathbf{C}_{1,0}^{(1)}$	$\mathbf{C}_{2,0}^{(1)}$	\dots	$\mathbf{C}_{\ell,0}^{(1)}$
	$\mathbf{C}_{1,1}^{(1)}$	$\mathbf{C}_{2,1}^{(1)}$	\dots	$\mathbf{C}_{\ell,1}^{(1)}$
Party 2	$\mathbf{C}_{1,0}^{(2)}$	$\mathbf{C}_{2,0}^{(2)}$	\dots	$\mathbf{C}_{\ell,0}^{(2)}$
	$\mathbf{C}_{1,1}^{(2)}$	$\mathbf{C}_{2,1}^{(2)}$	\dots	$\mathbf{C}_{\ell,1}^{(2)}$
Party 3	$\mathbf{C}_{1,0}^{(3)}$	$\mathbf{C}_{2,0}^{(3)}$	\dots	$\mathbf{C}_{\ell,0}^{(3)}$
	$\mathbf{C}_{1,1}^{(3)}$	$\mathbf{C}_{2,1}^{(3)}$	\dots	$\mathbf{C}_{\ell,1}^{(3)}$

Table 2. Public matrices for $N = 3$ generated during the **Setup** procedure.

Setup($1^\lambda, N$): given a security parameter λ and the number of participants N , we set the length μ of each parties' secret, the number of repetitions k , and the dimension m of the matrices, with $m \equiv 0 \pmod{3}$. We then instantiate the CLT13 multilinear map with degree of multilinearity $\ell + 2$ with $\ell := \mu N k$. Let $g = \prod_{i=1}^n g_i$ be the integer defining the message space \mathbb{Z}_g . Let ν be the number of high-order bits that can be extracted from a zero-tested value.

We first sample 2ℓ random invertible matrices $\mathbf{B}_{i,b}$ in $\mathbb{Z}_g^{m' \times m'}$ where $m' = m/3$, for $1 \leq i \leq \ell$ and $b \in \{0, 1\}$. For each $u \in [N]$, we additionally sample 2ℓ scalars $\alpha_{i,b}^{(u)}$ in \mathbb{Z}_g^* and 4ℓ random invertible matrices $\mathbf{S}_{i,b}^{(u)}$ and $\mathbf{T}_{i,b}^{(u)}$ in $\mathbb{Z}_g^{m' \times m'}$, for $1 \leq i \leq \ell$ and $b \in \{0, 1\}$. As illustrated in (6), we let

$$\mathbf{A}_{i,b}^{(u)} := \text{diag}(\mathbf{S}_{i,b}^{(u)}, \mathbf{T}_{i,b}^{(u)}, \alpha_{i,b}^{(u)} \cdot \mathbf{B}_{i,b}) \quad (22)$$

The scalars $\alpha_{i,b}^{(u)}$ must satisfy the following condition:

$$\forall u, v \in [N], \forall i \in [N\mu], \forall b \in \{0, 1\}, \quad \prod_{j=0}^{k-1} \alpha_{j \cdot N \cdot \mu + i - 1, b}^{(u)} = \prod_{j=0}^{k-1} \alpha_{j \cdot N \cdot \mu + i - 1, b}^{(v)} \pmod{g}$$

In addition, we sample the vectors $\mathbf{s}^*, \mathbf{t}^*$ uniformly from $\mathbb{Z}_g^{m'}$, and for each $u \in [N]$ we define a left bookend vector

$$\mathbf{s}^{(u)} := (0, \dots, 0, \$, \dots, \$, \mathbf{s}^*) \in \mathbb{Z}_g^m$$

where the block of 0's and the block of randoms have the same length $m' = m/3$ as \mathbf{s}^* , and similarly a right bookend vector $\mathbf{t}^{(u)} := (\$, \dots, \$, 0, \dots, 0, \mathbf{t}^*) \in \mathbb{Z}_g^m$.

We let $\tilde{\mathbf{A}}_{i,b}^{(u)} \in \mathbb{Z}_{x_0}^{m \times m}$ be the matrix obtained by encoding each entry of $\mathbf{A}_{i,b}^{(u)}$ at level 0, that is without any multiplicative mask z_{i_0} . Similarly we encode $\mathbf{s}^{(u)}$ and $\mathbf{t}^{(u)}$ entry-wise at level 0, obtaining $\tilde{\mathbf{s}}^{(u)}$ and $\tilde{\mathbf{t}}^{(u)}$. For each $u \in [N]$, we sample uniformly random invertible matrices $\mathbf{K}_i^{(u)} \in \mathbb{Z}_{x_0}^{m \times m}$ for $0 \leq i \leq \ell$. We then use Kilian's randomization "on the encoding side" and define:

$$\mathbf{C}_{i,b}^{(u)} := \mathbf{K}_{i-1}^{(u)} \tilde{\mathbf{A}}_{i,b}^{(u)} \left(\mathbf{K}_i^{(u)} \right)^{-1} \pmod{x_0}$$

Similarly, we define $\bar{\mathbf{s}}^{(u)} := \tilde{\mathbf{s}}^{(u)} \left(\mathbf{K}_0^{(u)} \right)^{-1} \pmod{x_0}$ and $\bar{\mathbf{t}}^{(u)} := \mathbf{K}_\ell^{(u)} \tilde{\mathbf{t}}^{(u)} p_{zt} \pmod{x_0}$. Finally we output **params**, which is defined as the set containing all the matrices $\mathbf{C}_{i,b}^{(u)}$, the bookend vectors $\bar{\mathbf{s}}^{(u)}$ and $\bar{\mathbf{t}}^{(u)}$, and the scalars $\mu, k, N, \ell, x_0, \nu$ and m .

Note that since we use Kilian's randomization on the encoding side, we don't need to use the level encoding from CLT13, as any multiplicative mask z_i can be equivalently integrated into the random invertible matrices $\mathbf{K}_i^{(u)}$; in particular, p_{zt} does not include the product of the z_i 's and is kept secret, as it is already integrated in the right bookend vectors $\bar{\mathbf{t}}^{(u)}$.

Publish(params, u): Party u samples a bit string $\text{sk}^{(u)} \in \{0, 1\}^\mu$ and for each $v \in [N]$ such that $u \neq v$, Party u computes k products using matrices from the row of party v . Namely, Party u computes and broadcasts the following products:

$$\mathbf{D}_r^{(u \rightarrow v)} := \prod_{i=0}^{\mu-1} \mathbf{C}_{(r-1)N\mu + (u-1)\mu + i, \text{sk}^{(u)}[i]}^{(v)} \pmod{x_0} \quad (23)$$

for each $v \neq u$ and $r \in [k]$. The notation $u \rightarrow v$ stands for "computed by u to be used by v ". We let $\text{pk}_u = \{\mathbf{D}_r^{(u \rightarrow v)} : v \in [N], v \neq u, r \in [k]\}$.

KeyGen(params, v, sk^(v), {pk_u}_{u≠v}): Using secret $\text{sk}^{(v)}$, party v computes the products $\mathbf{D}_r^{(v \rightarrow v)}$ for all $r \in [k]$ using (23), and then the product

$$z^{(v)} := \bar{\mathbf{s}}^{(v)} \left(\prod_{r=1}^k \left(\prod_{u=1}^N \mathbf{D}_r^{(u \rightarrow v)} \right) \right) \bar{\mathbf{t}}^{(v)} \pmod{x_0}. \quad (24)$$

Eventually the shared key is obtained by applying a strong randomness extractor to the ν most-significant bits of $z^{(v)}$. This terminates the description of our construction.

Correctness. It is easy to verify the correctness of our construction. Namely defining sk as the concatenated secret-keys with the k repetitions:

$$\text{sk} = \underbrace{(\text{sk}^{(1)}, \dots, \text{sk}^{(N)})}_{\text{First repetition}}, \dots, \underbrace{(\text{sk}^{(1)}, \dots, \text{sk}^{(N)})}_{k\text{-th repetition}} \quad (25)$$

we obtain from (23) and (24), and then from the cancellation of Kilian's randomization on the encoding side:

$$z^{(v)} = \bar{s}^{(v)} \left(\prod_{i=1}^{\ell} C_{i, \text{sk}[i]}^{(v)} \right) \bar{t}^{(v)} = \tilde{s}^{(v)} \left(\prod_{i=1}^{\ell} \tilde{A}_{i, \text{sk}[i]}^{(v)} \right) \tilde{t}^{(v)} p_{zt} \pmod{x_0}.$$

This corresponds to a zero-tested encoding of:

$$v_v = \mathbf{s}^{(v)} \cdot \left(\prod_{i=1}^{\ell} A_{i, \text{sk}[i]}^{(v)} \right) \cdot \mathbf{t}^{(v)} = \mathbf{s}^* \left(\prod_{i=1}^{\ell} \alpha_{i, \text{sk}[i]}^{(v)} B_{i, \text{sk}[i]} \right) \mathbf{t}^* \pmod{g}$$

From the condition satisfied by the $\alpha_{i,b}^{(v)}$'s, the values are independent from v . Therefore, each party v will extract from $z^{(v)}$ the same session-key, as required.

5.3 Additional safeguard: straddling sets

As an additional safeguard one can use the straddling set systems from [BGK⁺14]. Like the multiplicative bundling scalars $\alpha_{i,b}^{(u)}$, this prevents the adversary from switching the secret-key bits between the k repetitions. Additionally, the straddling set system prevents the adversary from mixing the matrices $\tilde{A}_{i,0}^{(u)}$ and $\tilde{A}_{i,1}^{(u)}$, since in that case the matrices are encoded at a different level set.

6 The Cheon et al. Attack and its Generalization using Tensor Products

At Eurocrypt 2015, Cheon *et al.* described in [CHL⁺15] a total break of the basic key-exchange protocol of CLT13. The attack was then extended and applied to several constructions based on CLT13. In this section, we argue that the complexity of the Cheon *et al.* attack against our construction is $\Omega(m^{2k-1})$, where m is the matrix dimension and k the number of repetitions. Therefore, the Cheon *et al.* attack is prevented by using a large enough k .

6.1 The original Cheon et al. attack

The Cheon et al. attack [CHL⁺15] against CLT13 consists in multiplying the level-one encodings of zero available in the original CLT13 by other encodings to obtain top-level encodings of zero, which are then zero-tested to provide equations over \mathbb{Z} instead of \mathbb{Z}_{x_0} ; the attack recovers all secret primes p_1, \dots, p_n from the public parameters.

More precisely, given level-one encodings of zero a_i (for $i \in [n]$), level-one encodings c_j (for $j \in [n]$) and a level-zero encoding b_0 , assuming only $\kappa = 2$ levels, the attacker defines $w_{i,j} := [a_i \cdot b_0 \cdot c_j \cdot p_{zt}]_{x_0}$ and $w'_{i,j} := [a_i \cdot c_j \cdot p_{zt}]_{x_0}$, and then computes two matrices $\mathbf{W}_0, \mathbf{W}_1 \in \mathbb{Z}_{x_0}^{n \times n}$ whose entries are defined as $\mathbf{W}_0[i, j] := w_{i,j}$ and $\mathbf{W}_1[i, j] := w'_{i,j}$.

From the definition of p_{zt} , we obtain $w_{i,j} = \sum_{k=1}^n a_{i,k} b_{0,k} c_{j,k} \xi_k \pmod{x_0}$, where $a_{i,k}$, $b_{0,k}$ and $c_{j,k}$ represent the modulo p_k component of the numerator of a_i , b_0 , and c_j respectively, and ξ_k gathers the terms from p_{zt} . Since we obtain encodings of zero, the equation also holds over \mathbb{Z} , hence it can be rewritten as

$$w_{i,j} = [a_{i,1} \ a_{i,2} \ \dots \ a_{i,n}] \cdot \begin{bmatrix} \xi_1 b_{0,1} & & & \\ & \xi_2 b_{0,2} & & \\ & & \ddots & \\ & & & \xi_n b_{0,n} \end{bmatrix} \cdot \begin{bmatrix} c_{j,1} \\ c_{j,2} \\ \vdots \\ c_{j,n} \end{bmatrix}.$$

Therefore we can write $\mathbf{W}_0 = \mathbf{A}\mathbf{B}_0\mathbf{C}$, where the rows of \mathbf{A} are the vectors in the left (for $i \in [n]$), \mathbf{B}_0 is the diagonal matrix in the middle, and \mathbf{C} is the matrix whose columns are the vectors in the right (for $j \in [n]$). By the same argument, $\mathbf{W}_1 = \mathbf{A}\mathbf{B}_1\mathbf{C}$, where $\mathbf{B}_1 = \text{diag}(\xi_1, \dots, \xi_n)$. Thus, the attacker can compute over \mathbb{Q} :

$$\mathbf{W} = \mathbf{W}_0\mathbf{W}_1^{-1} = (\mathbf{A}\mathbf{B}_0\mathbf{C})(\mathbf{A}\mathbf{B}_1\mathbf{C})^{-1} = \mathbf{A}\mathbf{B}_0\mathbf{B}_1^{-1}\mathbf{A}^{-1}$$

The eigenvalues of \mathbf{W} are the same as those of $\mathbf{B}_0\mathbf{B}_1^{-1}$ and are equal to $b_{0,1}, \dots, b_{0,n}$. The attacker can therefore recover the $b_{0,i}$'s and eventually the primes p_i 's by computing gcd's. We provide an implementation of the attack in Appendix E.1.

Variante modulo q . Since the eigenvalues $b_{0,i}$ are small, they can be computed modulo a small prime q of size η bits. Therefore it suffices to compute the matrix $\mathbf{W}_0\mathbf{W}_1^{-1}$ modulo q only. The characteristic polynomial of $\mathbf{W}_0\mathbf{W}_1^{-1}$ is computed modulo q and then factored to recover the $b_{0,i}$'s modulo q . Experimentally, computing the two matrices \mathbf{W}_0 and \mathbf{W}_1 takes time $\mathcal{O}(n^{3.5})$. Computing the full $\mathbf{W}_0\mathbf{W}_1^{-1}$ over \mathbb{Q} takes time $\mathcal{O}(n^6)$, whereas computing $\mathbf{W}_0\mathbf{W}_1^{-1} \pmod q$ and recovering the eigenvalues modulo q takes only $\mathcal{O}(n^3)$. Therefore the variant attack modulo q is much faster, and its dominant cost is to compute the two matrices \mathbf{W}_0 and \mathbf{W}_1 . We also provide an implementation of the variant in Appendix E.1.

6.2 Generalization to matrices

The previous attack was extended to matrices of encodings in [CGH⁺15]. More precisely, the extended attack defines an *attack set of dimension d* as 3 sets of matrices $\mathcal{A} := \{\mathbf{A}_i \in \mathbb{Z}_{x_0}^{d \times d} : i \in [nd]\}$, $\mathcal{B} = \{\mathbf{B}_\sigma \in \mathbb{Z}_{x_0}^{d \times d} : \sigma \in \{0, 1\}\}$, and $\mathcal{C} := \{\mathbf{C}_j \in \mathbb{Z}_{x_0}^{d \times d} : j \in [nd]\}$, and two vectors $\mathbf{s} \in \mathbb{Z}_{x_0}^d$ and $\mathbf{t} \in \mathbb{Z}_{x_0}^d$ such that the value $w_{i,\sigma,j} := \mathbf{s}\mathbf{A}_i\mathbf{B}_\sigma\mathbf{C}_j\mathbf{t} \pmod{x_0}$ is a zero-tested top-level encoding of zero. The attack then proceeds as previously by computing two matrices $\mathbf{W}_\sigma \in \mathbb{Z}^{nd \times nd}$ (for $\sigma \in \{0, 1\}$) whose each entry is defined as $\mathbf{W}_\sigma[i, j] = w_{i,\sigma,j}$, then computing the matrix $\mathbf{W} := \mathbf{W}_0\mathbf{W}_1^{-1}$ over \mathbb{Q} . As previously we can write:

$$\mathbf{W}_\sigma = \mathbf{A}\bar{\mathbf{B}}_\sigma\mathbf{C}$$

where the matrix $\bar{\mathbf{B}}_\sigma$ of dimension nd is block-diagonal with the matrices $\xi_i \cdot (\mathbf{B}_\sigma \pmod{p_i}) \in \mathbb{Z}^{d \times d}$ on the diagonal. We obtain:

$$\mathbf{W} = \mathbf{W}_0\mathbf{W}_1^{-1} = \mathbf{A}\bar{\mathbf{B}}_0\bar{\mathbf{B}}_1^{-1}\mathbf{A}^{-1}$$

The characteristic polynomial $f(X)$ of \mathbf{W} is the same as the characteristic polynomial of $\bar{\mathbf{B}}_0\bar{\mathbf{B}}_1^{-1}$, which is the product of the n characteristic polynomials $f_i(X)$ of the matrices $\bar{\mathbf{B}}_i = (\mathbf{B}_0 \pmod{p_i}) \cdot (\mathbf{B}_1 \pmod{p_i})^{-1}$. By the Cayley-Hamilton theorem, we must have $f_i(\bar{\mathbf{B}}_i) = 0$ for all $1 \leq i \leq n$. This implies $f_i(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1} \pmod{x_0}) = 0 \pmod{p_i}$. Therefore, if the polynomials $f_i(X)$ are irreducible, they can be recovered by computing $f(X)$ and factoring $f(X)$ into irreducible polynomials. Then each prime p_i can be recovered by computing the gcd of the entries of $\mathbf{M}_i = f_i(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1} \pmod{x_0})$ with x_0 . We provide the source code of the attack in Appendix E.2.

Alternatively, if the polynomials $f_i(X)$ are not irreducible, one can still factor $f(X)$ into monic irreducible factors $f'_1, \dots, f'_N \in \mathbb{Q}[X]$. Then for $k \in [N]$, the attacker defines $F_k := f/f'_k \in \mathbb{Q}[X]$ and $G_k = F_k \cdot d_k \in \mathbb{Z}[X]$, where d_k is the common denominator of F_k 's coefficients. As previously, by the Cayley-Hamilton theorem we have that $G_k(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1} \pmod{x_0}) = 0$ modulo all primes except one, and therefore the remaining prime p_i can be recovered by computing the gcd of the entries of $\mathbf{M}_k = G_k(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1}) \pmod{x_0}$ with x_0 .

Variation without $B_0 B_1^{-1} \bmod x_0$. We describe an alternative attack in which one does not need to compute the matrix $B_0 B_1^{-1} \bmod x_0$; only the matrix W is used. This alternative attack will be useful in the context of the tensoring attack from [CLLT17]; in that case we will not have to compute tensors explicitly as in [CLLT17], which makes the attack slightly simpler.

Our variant attack is as follows. We define the polynomials $G_k(X)$ as previously, and instead of computing the matrices $M_k = G_k(B_0 \cdot B_1^{-1}) \bmod x_0$, we compute the matrices:

$$M'_k = G_k(W) \cdot W_0 \bmod x_0$$

Then as previously each prime p_i can be recovered by computing the gcd of the entries of M'_k with x_0 . Namely we have:

$$\begin{aligned} M'_k &= G_k(A \bar{B}_0 \bar{B}_1^{-1} A^{-1}) \cdot W_0 \pmod{x_0} \\ &= A G_k(\bar{B}_0 \cdot \bar{B}_1^{-1}) A^{-1} A \bar{B}_0 C \pmod{x_0} \\ &= A G_k(\bar{B}_0 \cdot \bar{B}_1^{-1}) \bar{B}_0 C \pmod{x_0} \end{aligned}$$

The characteristic polynomial of W is the same as the one of $\bar{B}_0 \cdot \bar{B}_1^{-1}$. Therefore, by the Cayley-Hamilton theorem, all the blocks on the diagonal of $G_k(\bar{B}_0 \cdot \bar{B}_1^{-1})$ are zero except the block corresponding to $\tilde{B}_i = (B_0 \bmod p_i) \cdot (B_1 \bmod p_i)^{-1}$ for some i . When multiplying by \bar{B}_0 , such block is multiplied by $\xi_i \cdot (B_0 \bmod p_i) \in \mathbb{Z}^{d \times d}$. Therefore, the resulting block is a multiple of ξ_i , while all the other blocks are zero. This implies that all entries of M'_k are multiple of ξ_i , which is a multiple of all primes except p_i ; this enables to recover p_i by gcd. We also provide in Appendix E.2 an implementation of this variant.

6.3 Application to our construction

Our attack proceeds as follows. For simplicity we consider the case of 3 users only; the generalization to N users is straightforward. As in (25) we use $\text{sk} \in \{0, 1\}^\ell$ to compute the product matrices in each row, with:

$$\text{sk} = \underbrace{(\text{sk}^{(1)}, \text{sk}^{(2)}, \text{sk}^{(3)})}_{\text{First repetition}}, \dots, \underbrace{(\text{sk}^{(1)}, \text{sk}^{(2)}, \text{sk}^{(3)})}_{k\text{-th repetition}}$$

Since the session key must be the same in the first two rows, we obtain by difference a zero-tested top-level encoding of zero:

$$\omega = \bar{s}^{(1)} \prod_{i=1}^{\ell} C_{i, \text{sk}[i]}^{(1)} \bar{t}^{(1)} - \bar{s}^{(2)} \prod_{i=1}^{\ell} C_{i, \text{sk}[i]}^{(2)} \bar{t}^{(2)} \pmod{x_0}. \quad (26)$$

In principle, to produce the attack sets \mathcal{A} , \mathcal{B} , and \mathcal{C} needed for the extended Cheon *et al.* attack, one should find a partition of sk so that its first bits affect only the first matrices, the middle bits affect the matrices in the middle, and the last bits affect only the last matrices. However, the k repetitions in sk prevents us from constructing such independent sets, because flipping any bit of sk forces to flip the other $k - 1$ corresponding bits (otherwise, subtracting two rows does not result in a encoding of zero). Therefore to generate the attack sets, we use the tensoring technique from [CLLT17] to group the matrices that depend on the same input bits.

More precisely, given three secrets $\text{sk}^{(1)}$, $\text{sk}^{(2)}$, $\text{sk}^{(3)}$ and a given row u , we let the matrices $A_i := \prod_{j=1}^{\mu} C_{\phi(i,1,j), \text{sk}_j^{(1)}}^{(u)}$, $B_i := \prod_{j=1}^{\mu} C_{\phi(i,2,j), \text{sk}_j^{(2)}}^{(u)}$, $C_i := \prod_{j=1}^{\mu} C_{\phi(i,3,j), \text{sk}_j^{(3)}}^{(u)}$, where the function

$$\phi(r, v, j) = (r - 1)N\mu + (v - 1)\mu + j - 1$$

is used to access the matrices, where $1 \leq r \leq k$ is the repetition index, $1 \leq v \leq N$ is the user index, and $1 \leq j \leq \mu$ is the bit index in $\text{sk}^{(v)}$. Therefore A_i is the i -th matrix of the first user computed

using secret $\mathbf{sk}^{(1)}$ on row u , \mathbf{B}_i is the i -th matrix of the second user, and likewise for \mathbf{C}_i . Thus, given the number of repetitions k , the product of all matrices with respect to $\mathbf{sk}^{(1)}$, $\mathbf{sk}^{(2)}$, and $\mathbf{sk}^{(3)}$ on row u can be written as $\prod_{i=1}^k \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i$, where \mathbf{C}_k is considered to be an m -dimensional column vector (that is obtained by multiplying by the right bookend vector) and all the other factors are $m \times m$ matrices.

Using the same tensoring technique as in [CLLT17], we show that this product can be written as \mathbf{ABC} where \mathbf{A} is an $m \times m^{2k-1}$ matrix depending only on $\mathbf{sk}^{(1)}$, \mathbf{B} is an $m^{2k-1} \times m^{2k-1}$ matrix depending only of $\mathbf{sk}^{(2)}$, and \mathbf{C} is an $m^{2k-1} \times 1$ matrix (column vector) depending only of $\mathbf{sk}^{(3)}$; see Appendix B for the details. Since we must compute the difference of two rows as in (26), we must consider matrices of dimension $d = 2m^{2k-1}$ instead of m^{2k-1} . Therefore, the final matrices \mathbf{W}_0 and \mathbf{W}_1 from the Cheon *et al.* attack have dimension $d' = nd = 2nm^{2k-1}$. Note that the case for $N > 3$ users is analogous, since we can simply merge multiple users into the same secret-key, and proceed as if there were only three users. This implies that the Cheon *et al.* attack against our construction has complexity $\Omega(m^{2k-1})$; it is therefore prevented by taking a large enough k . We provide a basic implementation of the attack in Appendix E.3, including the variant without $\mathbf{B}_0 \mathbf{B}_1^{-1} \bmod x_0$ described in the previous section; in the latter case, the attack recovers the primes p_i 's from the matrices \mathbf{W}_0 and \mathbf{W}_1 only, without computing tensors explicitly.

6.4 Practical complexity

We have implemented the previous attack and verified for small n, m, k that it requires a minimal dimension $d' = 2nm^{2k-1}$ to recover the prime factors p_i . To estimate the practical complexity of the attack, we consider only the cost of constructing the matrices \mathbf{W}_0 and \mathbf{W}_1 . While the attack also requires to invert \mathbf{W}_1 , find the characteristic polynomial, and factor it over \mathbb{Z} , these operations could probably be performed more efficiently by working modulo a small prime q .¹

The matrices \mathbf{W}_0 and \mathbf{W}_1 have $d'^2 = 4n^2 m^{4k-2}$ elements, and the production of each element requires at least one vector-matrix multiplication modulo x_0 in dimension m , which takes at least m^2 multiplications modulo x_0 . In practice, the number of clock cycles to compute a modular multiplication of γ bit integers is well approximated by

$$T_{mul}(\gamma) = 0.5 \cdot \gamma \cdot \log^2 \gamma,$$

where $\gamma = n \cdot \eta$ in our scheme. Therefore the complexity of the Cheon *et al.* attack against our scheme is lower-bounded by

$$T_{Cheon}(\eta, n, m, k) = 4\eta \cdot n^3 \cdot m^{4k} \log^2(\eta \cdot n)$$

and we require $T_{Cheon}(\eta, n, m, k) > 2^\lambda$.

7 Optimizations and Implementation

In this section we describe a few optimizations in order to obtain a concrete implementation of our construction from Section 5.

7.1 Encoding without randomness

Our main optimization is to encode the matrices $\mathbf{A}_{i,b}^{(u)}$ from (22) without any additional randomness, while relying on the intrinsic randomness of the matrices $\mathbf{A}_{i,b}^{(u)}$ and on Kilian's randomization on

¹ At least this is true in the original Cheon *et al.* attack, where the eigenvalues can be computed modulo a small prime q ; see Section 6.1. Using the same approach in the extended attack with the Cayley-Hamilton theorem seems less straightforward.

the encoding side. Recall that the plaintext space of CLT13 is \mathbb{Z}_g where $g = \prod_{i=1}^n g_i$. Each matrix entry $a \in \mathbb{Z}_g$ is therefore encoded as an integer $c \in \mathbb{Z}_{x_0}$ with $c \equiv a_i \pmod{p_i}$ where $a_i = a \bmod g_i$, instead of $c \equiv a_i + r_i g_i \pmod{p_i}$; that is, we take $r_i = 0$ for all $1 \leq i \leq n$.

Moreover, to decrease the growth of the noise in the matrix product computation, we do not generate the non-zero plaintext entries of $\mathbf{A}_{i,b}^{(u)}$ uniformly in \mathbb{Z}_g ; instead we generate entries only with small components modulo each of the g_i 's. More precisely, for each slot modulo g_i , we use the discrete Gaussian distribution D_σ with mean 0 and parameter σ , as considered in Section 3.4. We still ensure that the matrices $\mathbf{A}_{i,b}^{(u)}$ are invertible, by rejection sampling. The bundling scalars $\alpha_{i,b}^{(u)} \in \mathbb{Z}_g$ are generated such that $\alpha_{i,b}^{(u)} = \pm 1 \pmod{g_j}$ for all $1 \leq j \leq n$, so that they do not increase the noise. Note that this optimization is only applied to the matrices $\mathbf{A}_{i,b}^{(u)}$. For the bookend vectors, the components are still uniformly generated in \mathbb{Z}_g , and encoded with CLT13 with randoms of size ρ bits. Letting α be the size of the g_i 's, for simplicity we take $\rho = \alpha$. Therefore the encoded bookend vectors have $\alpha \cdot (2m/3) + \rho \cdot m = 5\alpha m/3$ bits of entropy on each slot.

Without this optimization, when multiplying the encoding of ℓ matrices $\mathbf{A}_{i,b}^{(u)}$, the bitsize of the resulting product modulo each p_i would grow as $\ell \cdot (\alpha + \rho)$, where $\alpha + \rho \simeq 2\lambda$ in the original CLT13. Instead, when multiplying the matrices with $\rho = 0$ and small entries modulo each g_i , experimentally the bitsize of the entries can be approximated by:

$$\log_2 \left\| \prod_{i=1}^{\ell} \mathbf{A}_{i, \text{sk}[i]}^{(u)} \right\|_{\infty} \simeq 0.25 \cdot \ell \cdot h_\sigma \cdot \log_2 m \quad (27)$$

where h_σ is the entropy of D_σ . Equation (27) gives a lower bound for the size η of the primes p_i . In practice, since we take a parameter σ such that $h_\sigma \ll 2\lambda$, we can take a much smaller η than in the original CLT13 for the same degree ℓ .

These two optimizations (encoding without randomness and small components modulo g_i 's) correspond to the setting considered in the GCD attacks in Section 3.4, where the matrix entries before Kilian's randomization are generated using the discrete Gaussian distribution D_σ for a small σ . More precisely, to prevent the GCD attack considered in Section 3.4, we must ensure that $T_{GCD}(m, \gamma, h_\sigma, n) \geq 2^\lambda$ where $T_{GCD}(m, \gamma, h_\sigma, n)$ is given by (12). Since the matrices $\mathbf{A}_{i,b}^{(u)}$ are block diagonal with 3 blocks of size $m/3$ each, we must take $\rho = m^2 h_\sigma / 3$ instead of $\rho = m^2 h_\sigma$ in (12).

Similarly, for the bookend vectors, thanks to Kilian's randomization we can take a smaller α than in the original CLT13. More precisely, since the encoded bookend vectors have $5\alpha m/3$ bits of entropy on each slot, we require $T_{CN}(\rho, \gamma) > 2^\lambda$ with $\rho = 5\alpha m/3$, where $T_{CN}(\rho, \gamma)$ is given by (9).

7.2 Number of matrices per level

Instead of taking only two matrices $\mathbf{A}_{i,0}^{(u)}$, $\mathbf{A}_{i,1}^{(u)}$ for each $1 \leq i \leq \ell$, we can take 2^τ matrices for each i . In that case, the secret key of each user has μ words of τ bits, where each word selects one of the 2^τ matrices; the size of the secret-key is therefore $\mu \cdot \tau$ bits. For the same secret-key size, one can therefore divide the total degree ℓ by a factor τ , but the number of encoded matrices is multiplied by a factor $2^\tau/\tau$. In order to minimize the size of the public parameters, we use $\tau = 3$. Note that the straddling set system from [BGK⁺14] is easily adapted for $\tau > 1$.

7.3 Other attacks

From the optimization of Section 7.1, we can take a relatively small η even for a large degree ℓ . Therefore we must consider the ECM factoring attack.

ECM factoring. The asymptotic complexity of the ECM [Len87] to extract a prime factor p is

$$\exp\left((\sqrt{2} + o(1))\sqrt{\ln p \ln \ln p}\right).$$

To prevent the factorization of $x_0 = \prod_i p_i$, the size η of the prime p_i 's must therefore satisfy $\eta = \omega(\lambda^2)$. From recent factoring records, we used the following formula to estimate the running time of the ECM (in number of clock cycles):

$$T_{ECM}(\eta, \gamma) = 2^{-20} \exp\left(\sqrt{2\eta(\ln \eta)(\ln 2)}\right) \cdot \gamma$$

and we require $T_{ECM}(\eta, \gamma) \geq 2^\lambda$.

Meet-in-the-middle attack. There is a meet-in-the-middle attack on the secret key of each user with length $\mu \cdot \tau$ bits, with complexity $\mathcal{O}(2^{\mu\tau/2})$. The complexity is at least

$$M(m, \gamma) \cdot 2^{\mu\tau/2},$$

where $M(m, \gamma)$ is the time it takes to multiply $m \times m$ matrices with entries of size γ . We ensure $M(m, \gamma) \cdot 2^{\mu\tau/2} \geq 2^\lambda$.

7.4 Concrete parameters and implementation results

In this section we propose concrete parameters for our key-exchange construction with $N = 4$ parties. These parameters are generated so that all known attacks have running time $\geq 2^\lambda$. We provide the parameters in Table 3. The bookend vectors are encoded with noise $\rho = \alpha$. The total number of encoded matrices is $2^\tau \cdot \ell \cdot N$ with $\tau = 3$, with a total degree $\ell = \mu \cdot k \cdot N$. Therefore, the total number of CLT13 encodings is $N_{CLT13} \simeq 2^\tau \cdot \ell \cdot N \cdot m^2$. The size of the secret key is $\tau\mu = 3\mu$. We provide in Table 4 the security assessment against lattice attacks.

	λ	η	m	n	μ	σ	α	k	$\gamma = n \cdot \eta$	ℓ	N_{CLT13}	params
Small	52	220	12	70	14	0.51	10	2	$15 \cdot 10^3$	112	$5.2 \cdot 10^5$	0.9 GB
Medium	62	270	12	330	19	0.57	11	2	$89 \cdot 10^3$	152	$7.0 \cdot 10^5$	7.8 GB
Large	72	335	15	613	24	0.51	15	2	$205 \cdot 10^3$	192	$13.8 \cdot 10^5$	35.4 GB
Extra	80	530	18	940	28	0.51	15	3	$498 \cdot 10^3$	336	$34.8 \cdot 10^5$	216.9 GB

Table 3. Concrete parameters for a 4-party key-exchange.

	λ	Lattice dim. t	Hermite factor	Algorithm	Running time
Small	52	1680	1.021^t	LLL	2^{53}
Medium	62	7920	1.011^t	BKZ-60	2^{65}
Large	72	18390	1.010^t	BKZ-80	2^{76}
Extra	80	33840	1.009^t	BKZ-100	2^{86}

Table 4. Security of our parameter sets against lattice attacks. The lattice dimension is $t = 2nm$. The estimated running time is in number of clock cycles.

The main difference with the original (insecure) key-exchange protocol based on CLT13 is that we get a much larger public parameter size; for $\lambda = 62$ bits of security, we need 8 GB of public

parameters, instead of 70 MB originally. However our construction would be completely unpractical without Kilian’s randomization on the encoding side. Namely for $\lambda = 62$ and a degree $\ell = 152$, one would need primes p_i of size $\eta \simeq (\alpha + \rho) \cdot \ell \simeq 2.1 \cdot 10^4$ with $\alpha = 80$ and $\rho = 62$ as in [CLT13]. Since $\gamma = \omega(\eta^2 \log \lambda)$ in [CLT13], one would need $\gamma \simeq 1.2 \cdot 10^9$. With $N_{CLT13} = 7 \cdot 10^5$, that would require 100 TB of public parameter size. Hence Kilian’s randomization on the encoding side provides a reduction of the public parameter size by a factor $\simeq 10^4$.

We have implemented the key-exchange protocol in SAGE [S⁺17] and executed it on a machine with processor Intel i5-6500 (3.20GHz), 8 GB of RAM, and Ubuntu 16.04.5 LTS. The execution times are shown in Table 5. For $\lambda = 62$ we ran the experiment one row at time so that only 2 GB were needed instead of 8 GB. We could not run the Large and Extra instantiations ($\lambda = 72$ and $\lambda = 80$) because of the huge parameter size. While the Setup time is significant, the Publish and KeyGen times remain reasonable.

	Setup (once)	Publish (per party)	KeyGen (per party)
Small	32 min 10 s	26 s	11 s
Medium	15 h 44 min	39 s	47 s

Table 5. Timings for a 4-party key-exchange.

8 Conclusion

We have shown that Kilian’s randomization “on the encoding side” can bring orders of magnitude efficiency improvements for iO based constructions when instantiated with CLT13 multilinear maps. As an application, we have described the first concrete implementation of multipartite DH key exchange secure against existing attacks. The main advantage of Kilian’s randomization is that it can be applied essentially for free in any existing implementation; for example it could be easily integrated in the 5Gen framework [LMA⁺16] for experimenting with program obfuscation constructions.

References

- [AGIS14] Prabhanjan Vijendra Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding barrington’s theorem. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS*, pages 646–658. ACM, 2014.
- [Bar86] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1 . In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 1–5, 1986.
- [BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 10–24, 2016.
- [Ber03] Daniel J. Bernstein. Fast multiplication and its applications, 2003.
- [BGK⁺14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *Advances in Cryptology – EUROCRYPT 2014 - Proceedings*, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
- [BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT (2)*, volume 9666 of *LNCS*, pages 764–791. Springer, 2016.
- [BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.
- [CGH⁺15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrede Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 247–266. Springer, 2015.

- [CGH17] Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. In *Advances in Cryptology - EUROCRYPT 2017 - Proceedings, Part III*, pages 278–307, 2017.
- [CHL⁺15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 3–12. Springer, 2015.
- [CLLT16] Jean-Sébastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In *Advances in Cryptology - CRYPTO 2016 - Proceedings, Part II*, pages 607–628, 2016.
- [CLLT17] Jean-Sébastien Coron, Moon Sung Lee, Tancrede Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. In *Public-Key Cryptography - PKC 2017 - Proceedings, Part I*, pages 41–58, 2017.
- [CLT13] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO*, volume 8042 of *LNCS*, pages 476–493. Springer, 2013.
- [CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Advances in Cryptology - ASIACRYPT 2011 - Proceedings*, pages 1–20, 2011.
- [CN12] Yuanmi Chen and Phong Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 502–519. Springer, 2012.
- [CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In *Advances in Cryptology - CRYPTO 2018 - Proceedings, Part II*, pages 577–607, 2018.
- [DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.
- [GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 1–17. Springer, 2013.
- [GGH⁺13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49. IEEE Computer Society, 2013.
- [GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC*, volume 9015 of *LNCS*, pages 498–527, 2015.
- [GGM16] Steven D. Galbraith, Shishay W. Gebregiyorgis, and Sean Murphy. Algorithms for the approximate common divisor problem. *LMS Journal of Computation and Mathematics*, 19(A):58–72, 2016.
- [HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In *Advances in Cryptology - EUROCRYPT 2016 - Proceedings, Part I*, pages 537–565, 2016.
- [HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *CRYPTO 2011*, pages 447–464, 2011.
- [Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 20–31, 1988.
- [Lau04] Alan J. Laub. *Matrix Analysis For Scientists And Engineers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004.
- [Len87] H. W. Lenstra. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.
- [LMA⁺16] Kevin Lewi, Alex J. Malozemoff, Daniel Apon, Brent Carmer, Adam Foltzer, Daniel Wagner, David W. Archer, Dan Boneh, Jonathan Katz, and Mariana Raykova. 5gen: A framework for prototyping applications using multilinear maps and matrix branching programs. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, pages 981–992, 2016.
- [LS14] Hyung Tae Lee and Jae Hong Seo. Security analysis of multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2014 - Proceedings, Part I*, pages 224–240, 2014.
- [MSW14] Eric Miles, Amit Sahai, and Mor Weiss. Protecting obfuscation against arithmetic attacks. Cryptology ePrint Archive, Report 2014/878, 2014. Available at <https://eprint.iacr.org/2014/878>.
- [MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *Advances in Cryptology - CRYPTO 2016 - Proceedings, Part II*, pages 629–658, 2016.
- [MZ18] Fermi Ma and Mark Zhandry. The mmap strikes back: Obfuscation and new multilinear maps immune to CLT13 zeroizing attacks. In *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part II*, pages 513–543, 2018.
- [NS97] Phong Q. Nguyen and Jacques Stern. Merkle-hellman revisited: A cryptanalysis of the qu-vanstone cryptosystem based on group factorizations. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 198–212, 1997.

- [NS01] Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*, pages 146–180, 2001.
- [PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO (1)*, volume 8616 of *LNCS*, pages 500–517. Springer, 2014.
- [S⁺17] W. A. Stein et al. *Sage Mathematics Software (Version 8.0)*. The Sage Development Team, 2017. <http://www.sagemath.org>.
- [SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing, STOC '14*, pages 475–484, New York, NY, USA, 2014. ACM.

A Kronecker product of matrices

For any two matrices $\mathbf{A} \in R^{m \times n}$ and $\mathbf{B} \in R^{p \times q}$, we define the *Kronecker product* (or tensor product) of \mathbf{A} and \mathbf{B} as the block matrix $\mathbf{A} \otimes \mathbf{B} \in R^{(mp) \times (nq)}$ given by:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}, \quad \text{where } \mathbf{A} = (a_{ij}).$$

We recall the following property of the Kronecker product [Lau04, Ch. 13]. Given a matrix $\mathbf{C} \in R^{n \times m}$, we let $\mathbf{c}_i \in R^n$, $i = 1, \dots, m$ be its column vectors, so that $\mathbf{C} = [\mathbf{c}_1, \dots, \mathbf{c}_m]$. We denote by $\text{vec}(\mathbf{C})$ the column vector of dimension mn formed by stacking the columns \mathbf{c}_i of \mathbf{C} on top of one another:

$$\text{vec}(\mathbf{C}) = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_m \end{bmatrix} \in R^{mn}.$$

Using the fact that $\text{vec}(\mathbf{x}\mathbf{y}^T) = \mathbf{y} \otimes \mathbf{x}$ for any \mathbf{x}, \mathbf{y} , we obtain that for any three matrices \mathbf{A}, \mathbf{B} , and \mathbf{C} for which the matrix product $\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C}$ is defined:

$$\text{vec}(\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C}) = (\mathbf{C}^T \otimes \mathbf{A}) \cdot \text{vec}(\mathbf{B})$$

B The tensoring attack

We show that the product $\prod_{i=1}^k \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i$ from Section 6.3 can be written as \mathbf{ABC} , using the tensoring technique from [CLLT17]. The base case with $k = 1$ is clearly true. For $k \geq 2$, we use induction to write

$$\prod_{i=1}^k \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i = \mathbf{A}_1 \mathbf{B}_1 \mathbf{C}_1 \left(\prod_{i=2}^k \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i \right) = \mathbf{A}_1 \mathbf{B}_1 \mathbf{C}_1 \tilde{\mathbf{A}} \tilde{\mathbf{B}} \tilde{\mathbf{C}}$$

with $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$, and $\tilde{\mathbf{C}}$ having dimensions $m \times m^{2(k-1)-1}$, $m^{2(k-1)-1} \times m^{2(k-1)-1}$, and $m^{2(k-1)-1} \times 1$, respectively. Then, since $\tilde{\mathbf{C}}$ is a column vector, we have

$$\begin{aligned} \mathbf{A}_1 \mathbf{B}_1 \mathbf{C}_1 \tilde{\mathbf{A}} \tilde{\mathbf{B}} \tilde{\mathbf{C}} &= \mathbf{A}_1 \mathbf{B}_1 \text{vec}(\mathbf{C}_1 \tilde{\mathbf{A}} \tilde{\mathbf{B}} \tilde{\mathbf{C}}) = \mathbf{A}_1 \mathbf{B}_1 (\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) \text{vec}(\tilde{\mathbf{A}} \tilde{\mathbf{B}}) \\ &= \mathbf{A}_1 \mathbf{B}_1 (\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) \text{vec}(\mathbf{I}_m \tilde{\mathbf{A}} \tilde{\mathbf{B}}) \\ &= \mathbf{A}_1 \mathbf{B}_1 (\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) (\tilde{\mathbf{B}}^T \otimes \mathbf{I}_m) \text{vec}(\tilde{\mathbf{A}}) \\ &= \text{vec} \left(\mathbf{A}_1 \mathbf{B}_1 (\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) (\tilde{\mathbf{B}}^T \otimes \mathbf{I}_m) \text{vec}(\tilde{\mathbf{A}}) \right) \\ &= (\text{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1) \text{vec} \left(\mathbf{B}_1 (\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) (\tilde{\mathbf{B}}^T \otimes \mathbf{I}_m) \right) \\ &= (\text{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1) ((\tilde{\mathbf{B}}^T \otimes \mathbf{I}_m)^T \otimes \mathbf{B}_1) \text{vec}(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) \\ &= (\text{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1) ((\tilde{\mathbf{B}} \otimes \mathbf{I}_m) \otimes \mathbf{B}_1) \text{vec}(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) \end{aligned}$$

Defining $\mathbf{A} = \text{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1$, $\mathbf{B} = (\tilde{\mathbf{B}} \otimes \mathbf{I}_m) \otimes \mathbf{B}_1$, and $\mathbf{C} = \text{vec}(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)$, we obtain

$$\prod_{i=1}^k \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i = \mathbf{A} \mathbf{B} \mathbf{C}.$$

Because $\text{vec}(\tilde{\mathbf{A}})^T$ has dimension $1 \times m^{2(k-1)}$, the dimension of \mathbf{A} is $m \times m^{2(k-1)+1} = m \times m^{2k-1}$. Furthermore, the dimension of $\tilde{\mathbf{B}} \otimes \mathbf{I}_m$ is $m^{2(k-1)} \times m^{2(k-1)}$, therefore, the dimension of \mathbf{B} is $m^{2k-1} \times m^{2k-1}$. Similarly, the dimension of \mathbf{C} is $m^{2k-1} \times 1$, so the result holds.

C Implementation of the GCD Attacks

C.1 Source Code of the Chen-Nguyen Attack

```
import sage.rings.polynomial.polynomial_ring as pring

def genXi(rho, eta, gam, p):
    return p*ZZ.random_element(2^(gam-eta))+ZZ.random_element(2^rho)

def multiPointEval(f, li, R):
    n, x = len(li), R.0
    if n==1: return [f(x=li[0])]
    li1, li2 = li[:n//2], li[n//2:]
    f1 = f.quo_rem(prod((x-xi for xi in li1))) [1]
    f2 = f.quo_rem(prod((x-xi for xi in li2))) [1]
    return multiPointEval(f1, li1, R) + multiPointEval(f2, li2, R)

def attackGCDDMultiPoint(rho=12, eta=1000, gam=40000, verbose=True):
    p = random_prime(2^eta, lbound=2^(eta-1), proof=False)
    if verbose: print p
    x0 = p*prod([random_prime(2^eta, lbound=2^(eta-1), proof=False)
                for i in range(gam/eta-1)])
    R = pring.PolynomialRing_dense_mod_n(Integers(x0), 'x')
    x = R.0
    c = genXi(rho, eta, gam, p)

    t = cputime(subprocesses=True)
    f = prod((x+i-c for i in range(2^(rho/2))))
    ev = multiPointEval(f, range(0, 2^rho, 2^(rho/2)), R)
    pp = gcd(x0, prod(ev))
    if verbose: print pp
    return cputime(t)
```

C.2 Source Code of the Lee-Seo Attack

```
def attackGCDDMaskedMultiPoint(rho=10, eta=100, gam=200, verbose=True):
    p = random_prime(2^eta, lbound=2^(eta-1), proof=False)
    print "p=", p
    x0 = p*prod([random_prime(2^eta, lbound=2^(eta-1), proof=False)
                for i in range(gam/eta-1)])
    R = pring.PolynomialRing_dense_mod_n(Integers(x0), 'x')
    x = R.0
    z = Integers(x0).random_element()

    L1 = [genXi(rho, eta, gam, p)*z for i in range(2^(rho//2))]
    L2 = [genXi(rho, eta, gam, p)*z for i in range(2^(rho//2))]

    t = cputime(subprocesses=True)
    f = prod((x-c for c in L1))
```

```

ev=multiPointEval(f,L2,R)
pp=gcd(x0,prod(ev)).lift()
print pp
return cputime(t)

```

C.3 Source Code of our GCD Attack against the Vector Approximate-GCD Problem

```

def genVecXi(rho,eta,gam,m,p,K,x0):
    v=vector([genXi(rho,eta,gam,p) for i in range(m)])
    return K*v

def attackGCDMatrixMultiPoint(rho=6,eta=100,gam=200,m=2):
    p=random_prime(2^eta,lbound=2^(eta-1),proof=False)
    print "p=",p
    x0=p*prod([random_prime(2^eta,lbound=2^(eta-1),proof=False)
               for i in range(gam/eta-1)])

    R=Pring.PolynomialRing_dense_mod_n(Integers(x0),'x')
    x=R.0

    K=matrix(Integers(x0),m,m)
    for i in range(m):
        for j in range(m):
            K[i,j]=ZZ.random_element(x0)

    L1=[genVecXi(rho,eta,gam,m,p,K,x0)[0] for i in range(2^(m*rho//2+1))]
    L2=[genVecXi(rho,eta,gam,m,p,K,x0)[0] for i in range(2^(m*rho//2+1))]

    t=cputime(subprocesses=True)
    f=prod((x-c for c in L1))
    ev=multiPointEval(f,L2,R)
    pp=gcd(x0,prod(ev)).lift()
    print pp
    return cputime(t)

```

D Implementation of the Lattice Attacks against the Approximate-GCD Problem

D.1 Source Code of the Orthogonal Lattice Attack

```

def AttOrtho(eta=50,n=30,rho=10):
    p=[random_prime(2^eta,False,15*2^(eta-1-4)) for i in range(n)]
    x0=prod(p)
    r=[[ZZ.random_element(-2^rho+1,2^rho) for i in range(n)] for j in range(n)]
    x=[crt(rj,p) for rj in r]
    y=crt([ZZ.random_element(-2^rho+1,2^rho) for i in range(n)],p)

    x=x+[mod(xi*y,x0).lift() for xi in x]

    tau=2*n
    M=matrix(ZZ,tau,tau)
    for i in range(tau-1):
        M[i,i]=1
        M[i,tau-1]=mod(-x[i]*inverse_mod(x[tau-1],x0),x0)
    M[tau-1,tau-1]=x0

    ML=M.LLL()
    V=ML[:tau-n].right_kernel().matrix()
    W0,W1=V[:,n:].T,V[:,tau:].T
    v=(W1*W0^-1).eigenvalues()

```

```

rprimes=Set([gcd(y-vi,x0) for vi in v])
print "Number of primes recovered:", len(Set(p).intersection(rprimes)), "out of", n

```

D.2 Source Code of the Orthogonal Lattice Attack, Vector Variant

```

def smallMat(nrows,ncols,rho,p):
    n=len(p)
    r=[Matrix([[ZZ.random_element(-2^rho+1,2^rho) for k in range(ncols)]
               for j in range(nrows)]) for i in range(n)]
    return Matrix(ZZ,[[crt([r[i][j,k] for i in range(n)],p) for k in range(ncols)]
                     for j in range(nrows)])

def AttOrthoVec(eta=60,n=5,rho=10,m=2):
    p=[random_prime(2^eta,False,2^(eta-1)) for i in range(n)]
    x0=prod(p)
    V=smallMat(n*m,m,rho,p)
    K=random_matrix(Integers(x0),m)
    VT=V*K
    Kp=random_matrix(Integers(x0),m)
    A0,A1=smallMat(m,m,rho,p),smallMat(m,m,rho,p)
    C0,C1=K^-1*A0*Kp,K^-1*A1*Kp
    Vp0,Vp1=VT*C0,VT*C1

    Vp=Matrix(ZZ,2*n*m,m)
    Vp[:n*m,:],Vp[n*m:,:]=Vp0,Vp1

    tau=2*n*m
    M=matrix(ZZ,tau,tau)
    for i in range(tau-m):
        M[i,i]=1
    M[: ,tau-m:]=-Vp*Matrix(Integers(x0),Vp[tau-m:,:]).inverse()
    M[tau-m: ,tau-m:]=x0*matrix.identity(m)

    ML=M.LLL()

    V0=ML[:m*n].right_kernel().matrix()
    W0,W1=V0[: ,n*m].T,V0[: ,n*m:].T
    f=(W0*W1^-1).charpoly()
    B=matrix(Integers(x0),C0)*matrix(Integers(x0),C1)^-1
    rprimes=Set([gcd(f1[0].change_ring(Integers(x0))(B)[0,0],x0) for f1 in factor(f)])
    print "Number of primes recovered:", len(Set(p).intersection(rprimes)), "out of", n

```

E Implementation of the Cheon et al. Attacks

E.1 Source Code of the Basic Cheon et al. Attack

```

def encodeRand(rho,p):
    return crt([ZZ.random_element(2^rho) for i in range(len(p))],p)

def AtkCheon():
    eta=100
    n=10
    rho=20
    p=[random_prime(2^eta,False,2^(eta-1)) for i in range(n)]
    x0=prod(p)
    pzt=crt([ZZ.random_element(2^rho)*x0/pi for pi in p],p)
    c=[encodeRand(rho,p) for i in range(n)]
    d=[encodeRand(rho,p) for i in range(n)]
    b0=encodeRand(rho,p)
    b1=encodeRand(rho,p)

```



```

W0=matrix(ZZ,[[ci*dj*b0*pzt % x0 for dj in d] for ci in c])
W1=matrix(ZZ,[[ci*dj*b1*pzt % x0 for dj in d] for ci in c])

print "Basic attack",
W=W0*W1^-1
rec_primes=sorted([gcd(x0,a.denominator()*b0-a.numerator()*b1)
                    for a in W.eigenvalues()])
assert rec_primes==sorted(p)
print "OK"

print "Attack modulo q",
q=random_prime(2^eta,False,2^(eta-1))
W0q=matrix(Integers(q),W0)
W1q=matrix(Integers(q),W1)

Wq=W0q*W1q^-1
eigen=[a.rational_reconstruction() for a in Wq.eigenvalues(extend=False)]
rec_primes=sorted([gcd(x0,a.denominator()*b0-a.numerator()*b1) for a in eigen])
assert rec_primes==sorted(p)
print "OK"

```

E.2 Source Code of the Cheon et al. Attack with Matrices

```

def encodeVec(m,rho,p):
    return vector([encodeRand(rho,p) for i in range(m)])

def AtkMatrixCheon(m=2):
    eta=100
    n=5
    rho=20
    p=[random_prime(2^eta,False,2^(eta-1)) for i in range(n)]
    x0=prod(p)
    pzt=crt([ZZ.random_element(2^rho)*x0/pi for pi in p],p)
    d=m*n
    c=[encodeVec(m,rho,p) for i in range(d)]
    d=[encodeVec(m,rho,p) for i in range(d)]
    b0=matrix([encodeVec(m,rho,p) for i in range(m)])
    b1=matrix([encodeVec(m,rho,p) for i in range(m)])

    W0=matrix(ZZ,[[ci*b0*dj*pzt % x0 for dj in d] for ci in c])
    W1=matrix(ZZ,[[ci*b1*dj*pzt % x0 for dj in d] for ci in c])

    print "Basic attack",
    W=W0*W1^-1
    f=W.charpoly()
    B=matrix(Integers(x0),b0)*matrix(Integers(x0),b1)^-1
    rec_primes=sorted([gcd(f1[0].change_ring(Integers(x0))(B)[0,0],x0)
                       for f1 in factor(f)])
    assert rec_primes==sorted(p)
    print "OK"

    print "Variant attack",
    W=W0*W1^-1
    f=W.charpoly()
    rec_primes = sorted([gcd(matrix(Integers(x0),f1[0](W)*W0)[0,0],x0)
                          for f1 in factor(f)])
    assert rec_primes==sorted(p)
    print "OK"

```

E.3 Source Code of the Tensoring Attack

```
def encodeMat(m,rho,p):
    return matrix(ZZ,[[encodeRand(rho,p) for i in range(m)] for j in range(m)])

def AtkTensoringCheon(m=2):
    eta=180
    n=3
    rho=20
    p=[random_prime(2^eta,False,2^(eta-1)) for i in range(n)]
    x0=prod(p)
    pzt=crt([ZZ.random_element(2^rho)*x0/pi for pi in p],p)
    d=m^3*n
    a1=[encodeVec(m,rho,p) for i in range(d)]
    B1=[encodeMat(m,rho,p) for i in range(2)]
    C1=[encodeMat(m,rho,p) for i in range(d)]
    A2=[encodeMat(m,rho,p) for i in range(d)]
    B2=[encodeMat(m,rho,p) for i in range(2)]
    c2=[encodeVec(m,rho,p) for i in range(d)]

    W=[matrix(ZZ,[[a1[i]*B1[k]*C1[j]*A2[i]*B2[k]*c2[j]*pzt % x0
                  for j in range(d)] for i in range(d)])
        for k in range(2)]

    WW=W[0]*W[1]^(-1)
    f=WW.charpoly()

    print "Basic attack",
    BB=[matrix(Integers(x0),
               B2[k].tensor_product(matrix.identity(m)).tensor_product(B1[k]))
         for k in range(2)]
    BinvB=BB[0]*BB[1]^(-1)
    rec_primes=sorted([gcd(f1[0].change_ring(Integers(x0))(BinvB)[0,0],x0)
                       for f1 in factor(f)])
    assert rec_primes==sorted(p)
    print "OK"

    print "Variant attack",
    rec_primes = sorted([gcd(matrix(Integers(x0),f1[0](WW)*W[0])[0,0],x0)
                          for f1 in factor(f)])
    assert rec_primes==sorted(p)
    print "OK"
```