On Kilian's Randomization of Multilinear Map Encodings

Jean-Sébastien Coron and Hilder V. L. Pereira

University of Luxembourg

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 multilinear maps. 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 multiparty 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 that is resistant against known attacks, based on CLT13 multilinear maps. For $N = 4$ users and a medium level of security, our implementation requires 18 GB of public parameters, and a few minutes for the derivation of a shared key.

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]. Currently there are essentially only three known candidate constructions of multilinear maps:

- **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 GGHRSW 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 GGHRSW construction, because in GGHRSW 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 GGHRSW 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 GGHRSW 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 GGHRSW 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:

$$C(x) = \boldsymbol{b}_0 \times \prod_{i=1}^{n} \boldsymbol{B}_{i,x_{\mathsf{inp}(i)}} \times \boldsymbol{b}_{n+1} \tag{1}$$

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

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

Namely the successive randomization matrices $\boldsymbol{R}_i$ 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 $\tilde{\boldsymbol{B}}_{i,b}$ using a multilinear map scheme. Every entry of a given matrix is encoded separately; the bookend vectors $\tilde{\boldsymbol{b}}_0$ and $\tilde{\boldsymbol{b}}_n$ are also encoded similarly. Therefore one defines the matrices and vectors

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

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

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

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 $\mathsf{BP}(x)$, without revealing the values of the matrices $\boldsymbol{B}_{i,b}$. 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 one, 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 while keeping the same level of security; this leads to major improvements of performance. For the first issue, we craft a sequence of matrix products that only performs a multipartite DH key-exchange, rather than generating one from a circuit through Barrington's theorem, so that its degree becomes much more manageable. We can then describe the first concrete implementation of multipartite DH key-exchange based on multilinear maps that is resistant against existing attacks.

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 $\hat{\boldsymbol{B}}_{i,b}$ used to compute $\hat{C}(x)$ as in Equation (2), one can again choose $n+1$ random invertible matrices $\{\hat{\boldsymbol{R}}_i\}_{i=0}^{n}$ and then randomize the matrices $\hat{\boldsymbol{B}}_{i,b}$ with:

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

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

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

and therefore the same zero-testing procedure can be applied to $\hat{C}(x)$. Note that the $\hat{\boldsymbol{R}}_i$ matrices are applied on the encoding side, that is on the encoded matrices $\hat{\boldsymbol{B}}_{i,b}$, instead of the plaintext matrices $\boldsymbol{B}_{i,b}$ 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 $\{\hat{\boldsymbol{R}}_i\}_{i=0}^{n}$ are random invertible matrices modulo x_0. We 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 improve the efficiency of a construction by several orders of magnitude.

More precisely, the security of CLT13 is based on the hardness of the multi-prime Approximate-GCD problem. Given $x_0 = \prod_{i=1}^{n} p_i$ for random primes p_i, and polynomially many integers c_j such that

$$c_j \equiv r_{ij} \pmod{p_i} \tag{3}$$

for small integers r_{ij}'s, the goal is to recover the secret primes p_i's. The multi-prime Approximate-GCD problem is an extension of the single-prime problem, with a single prime p to be recovered from encodings $c_j = q_j \cdot p + r_j$ and $x_0 = q_0 \cdot p$, for small integers r_j. The two main approaches for solving the Approximate-GCD problem are the orthogonal lattice attacks and the GCD attacks.

First contribution: solving the multi-prime Approximate-GCD problem. For the single-prime Approximate-GCD problem, the classical orthogonal lattice attack 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 multi-prime case as in CLT13 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; we note that solving the multi-prime case was actually considered as an open problem in [GGM16].

Our first contribution is to solve this open problem with an algorithm 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 $\boldsymbol{r}_i = \boldsymbol{c} \bmod p_i$, where $\boldsymbol{c} = (c_1, \ldots, c_t)$. However, the vectors $\boldsymbol{r}_i$ cannot be recovered directly; namely by applying LLL or BKZ one recovers a basis of moderately short vectors, and not necessarily the $\boldsymbol{r}_i$'s which are the shortest vectors in the lattice. Therefore the approach described in [CLT13] does not work, except in low dimension. In the second step of our algorithm, using the lattice basis obtained from the first step, we show that by computing the eigenvalues of a well chosen matrix, we can recover the primes p_i's, as in the Cheon *et al.* attack [CHL+15]. The asymptotic complexity of the full 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 $2^{\Omega(n/\eta)}$. Therefore, as in [CLT13], one must take $n = \omega(\eta \log \lambda)$ to prevent the lattice attack, where λ is the security parameter.

Second contribution: extension to the Vector Approximate-GCD problem. When working with matrix branching programs and Kilian's randomization on the encoding side, we must actually consider a vector variant of the Approximate-GCD problem, in which we have access to randomized vectors of encodings instead of scalar values as in (3). Therefore, our second contribution is to extend the orthogonal lattice attack to the Vector Approximate-GCD problem, and to show that the extended attack has complexity $2^{\Omega(m \cdot n/\eta)}$, for vectors of dimension m. This implies that the new condition on the number n of primes p_i in CLT13 becomes:

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

Compared to the previous condition, 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 a factor m, which provides a significant improvement in efficiency.

Third contribution: GCD attacks against the Vector Approximate-GCD problem. The naive GCD attack against the Approximate-GCD problem with $c_1 = q_1 \cdot p + r_1$ and $x_0 = q_0 \cdot p$ consists in computing $\gcd(c_1 - r_1, x_0)$ for all possible r_1 and has complexity $\mathcal{O}(2^\rho)$, where ρ is the bitsize of r_1. 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 $\tilde{\mathcal{O}}(2^{\rho/2})$.

As previously, when working with matrix branching programs and Kilian's randomization on the encoding side, we must consider the vector variant of the Approximate-GCD problem. Our third contribution is therefore to extend the Lee-Seo attack to this vector variant; 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, 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. Combined with the previous improvement, this improves the efficiency of CLT13 based constructions by several orders of magnitude.

Fourth 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. This was originally described for GGH13, CLT13 and GGH15, but was quickly 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 candidate multilinear maps constructions?

In this paper we provide a first step in that direction. Namely our fourth contribution is to describe the first implementation of N-way DH key exchange resistant against known attacks. Our construction is based on CLT13 multilinear maps and is secure against the Cheon *et al.* attack and its variants. Our construction contains many ingredients from the GGHRSW and other similar constructions. 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. As explained previously, we use Kilian's randomization on the encoding side. With no additional cost, we can also use the straddling set systems from [BGK+14] to further constrain the attacker, and Kilian's randomization at the plaintext level. 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.

For $N = 4$ users and a medium (62 bits) level of security, our implementation requires 18 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.

Related work. In [MZ18], Ma and Zhandry described 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. 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 their construction. The authors still provide the following parameters for a 4-party DH key exchange with 80 bits of security; see Table 1. We provide our corresponding parameters for comparison (see more details in Section 7).

Scheme	MMap degree	Public encodings	Public-key size
Boneh et al. [BISW17]	4150	2^{44}	
Ma-Zhandry (setting 1)	52	2^{62}	
Ma-Zhandry (setting 2)	160	2^{33}	
Ma-Zhandry (setting 3)	1040	2^{19}	
Ma-Zhandry (setting 4)	2000	2^{14}	
Our construction	266	2^{20}	1848 GB

Table 1. Comparison of parameters for 4-party DH key exchange, with 80 bits of security.

The main advantage of the Ma-Zhandry construction is that it has a proof of security in a weak multilinear map model, whereas our construction has heuristic security only. It seems from Table 1 that our construction would require a smaller multilinear map degree for the same number of public encodings. We stress however that providing concrete parameters is actually a complex optimization problem (see Section 7), so Table 1 should be handled with care. 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.

Source code. We provide the source code of our construction, and the source code of the attacks, in [CP19].

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 $\{1, 2, \ldots, n\}$ 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 $\mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2} \times \cdots \times \mathbb{Z}_{g_n}$ for secret prime integers g_i's of α bits.

The CLT13 scheme is based on CRT representations. We denote by $\mathsf{CRT}(a_1, \ldots, a_n)$ or $\mathsf{CRT}(a_i)_i$ 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 $\mathbf{m} = (m_1, \ldots, m_n)$ at level set $S = \{j\}$ is an integer $c \in \mathbb{Z}_{x_0}$ such that $c = [\mathsf{CRT}(m_1 + g_1 r_1, \ldots, m_n + g_n r_n)/z_j]_{x_0}$ 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:

$$c \equiv \frac{m_i + g_i r_i}{z_j} \pmod{p_i} \tag{4}$$

for all $1 \le i \le n$. 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

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

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

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

we obtain after multiplication by p_{zt}:

$$c \cdot p_{zt} = \mathsf{CRT}(h_i p_i^* (m_i g_i^{-1} + r_i))_i = \sum_{i=1}^{n} h_i p_i^* (m_i g_i^{-1} + r_i) \pmod{x_0} \tag{5}$$

and therefore if $m_i = 0$ for all $1 \le i \le n$ then the result will be small compared to x_0. From the previous equation the high-order bits of $c \cdot p_{zt} \bmod x_0$ 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:

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

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 secret random invertible matrix $\boldsymbol{K}$ 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 $\boldsymbol{K}$ modulo x_0. Given x_0 and polynomially many samples $\tilde{\boldsymbol{v}} = \boldsymbol{v} \cdot \boldsymbol{K} \bmod x_0$ where $\boldsymbol{v} \leftarrow (\mathcal{D}_{\gamma,\rho}(p))^m$, 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 $\tilde{\boldsymbol{v}} = \boldsymbol{v} \cdot \boldsymbol{K} \pmod{x_0}$ for some random matrix $\boldsymbol{K}$. 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 sections that the generalizations of the attacks to the vector variant indeed have higher complexity.

In the context of the CLT13 scheme, one actually works with multiple primes p_i's. Therefore we consider the multi-prime 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 $c_j = \mathsf{CRT}(r_{ij})_i$ 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 $\boldsymbol{K}$ be a random invertible $m \times m$ matrix modulo x_0. Given x_0 and polynomially many vectors $\tilde{\boldsymbol{v}} = \boldsymbol{v} \cdot \boldsymbol{K} \bmod x_0$, where $\boldsymbol{v} = (v_1, \ldots, v_m)$ and $v_j = \mathsf{CRT}(r_{ij})_i$ 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 orthogonal lattice attacks and the GCD attacks. We consider the orthogonal lattice attacks in Section 3, and the GCD attacks in Section 4.

3 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 multi-prime 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. Therefore, we describe a new algorithm for solving the multi-prime Approximate-GCD problem, using a variant of the Cheon *et al.* attack against CLT13. We then extend the algorithm to the vector variant of the Approximate-GCD problem. Finally, we run our attacks against both the multi-prime Approximate-GCD problem and the vector variant, in order to derive concrete parameters for our construction. We provide the source code of our attacks in [CP19].

3.1 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 short 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 $\boldsymbol{b_i} \in \mathbb{Z}^m$, from Hadamard inequality we expect that $\det L \simeq \prod_{i=1}^{d} \|\boldsymbol{b_i}\|$. In that case, we therefore expect the short vectors of $L^\perp$ to have norm $\simeq (\det L^\perp)^{1/(m-d)} \simeq (\det L)^{1/(m-d)} \simeq (\prod_{i=1}^{d} \|\boldsymbol{b_i}\|)^{1/(m-d)}$.

3.2 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 $\boldsymbol{u}$ that are orthogonal to $\boldsymbol{x}$ modulo x_0, where $\boldsymbol{x} = (x_1, \ldots, x_t)$:

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

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

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

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

$$\|\boldsymbol{u}_k\| \leq 2^{\iota t} \cdot (\det L)^{1/t} \approx 2^{\iota t + \gamma/t} \tag{6}$$

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 $\boldsymbol{u} \in L$, we have $\boldsymbol{u} \cdot \boldsymbol{x} \equiv 0 \pmod{x_0}$, which implies that $\boldsymbol{u} \cdot \boldsymbol{r} \equiv 0 \pmod{p}$ where $\boldsymbol{r} = (r_1, \ldots, r_t)$. The main observation is that if $\boldsymbol{u}$ is short enough, the equality will hold over $\mathbb{Z}$. More precisely, if $\|\boldsymbol{u}\| \cdot \|\boldsymbol{r}\| < p$, we get $\boldsymbol{u} \cdot \boldsymbol{r} = 0$ in $\mathbb{Z}$. From (6), this happens under the condition:

$$2^{\iota t + \gamma/t} \cdot 2^{\rho} < 2^{\eta}. \tag{7}$$

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

Asymptotic complexity. We derive a heuristic lower bound for the complexity of the attack, as in [DGHV10, §5.2]. From condition (7) the attack requires a minimal lattice dimension $t > \gamma/\eta$; therefore from the same condition we must have

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

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

3.3 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 $\boldsymbol{x}$ formed by the first t integers x_j, and we consider the lattice L of vectors $\boldsymbol{u}$ that are orthogonal to $\boldsymbol{x}$ modulo x_0:

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

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

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

$$2^{\rho \cdot n/(t-n)} \cdot 2^{\iota t} < 2^{\eta - \rho}, \tag{8}$$

In that case, applying lattice reduction to L should yield a reduced basis $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t)$ where the first $t - n$ vectors belong to the sublattice $L_{\boldsymbol{r}}^{\perp}$. By computing the rank n lattice orthogonal to those vectors, one recovers a basis $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ of the lattice $\bar{L}_{\boldsymbol{r}} = (L_{\boldsymbol{r}}^{\perp})^{\perp}$, where $L_{\boldsymbol{r}}$ is the lattice generated by the n vectors $\boldsymbol{r}_i$, However this does not necessarily reveal the original vectors $\boldsymbol{r}_i$. Namely even by applying LLL or BKZ on the basis B, we do not necessarily recover the short vectors $\boldsymbol{r}_i$'s, except possibly in low dimension; therefore the approach described in [CLT13] only works when n is small.

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

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

for unknown coefficients $\lambda_{ji} \in \mathbb{Q}$. The above equation is analogous to Equation (5) on the zero-tested value $c \cdot p_{zt}$, which is a linear combination of the 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 $\boldsymbol{b_j}$ 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 $1 \le i \le n$.

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

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

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

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

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

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

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

By applying the orthogonal lattice attack of the first step on the known vector $\boldsymbol{z}$, we obtain a basis of the lattice intersection of $\mathbb{Z}^{2n}$ with the $\mathbb{Q}$-vector space generated by the n vectors $\boldsymbol{z} \bmod p_i$, which corresponds to the columns of the matrix $\boldsymbol{Z}$. Therefore we obtain two matrices $\boldsymbol{W_0}$ and $\boldsymbol{W_1}$ such that:

$$\boldsymbol{W_0} = \boldsymbol{R} \cdot \boldsymbol{A}$$
$$\boldsymbol{W_1} = \boldsymbol{R} \cdot \boldsymbol{U} \cdot \boldsymbol{A}$$

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

$$\boldsymbol{W} = \boldsymbol{W_1} \cdot \boldsymbol{W_0}^{-1} = \boldsymbol{R} \cdot \boldsymbol{U} \cdot \boldsymbol{R}^{-1}$$

and by computing the eigenvalues of $\boldsymbol{W}$, one recovers the components s_i of the diagonal matrix $\boldsymbol{U}$, from which we recover the p_i's by taking gcd's. We provide the source code of the attack in [CP19].

Asymptotic complexity. As previously, we derive a heuristic lower bound for the complexity of the attack. The attack requires a lattice dimension $t = 2n$, and moreover the vectors $\boldsymbol{r}_i$ have norm $\simeq 2^{2\rho}$ instead of 2^ρ; therefore condition (8) gives $4\rho + 2\iota n < \eta$ which implies the condition

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

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 at least $2^{\Omega(n/\eta)}$. Note that with $\gamma = \eta \cdot n$, we get the same time complexity lower bound $2^{\Omega(\gamma/\eta^2)}$ as for the single-prime Approximate-GCD problem. Finally, as shown in [CLT13], to prevent the orthogonal lattice attack, one must take:

$$n = \omega(\eta \log \lambda) \tag{9}$$

Namely, in that case there exists a function $c(\lambda)$ such that $n(\lambda) = c(\lambda)\eta(\lambda)\log_2 \lambda$ with $c(\lambda) \to \infty$ for $\lambda \to \infty$. With a time complexity at least $2^{k \cdot n/\eta}$ for some $k > 0$, the time complexity is therefore at least $2^{k \cdot c(\lambda) \log_2 \lambda} = \lambda^{k \cdot c(\lambda)}$. This implies that the attack is not polynomial time under Condition 9.

3.4 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 t row vectors $\boldsymbol{v}_j$, each with m components $(\boldsymbol{v}_j)_k$, such that:

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

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

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

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

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

$$\tilde{\boldsymbol{C}}_0 = \boldsymbol{K}^{-1} \cdot \boldsymbol{C}_0 \cdot \boldsymbol{K'} \pmod{x_0}$$
$$\tilde{\boldsymbol{C}}_1 = \boldsymbol{K}^{-1} \cdot \boldsymbol{C}_1 \cdot \boldsymbol{K'} \pmod{x_0}$$

for some random invertible matrix $\boldsymbol{K'}$ modulo x_0, where the components of the matrices $\boldsymbol{C}_0, \boldsymbol{C}_1 \in \mathbb{Z}_{x_0}^{m \times m}$ are small modulo each p_i. This assumption is verified in our construction of Section 5.

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

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

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

$$L = \{\boldsymbol{u} \in \mathbb{Z}^t \mid \boldsymbol{u} \cdot \boldsymbol{V} \equiv 0 \pmod{x_0}\ \} \tag{10}$$

The lattice L is of full rank t since it contains $x_0\mathbb{Z}^t$. Let $\boldsymbol{R}_i = \boldsymbol{V} \bmod p_i$. 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 $\boldsymbol{R}_i$ for $1 \le i \le n$. 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 $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t)$ where the first $t - m \cdot n$ vectors belong to the sublattice L', under the modified condition:

$$2^{\iota t + \rho \cdot m \cdot n/(t-m\cdot n)} < 2^{\eta - \rho} \tag{11}$$

As previously, by computing the rank $n \cdot m$ lattice orthogonal to the vectors $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{t-m\cdot n})$, 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 $\boldsymbol{R}_i$'s.

Second step: algebraic attack. Considering the original $t \times m$ matrix $\tilde{\boldsymbol{V}}$ and using $t = nm$ rows:

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

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

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

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

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

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

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

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

By applying the orthogonal lattice attack of the first step on the matrix $\tilde{\boldsymbol{D}}$, 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 $\boldsymbol{D} \bmod p_i$, which corresponds to the columns of the matrix $\boldsymbol{E}$. Therefore as previously we obtain two matrices $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ such that:

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

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

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

The characteristic polynomial $f(X)$ of $\boldsymbol{W}$ is therefore the same as the characteristic polynomial of $\hat{\boldsymbol{S}}_1 \cdot \hat{\boldsymbol{S}}_0^{-1}$ which is the product of the n characteristic polynomials $f_i(X)$ of the matrices $\boldsymbol{S}_{1,i} \cdot \boldsymbol{S}_{0,i}^{-1}$. By the Cayley-Hamilton theorem, we must have $f_i(\boldsymbol{S}_{1,i} \cdot \boldsymbol{S}_{0,i}^{-1}) = 0$ for all i, which implies $f_i(\boldsymbol{C}_1 \cdot \boldsymbol{C}_0^{-1}) = 0 \pmod{p_i}$, which also implies $f_i(\tilde{\boldsymbol{C}}_1 \cdot \tilde{\boldsymbol{C}}_0^{-1}) = 0 \pmod{p_i}$ for all $1 \le i \le n$. 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 $\boldsymbol{M}_i = f_i(\tilde{\boldsymbol{C}}_1 \cdot \tilde{\boldsymbol{C}}_0^{-1}) \bmod x_0$ with x_0. We provide the source code of the attack in [CP19].

Alternatively, if the polynomials $f_i(X)$ are not irreducible, one can still factor $f(X)$ into monic irreducible factors $f'_1, \ldots, f'_N \in \mathbb{Q}[X]$. 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 $G_k(\tilde{\boldsymbol{C}}_1 \cdot \tilde{\boldsymbol{C}}_0^{-1}) = 0$ modulo all primes except one, and therefore the remaining prime p_i can be recovered by computing the gcd of the entries of $\boldsymbol{M}_k = G_k(\tilde{\boldsymbol{C}}_1 \cdot \tilde{\boldsymbol{C}}_0^{-1}) \bmod x_0$ with x_0.

Asymptotic complexity. As previously, we derive a heuristic lower bound for the complexity of the attack. Since the attack requires a lattice dimension $t = 2mn$, condition (11) with noise size 2ρ instead of ρ gives $4\rho + 2\iota mn < \eta$ which gives the new condition

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

. Therefore, the orthogonal lattice attack has time complexity at least $2^{\Omega(n \cdot m/\eta)}$. 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 (9), 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. We show in Section 7 that this brings a significant improvement in practice.

3.5 Practical experiments and concrete parameters

Practical experiments. We have run our two attacks from sections 3.3 and 3.4 against the multi-prime approximate-GCD problem and its vector variant; we provide the source code in [CP19]. We summarize the running times for various values of n in tables 2 and 3. We see that the running time of the lattice step in the vector variant is roughly the same as in the non-vector variant, when the number of primes n is divided by m in the vector variant. This confirms the asymptotic analysis of the previous section.

For the algebraic step of the non-vector problem, it is significantly more efficient to compute the matrix kernel and eigenvalues modulo some arbitrary prime integer q of size η, instead of over the rationals. However we have not found a similar optimization for the vector variant; we see in Table 3 that for larger n the cost of the algebraic step becomes prohibitive (but still polynomial time) for the vector variant. In this paper we conservatively fix our concrete parameters by considering the lattice step only. We leave as an open problem the derivation of a "practical" algebraic step for the vector variant.

n	η	ρ	lat. dim.	Time LLL	Time alg.
20	335	80	40	1.5 s	0.6 s
30	335	80	60	9 s	0.7 s
40	335	80	80	37 s	1.5 s
60	335	80	120	4 min	4 s
80	335	80	160	20 min	8 s

Table 2. Running time of the LLL step and the algebraic step for solving the multi-prime Approximate-GCD problem, on a 3.2 GHz Intel Core i5.

n	m	η	ρ	lat. dim.	Time LLL	Time alg.
4	5	335	80	40	1.4 s	2.3 s
6	5	335	80	60	9 s	20 s
8	5	335	80	80	32 s	27 min
12	5	335	80	120	6 min	–
16	5	335	80	160	12 min	–

Table 3. Running time of the LLL step and the algebraic step for solving the vector multi-prime Approximate-GCD problem, on a 3.2 GHz Intel Core i5.

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

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

with $t = 2nm$. We write $\boldsymbol{u} = [\boldsymbol{u_1}, \boldsymbol{u_2}]$ with $\boldsymbol{u_1} \in \mathbb{Z}^{t-m}$ and $\boldsymbol{u_2} \in \mathbb{Z}^m$. Similarly we write $\tilde{\boldsymbol{V}} = \begin{bmatrix} \boldsymbol{A} \\ \boldsymbol{W} \end{bmatrix}$ where $\boldsymbol{W}$ is a $m \times m$ matrix. With high probability $\boldsymbol{W}$ is invertible modulo x_0, otherwise we can partially factor x_0. We obtain

$$\begin{aligned} \boldsymbol{u} \in L &\iff \boldsymbol{u_1 A} + \boldsymbol{u_2 W} \equiv 0 \pmod{x_0} \\ &\iff \boldsymbol{u_1 A W}^{-1} + \boldsymbol{u_2} \equiv 0 \pmod{x_0} \end{aligned}$$

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

$$L = \begin{bmatrix} \boldsymbol{I}_{t-m} & -\boldsymbol{AW}^{-1} \\ & x_0 \boldsymbol{I}_m \end{bmatrix}$$

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

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

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

$$T_{LLL}(t, \gamma, m) \simeq 2 \cdot t^{3.3} \cdot \gamma \cdot m \tag{12}$$

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

$$T_{BKZ}(t, \beta) \simeq b(\beta) \cdot t^{4.3} \tag{13}$$

where the observed constant $b(\beta)$ and the Hermite factor are given in Table 4. 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 $\mathsf{poly}(t) \cdot 2^{0.292\beta + \circ(\beta)}$ (see [BDGL16]). The Hermite factors for BKZ-80 and BKZ-100 are from [CN11].

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

Table 4. 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.

Setting concrete parameters. When applying LLL or BKZ with blocksize β on the original lattice L, we obtain an orthogonal vector $\boldsymbol{u}$ under the condition (11), which gives with $t = 2nm$ and vectors with noise size 2ρ instead of ρ:

$$\iota \cdot 2nm + 4\rho < \eta \tag{14}$$

Therefore we must run LLL or BKZ-β with a large enough blocksize β so that ι is small enough for condition (14) 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 (12) or (13), for $\gamma = \eta \cdot n$. We use that condition to provide concrete parameters for our scheme in Section 7.

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

4.1 The Naive GCD Attack.

For simplicity we first consider the single prime variant of the Approximate-GCD problem. More precisely, we consider $x_0 = q_0 \cdot p$ 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.

4.2 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:

$$p = \gcd\left(x_0, \prod_{i=0}^{2^{\rho}-1} (c - i) \pmod{x_0}\right) \tag{15}$$

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:

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

One can then rewrite (15) 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 $x_1, \ldots, x_d$. 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 $x_1, \ldots, x_{d/2}$ is equal to the evaluation of the degree $d/2$ polynomial $f_l(x) = f(x) \bmod (x - x_1) \cdots (x - x_{d/2})$ on $x_1, \ldots, x_{d/2}$. Therefore the evaluation of $f(x)$ can proceed with a recursive algorithm. First compute the left polynomial $f_l(x) = f(x) \bmod (x-x_1) \cdots (x-x_{d/2})$; 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 $f_r(x) = f(x) \bmod (x - x_{d/2+1}) \cdots (x - x_d)$. 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 [CP19] an implementation of the Chen-Nguyen attack in Sage; our running time is similar to [CN12, Table 1]; see Table 5 below for practical experiments. 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:

$$T_{CN}(\rho, \gamma) = 0.3 \cdot \rho^2 \cdot 2^{\rho/2} \cdot \gamma \cdot \log^2 \gamma \tag{16}$$

4.3 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 $p = \gcd(c_1/c_2 - r_1/r_2 \bmod x_0, x_0)$, 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:

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

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

$$f(x) = \prod_i (c_i - x) \bmod x_0$$

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

$$p = \gcd\left(\prod_j f(d_j) \bmod x_0, x_0\right)$$

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 [CP19] an implementation of the Lee-Seo attack in Sage. Its running time is roughly 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.

4.4 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 $\boldsymbol{v}_i$ of dimension m, such that for each vector $\boldsymbol{v}_i$, all components $(\boldsymbol{v}_i)_j$ of $\boldsymbol{v}_i$ are small modulo p:

$$(\boldsymbol{v_i})_j = r_{ij} \pmod{p}$$

However, we only obtain the randomized vectors:

$$\tilde{\boldsymbol{v}}_{\boldsymbol{i}} = \boldsymbol{v_i} \cdot \boldsymbol{K} \pmod{x_0}$$

for some random invertible matrix $\boldsymbol{K}$ 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 $c_i = (\tilde{\boldsymbol{v}}_{\boldsymbol{i}})_1$ of each vector $\tilde{\boldsymbol{v}}_{\boldsymbol{i}}$. We have:

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

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 modulo p of m random values r_{ij}'s (where the coefficients are initially generated at random modulo p), 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 $\tilde{\mathcal{O}}(2^{m \cdot \rho/2})$. 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 $\tilde{\mathcal{O}}(2^{\eta/2})$, which corresponds to the complexity of the Pollard's rho factoring algorithm. We provide in [CP19] an implementation of the attack in Sage; see Table 5 below for practical experiments.

With an attack complexity $\tilde{\mathcal{O}}(2^{m\rho/2})$ instead of $\tilde{\mathcal{O}}(2^{\rho/2})$, 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. Note that we can take $m \cdot \rho/2 < \lambda$ because we only require that the running time in number of clock-cycles is at least 2^λ. More precisely, the running time can be approximated by $T_{CN}(m\rho, \gamma)$ for a γ-bit x_0, where $T_{CN}(\rho, \gamma)$ is given by (16), and we require $T_{CN}(m\rho, \gamma) \geq 2^\lambda$.

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

$$(\boldsymbol{V_i})_{jk} = r_{ijk} \pmod p \tag{17}$$

for ρ-bit integers r_{ijk}. As previously, instead of publishing the matrices $\boldsymbol{V_i}$, we publish the randomized matrices

$$\tilde{\boldsymbol{V_i}} = \boldsymbol{K} \cdot \boldsymbol{V_i} \cdot \boldsymbol{K'} \pmod{x_0} \tag{18}$$

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

Formally, using the Kronecker product(see Appendix A), we can rewrite (18) as

$$\mathsf{vec}\,(\tilde{\boldsymbol{V_i}}) = (\boldsymbol{K'}^T \otimes \boldsymbol{K})\,\mathsf{vec}\,(\boldsymbol{V_i}),$$

where $\mathsf{vec}\,(\boldsymbol{V_i})$ denotes the column vector of dimension m^2 formed by stacking the columns of $\boldsymbol{V_i}$ on top of one another, and similarly for $\mathsf{vec}\,(\tilde{\boldsymbol{V_i}})$. We can therefore apply the previous attack with vectors of dimension m^2 instead of m; the attack complexity is therefore $\tilde{\mathcal{O}}(2^{m^2 \cdot \rho/2})$. This implies that we can divide the noise size ρ by a factor m^2 compared to [CLT13], where m is the matrix dimension. We provide in [CP19] an implementation of the attack in Sage; see Table 5 below for practical experiments.

With multiple primes $\boldsymbol{p_i}$'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 (15) from the Chen-Nguyen attack can be rewritten:

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

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, with noise size ρ_m, the running time of the attack in number of clock cycles can therefore be approximated by

$$T_{GCD}(m, \gamma, \rho_m, n) = T_{CN}(\rho, \gamma)/\sqrt{n} \tag{19}$$

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

AGCD: Chen-Nguyen	ρ	12	16	20	24
	time (s)	0.3	2.5	15	94
m-vector AGCD: our attack ($m = 4$)	ρ_v	3	4	5	6
	time (s)	1.5	9.3	53	301
$m \times m$-matrix AGCD: our attack ($m = 2$)	ρ_m	3	4	5	6
	time (s)	1.5	10	54	300

Table 5. Running time of the Chen-Nguyen attack against the Approximate-GCD problem and our attack against the vector variant and matrix variants with $\eta = 100$ and x_0 of size $\gamma = 16\,000$, on a 3.2 GHz Intel Core i5.

Practical experiments. We provide in Table 5 the result of practical experiments against the Approximate-GCD problem and its vector variant with a single prime p. We see that our attack against the vector variant with dimension m and noise size ρ_v has roughly the same running time as the Chen-Nguyen attack on the original problem with noise $\rho = m \cdot \rho_v$; similarly, the running time of our attack against $m \times m$ matrices with noise ρ_m has roughly the same running time as Chen-Nguyen with noise $\rho = m^2 \cdot \rho_m$; this confirms the above analysis. We provide the source code in [CP19].

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.

- $\mathsf{Setup}(1^\lambda, N)$: 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.
- $\mathsf{Publish}(\mathsf{params}, u)$: Given a party $u \in [N]$, this algorithm generates a pair of keys $(\mathsf{sk}_u, \mathsf{pk}_u)$. Party u broadcasts pk_u and keeps sk_u secret.
- $\mathsf{KeyGen}(\mathsf{params}, v, \mathsf{sk}_v, \{\mathsf{pk}_u\}_{u \neq v})$: Party $v \in [N]$ uses its secret sk_v and all the values pk_u broadcasted by other parties to generate a session key s_v.

We say that the protocol is correct if $s = s_1 = s_2 = \cdots = s_N$, 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 $\mathsf{pk}_1, \dots, \mathsf{pk}_N$.

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 by a trusted authority to generate the public parameters. As illustrated in Table 6, Setup generates for each party v two sequences of matrices $(\boldsymbol{C}_{i,b}^{(v)})_{i=1,\dots,\ell}$ for $b \in \{0,1\}$. In the KeyGen procedure, each party v will use the product of the matrices $\boldsymbol{C}_{i,b}^{(v)}$ on his row v to generate the session-key. The product is computed according to the secret-key sk_v of Party v and the secret-keys sk_u of the other parties. Therefore, in the $\mathsf{Publish}$ procedure, each party u will compute and publish the partial sub-products corresponding to his sk_u on the other rows $v \neq u$, to be used by each party v on his row v.

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

Table 6. Public matrices for $N = 3$ generated during the Setup procedure.

$\mathsf{Setup}(1^\lambda, N)$: 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.

To ensure that all users $1 \le u \le N$ compute the same session-key, we define $\boldsymbol{A}_{i,b}^{(u)}$ as a larger matrix embedding a matrix $\boldsymbol{B}_{i,b}$ 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:

$$\boldsymbol{A}_{i,b}^{(u)} \sim \begin{pmatrix} \begin{matrix} \$ & \dots & \$ \\ \vdots & \ddots & \vdots \\ \$ & \dots & \$ \end{matrix} & & \\ & \begin{matrix} \$ & \dots & \$ \\ \vdots & \ddots & \vdots \\ \$ & \dots & \$ \end{matrix} & \\ & & \alpha_{i,b}^{(u)} \cdot \boldsymbol{B}_{i,b} \end{pmatrix} \tag{20}$$

More precisely, we first sample 2ℓ random invertible matrices $\boldsymbol{B}_{i,b}$ in $\mathbb{Z}_g^{m' \times m'}$ where $m' = m/3$, for $1 \le i \le \ell$ and $b \in \{0,1\}$. For each $u \in [N]$, we additionally sample $2\ell N$ scalars $\alpha_{i,b}^{(u)}$ in $\mathbb{Z}_g^\star$ and $4\ell N$ random invertible matrices $\boldsymbol{S}_{i,b}^{(u)}$ and $\boldsymbol{T}_{i,b}^{(u)}$ in $\mathbb{Z}_g^{m' \times m'}$, for $1 \le i \le \ell$ and $b \in \{0,1\}$. As illustrated in (20), we let

$$\boldsymbol{A}_{i,b}^{(u)} := \mathsf{diag}(\boldsymbol{S}_{i,b}^{(u)},\ \boldsymbol{T}_{i,b}^{(u)},\ \alpha_{i,b}^{(u)} \cdot \boldsymbol{B}_{i,b}) \tag{21}$$

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

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

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

We let $\tilde{\boldsymbol{A}}_{i,b}^{(u)} \in \mathbb{Z}_{x_0}^{m \times m}$ be the matrix obtained by encoding each entry of $\boldsymbol{A}_{i,b}^{(u)}$ independently. Similarly we encode $\boldsymbol{s}^{(u)}$ and $\boldsymbol{t}^{(u)}$ entry-wise, obtaining $\tilde{\boldsymbol{s}}^{(u)}$ and $\tilde{\boldsymbol{t}}^{(u)}$. For each $u \in [N]$, we sample uniformly random invertible matrices $\boldsymbol{K}_i^{(u)} \in \mathbb{Z}_{x_0}^{m \times m}$ for $0 \le i \le \ell$. We then use Kilian's randomization "on the encoding side" and define:

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

Similarly, we define $\bar{\boldsymbol{s}}^{(u)} := \tilde{\boldsymbol{s}}^{(u)} \left(\boldsymbol{K}_0^{(u)}\right)^{-1} \pmod{x_0}$ and $\bar{\boldsymbol{t}}^{(u)} := \boldsymbol{K}_\ell^{(u)} \tilde{\boldsymbol{t}}^{(u)} p_{zt} \pmod{x_0}$. Note that thanks to Kilian's randomization "on the encoding side", the matrices $\boldsymbol{A}_{i,b}^{(u)}$ can be encoded with denominator $z_j = 1$ in (4) for all levels j; namely we obtain the same distribution in the final $\boldsymbol{C}_{i,b}^{(u)}$ as with random z_j's. Finally we output params, which is defined as the set containing all the matrices $\boldsymbol{C}_{i,b}^{(u)}$, the bookend vectors $\bar{\boldsymbol{s}}^{(u)}$ and $\bar{\boldsymbol{t}}^{(u)}$, and the scalars μ, k, N, ℓ, x_0, ν and m.

$\mathsf{Publish}(\mathsf{params}, u)$: Party u samples a bit string $\mathsf{sk}^{(u)} \in \{0,1\}^\mu$ and for each $v \in [N]$ such that $u \neq v$, Party u computes k products using matrices from the row of party v. This ensures that from the extraction procedure of the multilinear map scheme, each user u can derive the session key from his own $\mathsf{sk}^{(u)}$ by computing on his row u the partial products corresponding to his $\mathsf{sk}^{(u)}$, combined with the published partial matrix products from the other users. More precisely, Party u computes and broadcasts the following products:

$$\boldsymbol{D}_r^{(u \to v)} := \prod_{i=0}^{\mu-1} \boldsymbol{C}^{(v)}_{(r-1)N\mu + (u-1)\mu + i, \mathsf{sk}^{(u)}[i]} \pmod{x_0} \tag{22}$$

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

$\mathsf{KeyGen}(\mathsf{params}, v, \mathsf{sk}^{(v)}, \{\mathsf{pk}_u\}_{u \neq v})$: Using secret $\mathsf{sk}^{(v)}$, party v computes the products $\boldsymbol{D}_r^{(v \to v)}$ for all $r \in [k]$ using (22), and then the product

$$z^{(v)} := \bar{\boldsymbol{s}}^{(v)} \left(\prod_{r=1}^{k} \left(\prod_{u=1}^{N} \boldsymbol{D}_r^{(u \to v)} \right) \right) \bar{\boldsymbol{t}}^{(v)} \pmod{x_0}. \tag{23}$$

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:

$$\mathsf{sk} = (\underbrace{\mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(N)}}_{\text{First repetition}}, \ldots, \underbrace{\mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(N)}}_{k\text{-th repetition}}) \tag{24}$$

we obtain from (22) and (23), and then from the cancellation of Kilian's randomization on the encoding side:

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

This corresponds to a zero-tested encoding of:

$$v_v = \boldsymbol{s}^{(v)} \cdot \left(\prod_{i=1}^{\ell} \boldsymbol{A}^{(v)}_{i,\mathsf{sk}[i]} \right) \cdot \boldsymbol{t}^{(v)} = \boldsymbol{s}^* \cdot \left(\prod_{i=1}^{\ell} \alpha^{(v)}_{i,\mathsf{sk}[i]} \right) \cdot \left(\prod_{i=1}^{\ell} \boldsymbol{B}_{i,\mathsf{sk}[i]} \right) \cdot \boldsymbol{t}^* \pmod g$$

From the condition satisfied by the $\alpha^{(v)}_{i,b}$'s, the products $\prod_{i=1}^{\ell} \alpha^{(v)}_{i,\mathsf{sk}[i]}$ 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 $\alpha^{(u)}_{i,b}$, 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 $\tilde{\boldsymbol{A}}^{(u)}_{i,0}$ and $\tilde{\boldsymbol{A}}^{(u)}_{i,1}$, 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 $p_1, \ldots, p_n$ 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 $\boldsymbol{W_0}, \boldsymbol{W_1} \in \mathbb{Z}_{x_0}^{n\times n}$ whose entries are defined as $\boldsymbol{W_0}[i,j] := w_{i,j}$ and $\boldsymbol{W_1}[i,j] := w'_{i,j}$.

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

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

Therefore we can write $\boldsymbol{W_0} = \boldsymbol{A}\boldsymbol{B}_0\boldsymbol{C}$, where the rows of $\boldsymbol{A}$ are the vectors in the left (for $i \in [n]$), $\boldsymbol{B}_0$ is the diagonal matrix in the middle, and $\boldsymbol{C}$ is the matrix whose columns are the vectors in the right (for $j \in [n]$). By the same argument, $\boldsymbol{W_1} = \boldsymbol{A}\boldsymbol{B}_1\boldsymbol{C}$, where $\boldsymbol{B}_1 = \mathsf{diag}(\xi_1, \ldots, \xi_n)$. Thus, the attacker can compute over $\mathbb{Q}$:

$$\boldsymbol{W} = \boldsymbol{W_0}\boldsymbol{W_1}^{-1} = (\boldsymbol{A}\boldsymbol{B}_0\boldsymbol{C})(\boldsymbol{A}\boldsymbol{B}_1\boldsymbol{C})^{-1} = \boldsymbol{A}\boldsymbol{B}_0\boldsymbol{B}_1^{-1}\boldsymbol{A}^{-1}$$

The eigenvalues of $\boldsymbol{W}$ are the same as those of $\boldsymbol{B}_0\boldsymbol{B}_1^{-1}$ and are equal to $b_{0,1}, \ldots, b_{0,n}$. 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 [CP19].

Variant 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 $\boldsymbol{W_0}\boldsymbol{W_1}^{-1}$ modulo q only. The characteristic polynomial of $\boldsymbol{W_0}\boldsymbol{W_1}^{-1}$ is computed modulo q and then factored to recover the $b_{0,i}$'s modulo q. Experimentally, computing the two matrices $\boldsymbol{W_0}$ and $\boldsymbol{W_1}$ takes time $\mathcal{O}(n^{3.5})$. Computing the full $\boldsymbol{W_0}\boldsymbol{W_1}^{-1}$ over $\mathbb{Q}$ takes time $\mathcal{O}(n^6)$, whereas computing $\boldsymbol{W_0}\boldsymbol{W_1}^{-1}$ mod 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 $\boldsymbol{W_0}$ and $\boldsymbol{W_1}$. We also provide an implementation of the variant in [CP19].

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 $\mathcal{A} := \{\boldsymbol{A}_i \in \mathbb{Z}_{x_0}^{d\times d} : i \in [nd]\}$, $\mathcal{B} = \{\boldsymbol{B}_\sigma \in \mathbb{Z}_{x_0}^{d\times d} : \sigma \in \{0,1\}\}$, and $\mathcal{C} := \{\boldsymbol{C}_j \in \mathbb{Z}_{x_0}^{d\times d} : j \in [nd]\}$, and two vectors $\boldsymbol{s} \in \mathbb{Z}_{x_0}^d$ and $\boldsymbol{t} \in \mathbb{Z}_{x_0}^d$ such that the value $w_{i,\sigma,j} := \boldsymbol{s}\boldsymbol{A}_i\boldsymbol{B}_\sigma\boldsymbol{C}_j\boldsymbol{t} \pmod{x_0}$ is a zero-tested top-level encoding of zero. The attack then proceeds as previously by computing two matrices $\boldsymbol{W}_\sigma \in \mathbb{Z}^{nd\times nd}$ (for $\sigma \in \{0,1\}$) whose each entry is defined as $\boldsymbol{W}_\sigma[i,j] = w_{i,\sigma,j}$, then computing the matrix $\boldsymbol{W} := \boldsymbol{W_0}\boldsymbol{W_1}^{-1}$ over $\mathbb{Q}$. As previously we can write:

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

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

$$\boldsymbol{W} = \boldsymbol{W_0}\boldsymbol{W_1}^{-1} = \boldsymbol{A}\bar{\boldsymbol{B}}_0\bar{\boldsymbol{B}}_1^{-1}\boldsymbol{A}^{-1}$$

The characteristic polynomial $f(X)$ of $\boldsymbol{W}$ is the same as the characteristic polynomial of $\bar{\boldsymbol{B}}_0\bar{\boldsymbol{B}}_1^{-1}$, which is the product of the n characteristic polynomials $f_i(X)$ of the matrices $\tilde{\boldsymbol{B}}_i = (\boldsymbol{B_0} \bmod p_i) \cdot (\boldsymbol{B_1} \bmod p_i)^{-1}$. By the Cayley-Hamilton theorem, we must have $f_i(\tilde{\boldsymbol{B}}_i) = 0$ for all $1 \le i \le n$. This implies $f_i(\boldsymbol{B_0} \cdot \boldsymbol{B_1}^{-1} \bmod x_0) = 0 \pmod{p_i}$. 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 $\boldsymbol{M_i} = f_i(\boldsymbol{B_0} \cdot \boldsymbol{B_1}^{-1} \bmod x_0)$ with x_0. We provide the source code of the attack in [CP19].

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 $G_k(\boldsymbol{B}_0 \cdot \boldsymbol{B}_1^{-1} \bmod x_0) = 0$ modulo all primes except one, and therefore the remaining prime p_i can be recovered by computing the gcd of the entries of $\boldsymbol{M}_k = G_k(\boldsymbol{B}_0 \cdot \boldsymbol{B}_1^{-1}) \bmod x_0$ with x_0.

Variant without $\boldsymbol{B_0}\boldsymbol{B_1^{-1}}$ mod $\boldsymbol{x_0}$. We describe an alternative attack in which one does not need to compute the matrix $\boldsymbol{B}_0\boldsymbol{B}_1^{-1} \bmod x_0$; only the matrix $\boldsymbol{W}$ 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 $\boldsymbol{M}_k = G_k(\boldsymbol{B}_0 \cdot \boldsymbol{B}_1^{-1}) \bmod x_0$, we compute the matrices:

$$\boldsymbol{M}'_k = G_k(\boldsymbol{W}) \cdot \boldsymbol{W}_0 \bmod x_0$$

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

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

The characteristic polynomial of $\boldsymbol{W}$ is the same as the one of $\bar{\boldsymbol{B}}_0 \cdot \bar{\boldsymbol{B}}_1^{-1}$. Therefore, by the Cayley-Hamilton theorem, all the blocks on the diagonal of $G_k(\bar{\boldsymbol{B}}_0 \cdot \bar{\boldsymbol{B}}_1^{-1})$ are zero except the block corresponding to $\tilde{\boldsymbol{B}}_i = (\boldsymbol{B_0} \bmod p_i) \cdot (\boldsymbol{B_1} \bmod p_i)^{-1}$ for some i. When multiplying by $\bar{\boldsymbol{B}}_0$, such block is multiplied by $\xi_i \cdot (\boldsymbol{B}_0 \bmod p_i) \in \mathbb{Z}^{d\times d}$. Therefore, the resulting block is a multiple of ξ_i, while all the other blocks are zero. This implies that all entries of $\boldsymbol{M}'_k$ 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 an implementation of this variant in [CP19].

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 (24) we use $\mathsf{sk} \in \{0,1\}^\ell$ to compute the product matrices in each row, with:

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

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:

$$\omega = \bar{\boldsymbol{s}}^{(1)} \prod_{i=1}^{\ell} \boldsymbol{C}^{(1)}_{i,sk[i]} \bar{\boldsymbol{t}}^{(1)} - \bar{\boldsymbol{s}}^{(2)} \prod_{i=1}^{\ell} \boldsymbol{C}^{(2)}_{i,sk[i]} \bar{\boldsymbol{t}}^{(2)} \pmod{x_0}. \tag{25}$$

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 an 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 $\mathsf{sk}^{(1)}$, $\mathsf{sk}^{(2)}$, $\mathsf{sk}^{(3)}$ and a given row u, we let the matrices $\boldsymbol{A}_i := \prod_{j=1}^{\mu} \boldsymbol{C}^{(u)}_{\phi(i,1,j),\mathsf{sk}^{(1)}_j}$, $\boldsymbol{B}_i := \prod_{j=1}^{\mu} \boldsymbol{C}^{(u)}_{\phi(i,2,j),\mathsf{sk}^{(2)}_j}$, $\boldsymbol{C}_i := \prod_{j=1}^{\mu} \boldsymbol{C}^{(u)}_{\phi(i,3,j),\mathsf{sk}^{(3)}_j}$, where the function

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

is used to access the matrices, where $1 \le r \le k$ is the repetition index, $1 \le v \le N$ is the user index, and $1 \le j \le \mu$ is the bit index in $\mathsf{sk}^{(v)}$. Therefore $\boldsymbol{A}_i$ is the i-th matrix of the first user computed using secret $\mathsf{sk}^{(1)}$ on row u, $\boldsymbol{B}_i$ is the i-th matrix of the second user, and likewise for $\boldsymbol{C}_i$. Thus, given the number of repetitions k, the product of all matrices with respect to $\mathsf{sk}^{(1)}$, $\mathsf{sk}^{(2)}$, and $\mathsf{sk}^{(3)}$ on row u can be written as $\prod_{i=1}^{k} \boldsymbol{A}_i\boldsymbol{B}_i\boldsymbol{C}_i$, where $\boldsymbol{C}_k$ 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 $\boldsymbol{ABC}$ where $\boldsymbol{A}$ is an $m \times m^{2k-1}$ matrix depending only on $\mathsf{sk}^{(1)}$, $\boldsymbol{B}$ is an $m^{2k-1} \times m^{2k-1}$ matrix depending only of $\mathsf{sk}^{(2)}$, and $\boldsymbol{C}$ is an $m^{2k-1} \times 1$ matrix (column vector) depending only of $\mathsf{sk}^{(3)}$; see Appendix B for the details. Since we must compute the difference of two rows as in (25), we must consider matrices of dimension $d = 2m^{2k-1}$ instead of m^{2k-1}. Therefore, the final matrices $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ 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 [CP19], including the variant without $\boldsymbol{B}_0\boldsymbol{B}_1^{-1} \bmod x_0$ described in the previous section; in the latter case, the attack recovers the primes p_i's from the matrices $\boldsymbol{W_0}$ and $\boldsymbol{W_1}$ 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 $\boldsymbol{W_0}$ and $\boldsymbol{W_1}$. While the attack also requires to invert $\boldsymbol{W_1}$, 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 $\boldsymbol{W}_0$ and $\boldsymbol{W}_1$ have $d'^2 = 4n^2m^{4k-2}$ 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 our experiments with the Sage library, 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 $T_{Cheon}(\eta, n, m, k) \geq 2^\lambda$.

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 of elements

For the bookend vectors, the components are CLT13-encoded with random noise of size ρ_b bits. Letting α be the size of the g_i's, for simplicity we take $\rho_b = \alpha$. Therefore the encoded bookend vectors have $\alpha \cdot (2m/3) + \rho_b \cdot m = 5\alpha m/3$ bits of entropy on each slot. For the matrices, we can use a much smaller encoding noise thanks to the analysis from Section 4.4. On a single slot, the matrices $\boldsymbol{A}_{i,b}^{(u)}$ have entropy $\simeq \alpha \cdot m^2/3$, and when CLT13-encoded with noise ρ_m, the matrices $\tilde{\boldsymbol{A}}_{i,b}^{(u)}$ have entropy $\simeq \alpha \cdot m^2/3 + \rho_m \cdot m^2$ on each slot; the GCD attack complexity is therefore $\tilde{\mathcal{O}}(2^{m^2 \cdot (\rho_m + \alpha/3)/2})$. For the parameters from Table 7 below, it suffices to take $\rho_m = 2$ to prevent GCD attacks. [2]

7.2 Number of matrices per level

Instead of taking only two matrices $\boldsymbol{A}_{i,0}^{(u)}$, $\boldsymbol{A}_{i,1}^{(u)}$ 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 $\tau > 1$.

[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.

[2] In a previous version of this paper, we used an aggressive optimization in which the matrices were CLT13 encoded without noise. However this leads to an attack; we refer to Appendix C for a description of the optimization and the attack.

7.3 Other attacks

Orthogonal lattice attack on zero-tested values. There is an orthogonal lattice attack against the values obtained by subtracting two zero-tested last-level encodings from two different rows. The attack is analogous to the attack described in Section 3.3, and is prevented under the condition $n = \omega(\frac{\nu^2}{\eta-\nu} \log \lambda)$, where ν is the number of extracted bits in the zero-tested values.

Meet-in-the-middle attack. Given the matrix products $\boldsymbol{D}_r^{(u\to v)}$ published by each party u corresponding to his secret $\mathsf{sk}^{(u)}$, there is a meet-in-the-middle attack that can recover $\mathsf{sk}^{(u)}$. Since each $\mathsf{sk}^{(u)}$ has length $\mu \cdot \tau$ bits, the attack's complexity is $\mathcal{O}(2^{\mu\cdot\tau/2})$. More precisely, the attack complexity is at least

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

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

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 $\geq 2^\lambda$ clock cycles. In the construction 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 $\tau\mu = 3\mu$ bits. The size η of the primes p_i is adjusted so that we extract $\nu = \lambda$ bits. During the publish phase, each party must broadcast $k \cdot (N - 1)$ matrices of dimension $m \times m$ and γ-bit entries. The size of those broadcasted values along with the other parameters are shown in Table 7.

	λ	η	m	n	μ	α	k	$\gamma = n \cdot \eta$	ℓ	N_{CLT13}	params	broadcast
Small	52	1759	6	160	15	11	2	$281 \cdot 10^3$	120	$1.4 \cdot 10^5$	4.8 GB	7.6 MB
Medium	62	2602	6	294	21	12	2	$764 \cdot 10^3$	168	$1.9 \cdot 10^5$	18.5 GB	20 MB
Large	72	3761	6	1349	27	14	2	$5073 \cdot 10^3$	216	$2.5 \cdot 10^5$	157.8 GB	137 MB
High	82	5159	9	4188	33	16	2	$21605 \cdot 10^3$	264	$6.8 \cdot 10^5$	1848.0 GB	1312 MB

Table 7. Concrete parameters for a 4-party key-exchange.

The main difference with the original (insecure) key-exchange protocol from [CLT13] is that we get a much larger public parameter size; for $\lambda = 62$ bits of security, we need 18 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 $\ell = 168$, one would need primes p_i of size $\eta \simeq (\alpha + \rho) \cdot \ell \simeq 2.4 \cdot 10^4$ with $\alpha = 80$ and $\rho = 62$ as in [CLT13]. Since $\gamma = \omega(\eta^2 \log \lambda)$ in [CLT13], one would need $\gamma \simeq 4 \cdot 10^9$. With $N_{CLT13} = 1.9 \cdot 10^5$, 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 Core i5-8600K CPU (3.60GHz), 32 GB of RAM, and Ubuntu 18.04.2 LTS. The execution times are shown in Table 8. We could not run the Large and High instantiations ($\lambda = 72$ and $\lambda = 82$) because of the huge parameter size. While the Setup time is significant, since we need to sample all the random values and perform expensive operations like CRT and inverting matrices, the Publish and KeyGen times remain reasonable. In fact, each user just has to multiply $m \times m$

matrices $\mu \cdot k \cdot (N-1)$ times to publish their values and $k \cdot (\mu + N)$ times to derive the shared key. We provide the source code of the key-exchange in [CP19].

	Setup (once)	Publish (per party)	KeyGen (per party)
Small	2 h 20 min	45 s	19 s
Medium	12 h 23 min	3 min 35 s	1 min 24 s

Table 8. 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 *ACM CCS*. 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*, 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*, 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*, 2014.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based snargs and their application to more efficient obfuscation. In *Advances in Cryptology - EUROCRYPT 2017 - Proceedings, Part III*, 2017.

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In *EUROCRYPT (2)*, volume 9666 of *LNCS*. Springer, 2016.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324, 2003.

[CGH$^+$15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède 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 *CRYPTO 2015, Part I*, volume 9215 of *LNCS*. 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*, 2017.

[CHL$^+$15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*. Springer, 2015.

[CLLT16] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In *Advances in Cryptology - CRYPTO 2016 - Proceedings, Part II*, 2016.

[CLLT17] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. In *Public-Key Cryptography - PKC 2017 - Proceedings, Part I*, 2017.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, volume 8042 of *LNCS*. Springer, 2013.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *Advances in Cryptology - ASIACRYPT 2011 - Proceedings*, 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*. Springer, 2012.

[CP19] Jean-Sebastien Coron and Hilder V. L. Pereira. Implementation of key-exchange based on CLT13 multilinear maps, 2019. Publicly available at https://github.com/coron/cltexchangeimpl.

[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*, 2018.

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

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, volume 7881 of *LNCS*. 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*. IEEE Computer Society, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, volume 9015 of *LNCS*, 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), 2016.

[HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In *Advances in Cryptology - EUROCRYPT 2016 - Proceedings, Part I*, 2016.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *CRYPTO 2011*, 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*, 1988.

[Lau04] Alan J. Laub. *Matrix Analysis For Scientists And Engineers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2004.

[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, 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*, 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*, 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*, 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, Proceedings*, 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*, 2001.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO (1)*, volume 8616 of *LNCS*. 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, New York, NY, USA, 2014. ACM.

A Kronecker product of matrices

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

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

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

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

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

$$\mathsf{vec}(\boldsymbol{A} \cdot \boldsymbol{B} \cdot \boldsymbol{C}) = (\boldsymbol{C}^T \otimes \boldsymbol{A}) \cdot \mathsf{vec}(\boldsymbol{B})$$

B The tensoring attack

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

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

with $\tilde{\boldsymbol{A}}$, $\tilde{\boldsymbol{B}}$, and $\tilde{\boldsymbol{C}}$ 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 $\tilde{\boldsymbol{C}}$ is a column vector, we have

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

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

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

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

C Encoding the matrices without noise

In a previous version of this paper, we used an aggressive optimization in which the matrices $\boldsymbol{A}_{i,b}^{(u)}$ were CLT13-encoded without noise, in order to reduce the bitsize η of the primes p_i. Below we provide a short description of that optimization and an attack that completely breaks it.

Encoding without noise. One could try to encode the matrices $\boldsymbol{A}_{i,b}^{(u)}$ from (21) without any additional randomness, while relying on the intrinsic randomness of the matrices $\boldsymbol{A}_{i,b}^{(u)}$ and on Kilian's randomization on 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 then 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 $1 \leq i \leq n$.

Attack. The above optimization is broken by the following attack. Namely without noise an encoding of zero is simply 0 modulo x_0. Since the matrices $\boldsymbol{A}_{i,b}^{(u)}$ are block-diagonal with three $(m/3) \times (m/3)$ blocks on the diagonal, the encoded matrices $\tilde{\boldsymbol{A}}_{i,b}^{(u)}$ are then also block-diagonal. These matrices are hidden by the Kilian matrices, as follows:

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

By multiplying consecutive matrices for a given user u and various bits b, one can obtain a large set of ℓ known matrices $\boldsymbol{C}_i$ of the form:

$$\boldsymbol{C}_i = \boldsymbol{K_0} \cdot \boldsymbol{A_i} \cdot \boldsymbol{K_1}^{-1} \pmod{x_0}$$

where the unknown matrices $\boldsymbol{A_i}$ are as previously block-diagonal, and $\boldsymbol{K_0}$ and $\boldsymbol{K_1}$ are unknown $m \times m$ matrices. We can rewrite the previous equation as:

$$\boldsymbol{A}_i = \boldsymbol{K_0}^{-1} \cdot \boldsymbol{C_i} \cdot \boldsymbol{K_1} \pmod{x_0}$$

for $1 \leq i \leq \ell$. Since the matrices $\boldsymbol{A}_i$ have six $(m/3) \times (m/3)$ blocks outside the diagonal equal to 0, we can obtain a system of $\ell \cdot 6 \cdot m^2/9$ quadratic equations in the coefficients of $\boldsymbol{K_0}^{-1}$ and $\boldsymbol{K_1}$. By linearizing the system, we obtain a system of $\ell \cdot 6 \cdot m^2/9$ equations in $(m^2)^2 = m^4$ unknowns. The system can then be solved by linear algebra if $\ell \cdot 6 \cdot m^2/9 > m^4$, which gives the condition $\ell > 3m^2/2$. Once the Kilian matrices $\boldsymbol{K_0}$ and $\boldsymbol{K_1}$ have been recovered (up to a scalar value), the remaining secret CLT13 parameters can be recovered by solving the Approximate-GCD problem with much smaller parameters than in the original CLT13.