Cryptanalysis of GGH15 Multilinear Maps

Jean-Sébastien Coron[1], Moon Sung Lee[1], Tancrède Lepoint[2], and Mehdi Tibouchi[3]

[1] University of Luxembourg
[2] CryptoExperts
[3] NTT Secure Platform Laboratories

April 13, 2016

Abstract. We describe a cryptanalysis of the GGH15 multilinear maps. Our attack breaks the multipartite key-agreement protocol in polynomial time by generating an equivalent user private key; it also applies to GGH15 with safeguards. We also describe attacks against variants of the GGH13 multilinear maps proposed by Halevi (ePrint 2015/866) aiming at supporting graph-induced constraints, as in GGH15.

1 Introduction

Multilinear maps. For the past couple of years, cryptographic multilinear maps have found numerous applications in the design of cryptographic protocols, the most salient example of which is probably the construction of indistinguishability obfuscation (iO) [GGH+13b]. The first multilinear maps candidate (GGH13) was described by Garg, Gentry and Halevi [GGH13a] from ideal lattices. It was then followed by another candidate (aka, CLT13) due to Coron, Lepoint and Tibouchi [CLT13] using the same techniques but over the integers, and later by a third candidate (GGH15) by Gentry, Gorbunov and Halevi [GGH15], related to the homomorphic encryption scheme from [GSW13].

Unfortunately, these candidates do not rely on well-established hardness assumptions, and recent months have witnessed a number of attacks (including [CHL+15], [CGH+15], [HJ15], [BGH+15], [PS15], [MF15], [CLR15]) showing that they fail to meet a number of desirable security requirements, and that they cannot be used to securely instantiate such and such protocols. Some attempts to protect against these attacks have also known a similar fate [CLT15, BGH+15]. The security of the constructions based on these multilinear maps is currently unclear to the community [Hal15a]. While two recent works [CGH+15, MSZ16] have shown polynomial-time attacks against some obfuscation candidates, many iO candidates remain unaffected by the attacks proposed so far. The same cannot be said for the more immediate application of multilinear maps that is one-round multipartite key agreement.

One-round multipartite key-agreement protocol. Since its discovery in 1976, the Diffie–Hellman protocol [DH76] is one of the most widely used cryptographic protocol to create a common secret between two parties. A generalization of this one-round protocol to three parties was proposed in 2000 by Joux [Jou00] using cryptographic *bilinear* maps; it was later extended to $k \geq 4$ parties assuming the existence of a cryptographic $(k-1)$-linear map by Boneh and Silverberg [BS02]. In a nutshell, the protocol works as follows: assuming some public parameters are shared by all the parties, each party broadcasts some data and keeps some data secret, and then by combining their secret data with the other parties' published values using the multilinear map, they can derive a shared common secret key.

The first candidates for a k-partite Diffie–Hellman key-agreement protocol for arbitrary k were described in [GGH13a, CLT13] using respectively the GGH13 and CLT13 multilinear maps candidates. Unfortunately, the protocols were later shown to be insecure in [HJ15, CHL$^+$15]: using the public parameters and the broadcast data, an eavesdropper can recover the shared common secret key in polynomial time.

The GGH15 key-agreement protocol. Since the third proposed multilinear maps scheme, GGH15, does not fit the same graded encoding framework as the earlier candidates, one needs new constructions to use it to instantiate cryptographic protocols. And the first such application was again a Diffie–Hellman key-agreement protocol [GGH15, Section 5.1]. To avoid similar attacks as the one that targeted GGH13 and CLT13, based on encodings of zero, the protocol was designed in such a way that the adversary is never given encodings of the same element that could be subtracted without doing the full key-agreement computation. Namely, each party i has a directed path of matrices $\boldsymbol{A}_{i,1}, \dots, \boldsymbol{A}_{i,k+1}$ all sharing the same end-point $\boldsymbol{A}_{i,k+1} = \boldsymbol{A}_0$, and has a secret value s_i. She can then publish encodings of s_i on the chains of the other parties in a "round robin" fashion, *i.e.* s_i is encoded on the j-th edge of the chain of the party $i' = j - i + 1$, with index arithmetic modulo k. The graph for 3 parties is illustrated in Figure 1.

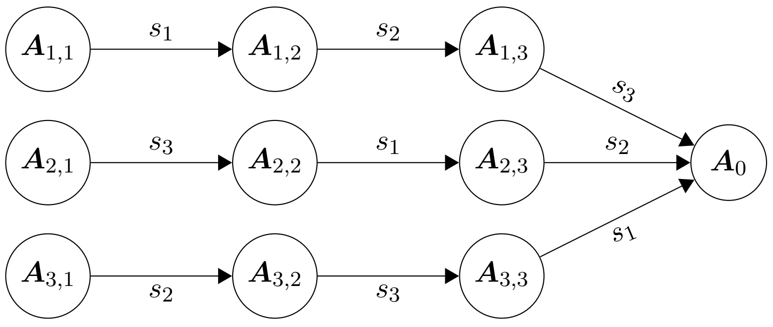

Fig. 1. Graph for a 3-partite key-agreement protocol with GGH15 multilinear maps.

On the i-th chain, Party i will then be able to multiply these encodings (the one he kept secret and the ones published by the other parties) to get an encoding of $\prod_j s_j$ relative to the path $\boldsymbol{A}_{i,1} \rightsquigarrow \boldsymbol{A}_0$. Now, since the encodings of s_i cannot be mixed before the end-point $\boldsymbol{A}_0$, it seems difficult to obtain an encoding of 0 on an edge in the middle of the graph to mount "zeroizing attacks" [GGH15].

Halevi's candidate key-agreement protocols. As no attack was known on GGH15 multilinear maps and in an attempt to reinstate a key-agreement protocol for GGH13, Halevi recently proposed, on the Cryptology ePrint Archive, two variants of GGH13 supporting a similar key-agreement protocol [Hal15a].[1] The first variant uses the "asymmetric" GGH13 scheme to handle the graph structure [Hal15a, Section 7]. Namely, in basic GGH13 each encoding is multiplicatively masked

[1] As mentioned in the last remark of the paper, although the key-agreement protocol can be described also based on CLT13, the attacks from [CGH$^+$15] can be used to break it.

by a power z^i of a secret mask z; in asymmetric GGH13, the encodings can be masked by powers of multiple z_j's. Therefore, in this new key-agreement protocol candidate, the public encodings are now associated with independent masks $z_{i,j}$'s such that their product yields the same value Z, *i.e.* $\prod_j z_{i,j} = Z$ for all i (so that the final encoding shall extract to the same shared key). The graph for 3 parties is illustrated in Figure 2.

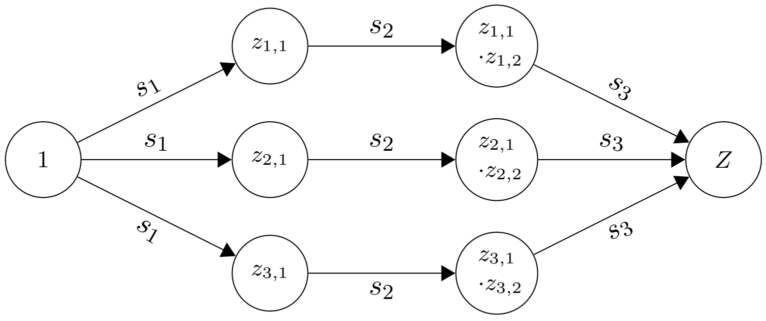

Fig. 2. Multipartite key agreement from asymmetric GGH13, with 3 parties, from [Hal15a, Section 7].

Once again, the fact that the encodings of the same value s_i are multiplied with different masks gives hope that no encoding of 0 multiplied by a value other than Z can be obtained, and therefore that zeroizing attacks are impossible [GGH13a, CGH$^+$15].

A second variant of GGH13, which we refer to as Graph-GGH13, mimics the structure of GGH15 encodings more closely and is described in [Hal15a, Section 6]. An encoding $c \in \alpha + gR$ relative to a path $u \rightsquigarrow v$ is now a matrix $\tilde{\boldsymbol{C}} = \boldsymbol{P}_u^{-1} \cdot \boldsymbol{C} \cdot \boldsymbol{P}_v$, where $\boldsymbol{C} \in \mathbb{Z}_q^{n \times n}$ is the multiply-by-c matrix, and the $\boldsymbol{P}_w$'s are secret random matrices. In the key-agreement protocol, each party i has a directed path of matrices $\boldsymbol{P}_{i,1}, \ldots, \boldsymbol{P}_{i,k+1}$ all sharing the same end-point $\boldsymbol{P}_{i,k+1} = \boldsymbol{P}_0$ and the same start-point $\boldsymbol{P}_{i,1} = \boldsymbol{P}_1$, and has a secret value s_i. She can then publish encodings of s_i on the chains of the other parties in a "round robin" fashion. The graph for 3 parties is illustrated in Figure 3.

$$\begin{tikzcd}
 & \boldsymbol{P}_{1,2} \arrow[r, "s_2"] & \boldsymbol{P}_{1,3} \arrow[dr, "s_3"] & \\
\boldsymbol{P}_1 \arrow[ur, "s_1"] \arrow[r, "s_3"] \arrow[dr, "s_2"'] & \boldsymbol{P}_{2,2} \arrow[r, "s_1"] & \boldsymbol{P}_{2,3} \arrow[r, "s_2"] & \boldsymbol{P}_0 \\
 & \boldsymbol{P}_{3,2} \arrow[r, "s_3"'] & \boldsymbol{P}_{3,3} \arrow[ur, "s_1"'] &
\end{tikzcd}$$

Fig. 3. Multipartite key agreement from GGH13 with graph constraints, with 3 parties, from [Hal15a, Section 6].

And here again, the fact that the encodings corresponding to the same s_i are multiplied on the left and on the right by completely random matrices $\boldsymbol{P}_{i,j}$ makes it difficult to cancel them out and obtain an encoding of 0 without evaluating the full "chains" (that is, the operations of the key agreement itself).

Finally, in order to capture the intuition of what it means for an attacker to break the scheme, Halevi defined, for both schemes, the "core computational task" of an adversary as recovering any basis of the (hidden) plaintext space [Hal15a, Section 2.2].

Our contributions. Our main contribution is to describe a cryptanalysis of the Diffie–Hellman key-agreement protocol when instantiated with GGH15 multilinear maps. Our attack makes it possible to generate an equivalent user private key in polynomial time, which in turn allows to recover the shared session key. Our attack proceeds in two steps: in the first step, we express the secret exponent of one user as a linear combination of some other secret exponents corresponding to public encodings, using a variant of the Cheon *et al.* attack [CHL+15]. This does not immediately break the protocol because the coefficients of the linear combination can be large. In the second step, we use the previous linear combination to derive an encoding equivalent to the user private encoding, by correcting the error resulting from the large coefficients of the linear combination. Our attack also applies to GGH15 with safeguards; we extend the basic attack by using another linear relation to estimate the error incurred from the large coefficients, thus enabling to recover the shared session key.

In Appendix B, we also describe attacks that break both variants of GGH13 proposed by Halevi in [Hal15a]. Our attacks apply some variant of the Cheon *et al.* attack [CHL+15] to recover a basis of the secret plaintext space R/gR in polynomial time. This was considered as the "core computational task of an attacker" in [Hal15a].

Source code. A proof-of-concept implementation of our cryptanalysis of GGH15, using the Sage [Dev16] mathematics software system, is available at:

`http://pastebin.com/7kZHnTXY`

2 The GGH15 Multilinear Map Scheme

We briefly recall the GGH15 multilinear map scheme; we refer to [GGH15] for a full description. In the following we only consider the commutative variant from [GGH15, Section 3.2], as only that commutative variant can be used in the multipartite key-agreement protocol from [GGH15, Section 5.1].

2.1 GGH15 Multilinear Maps

The construction works over polynomial rings $R = \mathbb{Z}[x]/(f(x))$ and $R_q = R/qR$ for some degree n irreducible integer polynomial $f(x) \in \mathbb{Z}[x]$ and an integer q. The construction is parametrized by a directed acyclic graph $G = (V, E)$. To each node $u \in V$ a random row vector $\boldsymbol{A}_u \in \mathbb{Z}_q^m$ is assigned, where m is a parameter. An encoding of a small plaintext element $s \in R$ relative to path $u \rightsquigarrow v$ is a matrix with small coefficients $\boldsymbol{D} \in R^{m \times m}$ such that:

$$\boldsymbol{A}_u \cdot \boldsymbol{D} = s \cdot \boldsymbol{A}_v + \boldsymbol{E} \pmod{q}$$

where $\boldsymbol{E}$ is a small error vector of dimension m; we refer to [GGH15] for how such encoding $\boldsymbol{D}$ can be generated, based on a trapdoor sampling procedure from [MP12]. Only small plaintext elements

$s \in R$ are encoded. As in [Hal15a] we use the row vector notation for $\boldsymbol{A}_u$, rather than the column vector notation used in [GGH15].[2] It is easy to see that two encodings $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ relative to the same path $u \rightsquigarrow v$ can be added; namely from:

$$\begin{aligned}\boldsymbol{A}_u \cdot \boldsymbol{D}_1 &= s_1 \cdot \boldsymbol{A}_v + \boldsymbol{E}_1 \pmod q\\ \boldsymbol{A}_u \cdot \boldsymbol{D}_2 &= s_2 \cdot \boldsymbol{A}_v + \boldsymbol{E}_2 \pmod q\end{aligned}$$

we obtain:

$$\boldsymbol{A}_u \cdot (\boldsymbol{D}_1 + \boldsymbol{D}_2) = (s_1 + s_2) \cdot \boldsymbol{A}_v + \boldsymbol{E}_1 + \boldsymbol{E}_2 \pmod q\,.$$

Moreover two encodings $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ relative to path $u \rightsquigarrow v$ and $v \rightsquigarrow w$ can be multiplied to get an encoding relative to path $u \rightsquigarrow w$. Namely given:

$$\begin{aligned}\boldsymbol{A}_u \cdot \boldsymbol{D}_1 &= s_1 \cdot \boldsymbol{A}_v + \boldsymbol{E}_1 \pmod q\\ \boldsymbol{A}_v \cdot \boldsymbol{D}_2 &= s_2 \cdot \boldsymbol{A}_w + \boldsymbol{E}_2 \pmod q\end{aligned}$$

we obtain by multiplying the matrix encodings $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$:

$$\begin{aligned}\boldsymbol{A}_u \cdot \boldsymbol{D}_1 \cdot \boldsymbol{D}_2 &= (s_1 \cdot \boldsymbol{A}_v + \boldsymbol{E}_1) \cdot \boldsymbol{D}_2 \pmod q\\ &= s_1 \cdot s_2 \cdot \boldsymbol{A}_w + s_1 \cdot \boldsymbol{E}_2 + \boldsymbol{E}_1 \cdot \boldsymbol{D}_2 \pmod q\\ &= s_1 \cdot s_2 \cdot \boldsymbol{A}_w + \boldsymbol{E}' \pmod q\end{aligned}$$

for some new error vector $\boldsymbol{E}'$. Since s_1, $\boldsymbol{E}_1$, $\boldsymbol{E}_2$ and $\boldsymbol{D}_2$ have small coefficients, $\boldsymbol{E}'$ still has small coefficients (compared to q), and therefore the product $\boldsymbol{D}_1 \cdot \boldsymbol{D}_2$ is an encoding of $s_1 \cdot s_2$ for the path $u \rightsquigarrow w$.

Finally, given an encoding $\boldsymbol{D}$ relative to path $u \rightsquigarrow w$ and the vector $\boldsymbol{A}_u$, extraction works by computing the high-order bits of $\boldsymbol{A}_u \cdot \boldsymbol{D}$. Namely we have:

$$\boldsymbol{A}_u \cdot \boldsymbol{D} = s \cdot \boldsymbol{A}_w + \boldsymbol{E} \pmod q$$

for some small $\boldsymbol{E}$, and therefore the high-order bits of $\boldsymbol{A}_u \cdot \boldsymbol{D}$ only depend on the secret exponent s.

Remark 1. As emphasized in [GGH15], only the plaintext space of the s_i's is commutative, not the space of the encoding matrices $\boldsymbol{D}_i$. The ability to multiply the plaintext elements s_i in arbitrary order will be used in the multipartite key-agreement protocol below.

2.2 The GGH15 Multipartite Key-Agreement Protocol

We briefly recall the multipartite key-agreement protocol from [GGH15, Section 5.1]. We consider the protocol with k users. As illustrated in Figure 4 for $k = 3$ users, each user i for $1 \leq i \leq k$ has a directed path of vectors $\boldsymbol{A}_{i,1}, \ldots, \boldsymbol{A}_{i,k+1}$, all sharing the same end-point $\boldsymbol{A}_0 = \boldsymbol{A}_{i,k+1}$. The i-th user will use the resulting chain to extract the session key. Each user i has a secret exponent s_i. Each secret exponent s_i will be encoded in each of the k chains; the encoding of s_i on the j-th chain for $j \neq i$ will be published, while the encoding of s_i on the i-th chain will be kept private by user i. Therefore on the i-th chain only user i will be able to compute the session key. The exponents s_i are encoded in a "round robin" fashion; namely the i-th secret s_i is encoded on the chain of user j at edge $\ell = i + j - 1$, with index arithmetic modulo k. Only the vectors $\boldsymbol{A}_{i,1}$ for $1 \leq i \leq k$ are made

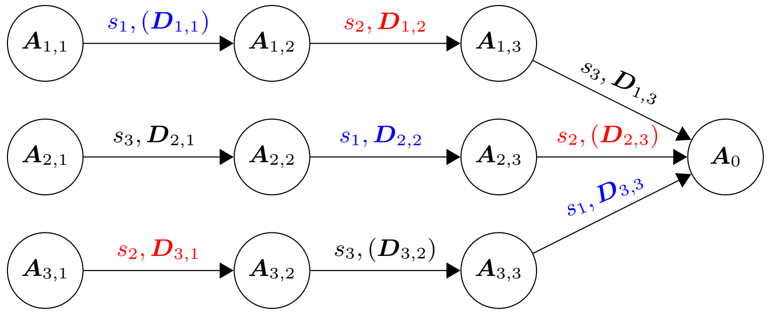

Fig. 4. Graph of a key agreement between 3 parties for GGH15. The vertices contain random vectors $\boldsymbol{A}_{ij}$, and encodings are represented on the edges. Each party is represented by a different color, keeps the encoding in parenthesis secret and publishes the two other encodings.

public to enable extraction of the session-key; the others are kept private. We recall the formal description of the protocol in Appendix A.

We illustrate the protocol for $k = 3$ users. For the chain corresponding to User 1, we have the following encodings:

$$\begin{aligned}
\boldsymbol{A}_{1,1} \cdot \boldsymbol{D}_{1,1} &= s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{F}_{1,1} \pmod q \\
\boldsymbol{A}_{1,2} \cdot \boldsymbol{D}_{1,2} &= s_2 \cdot \boldsymbol{A}_{1,3} + \boldsymbol{F}_{1,2} \pmod q \\
\boldsymbol{A}_{1,3} \cdot \boldsymbol{D}_{1,3} &= s_3 \cdot \boldsymbol{A}_0 + \boldsymbol{F}_{1,3} \pmod q
\end{aligned}$$

where $\boldsymbol{D}_{1,2}$ and $\boldsymbol{D}_{1,3}$ are public while $\boldsymbol{D}_{1,1}$ is kept private by User 1. Therefore User 1 can compute modulo q:

$$\begin{aligned}
\boldsymbol{A}_{1,1} \cdot \boldsymbol{D}_{1,1} \cdot \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}_{1,3} &= (s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{F}_{1,1}) \cdot \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}_{1,3} \pmod q \\
&= (s_1 \cdot s_2 \cdot \boldsymbol{A}_{1,3} + s_1 \cdot \boldsymbol{F}_{1,2} + \boldsymbol{F}_{1,1} \cdot \boldsymbol{D}_{1,2}) \cdot \boldsymbol{D}_{1,3} \pmod q .
\end{aligned}$$

Letting $\hat{\boldsymbol{F}}_{1,2} := s_1 \cdot \boldsymbol{F}_{1,2} + \boldsymbol{F}_{1,1} \cdot \boldsymbol{D}_{1,2}$, we obtain:

$$\begin{aligned}
\boldsymbol{A}_{1,1} \cdot \boldsymbol{D}_{1,1} \cdot \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}_{1,3} &= \left(s_1 \cdot s_2 \cdot \boldsymbol{A}_{1,3} + \hat{\boldsymbol{F}}_{1,2}\right) \cdot \boldsymbol{D}_{1,3} \pmod q \\
&= s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0 + s_1 \cdot s_2 \cdot \boldsymbol{F}_{1,3} + \hat{\boldsymbol{F}}_{1,2} \cdot \boldsymbol{D}_{1,3} \pmod q .
\end{aligned}$$

Since s_1, s_2 and s_3 are small and $\boldsymbol{F}_{1,3}$, $\hat{\boldsymbol{F}}_{1,2}$ and $\boldsymbol{D}_{1,3}$ have small components, User 1 can extract the most significant bits corresponding to $s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0$. Similarly User 2 will compute the session key using the following chain, where $\boldsymbol{D}_{2,1}$ and $\boldsymbol{D}_{2,2}$ are public while $\boldsymbol{D}_{2,3}$ is private to User 2:

$$\begin{aligned}
\boldsymbol{A}_{2,1} \cdot \boldsymbol{D}_{2,1} &= s_3 \cdot \boldsymbol{A}_{2,2} + \boldsymbol{F}_{2,1} \pmod q \\
\boldsymbol{A}_{2,2} \cdot \boldsymbol{D}_{2,2} &= s_1 \cdot \boldsymbol{A}_{2,3} + \boldsymbol{F}_{2,2} \pmod q \\
\boldsymbol{A}_{2,3} \cdot \boldsymbol{D}_{2,3} &= s_2 \cdot \boldsymbol{A}_0 + \boldsymbol{F}_{2,3} \pmod q .
\end{aligned}$$

Namely User 2 can compute:

$$\begin{aligned}
\boldsymbol{A}_{2,1} \cdot \boldsymbol{D}_{2,1} \cdot \boldsymbol{D}_{2,2} \cdot \boldsymbol{D}_{2,3} &= (s_3 \cdot s_1 \cdot \boldsymbol{A}_{2,3} + s_3 \cdot \boldsymbol{F}_{2,2} + \boldsymbol{F}_{2,1} \cdot \boldsymbol{D}_{2,2}) \cdot \boldsymbol{D}_{2,3} \pmod q \\
&= s_3 \cdot s_1 \cdot s_2 \cdot \boldsymbol{A}_0 + \boldsymbol{F} \pmod q
\end{aligned}$$

[2] With the column vector notation, the corresponding equation in [GGH15] is $\boldsymbol{D} \cdot \boldsymbol{A}_u = s \cdot \boldsymbol{A}_v + \boldsymbol{E} \pmod q$.

for some small vector $\boldsymbol{F}$, and extract the same most significant bits corresponding to $s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0$; the same holds for User 3.

The previous encodings are generated by random linear combination of public encodings, corresponding to secret exponents $t_{i,\ell}$ for $1 \leq \ell \leq N$, for large enough N. More precisely, for each $1 \leq i \leq k$ one generates random small plaintext elements $t_{i,\ell}$ for $1 \leq \ell \leq N$, which are then encoded on all chains j at edge $i' = i + j - 1$ (with index modulo k), by $\boldsymbol{C}_{j,i',\ell}$. This means that for $k = 3$ users, we have the following encodings corresponding to User 1:

$$\begin{aligned}
\boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}_{1,1,\ell} \pmod q \\
\boldsymbol{A}_{2,2} \cdot \boldsymbol{C}_{2,2,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_{2,3} + \boldsymbol{E}_{2,2,\ell} \pmod q \\
\boldsymbol{A}_{3,3} \cdot \boldsymbol{C}_{3,3,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_{0} + \boldsymbol{E}_{3,3,\ell} \pmod q
\end{aligned}$$

and the tuple $(\boldsymbol{D}_{1,1}, \boldsymbol{D}_{2,2}, \boldsymbol{D}_{3,3})$ is generated by linear combination of the tuple $(\boldsymbol{C}_{1,1,\ell}, \boldsymbol{C}_{2,2,\ell}, \boldsymbol{C}_{3,3,\ell})$, so that the matrices $\boldsymbol{D}_{1,1}$, $\boldsymbol{D}_{2,2}$ and $\boldsymbol{D}_{3,3}$ encode the same secret exponent s_1; the same holds for users 2 and 3. We refer to Appendix A for the formal description of the protocol.

3 Cryptanalysis of GGH15 Without Safeguards

In the following we describe a cryptanalysis of the multipartite key-agreement protocol based on GGH15 multilinear maps recalled in the previous section. Heuristically our attack recovers the session-key from public element in polynomial-time. Our attack proceeds in two steps.

1. In the first step, we are able to express one secret exponent s_1 as a linear combination of the other secret exponents $t_{1,\ell}$, using a variant of the Cheon *et al.* attack [CHL+15]. However this does not immediately break the protocol, because the coefficients are not small.
2. In the second step, we compute an equivalent of the private encoding of User 1 from the previous linear combination, by correcting the error due to the large coefficients. This breaks the key-exchange protocol.

3.1 Description With 3 Users

For simplicity we first consider the protocol with only 3 users; the extension to $k \geq 3$ users is relatively straightforward and described in Appendix D. Therefore we consider the following 3 rows corresponding to the 3 users:

$$\begin{array}{ll}
\boldsymbol{A}_{1,1} \cdot \boldsymbol{D}_{1,1} = s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{F}_{1,1} \pmod q & \boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,\ell} = t_{1,\ell} \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}_{1,1,\ell} \pmod q \\
\boldsymbol{A}_{1,2} \cdot \boldsymbol{D}_{1,2} = s_2 \cdot \boldsymbol{A}_{1,3} + \boldsymbol{F}_{1,2} \pmod q & \boldsymbol{A}_{1,2} \cdot \boldsymbol{C}_{1,2,\ell} = t_{2,\ell} \cdot \boldsymbol{A}_{1,3} + \boldsymbol{E}_{1,2,\ell} \pmod q \\
\boldsymbol{A}_{1,3} \cdot \boldsymbol{D}_{1,3} = s_3 \cdot \boldsymbol{A}_{0} + \boldsymbol{F}_{1,3} \pmod q & \boldsymbol{A}_{1,3} \cdot \boldsymbol{C}_{1,3,\ell} = t_{3,\ell} \cdot \boldsymbol{A}_{0} + \boldsymbol{E}_{1,3,\ell} \pmod q \\
\\
\boldsymbol{A}_{2,1} \cdot \boldsymbol{D}_{2,1} = s_3 \cdot \boldsymbol{A}_{2,2} + \boldsymbol{F}_{2,1} \pmod q & \boldsymbol{A}_{2,1} \cdot \boldsymbol{C}_{2,1,\ell} = t_{3,\ell} \cdot \boldsymbol{A}_{2,2} + \boldsymbol{E}_{2,1,\ell} \pmod q \\
\boldsymbol{A}_{2,2} \cdot \boldsymbol{D}_{2,2} = s_1 \cdot \boldsymbol{A}_{2,3} + \boldsymbol{F}_{2,2} \pmod q & \boldsymbol{A}_{2,2} \cdot \boldsymbol{C}_{2,2,\ell} = t_{1,\ell} \cdot \boldsymbol{A}_{2,3} + \boldsymbol{E}_{2,2,\ell} \pmod q \\
\boldsymbol{A}_{2,3} \cdot \boldsymbol{D}_{2,3} = s_2 \cdot \boldsymbol{A}_{0} + \boldsymbol{F}_{2,3} \pmod q & \boldsymbol{A}_{2,3} \cdot \boldsymbol{C}_{2,3,\ell} = t_{2,\ell} \cdot \boldsymbol{A}_{0} + \boldsymbol{E}_{2,3,\ell} \pmod q \\
\\
\boldsymbol{A}_{3,1} \cdot \boldsymbol{D}_{3,1} = s_2 \cdot \boldsymbol{A}_{3,2} + \boldsymbol{F}_{3,1} \pmod q & \boldsymbol{A}_{3,1} \cdot \boldsymbol{C}_{3,1,\ell} = t_{2,\ell} \cdot \boldsymbol{A}_{3,2} + \boldsymbol{E}_{3,1,\ell} \pmod q \\
\boldsymbol{A}_{3,2} \cdot \boldsymbol{D}_{3,2} = s_3 \cdot \boldsymbol{A}_{3,3} + \boldsymbol{F}_{3,2} \pmod q & \boldsymbol{A}_{3,2} \cdot \boldsymbol{C}_{3,2,\ell} = t_{3,\ell} \cdot \boldsymbol{A}_{3,3} + \boldsymbol{E}_{3,2,\ell} \pmod q \\
\boldsymbol{A}_{3,3} \cdot \boldsymbol{D}_{3,3} = s_1 \cdot \boldsymbol{A}_{0} + \boldsymbol{F}_{3,3} \pmod q & \boldsymbol{A}_{3,3} \cdot \boldsymbol{C}_{3,3,\ell} = t_{1,\ell} \cdot \boldsymbol{A}_{0} + \boldsymbol{E}_{3,3,\ell} \pmod q
\end{array}$$

where all encodings $\boldsymbol{C}_{i,j,\ell}$ and $\boldsymbol{D}_{i,j}$ are public, except $\boldsymbol{D}_{1,1}$ which is private on Row 1, $\boldsymbol{D}_{2,3}$ is private on Row 2, and $\boldsymbol{D}_{3,2}$ is private on Row 3. The corresponding graph is illustrated in Figure 4. Note that on each row we have used the same index ℓ for $t_{1,\ell}$, $t_{2,\ell}$ and $t_{3,\ell}$, but on a given row one can obviously compute product of encodings for different indices.

First step: linear relations. In the first step of the attack, we show that we can express s_1 as a linear combinations of the $t_{1,\ell}$'s. For this we consider the rows 2 and 3, for which the encodings $\boldsymbol{D}_{2,2}$ and $\boldsymbol{D}_{3,3}$ corresponding to s_1 are public. In the remaining of the attack, we always consider a fixed index $\ell = 1$ for the encodings corresponding to $t_{3,\ell}$, and for simplicity we write $t_3 := t_{3,1}$, $\boldsymbol{C}_{1,3} := \boldsymbol{C}_{1,3,1}$, $\boldsymbol{C}_{2,1} := \boldsymbol{C}_{2,1,1}$ and $\boldsymbol{C}_{3,2} := \boldsymbol{C}_{3,2,1}$.

Since we always work with the same t_3, on Row 2 we define the product encodings $\hat{\boldsymbol{C}}_{2,2,\ell} := \boldsymbol{C}_{2,1} \cdot \boldsymbol{C}_{2,2,\ell}$, and on Row 3 we define the product encodings $\hat{\boldsymbol{C}}_{3,2,\ell} := \boldsymbol{C}_{3,1,\ell} \cdot \boldsymbol{C}_{3,2}$; recall that we use a fixed index for t_3. Therefore we can write:

$$\begin{aligned} \boldsymbol{A}_{2,1} \cdot \hat{\boldsymbol{C}}_{2,2,\ell} &= t_{1,\ell} \cdot t_3 \cdot \boldsymbol{A}_{2,3} + \hat{\boldsymbol{E}}_{2,2,\ell} \pmod q \\ \boldsymbol{A}_{2,3} \cdot \boldsymbol{C}_{2,3,\ell} &= t_{2,\ell} \cdot \boldsymbol{A}_0 + \boldsymbol{E}_{2,3,\ell} \pmod q \\ \boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,\ell} &= t_{2,\ell} \cdot t_3 \cdot \boldsymbol{A}_{3,3} + \hat{\boldsymbol{E}}_{3,2,\ell} \pmod q \\ \boldsymbol{A}_{3,3} \cdot \boldsymbol{C}_{3,3,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_0 + \boldsymbol{E}_{3,3,\ell} \pmod q \end{aligned} \tag{1}$$

for some small error vectors $\hat{\boldsymbol{E}}_{2,2,\ell}$ and $\hat{\boldsymbol{E}}_{3,2,\ell}$.

For simplicity of notations, we first consider a fixed index i for the encodings corresponding to $t_{1,i}$, and we write $t_1 := t_{1,i}$, $\hat{\boldsymbol{C}}_{2,2} := \hat{\boldsymbol{C}}_{2,2,i}$ and $\boldsymbol{C}_{3,3} := \boldsymbol{C}_{3,3,i}$. Similarly we consider a fixed index j for the encodings corresponding to $t_{2,j}$ and we write $t_2 := t_{2,j}$, $\boldsymbol{C}_{2,3} := \boldsymbol{C}_{2,3,j}$ and $\hat{\boldsymbol{C}}_{3,2} := \hat{\boldsymbol{C}}_{3,2,j}$. We use similar notations for the corresponding error vectors.

All previous equations hold modulo q only. To get a result over R instead of only modulo q, we compute the difference between two rows, for the same product of secret exponents. More precisely, we compute:

$$\begin{aligned} \boldsymbol{\omega} &= \boldsymbol{A}_{2,1} \cdot \hat{\boldsymbol{C}}_{2,2} \cdot \boldsymbol{C}_{2,3} - \boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2} \cdot \boldsymbol{C}_{3,3} && (2) \\ &= t_1 \cdot t_3 \cdot t_2 \cdot \boldsymbol{A}_0 + t_1 \cdot t_3 \cdot \boldsymbol{E}_{2,3} + \hat{\boldsymbol{E}}_{2,2} \cdot \boldsymbol{C}_{2,3} \\ &\quad - t_2 \cdot t_3 \cdot t_1 \cdot \boldsymbol{A}_0 - t_2 \cdot t_3 \cdot \boldsymbol{E}_{3,3} - \hat{\boldsymbol{E}}_{3,2} \cdot \boldsymbol{C}_{3,3} \\ &= t_1 \cdot t_3 \cdot \boldsymbol{E}_{2,3} + \hat{\boldsymbol{E}}_{2,2} \cdot \boldsymbol{C}_{2,3} - t_2 \cdot t_3 \cdot \boldsymbol{E}_{3,3} - \hat{\boldsymbol{E}}_{3,2} \cdot \boldsymbol{C}_{3,3}\,. && (3) \end{aligned}$$

Namely the latter equation holds over R (and not only modulo q) because all the terms in (3) have small coefficients; namely the only term $t_1 \cdot t_2 \cdot t_3 \cdot \boldsymbol{A}_0$ with large coefficients modulo q is canceled when doing the subtraction.

We have that $\boldsymbol{\omega}$ is a vector of dimension m. Now an important step is to restrict ourselves to the first component of $\boldsymbol{\omega}$. Namely in order to apply the same technique as in the Cheon *et al.* attack, we would like to express $\boldsymbol{\omega}$ as the product of two vectors, where the left vector corresponds to User 1 and the right vector corresponds to User 2. However due to the "round-robin" fashion of exponent encodings, for this we would need to swap the product $\hat{\boldsymbol{E}}_{3,2} \cdot \boldsymbol{C}_{3,3}$ appearing in (3), since $\hat{\boldsymbol{E}}_{3,2}$ corresponds to User 2 while $\boldsymbol{C}_{3,3}$ corresponds to User 1; this cannot be done if we consider the full vector $\boldsymbol{\omega}$. By restricting ourselves to the first component of $\boldsymbol{\omega}$, the product $\hat{\boldsymbol{E}}_{3,2} \cdot \boldsymbol{C}_{3,3}$ becomes a simple scalar product that can be swapped; namely the scalar product of $\hat{\boldsymbol{E}}_{3,2}$ by the first column vector $\boldsymbol{C}'_{3,3}$ of the matrix $\boldsymbol{C}_{3,3}$. We obtain the scalar:

$$\omega = t_1 \cdot t_3 \cdot E_{2,3} + \hat{\boldsymbol{E}}_{2,2} \cdot \boldsymbol{C}'_{2,3} - t_2 \cdot t_3 \cdot E_{3,3} - \boldsymbol{C}'_{3,3} \cdot \hat{\boldsymbol{E}}_{3,2}$$

where $\boldsymbol{C}'_{2,3}$ and $\boldsymbol{C}'_{3,3}$ are the first column vectors of $\boldsymbol{C}_{2,3}$ and $\boldsymbol{C}_{3,3}$ respectively, both of dimension m; similarly $E_{2,3}$ and $E_{3,3}$ are the first components of $\boldsymbol{E}_{2,3}$ and $\boldsymbol{E}_{3,3}$ respectively.

We can now write ω as the scalar product of 2 vectors, the left one corresponding only to User 1, and the right one corresponding only to User 2:

$$\omega = \begin{bmatrix} t_1 & \hat{\boldsymbol{E}}_{2,2} & E_{3,3} & \boldsymbol{C}'_{3,3} \end{bmatrix} \cdot \begin{bmatrix} t_3 \cdot E_{2,3} \\ \boldsymbol{C}'_{2,3} \\ -t_2 \cdot t_3 \\ -\hat{\boldsymbol{E}}_{3,2} \end{bmatrix} .$$

Note that the two vectors in the product have dimension $2m+2$.

As in the Cheon *et al.* attack [CHL$^+$15], we can now extend ω to a matrix by considering many left row vectors and many right column vectors. However instead of a square matrix as in the Cheon *et al.* attack, we consider a rectangular matrix with $2m+3$ rows and $2m+2$ columns. In Equation (2), this is done by considering $2m+3$ public encodings $\hat{\boldsymbol{C}}_{2,2,i}$ and $\boldsymbol{C}_{3,3,i}$ corresponding to User 1, and similarly $2m+2$ encodings $\boldsymbol{C}_{2,3,j}$ and $\hat{\boldsymbol{C}}_{3,2,j}$ corresponding to User 2, for $1 \le i \le 2m+3$ and $1 \le j \le 2m+2$. More precisely we compute as previously over R the following matrix elements, restricting ourselves to the first component:

$$(\boldsymbol{W})_{ij} = \boldsymbol{A}_{2,1} \cdot \hat{\boldsymbol{C}}_{2,2,i} \cdot \boldsymbol{C}'_{2,3,j} - \boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \tag{4}$$

and as previously we can write:

$$(\boldsymbol{W})_{ij} = \begin{bmatrix} t_{1,i} & \hat{\boldsymbol{E}}_{2,2,i} & E_{3,3,i} & \boldsymbol{C}'_{3,3,i} \end{bmatrix} \cdot \begin{bmatrix} t_3 \cdot E_{2,3,j} \\ \boldsymbol{C}'_{2,3,j} \\ -t_{2,j} \cdot t_3 \\ -\hat{\boldsymbol{E}}_{3,2,j} \end{bmatrix} .$$

We obtain a $(2m+3) \times (2m+2)$ matrix $\boldsymbol{W}$ with:

$$\boldsymbol{W} = \underbrace{\begin{bmatrix} & \cdots & \\ t_{1,i} \;\; \hat{\boldsymbol{E}}_{2,2,i} & E_{3,3,i} & \boldsymbol{C}'_{3,3,i} \\ & \cdots & \end{bmatrix}}_{\boldsymbol{A}} \cdot \underbrace{\begin{bmatrix} & t_3 \cdot E_{2,3,j} & \\ \vdots & \boldsymbol{C}'_{2,3,j} & \vdots \\ & -t_{2,j} \cdot t_3 & \\ & -\hat{\boldsymbol{E}}_{3,2,j} & \end{bmatrix}}_{\boldsymbol{B}}$$

where the matrix $\boldsymbol{A}$ has $2m+3$ rows vectors, each of dimension $2m+2$, and the matrix $\boldsymbol{B}$ has $2m+2$ column vectors, each of dimension $2m+2$; hence $\boldsymbol{B}$ is a square matrix.

By doing linear algebra, we can find a vector $\boldsymbol{u}$ over R of dimension $2m+3$ such that $\boldsymbol{u} \cdot \boldsymbol{W} = 0$, which gives:

$$(\boldsymbol{u} \cdot \boldsymbol{A}) \cdot \boldsymbol{B} = 0 .$$

Heuristically with good probability the matrix $\boldsymbol{B}$ is invertible, which implies:

$$\boldsymbol{u} \cdot \boldsymbol{A} = 0 .$$

Since the first column of the matrix $\boldsymbol{A}$ is the column vector given by the $t_{1,i}$'s, such vector $\boldsymbol{u}$ gives a linear relation among the secret exponents $t_{1,i}$.

Moreover, since the encodings $\boldsymbol{D}_{2,2}$ and $\boldsymbol{D}_{3,3}$ corresponding to s_1 are public, we can express s_1 as a linear combination of the $t_{1,i}$'s, over R. Namely we can define as previously the product encoding $\hat{\boldsymbol{D}}_{2,2} := \boldsymbol{C}_{2,1} \cdot \boldsymbol{D}_{2,2}$, with:

$$\boldsymbol{A}_{2,1} \cdot \hat{\boldsymbol{D}}_{2,2} = s_1 \cdot t_3 \cdot \boldsymbol{A}_{2,3} + \hat{\boldsymbol{F}}_{2,2} \pmod q$$

for some small error vector $\hat{\boldsymbol{F}}_{2,2}$, and we can now compute the same $(\boldsymbol{W})_{ij}$ as in (4) but with $\hat{\boldsymbol{D}}_{2,2}$ and $\boldsymbol{D}'_{3,3}$ instead of $\hat{\boldsymbol{C}}_{2,2,i}$ and $\boldsymbol{C}'_{3,3,i}$, where $\boldsymbol{D}'_{3,3}$ is the first column of $\boldsymbol{D}_{3,3}$. More precisely, we compute for all $1 \le j \le 2m+2$:

$$\omega_j = \boldsymbol{A}_{2,1} \cdot \hat{\boldsymbol{D}}_{2,2} \cdot \boldsymbol{C}'_{2,3,j} - \boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,j} \cdot \boldsymbol{D}'_{3,3}$$

which gives as previously:

$$\omega_j = \begin{bmatrix} s_1 & \hat{\boldsymbol{F}}_{2,2} & F_{3,3} & \boldsymbol{D}'_{3,3} \end{bmatrix} \cdot \begin{bmatrix} t_3 \cdot E_{2,3,j} \\ \boldsymbol{C}'_{2,3,j} \\ -t_{2,j} \cdot t_3 \\ -\hat{\boldsymbol{E}}_{3,2,j} \end{bmatrix} .$$

This implies that we can replace any row vector $[t_{1,i}\ \hat{\boldsymbol{E}}_{2,2,i}\ E_{3,3,i}\ \boldsymbol{C}'_{3,3,i}]$ in the matrix $\boldsymbol{A}$ by the row vector:

$$[s_1\ \ \hat{\boldsymbol{F}}_{2,2}\ \ F_{3,3}\ \ \boldsymbol{D}'_{3,3}] \tag{5}$$

where $\boldsymbol{D}'_{3,3}$ is the first column of $\boldsymbol{D}_{3,3}$, and $F_{3,3}$ is the first component of $\boldsymbol{F}_{3,3}$. Using the previous technique, we can therefore obtain a linear relation between s_1 and the $t_{1,i}$'s over R. More precisely, with overwhelming probability, such a relation can be put in the form:

$$\mu \cdot s_1 = \sum_{i=1}^{2m+2} \lambda_i \cdot t_{1,i} \tag{6}$$

with $\mu \in \mathbb{Z}$ and $\lambda_1, \ldots, \lambda_{2m+2} \in R$. Indeed, we obtain such a relation by computing the kernel of the matrix analogous to $\boldsymbol{W}$ above in echelon form over the fraction field of R, which gives the kernel of the corresponding matrix $\boldsymbol{A}$ (assuming that $\boldsymbol{B}$ is invertible). Unless a minor of that matrix vanishes, which happens with only negligible probability, this gives a relation where the coefficient of s_1 is 1 and the other coefficients are in the fraction field $R \otimes_{\mathbb{Z}} \mathbb{Q}$ of R. By clearing denominators, we get an expression of the form (6).

Then, by considering exactly one additional $t_{1,i}$ (say $t_{1,2m+3}$) and carrying out the same computations with indices $i = 2, \ldots, 2m+3$ instead of $i = 1, \ldots, 2m+2$, we get a second relation:

$$\nu \cdot s_1 = \sum_{i=2}^{2m+3} \lambda'_i \cdot t_{1,i} .$$

If the integers μ and ν are relatively prime, which happens with significant probability[3], we can apply Bézout's identity to obtain a linear relation in R where the coefficient of s_1 is 1:

$$s_1 = \sum_{i=1}^{2m+3} \alpha_i \cdot t_{1,i} . \tag{7}$$

[3] Heuristically, it is the probability that two random elements of R have coprime norms, since the rational integer denominator of an element of the fraction field has the same prime factors as its norm. For $R = \mathbb{Z}[x]/(x^{2^n}+1)$, that probability is close to 3/4: see Appendix C.

Note that we have the same linear relations for the other components of the vector (5) corresponding to s_1, namely:

$$\hat{\boldsymbol{F}}_{2,2} = \sum_{i=1}^{2m+3} \alpha_i \cdot \hat{\boldsymbol{E}}_{2,2,i}, \quad F_{3,3} = \sum_{i=1}^{2m+3} \alpha_i \cdot E_{3,3,i}, \quad \boldsymbol{D}'_{3,3} = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{C}'_{3,3,i}\,. \tag{8}$$

Second step: equivalent private-key. In this second step, we show how to publicly compute an encoding equivalent to $\boldsymbol{D}_{1,1}$, which is private to User 1; this will break the key-agreement protocol. In the first step, we had considered rows 2 and 3 to derive the linear relations (7) and (8); we now consider Row 1. On Row 1, the encodings $\boldsymbol{D}_{1,2}$ and $\boldsymbol{D}_{1,3}$ are public, so we can define as previously the product encoding $\hat{\boldsymbol{D}}_{1,3} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}_{1,3}$, which gives:

$$\boldsymbol{A}_{1,2} \cdot \hat{\boldsymbol{D}}_{1,3} = s_2 \cdot s_3 \cdot \boldsymbol{A}_0 + \hat{\boldsymbol{F}}_{1,3} \pmod q$$

for some small error vector $\hat{\boldsymbol{F}}_{1,3}$. Recall that the encoding $\boldsymbol{D}_{1,1}$ is private to User 1, with:

$$\boldsymbol{A}_{1,1} \cdot \boldsymbol{D}_{1,1} = s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{F}_{1,1} \pmod q\,. \tag{9}$$

Therefore only User 1 can privately compute:

$$\boldsymbol{A}_{1,1} \cdot \boldsymbol{D}_{1,1} \cdot \hat{\boldsymbol{D}}_{1,3} = s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0 + s_1 \cdot \hat{\boldsymbol{F}}_{1,3} + \boldsymbol{F}_{1,1} \cdot \hat{\boldsymbol{D}}_{1,3} \pmod q \tag{10}$$

and extract the high order bits of $s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0 \bmod q$ to generate the session key.

We cannot compute the previous equation since $\boldsymbol{D}_{1,1}$ is private. However since we know a linear relation (7) between s_1 and the $t_{1,i}$'s, and the encodings $\boldsymbol{C}_{1,1,i}$ corresponding to $t_{1,i}$ are public, with:

$$\boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,i} = t_{1,i} \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}_{1,1,i} \pmod q$$

it is then natural to compute:

$$\tilde{\boldsymbol{D}}_{1,1} = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{C}_{1,1,i}\,,$$

which gives:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} = s_1 \cdot \boldsymbol{A}_{1,2} + \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \pmod q\,. \tag{11}$$

The difference with (9) is that the error term $\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i}$ is not necessarily small since the coefficients α_i can be large. Therefore if we compute:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} \cdot \hat{\boldsymbol{D}}_{1,3} = s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0 + s_1 \cdot \hat{\boldsymbol{F}}_{1,3} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{D}}_{1,3} \pmod q \tag{12}$$

then as opposed to (10) this does not reveal the high-order bits of $s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0 \bmod q$. In the following, we show how to derive an approximation of $\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i}$ over R, in order to correct the error in (11) and break the protocol. This is the second part of our attack.

As in the first step of the attack, to get equations over R and not only modulo q, we consider the difference between two rows, this time the difference between rows 1 and 3 (instead of rows 2 and 3). We have the public encodings:

$$\begin{aligned}
\boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}_{1,1,\ell} \pmod q \\
\boldsymbol{A}_{1,2} \cdot \hat{\boldsymbol{C}}_{1,3,\ell} &= t_{2,\ell} \cdot t_3 \cdot \boldsymbol{A}_0 + \hat{\boldsymbol{E}}_{1,3,\ell} \pmod q \\
\boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,\ell} &= t_{2,\ell} \cdot t_3 \cdot \boldsymbol{A}_{3,3} + \hat{\boldsymbol{E}}_{3,2,\ell} \pmod q \\
\boldsymbol{A}_{3,3} \cdot \boldsymbol{C}_{3,3,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_0 + \boldsymbol{E}_{3,3,\ell} \pmod q
\end{aligned}$$

where we let $\hat{\boldsymbol{C}}_{1,3,\ell} := \boldsymbol{C}_{1,2,\ell} \cdot \boldsymbol{C}_{1,3}$, for some small error vector $\hat{\boldsymbol{E}}_{1,3,\ell}$. As previously we can compute over R, restricting ourselves to the first component, where $\hat{\boldsymbol{C}}'_{1,3,j}$ and $\boldsymbol{C}'_{3,3,i}$ are the first columns of $\hat{\boldsymbol{C}}_{1,3,j}$ and $\boldsymbol{C}_{3,3,i}$ respectively:

$$\begin{aligned}
\omega_{ij} &= \boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,3,j} - \boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \\
&= t_{1,i} \cdot \hat{E}_{1,3,j} + \boldsymbol{E}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,3,j} - t_{2,j} \cdot t_3 \cdot E_{3,3,i} - \hat{\boldsymbol{E}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \,.
\end{aligned}$$

We can therefore compute over R, using the coefficients α_i from the linear relation (7):

$$\begin{aligned}
\mathit{\Omega}_j &= \sum_{i=1}^{2m+3} \alpha_i \cdot \Big(\boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,3,j} - \boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \Big) \qquad (13) \\
&= \sum_{i=1}^{2m+3} \alpha_i \cdot \Big(t_{1,i} \cdot \hat{E}_{1,3,j} + \boldsymbol{E}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,3,j} - t_{2,j} \cdot t_3 \cdot E_{3,3,i} - \hat{\boldsymbol{E}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \Big) \,.
\end{aligned}$$

Using the linear relations (7) and (8), we obtain:

$$\mathit{\Omega}_j = s_1 \cdot \hat{E}_{1,3,j} - t_{2,j} \cdot t_3 \cdot F_{3,3} - \hat{\boldsymbol{E}}_{3,2,j} \cdot \boldsymbol{D}'_{3,3} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}'_{1,3,j}$$

which gives:

$$\mathit{\Omega}_j = u_j + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}'_{1,3,j} \qquad (14)$$

for some small u_j in R. In summary we obtain a large scalar $\mathit{\Omega}_j$ because the coefficients α_i in (13) are large, but eventually what makes $\mathit{\Omega}_j$ large is only the contribution from $(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i}) \cdot \hat{\boldsymbol{C}}'_{1,3,j}$; namely because of the linear relations (7) and (8) the other terms remain small.

We can now write (14) in vectorial form, where we let $\hat{\boldsymbol{C}}''_{1,3}$ be the square matrix whose columns are the column vectors $\hat{\boldsymbol{C}}'_{1,3,j}$ for $1 \le j \le m$; recall that the $\hat{\boldsymbol{C}}'_{1,3,j}$ are the first column vectors of the matrix encodings $\hat{\boldsymbol{C}}_{1,3,j}$. We obtain a row vector $\boldsymbol{\mathit{\Omega}}$ of dimension m, where:

$$\boldsymbol{\mathit{\Omega}} = \boldsymbol{u} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}''_{1,3} \qquad (15)$$

where $\hat{\boldsymbol{C}}''_{1,3}$ is a public square matrix of dimension m.

Now the crucial observation is that because the vector $\boldsymbol{u}$ has small components, we can get an approximation of the vector $\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i}$ by reducing the vector $\boldsymbol{\Omega}$ modulo the matrix $\hat{\boldsymbol{C}}''_{1,3}$, assuming that $\hat{\boldsymbol{C}}''_{1,3}$ is an invertible matrix, which heuristically holds with good probability. This can be done by solving over the fraction field of R the linear system $\boldsymbol{\Omega} = \boldsymbol{y} \cdot \hat{\boldsymbol{C}}''_{1,3}$ and then rounding to R the coefficients of $\boldsymbol{y}$. Heuristically the vector $\boldsymbol{E} = \lfloor \boldsymbol{y} \rceil$ should be a good approximation of $\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i}$; namely letting:

$$\boldsymbol{E}' = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} - \boldsymbol{E} \tag{16}$$

we get using $\boldsymbol{y} = \boldsymbol{\Omega} \cdot \hat{\boldsymbol{C}}''^{-1}_{1,3}$:

$$\begin{aligned} \boldsymbol{E}' &= (\boldsymbol{\Omega} - \boldsymbol{u}) \cdot \hat{\boldsymbol{C}}''^{-1}_{1,3} - \boldsymbol{E} \\ &= \boldsymbol{y} - \boldsymbol{E} - \boldsymbol{u} \cdot \hat{\boldsymbol{C}}''^{-1}_{1,3} \end{aligned}$$

and therefore since $\boldsymbol{y} - \boldsymbol{E}$ and $\boldsymbol{u}$ are small, the difference vector $\boldsymbol{E'}$ should be small if the norm of the transpose of the matrix $\hat{\boldsymbol{C}}''^{-1}_{1,3}$ remains small. We know that such a bound holds with probability close to 1 if we model $\hat{\boldsymbol{C}}''_{1,3}$ as a random matrix (*e.g.* Rudelson [Rud08] provides a bound of the form $O(m^{3/2})$), and so we expect $\boldsymbol{E'}$ to be small for randomly generated encodings and sufficiently large m.

Combining (11) and (16), we get:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} - \boldsymbol{E} = s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}' \pmod q$$

for a small vector $\boldsymbol{E}'$. Note that the previous equation is very similar to the original equation for the private encoding $\boldsymbol{D}_{1,1}$:

$$\boldsymbol{A}_{1,1} \cdot \boldsymbol{D}_{1,1} = s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{F}_{1,1} \pmod q$$

the only difference being the publicly computed correction vector $\boldsymbol{E}$. Therefore the pair $(\tilde{\boldsymbol{D}}_{1,1}, \boldsymbol{E})$ gives us an equivalent of the private encoding $\boldsymbol{D}_{1,1}$, which breaks the protocol. More precisely we can eventually compute from public parameters:

$$\begin{aligned} \Big(\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} - \boldsymbol{E}\Big) \cdot \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}_{1,3} &= \big(s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}'\big) \cdot \hat{\boldsymbol{D}}_{1,3} \pmod q \\ &= s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0 + s_1 \cdot \hat{\boldsymbol{F}}_{1,3} + \boldsymbol{E}' \cdot \hat{\boldsymbol{D}}_{1,3} \pmod q \end{aligned}$$

Since all the error terms are small, this enables to extract the high-order bits of $s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0 \bmod q$, and breaks the protocol.

3.2 Extension to $k \geq 3$ Users

The extension of our attack to $k \geq 3$ users is relatively straightforward and described in Appendix D.

4 Cryptanalysis of GGH15 With Safeguards

In [GGH15, Section 5.1] two safeguards for multipartite key agreement based on GGH15 multilinear maps are described:

1. Kilian-style randomization of the encodings, where $\boldsymbol{C}$ is replaced by $\bar{\boldsymbol{C}} := \boldsymbol{R}^{-1} \cdot \boldsymbol{C} \cdot \boldsymbol{R}'$ using the randomizer matrices $\boldsymbol{R}$, $\boldsymbol{R}'$ belonging to two adjacent nodes.
2. Choosing the first encoding matrix in each chain to have large entries.

In the following, we show how to extend our previous attack when those two safeguards are used.

4.1 First Safeguard: Kilian-Style Randomization of the Encodings.

The following safeguard for GGH15 multilinear maps is described in [GGH15], using Kilian-type randomization [Kil88]. For each internal node v in the graph one can choose a random invertible $m \times m$ matrix $\boldsymbol{R}_v$ modulo q, and for the sinks and sources we set $\boldsymbol{R}_v = \boldsymbol{I}$. Then each encoding $\boldsymbol{C}$ relative to path $u \rightsquigarrow v$ is replaced by a masked encoding $\bar{\boldsymbol{C}} := \boldsymbol{R}_u^{-1} \cdot \boldsymbol{C} \cdot \boldsymbol{R}_v$. Concretely, in the GGH15 key-agreement protocol, instead of publishing encodings $\boldsymbol{C}_{i,j}$ with:

$$\boldsymbol{A}_{i,j} \cdot \boldsymbol{C}_{i,j,\ell} = t_{1+(j-i \bmod k),\ell} \cdot \boldsymbol{A}_{i,j+1} + \boldsymbol{E}_{i,j,\ell} \pmod q$$

one would only publish the masked encodings modulo q:

$$\bar{\boldsymbol{C}}_{i,j,\ell} := \boldsymbol{R}_{i,j}^{-1} \cdot \boldsymbol{C}_{i,j,\ell} \cdot \boldsymbol{R}_{i,j+1} \tag{17}$$

with $\boldsymbol{R}_{i,1} = \boldsymbol{R}_{i,k+1} = \boldsymbol{I}$ for all i; the same masking is applied to the encodings $\boldsymbol{D}_{i,j}$. Since the product of encoding on any source-to-sink path remains the same, the same value is eventually extracted. Namely for all i we have:

$$\prod_{j=1}^{k} \bar{\boldsymbol{C}}_{i,j} = \prod_{j=1}^{k} \boldsymbol{C}_{i,j}$$

and therefore exactly the same session-key as before is computed by all users.

4.2 Second Safeguard: First Encodings With Large Entries

The second safeguard described in [GGH15, Section 5.1] consists in choosing the first encodings $\boldsymbol{C}_{i,1}$ in each chain to have large entries modulo q, instead of small entries. Namely the first encoding $\boldsymbol{C}_{i,1}$ does not contribute in the error term when computing the session-key, so it can have large entries.

4.3 Cryptanalysis of GGH15 With Both Safeguards

In this section we show how to extend our attack from Section 3 when both safeguards are used. Note the first step of our attack still applies, since in the first step we are only using product of encodings from source to sink. Namely in Equation (4) exactly the same value $(\boldsymbol{W})_{ij}$ is obtained when using masked encodings. Therefore we can still derive the same linear relation between secret exponents as in (7) and (8).

However the second step of our attack does not apply directly, since our second step requires the knowledge of the matrix $\hat{\boldsymbol{C}}''_{1,3}$ in (15), which is obtained from the first columns of the encodings $\hat{\boldsymbol{C}}_{1,3,j} = \boldsymbol{C}_{1,2,j} \cdot \boldsymbol{C}_{1,3}$. Since these are partial products only, such partial products would be masked by the unknown randomization matrix $\boldsymbol{R}_{1,2}^{-1}$ modulo q, hence the matrix $\hat{\boldsymbol{C}}''_{1,3}$ is unknown.

We can however adapt our second step as follows. For simplicity we keep the same notations as previously, that is we describe our extended attack in term of the original encodings $\boldsymbol{C}_{i,j,\ell}$, instead of the masked encodings $\bar{\boldsymbol{C}}_{i,j,\ell}$ from (17); in that case we are only allowed to use products of encodings from source to sink. We first start with a slightly different equation from (15):

$$\boldsymbol{\Omega} = \boldsymbol{u} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{G}}''_{1,3} \tag{18}$$

where $\hat{\boldsymbol{G}}''_{1,3}$ is a matrix whose columns are the first column vectors of $\boldsymbol{D}_{1,2} \cdot \boldsymbol{C}_{1,3,j}$ for $1 \leq j \leq 2m+2$. Note that in (12) the error term that we must estimate to recover the session key is:

$$\boldsymbol{E} = \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{D}}_{1,3} \tag{19}$$

Using a similar approach as in the attack first step, our approach consists in finding a vector $\boldsymbol{x}$ with coefficients in the fraction field $R \otimes_{\mathbb{Z}} \mathbb{Q}$ of R such that:

$$\hat{\boldsymbol{D}}'_{1,3} = \hat{\boldsymbol{G}}''_{1,3} \cdot \boldsymbol{x}$$

where $\hat{\boldsymbol{D}}'_{1,3}$ is the first column vector of $\hat{\boldsymbol{D}}_{1,3}$. Applying the vector $\boldsymbol{x}$ on (18) and rounding in R, we obtain:

$$\lfloor \boldsymbol{\Omega} \cdot \boldsymbol{x} \rceil = \lfloor \boldsymbol{u} \cdot \boldsymbol{x} \rceil + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{D}}'_{1,3}$$

Since the components of $\boldsymbol{u}$ (over R) are small, and moreover the coefficients of $\boldsymbol{x}$ (over $R \otimes_{\mathbb{Z}} \mathbb{Q}$) are heuristically also small, the scalar $\lfloor \boldsymbol{u} \cdot \boldsymbol{x} \rceil$ in R is small compared to q, and therefore we obtain a good estimate of the first component of the error vector $\boldsymbol{E}$ from (19), which enables to recover the first component of the session key and breaks the scheme.

4.4 Detailed Description

First step: linear relations in $\boldsymbol{R}$. The first step of our attack is exactly the same as previously. Namely as mentioned previously the first step of our previous attack still applies, since in the first step we are only using product of encodings from source to sink. More precisely in Equation (4) exactly the same value $(\boldsymbol{W})_{ij}$ is obtained when using masked encodings, and therefore we can still derive the same linear relations as in (7) and (8):

$$s_1 = \sum_{i=1}^{2m+3} \alpha_i \cdot t_{1,i}, \quad \hat{\boldsymbol{F}}_{2,2} = \sum_{i=1}^{2m+3} \alpha_i \cdot \hat{\boldsymbol{E}}_{2,2,i}, \quad F_{3,3} = \sum_{i=1}^{2m+3} \alpha_i \cdot E_{3,3,i}, \quad \boldsymbol{D}'_{3,3} = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{C}'_{3,3,i} \tag{20}$$

Note that as opposed to Section 3 we don't know the value of the encodings $\boldsymbol{D}'_{3,3}$ and $\boldsymbol{C}'_{3,3,i}$, since they are masked by the $\boldsymbol{R}_{ij}$ matrices; we only recover the coefficients α_i in R.

Second step: another linear relation. In the second step, our goal is to find a vector $\boldsymbol{x}$ with coefficients in the fraction field $R \otimes_{\mathbb{Z}} \mathbb{Q}$ of R such that:

$$\boldsymbol{D}'_{1,3} = \sum_{i=1}^{2m+2} x_i \cdot \boldsymbol{C}'_{1,3,i}$$

where $\boldsymbol{D}'_{1,3}$ and $\boldsymbol{C}'_{1,3,i}$ are the first column vectors of $\boldsymbol{D}_{1,3}$ and $\boldsymbol{C}_{1,3,i}$ respectively. We show that this can be done using the same approach as in the attack first step.

Namely letting $\hat{\boldsymbol{C}}_{1,2,\ell} := \boldsymbol{C}_{1,1,\ell} \cdot \boldsymbol{C}_{1,2}$ where we let $\boldsymbol{C}_{1,2} := \boldsymbol{C}_{1,2,1}$ corresponding to $t_2 := t_{2,1}$, we obtain:

$$\begin{aligned}\boldsymbol{A}_{1,1} \cdot \hat{\boldsymbol{C}}_{1,2,\ell} &= t_{1,\ell} \cdot t_2 \cdot \boldsymbol{A}_{1,3} + \hat{\boldsymbol{E}}_{1,2,\ell} \pmod q\\ \boldsymbol{A}_{1,3} \cdot \boldsymbol{C}_{1,3,\ell} &= t_{3,\ell} \cdot \boldsymbol{A}_0 + \boldsymbol{E}_{1,3,\ell} \pmod q\end{aligned}$$

Similarly letting $\hat{\boldsymbol{C}}_{2,3,\ell} := \boldsymbol{C}_{2,2,\ell} \cdot \boldsymbol{C}_{2,3}$ where $\boldsymbol{C}_{2,3} := \boldsymbol{C}_{2,3,1}$, we get:

$$\begin{aligned}\boldsymbol{A}_{2,1} \cdot \boldsymbol{C}_{2,1,\ell} &= t_{3,\ell} \cdot \boldsymbol{A}_{2,2} + \boldsymbol{E}_{2,1,\ell} \pmod q\\ \boldsymbol{A}_{2,2} \cdot \hat{\boldsymbol{C}}_{2,3,\ell} &= t_1 \cdot t_{2,\ell} \cdot \boldsymbol{A}_0 + \hat{\boldsymbol{E}}_{2,3,\ell} \pmod q\end{aligned}$$

We can therefore compute the following matrix elements in R, restricting ourselves as previously to the first component of the vectors:

$$\begin{aligned}(\boldsymbol{W})_{i,j} &= \boldsymbol{A}_{1,1} \cdot \hat{\boldsymbol{C}}_{1,2,i} \cdot \boldsymbol{C}'_{1,3,j} - \boldsymbol{A}_{2,1} \cdot \boldsymbol{C}_{2,1,j} \cdot \hat{\boldsymbol{C}}'_{2,3,i}\\ &= t_{1,i} \cdot t_2 \cdot E_{1,3,j} + \hat{\boldsymbol{E}}_{1,2,i} \cdot \boldsymbol{C}'_{1,3,j} - t_{3,j} \cdot \hat{E}_{2,3,i} - \boldsymbol{E}_{2,1,j} \cdot \hat{\boldsymbol{C}}'_{2,3,i}\end{aligned}$$

for all $1 \leq i \leq 2m+2$ and $1 \leq j \leq 2m+2$, where $\boldsymbol{C}'_{1,3,j}$ and $\hat{\boldsymbol{C}}'_{2,3,i}$ are the first column vectors of $\boldsymbol{C}_{1,3,j}$ and $\hat{\boldsymbol{C}}_{2,3,i}$ respectively. This gives:

$$(\boldsymbol{W})_{ij} = \begin{bmatrix} t_{1,i}\, t_2 & \hat{\boldsymbol{E}}_{1,2,i} & \hat{E}_{2,3,i} & \hat{\boldsymbol{C}}'_{2,3,i} \end{bmatrix} \cdot \begin{bmatrix} E_{1,3,j} \\ \boldsymbol{C}'_{1,3,j} \\ -t_{3,j} \\ -\boldsymbol{E}_{2,1,j} \end{bmatrix}.$$

Moreover, since the encodings $\boldsymbol{D}_{1,3}$ and $\boldsymbol{D}_{2,1}$ corresponding to s_3 on rows 1 and 2 are public, we can additionally compute the corresponding vector:

$$\begin{aligned}(\boldsymbol{V})_i &= \boldsymbol{A}_{1,1} \cdot \hat{\boldsymbol{C}}_{1,2,i} \cdot \boldsymbol{D}'_{1,3} - \boldsymbol{A}_{2,1} \cdot \boldsymbol{D}_{2,1} \cdot \hat{\boldsymbol{C}}'_{2,3,i}\\ &= \begin{bmatrix} t_{1,i}\, t_2 & \hat{\boldsymbol{E}}_{1,2,i} & \hat{E}_{2,3,i} & \hat{\boldsymbol{C}}'_{2,3,i} \end{bmatrix} \cdot \begin{bmatrix} F_{1,3} \\ \boldsymbol{D}'_{1,3} \\ -s_3 \\ -\boldsymbol{F}_{2,1} \end{bmatrix}.\end{aligned}$$

where $\boldsymbol{D}'_{1,3}$ is the first column vector of $\boldsymbol{D}_{1,3}$. Therefore assuming that the matrix $\boldsymbol{W}$ is invertible, we can find $\boldsymbol{x}$ in $R \otimes_{\mathbb{Z}} \mathbb{Q}$ such that:

$$\boldsymbol{W} \cdot \boldsymbol{x} = \boldsymbol{V}$$

which gives as required:

$$\boldsymbol{D}'_{1,3} = \sum_{i=1}^{2m+2} x_i \cdot \boldsymbol{C}'_{1,3,i} \tag{21}$$

Note that the only difference with the linear relations from Step 1 is that we don't require the x_i's to be in R, only in the fraction field $R \otimes_{\mathbb{Z}} \mathbb{Q}$ of R; this implies that heuristically such coefficients should remain small in absolute value.

Third step: estimating the error term. In the third step our goal is to estimate the error term when computing the session-key, as in the second step of the basic attack. We first start with a slightly different equation from (15):

$$\boldsymbol{\Omega} = \boldsymbol{u} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{G}}''_{1,3} \tag{22}$$

where $\hat{\boldsymbol{G}}''_{1,3}$ is a matrix whose columns are the first column vectors of $\boldsymbol{D}_{1,2} \cdot \boldsymbol{C}_{1,3,j}$ for $1 \leq j \leq 2m+2$. Therefore the only difference with (15) is that we use the matrix $\hat{\boldsymbol{G}}''_{1,3}$ instead of $\hat{\boldsymbol{C}}''_{1,3}$.

To obtain (22) we proceed as follows. Instead of letting $\hat{\boldsymbol{C}}_{1,3,\ell} = \boldsymbol{C}_{1,2,\ell} \cdot \boldsymbol{C}_{1,3}$ as in the basic attack, we let $\hat{\boldsymbol{C}}_{1,3,\ell} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{C}_{1,3,\ell}$. Similarly we let $\hat{\boldsymbol{C}}_{3,2,\ell} := \boldsymbol{D}_{3,1} \cdot \boldsymbol{C}_{3,2,\ell}$. This is possible because on rows 1 and 3 the encodings $\boldsymbol{D}_{1,2}$ and $\boldsymbol{D}_{3,1}$ corresponding to s_2 are public. We obtain:

$$\begin{aligned}
\boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}_{1,1,\ell} \pmod q \\
\boldsymbol{A}_{1,2} \cdot \hat{\boldsymbol{C}}_{1,3,\ell} &= s_2 \cdot t_{3,\ell} \cdot \boldsymbol{A}_0 + \hat{\boldsymbol{E}}_{1,3,\ell} \pmod q \\
\boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,\ell} &= s_2 \cdot t_{3,\ell} \cdot \boldsymbol{A}_{3,3} + \hat{\boldsymbol{E}}_{3,2,\ell} \pmod q \\
\boldsymbol{A}_{3,3} \cdot \boldsymbol{C}_{3,3,\ell} &= t_{1,\ell} \cdot \boldsymbol{A}_0 + \boldsymbol{E}_{3,3,\ell} \pmod q
\end{aligned}$$

As previously we can compute over R, restricting ourselves to the first component, where $\hat{\boldsymbol{C}}'_{1,3,j}$ and $\boldsymbol{C}'_{3,3,i}$ are the first columns of $\hat{\boldsymbol{C}}_{1,3,j}$ and $\boldsymbol{C}_{3,3,i}$ respectively:

$$\begin{aligned}
\omega_{ij} &= \boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,3,j} - \boldsymbol{A}_{3,1} \cdot \hat{\boldsymbol{C}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \\
&= t_{1,i} \cdot \hat{E}_{1,3,j} + \boldsymbol{E}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,3,j} - s_2 \cdot t_{3,j} \cdot E_{3,3,i} - \hat{\boldsymbol{E}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \,.
\end{aligned}$$

We can therefore compute over R, using the coefficients α_i from the linear relations (20):

$$\Omega_j = \sum_{i=1}^{2m+3} \alpha_i \cdot \omega_{ij} = \sum_{i=1}^{2m+3} \alpha_i \cdot \left(t_{1,i} \cdot \hat{E}_{1,3,j} + \boldsymbol{E}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,3,j} - s_2 \cdot t_{3,j} \cdot E_{3,3,i} - \hat{\boldsymbol{E}}_{3,2,j} \cdot \boldsymbol{C}'_{3,3,i} \right)$$

Using the linear relations in (20), we obtain:

$$\Omega_j = s_1 \cdot \hat{E}_{1,3,j} - s_2 \cdot t_{3,j} \cdot F_{3,3} - \hat{\boldsymbol{E}}_{3,2,j} \cdot \boldsymbol{D}'_{3,3} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}'_{1,3,j}$$

where $\boldsymbol{D}'_{3,3}$ is the first column vector of $\boldsymbol{D}_{3,3}$. This gives:

$$\Omega_j = u_j + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}'_{1,3,j}$$

for some small u_j in R. Since we have let $\hat{\boldsymbol{C}}_{1,3,j} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{C}_{1,3,j}$ for $1 \le j \le 2m+2$, in vectorial form we obtain (22) as required, where $\hat{\boldsymbol{G}}''_{1,3}$ is the matrix whose columns are the first column vectors of $\boldsymbol{D}_{1,2} \cdot \boldsymbol{C}_{1,3,j}$ for $1 \le j \le 2m+2$.

Recall that in (12) the error term that we must estimate to recover the session key is:

$$\boldsymbol{E} = \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{D}}_{1,3} \tag{23}$$

where $\hat{\boldsymbol{D}}_{1,3} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}_{1,3}$. In the following we will only estimate the first component, so we let $\hat{\boldsymbol{D}}'_{1,3} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}'_{1,3}$, where $\hat{\boldsymbol{D}}'_{1,3}$ and $\boldsymbol{D}'_{1,3}$ are the first column vectors of $\hat{\boldsymbol{D}}_{1,3}$ and $\boldsymbol{D}_{1,3}$ respectively.

We now use the vector $\boldsymbol{x}$ computed in the second step. In matrix notation, Equation (21) gives:

$$\boldsymbol{D}'_{1,3} = \boldsymbol{C}''_{1,3} \cdot \boldsymbol{x}$$

where $\boldsymbol{C}''_{1,3}$ is the matrix whose columns are the first column vectors of $\boldsymbol{C}_{1,3,i}$ for $1 \le i \le 2m+2$. Using $\hat{\boldsymbol{G}}''_{1,3} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{C}''_{1,3}$, this gives:

$$\hat{\boldsymbol{D}}'_{1,3} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{D}'_{1,3} = \boldsymbol{D}_{1,2} \cdot \boldsymbol{C}''_{1,3} \cdot \boldsymbol{x} = \boldsymbol{G}''_{1,3} \cdot \boldsymbol{x}$$

where $\hat{\boldsymbol{D}}'_{1,3}$ is the first column vector of $\hat{\boldsymbol{D}}_{1,3}$. Applying the vector $\boldsymbol{x}$ on (22), we therefore get:

$$\boldsymbol{\Omega} \cdot \boldsymbol{x} = \boldsymbol{u} \cdot \boldsymbol{x} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{D}}'_{1,3}$$

We claim that this provides a good estimate of the first component of the error vector $\boldsymbol{E}$ from (23). Recall that the components of $\boldsymbol{x}$ are in $R \otimes_{\mathbb{Z}} \mathbb{Q}$, so by rounding to the nearest integer we can get the following value in R:

$$E' = \lfloor \boldsymbol{\Omega} \cdot \boldsymbol{x} \rceil = \lfloor \boldsymbol{u} \cdot \boldsymbol{x} \rceil + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{D}}'_{1,3} \tag{24}$$

Since the components of $\boldsymbol{u}$ (over R) are small, and moreover the coefficients of $\boldsymbol{x}$ (over $R \otimes_{\mathbb{Z}} \mathbb{Q}$) are also small (heuristically), the scalar $\lfloor \boldsymbol{u} \cdot \boldsymbol{x} \rceil$ in R is small.

Finally, letting as previously:

$$\tilde{\boldsymbol{D}}_{1,1} = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{C}_{1,1,i} \,,$$

we obtain:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} = s_1 \cdot \boldsymbol{A}_{1,2} + \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \pmod q \,.$$

which gives as previously:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} \cdot \hat{\boldsymbol{D}}'_{1,3} = s_1 \cdot s_2 \cdot s_3 \cdot A_0 + s_1 \cdot \hat{F}_{1,3} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{D}}'_{1,3} \pmod q$$

Therefore combining with (24) we can compute from public parameters:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} \cdot \hat{\boldsymbol{D}}'_{1,3} - E' = s_1 \cdot s_2 \cdot s_3 \cdot A_0 + s_1 \cdot \hat{F}'_{1,3} - \lfloor \boldsymbol{u} \cdot \boldsymbol{x} \rceil \pmod q$$

Since the terms $s_1 \cdot \hat{F}'_{1,3}$ and $\lfloor \boldsymbol{u} \cdot \boldsymbol{x} \rceil$ are small, this reveals the first component of the secret vector $s_1 \cdot s_2 \cdot s_3 \cdot \boldsymbol{A}_0$, which breaks the scheme.

Acknowledgments. This work has been supported in part by the European Union's H2020 Programme under grant agreement number ICT-644209.

References

[BF14] Jean-François Biasse and Claus Fieker. Subexponential class group and unit group computation in large degree number fields. *LMS Journal of Computation and Mathematics*, 17(suppl. A):385–403, 2014.

[BGH$^+$15] Zvika Brakerski, Craig Gentry, Shai Halevi, Tancrède Lepoint, Amit Sahai, and Mehdi Tibouchi. Cryptanalysis of the quadratic zero-testing of GGH. Cryptology ePrint Archive, Report 2015/845, 2015. Available at `https://eprint.iacr.org/2015/845`.

[Bia14] Jean-François Biasse. Subexponential time ideal decomposition in orders of number fields of large degree. *Advances in Mathematics of Communications*, 8(4):407–425, 2014.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2002.

[BS16] Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In Robert Krauthgamer, editor, *SODA 2016*, pages 893–902. ACM, 2016.

[CDPR15] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. Cryptology ePrint Archive, Report 2015/313, 2015. Available at `https://eprint.iacr.org/2015/313`. To appear at EUROCRYPT 2016.

[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 Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 247–266. Springer, 2015.

[CGS14] Peter Campbell, Michael Groves, and Dan Shepherd. Soliloquy: A cautionary tale. ETSI 2nd Quantum-Safe Crypto Workshop, 2014. Available at `https://docbox.etsi.org/Workshop/2014/201410_CRYPTO/S07_Systems_and_Attacks/S07_Groves_Annex.pdf`.

[CHL$^+$15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 3–12. Springer, 2015.

[CLR15] Jung Hee Cheon, Changmin Lee, and Hansol Ryu. Cryptanalysis of the new CLT multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015. Available at `https://eprint.iacr.org/2015/934`.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. Springer, 2013.

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 267–286. Springer, 2015.

[Dev16] The Sage Developers. *Sage Mathematics Software (Version 7.0)*, 2016. `http://www.sagemath.org`.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In Omer Reingold, editor, *FOCS 2013*, pages 40–49. IEEE Computer Society, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 498–527. Springer, 2015.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Cynthia Dwork, editor, *STOC 2008*, pages 197–206. ACM, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, 2013.

[Hal15a] Shai Halevi. Graded encoding, variations on a scheme. Cryptology ePrint Archive, Report 2015/866, 2015. Available at `https://eprint.iacr.org/2015/866`.

[Hal15b] Shai Halevi. Private communication, 2015.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. Cryptology ePrint Archive, Report 2015/301, 2015. Available at `https://eprint.iacr.org/2015/301`. To appear at EUROCRYPT 2016.

[Jou00] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In Wieb Bosma, editor, *ANTS-IV*, volume 1838 of *LNCS*, pages 385–394. Springer, 2000.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In Janos Simon, editor, *STOC 1988*, pages 20–31. ACM, 1988.

[LSS14] Adeline Langlois, Damien Stehlé, and Ron Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 239–256. Springer, 2014.

[MF15] Brice Minaud and Pierre-Alain Fouque. Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941, 2015. Available at `https://eprint.iacr.org/2015/941`.

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

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. *IACR Cryptology ePrint Archive*, 2016:147, 2016.

[PS15] Alice Pellet-Mary and Damien Stehlé. Cryptanalysis of Gu's ideal multilinear map. Cryptology ePrint Archive, Report 2015/759, 2015. Available at `https://eprint.iacr.org/2015/759`.

[Rud08] Mark Rudelson. Invertibility of random matrices: Norm of the inverse. *Annals of Mathematics*, 168(2):575–600, 2008.

A Multipartite Key Agreement Based on GGH15 Multilinear Maps

We recall the description of the GGH15-based multipartite key-agreement protocol from [GGH15, Section 5.1]:

- $\mathsf{Setup}(1^\lambda, k)$: the setup algorithm takes as input the security parameter 1^λ and the total number of users k.
  1. Run the parameter-generation and instance-generation of the graph-based encoding scheme for the graph with k chains with a common end-point, each of length k edges. Let denote $e_{i,j}$ the j-th edge on the i-th chain, for $1 \leq i, j \leq k$.
  2. Using the secret parameters, run the sampling procedure of the encoding scheme to choose random plaintext elements $t_{i,\ell}$ for $1 \leq i \leq k$ and $1 \leq \ell \leq N$, and for each $t_{i,\ell}$ compute an encoding of it relative to all the edges $e_{i',j}$ for $j = (i + i' - 2 \bmod k) + 1$, denoted by $\boldsymbol{C}_{i',j,\ell}$.[4]

  The public parameters pp include the public parameters of the encoding scheme, *i.e.* the matrices for all the source nodes $\boldsymbol{A}_{i,1}$, and also the encoding matrices $\boldsymbol{C}_{i,j,\ell}$ for all $1 \leq i, j \leq k$ and $1 \leq \ell \leq N$.
- $\mathsf{Publish}(\mathsf{pp}, i)$: the i-th party chooses random small plaintext elements $r_{i,\ell} \leftarrow \chi$ for all $1 \leq \ell \leq N$ and sets $\boldsymbol{D}_{i',j} \leftarrow \sum_\ell \boldsymbol{C}_{i',j,\ell}$ for all $1 \leq i' \leq k$, with $j = (i + i' - 2 \bmod k) + 1$. It keeps $\boldsymbol{D}_{i,(2i-2 \bmod k)+1}$ private and broadcast all the other $\boldsymbol{D}_{i',j}$.
- $\mathsf{Keygen}(\mathsf{PP}, i, \mathsf{sk}_i, \{\mathsf{pub}_j\}_{j \neq i})$: Party i collects all the matrices $\boldsymbol{D}_{i,j}$ (encoding the secrets $s_{i'}$ relative to its chain i). The session-key is then computed from the high-order bits of:

$$\boldsymbol{A}_{i,1} \cdot \prod_{j=1}^{k} \boldsymbol{D}_{i,j} \bmod q$$

[4] We use a slightly different notation from that used in [GGH15], where $\boldsymbol{C}_{i,\ell,i'}$ is used instead.

B Breaking Some Variants of GGH13

Before this paper, the multipartite Diffie–Hellman key agreement—first application of the candidates multilinear maps—has been shown to be insecure for two out of three candidates (namely GGH13 and CLT13). In an attempt to reinstate the protocol for GGH13, Halevi attempted to reinstate the key-agreement protocol for GGH13 by borrowing ideas from the key-agreement protocol of GGH15 [Hal15a]. In this section, we describe attacks against the variants proposed for GGH13. The attacks differ from the main attack described in Section 3.

B.1 The GGH13 Scheme

We briefly recall the (symmetric) GGH13 graded encoding scheme; we refer to [GGH13a] for a full description. The operations of that scheme work over the polynomial rings $R = \mathbb{Z}[x]/(f(x))$ and $R_q = R/qR$ for some degree n irreducible integer polynomial $f(x) \in \mathbb{Z}[x]$ and an integer q. The plaintext space is $R_g = R/\mathcal{I}$ where $\mathcal{I} = \langle g \rangle = gR$ for $g \in R$ a small (secret) element. The zero-test parameter is $p_{zt} = h \cdot z^k/g \in R_q$, where h is somewhat small (of size $\approx q^{1/2}$), z is a random (secret) multiplicative mask and k is the multi-linearity parameter.

Encoding. A level-i encoding u of $m \in R_g$ and size bound ν is $u = c/z^i \in R_q$ where $c \in m + gR$ is of size $\|c\| \leq \nu$. For $\nu > n \log n$, such a c can be found using the GPV sampler (Theorem 1 below) and using a basis of $\mathcal{I}$.

Theorem 1 ([GPV08], Theorem 4.1). *There is a probabilistic polynomial time algorithm that, given a basis $\boldsymbol{B}$ of an n-dimensional lattice L, a parameter $s \geq |\tilde{B}| \cdot \omega(\sqrt{\log n})$ and a center $c \in \mathbb{R}^n$, outputs a sample from a distribution that is statistically close to the discrete Gaussian distribution over L of center c and parameter s.*

Encodings at the same level can be added (and the underlying plaintext values get added modulo $\mathcal{I}$), and a level-i encoding can be multiplied with a level-j encoding when $i + j \leq k$ and the result is a level-$(i + j)$ encoding of the multiplication of the plaintext values modulo $\mathcal{I}$.

Extraction/zero-testing. Given a level-k encoding $v = (m + \xi \cdot g)/z^k \in R_q$ for some $\xi \in R$, we can compute

$$\omega = p_{zt} \cdot v \bmod q = h \cdot (m/g + \xi) \bmod q\,.$$

Since ξ is small and h is somewhat small, the high-order bits of ω only depends on m; we say that the extraction of v are the high-order bits of the coefficients of ω. Also, one can test whether v encodes $m = 0$: the high order bits of ω would all be equal to 0, *i.e.* ω would be small.

Public encodings. In order to perform a multipartite key-agreement protocol using GGH13 multilinear maps, it was suggested in [GGH13a,LSS14] to add to the public parameters the following encodings:

- $y = e/z \in R_q$ where $e \in 1 + gR$ is small;
- $x_0 = b_0 \cdot g/z \in R_q$ and $x_1 = b_1 \cdot g/z \in R_q$ where b_0, b_1 are small in R.

Therefore a randomized level-1 encoding corresponding to a level-0 encoding u (sampled directly from a small Gaussian distribution) can be generated as

$$\hat{u} = u \cdot y + \rho_0 \cdot x_0 + \rho_1 \cdot x_1 \in R_q$$

for small $\rho_0, \rho_1 \in R$.

Classical and quantum security. The parameters of GGH13 were chosen heuristically so that computing (classically and quantumly) the discrete-log of an encoding should be hard. In particular it is easy to see that g has to remain secret. Now, finding a short generator of a principal ideal (here recovering g from $\mathcal{I} = \langle g \rangle$) is called the *Short Generator of a Principal Ideal Problem* (SG-PIP). Recent research [BF14, Bia14, CGS14, BS16, CDPR15] suggests that there exists a quantum attack that solves this problem in polynomial time. Therefore, finding any basis of the plaintext space $R_g = R/\mathcal{I}$ (or of $\mathcal{I}$) seems intuitively at the heart of the security of GGH13 and is therefore considered as the *core computation task* of the adversary [Hal15a, Section 2.2].

B.2 Breaking the Key Agreement of GGH13

In [GGH13a], Garg *et al.* described a N-partite Diffie–Hellman key agreement using GGH13 with multi-linearity level $k = N - 1$. Each party $i \in [1, N]$ samples a random level-0 encoding $c_i = a_i + g \cdot \alpha_i \in R$, and publishes a randomized level-1 encoding

$$\hat{c}_i = c_i \cdot y + \rho_0 \cdot x_0 + \rho_1 \cdot x_1 \,,$$

with small ρ_0, ρ_1. Each party can then compute the product of all the public encodings except its own, and multiply it by its secret level-0 encoding; *i.e.* party i computes:

$$s_i = \left(\prod_{j \neq i} \hat{c}_j \right) \cdot c_i \,.$$

All the s_i's are then level-k encodings of the plaintext $\prod_{j=1}^{N} a_j$, and therefore extract to the same shared key s using the extraction procedure of GGH13. Security of the protocol means that the derived key s should be pseudo-random given the public parameters and all the broadcast information. We call the N-partite Diffie–Hellman assumption the assumption that it is hard to distinguish the high-order bits of the $(s_i \cdot p_{zt} \bmod q)$, *i.e.* s, from random bits.

Unfortunately, this protocol was shown to be insecure (and thus the N-partite Diffie–Hellman assumption to be easy for GGH13). The first attack is a weak discrete-log attack and further enables, from the public parameters, to recover the ideal $\mathcal{I} = \langle g \rangle$ [GGH13a, CGH$^+$15]. As discussed in the previous section, it likely follows that there exists a polynomial time quantum attack (and a subexponential classical attack) that recovers the secret element g from $\mathcal{I}$. The second attack [HJ15] builds upon the first attack and classically breaks the key agreement. We recall the two attacks below.

Attack 1: classical weak discrete-log and quantum discrete-log. We start with a level-1 encoding $u = (e + \xi \cdot g)/z \in R_q$ corresponding to a short secret $e \in R$. We compute

$$\omega_u = (u \cdot x_0 \cdot y^{k-2} \cdot p_{zt}) \bmod q = e \cdot b_0 \cdot h + \xi_u \cdot g\,.$$

The right hand side has no reduction modulo q. Similarly, we also compute

$$\omega_0 = (x_0 \cdot y^{k-1} \cdot p_{zt}) \bmod q = b_0 \cdot h + \xi_0 \cdot g\,.$$

We can now compute

$$E = \omega_u/\omega_0 \bmod \langle g \rangle = e \bmod \langle g \rangle\,,$$

which gives

$$E = e + \beta \cdot g\,.$$

This is a weak discrete-log attack: we recover a (large) element of the coset $e + \mathcal{I}$ rather than e itself (which is small).

Recovering $\mathcal{I} = \langle g \rangle$. This attacks enables to recover $\mathcal{I}$ from the public parameters. Indeed, when u is a level-0 encoding of 0, the previous attack yields $\omega_u = g \cdot \xi_u \in \mathcal{I}$. Therefore, by running the attack on x_0, x_1 and rerandomizations thereof, one obtains a lot of ω_u's, all in $\mathcal{I}$. By computing GCDs, with high probability one can recover a basis of $\mathcal{I}$.

Quantum discrete-log attack. If there indeed exists a quantum polynomial time attack that recovers g from $\mathcal{I} = \langle g \rangle$, it then suffices to compute ($E \bmod g$) to recover e directly, the discrete-log of u. This obviously would break the full key agreement, as one could recover the plaintext values of all the public encodings.

Attack 2: breaking GGH13 key agreement classically. Now, on a classical computer, recovering g from $\mathcal{I}$ takes at least subexponential time. We describe below a classical polynomial-time attack of Hu and Jia against the key-agreement protocol [HJ15]. First assume that we recovered $\mathcal{I}$ as in the previous attack. Recall that we wish to compute the high-order bits of (say)

$$\omega = c_1 \cdot \left(\prod_{j \neq 1} \hat{c}_j \right) \cdot p_{zt} \bmod q\,,$$

as this would break the N-Diffie–Hellman assumption. Note that this is similar to recovering the high-order bits of

$$\omega' = a_1 \cdot \left(\prod_{j \neq 1} \hat{c}_j \right) \cdot p_{zt} \bmod q\,.$$

To do so, we first compute

$$Y = x_0 \cdot \left(\prod_{j \neq 1,2} \hat{c}_j \right) \cdot p_{zt} \bmod q\,,$$

which gives

$$Y = h \cdot b_0 \cdot \left(\prod_{j \neq 1,2} a_j + \xi_y \cdot g \right) \equiv h \cdot b_0 \cdot \left(\prod_{j \neq 1,2} a_j \right) \pmod{\langle b_0 \cdot g \rangle}.$$

The previous attack allows to recover $C_1 = a_1 + \xi_1 \cdot g$ for a large ξ_1 from $\hat{c}_1$. We compute $W = Y \cdot C_1$ over R, which gives

$$W \equiv a_1 \cdot h \cdot b_0 \cdot \prod_{j \neq 1,2}^{k} a_i \equiv a_1 \cdot Y \pmod{\langle b_0 \cdot g \rangle}$$

We also compute

$$X = x_0^2 \cdot y^{k-2} \cdot p_{zt} \bmod q = \xi_X \cdot b_0 \cdot g.$$

We can now compute

$$W' = W \bmod \langle X \rangle.$$

Since X is a multiple of $b_0 \cdot g$, this also gives

$$W' \equiv a_1 \cdot Y \pmod{\langle b_0 \cdot g \rangle}.$$

The crucial point is that now W' is small; therefore we have

$$W' = a_1 \cdot Y + \xi \cdot b_0 \cdot g$$

for a somewhat small ξ. Eventually we can compute

$$W'' = W' \cdot \hat{c}_2 / x_0 \bmod q,$$

which gives using $\hat{c}_2 = (c_2 + \xi_2 \cdot g)/z$ for some small $\xi_2 \in R$:

$$\begin{aligned} W'' &= a_1 \cdot Y \cdot \hat{c}_2 / x_0 + \xi \cdot b_0 \cdot g \cdot \hat{c}_2 / x_0 \bmod q \\ &= a_1 \cdot \left(\prod_{j \neq 1} \hat{c}_j \right) \cdot p_{zt} + \xi \cdot (c_2 + \xi_2 g) \bmod q \\ &= \omega' + \xi \cdot (c_2 + \xi_2 g) \bmod q. \end{aligned}$$

Finally, since ξ, ξ_2 and c_2 are small, the high-order bits of W'' are the same as those of ω' (and therefore as those of ω); this breaks the key-exchange protocol.

B.3 Another Key-Agreement Protocol With GGH13 and Its Cryptanalysis

In [Hal15a, Section 7], Halevi proposed a key-agreement protocol for GGH13 inspired from the GGH15-based protocol, and put forward a simple hardness assumption as target for cryptanalysis. Below, we describe this protocol and describe a polynomial-time attack against the hardness assumption.

Asymmetric GGH13. We begin by describing how the *asymmetric* variant of GGH13 works. Instead of one single secret multiplicative mask z, we have (say) k different masks $z_1, \dots, z_k$. The zero-test parameter then becomes $p_{zt} = h \cdot \big(\prod_{j=1}^{k} z_j\big)/g \in R_q$, *i.e.* only encodings relative to all these masks can be extracted/zero-tested. In particular, when someone has k encodings, respectively relative to a single z_j for all j, the only-way to create an encoding that can be zero-tested is to multiply them altogether with multiplicity one. This asymmetric GGH13 is at the heart of the indistinguishability obfuscation (iO) candidates [GGH+13b].

A new key-agreement protocol. We recall the N-party key-agreement protocol from [Hal15a, Section 7]. We choose $k = N^2$ secret random masks $z_{i,j}$ under the constraint that for all i, the $\prod_{j=1}^{N} z_{i,j}$'s are equal to some random $Z \in R_q$. The zero-test parameter that will be used for the extraction is then $p_{zt} = h \cdot Z/g \in R_q$.

Public encodings. For every $j \in [N]$, sample co-prime $a_j, b_j \in R$ with coefficients drawn from a discrete Gaussian distribution over $\mathbb{Z}^n$. Then for all j, sample N level-0 encodings $a_{1,j}, \dots a_{N,j}$ of a_j, and N level-0 encodings $b_{1,j}, \dots b_{N,j}$ of b_j. Finally, for all (i,j), mask both $a_{i,j}$ and $b_{i,j}$ with $z_{i,j}$:

$$A_{i,j} = (a_{i,j}/z_{i,j}) \in R_q, \qquad B_{i,j} = (b_{i,j}/z_{i,j}) \in R_q.$$

The public encodings are the $A_{i,j}$'s and $B_{i,j}$'s.

Protocol. In the protocol, the party j chooses two random scalars $\alpha_j, \beta_j \in R$ from a Gaussian distribution, and set

$$C_{i,j} = \alpha_j \cdot A_{i,j} + \beta_j \cdot B_{i,j} \in R_q, \quad i \in [N]\,.$$

Party j publishes all the $C_{i,j}$ but keeps $C_{j,j}$ secret. Then using the broadcast values of the other parties and its own secret $C_{j,j}$, party j can compute

$$s_j = \prod_{j'=1}^{N} C_{j,j'} \in R_q\,.$$

All the s_j's are encodings of $\prod_{j'=1}^{N} (\alpha_{j'} \cdot a_{j'} + \beta_{j'} \cdot b_{j'})$ relative to the mask $\prod_{j'} z_{j,j'} = Z$. Party j can then use the extraction procedure with p_{zt}.

Simple hardness assumption. The hardness assumption proposed by Halevi in [Hal15a, Section 7.1] is, given the $A_{i,j}$'s, $B_{i,j}$'s and p_{zt} generated as above, it should be difficult to find any basis of $\mathcal{I} = \langle g \rangle$.

The key idea behind this key-agreement protocol was to avoid giving any encoding of 0, or the creation thereof, below the top-level. Therefore each party can, from two public encodings, generate a random-looking encoding for each $z_{i,j}$ and while keeping its index-(j,j) encoding secret, reveal the index-(i,j)'s encodings of the same plaintext value for all $i \neq j$. The protocol's graph is given in Figure 5.

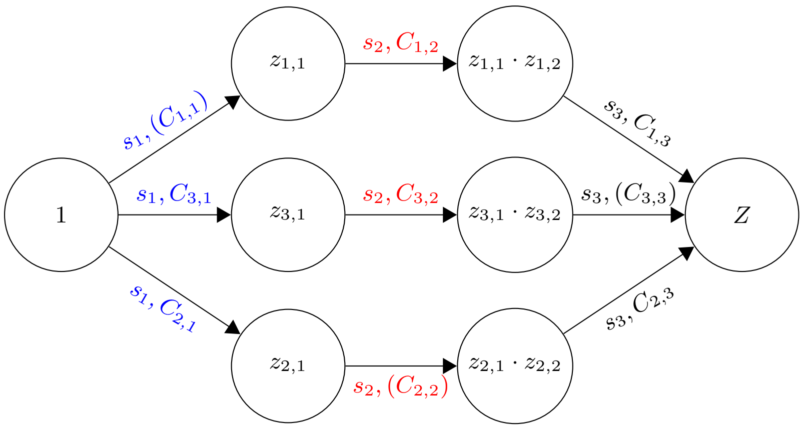

Fig. 5. Graph of a key agreement between 3 parties for GGH13 from [Hal15a, Section 7]. The vertices contain the secret masks, and encodings are represented on the edges (their mask is on the destination node). Each party is represented by a different color, keeps the encoding in parenthesis secret and publishes the two other encodings.

Attacking the simple hardness assumption. For ease of presentation, we describe the attack for three parties; the attack extends easily to more parties. For simplicity, we do not write the masks nor the reduction modulo q; of course the attack only do operations permitted by the masks, which basically means manipulating product of elements as long as they are masked by Z.

From the public encodings, one can compute

$$\begin{aligned} E_1 &= A_{1,1} \cdot A_{1,2} = a_1 \cdot a_2 + \varepsilon_1 \cdot g & F_1 &= A_{1,3} = a_3 + \phi_1 \cdot g \\ E_2 &= A_{2,1} \cdot A_{2,2} = a_1 \cdot a_2 + \varepsilon_2 \cdot g & F_2 &= A_{2,3} = a_3 + \phi_2 \cdot g \\ E_3 &= A_{3,1} \cdot A_{3,2} = a_1 \cdot a_2 + \varepsilon_3 \cdot g & F_3 &= A_{3,3} = a_3 + \phi_3 \cdot g \end{aligned}$$

Now, the above encodings can be multiplied on a row by row basis, and the resulting encoding encodes $a_1 \cdot a_2 \cdot a_3$ respective to the mask Z. One can therefore compute

$$\omega = p_{zt} \cdot (E_1 \cdot F_1 + E_2 \cdot F_2 - 2 \cdot E_3 \cdot F_3)$$

and since we zero-test an element that encodes 0, we get ω over R and not modulo q. Now, note that

$$\omega = \begin{bmatrix} E_1 & E_2 & -2E_3 \end{bmatrix} \cdot \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \cdot p_{zt} \,. \tag{25}$$

Assume for now that, for all (i, j) we have not two encodings $A_{i,j}$, $B_{i,j}$, but also a third encoding $C_{i,j}$ corresponding to plaintexts c_j. We can therefore compute the same ω as in (25) but with 3 different row vectors corresponding to a_j, b_j and c_j respectively, and 3 different column vectors corresponding to a_j, b_j and c_j. As in the zeroizing attacks [CHL$^+$15, CGH$^+$15], we can therefore extend ω into a matrix W by using these three different row vectors and three different column vectors, and the matrix W is full rank with high probability. Also, modulo $\mathcal{I} = \langle g \rangle$, we get

$$\omega = (\varepsilon_1 + \varepsilon_2 - 2\varepsilon_3) \cdot (h \cdot a_3) + (h \cdot a_1 \cdot a_2) \cdot (\phi_1 + \phi_2 - 2\phi_3) \bmod \langle g \rangle \,,$$

which can be written as

$$\omega = \begin{bmatrix} \varepsilon_1 + \varepsilon_2 - 2\varepsilon_3 & h \cdot a_1 \cdot a_2 \end{bmatrix} \cdot \begin{bmatrix} h \cdot a_3 \\ \phi_1 + \phi_2 - 2\phi_3 \end{bmatrix} \bmod \langle g \rangle \,.$$

The crucial observation is that since we now have vectors of dimension 2 instead of 3, the matrix W cannot be invertible modulo $\mathcal{I} = \langle g \rangle$.[5] Therefore we can compute the determinant of W over R and get a non-zero multiple of g. By collecting many such multiples and taking GCDs, we can recover a basis for $\mathcal{I} = \langle g \rangle$.

Note that we assumed that three encodings are available for all (i, j). We can simulate this by working with 6 parties and multiplying the encodings quadratically to get $2 \cdot 2 = 4$ encodings. Also, it was pointed out to us that the attack works even when the number of parties is three. Indeed, if only two encodings are available for each (i, j), the set of values that each party broadcast gives the attacker a set of $N - 1$ linear equations over R_q in two variables (therefore the protocol is definitively insecure if one has less than N encodings to choose from) [Hal15b].

B.4 Graph-GGH13: a GGH13 Variant With Graph Constraints, a New Candidate Key Agreement and Its Cryptanalysis.

Still in [Hal15a, Section 6], Halevi proposed a variant of GGH13 with the same graph constraints as in GGH15. This Graph-GGH13 scheme can then be used to instantiate the GGH15 key-agreement protocol. In the rest of this section, we describe Graph-GGH13 and the corresponding third candidate key-agreement protocol, and finally an attack thereon.

Graph-GGH13. In the following, we assume that all the graphs we consider are DAG, with a single source and a single sink [Hal15a].

Parameters generation. The parameters generation takes as inputs the security parameter λ and a graph $\mathcal{G} = (V, E)$ with source s and sink t. As for GGH13, the operations of Graph-GGH13 work over some polynomial rings $R = \mathbb{Z}[x]/(f(x))$ and $R_q = R/qR$ for some degree n irreducible integer polynomial $f(x) \in \mathbb{Z}[x]$ and an integer q (note that n, q also depends on the diameter of the graph). As in GGH13, the plaintext space is $R_g = R/\mathcal{I}$ where $\mathcal{I} = \langle g \rangle$ for $g \in R$ a small (secret) element. Each vertex $v \in V$ is associated a (secret) random invertible matrix $\boldsymbol{P}_v \in \mathbb{Z}_q^{n \times n}$. Finally, the (public) zero-test parameter is a tuple $(\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{w}})$ such that $\tilde{\boldsymbol{v}} = \boldsymbol{v}^T \cdot \boldsymbol{P}_s \bmod q$ and $\tilde{\boldsymbol{w}} = \boldsymbol{P}_t^{-1} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w} \bmod q$ where $\boldsymbol{v}, \boldsymbol{w}$ are small random vectors and $\boldsymbol{G}^{-1} \in \mathbb{Z}_q^{n \times n}$ is the divide-by-g matrix.

Encoding. In Graph-GGH13, encodings are relative to the paths of $\mathcal{G}$. An encoding of $m \in R_g$ relative to the path $u \rightsquigarrow v$ of $\mathcal{G}$ is a matrix

$$\tilde{\boldsymbol{C}} = \boldsymbol{P}_u^{-1} \cdot \boldsymbol{C} \cdot \boldsymbol{P}_v \bmod q\,,$$

where $\boldsymbol{C} \in \mathbb{Z}_q^{n \times n}$ is a multiply-by-c matrix for a small $c = m + \alpha \cdot g$ (namely, a level-0 encoding of GGH13).

Operations. Addition and multiplication are just matrix addition and multiplication over $\mathbb{Z}_q$. Note that we can only add encodings relative to the same path $u \rightsquigarrow v$. Two encodings relative to paths $u_1 \rightsquigarrow v_1$ and $u_2 \rightsquigarrow v_2$ can only be multiplied when $v_1 = u_2$.

[5] This key observation is also a key element in the cryptanalysis of CLT15 [CLT15] by Minaud and Fouque [MF15].

Extracting/zero-testing. To extract/zero-test an encoding, it has to be relative to the source-to-sink path $s \rightsquigarrow t$. Such an encoding is of the form

$$\tilde{\boldsymbol{C}} = \boldsymbol{P}_s^{-1} \cdot \boldsymbol{C} \cdot \boldsymbol{P}_t \bmod q$$

where the multiply-by-c matrix $\boldsymbol{C}$ is such that $c = m + g \cdot \xi$ for a small ξ. Similarly to GGH13, one can compute

$$\omega = \tilde{\boldsymbol{v}} \cdot \tilde{\boldsymbol{C}} \cdot \tilde{\boldsymbol{w}} = \boldsymbol{v}^T \cdot \boldsymbol{C} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w} \bmod q.$$

Now, $\boldsymbol{C} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w}$ is the vector representation of the ring element $c \cdot g^{-1} \cdot w = (m/g + \xi) \cdot w \in R_q$ (a classical GGH13 extraction) and as for GGH13, its high-order bits only depends on m. This remains true for ω when the coefficients of $\boldsymbol{v}$ are small.

GGH15-like key agreement for Graph-GGH13. Graph-GGH13 can be used to implement the graph-based key agreement protocol of [GGH15], which we briefly describe in the following. We only describe the setup procedure of the protocol, as it is the only step relevant to our attack. For simplicity, we describe the protocol for 3 users; the protocol and the attack can easily be extended to 4 users and more.

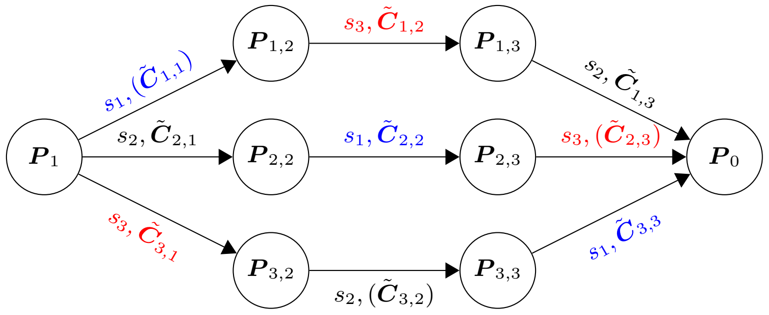
$\boldsymbol{P}_1$
$s_1, (\tilde{\boldsymbol{C}}_{1,1})$
$s_2, \tilde{\boldsymbol{C}}_{2,1}$
$s_3, \tilde{\boldsymbol{C}}_{3,1}$
$\boldsymbol{P}_{1,2}$
$\boldsymbol{P}_{2,2}$
$\boldsymbol{P}_{3,2}$
$s_3, \tilde{\boldsymbol{C}}_{1,2}$
$s_1, \tilde{\boldsymbol{C}}_{2,2}$
$s_2, (\tilde{\boldsymbol{C}}_{3,2})$
$\boldsymbol{P}_{1,3}$
$\boldsymbol{P}_{2,3}$
$\boldsymbol{P}_{3,3}$
$s_2, \tilde{\boldsymbol{C}}_{1,3}$
$s_3, (\tilde{\boldsymbol{C}}_{2,3})$
$s_1, \tilde{\boldsymbol{C}}_{3,3}$
$\boldsymbol{P}_0$

Fig. 6. Graph of a key agreement between 3 parties for Graph-GGH13 from [Hal15a, Section 6]. The vertices contain the secret random matrices, and encodings are represented on the edges. Each party is represented by a different color, keeps the encoding in parenthesis secret and publishes the two other encodings.

Let us consider the DAG of Figure 6. As explained before, we associate to each vertex a (secret) random invertible matrices: $\boldsymbol{P}_0$ for the sink, $\boldsymbol{P}_1$ for the source, and matrices $\boldsymbol{P}_{i,j}$'s over $\mathbb{Z}_q^{n \times n}$ for the inner vertices. At each edge, a large enough number N of small secret elements $s_{i,\ell} \in R$ are encoded in matrices in $\mathbb{Z}_q^{n \times n}$ in a "round robin" fashion. More precisely, for $1 \leq i, j \leq 3$, $1 \leq \ell \leq N$, define

$$c_{i,j,\ell} = s_{(j-i) \bmod 3+1,\ell} + \alpha_{i,j,\ell} \cdot g,$$

where $\alpha_{i,j,\ell}$ is a small random element in R. Then

$$\tilde{\boldsymbol{C}}_{i,j,\ell} = \boldsymbol{P}_{i,j}^{-1} \cdot \boldsymbol{C}_{i,j,\ell} \cdot \boldsymbol{P}_{i,j+1} \bmod q, \qquad 1 \leq i, j \leq 3, \quad 1 \leq \ell \leq N\,,$$

with $\boldsymbol{P}_{i,4} := \boldsymbol{P}_0$ and $\boldsymbol{P}_{1,j} := \boldsymbol{P}_1$, where $\boldsymbol{C}_{i,j,\ell}$'s are multiply-by-$c_{i,j,\ell}$ matrices. These matrices $\tilde{\boldsymbol{C}}_{i,j,\ell}$ as well as $\tilde{\boldsymbol{v}} = \boldsymbol{v}^T \cdot \boldsymbol{P}_1 \bmod q$ and $\tilde{\boldsymbol{w}} = \boldsymbol{P}_0^{-1} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w} \bmod q$ are part of the public parameters. Users can then generate public and private encodings by linear combinations of these public encodings, and eventually obtain a shared key as in Section 2.2.

In the following section, we describe how the secret basis $\mathcal{I} = \langle g \rangle$ can be obtained in polynomial time from these public encodings.

Breaking the key agreement of GGH15 when instantiated with Graph-GGH13. Similarly to the attack of Section 3, we consider:

- a fixed index $i = 1$ for the encodings corresponding to $s_{1,i}$, and for simplicity we write $\boldsymbol{C}_{1,1} := \boldsymbol{C}_{1,1,1}$ and $\boldsymbol{C}_{2,2} := \boldsymbol{C}_{2,2,1}$;
- a fixed index $j = 1$ for the encodings corresponding to $s_{2,j}$, and for simplicity we write $\boldsymbol{C}_{1,2} := \boldsymbol{C}_{1,2,1}$ and $\boldsymbol{C}_{2,3} := \boldsymbol{C}_{2,3,1}$;
- a fixed index $k = 1$ for the encodings corresponding to $s_{3,k}$, and for simplicity we write $\boldsymbol{C}_{1,3} := \boldsymbol{C}_{1,3,1}$ and $\boldsymbol{C}_{2,1} := \boldsymbol{C}_{2,1,1}$;

We start with taking the difference between keys from two source-to-sink paths, to obtain a single small integer over $\mathbb{Z}$:

$$\begin{aligned}
\omega &= \tilde{\boldsymbol{v}} \cdot \tilde{\boldsymbol{C}}_{1,1} \cdot \tilde{\boldsymbol{C}}_{1,2} \cdot \tilde{\boldsymbol{C}}_{1,3} \cdot \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{v}} \cdot \tilde{\boldsymbol{C}}_{2,1} \cdot \tilde{\boldsymbol{C}}_{2,2} \cdot \tilde{\boldsymbol{C}}_{2,3} \cdot \tilde{\boldsymbol{w}} \pmod q \\
&= \boldsymbol{v}^T \cdot \boldsymbol{C}_{1,1} \cdot \boldsymbol{C}_{1,2} \cdot \boldsymbol{C}_{1,3} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w} \\
&\quad - \boldsymbol{v}^T \cdot \boldsymbol{C}_{2,1} \cdot \boldsymbol{C}_{2,2} \cdot \boldsymbol{C}_{2,3} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w} \pmod q \\
&= \begin{bmatrix} \boldsymbol{v}^T \cdot \boldsymbol{C}_{1,1} & -\boldsymbol{v}^T \cdot \boldsymbol{C}_{2,2} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{C}_{1,2} & \\ & \boldsymbol{C}_{2,3} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{C}_{1,3} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w} \\ \boldsymbol{C}_{2,1} \cdot \boldsymbol{G}^{-1} \cdot \boldsymbol{w} \end{bmatrix} .
\end{aligned}$$

Since all the entries are small, the last equation holds over $\mathbb{Z}$. Note that we changed the order of the matrices $\boldsymbol{C}_{i,j}$ since they commute.

Now as in the Cheon *et al.* attack, we can extend ω to a matrix by considering many left row vectors (varying i) and many right column vectors (varying k). We can hence obtain a matrix $\boldsymbol{W} \in \mathbb{Z}^{2n \times 2n}$ with

$$\boldsymbol{W} = \boldsymbol{A} \cdot \boldsymbol{C}_j \cdot \boldsymbol{B}.$$

Similarly, using a $j' \neq j$, we can obtain another matrix $\boldsymbol{W'}$ with

$$\boldsymbol{W'} = \boldsymbol{A} \cdot \boldsymbol{C}_{j'} \cdot \boldsymbol{B} \quad \text{where } \boldsymbol{C}_{j'} = \begin{bmatrix} \boldsymbol{C}_{1,2,j'} & \\ & \boldsymbol{C}_{2,3,j'} \end{bmatrix} .$$

By computing over the rationals we obtain:

$$\boldsymbol{W} \cdot \boldsymbol{W'}^{-1} = \boldsymbol{A} \cdot \boldsymbol{C}_j \cdot \boldsymbol{C}_{j'}^{-1} \cdot \boldsymbol{A}^{-1} = \boldsymbol{A} \cdot \begin{bmatrix} \boldsymbol{C}_{1,2,j}/\boldsymbol{C}_{1,2,j'} & \\ & \boldsymbol{C}_{2,3,j}/\boldsymbol{C}_{2,3,j'} \end{bmatrix} \cdot \boldsymbol{A}^{-1}.$$

Now by the Cayley–Hamilton theorem, the rational matrices $\boldsymbol{C}_{1,2,j}/\boldsymbol{C}_{1,2,j'}$ as well as $\boldsymbol{C}_{2,3,j}/\boldsymbol{C}_{2,3,j'}$ are roots of the characteristic polynomial χ of $\boldsymbol{W} \cdot \boldsymbol{W'}^{-1}$. The crucial observation here is that both $c_{1,2,j}/c_{1,2,j'}$ and $c_{2,3,j}/c_{2,3,j'}$ are actually roots of χ over $\mathbb{Q}[x]/(f(x))$. Hence, we can recover these elements $n_1/d_1 = c_{1,2,j}/c_{1,2,j'}$ and $n_2/d_2 = c_{2,3,j}/c_{2,3,j'}$ with $n_1, n_2, d_1, d_2 \in R$. Now, we have that

$$\frac{n_1}{d_1} \equiv \frac{c_{1,2,j}}{c_{1,2,j'}} \equiv \frac{s_{1,j}}{s_{1,j'}} \equiv \frac{c_{2,3,j}}{c_{2,3,j'}} \equiv \frac{n_2}{d_2} \pmod{\langle g \rangle},$$

and thus $n_1 \cdot d_2 - n_2 \cdot d_1 \in \langle g \rangle$. Finally, by varying j and j' as well as using different paths, we obtain other elements in $\langle g \rangle$, which eventually enables us to find a basis of $\langle g \rangle$ using GCDs.

C Probability That Two Elements of R Have Coprime Norms

The analysis of our attack on GGH15 multilinear maps relies on the fact that, with significant probability, two random elements of the base ring R have coprime norms. This is easy to check for any given R, but we provide an asymptotic estimate of that probability in the main case of interest, namely the family of rings $R = R^{(n)} = \mathbb{Z}[x]/(x^{2^n}+1)$, the rings of integers of power-of-two cyclotomic fields.

As usual, what we mean by "probability that two random elements of R have coprime norms" is the Euler product $L(R) = \prod_p L_p(R)$ where for any rational prime p, $L_p(R)$ is the probability that two elements of R which are assumed to be uniformly and independently distributed modulo p do not both have algebraic norms divisible by p (*i.e.* at least one of them is invertible modulo p). Since the ring R/pR is finite, this probability is well-defined and equal to:

$$L_p(R) = 1 - \Big(\frac{\#(R/pR) - \#(R/pR)^\times}{\#(R/pR)}\Big)^2.$$

For example, $L(\mathbb{Z}) = \prod_p(1-1/p^2) = 1/\zeta(2) = 6/\pi^2$ is the well-known "probability that two rational integers are coprime".

In the case of the cyclotomic ring $R = R^{(n)}$, write $L_p(R) = L_{p,n}$ and $L(R) = L_n$. We have $R/pR = \mathbb{F}_p[x]/(x^{2^n}+1)$. Now for any odd prime p, it is standard that the cyclotomic polynomial $(x^{2^n}+1)$ decomposes as a product of d irreducible factors of degree $2^n/d$ modulo p, where d is the order of p in the multiplicative group $(\mathbb{Z}/2^{n+1}\mathbb{Z})^\times$. As a result, R/pR is isomorphic to $(\mathbb{F}_{p^d})^{2^n/d}$, and we have:

$$L_{p,n} = 1 - \left(\frac{p^{2^n} - (p^d-1)^{2^n/d}}{p^{2^n}}\right)^2 = 1 - \left(1 - \Big(1-\frac{1}{p^d}\Big)^{2^n/d}\right)^2.$$

Moreover, we have $R/2R = \mathbb{F}_2[x]/(x^{2^n}+1) = \mathbb{F}_2[x]/(x+1)^{2^n}$, and exactly half of the elements of that ring are invertible, namely those congruent to 1 modulo $(x+1)$. Therefore:

$$L_{2,n} = 1 - (1/2)^2 = 3/4.$$

So we would like to estimate the infinite product:

$$L_n = \frac{3}{4}\prod_{p\neq 2} L_{p,n} = \frac{3}{4}\prod_{d\mid 2^n}\ \prod_{p \text{ of order } d \text{ in } (\mathbb{Z}/2^{n+1}\mathbb{Z})^\times} \left(1 - \Big(1 - (1-\frac{1}{p^d})^{2^n/d}\Big)^2\right),$$

which is absolutely convergent since for fixed n, $L_{p,n} = 1 + O(1/p^2)$. We claim that in fact, L_n approaches 3/4 as n goes to infinity.

Indeed, for any odd prime p, the order d of p in $(\mathbb{Z}/2^{n+1}\mathbb{Z})^\times$ clearly satisfies $p^d > 2^{n+1}$, since $p^d \equiv 1 \pmod{2^{n+1}}$. In particular, $d > n/\log_2 p$. Thus, we have the following inequalities:

$$\begin{aligned}
\Big(1-\frac{1}{p^d}\Big)^{2^n/d} &> 1 - \frac{2^n/d}{p^d} > 1 - \frac{2^n/d}{2^{n+1}}\\
1 - \Big(1-\frac{1}{p^d}\Big)^{2^n/d} &< \frac{1}{2d} < \frac{\log p}{n}\\
L_{p,n} &> 1 - \frac{\log^2 p}{n^2},
\end{aligned}$$

and hence $\lim_{n\to\infty} L_{p,n} = 1$. Applying dominated convergence to the series $\sum \log L_{p,n}$, we can then deduce that

$$\lim_{n\to\infty} L_n = \frac{3}{4}$$

as required. Practical values are already close to $3/4$. For example, a numerical estimate gives $L_{10} \approx 0.741\ldots$

D Extension to $k \geq 3$ Users

In this section we describe the extension of our attack against GGH15 multilinear maps for $k \geq 3$ users. We have the following public encodings:

$$\boldsymbol{A}_{i,j} \cdot \boldsymbol{C}_{i,j,\ell} = t_{1+(j-i \bmod k),\ell} \cdot \boldsymbol{A}_{i,j+1} + \boldsymbol{E}_{i,j,\ell} \pmod{q} \tag{26}$$

where $\boldsymbol{A}_{i,k+1} = \boldsymbol{A}_0$. We also have the corresponding encodings $\boldsymbol{D}_{i,j}$ of the exponents $s_{1+(j-i \bmod k)}$, where on each row i, only the encoding of s_i is private and the other encodings are public.

Our attack uses only 3 rows, namely rows $i = 1$, $i = 2$ and the last row $i = k$. As in Section 3.1, we work with constants $t_3, \ldots, t_k$, so we can drop the corresponding index ℓ for such exponents and the corresponding encodings.

First step: linear relations. In the first step, we work with rows 2 and k. On row $i = 2$, we see from (26) that the exponents t_i are encoded in the following sequence: t_k, $t_1, t_2, \ldots, t_{k-1}$. As previously we can define the product encodings $\hat{\boldsymbol{C}}_{2,2,\ell} = \boldsymbol{C}_{2,1} \cdot \boldsymbol{C}_{2,2,\ell}$ and $\hat{\boldsymbol{C}}_{2,k,\ell} = \boldsymbol{C}_{2,3,\ell} \cdot \prod_{j=4}^{k} \boldsymbol{C}_{2,j}$, and we have:

$$\boldsymbol{A}_{2,1} \cdot \hat{\boldsymbol{C}}_{2,2,\ell} = t_k \cdot t_{1,\ell} \cdot \boldsymbol{A}_{2,2} + \hat{\boldsymbol{E}}_{2,2,\ell} \pmod{q} \tag{27}$$

$$\boldsymbol{A}_{2,2} \cdot \hat{\boldsymbol{C}}_{2,k,\ell} = t_{2,\ell} \cdot \prod_{j=3}^{k-1} t_j \cdot \boldsymbol{A}_0 + \hat{\boldsymbol{E}}_{2,k,\ell} \pmod{q} \tag{28}$$

for some small $\hat{\boldsymbol{E}}_{2,2,\ell}$ and $\hat{\boldsymbol{E}}_{2,k,\ell}$.

Similarly on row $i = k$, we see from (26) that the exponents t_i are encoded in the following sequence: $t_2, \ldots, t_k$, t_1. We can define the product encoding $\hat{\boldsymbol{C}}_{k,k-1,\ell} = \boldsymbol{C}_{k,1,\ell} \cdot \prod_{j=2}^{k-1} \boldsymbol{C}_{k,j}$ and we get:

$$\boldsymbol{A}_{k,1} \cdot \hat{\boldsymbol{C}}_{k,k-1,\ell} = t_{2,\ell} \cdot \prod_{j=3}^{k} t_j \cdot \boldsymbol{A}_{k,k-1} + \hat{\boldsymbol{E}}_{k,k-1,\ell} \pmod{q} \tag{29}$$

$$\boldsymbol{A}_{k,k-1} \cdot \boldsymbol{C}_{k,k,\ell} = t_{1,\ell} \cdot \boldsymbol{A}_0 + \boldsymbol{E}_{k,k,\ell} \pmod{q} \tag{30}$$

for some small $\hat{\boldsymbol{E}}_{k,k-1,\ell}$.

We obtain that equations (27), (28), (29) and (30) are sufficient to derive linear relations on the secret exponents $t_{1,\ell}$. Namely as previously we can compute over R the matrix elements, restricting ourselves to the first component:

$$\begin{aligned}(\boldsymbol{W})_{ij} &= \boldsymbol{A}_{2,1} \cdot \hat{\boldsymbol{C}}_{2,2,i} \cdot \hat{\boldsymbol{C}}'_{2,k,j} - \boldsymbol{A}_{k,1} \cdot \hat{\boldsymbol{C}}_{k,k-1,j} \cdot \boldsymbol{C}'_{k,k,i} \\ &= t_{1,i} \cdot t_k \cdot \hat{E}_{2,k,j} + \hat{\boldsymbol{E}}_{2,2,i} \cdot \hat{\boldsymbol{C}}'_{2,k,j} \\ &\qquad - t_{2,j} \cdot \prod_{a=3}^{k} t_a \cdot E_{k,k,i} - \hat{\boldsymbol{E}}_{k,k-1,j} \cdot \boldsymbol{C}'_{k,k,i} \\ &= \begin{bmatrix} t_{1,i} & \hat{\boldsymbol{E}}_{2,2,i} & E_{k,k,i} & \boldsymbol{C}'_{k,k,i} \end{bmatrix} \cdot \begin{bmatrix} t_k \cdot \hat{E}_{2,k,j} \\ \hat{\boldsymbol{C}}'_{2,k,j} \\ -t_{2,j} \cdot \prod\limits_{a=3}^{k} t_a \\ -\hat{\boldsymbol{E}}_{k,k-1,j} \end{bmatrix}\end{aligned}$$

and therefore as in Section 3.1 by computing the left kernel of $\boldsymbol{W}$ we can obtain linear relations between the $t_{1,i}$'s. Moreover, since on rows 2 and k the encodings $\boldsymbol{D}_{2,2}$ and $\boldsymbol{D}_{k,k}$ of s_1 are public, as previously we can express s_1 as a linear combination of the $t_{1,i}$'s:

$$s_1 = \sum_{i=1}^{2m+3} \alpha_i \cdot t_{1,i} \tag{31}$$

and we also have the linear relations:

$$\hat{\boldsymbol{F}}_{2,2} = \sum_{i=1}^{2m+3} \alpha_i \cdot \hat{\boldsymbol{E}}_{2,2,i}, \quad F_{k,k} = \sum_{i=1}^{2m+3} \alpha_i \cdot E_{k,k,i}, \quad \boldsymbol{D}'_{k,k} = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{C}'_{k,k,i}. \tag{32}$$

Second step: equivalent private-key. In the second step, we use rows 1 and k. On row $i = 1$, we see from (26) that the exponents t_i are encoded from t_1 to t_k. As previously we can define the product encoding $\hat{\boldsymbol{C}}_{1,k,\ell} := \boldsymbol{C}_{1,2,\ell} \cdot \prod_{j=3}^{k} \boldsymbol{C}_{1,j}$, and we get:

$$\boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,\ell} = t_{1,\ell} \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}_{1,1} \pmod q \tag{33}$$

$$\boldsymbol{A}_{1,2} \cdot \hat{\boldsymbol{C}}_{1,k,\ell} = t_{2,\ell} \cdot \prod_{j=3}^{k} t_j \cdot \boldsymbol{A}_0 + \hat{\boldsymbol{E}}_{1,k} \pmod q\,. \tag{34}$$

As in Section 3.1 we can compute:

$$\tilde{\boldsymbol{D}}_{1,1} = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{C}_{1,1,i}$$

which gives:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} = s_1 \cdot \boldsymbol{A}_{1,2} + \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \pmod q\,. \tag{35}$$

Using (29), (30), (33) and (34), we can compute over R:

$$\begin{aligned}
\varOmega_j &= \sum_{i=1}^{2m+3} \alpha_i \cdot \Big(\boldsymbol{A}_{1,1} \cdot \boldsymbol{C}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,k,j} - \boldsymbol{A}_{k,1} \cdot \hat{\boldsymbol{C}}_{k,k-1,j} \cdot \boldsymbol{C}'_{k,k,i} \Big) \\
&= \sum_{i=1}^{2m+3} \alpha_i \cdot \big(t_{1,i} \cdot \hat{E}_{1,k,j} + \boldsymbol{E}_{1,1,i} \cdot \hat{\boldsymbol{C}}'_{1,k,j} - t_{2,j} \cdot \prod_{a=3}^{k} t_a \cdot E_{k,k,i} \\
&\qquad - \hat{\boldsymbol{E}}_{k,k-1,j} \cdot \boldsymbol{C}'_{k,k,i} \big) \\
&= s_1 \hat{E}_{1,3,j} - t_{2,j} \cdot \prod_{a=3}^{k} t_a \cdot F_{k,k} - \hat{\boldsymbol{E}}_{k,k-1,j} \cdot \boldsymbol{D}'_{k,k} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}'_{1,k,j} \\
&= u_j + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}'_{1,k,j}
\end{aligned}$$

for some small $u_j \in R$. We can now extend the previous equation to a vector $\boldsymbol{\varOmega}$ with m components, by working with column vectors $\hat{\boldsymbol{C}}'_{1,k,j}$ for $1 \le j \le m$, and obtain:

$$\boldsymbol{\varOmega} = \boldsymbol{u} + \left(\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} \right) \cdot \hat{\boldsymbol{C}}''_{1,k}$$

where $\hat{\boldsymbol{C}}''_{1,k}$ is a public square matrix of dimension m.

As in Section 3.1, this enables to derive an approximation of $\sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i}$; namely we obtain a vector $\boldsymbol{E}$ such that:

$$\boldsymbol{E}' = \sum_{i=1}^{2m+3} \alpha_i \cdot \boldsymbol{E}_{1,1,i} - \boldsymbol{E} \tag{36}$$

has small components. Combining (35) and (36), we get:

$$\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} - \boldsymbol{E} = s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}' \pmod q$$

for a small vector $\boldsymbol{E}'$, which breaks the protocol. Namely we can eventually compute from public parameters:

$$\begin{aligned}
\Big(\boldsymbol{A}_{1,1} \cdot \tilde{\boldsymbol{D}}_{1,1} - \boldsymbol{E} \Big) \cdot \prod_{i=2}^{k} \boldsymbol{D}_{1,i} &= \big(s_1 \cdot \boldsymbol{A}_{1,2} + \boldsymbol{E}' \big) \cdot \prod_{i=2}^{k} \boldsymbol{D}_{1,i} \pmod q \\
&= \left(\prod_{i=1}^{k} s_i \right) \cdot \boldsymbol{A}_0 + \boldsymbol{F} \pmod q
\end{aligned}$$

for a small vector $\boldsymbol{F}$, which enables to extract the high-order bits of $\prod_{i=1}^{k} s_i \cdot \boldsymbol{A}_0 \bmod q$ and breaks the protocol.